Modul Praktikum
STRUKTUR DATA

MODUL PRAKTIKUM
STRUKTUR DATA

Fadhel Akhmad Hizham, S.Kom., M.Kom.

WIDYA GAMA Press

MODUL PRAKTIKUM STRUKTUR DATA

Cetakan Pertama, April 2024
vi + 146 hlm; 15,5 x 23 cm
QR Code Book Number: 62-2797-1983-627

Penulis : Fadhel Akhmad Hizham, S.Kom., M.Kom.
Desain Sampul: WIGA Press
Tata Letak Isi: WIGA Press

Diterbitkan Oleh:

WIDYA GAMA PRESS
ANGGOTA ASOSIASI PENERBIT PERGURUAN TINGGI INDONESIA (APPTI)
Office:
Institut Teknologi dan Bisnis (ITB) Widya Gama
Jl. Gatot Subroto No. 4, Karangsari, Kec. Sukodono,
Kabupaten Lumajang, Jawa Timur.
Telp. (0334) 881924

PENGANTAR

"Modul Praktikum Struktur Data" adalah panduan praktis yang menyajikan konsep dasar dan penerapan struktur data dalam pemrograman komputer. Modul ini dirancang untuk memperkenalkan pembaca pada konsep-konsep dasar struktur data dan memberikan pemahaman yang mendalam tentang implementasi praktisnya melalui berbagai bab yang disusun secara sistematis.

"Modul Praktikum Struktur Data" ini berisi pembahan penting sebagai beriku:

BAB 1 - PENDAHULUAN STRUKTUR DATA
BAB 2 - ARRAY
BAB 3 - REKURSIF
BAB 4 - PENGURUTAN (SORTING) (Bagian 1)
BAB 5 - PENGURUTAN (SORTING) (Bagian 2)
BAB 6 - PENCARIAN (SEARCHING)
BAB 7 - STACK
BAB 8 - QUEUE
BAB 9 - TUGAS AKHIR STRUKTUR DATA

"Modul Praktikum Struktur Data" adalah sumber yang berguna bagi para mahasiswa untuk memahami dan menguasai konsep dasar struktur data dalam pemrograman komputer. Dengan penyajian yang sistematis dan berbagai contoh praktis, modul ini menjadi panduan yang sangat berguna dalam memahami dan mengimplementasikan struktur data dalam pemrograman.

DAFTAR ISI

PENGANTAR v
DAFTAR ISI vi
BAB 1 PENDAHULUAN STRUKTUR DATA 1
BAB 2 ARRAY 11
BAB 3 REKURSIF 29
BAB 4 PENGURUTAN (*SORTING*) (Bagian 1) 45
BAB 5 PENGURUTAN (*SORTING*) (Bagian 2) 63
BAB 6 PENCARIAN (*SEARCHING*) 77
BAB 7 STACK 85
BAB 8 QUEUE 98
BAB 9 TUGAS AKHIR STRUKTUR DATA 110
REFERENSI 145

BAB 1
PENDAHULUAN
STRUKTUR DATA

1.1. Pengenalan Struktur Data

Sebelum mempelajari struktur data dan jenisnya, maka perlu memahami terlebih dahulu mengenai struktur data itu sendiri. Struktur data didefinisikan sebagai bagian dari ilmu pemrograman dasar untuk menyusun sebuah struktur penyimpanan data yang digunakan saat program berjalan. **Struktur data** berbeda dengan **basis data**. Perbedaan antara struktur data dan basis data ditunjukkan pada Tabel 1.1.

Tabel 1.1. Perbedaan Struktur Data dan Basis Data

Basis Data	Struktur Data
Ilmu mengakses dan memelihara data yang disimpan di dalam *storage*	Berada pada tingkat pemrograman dimana merupakan tempat penyimpanan data yang digunakan oleh program terkait dengan alokasinya di memori.

Struktur data meliputi *array* (larik) dan *record* (rekaman) pada berkas beruntun (*sequential file*) yang merupakan bagian dari struktur data untuk penyimpanan data di memori sebagai larik atau penyimpanan *record* di dalam *file*. Dalam hal ini, konsep penyimpanan *record* di dalam *file* merupakan cikal bakal dari adanya aplikasi basis data.

Pembuatan struktur data diawali dari analisis perancangan data yang dimanipulasi di memori supaya program lebih efisien. Kemudian mengimplementasikan struktur data ke dalam pemrograman, dan selanjutnya menggunakan struktur data yang telah dibuat untuk memanipulasi data dalam memori pada sebuah program. Tabel 1.2 berikut merupakan ilustrasi dari langkah pembuatan struktur data.

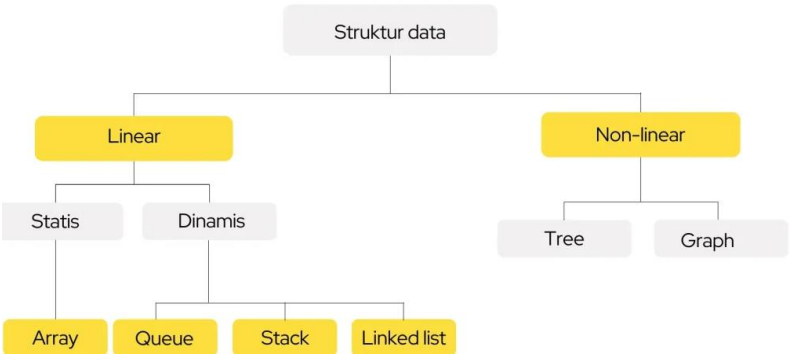
Tabel 1.2. Ilustrasi Pembuatan Struktur Data

<table>
<tr><th>Ilustrasi</th><th>Keterangan</th></tr>
<tr><td>
<table>
<tr><th>nama</th><th>nim</th><th>kelas</th><th>ipk</th></tr>
<tr><td>?</td><td>?</td><td>?</td><td>?</td></tr>
</table>
</td><td>Misalkan terdapat sebuah data mahasiswa yang terdiri dari:
Nama
NIM
Kelas
IPK
dan diperlukan untuk menyimpan data tersebut, maka akan dipersiapkan tempat untuk menyimpan data mahasiswa dan dibuatkan tipe data bentukan untuk menyimpan data mahasiswa.</td></tr>
<tr><td>
<table>
<tr><th>nama</th><th>nim</th><th>kelas</th><th>Ipk</th></tr>
<tr><td>Anto</td><td>230001</td><td>2ITA1</td><td>3.75</td></tr>
</table>
</td><td>Misalkan dari data mahasiswa diisikan dengan data seorang mahasiswa.</td></tr>
<tr><td>
<table>
<tr><th>nama</th><th>nim</th><th>kelas</th><th>ipk</th></tr>
<tr><td>Anto</td><td>230001</td><td>2ITA1</td><td>3.75</td></tr>
<tr><td>Budi</td><td>230002</td><td>2ITA1</td><td>4.00</td></tr>
<tr><td>Caca</td><td>230003</td><td>2ITA2</td><td>3.30</td></tr>
</table>
</td><td>Misalkan dari data mahasiswa digunakan untuk menampung beberapa data mahasiswa.</td></tr>
</table>

Berdasarkan ilustrasi tersebut maka dapat dilihat bahwa struktur data merupakan cara menyediakan tempat yang baik dan tersusun secara terstruktur agar data yang disimpan dapat dibaca dengan mudah.

1.2. Klasifikasi pada Struktur Data

Secara umum, struktur data dibagi menjadi dua bagian, yaitu struktur data statis dan struktur data dinamis. Pembagian klasifikasi struktur data ditunjukkan pada Gambar I.I.



Gambar I.I. Klasifikasi Struktur Data

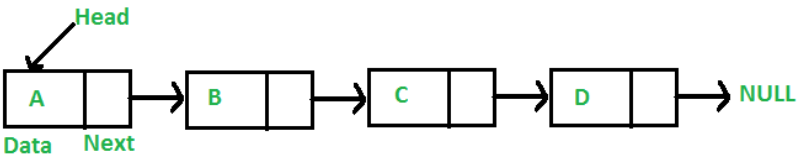
Struktur data statis berisi kumpulan data yang terdapat pada memori yang berukuran tetap. Pada struktur data jenis ini memori dialokasikan di waktu *compile* dan tidak dapat mengubah ukuran setelahnya. Kelemahan struktur daia jenis ini yaitu menghabiskan banyak memori dan tidak hemat ruang.

Struktur data dinamis adalah berlawanan sifat dari struktur data statis, dimana ukuran serta elemen yang disimpan pada struktur data ini dapat diubah ketika kode dijalankan. Akan tetapi, permasalahan overflow ataupun underflow akan terjadi pada tipe data ini, dikarenakan sifatnya yang dapat mengubah ukuran memori saat kode dijalankan.

1.3. Berbagai Contoh Permasalahan dalam Struktur Data

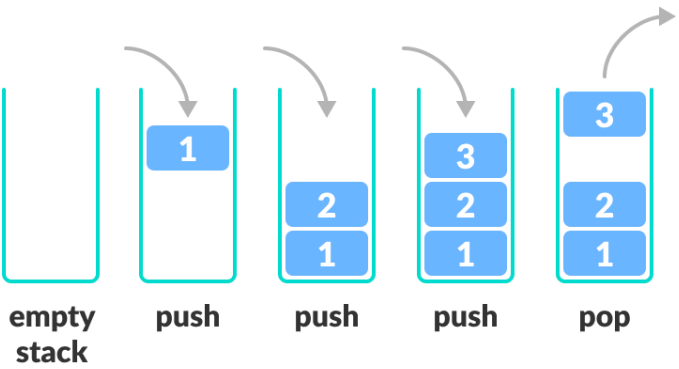
Struktur data perlu dipelajari untuk membuat sebuah program di dalam komputer. Berbagai contoh yang menerapkan struktur data diantaranya sebagai berikut:

- Program yang memproses banyak data dan jumlahnya yang berubah-ubah sehingga penyimpanan memori tidak dapat menggunakan *array* yang statis, maka dapat menggunakan struktur data *list* dengan representasi dinamis. *List* dapat digunakan untuk berbagai kebutuhan seperti *array* dua dimensi, tiga dimensi, dan multi dimensi. Ilustrasi struktur data list ditunjukkan pada Gambar 1.2.



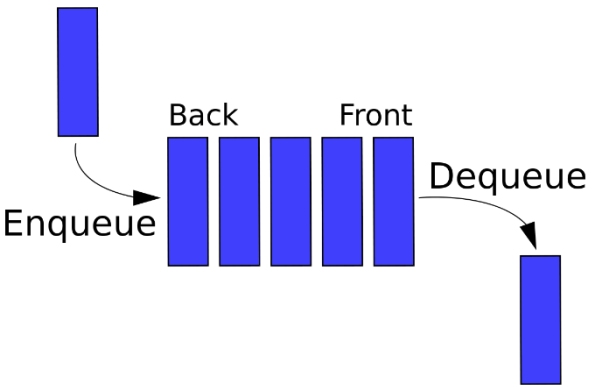
Gambar 1.2. Struktur Data *List*

- Program *compiler* memerlukan penyimpanan di memori yang disebut dengan struktur data *stack*. Ilustrasi mengenai struktur data stack ditunjukkan pada Gambar 1.3.



