MODUL PRAKTIKUM

# ALGORITMA PEMROGRAMAN

***TEKNIK INFORMATIKA***


20
26
20
27


**TIM PENYUSUN**

1. Ir. Siska Anraeni, S.Kom.,M.T.,MCF.
2. Ramdaniah, S.Kom., M.T.,MTA.
3. Fattima AR. Tuasamu, S.Kom., MTA., MOS

Laboratorium Terpadu
Fakultas Ilmu Komputer
Universitas Muslim Indonesia

**VISI**

Menjadi Program Studi yang berkontribusi dalam pengembangan sumber daya manusia islami bidang komputasi dan jaringan komputer yang dapat bersaing di tingkat global khususnya masyarakat Kawasan Timur Indonesia.

**MISI**

- Menjalankan kegiatan pendidikan dan pembelajaran yang berlandaskan nilai-nilai keislaman pada bidang komputasi dan jaringan komputer.
- Melaksanakan kegiatan penelitian dan pengembangan bidang komputasi dan jaringan komputer
- Memfasilitasi dosen dan mahasiswa dalam kegiatan pengabdian yang berfokus pada penerapan teknologi informasi untuk mendukung masyarakat mandiri khususnya di Kawasan Timur Indonesia
- Mendukung kegiatan pengembangan keterampilan minat dan bakat bidang komputasi dan jaringan komputer bagi dosen dan mahasiswa.

**TUJUAN**

- Menghasilkan lulusan islami yang memiliki kemampuan analitis dan keterampilan teknis yang mendalam di bidang komputasi dan jaringan komputer.
- Meningkatnya kuantitas dan kualitas penelitian yang relevan dengan kebutuhan industri dan masyarakat.
- Terlaksananya program pengabdian yang berfokus pada penerapan teknologi informasi untuk meningkatkan kemandirian masyarakat khususnya di Kawasan Timur Indonesia.
- Terbentuknya sumber daya manusia terampil dalam bidang komputasi dan jaringan komputer.

# TATA TERTIB PELAKSANAAN PRAKTIKUM

Tata Tertib Pelaksanaan Praktikum pada Laboratorium Terpadu Fakultas Ilmu Komputer UMI adalah sebagai berikut:

1. Mahasiswa wajib mengikuti pertemuan di laboratorium sebanyak minimal 10 kali pertemuan (75% dari 14 kali pertemuan). Jika tidak memenuhi 10 kali pertemuan maka tidak diperkenankan mengikuti UAS.
2. Praktikum dimulai sesuai jadwal, dengan catatan:
    a. Toleransi keterlambatan praktikan adalah maksimal 5 menit pada semua sesi.
    b. Khusus pada Hari Jumat, sesi siang setelah Shalat Jumat diberikan toleransi keterlambatan maksimal 10 menit
3. Praktikan hanya diizinkan melaksanakan perkuliahan di laboratorium apabila:
    a. Pria
        1) Berpakaian rapi memakai kemeja putih polos;
        2) Menggunakan celana kain berwarna hitam bukan dari bahan jeans atau semi jeans;
        3) Rambut rapi dan tidak panjang;
    b. Wanita
        1) Berpakaian rapi memakai kemeja tunik putih polos (tidak transparan)
        2) Memakai Jilbab Segitiga Hitam (bukan pasmina) dan menutupi dada.
        3) Menggunakan Rok Panjang berwarna hitam yang menutupi mata kaki, tidak terbelah, tidak span, dan bukan dari bahan jeans atau semi jeans;
        4) Memakai kaos kaki dengan tinggi minimal 10 cm di atas mata kaki;
4. Ketika memasuki dan selama berada dalam ruangan, praktikan diwajibkan:
    a. Tenang, tertib, dan sopan;
    b. Tidak mengganggu praktikan lain yang sedang melaksanakan praktikum;
    c. Tidak diperbolehkan merokok, membawa makanan atau minuman senjata tajam dan senjata api ke dalam ruangan praktikum;
    d. Tidak diperbolehkan membawa *handphone* ke meja praktikum dan *handphone* dalam mode senyap;
    e. Tidak diperbolehkan membawa media penyimpanan eksternal atau *flashdisk* ke meja praktikum tanpa seizin Dosen Pengampu atau Asisten Lab;
5. Dilarang membawa, mengambil, serta memindahkan perangkat yang digunakan pada saat praktikum tanpa instruksi dari Dosen Pengampu dan Asisten lab.
6. Penggunaan fasilitas Laboratorium menyesuaikan dengan kapasitas ruang Laboratorium.

7. Segala pelanggaran yang dilakukan oleh praktikan akan berakibat pada penutupan dan penghentian penggunaan seluruh fasilitas laboratorium dan ditindak sesuai dengan aturan yang berlaku.

## SANKSI-SANKSI

Sanksi terhadap pelanggaran:
Dosen Pengampu dan Asisten laboratorium berhak menjatuhkan sanksi, sesuai dengan aturan yang berlaku di Laboratorium Terpadu Fakultas Ilmu Komputer UMI apabila:

1. Praktikan merusak peralatan praktikum (*Personal Computer*) secara sengaja, maka praktikan bertanggung jawab untuk mengganti kerusakan tersebut.
2. Praktikan tidak mematuhi dan mentaati aturan praktikum maka tidak diperkenankan mengikuti praktikum.

Pelanggaran point lainnya dikenakan sanksi teguran, dikeluarkan/dicoret namanya dalam kegiatan praktikum (mengulang mata kuliah sesuai dengan semester berjalan) sampai sanksi akademik.

Mengetahui,
Kepala Laboratorium

**Ir. Abdul Rachman Manga', S.Kom., M.T., MTA**

# DAFTAR ISI

# DAFTAR GAMBAR

MODUL 01

# Pengenalan Notasi Algoritma: Pseudocode, Flowchart, dan Input/Output

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL03** | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. |
| **CPMK032** | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. |
| **Sub-CPMK** | Mahasiswa mampu memahami dan menggunakan notasi algoritmik berupa pseudocode dan flowchart untuk memodelkan masalah sederhana. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu memahami konsep dasar algoritma dan menggunakan notasi algoritmik untuk menggambarkan penyelesaian masalah sederhana. Mahasiswa mampu menentukan data masukan, proses pengolahan, dan keluaran yang dibutuhkan serta menuangkannya ke dalam pseudocode yang terstruktur dan flowchart menggunakan simbol yang sesuai.

Melalui kegiatan praktikum, mahasiswa juga diharapkan mampu membaca kembali alur algoritma yang telah dibuat, memeriksa kesesuaian antara masalah, pseudocode, dan flowchart, serta melakukan perbaikan apabila ditemukan ketidaktepatan logika. Fokus tersebut sesuai dengan materi pekan pertama pada RPS yang meliputi orientasi berpikir algoritmik, pemodelan masalah, input/output, pseudocode, dan flowchart.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 1 apabila mampu:

1. Menjelaskan pengertian algoritma dengan benar, termasuk fungsi algoritma sebagai langkah sistematis dalam penyelesaian masalah.
2. Mengidentifikasi input, proses, dan output dari suatu masalah sederhana dengan tepat.

3. Menyusun pseudocode untuk masalah sederhana dengan urutan langkah yang logis, jelas, dan terstruktur.
4. Menggambar flowchart menggunakan simbol yang tepat serta menunjukkan arah alur proses dengan benar.
5. Menunjukkan kesesuaian antara pseudocode dan flowchart, sehingga keduanya menggambarkan solusi yang sama.
6. Memeriksa dan memperbaiki kesalahan dasar pada algoritma berdasarkan hasil simulasi atau tracing.

## E. Teori Pendukung

1. Pengertian dan karakteristik algoritma

   Algoritma adalah urutan langkah yang logis, terstruktur, dan sistematis untuk menyelesaikan suatu masalah. Dalam pemrograman, algoritma menjadi rancangan awal sebelum solusi diterjemahkan ke dalam bahasa pemrograman. Algoritma tidak bergantung pada bahasa pemrograman tertentu sehingga satu algoritma dapat diimplementasikan menggunakan C++, Java, Python, atau bahasa lainnya.

   Algoritma yang baik memiliki input, proses yang jelas, output, langkah yang tidak ambigu, serta kondisi berhenti.
2. Orientasi berpikir algoritmik

   Berpikir algoritmik adalah cara menyelesaikan masalah secara bertahap dan terstruktur. Prosesnya meliputi memahami masalah, memecah masalah menjadi bagian kecil, menyusun urutan penyelesaian, lalu menguji hasilnya.
3. Identifikasi input, proses, dan output

   Salah satu cara sederhana untuk memahami masalah adalah menggunakan pendekatan **Input–Process–Output**, yang disingkat IPO. Pendekatan IPO digunakan untuk mengidentifikasi data yang diterima, transformasi yang dilakukan, dan hasil yang harus dihasilkan. IPO menjadi jembatan antara deskripsi masalah sehari-hari dengan rancangan algoritma yang dapat ditulis sebagai pseudocode atau flowchart. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

   a. **Input:** Input adalah data yang dimasukkan ke dalam algoritma untuk diolah. Input dapat berasal dari pengguna, berkas, sensor, basis data, atau hasil proses lain.

Contoh input:

- Panjang dan lebar;
- Nilai tugas, uts, dan uas;
- Jumlah barang dan harga satuan;
- Daftar nama mahasiswa;
- Angka yang akan dicari.

**b. Process:** Process adalah langkah pengolahan terhadap input. Proses dapat berupa:

- Perhitungan;
- Perbandingan;
- Pengurutan;
- Pencarian;
- Validasi;
- Pengulangan;
- Pengambilan keputusan.

**c. Output:** Output adalah hasil yang diperoleh setelah proses dijalankan. Output dapat berupa angka, teks, status, tabel, atau informasi lainnya.

4. Pseudocode

Pseudocode adalah cara menuliskan algoritma menggunakan bahasa sederhana yang menyerupai struktur program, tetapi tidak terikat pada aturan penulisan bahasa pemrograman tertentu. Pseudocode digunakan untuk menjelaskan logika penyelesaian sebelum algoritma diterjemahkan menjadi kode program.

Pseudocode menyajikan logika algoritma dengan bahasa terstruktur yang tidak terikat sintaks bahasa pemrograman tertentu. Penulisan yang konsisten, penggunaan indentasi, dan pemilihan nama variabel yang bermakna membuat algoritma lebih mudah diperiksa. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Struktur dasar pseudocode:

```
ALGORITMA Nama_Algoritma

DEKLARASI
    daftar variabel

DESKRIPSI
    langkah-langkah penyelesaian
```

Tabel 1.1. Perintah Umum dalam Pseudocode

| Perintah | Fungsi |
|---|---|
| MULAI | Menandai awal algoritma |
| SELESAI | Menandai akhir algoritma |
| INPUT atau READ | Menerima data masukan |
| OUTPUT, WRITE, atau PRINT | Menampilkan hasil |
| ← atau = | Memberikan nilai kepada variabel |
| IF | Memeriksa suatu kondisi |
| THEN | Menentukan proses jika kondisi benar |
| ELSE | Menentukan proses jika kondisi salah |
| FOR | Melakukan pengulangan dengan jumlah tertentu |
| WHILE | Mengulang selama kondisi bernilai benar |
| REPEAT–UNTIL | Mengulang sampai kondisi terpenuhi |

```
ALGORITMA Menghitung_Luas_Persegi_Panjang

DEKLARASI
    panjang, lebar, luas : real

DESKRIPSI
    INPUT panjang
    INPUT lebar
    luas ← panjang × lebar
    OUTPUT luas
```

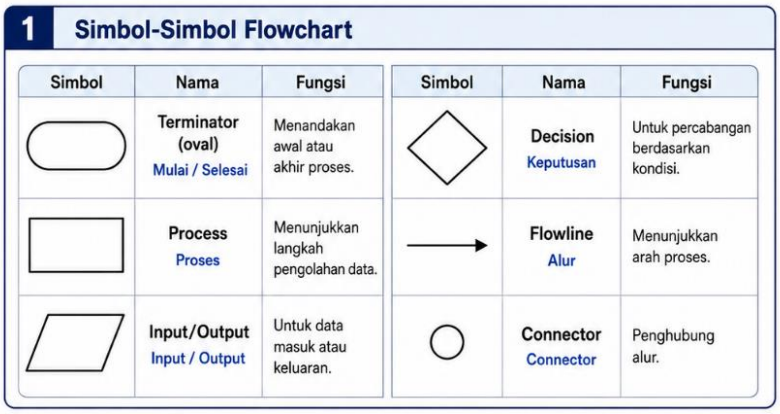
Gambar 1. Contoh Pseudocode Sekuensial

5. Simbol Flowchart

Flowchart menggambarkan alur proses secara visual melalui simbol terminator, input/output, proses, keputusan, connector, dan garis alir. Diagram ini membantu melihat arah proses serta hubungan antara percabangan dan pengulangan. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika

yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.



**1 Simbol-Simbol Flowchart**

| Simbol | Nama | Fungsi | Simbol | Nama | Fungsi |
|---|---|---|---|---|---|
| | Terminator (oval) Mulai / Selesai | Menandakan awal atau akhir proses. | | Decision Keputusan | Untuk percabangan berdasarkan kondisi. |
| | Process Proses | Menunjukkan langkah pengolahan data. | → | Flowline Alur | Menunjukkan arah proses. |
| | Input/Output Input / Output | Untuk data masuk atau keluaran. | ○ | Connector Connector | Penghubung alur. |

Gambar 2. Simbol Flowchart

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Studi Kasus

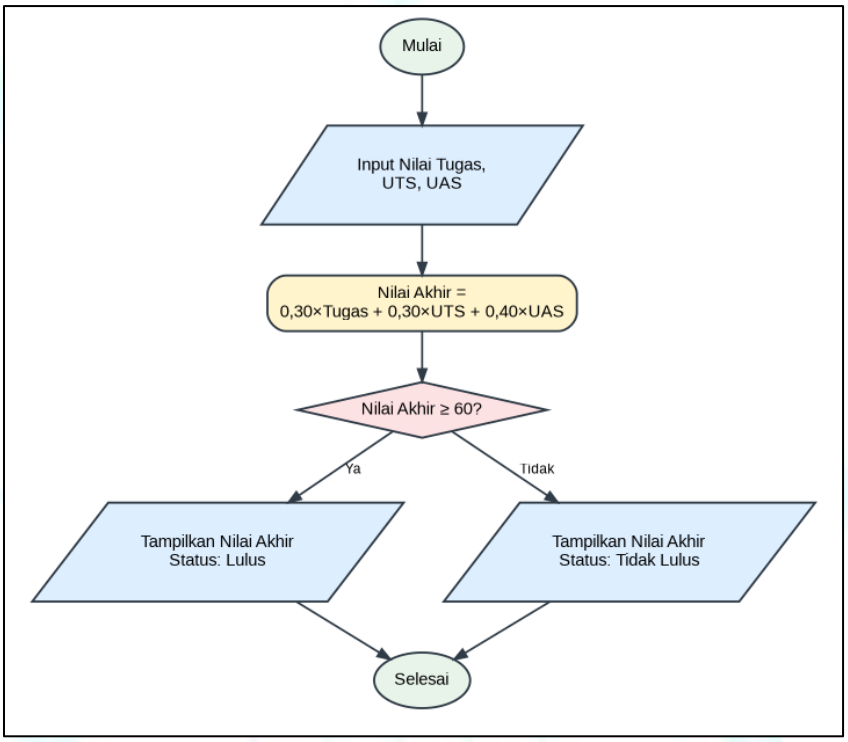
| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 1: Nilai Akhir Mahasiswa | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 2: Total Belanja dan Diskon | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 3: Konversi Suhu | 40 |
| | | **Total** | **100** |

### 1) Studi Kasus 1 - Nilai Akhir Mahasiswa

Buatlah algoritma pseudocode untuk menghitung nilai akhir mahasiswa dengan ketentuan:

| Komponen | Isi |
|---|---|
| **Input** | Nilai Tugas, Nilai UTS, Nilai UAS |
| **Proses** | Nilai Akhir = (30% × Tugas) + (30% × UTS) + (40% × UAS) |
| **Output** | Nilai Akhir dan Status Lulus/Tidak Lulus |

Flowchart:




Gambar 3. Flowchart Nilai Akhir Mahasiswa

### 2) Studi Kasus 2 - Total Belanja dan Diskon

Sebuah toko memberi diskon 10% jika total belanja minimal Rp500.000. Input berupa harga satuan dan jumlah barang. Tentukan total awal, diskon, dan total bayar. Buat pseudocode dan flowchart yang memuat proses serta keputusan.

### 3) Studi Kasus 3 - Konversi Suhu

Buat algoritma untuk menerima suhu Celsius dan menghasilkan Fahrenheit serta Kelvin. Validasi agar Kelvin tidak bernilai di bawah nol absolut. Sajikan IPO, pseudocode, flowchart, serta tracing untuk nilai normal dan batas.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 1

Sebuah program digunakan untuk mengolah nilai beberapa mahasiswa. Program menerima jumlah mahasiswa terlebih dahulu, kemudian untuk setiap mahasiswa dimasukkan **Nilai Tugas, UTS, dan UAS** dengan ketentuan:

1. Nilai Akhir = **30% Tugas + 30% UTS + 40% UAS**
2. Predikat nilai:
   a. A : Nilai Akhir ≥ 85
   b. B : 75 ≤ Nilai Akhir < 85
   c. C : 65 ≤ Nilai Akhir < 75
   d. D : 55 ≤ Nilai Akhir < 65
   e. E : Nilai Akhir < 55
3. Mahasiswa dinyatakan **Lulus** apabila Nilai Akhir ≥ 60 **dan tidak ada nilai Tugas, UTS, atau UAS yang kurang dari 40**.
4. Selain kondisi tersebut, mahasiswa dinyatakan **Tidak Lulus**.
5. Setelah seluruh data diproses, program harus menampilkan:
   a. jumlah mahasiswa yang Lulus;
   b. jumlah mahasiswa yang Tidak Lulus;
   c. rata-rata nilai akhir seluruh mahasiswa;
   d. nilai akhir tertinggi;
   e. nilai akhir terendah.

### Soal Evaluasi

1. Identifikasi komponen **input, proses, dan output** dari studi kasus di atas.
2. Susun **pseudocode terstruktur** yang dapat:
   a. menerima jumlah mahasiswa;
   b. menerima nilai Tugas, UTS, dan UAS setiap mahasiswa;
   c. menghitung Nilai Akhir;
   d. menentukan Predikat A, B, C, D, atau E;
   e. menentukan status Lulus atau Tidak Lulus;
   f. menghitung jumlah mahasiswa Lulus dan Tidak Lulus;
   g. menghitung rata-rata seluruh Nilai Akhir;
   h. menentukan Nilai Akhir tertinggi dan terendah.
3. Buat **flowchart menggunakan Flowrun.io** yang memuat:
   a. proses input;
   b. perulangan pengolahan mahasiswa;

c. seleksi bertingkat untuk predikat;
d. seleksi untuk menentukan status kelulusan;
e. proses menghitung total, rata-rata, nilai tertinggi, dan nilai terendah.

4. Lakukan **tracing** menggunakan data berikut:

| No. | Tugas | UTS | UAS | Nilai Akhir | Predikat | Status |
|---|---|---|---|---|---|---|
| 1 | 80 | 75 | 85 | … | … | … |
| 2 | 90 | 88 | 92 | … | … | … |
| 3 | 70 | 65 | 55 | … | … | … |
| 4 | 75 | 80 | 35 | … | … | … |
| 5 | 40 | 45 | 50 | … | … | … |

Setelah seluruh data selesai diproses, lengkapi hasil berikut:

| Hasil Rekapitulasi | Nilai |
|---|---|
| Jumlah Mahasiswa | 5 |
| Jumlah Lulus | … |
| Jumlah Tidak Lulus | … |
| Rata-rata Nilai Akhir | … |
| Nilai Tertinggi | … |
| Nilai Terendah | … |

**Hasil/Luaran yang Dikumpulkan**

1. Tabel identifikasi **input, proses, dan output**.
2. Dokumen pseudocode lengkap.
3. Flowchart algoritma.
4. Tabel tracing lima mahasiswa.
5. Rekapitulasi jumlah Lulus/Tidak Lulus, rata-rata, nilai tertinggi, dan nilai terendah.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|---|
| 1 | CPL03 | CPMK032 | Ketepatan mengidentifikasi input, proses, dan output | 15 |
| 2 | CPL03 | CPMK032 | Kebenaran dan keteraturan pseudocode | 25 |
| 3 | CPL03 | CPMK032 | Ketepatan penggunaan simbol dan alur flowchart | 25 |
| 4 | CPL03 | CPMK032 | Ketepatan hasil simulasi dan tracing | 20 |
| 5 | CPL03 | CPMK032 | Kemampuan menemukan dan memperbaiki kesalahan algoritma | 15 |
| | | | **Total** | 100 |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL
02

# Pola Dasar Algoritma dan Tracing

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL03** | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. |
| **CPMK032** | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. |
| **Sub-CPMK** | Mahasiswa mampu memahami dan menggunakan pola algoritma sekuensial, seleksi, dan iterasi untuk memodelkan masalah sederhana serta melakukan tracing dengan benar. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu merancang dan mengimplementasikan algoritma sederhana menggunakan pola sekuensial, struktur seleksi, dan struktur iterasi sesuai kebutuhan masalah.

Mahasiswa juga mampu melakukan tracing terhadap perubahan nilai variabel dan alur eksekusi program untuk memastikan algoritma menghasilkan keluaran yang benar. Melalui kegiatan ini, mahasiswa mulai memahami hubungan antara rancangan algoritma dan implementasinya dalam program.

Materi ini sesuai dengan RPS yang menekankan implementasi pola sekuensial, seleksi, iterasi, serta tracing algoritma.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 2 apabila mampu:

1. Menerapkan pola algoritma sekuensial secara tepat untuk menyelesaikan proses yang berjalan berurutan.
2. Menerapkan struktur seleksi if-else untuk menentukan keluaran berdasarkan kondisi tertentu.
3. Menggunakan struktur iterasi atau loop untuk menjalankan proses secara berulang.

4. Menentukan kondisi berhenti pada iterasi sehingga perulangan dapat berjalan dan berhenti dengan benar.
5. Melakukan tracing algoritma untuk menunjukkan perubahan nilai variabel pada setiap langkah atau iterasi.
6. Membandingkan hasil tracing dengan hasil program untuk memastikan ketepatan algoritma.
7. Mengidentifikasi kesalahan logika sederhana pada struktur sekuensial, seleksi, atau iterasi dan melakukan perbaikan.

## E. Teori Pendukung

Materi ini menekankan pemahaman konsep sebelum implementasi. Mahasiswa perlu melihat algoritma sebagai model penyelesaian masalah yang dapat dijelaskan, diuji, dan dianalisis. Pemahaman tersebut membantu membedakan antara sekadar kode yang dapat dijalankan dengan solusi yang benar, terstruktur, dan dapat dipertanggungjawabkan. Konsep utama berikut menjadi dasar untuk kegiatan praktikum dan digunakan kembali pada modul-modul berikutnya.

### 1. Pola Sekuensial

Pola sekuensial menjalankan instruksi satu per satu sesuai urutan. Pola ini digunakan ketika setiap langkah harus dikerjakan tanpa percabangan atau pengulangan, misalnya membaca data, menghitung hasil, kemudian menampilkan keluaran. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

### 2. Struktur Seleksi

Seleksi memilih satu jalur berdasarkan kondisi logika. Bentuk sederhana menggunakan if, sedangkan kondisi dua jalur menggunakan if-else. Seleksi majemuk diperlukan ketika lebih dari dua kategori keluaran harus dipilih. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

### 3. Struktur Iterasi

Iterasi mengulang satu atau beberapa instruksi selama syarat tertentu terpenuhi. Perulangan for cocok ketika jumlah pengulangan diketahui, sedangkan

while atau do-while digunakan ketika penghentian ditentukan oleh kondisi. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

**4. Tracing Algoritma**

Tracing adalah simulasi langkah demi langkah untuk memeriksa nilai variabel dan jalur yang dipilih algoritma. Tabel tracing membantu menemukan kesalahan logika yang tidak selalu terlihat dari kode. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

**5. Kombinasi Pola Dasar**

Sebagian besar program menggabungkan sekuensial, seleksi, dan iterasi. Penggabungan harus tetap terstruktur agar alur mudah dibaca, diuji, dan diperbaiki. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.

**6. Mekanisme Kerja dan Contoh Penerapan**

Pemilihan pola dasar ditentukan oleh karakteristik masalah. Jika semua langkah selalu dijalankan, gunakan sekuensial. Jika terdapat aturan yang menyebabkan keluaran berbeda, gunakan seleksi. Jika proses yang sama diterapkan pada banyak data atau sampai kondisi tertentu terpenuhi, gunakan iterasi. Dalam praktik, ketiga pola sering tersusun bertingkat: program membaca data secara sekuensial, mengulang proses untuk setiap data, lalu menggunakan seleksi di dalam loop untuk menentukan kategori atau tindakan.

Dalam praktikum, mekanisme tersebut tidak cukup hanya dibaca. Setiap langkah harus dapat dijelaskan dengan bahasa sendiri dan ditunjukkan melalui

data uji. Gunakan contoh berukuran kecil agar perubahan data mudah diamati. Setelah hasil manual konsisten, implementasi dapat diperluas ke ukuran input yang lebih besar. Cara ini memisahkan kesalahan konsep dari kesalahan sintaks dan mempercepat proses debugging.

**Contoh Penerapan**

Misalkan program menghitung total nilai lima mahasiswa dan menentukan status lulus. Input nilai dibaca berulang sebanyak lima kali. Di dalam setiap iterasi, program memeriksa apakah nilai minimal 60. Jika ya, status Lulus ditampilkan; jika tidak, status Tidak Lulus ditampilkan. Setelah setiap nilai diproses, total nilai ditambah. Pada akhir loop, rata-rata dihitung secara sekuensial. Tracing dapat mencatat nomor iterasi, nilai input, hasil kondisi, status, dan total sementara sehingga seluruh alur dapat diverifikasi.

Contoh tersebut sebaiknya dilengkapi tabel tracing yang mencatat keadaan penting pada setiap langkah. Kolom tabel disesuaikan dengan algoritma, misalnya indeks, nilai variabel, hasil kondisi, batas pencarian, jumlah perbandingan, atau isi array sementara. Tracing merupakan bukti bahwa mahasiswa memahami proses internal dan bukan hanya memperoleh output akhir dari program.

Setelah tracing, program diuji dengan sedikitnya tiga kelompok data: data normal, data batas, dan data yang tidak sesuai atau tidak menghasilkan kecocokan. Hasil setiap pengujian dibandingkan dengan hasil yang dihitung atau diprediksi sebelumnya. Perbedaan harus dijelaskan dan diperbaiki pada sumber logika yang tepat.

**7. Analisis dan Kesalahan Umum**

Struktur yang benar bukan hanya menghasilkan output yang benar pada satu data. Algoritma harus berhenti, tidak melewatkan data, dan memberikan hasil tepat pada semua cabang. Oleh karena itu, tracing perlu dilakukan untuk kondisi benar dan salah, iterasi pertama dan terakhir, serta kasus ketika loop tidak dijalankan sama sekali bila hal itu mungkin. Efisiensi juga mulai dapat diamati dari jumlah pemeriksaan kondisi dan jumlah pengulangan yang dilakukan.

**Kesalahan yang Perlu Dihindari**

- Urutan operasi salah pada pola sekuensial.
- Kondisi if terbalik atau saling tumpang tindih.
- Variabel kontrol loop tidak diperbarui.
- Batas awal/akhir iterasi menyebabkan off-by-one error.

Kesalahan-kesalahan tersebut sering tidak langsung terlihat karena program mungkin masih dapat dikompilasi. Oleh sebab itu, keberhasilan kompilasi tidak boleh digunakan sebagai satu-satunya indikator kebenaran. Mahasiswa harus memeriksa keluaran, jalur eksekusi, dan perilaku pada kondisi tepi. Jika algoritma mempunyai prasyarat, prasyarat tersebut harus dituliskan dan divalidasi sebelum proses utama dijalankan.

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Menyusun algoritma sekuensial untuk kasus perhitungan sederhana.
b) Membuat algoritma seleksi untuk menentukan keluaran berdasarkan kondisi.
c) Membuat algoritma iterasi untuk memproses data berulang.
d) Mengimplementasikan algoritma dasar dalam Dev C++.
e) Melakukan tracing terhadap nilai variabel dan hasil program untuk beberapa test case.

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 1: Biaya Parkir Bertingkat | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 2: Rekap Kelulusan Satu Kelas | 40 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 3: Tabel Perkalian | 40 |
| | | **Total** | **100** |

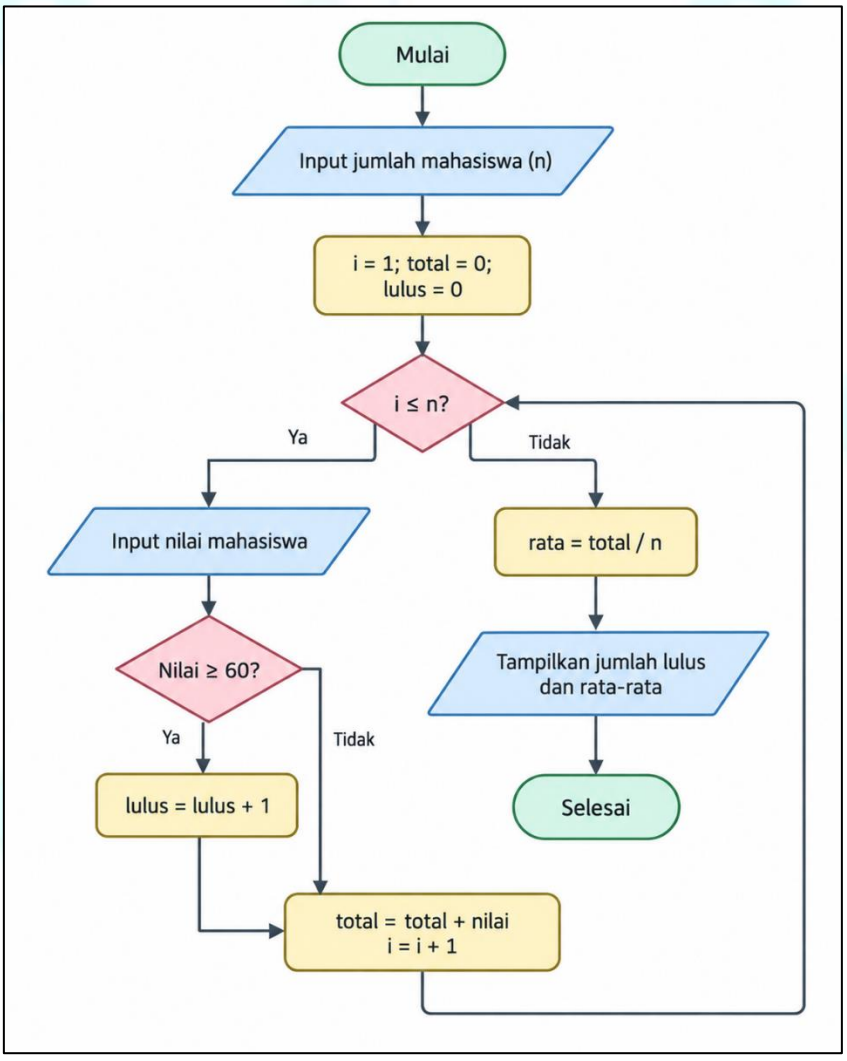
**1) Studi Kasus 1 - Biaya Parkir Bertingkat**

Hitung biaya parkir berdasarkan lama parkir. Gunakan proses sekuensial untuk membaca data, seleksi untuk tarif, dan iterasi bila beberapa kendaraan diproses. Buat tabel tracing.

**2) Studi Kasus 2 - Rekap Kelulusan Satu Kelas**

Baca n nilai mahasiswa, tentukan Lulus/Tidak Lulus untuk setiap nilai, hitung jumlah lulus, dan rata-rata kelas. Gunakan loop dan seleksi serta trace minimal lima data.

Flowchart:




Gambar 4. Flowchart Rekap Kelulusan Satu Kelas

**3) Studi Kasus 3 - Tabel Perkalian**

Terima angka n dan tampilkan tabel perkalian 1 sampai 10. Tambahkan validasi bahwa n berada pada rentang 1-100. Bandingkan alur for dan while.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 2

Sebuah minimarket ingin membuat program sederhana untuk mengolah transaksi beberapa pelanggan. Program terlebih dahulu menerima **jumlah pelanggan** yang akan diproses. Untuk setiap pelanggan dimasukkan **total belanja** dan **status keanggotaan** (Member atau Non-Member).

Ketentuan diskon:

- Total belanja ≥ Rp1.000.000 → diskon dasar **15%**
- Total belanja ≥ Rp500.000 dan < Rp1.000.000 → diskon dasar **10%**
- Total belanja ≥ Rp250.000 dan < Rp500.000 → diskon dasar **5%**
- Total belanja < Rp250.000 → **tidak mendapat diskon**
- Jika pelanggan berstatus **Member**, mendapat tambahan diskon **5%**
- Total diskon maksimal **20%** dari total belanja.

Untuk setiap pelanggan, program harus menghitung:

**Diskon = Persentase Diskon** x **Total Belanja**

**Total Bayar = Total Belanja - Diskon**

Setelah semua pelanggan selesai diproses, program menampilkan:

- jumlah pelanggan yang mendapat diskon;
- jumlah pelanggan yang tidak mendapat diskon;
- total seluruh transaksi sebelum diskon;
- total seluruh diskon;
- total pendapatan setelah diskon.

### Soal Evaluasi

1. Identifikasi komponen input, proses, dan output dari studi kasus di atas.
2. Susun **pseudocode** yang menerapkan:
   a. pola sekuensial untuk menerima dan mengolah data;
   b. struktur seleksi untuk menentukan persentase diskon;
   c. struktur seleksi untuk memeriksa status Member;
   d. struktur iterasi untuk memproses seluruh pelanggan;
   e. counter untuk menghitung pelanggan yang mendapat dan tidak mendapat diskon;
   f. accumulator untuk menghitung total transaksi, total diskon, dan total pendapatan.
3. Buat **flowchart** menggunakan Flowrun.io yang menunjukkan:
   a. input jumlah pelanggan;

b. proses perulangan pelanggan;
c. seleksi berdasarkan total belanja;
d. seleksi berdasarkan status Member;
e. pemeriksaan agar diskon tidak lebih dari 20%;
f. perhitungan total bayar;
g. proses rekapitulasi setelah perulangan selesai.

4. Lakukan **tracing** menggunakan data berikut:

| Pelanggan | Total Belanja | Status | Diskon Dasar | Tambahan Member | Total Diskon | Total Bayar |
|---|---|---|---|---|---|---|
| 1 | Rp1.200.000 | Member | … | … | … | … |
| 2 | Rp750.000 | Non-Member | … | … | … | … |
| 3 | Rp400.000 | Member | … | … | … | … |
| 4 | Rp200.000 | Non-Member | … | … | … | … |
| 5 | Rp1.500.000 | Member | … | … | … | … |

5. Setelah tracing selesai, lengkapi rekapitulasi berikut:

| Hasil Rekapitulasi | Hasil |
|---|---|
| Jumlah pelanggan | 5 |
| Pelanggan mendapat diskon | … |
| Pelanggan tanpa diskon | … |
| Total transaksi sebelum diskon | Rp … |
| Total seluruh diskon | Rp … |
| Total pendapatan setelah diskon | Rp … |

6. Periksa kembali algoritma dan jelaskan:
   a. Apa yang terjadi pada pelanggan Member dengan belanja Rp1.200.000?
   b. Bagaimana algoritma memastikan total diskon tidak melebihi 20%?
   c. Apa akibatnya jika variabel penghitung pelanggan tidak diperbarui di dalam loop?
   d. Apa akibatnya jika kondisi total belanja ≥ Rp1.000.000 diletakkan setelah kondisi total belanja ≥ Rp250.000 tanpa menggunakan struktur seleksi yang benar?
   e. Tuliskan minimal **dua kesalahan hasil tracing** yang mungkin terjadi dan cara memperbaikinya.

**Hasil/Luaran yang Dikumpulkan**

1. Tabel identifikasi **input, proses, dan output**.
2. Dokumen pseudocode.
3. Flowchart algoritma.
4. Tabel tracing lima pelanggan.
5. Tabel rekapitulasi hasil transaksi.
6. File implementasi program menggunakan **Dev C++**.
7. Catatan hasil pemeriksaan dan perbaikan algoritma.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|---|
| 1 | CPL03 | CPMK032 | Ketepatan menerapkan **pola algoritma sekuensial** sesuai urutan proses | 10 |
| 2 | CPL03 | CPMK032 | Ketepatan menerapkan **struktur seleksi (if-else)** sesuai kondisi yang diberikan | 15 |
| 3 | CPL03 | CPMK032 | Ketepatan menerapkan struktur iterasi (for/while) untuk memproses data berulang | 20 |
| 4 | CPL03 | CPMK032 | Ketepatan menggunakan counter, accumulator, dan perubahan nilai variabel selama iterasi | 15 |
| 5 | CPL03 | CPMK032 | Ketepatan melakukan tracing dan menentukan hasil setiap tahapan/iterasi | 20 |
| 6 | CPL03 | CPMK032 | Kemampuan menentukan kondisi berhenti, menemukan serta memperbaiki infinite loop/kesalahan logika | 10 |
| 7 | CPL03 | CPMK032 | Kemampuan membandingkan dan mengoptimalkan struktur pengulangan serta memperoleh hasil akhir yang benar | 10 |
| | | | **Total** | 100 |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 03

# Stepwise Refinement dan Dekomposisi Masalah

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL03** | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. |
| **CPMK032** | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. |
| **Sub-CPMK** | Mahasiswa mampu merancang algoritma dengan pola dasar dan menerapkan stepwise refinement untuk memecah masalah kompleks menjadi submasalah yang terstruktur. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menganalisis suatu masalah yang relatif kompleks kemudian memecahnya menjadi beberapa submasalah menggunakan pendekatan top-down dan teknik stepwise refinement.

Mahasiswa mampu menyusun solusi yang lebih terstruktur melalui pembentukan modul atau subalgoritma yang memiliki fungsi, input, proses, dan output yang jelas. Mahasiswa juga mampu menggambarkan hubungan antarbagian melalui hierarchy chart, menyusun pseudocode modular, serta mengidentifikasi dependensi antarmodul. Pada tahap akhir, mahasiswa mampu mengintegrasikan kembali seluruh subalgoritma menjadi satu solusi utuh dan melakukan tracing untuk memastikan alur keseluruhan berjalan dengan benar.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 3 apabila mampu:

1. Menganalisis masalah utama serta menentukan tujuan, input, proses, dan output secara tepat.
2. Menerapkan pendekatan top-down untuk memecah masalah menjadi beberapa bagian yang lebih sederhana.
3. Menerapkan teknik stepwise refinement secara bertahap dari proses umum menuju proses yang lebih rinci.
4. Memecah masalah kompleks menjadi submasalah yang logis dan memiliki tanggung jawab yang jelas.

5. Menyusun hierarchy chart yang menunjukkan struktur dan hubungan antar modul.
6. Merancang algoritma secara modular dengan membuat pseudocode untuk setiap subalgoritma.
7. Mengidentifikasi dependensi antarmodul, termasuk data yang diterima dan diteruskan antarbagian.
8. Mengintegrasikan seluruh subalgoritma menjadi solusi yang utuh dan konsisten.
9. Melakukan tracing alur modular untuk memeriksa kebenaran hubungan dan urutan eksekusi antar modul.

## E. Teori Pendukung

Materi ini menekankan pemahaman konsep sebelum implementasi. Mahasiswa perlu melihat algoritma sebagai model penyelesaian masalah yang dapat dijelaskan, diuji, dan dianalisis. Pemahaman tersebut membantu membedakan antara sekadar kode yang dapat dijalankan dengan solusi yang benar, terstruktur, dan dapat dipertanggungjawabkan. Konsep utama berikut menjadi dasar untuk kegiatan praktikum dan digunakan kembali pada modul-modul berikutnya.

### 1. Dekomposisi masalah

Dekomposisi membagi masalah besar menjadi submasalah yang lebih kecil dan mudah dikelola. Setiap submasalah memiliki tujuan, input, proses, dan output yang jelas. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

### 2. Pendekatan top-down

Pendekatan top-down dimulai dari fungsi utama sistem, kemudian diturunkan menjadi fungsi yang lebih rinci. Proses berlanjut sampai setiap bagian cukup sederhana untuk diimplementasikan. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

3. **Stepwise refinement**

Stepwise refinement memperhalus deskripsi solusi secara bertahap. Langkah yang masih umum dipecah menjadi operasi yang lebih spesifik tanpa kehilangan hubungan dengan tujuan utama. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

4. **Algoritma modular**

Modularitas memisahkan program menjadi fungsi atau prosedur yang memiliki tanggung jawab tertentu. Modul yang baik dapat diuji secara independen dan digunakan kembali. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

5. **Dependensi antarmodul**

Dependensi menunjukkan data atau hasil yang dibutuhkan satu modul dari modul lain. Dependensi yang jelas mencegah pertukaran data yang tidak terkontrol dan memudahkan integrasi. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.

6. **Mekanisme Kerja dan Contoh Penerapan**

Langkah awal dekomposisi adalah menuliskan fungsi utama sistem. Setelah itu, pecah fungsi tersebut menjadi kelompok kegiatan seperti input data, validasi, pengolahan, penyimpanan sementara, dan penyajian hasil. Setiap kelompok kemudian diperinci lagi sampai dapat ditulis sebagai pseudocode sederhana. Hierarchy chart dapat digunakan untuk menunjukkan hubungan antara modul

utama dan submodul. Data yang masuk dan keluar dari tiap modul sebaiknya dicatat sehingga ketergantungan tidak tersembunyi.

Dalam praktikum, mekanisme tersebut tidak cukup hanya dibaca. Setiap langkah harus dapat dijelaskan dengan bahasa sendiri dan ditunjukkan melalui data uji. Gunakan contoh berukuran kecil agar perubahan data mudah diamati. Setelah hasil manual konsisten, implementasi dapat diperluas ke ukuran input yang lebih besar. Cara ini memisahkan kesalahan konsep dari kesalahan sintaks dan mempercepat proses debugging.

**Contoh Penerapan**

Program menerima **Nama, Nilai Tugas, UTS, dan UAS** mahasiswa. Nilai akhir dihitung dengan ketentuan:

- Tugas = 30%
- UTS = 30%
- UAS = 40%
- Nilai ≥ 85 → A
- Nilai ≥ 75 → B
- Nilai ≥ 65 → C
- Nilai ≥ 55 → D
- Nilai < 55 → E
- Mahasiswa dinyatakan **Lulus** jika Nilai Akhir ≥ 60

**a. Dokumen Stepwise Refinement**

**Level 0 – Masalah Utama**

PROSES NILAI MAHASISWA

**Level 1 – Dekomposisi Utama**

1. Input data mahasiswa
2. Hitung nilai akhir
3. Tentukan predikat
4. Tentukan status kelulusan
5. Tampilkan hasil

**Level 2 – Perincian Setiap Proses**

1. Input data mahasiswa
   1.1 Input nama mahasiswa
   1.2 Input nilai tugas
   1.3 Input nilai UTS
   1.4 Input nilai UAS

2. Hitung nilai akhir
   2.1 Hitung 30% nilai tugas
   2.2 Hitung 30% nilai UTS
   2.3 Hitung 40% nilai UAS
   2.4 Jumlahkan seluruh komponen
3. Tentukan predikat
   3.1 Jika nilai >= 85 → A
   3.2 Jika nilai >= 75 → B
   3.3 Jika nilai >= 65 → C
   3.4 Jika nilai >= 55 → D
   3.5 Selain itu → E
4. Tentukan status kelulusan
   4.1 Jika nilai akhir >= 60 → Lulus
   4.2 Jika nilai akhir < 60 → Tidak Lulus
5. Tampilkan hasil
   5.1 Tampilkan nama
   5.2 Tampilkan nilai akhir
   5.3 Tampilkan predikat
   5.4 Tampilkan status

**Level 3 – Implementasi Modular**

Program Utama
│
├─── InputData()
├─── HitungNilaiAkhir()
├─── TentukanPredikat()
├─── TentukanStatus()
└─── TampilkanHasil()

**b. Hierarchy Chart**

Hierarchy chart menggambarkan struktur modul dari tingkat paling atas ke bagian yang lebih kecil.

Pengolahan Nilai Mahasiswa
│
├── Input Data
│ ├── Nama
│ ├── Nilai Tugas
│ ├── Nilai UTS
│ └── Nilai UAS
│
├── Hitung Nilai Akhir
│
├── Tentukan Predikat
│
├── Tentukan Status
│
└── Tampilkan Hasil

**c. Pseudocode Modular**

1) Program Utama

```
ALGORITMA Pengolahan_Nilai_Mahasiswa

DEKLARASI
    nama : string
    tugas, uts, uas, nilaiAkhir : real
    predikat, status : string

DESKRIPSI
    InputData(nama, tugas, uts, uas)

    nilaiAkhir ← HitungNilaiAkhir(tugas, uts, uas)

    predikat ← TentukanPredikat(nilaiAkhir)

    status ← TentukanStatus(nilaiAkhir)

    TampilkanHasil(nama, nilaiAkhir, predikat, status)
```

2) Modul Input Data

```
PROCEDURE InputData
    OUTPUT nama, tugas, uts, uas

    INPUT nama
    INPUT tugas
    INPUT uts
    INPUT uas
END PROCEDURE
```

3) Modul Hitung Nilai Akhir

```
FUNCTION HitungNilaiAkhir(tugas, uts, uas)

    nilaiAkhir ←
        (0.30 × tugas) +
        (0.30 × uts) +
        (0.40 × uas)

    RETURN nilaiAkhir

END FUNCTION
```

4) Modul Menentukan Predikat

```
FUNCTION TentukanPredikat(nilai)

    IF nilai >= 85 THEN
        predikat ← "A"
    ELSE IF nilai >= 75 THEN
        predikat ← "B"
    ELSE IF nilai >= 65 THEN
        predikat ← "C"
    ELSE IF nilai >= 55 THEN
        predikat ← "D"
    ELSE
        predikat ← "E"
    ENDIF

    RETURN predikat

END FUNCTION
```

5) Modul Menentukan Status

```
FUNCTION TentukanStatus(nilai)

    IF nilai >= 60 THEN
        status ← "Lulus"
    ELSE
        status ← "Tidak Lulus"
    ENDIF

    RETURN status

END FUNCTION
```

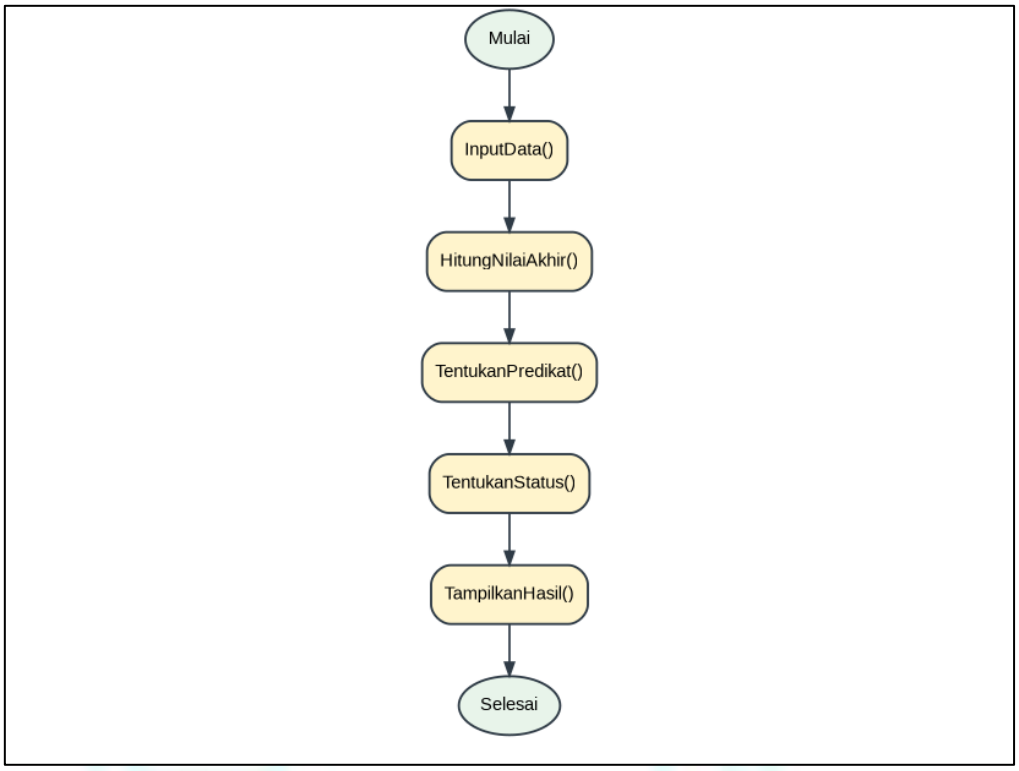
6) Modul Menampilkan Hasil

```
PROCEDURE TampilkanHasil
    INPUT nama, nilaiAkhir, predikat, status

    OUTPUT "Nama          : ", nama
    OUTPUT "Nilai Akhir   : ", nilaiAkhir
    OUTPUT "Predikat      : ", predikat
    OUTPUT "Status        : ", status

END PROCEDURE
```

### d. Tabel Dependensi Antarmodul

| No. | Modul | Memerlukan Data dari | Data Masukan | Menghasilkan | Digunakan oleh |
|---|---|---|---|---|---|
| 1 | InputData() | Pengguna | Nama, Tugas, UTS, UAS | Data mahasiswa | HitungNilaiAkhir() dan Program Utama |
| 2 | HitungNilaiAkhir() | InputData() | Tugas, UTS, UAS | Nilai Akhir | TentukanPredikat(), TentukanStatus(), TampilkanHasil() |
| 3 | TentukanPredikat() | HitungNilaiAkhir() | Nilai Akhir | Predikat A–E | TampilkanHasil() |
| 4 | TentukanStatus() | HitungNilaiAkhir() | Nilai Akhir | Lulus/Tidak Lulus | TampilkanHasil() |
| 5 | TampilkanHasil() | Semua modul sebelumnya | Nama, Nilai Akhir, Predikat, Status | Informasi hasil | Pengguna |

**Flowchart**




Gambar 5. Flowchart Pengolahan Nilai Mahasiswa Modular

## 7. Analisis dan Kesalahan Umum

Dekomposisi yang baik menghasilkan modul dengan kohesi tinggi, yaitu satu modul fokus pada satu tugas, dan coupling rendah, yaitu tidak terlalu bergantung pada detail internal modul lain. Struktur seperti ini memudahkan testing, debugging, dan perubahan kebutuhan. Sebaliknya, modul yang terlalu besar atau membagi pekerjaan secara tidak logis justru menambah kompleksitas. Evaluasi dilakukan dengan menelusuri apakah setiap kebutuhan sistem dipetakan ke modul tertentu dan apakah aliran data antar modul konsisten.

**Kesalahan yang Perlu Dihindari**

a. Submasalah terlalu besar sehingga tetap sulit dipahami.
b. Satu modul mengerjakan terlalu banyak tanggung jawab.
c. Parameter masuk/keluar tidak ditentukan.
d. Fungsi yang sama ditulis berulang di beberapa modul.

Kesalahan-kesalahan tersebut sering tidak langsung terlihat karena program mungkin masih dapat dikompilasi. Oleh sebab itu, keberhasilan kompilasi tidak boleh digunakan sebagai satu-satunya indikator kebenaran. Mahasiswa harus memeriksa keluaran, jalur eksekusi, dan perilaku pada kondisi tepi. Jika algoritma

mempunyai prasyarat, prasyarat tersebut harus dituliskan dan divalidasi sebelum proses utama dijalankan.

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Menganalisis sebuah masalah dan menentukan tujuan, masukan, proses, serta keluaran.
b) Memecah masalah menjadi beberapa submasalah menggunakan pendekatan top-down.
c) Menyusun hierarchy chart atau diagram struktur modul.
d) Membuat pseudocode untuk setiap subalgoritma.
e) Mengintegrasikan subalgoritma dan melakukan tracing alur keseluruhan.

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 1: Sistem Peminjaman Buku | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 2: Penggajian Karyawan | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 3: Pengelolaan Stok | 40 |
| | | **Total** | **100** |

#### 1) Studi Kasus 1 - Sistem Peminjaman Buku

Dekomposisi proses menjadi input anggota, validasi buku, proses pinjam/kembali, hitung denda, dan keluaran transaksi. Buat hierarchy chart dan pseudocode tiap modul.

**2) Studi Kasus 2 - Penggajian Karyawan**

Pecah proses penggajian menjadi perhitungan gaji pokok, tunjangan, pajak, dan gaji bersih. Tentukan parameter masuk/keluar setiap fungsi dan dependensinya.

**3) Studi Kasus 3 - Pengelolaan Stok**

Rancang modul untuk membaca barang, transaksi masuk, transaksi keluar, validasi stok, dan laporan stok akhir. Gunakan stepwise refinement minimal tiga tingkat.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 3

**Sistem Pemesanan dan Pembayaran Hotel**

Sebuah hotel ingin membuat sistem sederhana untuk menangani proses pemesanan kamar. Sistem menerima data pelanggan, memeriksa ketersediaan kamar, menentukan biaya menginap, memberikan diskon apabila memenuhi syarat, dan menampilkan rincian transaksi.

Ketentuan sistem:

- Data pelanggan terdiri atas **ID pelanggan, nama, dan nomor telepon**.
- Pelanggan memilih tipe kamar: **Standard, Deluxe, atau Suite**.
- Sistem harus memeriksa apakah kamar yang dipilih masih tersedia.
- Jika kamar tersedia, pelanggan memasukkan **lama menginap**.
- Biaya menginap dihitung:

**Total Awal = Harga Kamar × Lama Menginap**

- Jika lama menginap ≥ 5 malam, pelanggan memperoleh diskon **10%**.
- Jika lama menginap < 5 malam, tidak mendapat diskon.
- Total pembayaran:

**Total Bayar = Total Awal − Diskon**

- Setelah transaksi selesai, status kamar berubah menjadi **Terpesan**.
- Sistem menampilkan identitas pelanggan, tipe kamar, lama menginap, total awal, diskon, dan total bayar.

**Soal Evaluasi**

1. Lakukan **dekomposisi masalah menggunakan pendekatan top-down**. Pecah sistem tersebut minimal menjadi modul:
   a. Input Data Pelanggan
   b. Pilih dan Validasi Kamar
   c. Hitung Biaya Menginap
   d. Hitung Diskon
   e. Proses Pemesanan
   f. Tampilkan Transaksi
2. Susun **dokumen stepwise refinement minimal 3 level**, yaitu:
   a. Level 0: masalah utama;
   b. Level 1: proses-proses utama;
   c. Level 2: rincian aktivitas pada setiap proses.

3. Buat **hierarchy chart** yang menunjukkan hubungan antara program utama dan seluruh submodul.
4. Susun **pseudocode modular** untuk:
   a. InputPelanggan()
   b. ValidasiKamar()
   c. HitungBiaya()
   d. HitungDiskon()
   e. ProsesPemesanan()
   f. TampilkanTransaksi()
5. Tentukan **input dan output masing-masing modul**. Lengkapi tabel berikut:

| No. | Modul | Input | Proses Utama | Output |
|---|---|---|---|---|
| 1 | InputPelanggan | ... | ... | ... |
| 2 | ValidasiKamar | ... | ... | ... |
| 3 | HitungBiaya | ... | ... | ... |
| 4 | HitungDiskon | ... | ... | ... |
| 5 | ProsesPemesanan | ... | ... | ... |
| 6 | TampilkanTransaksi | ... | ... | ... |

6. Susun **tabel dependensi antarmodul** dengan format:

| Modul | Bergantung pada Modul | Data yang Digunakan | Hasil yang Diteruskan |
|---|---|---|---|
| InputPelanggan | ... | ... | ... |
| ValidasiKamar | ... | ... | ... |
| HitungBiaya | ... | ... | ... |
| HitungDiskon | ... | ... | ... |
| ProsesPemesanan | ... | ... | ... |
| TampilkanTransaksi | ... | ... | ... |

7. Lakukan **tracing alur modular** menggunakan data berikut:

| Data | Nilai |
|---|---|
| ID Pelanggan | P001 |
| Nama | Ahmad |
| Tipe Kamar | Deluxe |
| Harga Kamar | Rp750.000/malam |
| Status Kamar | Tersedia |
| Lama Menginap | 6 malam |

Tentukan:

- Total awal;
- Persentase diskon;
- Nominal diskon;

- Total pembayaran;
- Status kamar setelah transaksi.

8. Jelaskan apa yang terjadi apabila:
   a. kamar yang dipilih tidak tersedia;
   b. lama menginap bernilai 0;
   c. modul HitungDiskon() dijalankan sebelum HitungBiaya();
   d. modul TampilkanTransaksi() tidak menerima hasil dari modul HitungDiskon().
7. Identifikasi minimal **dua kesalahan dekomposisi atau dependensi modul** yang dapat menyebabkan sistem menghasilkan keluaran yang salah, kemudian jelaskan cara memperbaikinya.

**Hasil/Luaran yang Dikumpulkan**

1. Dokumen stepwise refinement minimal 3 level.
2. Hierarchy chart sistem.
3. Pseudocode modular seluruh proses.
4. Tabel input–proses–output tiap modul.
5. Tabel dependensi antarmodul.
6. Hasil tracing studi kasus.
7. Catatan analisis kesalahan dan perbaikan rancangan.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|---|
| 1 | CPL03 | CPMK032 | Ketepatan **menganalisis masalah utama** serta menentukan input, proses, dan output | 10 |
| 2 | CPL03 | CPMK032 | Ketepatan menerapkan **stepwise refinement** secara bertahap dari masalah utama hingga submasalah | 20 |
| 3 | CPL03 | CPMK032 | Ketepatan melakukan **dekomposisi masalah menggunakan pendekatan top-down** | 15 |
| 4 | CPL03 | CPMK032 | Ketepatan menyusun **hierarchy chart/struktur modul** sesuai hasil dekomposisi | 15 |
| 5 | CPL03 | CPMK032 | Kemampuan menemukan dan memperbaiki kesalahan algoritma Kebenaran dan keteraturan pseudocode modular untuk setiap subalgoritma | 20 |
| 6 | CPL03 | CPMK032 | Ketepatan mengidentifikasi input, output, dan dependensi antarmodul | 15 |
| 7 | CPL03 | CPMK032 | Ketepatan mengintegrasikan dan melakukan tracing alur keseluruhan modul | 5 |
| | | | **Total** | 100 |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 04

# Analisis Kompleksitas Algoritma dengan Notasi Big-O

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL08** | Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| **CPMK084** | Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| **Sub-CPMK** | Mahasiswa mampu menjelaskan notasi Big-O serta menganalisis kompleksitas waktu dan ruang algoritma dasar untuk mengevaluasi efisiensi solusi. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menganalisis efisiensi algoritma berdasarkan kompleksitas waktu dan ruang. Mahasiswa mampu mengidentifikasi operasi dominan, menentukan T(n) dan notasi Big-O, membedakan best, average, dan worst case, serta melakukan eksperimen waktu eksekusi untuk membandingkan hasil empiris dengan analisis teoretis.