Gambar 1.3. Struktur Data *Stack*

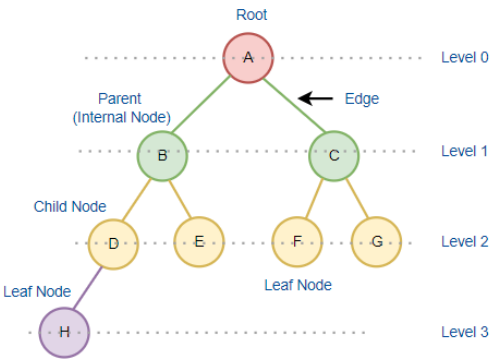
- Program yang memproses antrean (*queue*) seperti halnya dengan antrean di restoran, antrean transaksi di perbankan, dan lain sebagainya. Ilustrasi mengenai *queue* ditunjukkan pada Gambar 1.4.



Gambar 1.4. Struktor Data *Queue*

- Program yang memproses segala kemungkinan pilihan seperti contohnya sistem pendukung keputusan yang menggunakan struktur data pohon (*tree*). Ilustrasi

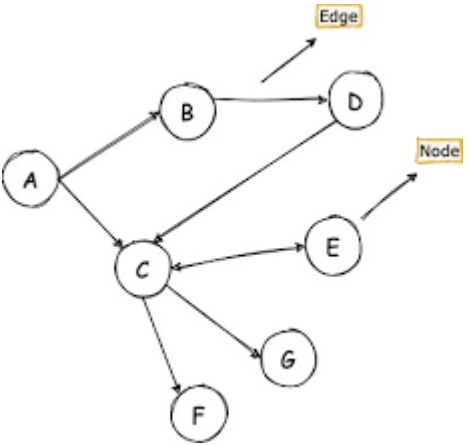
mengenai struktur data tree ditunjukkan pada Gambar 1.5.



Gambar 1.5. Struktur Data *Tree*

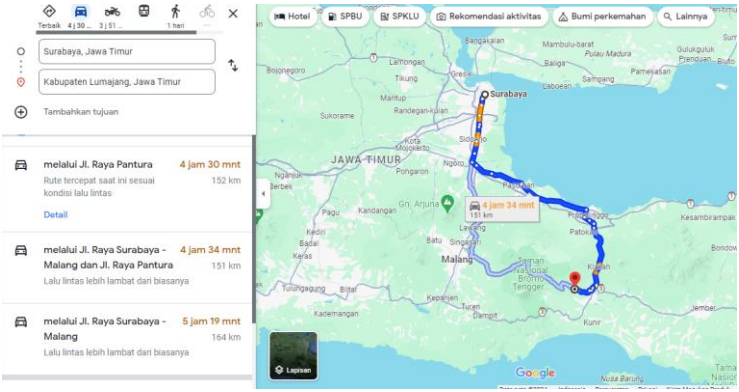
Struktur data *tree* banyak digunakan untuk pembuatan program *bot game*, seperti pada permainan catur, *tic tac toe*, *othello*, dan sebagainya. Struktur data tree juga dapat digunakan untuk membuat program sistem penunjang pengambilan keputusan, seperti sistem pakar dalam hal mendeteksi jenis penyakit suatu tanaman. Struktur data ini juga dapat digunakan sebagai penerapan klasifikasi dengan menggunakan metode C4.5 yang lebih dikenal dengan metode *decision tree*.

- Program yang memproses jalur-jalur peta atau denah yang menggunakan struktur data graf. Ilustrasi dari struktur data graf ditunjukkan pada Gambar 1.6.



Gambar 1.6. Struktur Data Graf

Struktur data graf biasa dijumpai pada proses pemrograman pemilihan rute terpendek (shortest path). Misalkan pada pemilihan rute untuk perjalanan dari Surabaya ke Lumajang, maka program akan menerapkan struktur data graf untuk menampilkan rute mana saja yang memungkinkan dengan jarak dan waktu tempuh yang bervariasi, dan tentunya bervariasi untuk jenis kendaraan yang digunakan (misal untuk kendaraan beroda 4 lebih direkomendasikan melewati jalan tol untuk mempersingkat waktu meskipun jarak tempuh terkadang lebih panjang daripada jalan biasa). Ilustrasi penerapan struktur graf tersebut biasa ditemukan pada aplikasi *Google Maps*, sesuai dengan Gambar 1.7 berikut.



Gambar 1.7. Ilustrasi Rute Surabaya-Lumajang pada *Google Maps*

1.4. Latihan Soal

1. Apakah struktur data menurut pengertian Anda dalam bidang pemrograman?

2. Menurut Anda, apakah hal yang mendasari tentang perbedaan struktur data dengan basis data?
3. Jelaskan menurut Anda perbedaan dari struktur data linear dengan yang non-linear.

4. Termasuk ke dalam apakah jenis struktur data berikut ini?
   a. *Array*
   b. *Stack*
   c. *Queue*
   d. *Tree*
   e. Graf

5. Digolongkan sebagai struktur data apakah untuk kasus dibawah ini?
   a. Tatanan piring di dalam rak piring
   b. Barisan pembeli obat-obatan di apotek
   c. Sistem pakar penentuan jenis penyakit pada hewan

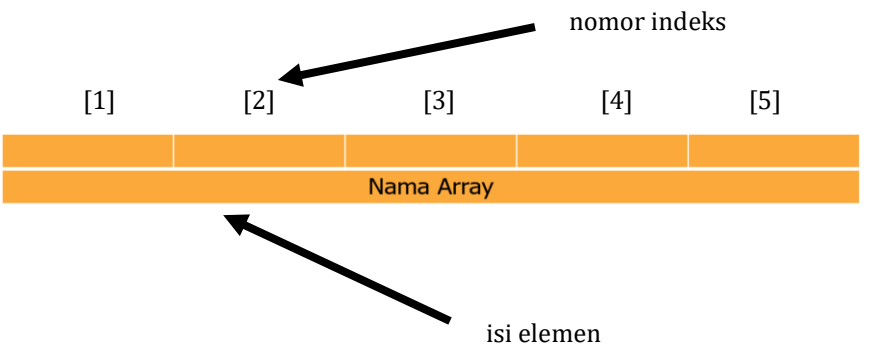
d. Klasifikasi penentuan kredit macet dengan metode C4.5
e. Pemilihan rute pengiriman paket yang tidak mengunjungi kelurahan yang sama dua kali

BAB 2
ARRAY

Pembahasan materi tentang *array* sudah dibahas pada Algoritma Pemrograman, akan tetapi pada Struktur Data, *array* menjadi suatu hal yang sangat penting. *Array* juga merupakan hal yang mendasar dari struktur data, karena untuk menggunakan komponen struktur data seperti *stack*, *queue*, *tree*, graf, hampir seluruh program menggunakan *array* sebagai tempat penampungan data.

2.1. *Array* / Larik

Secara garis besar, array dapat didefinisikan sebagai tabel di mana tabel merupakan sekumpulan elemen yang dapat diakses untuk setiap elemennya. Array menjadi hal yang penting dalam penyimpanan data, karena apabila array tidak ada, maka semisal memerlukan 5 tempat untuk 5 nilai, maka secara tidak langsung harus membuat 5 macam variabel yang berbeda. Hal tersebut dapat mempersulit pengaksesan data. Ilustrasi mengenai array ditunjukkan pada Gambar 2.1.

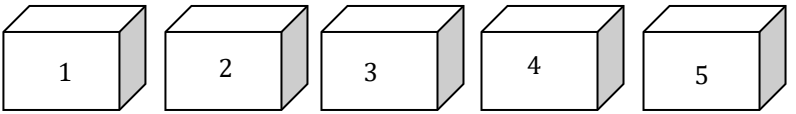


Gambar 2.1. Ilustrasi *Array*

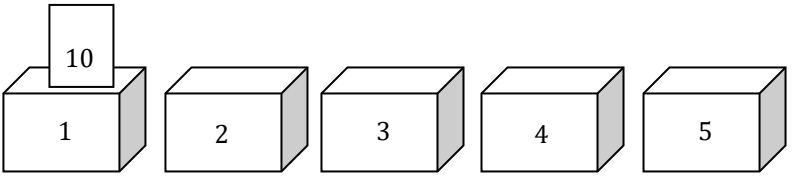
Dalam logika pemikiran manusia, array dianalogikan sebagai beberapa kotak penampungan (variabel) yang diikat menjadi satu sebagai kesatuan, dimana tiap kotak mempunyai

nomor yang didefinisikan sebagai nomor indeks. Urutan analogi dari array yaitu sebagai berikut:

- Misalkan ada 5 buah kotak kosong dalam satu ikatan. Kelima kotak kosong itu berisikan nilai yang sesuai dengan tipe kotak itu sendiri (analogi sebagai tipe data).



  Bahasa algoritmiknya yaitu: ikatan_integer : array [1..5] of **integer**
- Misalkan kotak bernomor 1 itu diisi dengan memasukkan nilai dari kotak tersebut:



  Maka dapat disimpulkan bawa kotak ke-1 berisi angka 10. Nomor kotak tersebut sering disebut dengan **indeks**.
  Bahasa algoritmiknya: ikatan_integer1 <- 10

Secara umum, array dapat dibagi menjadi sebagai berikut:

- *Array* satu dimensi
- *Array* dua atau lebih dimensi (untuk hal ini disebut *array* multi dimensi)

2.2. *Array* Satu Dimensi

Sesuai dengan namanya, *array* satu dimensi yaitu *array* yang terdiri dari beberapa baris dan 1 kolom atau beberapa kolom dan 1 baris. Intinya dimensi dari *array* satu dimensi

yaitu (1 x n) atau (n x 1). Misalkan ingin membuat *array* berupa tinggi badan 5 orang mahasiswa, maka berdasarkan subbab 2.1, maka langkah-langkah analoginya apabila ditulis dalam kode pemrograman *Java* yaitu sebagai berikut:

Langkah 1, deklarasi *array*:
Bahasa algoritmik: tinggi_badan : array [1..5] of **integer**

Bahasa pemrograman *Java*:

```
1  int[] tinggi_badan;
2  tinggi_badan = new int[5];
```

Atau dapat ditulis:

```
1  int tinggi_badan[];
2  tinggi_badan = new int[5];
```

Atau dapat ditulis:

```
1  int[] tinggi_badan = new int[5];
```

Langkah 2, pengaksesan *array*:
Misalkan tinggi mahasiswa 1 170 cm, mahasiswa 2 160 cm, dan mahasiswa 3 177 cm, maka dapat ditulis sebagai berikut:

Bahasa algoritmik:
tinggi_badan1 <- 170
tinggi_badan2 <- 160
tinggi_badan3 <- 177

Bahasa pemrograman *Java*:

```
1  tinggi_badan[0] = 170;
2  tinggi_badan[1] = 160;
3  tinggi_badan[2] = 177;
```

Perlu diketahui, bahwa dalam bahasa pemrograman *Java*, indeks *array* dimulai dari 0 sampai dengan jumlah indeks – 1. Untuk menampilkan hasil pengaksesan *array* di output, maka penulisannya yaitu:

```
1  System.out.println(tinggi_badan[0]);
2  System.out.println(tinggi_badan[1]);
3  System.out.println(tinggi_badan[2]);
```

Maka hasil outputnya yaitu:

```
170
160
177
```

Namun, apabila pada bagian System.out.println diubah menjadi (tinggi_badan[3]), maka hasil outputnya yaitu **0**, dikarenakan indeks ke-3 tidak ada isinya. Untuk tipe data lain, hasil output untuk menampilkan array yang tidak ada isinya yaitu sebagai berikut:

- Tipe data numerik (int, byte, short, long): 0
- Tipe data riil (double, float): 0.0
- Tipe data char: (tidak ada isinya)
- Tipe data string: null

Jika pada bagian System.out.println diubah menjadi (tinggi_badan[5]), maka hasil outputnya yaitu galat **ArrayIndexOutOfBoundsException.** Arti dari galat tersebut yaitu program tidak dapat menampilkan indeks *array* yang melebihi daya tampung yang ditentukan sebelumnya. Pada bagian ini, daya tampung *array* yang ditentukan yaitu sebanyak 5 elemen, sementara tinggi_badan[5] itu merupakan indeks ke-6 (**INGAT**: Bahasa pemrograman *Java* penomoran indeks dimulai dari 0), sehingga melebihi daya tampung yang ditentukan dan terjadilah galat tersebut.

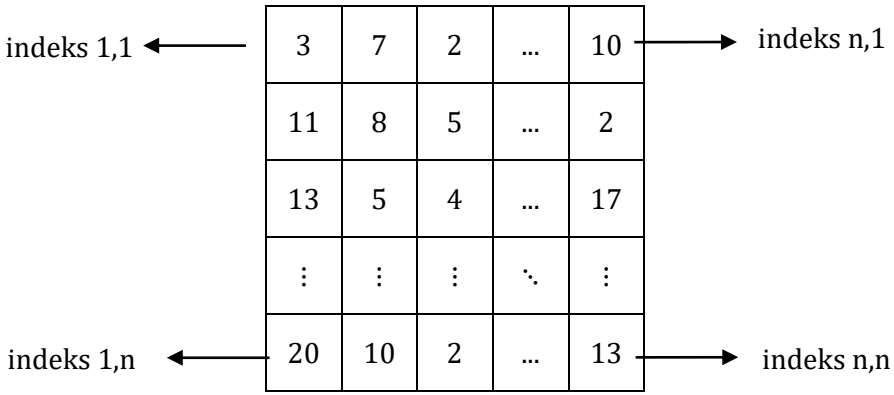
Array juga dapat dideklarasikan sekaligus diinisialisasi dengan kode penulisan sebagai berikut:

```
1  int tinggi_badan[] = {154, 160, 176, 163, 171, 170};
```

Dari kode diatas, maka dapat diketahui bahwa array tinggi_badan memiliki 6 buah indeks. Maka apabila ingin menampilkan tinggi badan pada indeks ke-3, maka outputnya yaitu 163, karena aturan indeks ke-0 tetap berlaku.

2.3. *Array* Dua Dimensi

Array dua dimensi yaitu *array* yang memiliki dua atau lebih kolom dengan beberapa baris, atau dua atau lebih baris dengan beberapa kolom. Ilustrasinya sebagai sebuah tabel yang berukuran m x n dimana m = jumlah baris dan n = jumlah kolom. Gambar 2.2 dibawah ini merupakan ilustrasi dari *array* dua dimensi.



Gambar 2.2. ilustrasi *Array* Dua Dimensi

Proses penelusuran pada matriks *array* atau yang disebut sebagai scanning dilakukan dari tiap baris dan tiap baris ditelusuri tiap kolomnya. Gambar 2.3 merupakan matriks *array* ukuran 3 x 4.

3	7	2	13
11	8	5	14
13	5	4	10

Gambar 2.3 Matriks ukuran 3 x 4

Berdasarkan Gambar 2.3, maka penelusuran dari matriks tersebut akan mendapatkan urutan jalan 3 – 7 – 2 – 13 – 11 – 8 – 5 – 14 – 13 – 5 – 4 – 10, untuk setiap baris ditelusuri tiap kolomnya. Berikut merupakan algoritma untuk membuat matriks ukuran 3 x 4 (12 elemen), beserta mengisi *array* dan menampilkan *array* dengan bahasa algoritmik beserta kode program *Java*nya.

Langkah 1, deklarasi *array*:
Bahasa algoritmik:
matriks : array [1..3, 1..4] of **integer**

Bahasa pemrograman *Java*:

```
1  import java.util.Scanner;
2
3  public class App{
4  public static void main(String[] args) throws
5  Exeption {
6
      int matriks [][] = new int[3][4];
```

Langkah 2, membuat variabel baris dan kolom:
Bahasa algoritmik:
baris : **integer**
kolom : **integer**

Bahasa pemrograman *Java*:

```
1  int baris;
2  int kolom;
```

Langkah 3, membuat perulangan untuk mengisi masing-masing baris dan kolom:

Bahasa algoritmik:
for baris <- 1 to 3 do
    for kolom <- 1 to 4 do

        input (matriks$_{baris, kolom}$)

    (end for)

(end for)

Bahasa pemrograman *Java*:

```
1  Scanner sc = new Scanner (System.in);
2
3    for (baris = 0, baris = matriks.length; baris++ {
4      for (kolom = 0, kolom = matriks[baris].length; baris++ {
5        System.out.print("masukkan angka:");
6        matriks[baris][kolom] = sc.nextInt();
7      }
8    }
```

Langkah 4, membuat perulangan untuk menampilkan *array*:

Bahasa algoritmik:

for baris <- 1 to 3 do

    for kolom <- 1 to 4 do

        output ($matriks_{baris, kolom}$)

    (end for)

    output ("\n")

(end for)

Bahasa pemrograman *Java*:

```
1    for (baris = 0, baris = matriks.length; baris++ {
2      for (kolom = 0, kolom = matriks[baris].length; baris++ {
3        System.out.print(matriks[baris][kolom] + " ");
4      }
5      System.out.println ("");
6    }
7  }
8  }
```

Maka output yang dihasilkan, apabila pengisian array sesuai dengan matriks pada Gambar 2.3 yaitu:

```
3  7  2  13
11  8  5  14
13  5  4  10
```

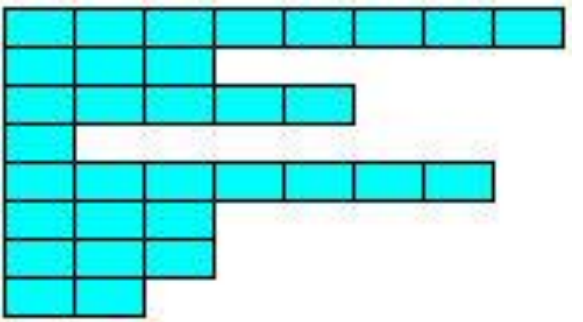
Kode program di atas tidak hanya untuk *array* dua dimensi, bisa juga untuk *array* tiga dimensi, empat dimensi, dan seterusnya, sehingga disebut sebagai *array* multi dimensi. Apabila ingin membuat *array* tiga dimensi, maka deklarasinya menjadi sebagai berikut:

```
1  import java.util.Scanner;
2
3  public class App{
4  public static void main(String[] args) throws Exeption {
5
6     int matriks [][][] = new int[3][4][5];
```

Contoh di atas apabila ingin membuat matriks 3 x 4 sebanyak 5 lapisan (*layer*). Maka jumlah elemen yang dapat diisi yaitu 3 x 4 x 5 = 60 elemen.

2.4. *Jagged Array*

Jagged array yaitu *array* dua dimensi yang memiliki jumlah kolom yang tidak sama untuk tiap barisnya ataupun jumlah baris yang tidak sama untuk tiap kolomnya. Ilustrasi *jagged array* ditunjukkan pada Gambar 2.4.



Gambar 2.4. *Jagged Array*

Jika dilihat dari Gambar 2.4, maka baris ke-1 memiliki 8 kolom, baris ke-2 memiliki 3 kolom, baris ke-3 memiliki 5 kolom, demikian seterusnya dimana jumlah kolom berbeda-beda dari tiap barisnya. Untuk membuat *jagged array*, sesuai Gambar 2.4, berikut langkah-langkahnya.

Langkah 1, membuat *array* dan jumlah barisnya:
Bahasa pemrograman *Java*:

```
1  int jagged [][] = new int[8][];
```

Langkah 2, menentukan jumlah kolom untuk tiap barisnya:
Bahasa pemrograman *Java*:

```
1  jagged[0] = new int[8];
2  jagged[1] = new int[3];
3  jagged[2] = new int[5];
4  jagged[3] = new int[1];
5  jagged[4] = new int[7];
6  jagged[5] = new int[3];
7  jagged[6] = new int[3];
8  jagged[7] = new int[2];
```

2.5. Contoh Kasus *Array*

Berikut ini beberapa contoh kasus dari *array* yang dapat Anda gunakan untuk latihan praktikum Struktur Data.

CONTOH 1: Buatlah sebuah program untuk memasukkan sembarang bilangan, lalu tampilkan angka yang hanya memiliki puluhan 4 atau 8.

Langkah 1, buatlah array beserta impor library Scanner untuk memasukkan bilangan di bagian output. Kode programnya yaitu sebagai berikut:

```
1  import java.util.Scanner;
2
3  public class App {
4      public static void main(String[] args) throws Exception {
5          Scanner sc = new Scanner(System.in);
6          System.out.print("Masukkan banyak bilangan: ");
7          int bil = sc.nextInt();
8          int [] bilangan = new int [bil];
```

Pada langkah tersebut, *array* satu dimensi dibuat untuk menampung beberapa bilangan dengan banyak bilangan sesuai yang ditentukan oleh *user*.

Langkah 2, buatlah perulangan untuk mengisi *array* dengan jumlah elemen sebanyak yang telah ditentukan *user*. Kode programnya yaitu sebagai berikut:

```
1   for (int a = 0; a < bilangan.length; a++) {
2      System.out.print("Masukkan bilangan ke-" + (a+1) + ": ");
3          bilangan[a] = sc.nextInt();
4   }
```

Pada langkah tersebut, disebutkan pernyataan bilangan.length pada perulangan untuk menandakan bahwa array akan diisi dari indeks ke-0 sampai sebanyak indeks yang telah ditentukan.

Langkah 3, buatlah perulangan beserta kondisi untuk menampilkan bilangan yang memiliki puluhan 4 atau 8. Kode programnya yaitu sebagai berikut:

```
1  System.out.println("Bilangan yang berpuluhan 4 atau 8 yaitu:");
2    for (int a = 0; a < bilangan.length; a++) {
3      if ((bilangan[a] % 100 >= 40 && bilangan[a] % 100 <= 49) ||
   (bilangan[a] % 100 >= 80 && bilangan[a] % 100 <= 89)) {
4        System.out.println("Bilangan ke-" + (a+1) + ", yakni " +
   bilangan[a]);
5        }
6      }
7    }
8  }
```

Dari langkah di atas, baris ke-3 memberikan persyaratan untuk menampilkan bilangan yang hanya mengandung puluhan 4 atau 8, dengan cara menggunakan modulo 100 pada nilai *array* bilangan[a].

Setelah itu, penulisan kode program selesai dan mulai menampilkan hasil *output*nya. Sebagai contoh, kita akan memasukkan 4 buah bilangan acak sebagai berikut:

```
Masukkan banyak bilangan: 4
Masukkan bilangan ke-1: 355
Masukkan bilangan ke-2: 449
Masukkan bilangan ke-3: 586
Masukkan bilangan ke-4: 784
```

Maka setelah memasukkan keempat bilangan itu, hasil outputnya yaitu:

```
Bilangan yang berpuluhan 4 atau 8 yaitu:
Bilangan ke-2, yakni 449
Bilangan ke-3, yakni 586
Bilangan ke-4, yakni 784
```

CONTOH 2: Buatlah sebuah program untuk memasukkan beberapa nilai peserta babak penyisihan olimpiade sains. Kemudian, tampilkan peserta yang lolos babak penyisihan dengan ketentuan nilai olimpiade sains minimal 150 poin. Format *output* terdiri atas nomor peserta, nama, dan nilai.