## D. Indikator Ketercapaian Pembelajaran

1. Menjelaskan konsep kompleksitas waktu dan ruang dengan benar.
2. Mengidentifikasi operasi dominan dan menentukan fungsi T(n).
3. Menentukan notasi Big-O untuk struktur sekuensial, loop tunggal, dan loop bersarang.
4. Menganalisis best, average, dan worst case pada algoritma dasar.
5. Melakukan pengukuran waktu eksekusi secara konsisten untuk beberapa ukuran input.
6. Membandingkan hasil eksperimen dengan analisis teoretis dan menarik kesimpulan efisiensi.

## E. Teori Pendukung

Materi ini menekankan pemahaman konsep sebelum implementasi. Mahasiswa perlu melihat algoritma sebagai model penyelesaian masalah yang dapat dijelaskan, diuji, dan dianalisis. Pemahaman tersebut membantu

membedakan antara sekadar kode yang dapat dijalankan dengan solusi yang benar, terstruktur, dan dapat dipertanggungjawabkan. Konsep utama berikut menjadi dasar untuk kegiatan praktikum dan digunakan kembali pada modul-modul berikutnya.

1. **Kompleksitas waktu**

Kompleksitas waktu menggambarkan pertumbuhan jumlah operasi terhadap ukuran input n. Fokusnya bukan detik absolut, tetapi pola pertumbuhan ketika data semakin besar. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

2. **Kompleksitas ruang**

Kompleksitas ruang menunjukkan tambahan memori yang diperlukan algoritma, termasuk variabel, struktur data sementara, dan penggunaan stack pada rekursi. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

3. **Notasi Big-O**

Big-O menyatakan batas atas pertumbuhan dan umum digunakan untuk menggambarkan performa pada input besar. Bentuk yang sering ditemui antara lain O(1), O(log n), O(n), O(n log n), dan O(n^2). Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

4. **Best, average, worst case**

Satu algoritma dapat memiliki perilaku berbeda tergantung posisi atau susunan data. Analisis best, average, dan worst case membantu memahami rentang biaya komputasi. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

5. **Eksperimen performa**

Pengukuran waktu eksekusi melengkapi analisis teoretis. Hasil eksperimen dipengaruhi perangkat, compiler, beban sistem, dan ukuran data sehingga perlu dilakukan secara konsisten. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.

6. **Mekanisme Kerja dan Contoh Penerapan**

Analisis Big-O dimulai dengan menentukan operasi dominan. Instruksi sekuensial yang jumlahnya tetap dianggap O(1). Loop yang berjalan n kali dianggap O(n). Dua loop bersarang yang masing-masing berjalan n kali menghasilkan O(n^2). Pada algoritma yang membagi ruang pencarian menjadi setengah secara berulang, jumlah langkah tumbuh logaritmik sehingga O(log n). Konstanta dan suku berorde rendah diabaikan karena pengaruhnya semakin kecil ketika n membesar.

Dalam praktikum, mekanisme tersebut tidak cukup hanya dibaca. Setiap langkah harus dapat dijelaskan dengan bahasa sendiri dan ditunjukkan melalui data uji. Gunakan contoh berukuran kecil agar perubahan data mudah diamati. Setelah hasil manual konsisten, implementasi dapat diperluas ke ukuran input yang lebih besar. Cara ini memisahkan kesalahan konsep dari kesalahan sintaks dan mempercepat proses debugging.

**Contoh Penerapan**

1. **Penerapan O(1) – Akses Elemen Array**

   Diberikan sebuah array nilai mahasiswa. Program mengambil nilai mahasiswa pada indeks tertentu.

**Pseudocode**

```
ALGORITMA Ambil_Nilai

INPUT indeks
nilai ← data[indeks]
OUTPUT nilai
```

**Analisis Kompleksitas**

Operasi pengambilan data dilakukan satu kali dan tidak bergantung pada banyaknya data n.

**Kompleksitas waktu: O(1)**

| Jumlah Data (n) | Operasi Akses |
|---|---|
| 10 | 1 |
| 100 | 1 |
| 1.000 | 1 |
| 10.000 | 1 |

Walaupun jumlah data bertambah, jumlah operasi tetap.

2. **Penerapan O(n) – Menjumlahkan Seluruh Data**

Program menerima n nilai mahasiswa dan menghitung total seluruh nilai.

**Pseudocode**

```
ALGORITMA Hitung_Total

total ← 0

FOR i ← 0 TO n-1 DO
    total ← total + data[i]
ENDFOR

OUTPUT total
```

**Analisis Kompleksitas**

Loop berjalan sebanyak n kali.

Jika:

- n = 10 → 10 kali operasi
- n = 100 → 100 kali operasi
- n = 1.000 → 1.000 kali operasi

Maka:

**T(n) = n**

Sehingga:

**Kompleksitas waktu = O(n)**

| n | Jumlah Iterasi |
|---|---|
| 10 | 10 |
| 100 | 100 |
| 1.000 | 1.000 |
| 10.000 | 10.000 |

Artinya, jika jumlah data menjadi dua kali lebih besar, jumlah proses kurang lebih juga menjadi dua kali lebih besar.

3. **Penerapan O(n²) – Membandingkan Setiap Pasangan Data**

Program membandingkan setiap nilai mahasiswa dengan seluruh nilai mahasiswa lainnya untuk mencari apakah terdapat nilai yang sama.

**Pseudocode**

```
ALGORITMA Cari_Duplikat

FOR i ← 0 TO n-1 DO
    FOR j ← i+1 TO n-1 DO

        IF data[i] = data[j] THEN
            OUTPUT "Data sama ditemukan"
        ENDIF

    ENDFOR
ENDFOR
```

**Analisis Kompleksitas**

Terdapat dua loop bersarang.

Secara sederhana:

$T(n) \approx n \times n$

Sehingga:

**Kompleksitas waktu = O(n²)**

| n | Perkiraan Operasi |
|---|---|
| 10 | 100 |
| 100 | 10.000 |
| 1.000 | 1.000.000 |
| 10.000 | 100.000.000 |

Dari tabel terlihat bahwa algoritma $O(n^2)$ mengalami peningkatan operasi jauh lebih cepat dibandingkan O(n).

4. **Contoh Perbandingan Tiga Algoritma**

Mahasiswa dapat membandingkan tiga struktur berikut.

**Algoritma A**

```
x ← data[0]
```

**Big-O = O(1)**

**Algoritma B**

```
FOR i ← 1 TO n DO
   OUTPUT i
ENDFOR
```

**Big-O = O(n)**

**Algoritma C**

```
FOR i ← 1 TO n DO
   FOR j ← 1 TO n DO
      OUTPUT i, j
   ENDFOR
ENDFOR
```

**Big-O = $O(n^2)$**

**Tabel Perbandingan**

| Ukuran Input | O(1) | O(n) | $O(n^2)$ |
|---|---|---|---|
| 10 | 1 | 10 | 100 |
| 100 | 1 | 100 | 10.000 |
| 1.000 | 1 | 1.000 | 1.000.000 |
| 10.000 | 1 | 10.000 | 100.000.000 |

Tabel ini dapat digunakan mahasiswa untuk memahami bahwa Big-O menunjukkan pola pertumbuhan, bukan sekadar waktu dalam detik.

**5. Penerapan Best Case dan Worst Case**

Carilah angka tertentu pada sebuah array dengan memeriksa data satu per satu.

Data: [10, 20, 30, 40, 50]

Target: 10

Data ditemukan pada pemeriksaan pertama.

**Best Case = O(1)**

Jika target adalah 50, maka seluruh elemen hampir harus diperiksa.

**Worst Case = O(n)**

Jika target tidak terdapat pada array, misalnya 100, semua data juga harus diperiksa.

**Worst Case = O(n)**

| Kondisi | Jumlah Perbandingan | Kompleksitas |
|---|---|---|
| Target di awal | 1 | O(1) |
| Target di tengah | ≈ n/2 | O(n) |
| Target di akhir | n | O(n) |
| Target tidak ditemukan | n | O(n) |

**6. Contoh Eksperimen Waktu Eksekusi**

Sesuai kegiatan praktikum Modul 4, mahasiswa dapat menjalankan algoritma dengan beberapa ukuran input dan mencatat waktu eksekusinya.

Misalnya digunakan:

- n = 1.000
- n = 5.000
- n = 10.000
- n = 50.000

**Contoh Kode Eksperimen C++**

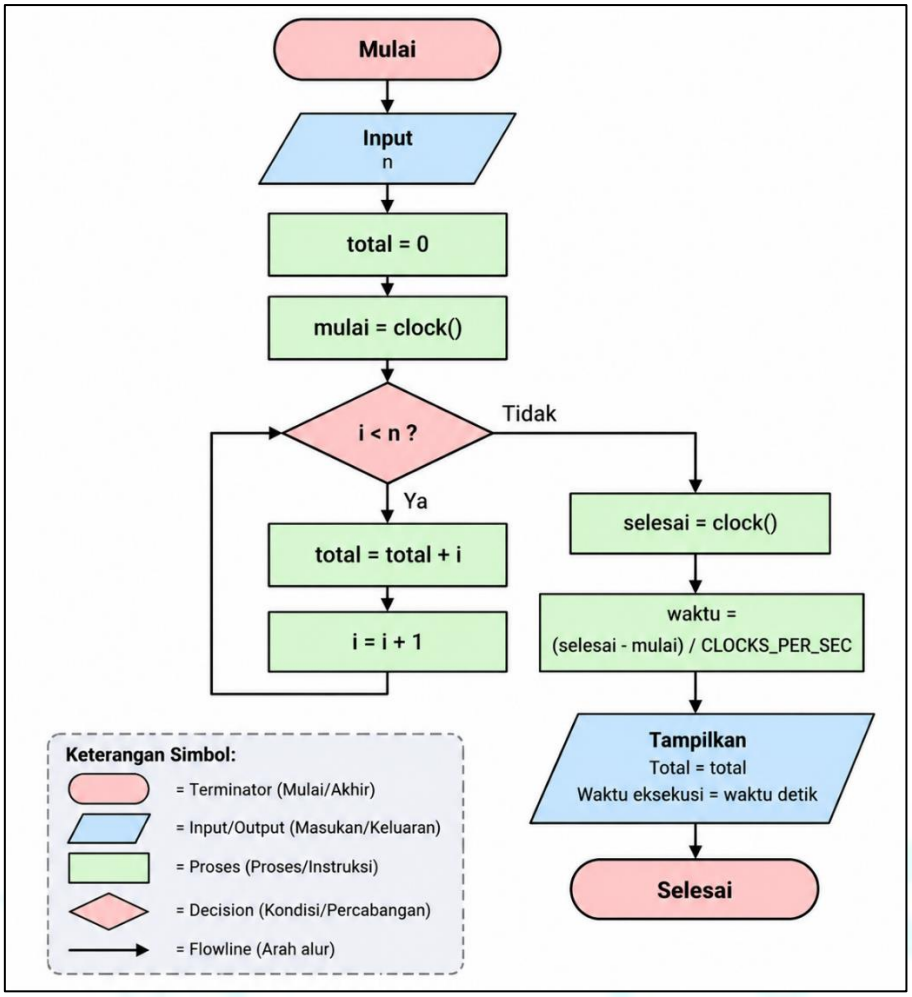
```cpp
#include <iostream>
#include <ctime>
using namespace std;

int main() {

    int n;
    cout << "Masukkan n: ";
    cin >> n;

    long long total = 0;

    clock_t mulai = clock();

    for (int i = 0; i < n; i++) {
        total += i;
    }

    clock_t selesai = clock();

    double waktu =
        double(selesai - mulai) / CLOCKS_PER_SEC;

    cout << "Total = " << total << endl;
    cout << "Waktu eksekusi = "
         << waktu
         << " detik" << endl;

    return 0;
}
```

**Tabel Hasil Eksperimen**

Mahasiswa mengisi berdasarkan hasil komputer masing-masing.

| n | Kompleksitas Teoretis | Waktu Eksekusi |
|---|---|---|
| 1.000 | O(n) | … µs |
| 5.000 | O(n) | … µs |
| 10.000 | O(n) | … µs |
| 50.000 | O(n) | … µs |

Waktu nyata tidak harus meningkat secara tepat karena dipengaruhi perangkat keras, sistem operasi, compiler, dan proses lain yang berjalan. Yang diamati adalah kecenderungan pertumbuhannya.




Gambar 6. Flowchart Menghitung Total dan Waktu Eksekusi

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Mengidentifikasi operasi dominan pada beberapa algoritma sederhana.

b) Menentukan kompleksitas Big-O dari struktur sekuensial, loop tunggal, dan loop bersarang.
c) Membuat program eksperimen untuk beberapa ukuran masukan.
d) Mengukur waktu eksekusi dan mencatat hasil pengujian.
e) Membandingkan hasil eksperimen dengan analisis Big-O teoretis.

**4. Studi Kasus**

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Analisis Tiga Potongan Kode | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Perbandingan Pencarian | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Matriks dan Nested Loop | 40 |
| | | **Total** | **100** |

**1) Studi Kasus 1 - Analisis Tiga Potongan Kode**

Buat tiga program: operasi konstan, loop tunggal n kali, dan loop bersarang n x n. Tentukan Big-O teoretis, ukur waktu pada beberapa n, lalu bandingkan.

**2) Studi Kasus 2 - Perbandingan Pencarian**

Implementasikan pencarian sederhana pada data berukuran meningkat. Catat jumlah perbandingan untuk target di awal dan akhir, lalu jelaskan hubungan dengan O(n).

**3) Studi Kasus 3 - Matriks dan Nested Loop**

Buat program menjumlahkan seluruh elemen matriks n x n. Analisis waktu dan ruang, kemudian ukur untuk n bertambah dan jelaskan pertumbuhan O(n^2).

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 4

Analisis Efisiensi Tiga Algoritma Pengolahan Data

Diberikan tiga algoritma berikut.

**Algoritma A**

```
INPUT n
hasil ← n × 2
OUTPUT hasil
```

**Algoritma B**

```
INPUT n
total ← 0

FOR i ← 1 TO n DO
    total ← total + i
ENDFOR

OUTPUT total
```

**Algoritma C**

```
INPUT n
jumlah ← 0

FOR i ← 1 TO n DO
    FOR j ← 1 TO n DO
        jumlah ← jumlah + 1
    ENDFOR
ENDFOR

OUTPUT jumlah
```

**Soal Evaluasi**

1. Identifikasi operasi dominan pada Algoritma A, B, dan C.
2. Tentukan fungsi jumlah operasi T(n) dan kompleksitas Big-O masing-masing algoritma.

Lengkapi tabel berikut:

| Algoritma | Operasi Dominan | T(n) | Big-O |
|---|---|---|---|
| A | ... | ... | ... |
| B | ... | ... | ... |
| C | ... | ... | ... |

3. Hitung perkiraan jumlah operasi untuk ukuran input berikut:

| n | Algoritma A | Algoritma B | Algoritma C |
|---|---|---|---|
| 10 | ... | ... | ... |
| 100 | ... | ... | ... |
| 1.000 | ... | ... | ... |
| 10.000 | ... | ... | ... |

4. Implementasikan ketiga algoritma tersebut menggunakan Dev C++.
5. Tambahkan pengukuran waktu eksekusi pada setiap algoritma, kemudian lakukan eksperimen menggunakan ukuran data:
   - n = 1.000
   - n = 5.000
   - n = 10.000
   - n = 20.000
   - n = 50.000

   Catat hasilnya pada tabel berikut:

| n | Waktu A | Waktu B | Waktu C |
|---|---|---|---|
| 1.000 | ... | ... | ... |
| 5.000 | ... | ... | ... |
| 10.000 | ... | ... | ... |
| 20.000 | ... | ... | ... |
| 50.000 | ... | ... | ... |

6. Buat grafik sederhana yang membandingkan **ukuran input n terhadap waktu eksekusi** ketiga algoritma.
7. Analisis hasil eksperimen dengan menjawab:
   b. Algoritma mana yang memiliki pertumbuhan waktu paling lambat?
   c. Algoritma mana yang mengalami peningkatan waktu paling cepat ketika n bertambah?
   d. Apakah hasil eksperimen sesuai dengan analisis Big-O teoretis? Jelaskan.
   e. Mengapa waktu eksekusi nyata tidak selalu meningkat secara tepat sesuai nilai n?

f. Apa pengaruh loop bersarang terhadap efisiensi algoritma?

8. Perhatikan algoritma berikut:

```
FOR i ← 1 TO n DO
    OUTPUT i
ENDFOR

FOR j ← 1 TO n DO
    OUTPUT j
ENDFOR
```

Tentukan kompleksitas waktunya dan jelaskan apakah kompleksitasnya:

- O(n)
- O(2n)
- atau O($n^2$)

Berikan alasan.

9. Analisis algoritma berikut:

```
FOR i ← 1 TO n DO
    FOR j ← 1 TO 10 DO
        OUTPUT i, j
    ENDFOR
ENDFOR
```

Tentukan:

- jumlah operasi;
- bentuk T(n);
- kompleksitas Big-O;
- alasan mengapa loop kedua tidak menyebabkan kompleksitas menjadi O($n^2$).

10. Buat kesimpulan singkat mengenai perbedaan karakteristik:

- **O(1)**
- **O(n)**
- **O($n^2$)**

**Hasil/Luaran yang Dikumpulkan**

1. Tabel identifikasi operasi dominan.
2. Tabel perhitungan T(n) dan Big-O.
3. Kode program Algoritma A, B, dan C.
4. Tabel hasil pengukuran waktu eksekusi.
5. Grafik perbandingan waktu eksekusi.
6. Analisis kesesuaian hasil eksperimen dengan Big-O teoretis.

7. Kesimpulan efisiensi masing-masing algoritma.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL08 | CPMK084 | Ketepatan mengidentifikasi operasi dominan | 15 |
| 2 | CPL08 | CPMK084 | Ketepatan menentukan T(n) | 15 |
| 3 | CPL08 | CPMK084 | Ketepatan menentukan notasi Big-O | 20 |
| 4 | CPL08 | CPMK084 | Ketepatan implementasi eksperimen | 15 |
| 5 | CPL08 | CPMK084 | Ketepatan pengukuran dan pencatatan waktu eksekusi | 15 |
| 6 | CPL08 | CPMK084 | Kemampuan membandingkan hasil eksperimen dengan teori | 10 |
| 7 | CPL08 | CPMK084 | Kemampuan menarik kesimpulan mengenai efisiensi algoritma | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 05

# Implementasi dan Optimasi Algoritma Seleksi

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL03** | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. |
| **CPMK032** | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. |
| **Sub-CPMK** | Mahasiswa mampu merancang dan mengimplementasikan algoritma seleksi serta mengevaluasi efisiensi dan ketepatan logika percabangan. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu merancang dan mengimplementasikan struktur seleksi tunggal, dua arah, majemuk, dan bertingkat sesuai kebutuhan masalah. Mahasiswa mampu menggunakan operator logika secara tepat, menemukan false logic, melakukan tracing jalur keputusan, serta menyederhanakan struktur seleksi tanpa mengubah kebenaran hasil.

## D. Indikator Ketercapaian Pembelajaran

1. Menerapkan seleksi tunggal, dua arah, dan majemuk sesuai kebutuhan.
2. Menggunakan operator perbandingan dan logika AND, OR, dan NOT dengan tepat.
3. Menyusun urutan kondisi agar tidak tumpang tindih atau menutup kondisi yang lebih khusus.
4. Menemukan dan memperbaiki false logic selection.
5. Melakukan tracing jalur keputusan untuk berbagai test case.
6. Membandingkan dan mengoptimalkan struktur seleksi dengan tetap menjaga kebenaran keluaran.

## E. Teori Penunjang

Materi ini menekankan pemahaman konsep sebelum implementasi. Mahasiswa perlu melihat algoritma sebagai model penyelesaian masalah yang dapat dijelaskan, diuji, dan dianalisis. Pemahaman tersebut membantu membedakan antara sekadar kode yang dapat dijalankan dengan solusi yang

benar, terstruktur, dan dapat dipertanggungjawabkan. Konsep utama berikut menjadi dasar untuk kegiatan praktikum dan digunakan kembali pada modul-modul berikutnya.

1. **Seleksi tunggal**

Seleksi tunggal menjalankan suatu blok hanya ketika kondisi bernilai benar. Bentuk ini sesuai untuk tindakan opsional seperti memberi bonus jika target tercapai. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

2. **Seleksi dua arah**

If-else memilih tepat satu dari dua jalur. Kondisi harus dirancang agar kedua kemungkinan saling melengkapi. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

3. **Seleksi majemuk/bertingkat**

Beberapa kategori dapat ditangani dengan else-if atau seleksi bertingkat. Urutan pemeriksaan penting karena kondisi yang lebih umum dapat menutup kondisi yang lebih khusus. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

4. **Ekspresi logika**

Operator perbandingan dan logika AND, OR, dan NOT digunakan untuk membentuk aturan keputusan. Prioritas operator perlu diperhatikan agar kondisi sesuai maksud. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

**5. Optimasi seleksi**

Optimasi dilakukan dengan mengurangi pemeriksaan yang tidak perlu, menghindari kondisi duplikat, dan menempatkan kondisi yang paling membatasi pada posisi yang tepat. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.

**6. Mekanisme Kerja dan Contoh Penerapan**

Perancangan seleksi dimulai dari tabel aturan. Setiap kategori keluaran dituliskan bersama syaratnya, lalu diperiksa apakah terdapat celah atau tumpang tindih. Kondisi dapat disederhanakan menggunakan rentang nilai. Jika beberapa kondisi memiliki aksi sama, kondisi tersebut dapat digabung. Untuk aturan bertingkat, periksa urutan dari kondisi paling spesifik atau sesuai prioritas bisnis. Tujuan optimasi bukan sekadar memendekkan kode, tetapi menjaga kebenaran dan keterbacaan.

Dalam praktikum, mekanisme tersebut tidak cukup hanya dibaca. Setiap langkah harus dapat dijelaskan dengan bahasa sendiri dan ditunjukkan melalui data uji. Gunakan contoh berukuran kecil agar perubahan data mudah diamati. Setelah hasil manual konsisten, implementasi dapat diperluas ke ukuran input yang lebih besar. Cara ini memisahkan kesalahan konsep dari kesalahan sintaks dan mempercepat proses debugging.

**Contoh Penerapan**

**Penentuan Potongan UKT Mahasiswa**

Sebuah sistem menentukan potongan UKT mahasiswa berdasarkan IPK dan penghasilan orang tua.

Ketentuan:

- Jika IPK ≥ 3,75 dan penghasilan orang tua ≤ Rp3.000.000 → potongan 50%
- Jika IPK ≥ 3,50 dan penghasilan orang tua ≤ Rp5.000.000 → potongan 30%

- Jika IPK ≥ 3,00 → potongan 10%
- Selain itu → tidak mendapat potongan
- IPK harus berada pada rentang 0–4.
- Penghasilan tidak boleh negatif.

Studi kasus ini memungkinkan mahasiswa menguji seleksi majemuk, kondisi logika, urutan kondisi, dan optimasi percabangan.

1. Penerapan Seleksi Tunggal

   Misalnya terdapat aturan tambahan:

   Mahasiswa mendapat predikat akademik apabila IPK ≥ 3,50.

   Pseudocode:

```
INPUT ipk

IF ipk >= 3.50 THEN
    OUTPUT "Mahasiswa Berprestasi"
ENDIF
```

   Pada seleksi tunggal, proses hanya dijalankan apabila kondisi bernilai benar.

2. Penerapan Seleksi Dua Arah

   Contoh untuk menentukan apakah mahasiswa memenuhi batas minimum IPK:

```
INPUT ipk

IF ipk >= 3.00 THEN
    OUTPUT "Memenuhi Syarat"
ELSE
    OUTPUT "Tidak Memenuhi Syarat"
ENDIF
```

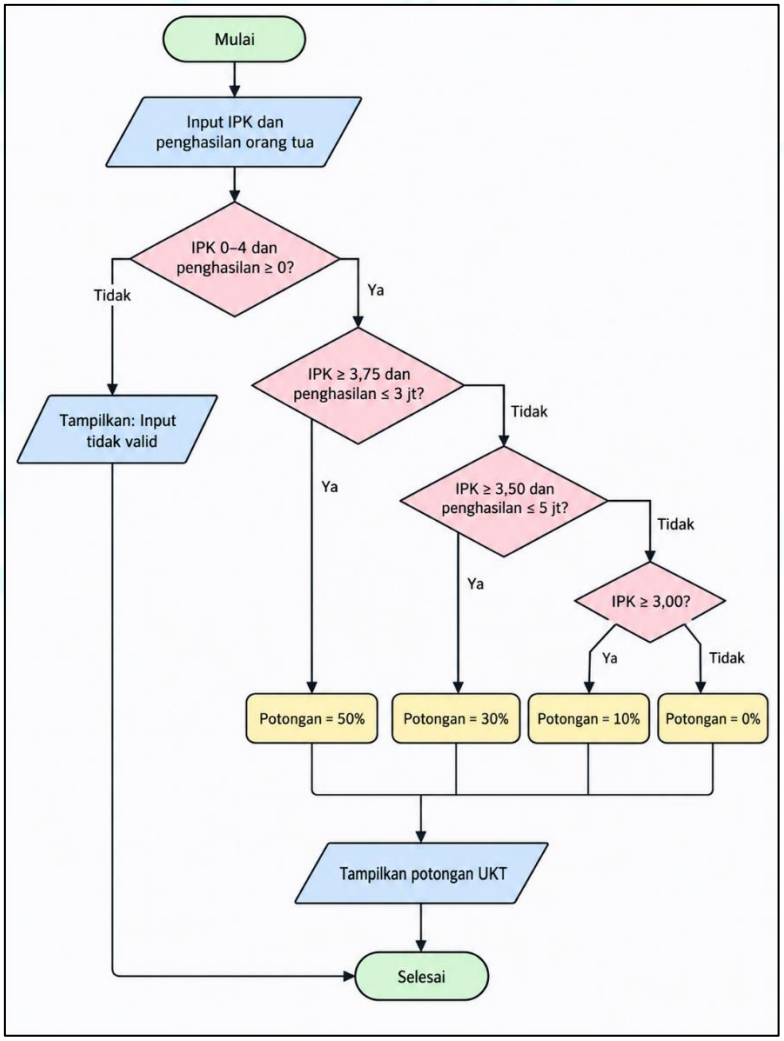
   Struktur **IF–ELSE** menghasilkan tepat satu dari dua kemungkinan keluaran.

3. Penerapan Seleksi Majemuk/Bertingkat

   Untuk menentukan besarnya potongan UKT:

```
INPUT ipk
INPUT penghasilan

IF ipk >= 3.75 AND penghasilan <= 3000000 THEN
    potongan ← 50

ELSE IF ipk >= 3.50 AND penghasilan <= 5000000 THEN
    potongan ← 30

ELSE IF ipk >= 3.00 THEN
    potongan ← 10

ELSE
    potongan ← 0
ENDIF

OUTPUT potongan
```

Urutan pemeriksaan sangat penting karena kondisi paling khusus sebaiknya ditempatkan lebih dahulu.




Gambar 7. Flowchart Penentuan Potongan UKT

4. Penerapan Kondisi Logika

   Operator logika dapat digunakan untuk menggabungkan beberapa persyaratan.

   **Operator AND**

```
IF ipk >= 3.75 AND penghasilan <= 3000000 THEN
    potongan ← 50
ENDIF
```

   Kedua kondisi harus bernilai benar.

   **Operator OR**

   Misalnya mahasiswa diperbolehkan mengikuti program khusus jika memiliki IPK tinggi **atau** prestasi nasional:

```
IF ipk >= 3.75 OR prestasiNasional = TRUE THEN
    OUTPUT "Memenuhi Syarat"
ENDIF
```

   **Operator NOT**

```
IF NOT statusAktif THEN
    OUTPUT "Mahasiswa Tidak Aktif"
ENDIF
```

5. Contoh False Logic Selection

   Perhatikan algoritma berikut:

```
IF ipk >= 3.00 THEN
    potongan ← 10

ELSE IF ipk >= 3.50 THEN
    potongan ← 30

ELSE IF ipk >= 3.75 THEN
    potongan ← 50
ENDIF
```

   Algoritma tersebut **salah secara logika**.

   Jika IPK = 3.80, kondisi pertama:

   ipk >= 3.00

sudah bernilai benar. Akibatnya program langsung memberikan potongan 10% dan tidak pernah memeriksa kondisi 30% atau 50%.

**Perbaikannya**

Susun kondisi dari nilai tertinggi atau kondisi paling khusus:

```
IF ipk >= 3.75 THEN
    potongan ← 50

ELSE IF ipk >= 3.50 THEN
    potongan ← 30

ELSE IF ipk >= 3.00 THEN
    potongan ← 10

ELSE
    potongan ← 0
ENDIF
```

6. Validasi Input Menggunakan Seleksi

Sebelum menentukan potongan, input sebaiknya divalidasi.

```
INPUT ipk
INPUT penghasilan

IF ipk < 0 OR ipk > 4 THEN
    OUTPUT "IPK tidak valid"

ELSE IF penghasilan < 0 THEN
    OUTPUT "Penghasilan tidak valid"

ELSE
    proses penentuan potongan
ENDIF
```

Validasi mencegah program memproses data yang tidak sesuai.

7. Optimasi Struktur Seleksi

   Misalkan digunakan struktur seperti ini:

```
IF ipk >= 3.75 THEN
    IF penghasilan <= 3000000 THEN
        potongan ← 50
    ELSE
        IF penghasilan <= 5000000 THEN
            potongan ← 30
        ELSE
            potongan ← 10
        ENDIF
    ENDIF
ELSE
    ...
ENDIF
```

   Struktur tersebut dapat bekerja, tetapi menjadi lebih panjang dan sulit dibaca.

   Versi yang lebih sederhana

```
IF ipk >= 3.75 AND penghasilan <= 3000000 THEN
    potongan ← 50

ELSE IF ipk >= 3.50 AND penghasilan <= 5000000 THEN
    potongan ← 30

ELSE IF ipk >= 3.00 THEN
    potongan ← 10

ELSE
    potongan ← 0
ENDIF
```

   Optimasi di sini bukan hanya mengurangi kode, tetapi juga membuat kondisi:

   - lebih mudah dibaca;
   - lebih mudah diuji;
   - lebih mudah ditelusuri;
   - mengurangi kemungkinan kesalahan logika.