Langkah 1, dari soal ini, maka perlu dibuat array dua dimensi, karena akan memasukkan beberapa nilai peserta, dan untuk masing-masing peserta akan dimasukkan sesuai dengan format *output*, yakni nomor peserta, nama, dan nilai. Apabila digambarkan berupa tabel, maka data menjadi sebagai berikut:

Nomor peserta	Nama	Nilai
12303-001	Ahmad	182
12303-002	Koko	144
12303-003	Dadang	190
12303-004	Panca	140
12303-005	Eko	160

Sebelum membuat array dua dimensi, selidiki dulu tipe datanya. Dari soal di atas, maka atribut nomor peserta dan nama termasuk tipe data String, sementara nilai termasuk tipe data int. Maka kode program untuk membuat array yaitu sebagai berikut:

```
1   import java.util.*;
2
3   public class App {
4       public static void main(String[] args) throws Exception {
5           Scanner sc = new Scanner(System.in);
6           System.out.print("Masukkan banyak peserta: ");
7           int peserta = sc.nextInt();
8           String [][] data = new String[peserta][2];
9           int [] nilai = new int [peserta];
10
```

Pada baris ke-9 dan 10 ada 2 *array* yang dibuat karena keduanya memiliki tipe data yang berbeda. *Array* data untuk tipe data string dengan dua buah kolom yaitu nomor peserta dan nama, sementara nilai untuk tipe data int dengan satu kolom (*array* 1 dimensi).

Langkah 2, buatlah perulangan untuk mengisi *array* dengan jumlah peserta sebanyak yang telah ditentukan *user*. Kode programnya yaitu sebagai berikut:

```
1   for (int a = 0; a < peserta; a++) {
2     System.out.print("Nomor peserta ke-" + (a+1) + ":
3   ");
4       data[a][0] = sc.next();
5       System.out.print("Nama: ");
6       data[a][1] = sc.next();
7       System.out.print("Nilai: ");
8       nilai[a] = sc.nextInt();
      }
```

Langkah 3, buatlah perulangan beserta kondisi untuk menampilkan Peserta yang lolos babak penyisihan, dengan nilai minimal 150. Kode programnya yaitu sebagai berikut:

```
System.out.println();
System.out.println("Peserta yang lulus babak penyisihan yaitu:");
  for (int a = 0; a < peserta; a++) {
    if (nilai[a] >= 150) {
      System.out.println("Nomor Peserta: " + data[a][0]);
      System.out.println("Nama: " + data[a][1]);
      System.out.println("Nilai: " + nilai[a]);
      System.out.println("");
    }
  }
 }
}
```

Setelah itu, penulisan kode program selesai dan mulai menampilkan hasil *output*nya. Sebagai contoh, kita akan memasukkan 5 peserta olimpiade sesuai dengan contoh tabel pada langkah 1 sebagai berikut:

```
Masukkan banyak peserta: 5
Nomor peserta ke-1: 12303-001
Nama: Ahmad
Nilai: 182
Nomor peserta ke-2: 12303-002
Nama: Koko
Nilai: 144
Nomor peserta ke-3: 12303-003
Nama: Dadang
Nilai: 190
Nomor peserta ke-4: 12303-004
Nama: Panca
Nilai: 140
Nomor peserta ke-5: 12303-005
Nama: Eko
Nilai: 160
```

Maka setelah memasukkan kelima peserta itu, hasil *output*nya yaitu:

```
Peserta yang lulus babak penyisihan yaitu:
Nomor Peserta: 12303-001
```

```
Nama: Ahmad
Nilai: 182

Nomor Peserta: 12303-003
Nama: Dadang
Nilai: 190

Nomor Peserta: 12303-005
Nama: Eko
Nilai: 160
```

2.6. Latihan Soal

1. Menurut Anda, mengapa *array* sangat penting dalam pemrograman, baik dalam bidang algoritma pemrograman maupun struktur data?
2. Jelaskan secara singkat perbedaan antara *array* satu dimensi dengan *array* multi dimensi.
3. Berapa elemen yang dapat dimasukkan untuk array berikut?
   a. int array[][] = new int [2][3]
   b. int array[][] = new int [7][5]
   c. int array[][][] = new int [5][5][5]
   d. int array[][][] = new int [1][1][1]
4. Buatlah kode program untuk memasukkan 21 nama kecamatan di Kabupaten Lumajang, lalu tampilkan kecamatan yang terdiri dari 7 sampai 10 huruf.
5. Buatlah kode program untuk memasukkan 5 peserta olimpiade Matematika tingkat kabupaten berikut, dengan ketentuan:
   a. Jumlah soal sebanyak 50 soal
   b. Kosong = 50 – Benar – Salah
   c. Nilai = Benar x 4 – Salah x 1

Nomor peserta	Nama	Benar	Salah	Kosong	Nilai
33501-001	Ruli	35	4	11	136
33501-002	Cahya	27	10	13	98
33501-003	Fani	40	10	0	150
33501-004	Jack	43	0	7	172
33501-005	Lala	30	5	15	115

   Tampilkan peserta yang lolos ke tingkat provinsi dengan ketentuan nilai minimal 125 poin.

BAB 3
REKURSIF

3.1. Rekursif

Rekursif dapat didefinisikan sebagai proses untuk memanggil diri sendiri yang dilakukan oleh fungsi atau prosedur. Konsep rekursif hampir sama dengan *looping* / perulangan, dimana akan berjalan sampai dengan kondisi penghentian terpenuhi. Dalam sebuah rekursif perlu adanya blok-blok kode berikut:

- **Basis**, yaitu kode yang berperan sebagai titik henti dari proses rekursif karena rekursif bersifat berputar terus memanggil diri sendiri sampai mencapai titik hentinya.
- **Rekursif**, yaitu kode dimana sebuah blok program (prosedur, fungsi, atau metode) akan memanggil dirinya sendiri.

3.2. Proses Rekursif

Tahapan pembuatan kode program rekursif dilakukan dengan proses sebagai berikut:

Langkah 1, mendeklarasikan prosedur yang mengandung rekursif dengan tipe bilangan bulat.

Bahasa Algoritmik:

procedure write (input : n : integer)

Kode pemrograman Java:

```
1  public class App {
2
3      public static void write(int n) {
```

Langkah 2, mendeklarasikan variabel bilangan bulat yang mana nilainya akan dimanipulasi sebagai basis yang nilainya akan diisi dengan masukan n.

Bahasa Algoritmik:
counter : integer
counter <- n

Kode pemrograman Java:

```
1       int counter = n;
```

Langkah 3, membuat kondisi basis, dengan kondisi apabila counter lebih dari 0, maka proses di percabangan if akan dikerjakan terus menerus hingga nilai variabel counter adalah 0.

Bahasa Algoritmik:
if counter > 0 then

Kode pemrograman Java:

```
1       if (counter > 0) {
```

Langkah 4, melakukan proses rekursif dan inisialisasinya dimana prosedur write akan dipanggil kembali dengan nilai masukan = nilai sebelumnya - 1.

Bahasa Algoritmik:
output (“Saatnya hitung mundur dari “, counter)
counter <- counter – 1
write(counter)

{*end if*}

{*end procedure*}

Kode pemrograman Java:

```
    System.out.println("Saatnya hitung mundur dari " + counter);
    counter = counter - 1;
    write(counter);
   }
}
```

Langkah 5, memanggil prosedur write pada algoritma utama (*public static void main*).

Bahasa Algoritmik:
{*main algorithm*}
  write(20)
{*end main algorithm*}

Kode pemrograman Java:

```
public static void main(String[] args) throws Exception {
   write(20);
   }
}
```

Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya, yaitu sebagai berikut:

```
Saatnya hitung mundur dari 20
Saatnya hitung mundur dari 19
Saatnya hitung mundur dari 18
Saatnya hitung mundur dari 17
Saatnya hitung mundur dari 16
Saatnya hitung mundur dari 15
Saatnya hitung mundur dari 14
Saatnya hitung mundur dari 13
Saatnya hitung mundur dari 12
Saatnya hitung mundur dari 11
Saatnya hitung mundur dari 10
Saatnya hitung mundur dari 9
```

```
Saatnya hitung mundur dari 8
Saatnya hitung mundur dari 7
Saatnya hitung mundur dari 6
Saatnya hitung mundur dari 5
Saatnya hitung mundur dari 4
Saatnya hitung mundur dari 3
Saatnya hitung mundur dari 2
Saatnya hitung mundur dari 1
```

Proses rekursif dari contoh tersebut dapat dituliskan sebagai berikut:

- **Proses rekursif pertama**
  Variabel counter bernilai 20, jalannya program masuk ke blok percabangan if karena nilai counter 20 > 0. Proses percabangan if dijalankan dengan cara menuliskan nilai variabel counter ke layar, kemudian nilai counter – 1, dan prosedur write dipanggil kembali dengan nilai 19.

- **Proses rekursif kedua**
  Variabel counter bernilai 19, jalannya program masuk ke blok percabangan if karena nilai counter 19 > 0. Proses percabangan if dijalankan dengan cara menuliskan nilai variabel counter ke layar, kemudian nilai counter – 1, dan prosedur write dipanggil kembali dengan nilai 18.

  (demikian seterusnya)
- **Proses rekursif ke-20**
  Variabel counter bernilai 1, jalannya program masuk ke blok percabangan if karena nilai counter 1 > 0. Proses percabangan if dijalankan dengan cara menuliskan nilai variabel counter ke layar, kemudian nilai counter – 1, dan prosedur write dipanggil kembali dengan nilai 0.

- **Proses rekursif ke-21**
  Variabel counter bernilai 0, jalannya program tidak masuk ke blok percabangan if karena nilai counter = 0, sehingga prosedur write tidak dipanggil kembali dan rekursif selesai.

  Proses rekursif sebenarnya hampir mirip dengan proses *looping* / perulangan, hanya saja pada rekursif, program akan berjalan semakin lambat dikarenakan sifat rekursif yaitu memanggil dirinya sendiri di dalam blok prosedur, fungsi, ataupun metode. Sementara pada *looping*, proses cukup pada blok *void main* saja. Sebagai perbandingan, berikut kode program untuk *looping* dari contoh rekursif di atas.

```
1  public static void main(String[] args) throws Exception {
2     for (int n = 1; n <= 20; n++) {
3        System.out.println("Saatnya hitung mundur dari" + n);
4     }
5  }
```

  Hasilnya akan sama dengan *output* pada proses rekursif. Oleh karena itu, jika memungkinkan sebaiknya menggunakan proses *looping* untuk efisiensi *running* pada program, namun jika tidak memungkinkan menggunakan *looping*, maka dapat menggunakan rekursif.

3.3. Contoh Rekursif

Berikut ini beberapa contoh dari penerapan rekursif yang dapat Anda gunakan untuk latihan praktikum Struktur Data.

CONTOH 1: Buatlah program rekursif untuk menghitung nilai faktorial.