8. Contoh Tracing

   Gunakan beberapa test case berikut.

| Test Case | IPK | Penghasilan | Potongan |
|---|---|---|---|
| 1 | 3,85 | Rp2.500.000 | 50% |
| 2 | 3,60 | Rp4.000.000 | 30% |
| 3 | 3,20 | Rp7.000.000 | 10% |

| | | | |
|---|---|---|---|
| 4 | 2,80 | Rp3.000.000 | 0% |
| 5 | 4,50 | Rp2.000.000 | Input tidak valid |

**Contoh tracing Test Case 1**

Input:

- IPK = 3,85
- Penghasilan = Rp2.500.000

Pemeriksaan:

```
IPK >= 3.75                    → TRUE
Penghasilan <= 3.000.000       → TRUE
```

Karena:

TRUE AND TRUE = TRUE

maka:

**Potongan = 50%**

9. Contoh Implementasi Dev C++

```cpp
#include <iostream>
using namespace std;

int main() {

    double ipk;
    long long penghasilan;
    int potongan;

    cout << "Masukkan IPK: ";
    cin >> ipk;

    cout << "Masukkan Penghasilan Orang Tua: ";
    cin >> penghasilan;

    if (ipk < 0 || ipk > 4) {
        cout << "IPK tidak valid";

    } else if (penghasilan < 0) {
        cout << "Penghasilan tidak valid";

    } else {

        if (ipk >= 3.75 && penghasilan <= 3000000) {
            potongan = 50;

        } else if (ipk >= 3.50 && penghasilan <= 5000000) {
            potongan = 30;

        } else if (ipk >= 3.00) {
            potongan = 10;

        } else {
            potongan = 0;
        }

        cout << "Potongan UKT = "
             << potongan << "%" << endl;
    }

    return 0;
}
```

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Mengimplementasikan algoritma seleksi untuk menyelesaikan kasus yang diberikan.
b) Menguji kondisi tunggal, kondisi majemuk, dan seleksi bertingkat.
c) Mengidentifikasi dan memperbaiki kesalahan logika pada struktur seleksi.
d) Membandingkan dua rancangan seleksi yang menghasilkan keluaran sama.
e) Menganalisis jumlah pemeriksaan kondisi dan memilih rancangan yang lebih efisien.

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 1: Seleksi Beasiswa | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 2: Tarif Pengiriman | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 3: Bonus Karyawan | 40 |
| | | **Total** | **100** |

#### 1) Studi Kasus 1 - Seleksi Beasiswa

Tentukan kelayakan berdasarkan IPK, penghasilan orang tua, dan status prestasi. Susun tabel keputusan, implementasikan if-else, dan uji batas tiap syarat.

**2) Studi Kasus 2 - Tarif Pengiriman**

Hitung tarif berdasarkan berat, jenis layanan, dan tujuan. Buat minimal tiga kategori dan optimalkan struktur seleksi agar tidak ada kondisi tumpang tindih.

**3) Studi Kasus 3 - Bonus Karyawan**

Bonus ditentukan oleh masa kerja, penilaian kinerja, dan target. Buat dua rancangan seleksi yang menghasilkan output sama, lalu bandingkan jumlah kondisi yang diperiksa.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 5

**Sistem Penentuan Tarif Parkir**

Sebuah pusat perbelanjaan menerapkan tarif parkir berdasarkan **jenis kendaraan, lama parkir, status member, dan waktu parkir**.

Ketentuan:

1. Jenis kendaraan:
   a. Motor: tarif dasar Rp3.000
   b. Mobil: tarif dasar Rp5.000
2. Tambahan biaya per jam setelah 2 jam pertama:
   a. Motor: Rp2.000/jam
   b. Mobil: Rp3.000/jam
3. Jika lama parkir ≤ 2 jam, hanya dikenakan tarif dasar.
4. Jika lama parkir > 2 jam, dikenakan tambahan tarif sesuai kelebihan jam.
5. Jika pelanggan berstatus **Member**, mendapat diskon 10%.
6. Jika waktu keluar setelah pukul 22.00, dikenakan tambahan biaya malam Rp5.000.
7. Total pembayaran tidak boleh bernilai negatif.
8. Jenis kendaraan selain Motor dan Mobil dianggap **tidak valid**.
9. Lama parkir harus lebih dari 0 jam.

**Soal Evaluasi**

1. Identifikasi komponen **input, proses, kondisi seleksi, dan output** dari kasus tersebut.
2. Buat pseudocode untuk:
   a. menerima jenis kendaraan;
   b. menerima lama parkir;
   c. menerima status member;
   d. menerima jam keluar;
   e. menentukan tarif dasar;
   f. menghitung biaya tambahan parkir;
   g. menentukan diskon member;
   h. menentukan tambahan biaya malam;
   i. menghitung total bayar.
3. Buat flowchart menggunakan Flowrun.io atau Flowgorithm yang memuat seluruh proses seleksi.
4. Lakukan tracing pada data berikut:

| Test Case | Kendaraan | Lama Parkir | Member | Jam Keluar | Total Bayar |
|---|---|---|---|---|---|
| 1 | Motor | 1 | Tidak | 18.00 | ... |
| 2 | Motor | 5 | Ya | 20.00 | ... |
| 3 | Mobil | 4 | Tidak | 23.00 | ... |
| 4 | Mobil | 6 | Ya | 23.30 | ... |
| 5 | Bus | 3 | Tidak | 19.00 | ... |
| 6 | Motor | 0 | Ya | 21.00 | ... |

5. Perhatikan algoritma berikut:

```
IF kendaraan = "Motor" THEN
    tarifDasar ← 3000

IF kendaraan = "Mobil" THEN
    tarifDasar ← 5000
ELSE
    OUTPUT "Kendaraan tidak valid"
ENDIF
```

Jelaskan kesalahan logika yang terjadi dan perbaiki menggunakan struktur seleksi yang benar.

6. Analisis kondisi berikut:

```
IF lamaParkir > 2 OR member = "Ya" THEN
    diskon ← 10%
ENDIF
```

Apakah penggunaan operator OR sudah tepat untuk menentukan diskon member? Jelaskan dan perbaiki.

7. Buat dua versi algoritma:
   - versi dengan banyak IF terpisah;
   - versi menggunakan IF–ELSE IF–ELSE.

   Bandingkan keduanya dari sisi:
   - jumlah pemeriksaan kondisi;
   - keterbacaan;
   - risiko false logic;
   - kemudahan pemeliharaan.

8. Jelaskan hasil yang seharusnya terjadi jika:
   a. jenis kendaraan = Bus;

b. lama parkir = 0;
c. mobil parkir 5 jam dan keluar pukul 23.00;
d. pelanggan Member keluar sebelum pukul 22.00;
e. pelanggan Member keluar setelah pukul 22.00.

**Hasil/Luaran yang Dikumpulkan**

1. Tabel input–proses–kondisi–output.
2. Pseudocode algoritma.
3. Flowchart.
4. Tabel tracing.
5. Analisis false logic.
6. Pseudocode sebelum dan sesudah optimasi.
7. File program Dev C++.
8. Kesimpulan mengenai struktur seleksi yang paling efisien.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL03 | CPMK032 | Ketepatan penerapan seleksi tunggal dan majemuk | 20 |
| 2 | CPL03 | CPMK032 | Ketepatan penggunaan operator logika | 15 |
| 3 | CPL03 | CPMK032 | Ketepatan urutan kondisi seleksi | 15 |
| 4 | CPL03 | CPMK032 | Kemampuan menemukan dan memperbaiki false logic | 15 |
| 5 | CPL03 | CPMK032 | Kemampuan mengoptimalkan struktur pemilihan | 15 |
| 6 | CPL03 | CPMK032 | Ketepatan tracing dan hasil keputusan | 10 |
| 7 | CPL03 | CPMK032 | Kebenaran implementasi program | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 06

# Implementasi dan Optimasi Algoritma Iteratif

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL03** | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. |
| **CPMK032** | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. |
| **Sub-CPMK** | Mahasiswa mampu merancang dan mengimplementasikan algoritma iteratif serta mengevaluasi efisiensi dan ketepatan struktur pengulangan. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu memilih dan menerapkan struktur perulangan for, while, dan do-while sesuai karakteristik masalah. Mahasiswa mampu menggunakan loop bersarang, counter, accumulator, dan kondisi berhenti dengan benar, menemukan penyebab infinite loop, serta membandingkan rancangan pengulangan untuk memperoleh solusi yang lebih efisien.

## D. Indikator Ketercapaian Pembelajaran

1. Memilih struktur for, while, atau do-while sesuai kebutuhan proses.
2. Menerapkan loop tunggal dan loop bersarang secara tepat.
3. Menggunakan counter, accumulator, dan variabel kontrol dengan benar.
4. Menentukan kondisi awal, perubahan variabel, dan kondisi berhenti yang konsisten.
5. Menemukan dan memperbaiki infinite loop serta kesalahan batas iterasi.
6. Membandingkan dan mengoptimalkan dua rancangan loop yang menghasilkan keluaran sama.

## E. Teori Pendukung

Materi ini menekankan pemahaman konsep sebelum implementasi. Mahasiswa perlu melihat algoritma sebagai model penyelesaian masalah yang

dapat dijelaskan, diuji, dan dianalisis. Pemahaman tersebut membantu membedakan antara sekadar kode yang dapat dijalankan dengan solusi yang benar, terstruktur, dan dapat dipertanggungjawabkan. Konsep utama berikut menjadi dasar untuk kegiatan praktikum dan digunakan kembali pada modul-modul berikutnya.

1. **Iterasi terkendali hitungan**

   Perulangan for cocok untuk proses dengan jumlah pengulangan yang telah diketahui, misalnya memproses n data. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

2. **Iterasi terkendali kondisi**

   While digunakan ketika proses diulang selama kondisi masih benar. Pemeriksaan terjadi sebelum tubuh loop sehingga loop dapat tidak dijalankan sama sekali. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

3. **Do-while**

   Do-while menjalankan tubuh loop minimal satu kali karena kondisi diperiksa setelah proses. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

4. **Loop bersarang**

   Loop di dalam loop digunakan untuk data dua dimensi atau kombinasi pasangan. Biaya komputasinya dapat meningkat cepat, sering kali menjadi O(n^2). Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

5. **Kondisi berhenti dan optimasi**

   Loop harus mempunyai perubahan keadaan yang menuju kondisi berhenti. Optimasi dilakukan dengan mengurangi iterasi tidak perlu dan menghentikan

proses lebih awal bila hasil sudah diketahui. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.

**6. Mekanisme Kerja dan Contoh Penerapan**

Sebelum memilih jenis loop, tentukan variabel kontrol, nilai awal, syarat lanjut, dan perubahan setiap iterasi. Empat komponen tersebut harus konsisten. Pada loop bersarang, tentukan hubungan batas loop luar dan dalam. Jika proses pencarian dapat dihentikan saat elemen ditemukan, gunakan mekanisme break secara terkontrol. Jika hanya beberapa data yang perlu dilewati, continue dapat digunakan, tetapi terlalu banyak penggunaan break/continue dapat membuat alur sulit ditelusuri.

Dalam praktikum, mekanisme tersebut tidak cukup hanya dibaca. Setiap langkah harus dapat dijelaskan dengan bahasa sendiri dan ditunjukkan melalui data uji. Gunakan contoh berukuran kecil agar perubahan data mudah diamati. Setelah hasil manual konsisten, implementasi dapat diperluas ke ukuran input yang lebih besar. Cara ini memisahkan kesalahan konsep dari kesalahan sintaks dan mempercepat proses debugging.

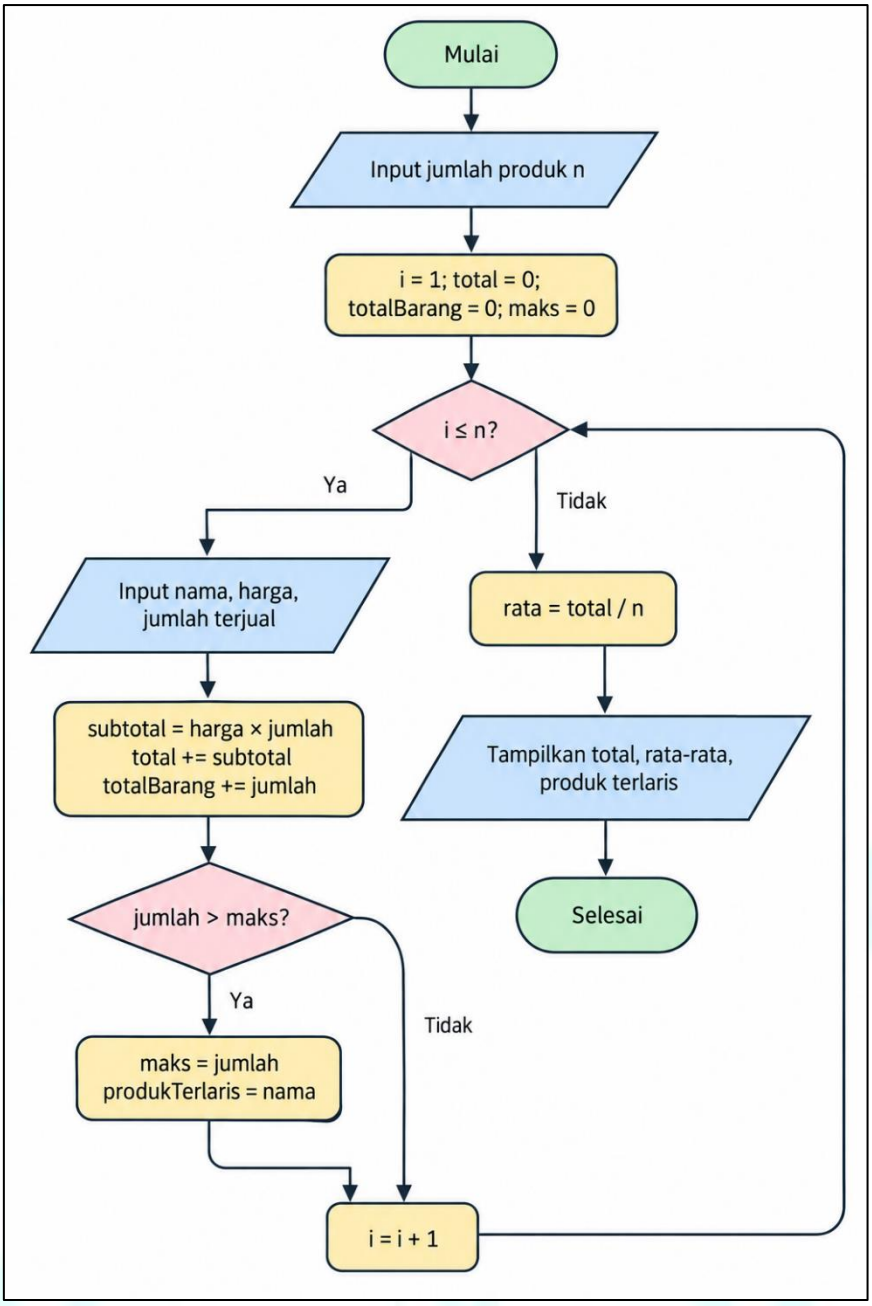
**Contoh Penerapan**

**Rekap Penjualan Produk**

Sebuah toko ingin mengolah data penjualan beberapa produk. Program menerima jumlah produk, kemudian untuk setiap produk dimasukkan nama produk, harga satuan, dan jumlah terjual.

Program harus menghasilkan:

- total penjualan setiap produk;
- total pendapatan seluruh produk;
- total barang terjual;
- rata-rata pendapatan per produk;
- produk dengan jumlah penjualan tertinggi.




Gambar 8. Flowchart Rekap Penjualan Produk

1. Penerapan Loop Terkendali
   Karena jumlah produk telah diketahui, perulangan dapat menggunakan FOR.
   **Pseudocode**

```
INPUT jumlahProduk

totalPendapatan ← 0
totalBarang ← 0

FOR i ← 1 TO jumlahProduk DO

    INPUT namaProduk
    INPUT harga
    INPUT jumlahTerjual

    subtotal ← harga × jumlahTerjual

    totalPendapatan ← totalPendapatan + subtotal
    totalBarang ← totalBarang + jumlahTerjual

    OUTPUT namaProduk, subtotal

ENDFOR

rataRata ← totalPendapatan / jumlahProduk

OUTPUT totalPendapatan
OUTPUT totalBarang
OUTPUT rataRata
```

Jika jumlah produk = 5, maka loop dilakukan tepat **5 kali**.

2. Contoh Tracing Loop

   Gunakan data:

| Iterasi | Produk | Harga | Jumlah | Subtotal | Total Pendapatan |
|---|---|---|---|---|---|
| 1 | Buku | 20.000 | 5 | 100.000 | 100.000 |
| 2 | Pulpen | 5.000 | 10 | 50.000 | 150.000 |
| 3 | Tas | 150.000 | 2 | 300.000 | 450.000 |
| 4 | Penggaris | 8.000 | 4 | 32.000 | 482.000 |

   Hasil:

   **Total Pendapatan = Rp482.000**

   **Total Barang = 21**

   **Rata-rata Pendapatan = Rp120.500**

   Tracing menunjukkan bahwa nilai accumulator totalPendapatan berubah pada setiap iterasi.

3. Penerapan WHILE

   Jika jumlah transaksi tidak diketahui sejak awal, dapat digunakan nilai khusus sebagai **sentinel**.

Misalnya transaksi berhenti jika nama produk diisi "SELESAI".

```
totalPendapatan ← 0

INPUT namaProduk

WHILE namaProduk ≠ "SELESAI" DO

    INPUT harga
    INPUT jumlah

    subtotal ← harga × jumlah

    totalPendapatan ←
        totalPendapatan + subtotal

    INPUT namaProduk
ENDWHILE

OUTPUT totalPendapatan
```

Pada struktur ini, kondisi berhenti bergantung pada input pengguna.

4. Penerapan DO-WHILE

   Jika proses harus dilakukan minimal satu kali:

```
DO

    INPUT nilai

    OUTPUT nilai

    INPUT ulang

WHILE ulang = "Y"
```

   Perbedaannya dengan WHILE adalah kondisi diperiksa **setelah** blok dijalankan.

5. Penerapan Loop Bersarang

   Misalnya toko mencatat penjualan:

   - 3 hari;
   - 4 produk setiap hari.

Maka dapat digunakan dua loop.

```
FOR hari ← 1 TO 3 DO
    OUTPUT "Hari ke-", hari
    FOR produk ← 1 TO 4 DO
        INPUT jumlahTerjual
        OUTPUT jumlahTerjual
    ENDFOR
ENDFOR
```

Jika:

- loop luar = 3 kali;
- loop dalam = 4 kali;

maka proses input dilakukan:

**3 × 4 = 12 kali**

Loop bersarang cocok digunakan apabila data memiliki dua tingkat, misalnya:

- hari × produk;
- mahasiswa × mata kuliah;
- kelas × mahasiswa;
- bulan × transaksi.

6. Menentukan Nilai Terbesar dalam Loop

   Program dapat menentukan produk paling banyak terjual tanpa membuat loop tambahan.

```
maksTerjual ← 0

FOR i ← 1 TO jumlahProduk DO

    INPUT namaProduk
    INPUT jumlahTerjual

    IF jumlahTerjual > maksTerjual THEN
        maksTerjual ← jumlahTerjual
        produkTerlaris ← namaProduk
    ENDIF

ENDFOR

OUTPUT produkTerlaris
OUTPUT maksTerjual
```

Pada setiap iterasi, data baru dibandingkan dengan nilai maksimum sementara.

7. Contoh Infinite Loop
   Perhatikan pseudocode berikut:

```
i ← 1

WHILE i <= 10 DO
    OUTPUT i
ENDWHILE
```

Program akan terus berjalan karena nilai i tidak pernah berubah.

**Penyebab**

Nilai:

i = 1

selalu memenuhi:

i <= 10

sehingga kondisi tidak pernah menjadi FALSE.

**Perbaikan**

```
i ← 1

WHILE i <= 10 DO

    OUTPUT i

    i ← i + 1

ENDWHILE
```

Sekarang nilai i berkembang:

1, 2, 3, 4, ..., 10, 11

Saat i = 11, kondisi menjadi salah dan loop berhenti.

8. Contoh Kesalahan Kondisi Berhenti
   Perhatikan:

```
i ← 10

WHILE i >= 1 DO
    OUTPUT i
    i ← i + 1
ENDWHILE
```

Ini juga menghasilkan infinite loop karena i justru terus bertambah.
Seharusnya:

```
i ← 10

WHILE i >= 1 DO
    OUTPUT i
    i ← i - 1
ENDWHILE
```

9. Optimasi Struktur Pengulangan
   Misalnya ingin menghitung:
   - total seluruh nilai;
   - jumlah nilai ≥ 60.

   **Versi A – Dua Loop**

```
total ← 0

FOR i ← 1 TO n DO
    total ← total + nilai[i]
ENDFOR

jumlahLulus ← 0

FOR i ← 1 TO n DO

    IF nilai[i] >= 60 THEN
        jumlahLulus ← jumlahLulus + 1
    ENDIF

ENDFOR
```

Data diperiksa sebanyak: **n + n = 2n kali**

**Versi B – Satu Loop**

```
total ← 0
jumlahLulus ← 0

FOR i ← 1 TO n DO

    total ← total + nilai[i]

    IF nilai[i] >= 60 THEN
        jumlahLulus ← jumlahLulus + 1
    ENDIF

ENDFOR
```

Data hanya dilewati: **n kali**

Versi B lebih efisien karena dua proses yang menggunakan data sama dilakukan dalam satu perulangan.

10. Perbandingan Efisiensi

| Rancangan | Jumlah Loop | Perkiraan Iterasi |
|---|---|---|
| Dua loop terpisah | 2 | 2n |
| Satu loop terintegrasi | 1 | n |
| Dua loop bersarang | 2 tingkat | $n^2$ |

Walaupun O(2n) disederhanakan menjadi **O(n)** dalam Big-O, mengurangi jumlah operasi tetap dapat meningkatkan efisiensi praktis program.

11. Eksperimen Waktu Eksekusi

Mahasiswa dapat membandingkan waktu antara dua rancangan loop.

Contoh ukuran data:

| n | Dua Loop | Satu Loop |
|---|---|---|
| 1.000 | … µs | … µs |
| 10.000 | … µs | … µs |
| 100.000 | … µs | … µs |
| 1.000.000 | … µs | … µs |

Mahasiswa kemudian menyimpulkan apakah penggabungan proses ke dalam satu loop memberikan penghematan waktu.

12. Contoh Implementasi Dev C++

```cpp
#include <iostream>
using namespace std;

int main() {

    int jumlahProduk;
    int jumlahTerjual;
    int totalBarang = 0;
    int maksTerjual = 0;

    double harga;
    double subtotal;
    double totalPendapatan = 0;

    string namaProduk;
    string produkTerlaris;

    cout << "Jumlah produk: ";
    cin >> jumlahProduk;

    for (int i = 1; i <= jumlahProduk; i++) {

        cout << "\nProduk ke-" << i << endl;
        cout << "Nama produk: ";
        cin >> namaProduk;
        cout << "Harga: ";
        cin >> harga;
        cout << "Jumlah terjual: ";
        cin >> jumlahTerjual;
        subtotal = harga * jumlahTerjual;
        totalPendapatan += subtotal;
        totalBarang += jumlahTerjual;

        if (jumlahTerjual > maksTerjual) {
            maksTerjual = jumlahTerjual;
            produkTerlaris = namaProduk;
        }

        cout << "Subtotal = Rp"
             << subtotal << endl;
    }
    cout << "\nTotal Pendapatan = Rp"
         << totalPendapatan << endl;

    cout << "Total Barang Terjual = "
         << totalBarang << endl;

    cout << "Rata-rata Pendapatan = Rp"
         << totalPendapatan / jumlahProduk
         << endl;

    cout << "Produk Terlaris = "
         << produkTerlaris << endl;

    return 0;
}
```

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Mengimplementasikan algoritma iteratif untuk memproses sekumpulan data.
b) Membuat variasi perulangan dengan kondisi berhenti yang berbeda.
c) Mengidentifikasi penyebab infinite loop dan memperbaikinya.
d) Membandingkan efisiensi dua rancangan loop yang setara.
e) Mengukur waktu eksekusi untuk beberapa ukuran data.

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 1: Rekap Penjualan Harian | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 2: Bilangan Prima pada Rentang | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 3: Pertumbuhan Tabungan | 40 |
| | | **Total** | **100** |

#### 1) Studi Kasus 1 - Rekap Penjualan Harian

Baca n transaksi, hitung total, rata-rata, transaksi terbesar, dan jumlah transaksi di atas target. Gunakan satu loop seefisien mungkin.

#### 2) Studi Kasus 2 - Bilangan Prima pada Rentang

Tampilkan bilangan prima dari 2 sampai n. Gunakan loop bersarang dan analisis bagian yang dapat dioptimalkan untuk mengurangi pemeriksaan.

**3) Studi Kasus 3 - Pertumbuhan Tabungan**

Hitung saldo setiap bulan sampai target tercapai dengan bunga tetap. Bandingkan implementasi while dan do-while serta pastikan kondisi berhenti benar.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 6

**Sistem Rekap Penjualan Harian Minimarket**

Sebuah minimarket ingin membuat program untuk mengolah transaksi penjualan selama beberapa hari. Program menerima jumlah hari pengamatan dan jumlah transaksi pada setiap hari.

Untuk setiap transaksi dimasukkan:

- Nomor transaksi
- Total belanja
- Jenis pembayaran: Tunai atau Non-Tunai

Program harus melakukan proses berikut:

- Menghitung total pendapatan setiap hari.
- Menghitung jumlah transaksi setiap hari.
- Menghitung total pendapatan seluruh periode.
- Menghitung jumlah seluruh transaksi.
- Menentukan transaksi dengan nilai terbesar.
- Menghitung jumlah transaksi Tunai dan Non-Tunai.
- Menghitung rata-rata nilai transaksi.
- Menentukan hari dengan pendapatan tertinggi.
- Program harus memastikan setiap loop memiliki kondisi berhenti yang benar.

**Soal Evaluasi**

1. Identifikasi bagian algoritma yang membutuhkan:
   a. loop tunggal;
   b. loop bersarang;
   c. counter;
   d. accumulator;
   e. pencarian nilai maksimum;
   f. kondisi berhenti.
2. Susun pseudocode yang menggunakan loop bersarang dengan ketentuan:
   a. loop luar digunakan untuk memproses hari;
   b. loop dalam digunakan untuk memproses transaksi pada setiap hari;
   c. total pendapatan harian di-reset setiap memasuki hari baru;
   d. total keseluruhan terus diakumulasi sampai seluruh hari selesai.
3. Buat flowchart menggunakan Flowrun.io atau Flowgorithm yang memuat:
   a. input jumlah hari;

b. perulangan setiap hari;
c. input jumlah transaksi;
d. perulangan transaksi;
e. perhitungan total harian;
f. perhitungan total keseluruhan;
g. pencarian transaksi terbesar;
h. penentuan hari dengan pendapatan tertinggi.

4. Lakukan tracing menggunakan data berikut:

**Hari ke-1**

| Transaksi | Total Belanja | Pembayaran | Total Harian Sementara |
|---|---|---|---|
| 1 | Rp150.000 | Tunai | … |
| 2 | Rp275.000 | Non-Tunai | … |
| 3 | Rp100.000 | Tunai | … |

**Hari ke-2**

| Transaksi | Total Belanja | Pembayaran | Total Harian Sementara |
|---|---|---|---|
| 1 | Rp500.000 | Non-Tunai | … |
| 2 | Rp250.000 | Tunai | … |
| 3 | Rp400.000 | Non-Tunai | … |
| 4 | Rp200.000 | Tunai | … |

**Hari ke-3**

| Transaksi | Total Belanja | Pembayaran | Total Harian Sementara |
|---|---|---|---|
| 1 | Rp300.000 | Tunai | … |
| 2 | Rp350.000 | Non-Tunai | … |

5. Berdasarkan hasil tracing, lengkapi rekapitulasi berikut:

| Hasil Rekapitulasi | Nilai |
|---|---|
| Total pendapatan Hari 1 | Rp … |
| Total pendapatan Hari 2 | Rp … |
| Total pendapatan Hari 3 | Rp … |
| Total pendapatan seluruh periode | Rp … |
| Jumlah seluruh transaksi | … |
| Jumlah transaksi Tunai | … |
| Jumlah transaksi Non-Tunai | … |
| Transaksi terbesar | Rp … |
| Rata-rata transaksi | Rp … |
| Hari dengan pendapatan tertinggi | Hari ke-… |

6. Perhatikan algoritma berikut:

```
i ← 1

WHILE i <= 10 DO
    OUTPUT i
ENDWHILE
```

Jelaskan:

a. Apa kesalahan algoritma tersebut?
b. Mengapa terjadi **infinite loop**?
c. Perbaiki pseudocode agar loop berhenti dengan benar.

7. Perhatikan algoritma berikut:

```
i ← 10

WHILE i >= 1 DO
    OUTPUT i

    i ← i + 1
ENDWHILE
```

Jelaskan:

a. apakah loop akan berhenti;
b. penyebab kesalahan;
c. bentuk perbaikan yang benar.

8. Bandingkan dua algoritma berikut.

**Algoritma A**

```
total ← 0

FOR i ← 1 TO n DO
    total ← total + transaksi[i]
ENDFOR

jumlahBesar ← 0

FOR i ← 1 TO n DO
    IF transaksi[i] >= 500000 THEN
        jumlahBesar ← jumlahBesar + 1
    ENDIF
ENDFOR
```

**Algoritma B**

```
total ← 0
jumlahBesar ← 0

FOR i ← 1 TO n DO
    total ← total + transaksi[i]

    IF transaksi[i] >= 500000 THEN
        jumlahBesar ← jumlahBesar + 1
    ENDIF
ENDFOR
```

Jelaskan:

a. Apakah hasil kedua algoritma sama?
b. Berapa kali data dilewati pada masing-masing algoritma?
c. Algoritma mana yang lebih efisien?
d. Mengapa penggabungan proses dapat menjadi bentuk optimasi loop?