Faktorial yaitu perhitungan yang berupa perkalian bilangan bulat positif secara berurutan. Misal 5 faktorial (atau ditulis 5!) = 1 x 2 x 3 x 4 x 5 = 5 x 4 x 3 x 2 x 1 = 120. Cara membuat program faktorial dapat dilakukan dengan perulangan (terletak pada Bab Studi Kasus room 18 pada buku Praktikum Algoritma Pemrograman), dengan penulisan perulangan sebagai berikut (menggunakan perulangan for):

```
1   import java.utll.Scanner;
2
3   public class App {
4
5   public static void main(String[] args) throws Exception {
6    Scanner sc = new Scanner(System.in);
7    int hasil = 1;
8    System.out.println("Berapa faktorial?");
9    System.out.print("Input = ");
10   int bilangan = sc.nextInt();
11
12   if (bilangan == 0 || bilangan == 1) {
13     System.out.println(bilangan + "! = 1");
14   } else {
15     System.out.print(bilangan + "! = ");
16     for (int i = bilangan; i >= 2; i--) {
17       hasil *= i;
18       System.out.print(i + " x ");
19       }
20     System.out.print("1 = " + hasil);
21     }
22   }
23  }
```

Sementara pembuatan program dengan rekursif yaitu sebagai berikut:

Langkah 1, mendeklarasikan fungsi faktorial dengan masukan bilangan bulat yang dihitung faktorialnya dan variabel hasil yang menyimpan hasil perhitungan. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static int faktorial(int n) {
```

Langkah 2, membuat kondisi basis apabila nilai masukan = 0 atau 1, maka akan mengembalikan nilai 1. Kode programnya yaitu sebagai berikut:

```
1    if (n == 0 || n == 1) {
2      return 1;
3    }
```

Langkah 3, membuat blok rekursif untuk nilai n lebih dari 1. Berlaku perhitungan faktorial yaitu faktorial (n) = n * faktorial (n-1). Kode programnya yaitu sebagai berikut:

```
1    else {
2      return (n * faktorial(n-1));
3    }
4  }
```

Langkah 4, memanggil fungsi rekursif pada algoritma utama:

```
1  public static void main(String[] args) throws Ezception {
2    int n = 10;
3    System.out.println(n + "! = " + faktorial(n));
4    }
5  }
```

Setelah itu, penulisan kode program selesai dan mulai menampilkan hasil *output*nya. Dari kode pada algoritma utama (*void main*) memanggil nilai n = 10, sehingga hasilnya yaitu sebagai berikut:

```
10! = 3628800
```

CONTOH 2: Buatlah program rekursif untuk menghitung pangkat bilangan.

Pangkat bilangan yaitu sebuah konsep matematis yaitu sebuah bilangan dikalikan oleh bilangan itu sendiri sebanyak pangkatnya. Cara membuat program pangkat bilangan dapat dilakukan dengan perulangan (terletak pada Bab Studi Kasus room 27 pada buku Praktikum Algoritma Pemrograman), dengan penulisan perulangan sebagai berikut (menggunakan perulangan for):

```
1  import java.utll.Scanner;
2
3  public class App {
4
5  public static void main(String[] args) throws Exception {
6   Scanner sc = new Scanner(System.in);
7   System.out.print("Masukkan bilangan? ");
8   int bilangan = sc.nextInt();
9   System.out.print("Pangkat berapa? ");
10  int pangkat = sc.nextInt();
11  int hasil = 1;
12
13  if (pangkat == 0) {
14    System.out.print(bilangan + " pangkat " + pangkat + " = 1");
15    } else {
16    System.out.print(bilangan + " pangkat " + pangkat + " = ");
17      for (int a = 1; a <= pangkat-1; a++) {
18        System.out.print(bilangan + " x ");
19        hasil *= bilangan;
20      }
21      for (int a = pangkat; a <= pangkat; a++) {
22        System.out.print(bilangan + " = ");
23        hasil *= bilangan;
24      }
25      System.out.print(hasil);
26    }
27  }
28 }
```

Sementara pembuatan program dengan rekursif yaitu sebagai berikut:

Langkah 1, membuat fungsi yang terdiri atas dua buah bilangan bulat (integer) berupa bilangan dan pangkatnya. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static int pangkat(int a, int b) {
```

Langkah 2, membuat basis rekursif di dalam fungsi, yaitu apabila semua bilangan dipangkatkan dengan 0, maka hasilnya adalah 1. Kode programnya yaitu sebagai berikut:

```
  if (b == 0) {
    return 1;
  }
```

Langkah 3, membuat proses rekursif untuk proses mengalikan bilangan yang dipangkatkan sebanyak pangkatnya. Berlaku perhitungan perpangkatan yaitu pangkat(a, b) = a * pangkat(a, b – 1) sebagai persamaan pada proses rekursif. Kode programnya yaitu sebagai berikut:

```
  else {
    return (a * pangkat(a, b-1));
  }
}
```

Langkah 4, memanggil fungsi rekursif pada algoritma utama:

```
public static void main(String[] args) throws Ezception {
  int a = 3;
  int b = 4;
  int hasil = 1
  hasil = pangkat(a, b);
  System.out.println(a + " pangkat " + b + " = " + pangkat(a, b));
  }
}
```

Setelah itu, penulisan kode program selesai dan mulai menampilkan hasil *output*nya. Dari kode pada algoritma utama (*void main*) memanggil nilai a = 3 dan b = 4, artinya program akan menghitung 3 pangkat 4, sehingga hasilnya yaitu sebagai berikut:

```
3 pangkat 4 = 81
```

CONTOH 3: Buatlah program rekursif untuk deret Fibonacci.

Deret Fibonacci yaitu sebuah deret yang mana nilai suatu bilangan adalah penjumlahan dua bilangan sebelumnya. Deret Fibonacci pada umumnya yaitu:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Dari deret tersebut, diketahui dua bilangan awal yaitu 0 dan 1, kemudian bilangan ketiga yaitu hasil penjumlahan bilangan pertama dan kedua, yaitu 0 + 1 = 1. Deret Fibonacci juga dapat dimodifikasi dengan menentukan sendiri bilangan pertama dan keduanya. Misalkan angka pertama = 2 dan angka kedua = 6, maka hasil deretnya menjadi sebagai berikut:

2, 6, 8, 14, 22, 36, 58, 94, 152, 246, ...

Cara membuat program pangkat bilangan dapat dilakukan dengan perulangan (terletak pada Bab Studi Kasus room 26 pada buku Praktikum Algoritma Pemrograman), dengan penulisan perulangan sebagai berikut (menggunakan perulangan for):

```
1   import java.utll.Scanner;
2
3   public class App {
4
5   public static void main(String[] args) throws Exception {
6    Scanner sc = new Scanner(System.in);
7      System.out.print("Berapa bilangan? ");
8      int bilangan = sc.nextInt();
9      System.out.print("Angka awal? ");
10     int angka1 = sc.nextInt();
11     System.out.print("Angka kedua? ");
12     int angka2 = sc.nextInt();
13
14     System.out.print("Deret bilangan: ");
15     for (int a = 1; a <= bilangan; a++) {
16       System.out.print(angka1 + " ");
17       int jumlah = angka1 + angka2;
18       angka1 = angka2;
19       angka2 = jumlah;
20     }
21   }
22  }
```

Sementara pembuatan program dengan rekursif yaitu sebagai berikut:

Langkah 1, membuat prosedur yang terdiri atas masukan sebuah array, integer a sebagai penentu angka pertama, integer b sebagai penentu angka kedua, integer n sebagai posisi Fibonacci yang ingin ditampilkan, dan integer i sebagai posisi yang sedang dihitung di dalam array. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static void fibo(int [] arr, int a, int
   b, int n, int i) {
```

Langkah 2, proses rekursif pertama yaitu membuat kondisi dimana indeks *array* dengan indeks 0 maka diisi dengan nilai a (bilangan fibonacci pertama), lalu dilanjutkan dengan proses rekursif untuk mengisi *array* indeks berikutnya (i + 1). Kode programnya yaitu sebagai berikut:

```
1    if (i == 0) {
2      arr[i] = a;
3      fibo(arr, a, b, n, i+1);
4    }
```

Langkah 3, proses rekursif kedua yaitu membuat kondisi dimana indeks *array* dengan indeks 1 maka diisi dengan nilai b (bilangan fibonacci kedua), lalu dilanjutkan dengan proses rekursif untuk mengisi *array* indeks berikutnya (i + 1). Kode programnya yaitu sebagai berikut:

```
1    else if (i == 1) {
2      arr[i] = b;
3      fibo(arr, a, b, n, i+1);
4    }
```

Langkah 4, proses rekursif ketiga yaitu membuat kondisi dimana indeks *array* dengan indeks lebih dari 1, maka diisi dengan nilai penjumlahan dua bilangan sebelumnya (jika yang dihitung indeks i, maka dua indeks sebelumnya yaitu i-1 dan i-2), lalu dilanjutkan dengan proses rekursif untuk mengisi *array* indeks berikutnya. Dari langkah ini dapat dilihat bahwa basis atau pemberhentian proses rekursif yaitu

jika i sama dengan n atau lebih. Kode programnya yaitu sebagai berikut:

```
  else {
    if (i < n) {
      arr[i] = arr[i-1] + arr[i-2];
      fibo(arr, a, b, n, i+1);
    }
  }
}
```

Langkah 5, memanggil fungsi rekursif pada program utama. Pada langkah ini, diatur angka pertama deret fibonacci (nilai a) yaitu 2, angka kedua (nilai b) yaitu 6, jumlah deret (nilai n) yaitu 10, dan nomor urut indeks (nilai i) dimulai dari 0 Kode programnya yaitu sebagai berikut:

```
public static void main(String[] args) throws Exception {
  int i = 0;
  int arr[] = new int[10];
  fibo(arr, 2, 6, 10, 0);

  for (i = 0; i < 10; i++) {
    System.out.print(arr[i] + " ");
  }
  }
}
```

Setelah itu, penulisan kode program selesai dan mulai menampilkan hasil *output*nya. Dari kode pada algoritma utama (*void main*) memanggil nilai a = 2, b = 6, n = 10, dan i = 0, sehingga hasilnya yaitu sebagai berikut:

```
2 6 8 14 22 36 58 94 152 246
```

3.4. Latihan Soal

1. Menurut Anda, apakah rekursif itu? Lalu bagian apa saja yang perlu dipenuhi untuk membuat program rekursif?
2. Dalam beberapa contoh rekursif pada subbab 3.3, langkah awalnya yaitu membuat fungsi dan prosedur, sebelum digunakan untuk pemanggilan rekursif di program utama (*void main*). Jelaskan perbedaan keduanya.
3. Buatlah program rekursif untuk menghitung jumlah bakteri yang berkembang biak setiap menitnya. Asumsikan dalam cairan kimia terdapat 6 buah bakteri dan setiap menit bakteri berkembang biak menjadi 2 kali lipatnya. Dengan program rekursif, carilah jumlah bakteri pada menit ke-10.
4. Buatlah program rekursif untuk menghitung saldo nasabah dengan bunga tunggal. Misalkan seorang nasabah menabung di bank A dengan saldo awal Rp4.000.000 dan diketahui bunga per bulannya 3%, maka dengan program rekursif, carilah saldo nasabah tersebut setelah 1 tahun (12 bulan).

BAB 4
PENGURUTAN (*SORTING*)
(Bagian 1)

4.1. Pengurutan (*Sorting*)

Pengurutan atau *sorting* menjadi suatu hal yang penting dalam kehidupan sehari-hari untuk memudahkan pengelolaan data. Pengurutan dapat dilakukan dengan dua cara, yaitu pengurutan secara naik (*ascending*) maupun secara turun (*descending*). Misalkan diberikan data berupa bilangan acak sebagai berikut:

26	17	34	31	59	98	88	67	40	19

Jika diurutkan secara naik (*ascending*) menjadi sebagai berikut:

17	19	26	31	34	40	59	67	88	98

Jika diurutkan secara turun (*descending*) menjadi sebagai berikut:

98	88	67	59	40	34	31	26	19	17

Apabila yang diurutkan berupa rekaman (*record*) yang terdiri dari beberapa jenis data, maka pengurutan dilakukan berdasarkan salah satu jenis data yang disimpan. Misalkan, sebuah *record* data pengadaan barang di gudang dapat diurutkan berdasarkan jumlah stok paling sedikit untuk menentukan barang apa saja yang perlu ditambah stoknya.

Metode pengurutan ada banyak sekali. Pada kanal *Youtube* PCBoy Studios, yang memuat konten algoritma pengurutan secara visual, salah satu videonya memuat konten 256 jenis algoritma pengurutan yang berbeda-beda pada video yang berjudul "*256(?) Sorting Algorithms in 3 Hours (feat. ArrayV Has a*

Mental Breakdown)". Namun, pada buku ini hanya membahas 5 metode pengurutan utama, diantaranya yaitu:

- Metode Penyisipan (*Insertion sort*)
- Metode Seleksi (*Selection sort*)
- Metode Gelembung (*Bubble sort*)
- Metode Urut Cepat (*Quick sort*)
- Metode Urut Gabung (*Merge sort*)

Pada bab ini akan membahas tiga metode pertama yaitu *insertion, selection*, dan *bubble sort*. Metode *quick sort* dan *merge sort* akan dibahas pada bab selanjutnya yaitu Pengurutan (*Sorting*) Bagian 2.

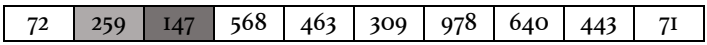
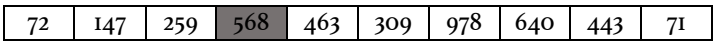
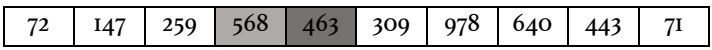
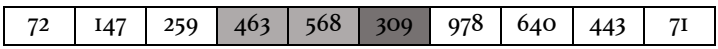
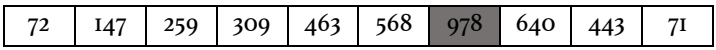
4.2. Metode Penyisipan (*Insertion Sort*)

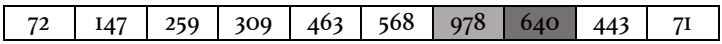
Metode penyisipan yaitu metode yang mengambil sebuah kunci pada data yang diurutkan dan menggeser data yang lebih besar dari kunci agar data kunci dapat diletakkan pada tempat yang benar. Misalkan ada sebuah array dengan isian angka berikut:

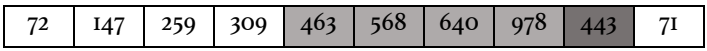
72	259	147	568	463	309	978	640	443	71

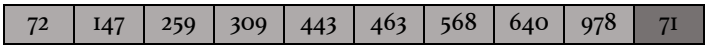
Apabila data tersebut akan diurutkan secara naik (*ascending*) dengan metode *insertion sort*, maka tahapannya sebagai berikut:

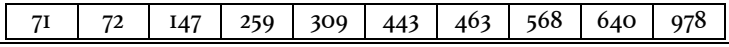
<table>
<tr><th>Kunci</th><th>Hasil Pengurutan</th></tr>
<tr><td>259</td><td>
<table>
<tr><td>72</td><td>259</td><td>147</td><td>568</td><td>463</td><td>309</td><td>978</td><td>640</td><td>443</td><td>71</td></tr>
</table>
Pada perulangan ke-1, data pada array indeks 2 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Jika data sebelumnya tidak ada yang lebih besar dari data kunci, maka
</td></tr>
</table>

	tidak ada yang digeser ke belakang.
147	 Pada perulangan ke-2, data pada *array* indeks 3 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Ada satu data yang lebih besar dari data kunci (259 > 147), maka data sebelumnya (259) harus digeser satu tempat dan data kunci dipindah ke tempat sebelum satu data digeser.
568	 Pada perulangan ke-3, data pada *array* indeks 4 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Jika data sebelumnya tidak ada yang lebih besar dari data kunci, maka tidak ada yang digeser ke belakang.
463	 Pada perulangan ke-4, data pada *array* indeks 5 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Ada satu data yang lebih besar dari data kunci (568 > 463), maka data sebelumnya (568) harus digeser satu tempat dan data kunci dipindah ke tempat sebelum satu data digeser.
309	 Pada perulangan ke-5, data pada *array* indeks 6 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Ada dua data yang lebih besar dari data kunci (463, 568 > 309), maka data sebelumnya (463, 568) harus digeser satu tempat dan data kunci dipindah ke tempat sebelum dua data digeser.
978	 Pada perulangan ke-6, data pada *array* indeks 7 dijadikan kunci

<table>
<tr><td></td><td>untuk dibandingkan dengan data sebelumnya. Jika data sebelumnya tidak ada yang lebih besar dari data kunci, maka tidak ada yang digeser ke belakang.</td></tr>
<tr><td>640</td><td> 

Pada perulangan ke-7, data pada array indeks 8 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Ada satu data yang lebih besar dari data kunci (978 > 640), maka data sebelumnya (978) harus digeser satu tempat dan data kunci dipindah ke tempat sebelum satu data digeser.</td></tr>
<tr><td>443</td><td> 

Pada perulangan ke-8, data pada array indeks 9 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Ada empat data yang lebih besar dari data kunci (463, 568, 640, 978 > 443), maka data sebelumnya (463, 568, 640, 978) harus digeser satu tempat dan data kunci dipindah ke tempat sebelum empat data digeser.</td></tr>
<tr><td>71</td><td> 

Pada perulangan ke-9, data pada array indeks 10 dijadikan kunci untuk dibandingkan dengan data sebelumnya. Ada sembilan data yang lebih besar dari data kunci (72, 147, 259, 309, 443, 463, 568, 640, 978 > 71), maka data sebelumnya (72, 147, 259, 309, 443, 463, 568, 640, 978) harus digeser satu tempat dan data kunci dipindah ke tempat paling depan (sebelum sembilan data digeser).</td></tr>
<tr><td>Hasil akhir</td><td> 
</td></tr>
</table>

Jumlah penelusuran untuk algoritma *Insertion Sort* diketahui dengan rumus:

n + (n-1) + (n-2) + (n-3) + ... + 1, untuk n = jumlah elemen pada *array*. Maka jika contoh di atas terdapat 10 elemen, maka ada 55 proses penelusuran.

Berikut penulisan kode program untuk pengurutan metode *insertion sort*:

Langkah 1, membuat prosedur untuk menampilkan isi array. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static void Array(int n, int [] arr) {
4      int i = 0;
5
6      for (i = 0; i < n; i++) {
7        System.out.print(arr[i] + " ");
8      }
9    }
```

Langkah 2, membuat deklarasi *array* yang berisi bilangan sesuai contoh di atas. Kode programnya yaitu sebagai berikut:

```
1  public static void main(String[] args) throws Exception {
2    int tabInt[] = {72, 259, 147, 568, 463, 309, 978, 640,
   443, 71};
```

Langkah 3, membuat deklarasi bilangan *counter* sebagai *counter* perulangan. Kode programnya yaitu sebagai berikut:

```
1    int i = 0;
```

Langkah 4, membuat deklarasi perulangan yang memproses data kunci, yang di dalamnya dideklarasikan bilangan *counter* untuk perulangan yang memproses pergeseran data. Kode programnya yaitu sebagai berikut:

```
1     int kunci = 0;
2     int j;
3
4     for (i = 1; i < tabInt.length; i++) {
5       kunci = tabInt[i];
6       j = i - 1;
7
8       while ((j >= 0) && (kunci < tabInt[j])) {
9         tabInt[j + 1] = tabInt[j];
10        j = j - 1;
11      }
12      tabInt[j + 1] = kunci;
13    }
14    tampilArray(tabInt.length, tabInt);
15    }
16  }
```

Pada langkah tersebut, proses pengurutan *insertion sort* dimulai pada baris 4, dimana data kunci dipilih sebagai acuan untuk pengurutan. Sementara pada baris 8, proses pengurutan dilakukan apabila data kunci lebih kecil dari data *array*, maka data *array* dipindah ke belakang.

Kode di atas merupakan langkah pengurutan secara naik (*ascending*). Apabila ingin diurutkan dari yang terbesar ke terkecil (*descending*), maka kode pada baris 8 diubah menjadi (kunci > tabInt[j]).

Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya yang telah terurut, yaitu sebagai berikut:

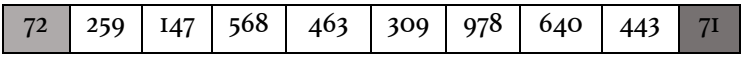
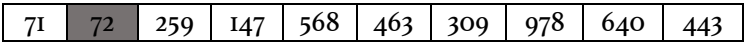
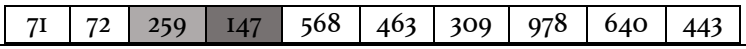
```
71 72 147 259 309 443 463 568 640 978
```

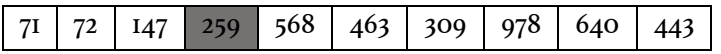
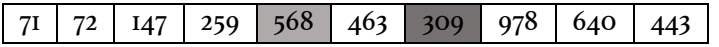
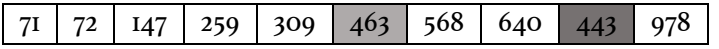
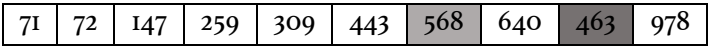
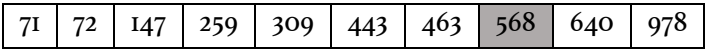
4.3. Metode Seleksi (*Selection Sort*)

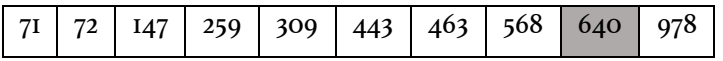
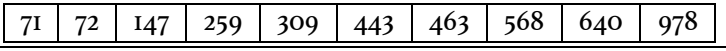
Metode seleksi yaitu metode yang mencari nilai terkecil atau terbesar (sesuai permintaan apakah diurutkan secara *ascending* maupun *descending*) untuk selanjutnya ditempatkan ke tempat terdepan, lalu selanjutnya mencari lagi nilai terkecil atau terbesar kedua, lalu ditukar dengan nilai minimum, demikian seterusnya, sehingga jumlah proses penelusuran *selection sort* sama dengan *insertion sort*. Misalkan ada sebuah array dengan isian angka berikut:

72	259	147	568	463	309	978	640	443	71

Apabila data tersebut akan diurutkan secara naik (*ascending*) dengan metode *selection sort*, maka tahapannya sebagai berikut:

Nilai min.	Hasil Pengurutan
71	 Pada perulangan ke-1, dicari nilai terkecil dari elemen *array* indeks 1 – 10, dan ditemukan nilai terkecilnya yaitu 71. Kemudian, nilai 71 ditempatkan di elemen paling depan.
72	 Pada perulangan ke-2, dicari nilai terkecil dari elemen *array* indeks 2 – 10, dan ditemukan nilai terkecilnya yaitu 72. Nilai 72 sesuai dengan tempatnya, maka tidak perlu ada pertukaran elemen.
147	