9. Analisis loop bersarang berikut:

```
FOR hari ← 1 TO 5 DO

    FOR transaksi ← 1 TO 10 DO
        OUTPUT hari, transaksi
    ENDFOR

ENDFOR
```

Tentukan:

a. berapa kali loop luar dijalankan;
b. berapa kali loop dalam dijalankan untuk setiap hari;
c. berapa total proses yang dilakukan;
d. bagaimana jumlah proses berubah apabila jumlah hari dan transaksi dinyatakan sebagai **n**.

10. Modifikasi program sehingga proses transaksi pada suatu hari dapat dihentikan menggunakan nilai sentinel:

totalBelanja = -1

Artinya, jika pengguna memasukkan -1, maka input transaksi untuk hari tersebut dihentikan dan program melanjutkan ke hari berikutnya.

Tuliskan pseudocode menggunakan WHILE.

11. Implementasikan algoritma final menggunakan **Dev C++** dan uji dengan minimal tiga skenario:
    - jumlah transaksi sedikit;
    - jumlah transaksi banyak;
    - transaksi dihentikan menggunakan sentinel.

12. Buat kesimpulan mengenai:
    - penggunaan FOR dan WHILE;
    - fungsi loop bersarang;
    - pentingnya kondisi berhenti;
    - penyebab infinite loop;
    - keuntungan optimasi struktur pengulangan.

**Hasil/Luaran yang Dikumpulkan**

1. Pseudocode algoritma iteratif.
2. Flowchart loop tunggal dan loop bersarang.
3. Tabel tracing setiap iterasi.
4. Tabel rekapitulasi hasil transaksi.
5. Analisis dan perbaikan infinite loop.
6. Perbandingan algoritma sebelum dan sesudah optimasi.
7. Pseudocode menggunakan sentinel.
8. File program Dev C++.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL03 | CPMK032 | Ketepatan menerapkan algoritma iteratif | 15 |
| 2 | CPL03 | CPMK032 | Ketepatan penggunaan loop tunggal dan bersarang | 15 |
| 3 | CPL03 | CPMK032 | Ketepatan penggunaan counter dan accumulator | 10 |
| 4 | CPL03 | CPMK032 | Ketepatan menentukan kondisi berhenti | 15 |
| 5 | CPL03 | CPMK032 | Kemampuan menemukan dan memperbaiki infinite loop | 15 |
| 6 | CPL03 | CPMK032 | Ketepatan tracing perubahan nilai variabel | 10 |
| 7 | CPL03 | CPMK032 | Kemampuan membandingkan efisiensi dua rancangan loop | 10 |
| 8 | CPL03 | CPMK032 | Kemampuan mengoptimalkan struktur pengulangan | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL
07

# Rekursi dan Perbandingan Pendekatan Iteratif

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL03** | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. |
| **CPMK032** | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. |
| **Sub-CPMK** | Mahasiswa mampu menerapkan konsep rekursi dalam pemecahan masalah dan mengimplementasikannya dalam bentuk kode program yang efisien. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menjelaskan mekanisme rekursi melalui base case, recursive case, dan call stack. Mahasiswa mampu mengimplementasikan solusi rekursif, membuat versi iteratif untuk masalah yang sama, melakukan tracing pemanggilan fungsi, serta membandingkan waktu eksekusi, penggunaan memori, dan keterbacaan kedua pendekatan.

## D. Indikator Ketercapaian Pembelajaran

1. Menentukan base case dan recursive case secara tepat.
2. Menjelaskan alur pemanggilan fungsi dan call stack.
3. Mengimplementasikan fungsi rekursif yang berhenti dan menghasilkan keluaran benar.
4. Membuat solusi iteratif untuk masalah yang sama.
5. Melakukan tracing nilai masuk dan nilai kembali pada setiap pemanggilan.
6. Membandingkan trade-off waktu, ruang, keterbacaan, dan risiko stack overflow.

## E. Teori Pendukung

Materi ini menekankan pemahaman konsep sebelum implementasi. Mahasiswa perlu melihat algoritma sebagai model penyelesaian masalah yang dapat dijelaskan, diuji, dan dianalisis. Pemahaman tersebut membantu

membedakan antara sekadar kode yang dapat dijalankan dengan solusi yang benar, terstruktur, dan dapat dipertanggungjawabkan. Konsep utama berikut menjadi dasar untuk kegiatan praktikum dan digunakan kembali pada modul-modul berikutnya.

1. **Rekursi**

   Rekursi adalah teknik ketika fungsi memanggil dirinya sendiri untuk menyelesaikan versi masalah yang lebih kecil. Solusi terbentuk dari hubungan antara kasus kecil dan kasus yang lebih besar. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

2. **Base case**

   Base case adalah kondisi penghentian yang dapat diselesaikan langsung tanpa pemanggilan rekursif berikutnya. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

3. **Recursive case**

   Recursive case mengubah masalah menjadi submasalah yang ukurannya lebih kecil dan memanggil fungsi kembali. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

4. **Call stack**

   Setiap pemanggilan fungsi menyimpan konteks pada stack. Rekursi yang terlalu dalam dapat menggunakan banyak memori atau menyebabkan stack overflow. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

**5. Perbandingan rekursif dan iteratif**

Banyak masalah dapat diselesaikan dengan kedua pendekatan. Rekursi sering lebih dekat dengan definisi matematis, sedangkan iterasi biasanya menggunakan stack lebih sedikit. Dalam praktik, konsep ini perlu dikaitkan dengan data yang diproses, kondisi yang mungkin terjadi, serta cara memeriksa kebenaran hasil. Mahasiswa disarankan menuliskan asumsi dan contoh kecil agar perilaku algoritma dapat ditelusuri secara manual sebelum diuji pada data yang lebih besar.

Hubungan antar konsep tersebut membentuk satu kerangka berpikir. Masalah terlebih dahulu dipahami, kemudian direpresentasikan dalam langkah yang jelas, setelah itu solusi diimplementasikan dan diuji. Jika hasil tidak sesuai, proses kembali ke tahap analisis untuk menemukan bagian logika yang perlu diperbaiki. Siklus ini merupakan kebiasaan penting dalam pengembangan program yang sistematis.

**6. Mekanisme Kerja dan Contoh Penerapan**

Untuk merancang fungsi rekursif, tentukan ukuran masalah dan pastikan setiap recursive case mengurangi ukuran tersebut. Base case harus dapat dicapai. Setelah pemanggilan terdalam selesai, nilai kembali melalui call stack ke pemanggil sebelumnya. Tracing rekursi sebaiknya mencatat parameter setiap pemanggilan, kondisi base case, nilai yang dikembalikan, dan urutan unwinding. Diagram pohon panggilan membantu pada fungsi seperti Fibonacci yang menghasilkan lebih dari satu pemanggilan rekursif.

Dalam praktikum, mekanisme tersebut tidak cukup hanya dibaca. Setiap langkah harus dapat dijelaskan dengan bahasa sendiri dan ditunjukkan melalui data uji. Gunakan contoh berukuran kecil agar perubahan data mudah diamati. Setelah hasil manual konsisten, implementasi dapat diperluas ke ukuran input yang lebih besar. Cara ini memisahkan kesalahan konsep dari kesalahan sintaks dan mempercepat proses debugging.

**Contoh Penerapan**

**Menghitung Faktorial Bilangan**

Faktorial bilangan bulat positif n dinyatakan sebagai:

**n! = n × (n−1) × (n−2) × ... × 1**

Contoh:

**5! = 5 × 4 × 3 × 2 × 1 = 120**

Kasus faktorial cocok digunakan untuk memperlihatkan dengan jelas perbedaan antara pendekatan rekursif dan iteratif.

1. Identifikasi Base Case dan Recursive Case
   Pada algoritma rekursif terdapat dua bagian penting:
   **Base Case** merupakan kondisi yang menghentikan rekursi.
   ***Jika n = 0 atau n = 1***
   ***maka hasil = 1***

   **Recursive Case** merupakan bagian yang memanggil fungsi itu sendiri dengan masalah yang lebih kecil.
   ***Faktorial(n) = n × Faktorial(n - 1)***
   Setiap pemanggilan harus mendekati base case agar rekursi dapat berhenti.
2. Pseudocode Rekursif

```
FUNCTION FaktorialRekursif(n)

    IF n = 0 OR n = 1 THEN
        RETURN 1
    ELSE
        RETURN n × FaktorialRekursif(n - 1)
    ENDIF

END FUNCTION
```

   Program utama:

```
INPUT n

IF n < 0 THEN
    OUTPUT "Input tidak valid"
ELSE
    hasil ← FaktorialRekursif(n)
    OUTPUT hasil
ENDIF
```

3. Tracing Call Stack
   Jika input:
   **n = 5**
   proses pemanggilan fungsi berlangsung sebagai berikut:

```
Faktorial(5)
    5 × Faktorial(4)
        4 × Faktorial(3)
            3 × Faktorial(2)
                2 × Faktorial(1)
                    1
```

Setelah mencapai Faktorial(1), nilai dikembalikan secara bertahap:

```
Faktorial(1) = 1
Faktorial(2) = 2 × 1   = 2
Faktorial(3) = 3 × 2   = 6
Faktorial(4) = 4 × 6   = 24
Faktorial(5) = 5 × 24 = 120
```

Tabel tracing:

| Tahap | Pemanggilan Fungsi | Jenis Proses | Hasil |
|---|---|---|---|
| 1 | Faktorial(5) | Recursive case | 5 × Faktorial(4) |
| 2 | Faktorial(4) | Recursive case | 4 × Faktorial(3) |
| 3 | Faktorial(3) | Recursive case | 3 × Faktorial(2) |
| 4 | Faktorial(2) | Recursive case | 2 × Faktorial(1) |
| 5 | Faktorial(1) | Base case | 1 |
| 6 | Kembali ke Faktorial(2) | Return | 2 |
| 7 | Kembali ke Faktorial(3) | Return | 6 |
| 8 | Kembali ke Faktorial(4) | Return | 24 |
| 9 | Kembali ke Faktorial(5) | Return | 120 |

4. Versi Iteratif

   Masalah yang sama dapat diselesaikan menggunakan perulangan.

```
FUNCTION FaktorialIteratif(n)

    hasil ← 1

    FOR i ← 1 TO n DO
        hasil ← hasil × i
    ENDFOR

    RETURN hasil

END FUNCTION
```

Tracing untuk n = 5:

| Iterasi | i | Hasil Sebelum | Hasil Sesudah |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 2 | 1 | 2 |
| 3 | 3 | 2 | 6 |
| 4 | 4 | 6 | 24 |
| 5 | 5 | 24 | 120 |

5. Perbandingan Rekursif dan Iteratif

| Aspek | Rekursif | Iteratif |
|---|---|---|
| Mekanisme | Fungsi memanggil dirinya sendiri | Menggunakan loop |
| Kondisi berhenti | Base case | Kondisi perulangan |
| Waktu | O(n) | O(n) |
| Ruang tambahan | O(n) karena call stack | O(1) |
| Keterbacaan | Ringkas untuk masalah yang alami bersifat rekursif | Sederhana untuk masalah berulang |
| Risiko | Stack overflow | Infinite loop |
| Faktorial | Mudah ditulis | Umumnya lebih hemat memori |

Walaupun kompleksitas waktunya sama-sama **O(n)**, versi rekursif memerlukan ruang tambahan untuk menyimpan setiap pemanggilan fungsi pada call stack.

6. Contoh Kesalahan Rekursi

```
FUNCTION Faktorial(n)

    RETURN n × Faktorial(n - 1)

END FUNCTION
```

Algoritma tersebut tidak memiliki **base case**, sehingga pemanggilan fungsi akan terus berlangsung:

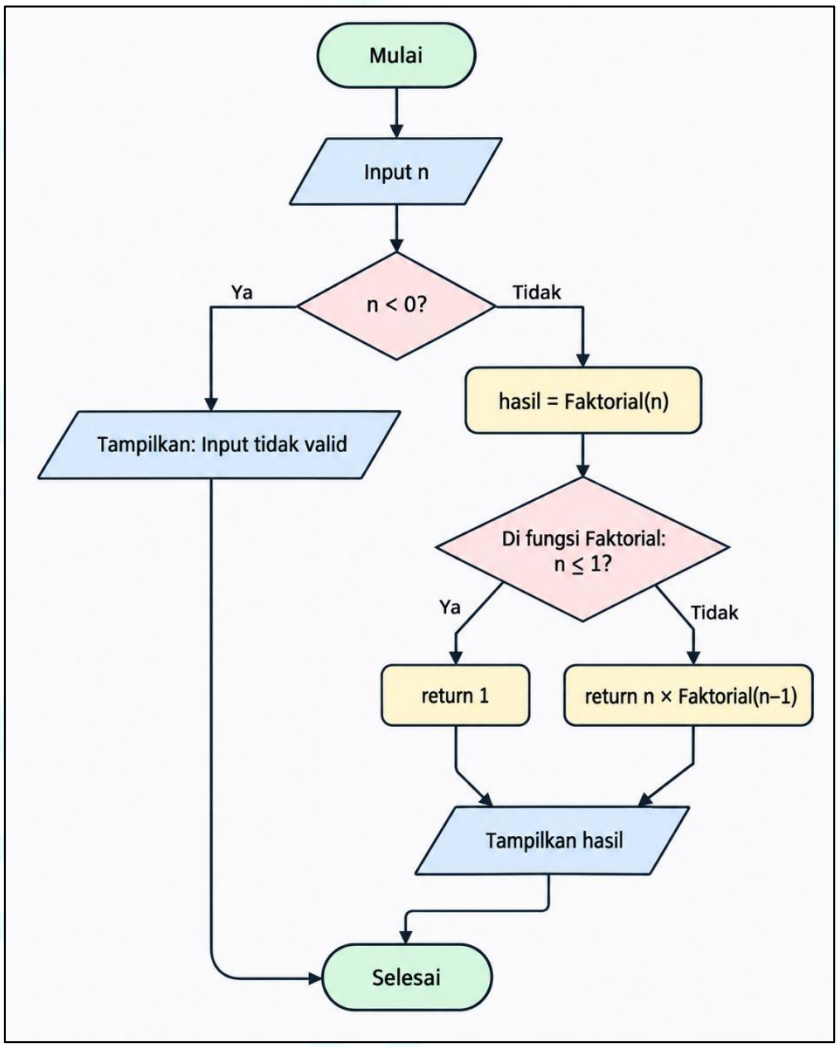
Faktorial(5)
Faktorial(4)
Faktorial(3)
Faktorial(2)
Faktorial(1)
Faktorial(0)
Faktorial(-1)
...

Hal ini dapat menyebabkan **stack overflow**.

Perbaikannya:

```
IF n <= 1 THEN
    RETURN 1
ENDIF
```

7. Flowchart




Gambar 9. Flowchart Faktorial Rekursif

8. Contoh Implementasi Dev C++

```cpp
#include <iostream>
using namespace std;

long long faktorialRekursif(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }

    return n * faktorialRekursif(n - 1);
}

long long faktorialIteratif(int n) {
    long long hasil = 1;

    for (int i = 1; i <= n; i++) {
        hasil *= i;
    }

    return hasil;
}

int main() {
    int n;

    cout << "Masukkan bilangan: ";
    cin >> n;

    if (n < 0) {
        cout << "Input tidak valid";
    } else {
        cout << "Rekursif : "
             << faktorialRekursif(n) << endl;

        cout << "Iteratif : "
             << faktorialIteratif(n) << endl;
    }

    return 0;
}
```

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

**2. Prosedur Umum**

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

**3. Daftar Kegiatan Praktikum**

a) Mengidentifikasi base case dan recursive case pada algoritma rekursif sederhana.
b) Mengimplementasikan fungsi rekursif dalam Dev C++.
c) Membuat versi iteratif untuk masalah yang sama.
d) Melakukan tracing pemanggilan fungsi dan nilai yang dikembalikan.
e) Membandingkan waktu eksekusi, penggunaan memori, dan keterbacaan kedua pendekatan.

**4. Studi Kasus**

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 1: Faktorial | 30 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 2: Fibonacci | 40 |
| CPL03 | CPMK032 | Selesaikan langkah praktikum – Studi Kasus 3: FPB Euclid | 30 |
| | | **Total** | **100** |

**1) Studi Kasus 1 - Faktorial**

Implementasikan faktorial rekursif dan iteratif. Trace call stack untuk n=5, ukur waktu untuk beberapa n, dan bandingkan penggunaan memori.

**2) Studi Kasus 2 - Fibonacci**

Buat Fibonacci rekursif naif dan iteratif. Hitung jumlah pemanggilan atau waktu pada n meningkat dan jelaskan mengapa performanya berbeda.

**3) Studi Kasus 3 - FPB Euclid**

Implementasikan algoritma Euclid secara rekursif dan iteratif untuk mencari FPB dua bilangan. Trace perubahan parameter sampai base case.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 7

Buat program yang menghitung faktorial n dan suku Fibonacci ke-n dengan dua pendekatan: rekursif dan iteratif. Program harus memvalidasi input n >= 0, menampilkan hasil, dan menyediakan data yang diperlukan untuk membandingkan jumlah pemanggilan/iterasi serta waktu eksekusi.

### Soal Evaluasi

1. Identifikasi base case dan recursive case untuk fungsi faktorial dan Fibonacci.
2. Susun pseudocode versi rekursif dan iteratif untuk kedua masalah.
3. Lakukan tracing call stack Faktorial(5) sampai nilai 120 dikembalikan.
4. Uji n = 0, 1, 5, 10, dan 30; catat hasil serta waktu eksekusi kedua pendekatan.
5. Analisis kompleksitas waktu dan ruang faktorial rekursif vs iteratif, serta Fibonacci rekursif sederhana vs iteratif.
6. Perbaiki contoh fungsi rekursif yang tidak memiliki base case atau menggunakan pemanggilan n+1 sehingga tidak mendekati kondisi berhenti.
7. Jelaskan kapan pendekatan rekursif lebih mudah dibaca dan kapan pendekatan iteratif lebih efisien.

### Hasil/Luaran yang Dikumpulkan

1. Pseudocode rekursif dan iteratif.
2. Kode program Dev C++.
3. Tabel tracing call stack Faktorial(5).
4. Tabel hasil pengujian dan waktu eksekusi.
5. Analisis kompleksitas waktu/ruang.
6. Kesimpulan trade-off rekursif dan iteratif.

### Aturan Penilaian (Total Skor : 100)

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL03 | CPMK032 | Ketepatan menentukan base case dan recursive case | 15 |
| 2 | CPL03 | CPMK032 | Kebenaran pseudocode rekursif dan iteratif | 20 |
| 3 | CPL03 | CPMK032 | Ketepatan tracing call stack dan nilai kembali | 20 |
| 4 | CPL03 | CPMK032 | Kebenaran implementasi program | 15 |
| 5 | CPL03 | CPMK032 | Ketepatan analisis waktu dan ruang | 15 |
| 6 | CPL03 | CPMK032 | Kemampuan menemukan kesalahan rekursi dan memperbaikinya | 10 |

| | | | | |
|---|---|---|---|---|
| 7 | CPL03 | CPMK032 | Kesimpulan trade-off rekursif dan iteratif | 5 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 08

# Linear Search dan Penanganan Kasus Tepi

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| **CPL08** | Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| **CPMK084** | Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| **Sub-CPMK** | Mahasiswa mampu mengimplementasikan linear search serta membuktikan kompleksitasnya melalui analisis dan eksperimen, termasuk penanganan kasus tepi. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menerapkan linear search pada data terurut maupun tidak terurut, melakukan tracing proses pencarian, menangani kondisi data kosong, duplikasi, dan target tidak ditemukan, serta menganalisis best, average, dan worst case. Mahasiswa juga mampu mengukur jumlah perbandingan dan waktu eksekusi untuk menunjukkan karakteristik kompleksitas O(n).

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 8 apabila mampu:

1. Menjelaskan prinsip kerja linear search dan kondisi penghentiannya.
2. Mengimplementasikan linear search untuk menemukan satu atau seluruh kecocokan.
3. Menangani data kosong, target tidak ditemukan, dan data duplikat secara benar.
4. Melakukan tracing posisi indeks, nilai elemen, dan jumlah perbandingan.
5. Menganalisis best, average, dan worst case linear search.
6. Melakukan eksperimen waktu eksekusi dan mengaitkannya dengan kompleksitas O(n).

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

### 1. Prinsip linear search

Linear search memeriksa elemen satu per satu mulai dari indeks awal sampai target ditemukan atau seluruh data selesai diperiksa. Algoritma ini tidak mensyaratkan data sudah terurut sehingga cocok untuk dataset sederhana, data yang sering berubah, atau pencarian satu kali. Keunggulannya adalah implementasi sangat langsung, sedangkan kelemahannya adalah jumlah pemeriksaan dapat tumbuh sebanding dengan ukuran data.

### 2. Kondisi berhenti

Pencarian dapat berhenti ketika target ditemukan jika hanya satu kecocokan yang diperlukan. Jika tujuan mencari seluruh kemunculan, proses harus dilanjutkan sampai elemen terakhir. Kondisi berhenti harus dirancang jelas agar tidak terjadi akses indeks di luar batas array.

### 3. Best, average, dan worst case

Best case terjadi ketika target berada pada elemen pertama sehingga hanya satu perbandingan diperlukan, yaitu O(1). Pada kondisi rata-rata, target berada di sekitar bagian tengah dan jumlah perbandingan mendekati n/2, tetapi tetap disederhanakan menjadi O(n). Worst case terjadi ketika target berada di elemen terakhir atau tidak ditemukan sehingga seluruh n elemen diperiksa.

### 4. Kasus tepi

Kasus tepi yang harus diuji mencakup array kosong, satu elemen, target di awal, target di akhir, target tidak ditemukan, dan data duplikat. Penanganan kasus tepi penting karena program dapat tampak benar pada data normal tetapi gagal ketika ukuran data sangat kecil atau kondisi tertentu tidak terpenuhi.

### 5. Counter perbandingan dan tracing

Counter perbandingan digunakan untuk membuktikan biaya pencarian secara empiris. Tabel tracing dapat memuat indeks, nilai yang diperiksa, hasil perbandingan, status ditemukan, dan total perbandingan. Tracing membantu membedakan kesalahan logika dengan kesalahan sintaks.

**6. Kompleksitas dan eksperimen**

Linear search memiliki kompleksitas waktu O(n) pada average dan worst case serta ruang tambahan O(1). Untuk menguji pertumbuhan ini, program dapat dijalankan pada n yang meningkat dan waktu eksekusi dicatat secara konsisten. Waktu nyata dapat berfluktuasi, sehingga fokus analisis adalah kecenderungan pertumbuhannya.

**7. Mekanisme Kerja dan Contoh Penerapan**

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

**Contoh Penerapan - Pencarian NIM Mahasiswa**

Diberikan daftar NIM yang belum terurut. Program menerima satu NIM target dan menelusuri data dari awal. Selain posisi target, program menghitung jumlah perbandingan sehingga mahasiswa dapat mengamati perbedaan ketika target berada di awal, tengah, akhir, atau tidak ditemukan.

```
FUNCTION LinearSearch(data, n, target)
    perbandingan <- 0
    FOR i <- 0 TO n-1 DO
        perbandingan <- perbandingan + 1
        IF data[i] = target THEN
            RETURN i, perbandingan
        ENDIF
    ENDFOR
    RETURN -1, perbandingan
END FUNCTION
```

| Kasus | Posisi Target | Perbandingan | Keterangan |
|---|---|---|---|
| A | Indeks 0 | 1 | Best case |
| B | Indeks tengah | sekitar n/2 | Average case |
| C | Indeks n-1 | n | Worst case |
| D | Tidak ada | n | Worst case |

**8. Contoh Implementasi Kode Dev++**

Program menyimpan beberapa NIM dan nama mahasiswa. Pengguna memasukkan NIM yang ingin dicari, lalu program melakukan pencarian satu per satu menggunakan Linear Search.

```cpp
#include <iostream>
using namespace std;

int main() {
    string nim[5] = {
        "13020230010",
        "13020230040",
        "13020230043",
        "13020230070",
        "13020230073"
    };

    string nama[5] = {
        "Nur Azani Labadja",
        "Nova Febryna",
        "Sahra Zhafirah",
        "Artika Sari Murti",
        "Satriani"
    };

    string cariNIM;
    int posisi = -1;

    cout << "Masukkan NIM yang dicari: ";
    cin >> cariNIM;

    for (int i = 0; i < 5; i++) {
        if (nim[i] == cariNIM) {
            posisi = i;
            break;
        }
    }

    if (posisi != -1) {
        cout << "\nData ditemukan" << endl;
        cout << "NIM  : " << nim[posisi] << endl;
        cout << "Nama : " << nama[posisi] << endl;
    } else {
        cout << "\nData mahasiswa tidak ditemukan" << endl;
    }

    return 0;
}
```

**Output:**

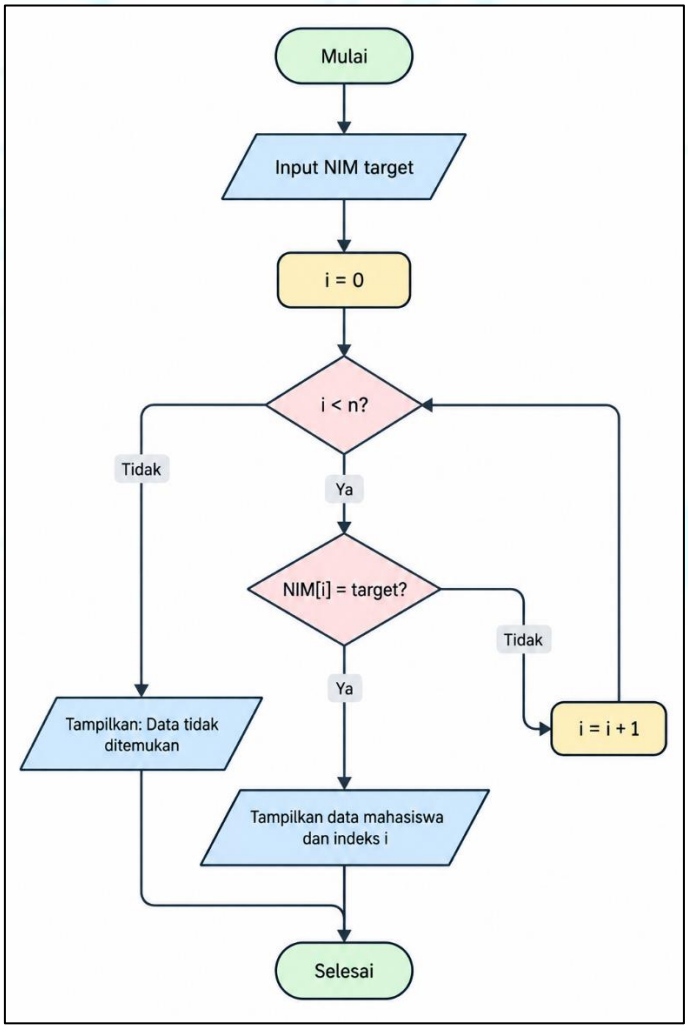
```
D:\1. LABORATORIUM\MODU
Masukkan NIM yang dicari: 13020230043

Data ditemukan
NIM  : 13020230043
Nama : Sahra Zhafirah

--------------------------------
Process exited after 1.438 seconds with return value 0
Press any key to continue . . .
```

## Contoh Tracing

Jika NIM yang dicari adalah 13020230043:

| Iterasi | NIM yang diperiksa | Hasil |
|---|---|---|
| 1 | 13020230010 | Tidak sama |
| 2 | 13020230040 | Tidak sama |
| 3 | 13020230043 | **Ditemukan** |

## Flowchart




Gambar 10. Flowchart Pencarian NIM Mahasiswa

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Mengimplementasikan linear search pada data berukuran berbeda.
b) Menguji target di awal, tengah, akhir, tidak ditemukan, dan data kosong.
c) Memodifikasi algoritma untuk menampilkan semua posisi pada data duplikat.
d) Mencatat jumlah perbandingan setiap test case.
e) Melakukan eksperimen beberapa ukuran n dan menganalisis O(n).

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Pencarian NIM | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Pencarian Kode Barang | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Pencarian Nama Pengunjung | 40 |
| | | **Total** | **100** |

#### 1) Studi Kasus 1 - Pencarian NIM

Cari NIM tertentu pada daftar mahasiswa tidak terurut. Uji target di awal, tengah, akhir, tidak ditemukan, dan daftar kosong; catat jumlah perbandingan.

#### 2) Studi Kasus 2 - Pencarian Kode Barang

Cari kode barang pada daftar inventori. Jika terdapat duplikasi, tampilkan semua posisi dan bandingkan dengan varian yang berhenti pada kecocokan pertama.

**3) Studi Kasus 3 - Pencarian Nama Pengunjung**

Cari nama pada daftar pengunjung. Tentukan aturan case-sensitive atau case-insensitive, uji variasi input, dan jelaskan dampaknya pada hasil pencarian.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 8

### Pencarian Data Mahasiswa dengan Linear Search

Sebuah program menyimpan daftar NIM mahasiswa yang belum terurut. Program harus mencari NIM target, menampilkan posisi kemunculan, menghitung jumlah perbandingan, serta menangani data kosong, target tidak ditemukan, dan NIM yang muncul lebih dari satu kali.