	Pada perulangan ke-3, dicari nilai terkecil dari elemen *array* indeks 3 – 10, dan ditemukan nilai terkecilnya yaitu 147. Kemudian, nilai 147 ditempatkan di elemen indeks 3.
259	 Pada perulangan ke-4, dicari nilai terkecil dari elemen *array* indeks 4 – 10, dan ditemukan nilai terkecilnya yaitu 259. Nilai 259 sesuai dengan tempatnya, maka tidak perlu ada pertukaran elemen.
309	 Pada perulangan ke-5, dicari nilai terkecil dari elemen *array* indeks 5 – 10, dan ditemukan nilai terkecilnya yaitu 309. Kemudian, nilai 309 ditempatkan di elemen indeks 5.
443	 Pada perulangan ke-6, dicari nilai terkecil dari elemen *array* indeks 6 – 10, dan ditemukan nilai terkecilnya yaitu 443. Kemudian, nilai 443 ditempatkan di elemen indeks 6.
463	 Pada perulangan ke-7, dicari nilai terkecil dari elemen *array* indeks 7 – 10, dan ditemukan nilai terkecilnya yaitu 463. Kemudian, nilai 443 ditempatkan di elemen indeks 7.
568	

	Pada perulangan ke-8, dicari nilai terkecil dari elemen *array* indeks 8 – 10, dan ditemukan nilai terkecilnya yaitu 568. Nilai 568 sesuai dengan tempatnya, maka tidak perlu ada pertukaran elemen.
640	 Pada perulangan ke-9, dicari nilai terkecil dari elemen *array* indeks 9 – 10, dan ditemukan nilai terkecilnya yaitu 640. Nilai 640 sesuai dengan tempatnya, maka tidak perlu ada pertukaran elemen.
Hasil akhir	

Berikut penulisan kode program untuk pengurutan metode *insertion sort*:

Langkah 1, membuat prosedur untuk menampilkan isi array. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static void Array(int n, int [] arr) {
4      int i = 0;
5
6      for (i = 0; i < n; i++) {
7        System.out.print(arr[i] + " ");
8      }
9    }
```

Langkah 2, membuat deklarasi *array* yang berisi bilangan sesuai contoh di atas. Kode programnya yaitu sebagai berikut:

```
1  public static void main(String[] args) throws Exception {
2    int tabInt[] = {72, 259, 147, 568,
   463, 309, 978, 640, 443, 71};
```

Langkah 3, membuat deklarasi bilangan *counter* sebagai *counter* perulangan dan variabel temp sebagai tempat penyimpanan nilai sementara. Kode programnya yaitu sebagai berikut:

```
1    int i = 0;
2    int temp = 0;
```

Langkah 4, membuat deklarasi perulangan yang memproses penukaran nilai elemen dan pencarian nilai minimum. Kode programnya yaitu sebagai berikut:

```
1    int min = 0;
2    int j;
3
4    for (i = 0; i < tabInt.length-1; i++) {
5      min = i;
6
7      for (j = (i + 1); j < tabInt.length; j++) {
8        if (tabInt[min] > tabInt[j]) {
9          min = j;
10       }
11     }
12     temp = tabInt[i];
13     tabInt[i] = tabInt[min];
14     tabInt[min] = temp;
15   }
16   tampilArray(tabInt.length, tabInt);
17   }
18 }
```

Pada langkah tersebut, proses pengurutan *selection sort* dimulai pada baris 4, dimana nilai minimum diinisialisasi.

Kemudian pada baris 7, dilakukan pencarian nilai minimum di sepanjang *array*. Lalu pada baris 12, dilakukan penukaran posisi elemen untuk nilai minimum yang telah ditemukan.

Kode di atas merupakan langkah pengurutan secara naik (*ascending*). Apabila ingin diurutkan dari yang terbesar ke terkecil (*descending*), maka kode pada baris 8 diubah menjadi (tabInt[min] < tabInt[j]).

Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya yang telah terurut, yaitu sebagai berikut:

```
71 72 147 259 309 443 463 568 640 978
```

4.4. Metode Gelembung (*Bubble Sort*)

Metode gelembung (bubble sort) yaitu metode pengurutan yang dilakukan dengan menukarkan dua elemen secara terus menerus hingga pengurutan selesai. Jumlah tahapan pada metode ini yaitu perkalian kuadrat dari jumlah elemen array – 1. Misalkan ada sebuah array dengan isian angka berikut:

72	259	147	568	463	309	978	640	443	71

Sehingga diperlukan $(10 - 1)^2 = 81$ langkah pengerjaan untuk metode tersebut. Tahapan pengurutan metode *bubble sort* secara naik (*ascending*) yaitu sebagai berikut (CATATAN: langkah-langkah ini hanya menampilkan langkah pentingnya saja, yaitu pada langkah penukaran nilainya saja):

Hasil Pengurutan

72	259	147	568	463	309	978	640	443	71

Pada perulangan ke-1, membandingkan nilai elemen indeks 1 dan 2. Karena indeks 1 lebih kecil dari indeks 2, maka tidak ada proses pertukaran.

72	147	259	568	463	309	978	640	443	71

Pada perulangan ke-2, membandingkan nilai elemen indeks 2 dan 3. Karena indeks 2 lebih besar dari indeks 3, maka dilakukan proses pertukaran.

72	147	259	568	463	309	978	640	443	71

Pada perulangan ke-3, membandingkan nilai elemen indeks 3 dan 4. Karena indeks 3 lebih kecil dari indeks 4, maka tidak ada proses pertukaran.

72	147	259	463	568	309	978	640	443	71

Pada perulangan ke-4, membandingkan nilai elemen indeks 4 dan 5. Karena indeks 4 lebih besar dari indeks 5, maka dilakukan proses pertukaran.

72	147	259	463	309	568	978	640	443	71

Pada perulangan ke-5, membandingkan nilai elemen indeks 5 dan 6. Karena indeks 5 lebih besar dari indeks 6, maka dilakukan proses pertukaran.

72	147	259	463	309	568	978	640	443	71

Pada perulangan ke-6, membandingkan nilai elemen indeks 6 dan 7. Karena indeks 6 lebih kecil dari indeks 7, maka tidak ada proses pertukaran.

72	147	259	463	309	568	640	978	443	71

Pada perulangan ke-7, membandingkan nilai elemen indeks 7 dan 8. Karena indeks 7 lebih besar dari indeks 8, maka dilakukan proses pertukaran.

72	147	259	463	309	568	640	443	978	71

Pada perulangan ke-8, membandingkan nilai elemen indeks 8 dan 9. Karena indeks 8 lebih besar dari indeks 9, maka dilakukan proses pertukaran.

72	147	259	463	309	568	640	443	71	978

Pada perulangan ke-9, membandingkan nilai elemen indeks 9 dan 10. Karena indeks 9 lebih besar dari indeks 10, maka dilakukan proses pertukaran.

Sampai dengan tahap ini, maka perulangan ke-10 diulang lagi dari perulangan ke-1 sebanyak 9 kali. Maka selanjutnya yang dilakukan penukaran yaitu: perulangan ke-13

72	147	259	309	463	568	640	443	71	978

perulangan ke-16

72	147	259	309	463	568	443	640	71	978

perulangan ke-17

72	147	259	309	463	568	443	71	640	978

perulangan ke-24

72	147	259	309	463	443	568	71	640	978

perulangan ke-26

72	147	259	309	463	443	71	568	640	978

perulangan ke-32

72	147	259	309	443	463	71	568	640	978

perulangan ke-33

72	147	259	309	443	71	463	568	640	978

perulangan ke-41

72	147	259	309	71	443	463	568	640	978

perulangan ke-49

72	147	259	71	309	443	463	568	640	978

perulangan ke-57

72	147	71	259	309	443	463	568	640	978

perulangan ke-65

72	71	147	259	309	443	463	568	640	978

perulangan ke-73

71	72	147	259	309	443	463	568	640	978

Setelah perulangan ke-73, perulangan ke-74 sampai ke-81 hasilnya sama saja, sehingga bilangan telah terurut. Dari penjelasan ini, metode *bubble sort* dinilai kurang efisien, tetapi sangat mudah dipahami. Berikut penulisan kode program untuk pengurutan metode *bubble sort*:

Langkah 1, membuat prosedur untuk menampilkan isi array. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static void Array(int n, int [] arr) {
4      int i = 0;
5
6      for (i = 0; i < n; i++) {
7        System.out.print(arr[i] + " ");
8      }
9    }
```

Langkah 2, membuat deklarasi *array* yang berisi bilangan sesuai contoh di atas. Kode programnya yaitu sebagai berikut:

```
1  public static void main(String[] args) throws Exception {
2    int tabInt[] = {72, 259, 147, 568, 463, 309, 978, 640, 443, 71};
```

Langkah 3, membuat deklarasi bilangan *counter* sebagai *counter* perulangan, variabel temp sebagai tempat penyimpanan nilai sementara dan boolean tukar sebagai tanda apakah terjadi pertukaran atau tidak. Kode programnya yaitu sebagai berikut:

```
1    int i = 0;
2    int temp = 0;
3    Boolean tukar = false;
```

Langkah 4, membuat deklarasi perulangan yang memproses penukaran nilai elemen. Kode programnya yaitu sebagai berikut:

```
1   do {
2     tukar = false;
3     for (i = 0; i < tabInt.length-1; i++) {
4       if (tabInt[i] > tabInt[i + 1]) {
5         temp = tabInt[i];
6         tabInt[i] = tabInt[i+1];
7         tabInt[i+1] = temp;
8         tukar = true;
9       }
10    }
11  } while (tukar == true);
12
13  tampilArray(tabInt.length, tabInt);
14  }
15 }
```

Pada langkah tersebut, proses pengurutan *selection sort* dimulai pada baris 1, dimana nilai tukar sebelum adanya pertukaran maka diatur false. Kemudian pada baris 3 dan 4, dilakukan pengulangan sekaligus pemeriksaan apakah terjadi penukaran. Lalu pada baris 5, dilakukan penukaran posisi elemen.

Kode di atas merupakan langkah pengurutan secara naik (*ascending*). Apabila ingin diurutkan dari yang terbesar ke terkecil (*descending*), maka kode pada baris 4 diubah menjadi (tabInt[i] < tabInt[i + 1]).

Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya yang telah terurut, yaitu sebagai berikut:

```
71 72 147 259 309 443 463 568 640 978
```

4.5. Latihan Soal

1. Carilah salah satu metode pengurutan selain yang dibahas pada buku ini. Berikan penjelasan algoritmanya. CATATAN: Pastikan setiap mahasiswa berbeda metode pengurutannya.
2. Pisahkan NIM Anda menjadi satu digit angka per indeks *array*. Contoh, NIM 223140001, maka terdapat 9 buah indeks *array*, yaitu 2, 2, 3, 1, 4, 0, 0, 0, 1. Kemudian urutkan naik ataupun turun dengan algoritma *insertion sort* (langkah pengurutan maupun kode programnya).
3. Pisahkan NIM Anda menjadi satu digit angka per indeks *array*. Contoh, NIM 223140001, maka terdapat 9 buah indeks *array*, yaitu 2, 2, 3, 1, 4, 0, 0, 0, 1. Kemudian urutkan naik ataupun turun dengan algoritma *selection sort* (langkah pengurutan maupun kode programnya).
4. Pisahkan NIM Anda menjadi satu digit angka per indeks *array*. Contoh, NIM 223140001, maka terdapat 9 buah indeks *array*, yaitu 2, 2, 3, 1, 4, 0, 0, 0, 1. Kemudian urutkan naik ataupun turun dengan algoritma *bubble sort* (langkah pengurutan maupun kode programnya).