### Soal Evaluasi

1. Identifikasi input, proses, output, kondisi berhenti, dan kasus tepi yang harus ditangani.
2. Susun pseudocode linear search untuk pencarian kecocokan pertama dan versi yang mengembalikan semua posisi target.
3. Lakukan tracing pada data [1301, 1307, 1312, 1307, 1320, 1331] untuk target 1307, 1331, dan 1400.
4. Catat jumlah perbandingan untuk target di awal, tengah, akhir, serta tidak ditemukan.
5. Implementasikan program menggunakan Dev C++ dan uji array kosong, satu elemen, data duplikat, dan target tidak ditemukan.
6. Lakukan eksperimen untuk n = 1.000, 5.000, 10.000, dan 50.000; catat waktu eksekusi dan jelaskan kecenderungan O(n).
7. Jelaskan perbedaan best, average, dan worst case serta kapan linear search masih layak digunakan.

| Test Case | Data/Target | Hasil Diharapkan | Perbandingan |
|---|---|---|---|
| 1 | Target pada elemen pertama | Ditemukan indeks 0 | ... |
| 2 | Target pada elemen tengah | Ditemukan | ... |
| 3 | Target pada elemen terakhir | Ditemukan | ... |
| 4 | Target tidak ada | Tidak ditemukan | ... |
| 5 | Data kosong | Tidak ditemukan tanpa error | 0 |

### Hasil/Luaran yang Dikumpulkan

1. Pseudocode linear search satu kecocokan dan semua kecocokan.
2. Kode program Dev C++.
3. Tabel tracing dan jumlah perbandingan.
4. Tabel hasil eksperimen waktu eksekusi.
5. Analisis best/average/worst case dan kompleksitas O(n).

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL08 | CPMK084 | Ketepatan implementasi linear search | 20 |
| 2 | CPL08 | CPMK084 | Ketepatan menangani kasus tepi | 15 |
| 3 | CPL08 | CPMK084 | Ketepatan tracing dan jumlah perbandingan | 15 |
| 4 | CPL08 | CPMK084 | Analisis best, average, dan worst case | 15 |
| 5 | CPL08 | CPMK084 | Ketepatan eksperimen waktu eksekusi | 15 |
| 6 | CPL08 | CPMK084 | Analisis kompleksitas O(n) | 10 |
| 7 | CPL08 | CPMK084 | Kebenaran hasil dan kesimpulan | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 09

# Binary Search Iteratif dan Rekursif

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| CPL08 | Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| CPMK084 | Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| Sub-CPMK | Mahasiswa mampu mengimplementasikan binary search iteratif dan rekursif, menganalisis kompleksitas O(log n), serta membandingkan performanya dengan linear search. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menjelaskan prasyarat data terurut, menerapkan binary search dengan pendekatan iteratif dan rekursif, melakukan tracing perubahan batas low, high, dan mid, serta membandingkan jumlah perbandingan dan waktu eksekusi dengan linear search. Mahasiswa juga mampu mengenali kondisi ketika binary search tidak valid karena data belum terurut.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 9 apabila mampu:

1. Menjelaskan mengapa binary search membutuhkan data terurut.
2. Mengimplementasikan binary search iteratif dengan batas low-high yang benar.
3. Mengimplementasikan binary search rekursif dengan base case yang tepat.
4. Melakukan tracing perubahan low, high, mid, dan nilai tengah.
5. Menganalisis kompleksitas waktu O(log n) dan ruang tambahan kedua versi.
6. Membandingkan performa binary search dengan linear search pada ukuran data berbeda.

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep

perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

1. **Prasyarat data terurut**

   Binary search hanya benar apabila data telah diurutkan berdasarkan kunci yang sama dengan yang digunakan untuk pencarian. Jika data tidak terurut, keputusan membuang setengah ruang pencarian tidak dapat dijamin benar. Karena itu program sebaiknya mendokumentasikan prasyarat ini atau melakukan proses sorting terlebih dahulu.

2. **Pembagian ruang pencarian**

   Setiap langkah menghitung indeks tengah mid = low + (high-low)/2. Jika target sama dengan elemen tengah, pencarian selesai. Jika target lebih kecil, high dipindahkan ke mid-1; jika lebih besar, low dipindahkan ke mid+1. Ukuran ruang pencarian berkurang kira-kira separuh setiap iterasi.

3. **Binary search iteratif**

   Versi iteratif menggunakan while selama low <= high. Pendekatan ini menggunakan ruang tambahan O(1) karena hanya memerlukan sejumlah variabel indeks. Kesalahan umum adalah memperbarui low atau high ke mid tanpa +1/-1 sehingga loop dapat tidak maju.

4. **Binary search rekursif**

   Versi rekursif menggunakan parameter low dan high. Base case terjadi ketika low > high, yang berarti target tidak ditemukan. Recursive case memilih subarray kiri atau kanan. Setiap pemanggilan mengurangi ukuran masalah sehingga base case harus selalu dapat dicapai.

5. **Kompleksitas O(log n)**

   Karena ruang pencarian dibagi dua pada setiap langkah, jumlah langkah maksimum sekitar log2(n). Untuk satu juta elemen, binary search hanya membutuhkan puluhan perbandingan, jauh lebih sedikit daripada linear search worst case yang dapat memeriksa seluruh elemen.

6. **Trade-off iteratif dan rekursif**

   Kedua versi memiliki kompleksitas waktu O(log n). Versi iteratif menggunakan ruang O(1), sedangkan versi rekursif membutuhkan call stack O(log n). Versi rekursif dapat lebih dekat dengan definisi algoritma, tetapi versi iteratif umumnya lebih hemat memori.

### 7. Mekanisme Kerja dan Contoh Penerapan

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

**Contoh Penerapan - Pencarian Nilai pada Data Terurut**

Gunakan data [10, 20, 30, 40, 50, 60, 70, 80]. Untuk target 70, ruang pencarian dipersempit melalui elemen tengah sampai target ditemukan.

```
FUNCTION BinarySearchIteratif(data, n, target)
    low <- 0
    high <- n-1
    WHILE low <= high DO
        mid <- low + (high-low)/2
        IF data[mid] = target THEN RETURN mid
        ELSE IF target < data[mid] THEN high <- mid-1
        ELSE low <- mid+1
        ENDIF
    ENDWHILE
    RETURN -1
END FUNCTION
```

| Langkah | low | high | mid | data[mid] | Keputusan |
|---|---|---|---|---|---|
| 1 | 0 | 7 | 3 | 40 | target > 40 |
| 2 | 4 | 7 | 5 | 60 | target > 60 |
| 3 | 6 | 7 | 6 | 70 | ditemukan |

## 8. Contoh Implementasi Kode Dev++

```cpp
#include <iostream>
using namespace std;

int main() {
    int data[5] = {10, 20, 30, 40, 50};
    int target;
    int posisi = -1;
    int perbandingan = 0;

    cout << "Masukkan angka yang dicari: ";
    cin >> target;

    for (int i = 0; i < 5; i++) {
        perbandingan++;

        if (data[i] == target) {
            posisi = i;
            break;
        }
    }

    if (posisi != -1) {
        cout << "Data ditemukan pada indeks " << posisi << endl;
    } else {
        cout << "Data tidak ditemukan" << endl;
    }

    cout << "Jumlah perbandingan = "
         << perbandingan << endl;

    return 0;
}
```

Jalankan, misalnya input: 40

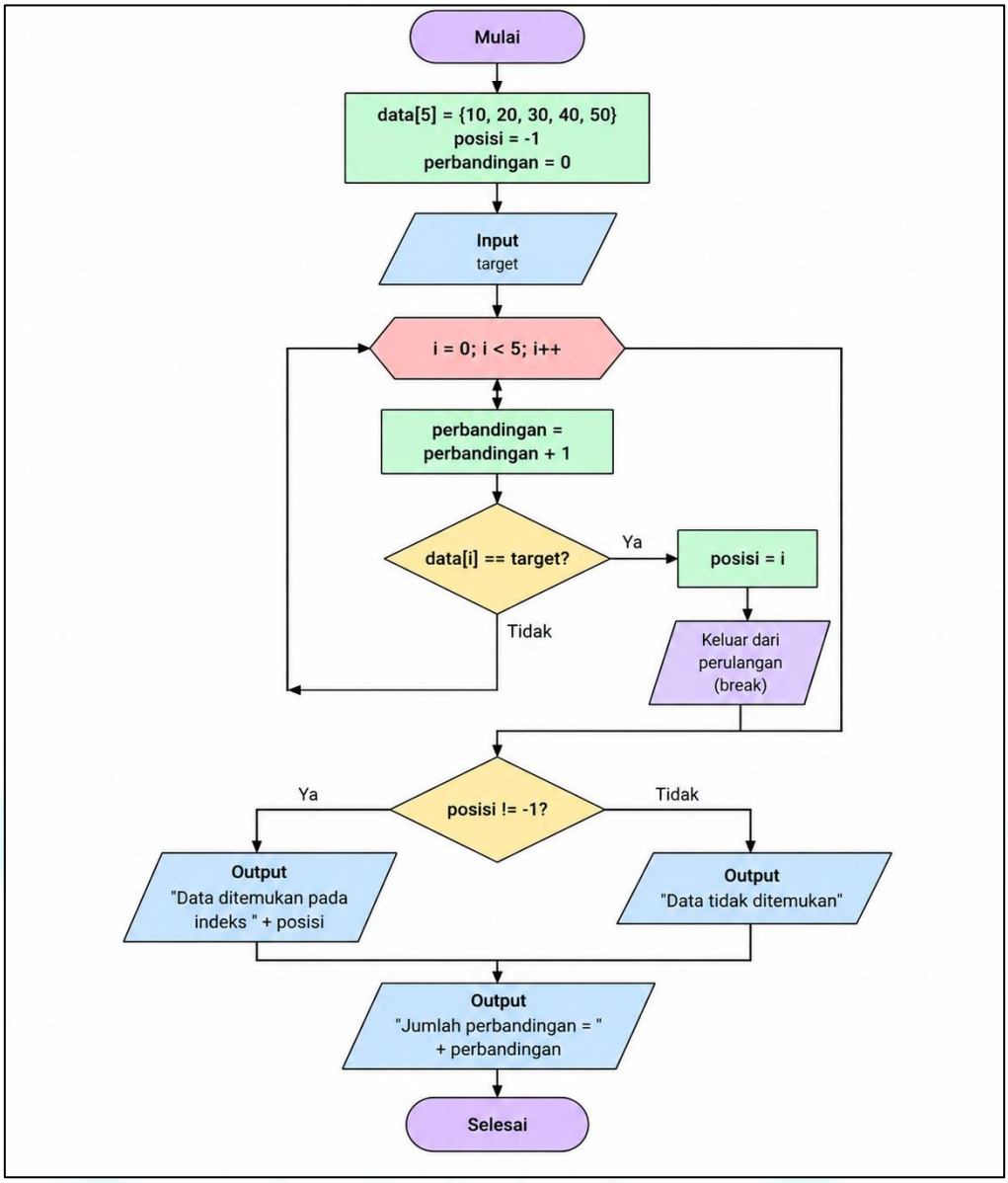
**Prosesnya:**

| Langkah | Low | High | Mid | Data Mid | Hasil |
|---|---|---|---|---|---|
| 1 | 0 | 4 | 2 | 30 | 40 > 30, cari ke kanan |
| 2 | 3 | 4 | 3 | 40 | Ditemukan |

**Output:**

```
D:\1. LABORATORIUM\MODU
Masukkan angka yang dicari: 40
Data ditemukan pada indeks 3

--------------------------------
Process exited after 14.34 seconds with return value 0
Press any key to continue . . .
```

**Flowchart**




Gambar 11. Flowchart Pencarian Nilai pada Data Terurut

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.

d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

**3. Daftar Kegiatan Praktikum**

a) Menyiapkan data terurut sebagai masukan binary search.
b) Mengimplementasikan versi iteratif dan rekursif.
c) Menguji target pada posisi berbeda dan data tidak ditemukan.
d) Melakukan tracing low, high, mid, serta jumlah perbandingan.
e) Membandingkan linear dan binary search untuk n yang meningkat.

**4. Studi Kasus**

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Binary Search Data Nilai | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Iteratif vs Rekursif | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Linear vs Binary Search | 40 |
| | | **Total** | **100** |

**1) Studi Kasus 1 - Binary Search Data Nilai**

Cari nilai tertentu pada daftar terurut. Trace low-high-mid dan uji target di awal, tengah, akhir, serta tidak ditemukan.

**2) Studi Kasus 2 - Iteratif vs Rekursif**

Implementasikan kedua versi pada dataset yang sama. Bandingkan jumlah perbandingan, waktu, dan penggunaan ruang.

**3) Studi Kasus 3 - Linear vs Binary Search**

Bandingkan linear dan binary search pada data 1.000 sampai 100.000 elemen. Jelaskan pengaruh prasyarat sorting dan perbedaan kompleksitas.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 9

### Pencarian Kode Barang Terurut

Sebuah sistem inventori menyimpan kode barang terurut menaik. Mahasiswa diminta membuat binary search iteratif dan rekursif untuk menemukan kode, kemudian membandingkannya dengan linear search.

### Soal Evaluasi

1. Jelaskan prasyarat yang harus dipenuhi sebelum binary search dijalankan.
2. Susun pseudocode binary search iteratif dan rekursif dengan keluaran indeks atau -1.
3. Lakukan tracing low, high, mid pada data [101,105,110,115,120,125,130,135,140] untuk target 125 dan 133.
4. Implementasikan linear search dan kedua versi binary search menggunakan Dev C++.
5. Uji target pada elemen pertama, tengah, terakhir, tidak ditemukan, dan data satu elemen.
6. Bandingkan jumlah perbandingan dan waktu eksekusi untuk n = 1.000, 10.000, 100.000.
7. Analisis kompleksitas waktu dan ruang ketiga implementasi serta jelaskan kapan binary search lebih unggul.

| Test Case | Target | Iteratif | Rekursif | Perbandingan |
|---|---|---|---|---|
| 1 | 101 | ... | ... | ... |
| 2 | 120 | ... | ... | ... |
| 3 | 140 | ... | ... | ... |
| 4 | 133 | -1 | -1 | ... |
| 5 | data 1 elemen | ... | ... | ... |

### Hasil/Luaran yang Dikumpulkan

1. Pseudocode binary search iteratif dan rekursif.
2. Kode linear search dan binary search.
3. Tabel tracing low-high-mid.
4. Tabel perbandingan jumlah perbandingan dan waktu.
5. Analisis kompleksitas O(log n) dan trade-off ruang.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL08 | CPMK084 | Pemahaman prasyarat data terurut | 10 |
| 2 | CPL08 | CPMK084 | Kebenaran binary search iteratif | 20 |
| 3 | CPL08 | CPMK084 | Kebenaran binary search rekursif | 20 |
| 4 | CPL08 | CPMK084 | Ketepatan tracing low-high-mid | 15 |
| 5 | CPL08 | CPMK084 | Analisis kompleksitas waktu dan ruang | 15 |
| 6 | CPL08 | CPMK084 | Perbandingan dengan linear search | 10 |
| 7 | CPL08 | CPMK084 | Penanganan kasus tepi dan hasil akhir | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 10

# Bubble Sort dan Selection Sort

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| CPL08 | Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| CPMK084 | Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| Sub-CPMK | Mahasiswa mampu mengimplementasikan bubble sort dan selection sort serta membandingkan efisiensi kedua algoritma pengurutan sederhana. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu memahami konsep pengurutan, mengimplementasikan bubble sort dan selection sort, melakukan tracing setiap pass, menghitung jumlah perbandingan dan pertukaran, serta menganalisis kompleksitas $O(n^2)$. Mahasiswa juga mampu menerapkan optimasi early-stop pada bubble sort dan menjelaskan perbedaan karakteristik kedua algoritma.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 10 apabila mampu:

1. Mengimplementasikan bubble sort secara benar untuk urutan menaik/menurun.
2. Mengimplementasikan selection sort pada data yang sama.
3. Melakukan tracing isi array pada setiap pass.
4. Menghitung jumlah perbandingan dan pertukaran secara konsisten.
5. Menganalisis kompleksitas $O(n^2)$ dan best case bubble sort teroptimasi.
6. Membandingkan karakteristik bubble sort dan selection sort pada beberapa pola data.

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

### 1. Konsep sorting

Sorting menyusun data berdasarkan kunci tertentu, misalnya nilai, nama, atau kode. Data terurut mempermudah pencarian, penyajian ranking, dan analisis. Pengurutan dapat bersifat ascending atau descending dan harus konsisten terhadap kunci yang dipilih.

Contoh data:

80, 60, 90, 70, 75

**Ascending: 60, 70, 75, 80, 90**

**Descending: 90, 80, 75, 70, 60**

Contoh perintah menampilkan isi array:

```
int nilai[5] = {80, 60, 90, 70, 75};

for (int i = 0; i < 5; i++) {
    cout << nilai[i] << " ";
}
```

Perintah utama yang digunakan dalam proses sorting adalah **perbandingan**:

```
if (nilai[i] > nilai[j]) {
    // lakukan pertukaran
}
```

### 2. Bubble sort

Bubble sort membandingkan elemen berdekatan dan menukar jika urutannya salah. Setelah satu pass, satu elemen ekstrem berada pada posisi akhir. Proses diulang pada bagian yang belum terurut.

Contoh perintah perbandingan:

```
if (nilai[j] > nilai[j + 1]) {
    // tukar
}
```

Contoh perintah pertukaran:

```
int temp = nilai[j];
nilai[j] = nilai[j + 1];
nilai[j + 1] = temp;
```

**3. Selection sort**

Selection sort memilih elemen minimum atau maksimum dari bagian yang belum terurut, kemudian menukarnya ke posisi yang tepat. Algoritma melakukan banyak perbandingan tetapi relatif sedikit pertukaran.

Contoh menentukan indeks minimum:

```
int indeksMin = i;
```

Mencari nilai yang lebih kecil:

```
if (nilai[j] < nilai[indeksMin]) {
    indeksMin = j;
}
```

Menukar nilai terkecil dengan posisi awal:

```
int temp = nilai[i];
nilai[i] = nilai[indeksMin];
nilai[indeksMin] = temp;
```

**4. Tracing pass dan swap**

Tracing sorting sebaiknya mencatat keadaan array setelah setiap pass, indeks yang dibandingkan, dan apakah terjadi swap. Ini membantu menemukan kesalahan batas loop serta memastikan arah pengurutan benar.

Contoh menampilkan kondisi array pada setiap pass:

```
cout << "Pass ke-" << i + 1 << ": ";

for (int k = 0; k < 5; k++) {
    cout << nilai[k] << " ";
}

cout << endl;
```

**5. Kompleksitas**

Bubble sort dan selection sort secara umum memiliki kompleksitas $O(n^2)$. Selection sort tetap melakukan sekitar $n(n-1)/2$ perbandingan pada data sudah terurut. Bubble sort dapat dioptimalkan dengan flag swapped sehingga berhenti lebih awal jika satu pass tidak menghasilkan pertukaran.

**6. Karakteristik data**

Pengujian sebaiknya mencakup data acak, sudah terurut, terbalik, duplikat, satu elemen, dan data kosong. Pola data memengaruhi jumlah swap dan waktu praktis meskipun kelas kompleksitas teoretis dapat sama.

**7. Mekanisme Kerja dan Contoh Penerapan**

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

**Contoh Penerapan - Mengurutkan Nilai Ujian**

Urutkan data [5, 1, 4, 2] secara menaik. Bubble sort memindahkan elemen besar secara bertahap melalui pertukaran berdekatan, sedangkan selection sort memilih nilai minimum untuk ditempatkan pada posisi berikutnya.

```
BUBBLE SORT
FOR i <- 0 TO n-2 DO
    swapped <- FALSE
    FOR j <- 0 TO n-2-i DO
        IF data[j] > data[j+1] THEN
            SWAP data[j], data[j+1]
            swapped <- TRUE
        ENDIF
    ENDFOR
    IF swapped = FALSE THEN BREAK
ENDFOR
```

| Pass | Bubble Sort | Selection Sort |
|---|---|---|
| Awal | [5,1,4,2] | [5,1,4,2] |
| 1 | [1,4,2,5] | [1,5,4,2] |
| 2 | [1,2,4,5] | [1,2,4,5] |
| Akhir | [1,2,4,5] | [1,2,4,5] |

**8. Implementasi Kode Dev C++**

Mengurutkan Nilai Mahasiswa

Data awal:

75, 60, 90, 70, 80

Target: urutkan dari nilai terkecil ke terbesar.

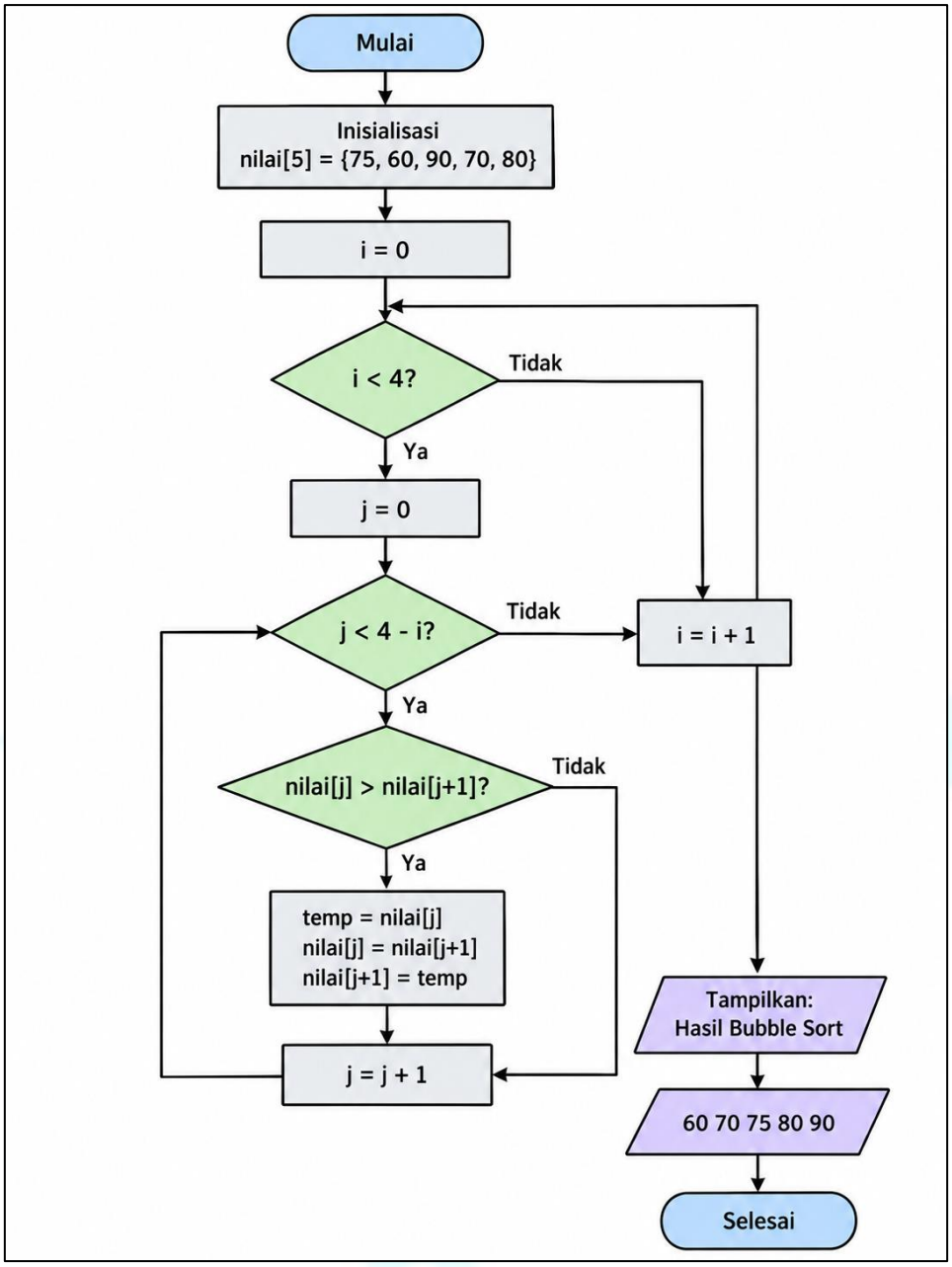
a. Bubble Sort

```cpp
#include <iostream>
using namespace std;

int main() {
    int nilai[5] = {75, 60, 90, 70, 80};

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4 - i; j++) {

            if (nilai[j] > nilai[j + 1]) {
                int temp = nilai[j];
                nilai[j] = nilai[j + 1];
                nilai[j + 1] = temp;
            }
        }
    }

    cout << "Hasil Bubble Sort: ";

    for (int i = 0; i < 5; i++) {
        cout << nilai[i] << " ";
    }

    return 0;
}
```

**Output:**

```
D:\1. LABORATORIUM\MODU
Hasil Bubble Sort: 60 70 75 80 90
--------------------------------
Process exited after 0.2389 seconds with return value 0
Press any key to continue . . .
```

Bubble sort membandingkan elemen yang bersebelahan dan menukarnya jika urutannya salah. Setelah setiap pass, elemen terbesar akan bergerak ke bagian akhir array.




Gambar 12. Flowchart Mengurutkan Nilai Mahasiswa

b. Selection Sort

```cpp
#include <iostream>
using namespace std;

int main() {
    int nilai[5] = {75, 60, 90, 70, 80};

    for (int i = 0; i < 4; i++) {

        int indeksMin = i;

        for (int j = i + 1; j < 5; j++) {

            if (nilai[j] < nilai[indeksMin]) {
                indeksMin = j;
            }
        }

        int temp = nilai[i];
        nilai[i] = nilai[indeksMin];
        nilai[indeksMin] = temp;
    }

    cout << "Hasil Selection Sort: ";

    for (int i = 0; i < 5; i++) {
        cout << nilai[i] << " ";
    }

    return 0;
}
```

**Output:**

```
D:\1. LABORATORIUM\MODU
Hasil Selection Sort: 60 70 75 80 90
--------------------------------
Process exited after 0.1134 seconds with return value 0
Press any key to continue . . .
```

Selection sort bekerja dengan mencari nilai terkecil dari bagian array yang belum terurut, lalu menempatkannya pada posisi yang sesuai.

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Mengimplementasikan bubble sort ascending dan descending.
b) Mengimplementasikan selection sort pada data yang sama.
c) Melakukan tracing setiap pass dan pertukaran.
d) Menghitung perbandingan serta swap untuk berbagai pola data.
e) Menguji optimasi early-stop dan membandingkan waktu eksekusi.

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Sorting Nilai Mahasiswa | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Data Sudah Terurut dan Terbalik | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Ranking Produk | 40 |
| | | **Total** | **100** |

#### 1) Studi Kasus 1 - Sorting Nilai Mahasiswa

Urutkan nilai mahasiswa menggunakan bubble sort dan selection sort. Catat isi array setiap pass, perbandingan, dan swap.

#### 2) Studi Kasus 2 - Data Sudah Terurut dan Terbalik

Bandingkan kedua algoritma pada data terurut dan terbalik. Aktifkan early-stop pada bubble sort dan analisis perubahan biaya.

#### 3) Studi Kasus 3 - Ranking Produk

Urutkan data penjualan produk dari terbesar ke terkecil. Bandingkan jumlah operasi dan simpulkan algoritma yang lebih sesuai untuk data kecil.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 10

### Perbandingan Bubble Sort dan Selection Sort

Diberikan data nilai [78, 65, 90, 72, 65, 88, 60]. Program harus mengurutkan data menaik menggunakan bubble sort dan selection sort serta mencatat proses internal kedua algoritma.

### Soal Evaluasi

1. Susun pseudocode bubble sort teroptimasi dan selection sort.
2. Lakukan tracing minimal sampai data terurut dan catat array pada setiap pass.
3. Hitung jumlah perbandingan dan pertukaran masing-masing algoritma.
4. Uji data acak, sudah terurut, terbalik, duplikat, satu elemen, dan data kosong.
5. Implementasikan kedua algoritma pada Dev C++ dengan counter operasi.
6. Ukur waktu untuk n = 100, 1.000, 5.000, dan 10.000.
7. Jelaskan kompleksitas $O(n^2)$, efek early-stop, dan perbedaan jumlah swap antara kedua algoritma.

| Dataset | Bubble: Banding | Bubble: Swap | Selection: Banding | Selection: Swap |
|---|---|---|---|---|
| Acak | ... | ... | ... | ... |
| Terurut | ... | ... | ... | ... |
| Terbalik | ... | ... | ... | ... |
| Duplikat | ... | ... | ... | ... |

### Hasil/Luaran yang Dikumpulkan

1. Kode bubble sort dan selection sort.
2. Tabel tracing setiap pass.
3. Tabel jumlah perbandingan dan pertukaran.
4. Tabel waktu eksekusi beberapa ukuran data.
5. Analisis kompleksitas dan efek optimasi.

## Aturan Penilaian (Total Skor : 100)

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL08 | CPMK084 | Kebenaran implementasi bubble sort | 15 |
| 2 | CPL08 | CPMK084 | Kebenaran implementasi selection sort | 15 |
| 3 | CPL08 | CPMK084 | Ketepatan tracing setiap pass | 15 |
| 4 | CPL08 | CPMK084 | Ketepatan perhitungan perbandingan dan swap | 15 |
| 5 | CPL08 | CPMK084 | Analisis kompleksitas $O(n^2)$ | 15 |
| 6 | CPL08 | CPMK084 | Penerapan dan analisis optimasi | 15 |
| 7 | CPL08 | CPMK084 | Kesimpulan perbandingan efisiensi | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 11

# Merge Sort dengan Pendekatan Divide and Conquer

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| CPL08 | Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| CPMK084 | Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| Sub-CPMK | Mahasiswa mampu mengimplementasikan merge sort dengan pendekatan divide and conquer serta menganalisis kompleksitas waktu dan ruangnya. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menjelaskan strategi divide and conquer, menerapkan proses divide dan merge secara rekursif, melakukan tracing pembagian dan penggabungan subarray, serta menganalisis kompleksitas waktu O(n log n) dan kebutuhan memori tambahan. Mahasiswa juga mampu membandingkan performa merge sort dengan algoritma pengurutan sederhana.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 11 apabila mampu:

1. Menjelaskan konsep divide and conquer pada merge sort.
2. Mengimplementasikan proses divide dan fungsi merge dengan benar.
3. Melakukan tracing pohon pembagian dan tahap penggabungan.
4. Menganalisis kompleksitas waktu O(n log n).
5. Menganalisis space complexity akibat buffer sementara dan rekursi.
6. Membandingkan performa merge sort dengan bubble/selection sort pada data besar.

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh

keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

**1. Konsep Divide and Conquer**

Strategi divide and conquer membagi masalah menjadi submasalah lebih kecil, menyelesaikan setiap bagian, kemudian menggabungkan hasil. Merge sort menggunakan pola ini secara alami sehingga struktur algoritmanya mudah dianalisis melalui pohon rekursi.

Contoh data: 80 60 90 70

Prosesnya:

```
DIVIDE
[80 60 90 70]
      ↓
[80 60] [90 70]
      ↓
[80] [60] [90] [70]

CONQUER
Setiap bagian satu elemen dianggap sudah terurut.

MERGE
[60 80] [70 90]
      ↓
[60 70 80 90]
```

Perintah utama untuk menjalankan proses tersebut:

```
mergeSort(data, 0, n - 1);
```

Artinya fungsi mergeSort() akan memproses seluruh array dari indeks pertama sampai indeks terakhir.

**2. Tahap Divide**

Array dibagi menjadi dua bagian berdasarkan indeks tengah. Pembagian dilanjutkan secara rekursif sampai subarray berukuran satu elemen. Subarray satu elemen dianggap sudah terurut dan menjadi base case.

Menentukan posisi tengah:

```
int mid = left + (right - left) / 2;
```

Kemudian membagi bagian kiri dan kanan:

```
mergeSort(data, left, mid);
mergeSort(data, mid + 1, right);
```

Contoh:

Data : 80 60 90 70
Indeks : 0 1 2 3
Maka:

```
left  = 0;
right = 3;
mid   = 1;
```

Data dibagi menjadi:

Kiri : indeks 0 - 1 → 80 60

Kanan : indeks 2 - 3 → 90 70

**3. Tahap Merge**

Dua subarray terurut digabung menggunakan dua pointer. Elemen terkecil pada kedua subarray dipilih bertahap sampai salah satu bagian habis, kemudian sisa elemen disalin. Ketepatan indeks sangat penting pada tahap ini.

Misalnya:

Kiri : 60 80

Kanan : 70 90

Perbandingan pertama:

```
if (kiri[i] <= kanan[j]) {
    data[k] = kiri[i];
}
```

Karena:

60 < 70

maka 60 dimasukkan lebih dulu.

Contoh fungsi merge() sederhana:

```
void merge(int data[], int left, int mid, int right) {

    int n1 = mid - left + 1;
    int n2 = right - mid;

    int kiri[50];
    int kanan[50];

    for (int i = 0; i < n1; i++) {
        kiri[i] = data[left + i];
    }

    for (int j = 0; j < n2; j++) {
        kanan[j] = data[mid + 1 + j];
    }

    int i = 0;
    int j = 0;
    int k = left;

    while (i < n1 && j < n2) {

        if (kiri[i] <= kanan[j]) {
            data[k] = kiri[i];
            i++;
        }
        else {
            data[k] = kanan[j];
            j++;
        }

        k++;
    }

    while (i < n1) {
        data[k] = kiri[i];
        i++;
        k++;
    }

    while (j < n2) {
        data[k] = kanan[j];
        j++;
        k++;
    }
}
```

**4. Rekursi pada merge sort**

Setiap pemanggilan mergeSort(left,right) membagi interval menjadi [left,mid] dan [mid+1,right]. Kedalaman rekursi sekitar log2(n), sehingga call stack tumbuh O(log n).

Contoh:

```
void mergeSort(int data[], int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(data, left, mid);

        mergeSort(data, mid + 1, right);

        merge(data, left, mid, right);
    }
}
```

**5. Kompleksitas waktu**

Merge Sort memiliki kompleksitas:

- Best case = O(n log n)
- Average case = O(n log n)
- Worst case = O(n log n)

Contoh:

| n | Perkiraan Level Divide | Kompleksitas |
|---|---|---|
| 8 | 3 | O(8 × 3) |
| 16 | 4 | O(16 × 4) |
| 32 | 5 | O(32 × 5) |
| 64 | 6 | O(64 × 6) |

**6. Kompleksitas ruang dan stabilitas**

Implementasi merge sort klasik membutuhkan buffer tambahan O(n). Keuntungan pentingnya adalah performa yang konsisten dan dapat dibuat stabil, yaitu record berkunci sama mempertahankan urutan relatifnya.

**7. Mekanisme Kerja dan Contoh Penerapan**

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini

memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

**Contoh Penerapan - Merge Sort Nilai Ujian**

Urutkan [38, 27, 43, 3, 9, 82, 10]. Data dibagi sampai satu elemen kemudian digabung bertahap dalam keadaan terurut.

| Tahap | Bagian/Hasil |
|---|---|
| Divide 1 | [38,27,43,3] \| [9,82,10] |
| Divide lanjut | [38,27] [43,3] [9,82] [10] |
| Merge awal | [27,38] [3,43] [9,82] [10] |
| Merge akhir | [3,9,10,27,38,43,82] |

```
PROCEDURE MergeSort(A, left, right)
    IF left < right THEN
        mid <- (left + right) / 2
        MergeSort(A, left, mid)
        MergeSort(A, mid+1, right)
        Merge(A, left, mid, right)
    ENDIF
END PROCEDURE
```

### Implementasi Kode Dev C++

```
#include <iostream>
using namespace std;
// Fungsi untuk menggabungkan dua bagian array
void merge(int data[], int kiri, int tengah, int kanan) {

    int n1 = tengah - kiri + 1;
    int n2 = kanan - tengah;
    int L[50], R[50];

    // Salin data ke array sementara
    for (int i = 0; i < n1; i++) {
        L[i] = data[kiri + i];
    }

    for (int j = 0; j < n2; j++) {
        R[j] = data[tengah + 1 + j];
    }

    int i = 0;
    int j = 0;
    int k = kiri;

    // Proses penggabungan
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            data[k] = L[i];
            i++;
        }
        else {
            data[k] = R[j];
            j++;
        }
        k++;
    }

    // Salin sisa data bagian kiri
    while (i < n1) {
        data[k] = L[i];
        i++;
        k++;
    }

    // Salin sisa data bagian kanan
    while (j < n2) {
        data[k] = R[j];
        j++;
        k++;
    }
}
```

```cpp
// Fungsi Merge Sort
void mergeSort(int data[], int kiri, int kanan) {

    if (kiri < kanan) {
        int tengah = (kiri + kanan) / 2;

        // Divide bagian kiri
        mergeSort(data, kiri, tengah);
        // Divide bagian kanan
        mergeSort(data, tengah + 1, kanan);
        // Merge
        merge(data, kiri, tengah, kanan);
    }
}

int main() {

    int data[6] = {80, 60, 90, 70, 50, 85};
    int n = 6;
    cout << "Data sebelum diurutkan:" << endl;
    for (int i = 0; i < n; i++) {
        cout << data[i] << " ";
    }

    // Proses Merge Sort
    mergeSort(data, 0, n - 1);
    cout << "\n\nData setelah diurutkan:" << endl;
    for (int i = 0; i < n; i++) {
        cout << data[i] << " ";
    }

    return 0;
}
```

**Output:**

```
D:\1. LABORATORIUM\MODU
Data sebelum diurutkan:
80 60 90 70 50 85

Data setelah diurutkan:
50 60 70 80 85 90
--------------------------------
Process exited after 0.2029 seconds with return value 0
Press any key to continue . . .
```

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

### 3. Daftar Kegiatan Praktikum

a) Menggambarkan proses pembagian data menjadi subarray.
b) Mengimplementasikan fungsi merge dan merge sort.
c) Melakukan tracing proses divide dan merge.
d) Mengukur waktu eksekusi untuk beberapa ukuran data.
e) Membandingkan waktu dan penggunaan ruang dengan bubble sort.

### 4. Studi Kasus

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Merge Sort Nilai Ujian | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Merge Sort Transaksi | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Merge Sort Data Duplikat | 40 |
| | | **Total** | **100** |

#### 1) Studi Kasus 1 - Merge Sort Nilai Ujian

Urutkan nilai ujian menggunakan merge sort. Gambar pohon pembagian, trace proses merge, dan bandingkan waktu dengan bubble sort.

#### 2) Studi Kasus 2 - Merge Sort Transaksi

Urutkan nominal transaksi berjumlah besar. Uji n bertambah dan catat waktu serta penggunaan buffer sementara.

#### 3) Studi Kasus 3 - Merge Sort Data Duplikat

Urutkan record dengan beberapa kunci sama. Amati kestabilan urutan relatif dan jelaskan manfaat stabilitas sorting.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 11

### Merge Sort Data Transaksi

Sebuah aplikasi menyimpan nominal transaksi harian. Data perlu diurutkan menaik menggunakan merge sort dan dibandingkan dengan bubble sort untuk melihat perbedaan performa pada ukuran data yang meningkat.

### Soal Evaluasi

1. Gambarkan pohon divide untuk data [42,17,8,99,23,55,12,31].
2. Susun pseudocode MergeSort() dan Merge() lengkap dengan parameter indeks.
3. Lakukan tracing proses merge sampai seluruh data terurut.
4. Implementasikan merge sort dan bubble sort pada Dev C++.
5. Uji data acak, terurut, terbalik, dan duplikat untuk n = 100, 1.000, 10.000.
6. Catat waktu eksekusi dan jelaskan mengapa merge sort mempertahankan O(n log n).
7. Analisis kebutuhan buffer tambahan dan trade-off time-space.

| n | Merge Sort | Bubble Sort | Catatan |
|---|---|---|---|
| 100 | ... µs | ... µs | ... |
| 1.000 | ... µs | ... µs | ... |
| 10.000 | ... µs | ... µs | ... |

### Hasil/Luaran yang Dikumpulkan

1. Diagram divide-and-conquer.
2. Pseudocode dan kode merge sort.
3. Tabel tracing divide/merge.
4. Tabel perbandingan waktu dengan bubble sort.
5. Analisis kompleksitas waktu dan ruang.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL08 | CPMK084 | Pemahaman divide and conquer | 15 |
| 2 | CPL08 | CPMK084 | Kebenaran implementasi merge sort | 20 |
| 3 | CPL08 | CPMK084 | Ketepatan tracing divide dan merge | 15 |
| 4 | CPL08 | CPMK084 | Analisis kompleksitas waktu | 15 |
| 5 | CPL08 | CPMK084 | Analisis space complexity | 10 |
| 6 | CPL08 | CPMK084 | Perbandingan dengan sorting sederhana | 15 |
| 7 | CPL08 | CPMK084 | Pengujian dan kebenaran hasil | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 12

# Quicksort dan Strategi Pemilihan Pivot

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| CPL08 | Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| CPMK084 | Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| Sub-CPMK | Mahasiswa mampu mengimplementasikan quicksort dengan berbagai strategi pivot serta menganalisis best case, average case, dan worst case. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu memahami proses partition, memilih dan menguji beberapa strategi pivot, mengimplementasikan quicksort rekursif, melakukan tracing posisi pivot serta subarray, dan menganalisis pengaruh keseimbangan partisi terhadap performa. Mahasiswa mampu menjelaskan average case $O(n \log n)$, worst case $O(n^2)$, dan strategi sederhana untuk mengurangi risiko partisi buruk.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 12 apabila mampu:

1. Menjelaskan fungsi partition dan peran pivot.
2. Mengimplementasikan quicksort rekursif dengan partition yang benar.
3. Menguji strategi pivot pertama, terakhir, tengah, atau acak.
4. Melakukan tracing hasil partisi dan posisi pivot.
5. Menganalisis best/average $O(n \log n)$ dan worst $O(n^2)$.
6. Membandingkan strategi pivot pada data acak, terurut, dan terbalik.

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh

keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

1. **Partition**

Partition mengatur elemen sehingga bagian tertentu berada di sisi kiri pivot dan elemen lainnya di sisi kanan sesuai aturan perbandingan. Setelah partition, pivot berada pada posisi yang tidak perlu dipindahkan lagi pada langkah berikutnya.

Contoh data: 70 40 80 20 60

Jika pivot adalah 60, maka setelah partition secara konsep:

40 20 | 60 | 70 80

Contoh fungsi partition sederhana:

```
int partition(int data[], int low, int high) {

    int pivot = data[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {

        if (data[j] < pivot) {

            i++;

            int temp = data[i];
            data[i] = data[j];
            data[j] = temp;
        }
    }

    int temp = data[i + 1];
    data[i + 1] = data[high];
    data[high] = temp;

    return i + 1;
}
```

2. **Pemilihan pivot**

Pivot dapat dipilih dari elemen pertama, terakhir, tengah, median-of-three, atau acak. Pilihan pivot memengaruhi keseimbangan ukuran subarray dan kedalaman rekursi.

**a. Pivot Elemen Pertama**

int pivot = data[low];

Contoh:

Data : 50 20 70 10 40

Pivot : 50

**b. Pivot Elemen Terakhir**

int pivot = data[high];

Contoh:

Data : 50 20 70 10 40

Pivot : 40

Strategi ini paling mudah digunakan bersama partition sederhana.

**c. Pivot Elemen Tengah**

int tengah = (low + high) / 2;

int pivot = data[tengah];

Contoh:

Data : 50 20 70 10 40

↑

Pivot

**d. Pivot Acak**

Jika ingin mencoba strategi pivot acak:

int indeksPivot = low + rand() % (high - low + 1);

int pivot = data[indeksPivot];

**3. Quicksort rekursif**

Setelah partition menghasilkan indeks pivot p, quicksort dipanggil pada bagian left..p-1 dan p+1..right. Base case terjadi ketika left >= right.

**4. Best dan average case**

Jika partition relatif seimbang, kedalaman rekursi sekitar log n dan total kerja setiap level O(n), sehingga waktu keseluruhan O(n log n). Pada data acak, perilaku rata-rata biasanya mendekati kondisi ini.

**5. Worst case**

Jika pivot berulang kali menjadi elemen terkecil atau terbesar, satu subarray hampir sebesar n-1 dan yang lain kosong. Kedalaman rekursi menjadi O(n), menghasilkan waktu $O(n^2)$ dan penggunaan stack lebih besar.

**6. Cara Kerja Quick Sort**

**Pseudocode Sederhana:**

```
ALGORITMA QuickSort(arr, awal, akhir)
JIKA awal < akhir MAKA
    1. Pilih pivot (biasanya elemen terakhir)
    2. Pisahkan array (partition):
       - Pindahkan semua angka < pivot ke kiri
       - Pindahkan semua angka > pivot ke kanan
       - Pivot di tengah
    3. Ulangi QuickSort untuk bagian kiri
    4. Ulangi QuickSort untuk bagian kanan
SELESAI
```

## 7. Mekanisme Kerja dan Contoh Penerapan

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

### Contoh Penerapan

### Quicksort dengan Pivot Terakhir

Gunakan data [9, 4, 7, 3, 10, 5] dengan pivot terakhir 5. Setelah partition, elemen <=5 ditempatkan di kiri dan elemen >5 di kanan, lalu proses diterapkan secara rekursif pada kedua bagian.

```
PROCEDURE QuickSort(A, low, high)
    IF low < high THEN
        p <- Partition(A, low, high)
        QuickSort(A, low, p-1)
        QuickSort(A, p+1, high)
    ENDIF
END PROCEDURE
```

| Strategi Pivot | Data Acak | Data Terurut | Risiko Worst Case |
|---|---|---|---|
| Pertama | baik/bervariasi | tinggi | tinggi |
| Terakhir | baik/bervariasi | tinggi | tinggi |
| Tengah | sering lebih seimbang | lebih baik | sedang |
| Acak | umumnya baik | umumnya baik | lebih rendah |

### 8. Implementasi Kode Dev C++

```cpp
#include <iostream>
using namespace std;

// Fungsi untuk menukar dua nilai
void tukar(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

// Fungsi Partition
int partition(int data[], int low, int high) {

    // Elemen terakhir dijadikan pivot
    int pivot = data[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        // Jika data lebih kecil dari pivot
        if (data[j] < pivot) {

            i++;

            tukar(data[i], data[j]);
        }
    }

    // Letakkan pivot pada posisi yang benar
    tukar(data[i + 1], data[high]);
    return i + 1;
}

// Fungsi Quick Sort
void quickSort(int data[], int low, int high) {

    if (low < high) {
        // Menentukan posisi pivot
        int posisiPivot = partition(data, low, high);
        // Mengurutkan bagian kiri pivot
        quickSort(data, low, posisiPivot - 1);
        // Mengurutkan bagian kanan pivot
        quickSort(data, posisiPivot + 1, high);
    }
}
```

```
int main() {

    int data[] = {70, 40, 80, 20, 60};
    int n = 5;

    cout << "Data sebelum diurutkan:" << endl;

    for (int i = 0; i < n; i++) {
        cout << data[i] << " ";
    }

    // Proses Quick Sort
    quickSort(data, 0, n - 1);
    cout << "\n\nData setelah diurutkan:" << endl;
    for (int i = 0; i < n; i++) {
        cout << data[i] << " ";
    }

    return 0;
}
```

**Output:**

```
D:\1. LABORATORIUM\MODU
Data sebelum diurutkan:
70 40 80 20 60

Data setelah diurutkan:
20 40 60 70 80
--------------------------------
Process exited after 0.1856 seconds with return value 0
Press any key to continue . . .
```

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.

d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

**3. Daftar Kegiatan Praktikum**

a) Mengimplementasikan fungsi partition dan quicksort.
b) Membuat beberapa strategi pemilihan pivot.
c) Melakukan tracing posisi pivot dan subarray hasil partition.
d) Menguji data acak, terurut, terbalik, dan banyak duplikat.
e) Membandingkan waktu dan kedalaman rekursi antar strategi.

**4. Studi Kasus**

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Quicksort Nilai Mahasiswa | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Perbandingan Strategi Pivot | 30 |
| CPL08 | CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Best vs Worst Case | 40 |
| | | **Total** | **100** |

**1) Studi Kasus 1 - Quicksort Nilai Mahasiswa**

Urutkan nilai dengan pivot terakhir. Trace setiap partition, posisi pivot, dan subarray yang terbentuk.

**2) Studi Kasus 2 - Perbandingan Strategi Pivot**

Gunakan pivot pertama, tengah, dan acak pada dataset yang sama. Catat waktu dan kedalaman/ jumlah partition.

**3) Studi Kasus 3 - Best vs Worst Case**

Uji data acak, sudah terurut, dan terbalik pada n meningkat. Jelaskan kondisi yang mendekati $O(n \log n)$ dan $O(n^2)$.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 12

### Analisis Strategi Pivot Quicksort

Diberikan beberapa dataset dengan pola berbeda. Mahasiswa harus mengimplementasikan quicksort menggunakan tiga strategi pivot dan menganalisis pengaruhnya terhadap partition serta waktu eksekusi.

### Soal Evaluasi

1. Susun pseudocode partition dan quicksort rekursif.
2. Implementasikan pivot pertama, pivot tengah, dan pivot acak/terakhir.
3. Lakukan tracing partition data [8,3,1,7,0,10,2] untuk salah satu strategi.
4. Uji data acak, terurut, terbalik, dan duplikat untuk n = 100, 1.000, 10.000.
5. Catat jumlah partition/perbandingan atau waktu eksekusi setiap strategi.
6. Jelaskan mengapa pivot ekstrem pada data terurut dapat menghasilkan worst case $O(n^2)$.
7. Bandingkan quicksort dengan merge sort dari sisi average time, worst case, dan kebutuhan memori.

| Dataset | Pivot Pertama | Pivot Tengah | Pivot Lain | Analisis |
|---|---|---|---|---|
| Acak | ... | ... | ... | ... |
| Terurut | ... | ... | ... | ... |
| Terbalik | ... | ... | ... | ... |
| Duplikat | ... | ... | ... | ... |

### Hasil/Luaran yang Dikumpulkan

1. Pseudocode partition dan quicksort.
2. Kode quicksort dengan minimal tiga strategi pivot.
3. Tabel tracing partition.
4. Tabel hasil eksperimen beberapa pola data.
5. Analisis best/average/worst case dan strategi pivot.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL08 | CPMK084 | Ketepatan fungsi partition | 15 |
| 2 | CPL08 | CPMK084 | Implementasi strategi pivot | 15 |
| 3 | CPL08 | CPMK084 | Kebenaran quicksort rekursif | 20 |
| 4 | CPL08 | CPMK084 | Ketepatan tracing partition | 15 |
| 5 | CPL08 | CPMK084 | Analisis best/average/worst case | 15 |
| 6 | CPL08 | CPMK084 | Eksperimen dan perbandingan strategi | 10 |
| 7 | CPL08 | CPMK084 | Kesimpulan performa dan trade-off | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 13

# Mini Project Integrasi Searching dan Sorting

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| CPL03 / CPL08 | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| CPMK032 / CPMK084 | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| Sub-CPMK | Mahasiswa mampu mengintegrasikan algoritma searching dan sorting dalam mini project serta menyajikan hasil analisis performa secara sistematis. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menganalisis kebutuhan mini project, merancang solusi modular, mengintegrasikan minimal satu algoritma sorting dan satu algoritma searching, menangani prasyarat data pencarian, serta melakukan pengujian dan pengukuran performa. Hasil implementasi harus didokumentasikan melalui flowchart/pseudocode, test case, analisis kompleksitas, dan laporan teknis.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 13 apabila mampu:

1. Menentukan kebutuhan, struktur data, input, proses, dan output mini project.
2. Menyusun rancangan modular serta alur program yang jelas.
3. Mengimplementasikan dan mengintegrasikan algoritma sorting.
4. Mengimplementasikan searching sesuai prasyarat data.
5. Menangani data kosong, duplikat, dan data tidak ditemukan.
6. Melakukan pengujian fungsional dan analisis performa.

7. Menyusun dokumentasi teknis yang konsisten dengan implementasi.

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

### 1. Analisis kebutuhan mini project

Mini project dimulai dari definisi masalah, pengguna, data yang dikelola, dan keluaran yang dibutuhkan. Kebutuhan perlu dibuat cukup sempit agar dapat diselesaikan dalam waktu praktikum tetapi tetap menunjukkan integrasi algoritma.

Analisis kebutuhan dilakukan untuk menentukan **data yang dikelola, proses yang dibutuhkan, dan keluaran yang harus dihasilkan**. Pada mini project Data Mahasiswa, misalnya:

- Data: NIM, nama, nilai.
- Input: data mahasiswa.
- Proses: tampil data, sorting, searching.
- Output: data terurut dan hasil pencarian.

**Contoh stuktur data:**

```
struct Mahasiswa {
    string nim;
    string nama;
    float nilai;
};
```

**Contoh mendeklarasikan beberapa data:**

```
Mahasiswa data[5] = {
    {"13020230001", "Andi", 80},
    {"13020230002", "Budi", 75},
    {"13020230003", "Citra", 90},
    {"13020230004", "Dinda", 70},
    {"13020230005", "Fajar", 85}
};
```

**Contoh menerima input:**

```
cout << "NIM   : ";
cin >> data[i].nim;

cout << "Nama  : ";
cin >> data[i].nama;

cout << "Nilai : ";
cin >> data[i].nilai;
```

**Contoh menampilkan data:**

```
cout << data[i].nim << " "
     << data[i].nama << " "
     << data[i].nilai << endl;
```

2. **Perancangan modular**

Program dibagi menjadi fungsi seperti inputData, tampilData, sortData, searchData, validasi, dan laporan. Modularitas membuat algoritma dapat diuji per bagian dan memudahkan debugging.

**Contoh rancangan fungsi:**

```
void inputData();
void tampilData();
void sortingData();
int searchingData();
```

**Contoh fungsi menampilkan data:**

```
void tampilData(Mahasiswa data[], int n) {

    for (int i = 0; i < n; i++) {

        cout << data[i].nim << " "
             << data[i].nama << " "
             << data[i].nilai << endl;
    }
}
```

3. **Integrasi sorting dan searching**

Jika binary search digunakan, data harus disortir berdasarkan kunci yang sama. Jika data sering berubah, program harus menentukan kapan sorting

dilakukan. Linear search dapat digunakan tanpa sorting tetapi memiliki biaya O(n).

**4. Pemilihan algoritma**

Pemilihan sorting dan searching harus disertai alasan. Untuk dataset kecil, algoritma sederhana dapat cukup. Untuk data lebih besar, merge sort atau quicksort rata-rata memberi performa lebih baik dibanding $O(n^2)$.

**5. Pengukuran performa**

Program dapat mencatat waktu sorting, waktu searching, jumlah perbandingan, atau ukuran data. Pengukuran digunakan untuk mendukung analisis, bukan sekadar menunjukkan program dapat berjalan.

**6. Dokumentasi dan traceability**

Dokumen rancangan harus konsisten dengan kode. Nama fungsi, alur proses, test case, serta hasil analisis sebaiknya dapat ditelusuri dari kebutuhan awal sampai output akhir. Ini membantu review dan presentasi pada modul berikutnya.

**7. Mekanisme Kerja dan Contoh Penerapan**

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

**Contoh Penerapan**

**Mini Project Integrasi Searching dan Sorting Data Mahasiswa**

Program menggunakan Bubble Sort berdasarkan NIM dan Binary Search untuk pencarian NIM.

| Modul | Tugas | Algoritma |
|---|---|---|
| InputData | Menerima dan validasi record | Sekuensial/seleksi |
| SortData | Mengurutkan record | Bubble/Merge/Quick Sort |
| SearchData | Mencari berdasarkan kunci | Linear/Binary Search |
| Report | Menampilkan hasil dan metrik | Iterasi |

**Implementasi Kode Dev C++**

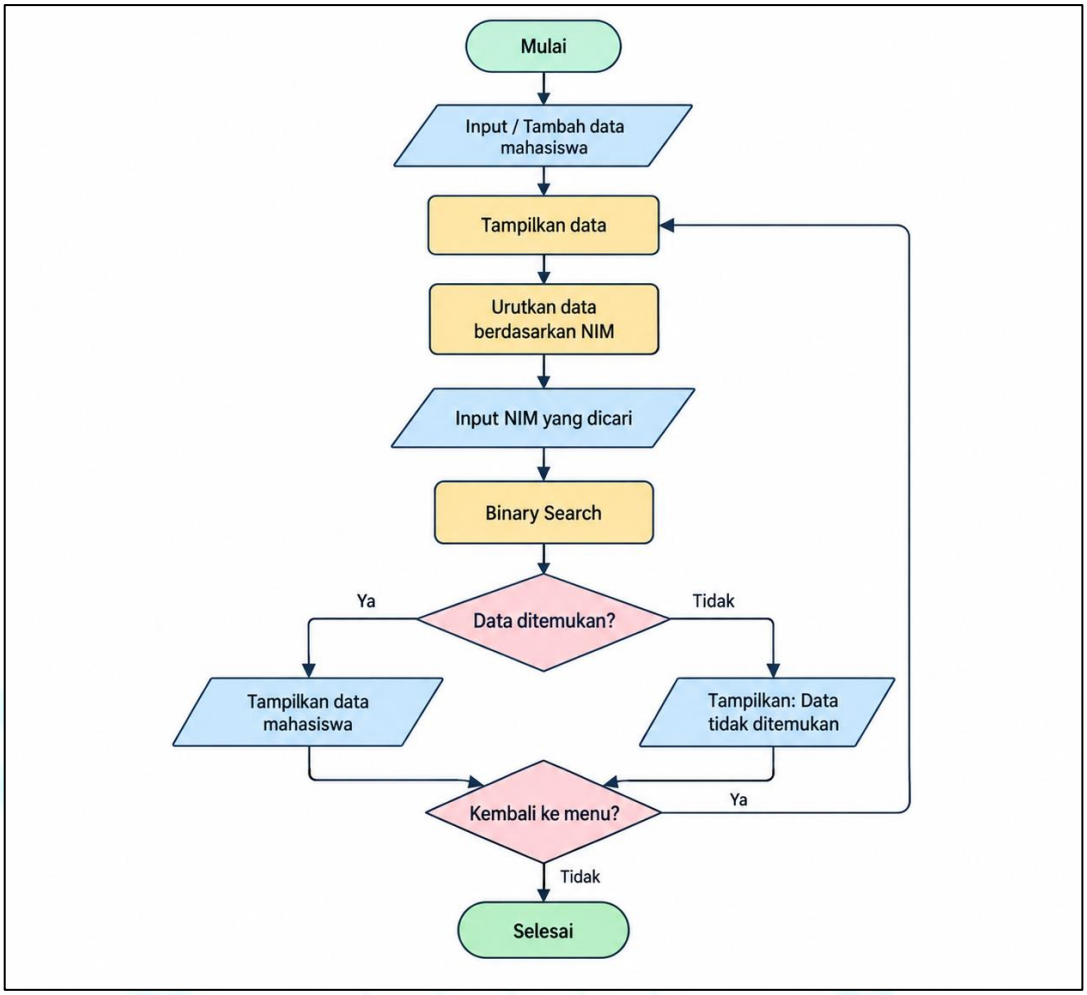
```cpp
#include <iostream>
#include <string>
using namespace std;
// Struktur data mahasiswa
struct Mahasiswa {
    string nim;
    string nama;
    float nilai;
};

// Fungsi menampilkan seluruh data mahasiswa
void tampilData(Mahasiswa data[], int jumlah) {

    if (jumlah == 0) {
        cout << "\nData mahasiswa masih kosong.\n";
        return;
    }

    cout << "\n=======================================\n";
    cout << "           DATA MAHASISWA\n";
    cout << "=======================================\n";

    for (int i = 0; i < jumlah; i++) {

        cout << "Data ke-" << i + 1 << endl;
        cout << "NIM   : " << data[i].nim << endl;
        cout << "Nama  : " << data[i].nama << endl;
        cout << "Nilai : " << data[i].nilai << endl;

        cout << "---------------------------------------\n";
    }
}

// Fungsi Bubble Sort berdasarkan NIM
void bubbleSort(Mahasiswa data[], int jumlah) {

    for (int i = 0; i < jumlah - 1; i++) {
        for (int j = 0; j < jumlah - 1 - i; j++) {
            if (data[j].nim > data[j + 1].nim) {
                Mahasiswa temp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = temp;
            }
        }
    }
}
```

```
// Fungsi Binary Search berdasarkan NIM
int binarySearch(Mahasiswa data[],
                 int jumlah,
                 string target) {

    int low = 0;
    int high = jumlah - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        if (data[mid].nim == target) {
            return mid;
        }

        else if (data[mid].nim < target) {
            low = mid + 1;
        }
        else {

            high = mid - 1;
        }
    }

    return -1;
}

int main() {

    Mahasiswa data[50];
    int jumlah = 0;
    int pilihan;

    do {

        cout << "\n========================================\n";
        cout << "      MINI PROJECT DATA MAHASISWA\n";
        cout << "========================================\n";
        cout << "1. Tambah Data Mahasiswa\n";
        cout << "2. Tampilkan Data Mahasiswa\n";
        cout << "3. Urutkan Data Berdasarkan NIM\n";
        cout << "4. Cari Mahasiswa Berdasarkan NIM\n";
        cout << "0. Keluar\n";
        cout << "========================================\n";
        cout << "Pilih menu : ";
        cin >> pilihan;
```

```
        switch (pilihan) {
            // ==================================
            // MENU 1 - TAMBAH DATA
            // ==================================
            case 1: {
                if (jumlah >= 50) {
                    cout << "\nData mahasiswa sudah penuh.\n";
                    break;
                }

                cout << "\n========================================\n";
                cout << "        TAMBAH DATA MAHASISWA\n";
                cout << "========================================\n";
                cout << "Masukkan NIM   : ";
                cin >> data[jumlah].nim;

                cout << "Masukkan Nama  : ";
                cin >> ws;
                getline(cin, data[jumlah].nama);

                cout << "Masukkan Nilai : ";
                cin >> data[jumlah].nilai;
                jumlah++;
                cout << "\nData mahasiswa berhasil ditambahkan.\n";
                break;
            }
            // ==================================
            // MENU 2 - TAMPILKAN DATA
            // ==================================
            case 2: {
                tampilData(data, jumlah);
                break;
            }
            // ==================================
            // MENU 3 - SORTING DATA
            // ==================================
            case 3: {
                if (jumlah == 0) {
                    cout << "\nData mahasiswa masih kosong.\n";

                } else {
                  bubbleSort(data, jumlah);
                  cout << "\nData berhasil diurutkan berdasarkan NIM.\n";
                  tampilData(data, jumlah);
                }
                break;
            }
```

```
            case 4: {
                if (jumlah == 0) {
                    cout << "\nData mahasiswa masih kosong.\n";
                    break;
                }
                // Binary Search membutuhkan data terurut
                bubbleSort(data, jumlah);
                string cariNIM;
                int posisi;
                cout << "\n========================================\n";
                cout << "          PENCARIAN MAHASISWA\n";
                cout << "========================================\n";
                cout << "Masukkan NIM yang dicari : ";
                cin >> cariNIM;
                posisi = binarySearch(
                    data,
                    jumlah,
                    cariNIM
                );

              if (posisi != -1) {
               cout <<  "\n========================================\n";
               cout << "             DATA DITEMUKAN\n";
               cout << "========================================\n";
                    cout << "NIM   : " << data[posisi].nim
                         << endl;
                    cout << "Nama  : " << data[posisi].nama
                         << endl;
                    cout << "Nilai : " << data[posisi].nilai
                         << endl;
                } else {
                    cout << "\nData mahasiswa tidak ditemukan.\n";
                }
                break;
            }
            case 0: {
                cout << "\nProgram selesai.\n";
                break;
            }
            // ================================
            // PILIHAN TIDAK VALID
            // ================================
            default: {
                cout << "\nPilihan menu tidak valid.\n";
                break;
            }
        }
    } while (pilihan != 0);
    return 0;
}
```

**Flowchart**




Gambar 13. Flowchart Searching dan Sorting Data Mahasiswa

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.
c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.

e) Rapikan perangkat dan area kerja setelah praktikum.

**3. Daftar Kegiatan Praktikum**

a) Menentukan studi kasus dan kebutuhan data mini project.
b) Menyusun flowchart, pseudocode, dan struktur modul.
c) Mengimplementasikan minimal satu sorting dan satu searching.
d) Mengintegrasikan proses pengurutan dan pencarian.
e) Mengukur performa dan menyusun dokumentasi teknis.

**4. Studi Kasus**

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 / CPL08 | CPMK032 / CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Mini Project Data Mahasiswa | 30 |
| CPL03 / CPL08 | CPMK032 / CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Mini Project Inventori | 30 |
| CPL03 / CPL08 | CPMK032 / CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Mini Project Perpustakaan | 40 |
| | | **Total** | **100** |

**1) Studi Kasus 1 - Mini Project Data Mahasiswa**

Kelola NIM, nama, dan nilai. Sediakan pengurutan nilai serta pencarian NIM. Buat fungsi modular, test case, dan analisis performa.

**2) Studi Kasus 2 - Mini Project Inventori**

Kelola kode, nama, stok, dan harga. Integrasikan sorting harga/kode dan searching kode. Tangani data kosong serta kode tidak ditemukan.

**3) Studi Kasus 3 - Mini Project Perpustakaan**

Kelola kode buku, judul, dan status pinjam. Integrasikan sorting dan pencarian serta bandingkan linear dan binary search pada data terurut.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 13

### Rancangan dan Implementasi Mini Project

Mahasiswa menyelesaikan mini project pengelolaan data sederhana yang mengintegrasikan minimal satu algoritma sorting dan satu searching. Project harus modular, dapat diuji, dan menghasilkan analisis performa.

### Soal Evaluasi

1. Tentukan masalah, kebutuhan fungsional, struktur record, input, proses, dan output project.
2. Buat flowchart utama, hierarchy/modul fungsi, dan pseudocode fitur sorting serta searching.
3. Implementasikan minimal satu algoritma sorting dan satu searching tanpa menggunakan fungsi sort/search bawaan sebagai solusi utama.
4. Pastikan prasyarat binary search dipenuhi jika algoritma tersebut digunakan.
5. Siapkan minimal 10 test case yang mencakup normal, boundary, data kosong, duplikat, dan tidak ditemukan.
6. Ukur waktu atau jumlah operasi pada sedikitnya tiga ukuran data.
7. Jelaskan kompleksitas algoritma yang dipilih dan alasan desain.
8. Susun ringkasan hasil project dan daftar masalah yang masih perlu diperbaiki sebelum presentasi final.

| Fitur | Algoritma | Test Case | Hasil | Status |
|---|---|---|---|---|
| Input/Validasi | ... | ... | ... | ... |
| Sorting | ... | ... | ... | ... |
| Searching | ... | ... | ... | ... |
| Integrasi | ... | ... | ... | ... |

### Hasil/Luaran yang Dikumpulkan

1. Dokumen rancangan mini project.
2. Flowchart/pseudocode dan struktur modul.
3. Kode program mini project.
4. Data uji dan test report.
5. Tabel/grafik performa serta analisis kompleksitas.
6. Dokumentasi teknis untuk presentasi final.

## Aturan Penilaian (Total Skor : 100)

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL03 / CPL08 | CPMK032 / CPMK084 | Analisis kebutuhan dan rancangan | 15 |
| 2 | CPL03 / CPL08 | CPMK032 / CPMK084 | Struktur modular program | 10 |
| 3 | CPL03 / CPL08 | CPMK032 / CPMK084 | Implementasi algoritma sorting | 15 |
| 4 | CPL03 / CPL08 | CPMK032 / CPMK084 | Implementasi algoritma searching | 15 |
| 5 | CPL03 / CPL08 | CPMK032 / CPMK084 | Integrasi searching dan sorting | 15 |
| 6 | CPL03 / CPL08 | CPMK032 / CPMK084 | Pengujian dan penanganan kasus tepi | 10 |
| 7 | CPL03 / CPL08 | CPMK032 / CPMK084 | Analisis performa | 10 |
| 8 | CPL03 / CPL08 | CPMK032 / CPMK084 | Dokumentasi teknis | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

MODUL 14

# Testing, Debugging, Review, dan Presentasi Mini

## A. Total Alokasi Waktu : 200 Menit

Pengantar dan Teori : 50 Menit
Praktik/Studi Kasus : 120 Menit
Evaluasi : 30 Menit

## B. Pemenuhan CPL dan CPMK

| | |
|---|---|
| CPL03 / CPL08 | Memiliki kemampuan memahami cara kerja sistem komputer serta menerapkan berbagai algoritma/metode untuk memecahkan masalah dalam suatu organisasi. Memiliki kemampuan untuk mengimplementasikan kebutuhan computing dengan menggunakan berbagai metode/algoritma yang sesuai dengan kebutuhan pengguna. |
| CPMK032 / CPMK084 | Mampu menerapkan berbagai metode/algoritma untuk memecahkan masalah dalam suatu organisasi. Mampu mengimplementasikan kebutuhan computing dengan sistematis. |
| Sub-CPMK | Mahasiswa mampu menyelesaikan implementasi mini project, melakukan testing dan debugging, menyiapkan presentasi, serta mengevaluasi hasil project. |

## C. Deskripsi Capaian Pembelajaran

Mahasiswa diharapkan mampu menyusun test scenario yang sistematis, menjalankan testing untuk kasus normal, boundary, dan invalid, menemukan sumber kesalahan melalui tracing/debugging, melakukan regression test setelah perbaikan, serta mereview performa algoritma. Mahasiswa juga mampu mempresentasikan rancangan, implementasi, data uji, hasil analisis, dan proses perbaikan project secara ringkas dan dapat dipertanggungjawabkan.

## D. Indikator Ketercapaian Pembelajaran

Mahasiswa dinyatakan mencapai tujuan pembelajaran Modul 14 apabila mampu:

1. Menyusun test case normal, boundary, dan invalid berdasarkan fitur project.
2. Mencatat expected result, actual result, dan status pengujian secara konsisten.
3. Mengidentifikasi bug menggunakan tracing atau debugger dan memperbaikinya.
4. Melakukan regression test setelah perubahan.
5. Mereview ketepatan algoritma dan analisis performa.
6. Menyiapkan versi final program dan dokumentasi.

7. Mempresentasikan project serta menjawab pertanyaan teknis dengan bukti implementasi.

## E. Teori Pendukung

Materi pada modul ini menekankan pemahaman konsep, kemampuan melakukan tracing, implementasi program, serta analisis efisiensi. Setiap konsep perlu dibuktikan melalui data uji agar mahasiswa tidak hanya memperoleh keluaran akhir, tetapi juga memahami proses internal algoritma dan kondisi yang dapat menyebabkan hasil tidak sesuai.

### 1. Testing fungsional

Testing fungsional memeriksa apakah fitur menghasilkan keluaran sesuai kebutuhan tanpa bergantung pada detail internal implementasi. Setiap kebutuhan utama seharusnya memiliki satu atau lebih test case.

### 2. Kasus normal, boundary, dan invalid

Kasus normal mewakili penggunaan umum. Boundary menguji nilai tepat pada batas seperti data kosong, satu elemen, indeks awal/akhir, atau ukuran maksimum. Invalid menguji input yang tidak memenuhi aturan dan memastikan program menolaknya dengan aman.

### 3. Expected dan actual result

Test report yang baik mencatat input, langkah, expected result, actual result, dan status Pass/Fail. Perbedaan antara expected dan actual menjadi dasar investigasi bug.

### 4. Debugging berbasis bukti

Debugging dilakukan dengan mereproduksi bug, mempersempit lokasi kesalahan, melakukan tracing variabel/batas indeks, memperbaiki sumber masalah, kemudian menguji ulang. Mengubah kode tanpa bukti sering menciptakan bug baru.

### 5. Regression testing

Setelah bug diperbaiki, test case lama dijalankan kembali untuk memastikan perubahan tidak merusak fitur yang sebelumnya benar. Regression test sangat penting pada mini project yang beberapa fiturnya saling bergantung.

### 6. Review performa dan presentasi

Final review memeriksa kebenaran output, kompleksitas algoritma, waktu pengujian, penanganan kasus tepi, dan keterbacaan kode. Presentasi harus menjelaskan masalah, rancangan, algoritma, bukti pengujian, hasil performa, serta satu contoh bug dan proses perbaikannya.

### 7. Mekanisme Kerja dan Contoh Penerapan

Dalam praktikum, mahasiswa disarankan memulai dari contoh data kecil, melakukan tracing manual, kemudian menerjemahkan algoritma ke Dev C++. Setelah hasil manual dan program konsisten, pengujian diperluas ke data normal, boundary, invalid, serta ukuran input yang lebih besar. Pendekatan ini memisahkan kesalahan konsep dari kesalahan sintaks dan membantu analisis performa secara lebih terukur.

**Contoh Penerapan - Debugging Binary Search pada Mini Project**

1) Menyusun Test Case Normal, Boundary, dan Invalid
   Misalnya fitur yang diuji adalah **Tambah Data, Sorting berdasarkan NIM, dan Searching berdasarkan NIM**.

| ID | Jenis Test | Fitur | Input/Kondisi | Expected Result |
|---|---|---|---|---|
| TC-01 | Normal | Tambah Data | NIM 13020230001, Nama Andi, Nilai 80 | Data berhasil disimpan |
| TC-02 | Normal | Sorting | NIM: 003, 001, 002 | Urutan menjadi 001, 002, 003 |
| TC-03 | Normal | Searching | Cari NIM 002 | Data mahasiswa NIM 002 ditemukan |
| TC-04 | Boundary | Searching | Target berada pada data pertama | Data pertama ditemukan |
| TC-05 | Boundary | Searching | Target berada pada data terakhir | Data terakhir ditemukan |
| TC-06 | Boundary | Data | Jumlah data = 0 | Muncul pesan **Data masih kosong** |
| TC-07 | Boundary | Data | Hanya terdapat 1 mahasiswa | Program tetap dapat sorting/searching |
| TC-08 | Invalid | Nilai | Nilai = 120 | Input ditolak karena nilai di luar 0–100 |
| TC-09 | Invalid | NIM | NIM kosong | Input ditolak |
| TC-10 | Invalid | Menu | Pilihan menu = 9 | Muncul pesan **Pilihan menu tidak valid** |
| TC-11 | Normal | Searching | NIM tidak terdapat pada data | Muncul pesan **Data tidak ditemukan** |
| TC-12 | Boundary | Kapasitas | Jumlah mahasiswa mencapai 50 | Data ke-51 ditolak |

2) Mencatat Expected Result, Actual Result, dan Status Pengujian
Setelah test case dijalankan, hasil aktual dibandingkan dengan hasil yang diharapkan.

| ID | Input/Kondisi | Expected Result | Actual Result | Status |
|---|---|---|---|---|
| TC-01 | Tambah mahasiswa NIM 001 | Data berhasil disimpan | Data berhasil disimpan | PASS |
| TC-02 | Sorting 003,001,002 | 001,002,003 | 001,002,003 | PASS |
| TC-03 | Cari NIM 002 | Data ditemukan | Data ditemukan | PASS |
| TC-04 | Cari data pertama | Data ditemukan | Data ditemukan | PASS |
| TC-05 | Cari data terakhir | Data ditemukan | Data tidak ditemukan | FAIL |
| TC-06 | Data kosong | Pesan data kosong | Pesan data kosong | PASS |
| TC-08 | Nilai 120 | Input ditolak | Data tetap disimpan | FAIL |
| TC-10 | Menu 9 | Pilihan tidak valid | Pilihan tidak valid | PASS |

Aturannya sederhana:
**PASS** diberikan apabila: Expected Result = Actual Result
Sedangkan **FAIL** apabila: Expected Result ≠ Actual Result

3) Mengidentifikasi Bug Menggunakan Tracing dan Memperbaikinya
Contoh Bug: Binary Search Tidak Menemukan Data Terakhir
Misalnya fungsi ditulis:

```
while (low < high) {
    int mid = (low + high) / 2;

    if (data[mid].nim == target)
        return mid;
    else if (data[mid].nim < target)
        low = mid + 1;
    else
        high = mid - 1;
}
```

Data:
`001  002  003  004  005`
Target:
`005`

Tracing

| Langkah | Low | High | Mid | NIM Mid | Hasil |
|---|---|---|---|---|---|
| 1 | 0 | 4 | 2 | 003 | Target lebih besar |
| 2 | 3 | 4 | 3 | 004 | Target lebih besar |
| 3 | 4 | 4 | - | - | Loop berhenti |

Masalahnya adalah ketika:
`low = 4`
`high = 4`
kondisi:
`low < high`
bernilai salah sehingga indeks terakhir tidak diperiksa.

**Root Cause**

Kondisi loop seharusnya memperbolehkan:
`low = high`
karena masih terdapat satu elemen yang harus diperiksa.

**Perbaikan**

Ubah:
`while (low < high)`
menjadi:
`while (low <= high)`
Kode yang benar:

```
int binarySearch(Mahasiswa data[], int jumlah, string target) {

    int low = 0;
    int high = jumlah - 1;
    while (low <= high) {

        int mid = (low + high) / 2;

        if (data[mid].nim == target) {
            return mid;
        }
        else if (data[mid].nim < target) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }

    return -1;
}
```

4) Melakukan Regression Test Setelah Perubahan
Setelah bug diperbaiki, tidak cukup hanya menguji kasus yang sebelumnya gagal. Seluruh fungsi pencarian perlu diuji kembali.

| ID | Skenario Regression | Expected | Actual Setelah Fix | Status |
|---|---|---|---|---|
| RT-01 | Target di awal | Ditemukan | Ditemukan | PASS |
| RT-02 | Target di tengah | Ditemukan | Ditemukan | PASS |
| RT-03 | Target di akhir | Ditemukan | Ditemukan | PASS |
| RT-04 | Target tidak ada | Tidak ditemukan | Tidak ditemukan | PASS |
| RT-05 | Data satu elemen, target ada | Ditemukan | Ditemukan | PASS |
| RT-06 | Data satu elemen, target tidak ada | Tidak ditemukan | Tidak ditemukan | PASS |

Kesimpulannya, perubahan kondisi dari:

```
low < high
```

menjadi:

```
low <= high
```

memperbaiki kasus target pada indeks terakhir tanpa merusak test case lain.

5) Mereview Ketepatan Algoritma dan Analisis Performa

Mini project menggunakan dua algoritma utama.

| Fitur | Algoritma | Kompleksitas |
|---|---|---|
| Sorting NIM | Bubble Sort | $O(n^2)$ |
| Searching NIM | Binary Search | O(log n) |

**Review Bubble Sort**

Bubble Sort cukup mudah dipahami dan sesuai untuk data praktikum berukuran kecil.

Contoh:

```
for (int i = 0; i < jumlah - 1; i++) {

    for (int j = 0; j < jumlah - 1 - i; j++) {

        if (data[j].nim > data[j + 1].nim) {

            Mahasiswa temp = data[j];
            data[j] = data[j + 1];
            data[j + 1] = temp;
        }
    }
}
```

Karena terdapat dua loop, kompleksitas waktunya:

**$O(n^2)$**.

Untuk jumlah data yang besar, Bubble Sort kurang efisien dibanding Merge Sort atau Quick Sort.

**Review Binary Search**

Binary Search mengurangi ruang pencarian menjadi setengah pada setiap proses.

Kompleksitas waktunya:

**O(log n)**.

Namun terdapat prasyarat penting:

Data harus sudah terurut berdasarkan NIM sebelum Binary Search dijalankan.

Karena itu kode:

```
bubbleSort(data, jumlah);
```

harus dijalankan sebelum:

```
binarySearch(data, jumlah, cariNIM);
```

**Contoh Analisis Performa**

| Jumlah Data | Bubble Sort | Binary Search |
|---|---|---|
| 10 | O(100) | sekitar 4 langkah |
| 100 | O(10.000) | sekitar 7 langkah |
| 1.000 | O(1.000.000) | sekitar 10 langkah |
| 10.000 | O(100.000.000) | sekitar 14 langkah |

Dari hasil tersebut terlihat bahwa proses pencarian menggunakan Binary Search tetap relatif cepat ketika ukuran data bertambah, sedangkan biaya Bubble Sort meningkat jauh lebih cepat.

6) Menyiapkan Versi Final Program dan Dokumentasi

Sebelum project dinyatakan selesai, versi final harus diperiksa kembali.

**Contoh Identitas Versi Final**

| Komponen | Isi |
|---|---|
| Nama Project | Sistem Data Mahasiswa |
| Bahasa | C++ |
| Compiler | Dev-C++ |
| Data | NIM, Nama, Nilai |
| Sorting | Bubble Sort berdasarkan NIM |
| Searching | Binary Search berdasarkan NIM |
| Kapasitas | Maksimal 50 mahasiswa |
| Versi | Final 1.0 |

Fitur Final

```
1. Tambah Data Mahasiswa
2. Tampilkan Data
3. Urutkan Data Berdasarkan NIM
4. Cari Data Berdasarkan NIM
0. Keluar
```

**Contoh Dokumentasi Teknis Singkat**

**Nama Program:** Mini Project Data Mahasiswa

**Tujuan:**

Mengelola data mahasiswa dan menerapkan integrasi algoritma sorting dan searching.

**Input:**

NIM, nama, dan nilai mahasiswa.

**Proses:**

Data dapat ditambahkan, ditampilkan, diurutkan berdasarkan NIM, dan dicari berdasarkan NIM.

**Algoritma Sorting:**

Bubble Sort.

**Algoritma Searching:**

Binary Search.

**Prasyarat Searching:**

Data harus sudah terurut berdasarkan NIM.

**Output:**

Data mahasiswa, data terurut, atau informasi hasil pencarian.

**Test:**

Normal test, boundary test, invalid test, dan regression test.

**Bug yang ditemukan:**

Binary Search gagal menemukan data terakhir karena kondisi low < high.

**Perbaikan:**

Kondisi diubah menjadi:

```
low <= high
```

**Hasil Final:**

Seluruh test case utama menghasilkan status PASS.

## F. Kegiatan Praktikum

### 1. Instrumen

a) Perangkat komputer/PC/laptop/notebook.
b) Sistem operasi Windows, Linux, atau Mac OS.
c) Flowrun.io atau Flowgorithm untuk perancangan visual.
d) Dev C++ untuk implementasi program.

### 2. Prosedur Umum

a) Baca dan pahami tujuan serta tahapan praktikum.
b) Gunakan fasilitas laboratorium secara bertanggung jawab.

c) Simpan file sesuai format dan penamaan yang ditentukan.
d) Uji program menggunakan test case yang tersedia.
e) Rapikan perangkat dan area kerja setelah praktikum.

**3. Daftar Kegiatan Praktikum**

a) Menyusun test scenario normal, boundary, dan invalid.
b) Melakukan testing seluruh fitur mini project.
c) Mengidentifikasi kesalahan dan melakukan debugging berbasis tracing.
d) Menjalankan regression test dan review performa.
e) Mempresentasikan project dan demonstrasi implementasi langsung.

**4. Studi Kasus**

| CPL | CPMK | Pertanyaan | Skor |
|---|---|---|---|
| CPL03 / CPL08 | CPMK032 / CPMK084 | Selesaikan langkah praktikum – Studi Kasus 1: Uji Fungsional Mini Project | 30 |
| CPL03 / CPL08 | CPMK032 / CPMK084 | Selesaikan langkah praktikum – Studi Kasus 2: Debugging Kasus Pencarian | 30 |
| CPL03 / CPL08 | CPMK032 / CPMK084 | Selesaikan langkah praktikum – Studi Kasus 3: Review dan Presentasi | 40 |
| | | **Total** | **100** |

**1) Studi Kasus 1 - Uji Fungsional Mini Project**

Susun minimal 10 test case yang mencakup input normal, boundary, dan invalid untuk seluruh fitur. Catat expected result, actual result, dan status.

**2) Studi Kasus 2 - Debugging Kasus Pencarian**

Gunakan versi program yang gagal menemukan elemen terakhir. Trace penyebab, perbaiki, lalu jalankan regression test.

**3) Studi Kasus 3 - Review dan Presentasi**

Siapkan demo 5-7 menit yang menjelaskan masalah, rancangan, algoritma, data uji, hasil performa, dan satu bug yang pernah diperbaiki.

## LEMBAR EVALUASI PRAKTIKUM

### Evaluasi Praktikum 14

### Final Testing, Debugging, dan Presentasi Mini Project

Mahasiswa melakukan finalisasi mini project Modul 13 melalui testing terstruktur, debugging, regression test, review performa, dan presentasi. Penilaian menekankan bukti proses, bukan hanya output akhir.

### Soal Evaluasi

1. Susun minimal 12 test case: sedikitnya 5 normal, 4 boundary, dan 3 invalid untuk seluruh fitur utama.
2. Lengkapi test report dengan input, expected result, actual result, status, dan catatan.
3. Pilih minimal satu bug nyata atau bug yang disediakan, reproduksi bug, lakukan tracing, identifikasi root cause, dan dokumentasikan perbaikannya.
4. Lakukan regression test terhadap fitur terkait setelah perbaikan.
5. Review kembali algoritma searching/sorting yang digunakan dan pastikan analisis kompleksitas sesuai implementasi final.
6. Siapkan versi final program yang dapat dikompilasi dan dijalankan tanpa error pada test case yang disepakati.
7. Siapkan presentasi 5-7 menit dan demo yang memuat masalah, rancangan, algoritma, hasil test, performa, bug/fix, dan kesimpulan.
8. Jawab pertanyaan praktik singkat dengan menunjukkan bagian kode atau melakukan modifikasi kecil secara langsung.

| ID | Jenis | Input | Expected | Actual | Status |
|---|---|---|---|---|---|
| TS-01 | Normal | ... | ... | ... | ... |
| TS-02 | Boundary | ... | ... | ... | ... |
| TS-03 | Invalid | ... | ... | ... | ... |
| TS-04 | Regression | ... | ... | ... | ... |

### Hasil/Luaran yang Dikumpulkan

1. Test report lengkap.
2. Log bug, tracing, root cause, dan perbaikan.
3. Hasil regression test.
4. Versi final mini project.
5. Laporan analisis performa final.
6. Slide presentasi dan demo.

**Aturan Penilaian (Total Skor : 100)**

| No | CPL | CPMK | Pertanyaan / Indikator Penilaian | Skor |
|---|---|---|---|---|
| 1 | CPL03 / CPL08 | CPMK032 / CPMK084 | Kelengkapan skenario test | 15 |
| 2 | CPL03 / CPL08 | CPMK032 / CPMK084 | Ketepatan test report | 15 |
| 3 | CPL03 / CPL08 | CPMK032 / CPMK084 | Kemampuan debugging dan tracing | 15 |
| 4 | CPL03 / CPL08 | CPMK032 / CPMK084 | Regression testing | 10 |
| 5 | CPL03 / CPL08 | CPMK032 / CPMK084 | Review performa dan kompleksitas | 10 |
| 6 | CPL03 / CPL08 | CPMK032 / CPMK084 | Kualitas versi final program | 10 |
| 7 | CPL03 / CPL08 | CPMK032 / CPMK084 | Presentasi dan demo | 15 |
| 8 | CPL03 / CPL08 | CPMK032 / CPMK084 | Kemampuan menjelaskan aspek teknis | 10 |
| | | | **Total** | **100** |

## HASIL CAPAIAN PRAKTIKUM

| No | Skema Penilaian | CPL | CPMK | Bobot | Skor (0-100) | Nilai Akhir (Bobot x Skor) |
|---|---|---|---|---|---|---|
| 1. | Teori | | | 15% | | |
| 2. | Studi Kasus | | | 35% | | |
| 3. | Evaluasi Praktikum | | | 25% | | |
| 4. | Asistensi | | | 25% | | |
| | | | | | **Total** | |

**Catatan Asisten :**

**Dosen :**

**Asisten 1 :**

**Asisten 2 :**

# REFERENSI

**Link RPS:**
https://bit.ly/130107RP4ALPRO-RPS

**Referensi Utama:**

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[2] J. Erickson, *Algorithms*. Urbana, IL, USA: University of Illinois, 2019. [Online]. Available: http://jeffe.cs.illinois.edu/teaching/algorithms/

[3] C. A. Shaffer, *Data Structures and Algorithm Analysis*, Version 3.2, C++ Edition. Blacksburg, VA, USA: Virginia Tech, 2013. [Online]. Available: https://people.cs.vt.edu/shaffer/Book/

[4] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Boston, MA, USA: Addison-Wesley Professional, 2011.

[5] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. New York, NY, USA: McGraw-Hill, 2008. [Online]. Available: http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

[6] G. T. Heineman, G. Pollice, and S. Selkow, *Algorithms in a Nutshell: A Practical Guide*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2016.

[7] S. S. Skiena, *The Algorithm Design Manual*, 3rd ed. Cham, Switzerland: Springer, 2020.

[8] A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd ed. Boston, MA, USA: Pearson, 2011.

[9] D. Anggreani, A. P. Wibawa, Purnawansyah, and Herman, "Perbandingan efisiensi algoritma sorting dalam penggunaan bandwidth," *ILKOM Jurnal Ilmiah*, vol. 12, no. 2, pp. 96–103, 2020, doi: 10.33096/ilkom.v12i2.538.96-103.

[10] B. A. Ashad, R. Ramdaniah, S. Sriwijanaka, and S. H. Mansyur, "Pemanfaatan aplikasi pendeteksi warna pakaian berbasis Android bagi penyandang tunanetra di SLB Yukartuni Makassar," *Jurnal Abdi Masyarakat Indonesia*, vol. 2, no. 1, pp. 263–268, 2022, doi: 10.54082/jamsi.191.

**Referensi Tambahan:**

[11] A. B. Downey, *Think Data Structures: Algorithms and Information Retrieval in Java*. Green Tea Press, 2016. [Online]. Available: https://greenteapress.com/wp/think-data-structures/

[12] B. N. Miller and D. L. Ranum, *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle & Associates, 2013. [Online]. Available: https://runestone.academy/ns/books/published/pythonds/index.html