BAB 5
PENGURUTAN (*SORTING*)
(Bagian 2)

Pada bab sebelumnya, telah dibahas mengenai beberapa metode sederhana pada *sorting* seperti *insertion sort, selection sort*, dan *bubble sort*. Pada bab ini akan dibahas metode *advanced* untuk *sorting*, yaitu *quick sort* dan *merge sort*.

5.1. Metode Urut Cepat (*Quick Sort*)

Metode urut cepat yaitu metode pengurutan yang menjadikan sebuah tabel yang diurutkan akan dijadikan dua sub bagian yang ditelusuri dari kiri dan dari kanan. Metode ini mengumpulkan nilai-nilai yang kecil di bagian depan atau kiri dan nilai yang besar di bagian belakang atau kanan. Dalam metode ini, bergantung pada pivot (acuan) angka yang dipilih, baik pivot pinggir kiri atau kanan, maupun pivot tengah. Misalkan ada sebuah array dengan isian angka berikut:

72	259	147	568	463	309	978	640	443	71

Tahapan pengurutannya secara naik (*ascending*) yaitu sebagai berikut:

Langkah 1, menentukan pivot terlebih dahulu. Dalam hal ini, yang dipilih sebagai pivot yaitu nomor 71 (*array* paling kanan). Kemudian, menentukan nilai i dan j, dimana nilai i = *array* paling kiri dan j = *array* paling kanan – 1.

72	259	147	568	463	309	978	640	443	71

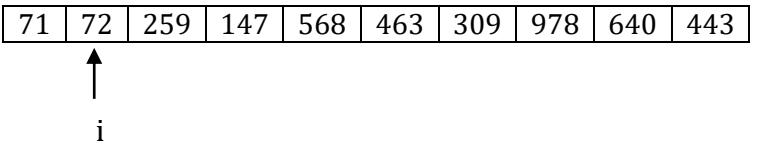
i

Langkah 2, membandingkan nilai i. Posisi i akan terus bergerak ke kanan jika i < pivot dan j akan bergerak ke kiri jika j >= pivot dan tidak melewati indeks terkiri. Dari ilustrasi di atas, dapat diketahui bahwa i = 72, j = 443, dan pivot = 71.

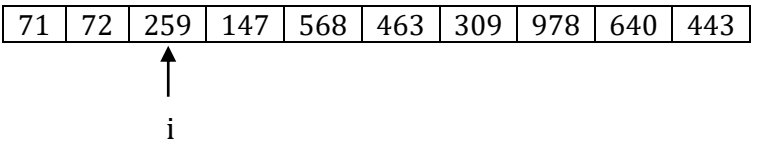
Karena 72 > 71, maka posisi i tidak bergeser. Sementara nilai j = 443, karena 443 >= 71, maka posisi j digeser ke kiri untuk membandingkan angka lainnya. Dari angka tersebut, semua angka hingga bertemu nilai i (72) bernilai lebih besar atau sama dengan 71, maka posisi pivot dipindah ke posisi pertama, dan angka lain bergeser 1 langkah menjadi sebagai berikut.

71	72	259	147	568	463	309	978	640	443

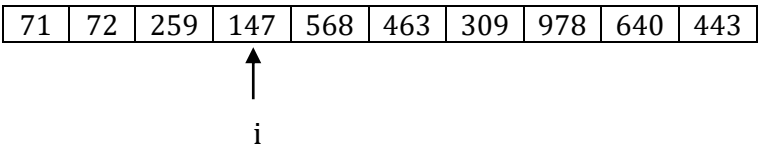
Langkah 3, posisi angka 71 dipastikan menempati urutan yang tepat di indeks 1. Kemudian menggunakan nilai i pada angka 72. Dalam hal ini, j awal di 640 dan pivot di 443.



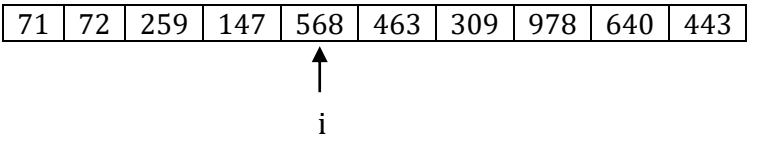
Langkah 4, membandingkan nilai i. Karena 72 < 443, maka posisi i bergeser ke 259.



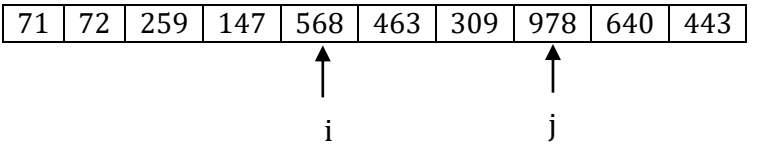
Langkah 5, membandingkan nilai i. Karena 259 < 443, maka posisi i bergeser ke 147.



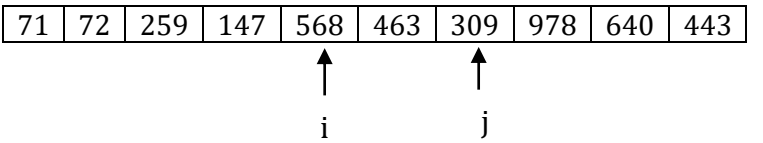
Langkah 6, membandingkan nilai i. Karena 147 < 443, maka posisi i bergeser ke 568.



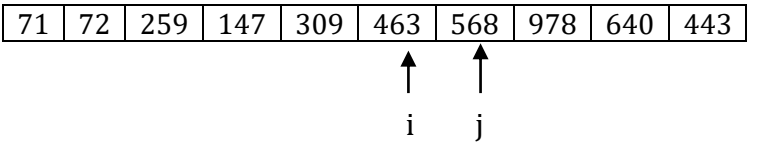
Langkah 7, membandingkan nilai i. Karena 568 > 443, maka posisi i tidak bergeser. Kemudian membandingkan nilai j. Karena 640 > 443, maka posisi j dipindah ke 978.



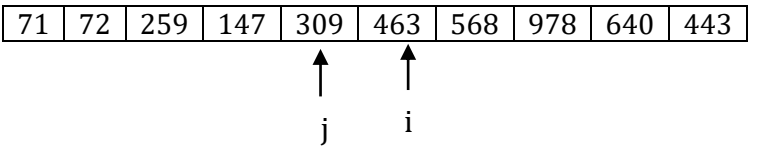
Langkah 8, membandingkan nilai j. Karena 978 > 443, maka posisi j dipindah ke 309.



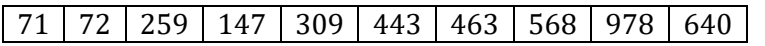
Langkah 9, membandingkan nilai j. Karena 309 < 443, maka posisi j tidak dipindahkan. Pada tahapan ini, karena memenuhi syarat i > pivot dan j < pivot, maka posisi i dan j bertukar menjadi sebagai berikut.



Langkah 10, membandingkan nilai i. Karena 463 > 443, maka posisi i tidak dipindahkan. Kemudian, membandingkan nilai j. Karena 568 > 443, maka posisi j dipindahkan dan bertemu dengan i di 463. Karena 463 > 443 juga, maka posisi j dipindahkan ke 309.



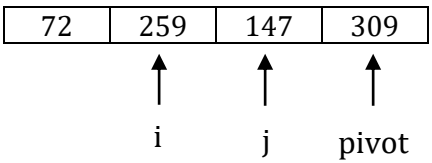
Langkah 11, posisi i dan j saling berlawanan, sehingga dapat disimpulkan bahwa nilai pivot (443) berada di antara j (309) dan i (463) dan susunannya menjadi sebagai berikut.



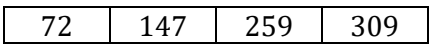
Langkah 12, posisi angka 443 dipastikan telah menempati urutan yang benar pada indeks 6. Selanjutnya yaitu membagi dua sub bagian yaitu sub bagian 1 pada indeks 2 – 5 dan sub bagian 2 pada indeks 7 – 10.

Langkah 13, lakukan penentuan i, j, dan pivot pada sub bagian 1 dengan penentuan sebagai berikut.

Langkah 14, membandingkan nilai i. Karena 72 < 309, maka posisi i dipindahkan ke 259.

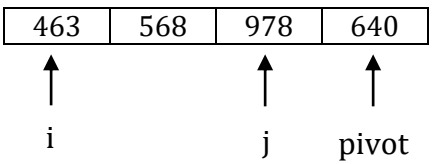


Langkah 15, membandingkan nilai i. Karena 259 < 309, dan tidak ada tempat lagi untuk dipindahkan, maka langsung menuju nilai j. Karena 147 < 309, maka posisi tidak dipindahkan. Karena i dan j tidak berpindah, maka posisi i dan j saling bertukar menjadi sebagai berikut.

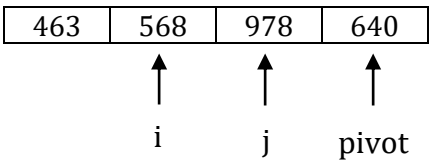


Posisi bilangan sudah menempati urutan yang tepat. Maka dari itu, sub bagian 1 telah selesai diurutkan.

Langkah 16, lakukan penentuan i, j, dan pivot pada sub bagian 2 dengan penentuan sebagai berikut.



Langkah 17, membandingkan nilai i. Karena 463 < 640, maka posisi i dipindahkan ke 568.



Langkah 18, membandingkan nilai i. Karena 568 < 640, dan tidak ada tempat lagi untuk dipindahkan, maka langsung menuju nilai j. Karena 978 > 640, dan jika dipindahkan posisi j bertemu posisi i, maka nilai pivot dapat diletakkan di antara i dan j menjadi sebagai berikut.

463	568	640	978

Posisi bilangan sudah menempati urutan yang tepat. Maka dari itu, sub bagian 2 telah selesai diurutkan.

Langkah 19, seluruh angka telah diposisikan dengan urutan yang tepat. Maka dari itu, proses metode *quick sort* telah selesai dengan urutan sebagai berikut:

71	72	147	259	309	443	463	568	640	978

Berikut penulisan kode program untuk pengurutan metode *quick sort*:

Langkah 1, membuat prosedur untuk menampilkan isi array. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static void Array(int [] arr) {
4      int i = 0;
5
6      for (i = 0; i < arr.length; i++) {
7        System.out.print(arr[i] + " ");
8      }
9    }
```

Langkah 2, membuat prosedur tukar yang berisi *array*, variabel i dan j untuk menukar dua elemen *array*. Kode programnya yaitu sebagai berikut:

```
1  public static void tukar(int[] arr, int i, int j) {
2
3    int temp = arr[i];
4    arr[i] = arr[j];
5    arr[j] = temp;
6  }
```

Langkah 3, membuat fungsi partisi dan quickSort yang berisi *array*, variabel rendah dan tinggi untuk proses pengurutannya. Kode programnya yaitu sebagai berikut:

```
1   public static int partisi(int[] arr, int rendah, int tinggi) {
2     int pivot = arr[tinggi];
3     int i = (rendah - 1);
4     for (int j = rendah; j <= tinggi - 1; j++) {
5       if (arr[j] < pivot) {
6         i++;
7         tukar(arr, i, j);
8       }
9     }
10    tukar(arr, i + 1, tinggi);
11    return (i + 1);
12  }
13
14  public static void quickSort(int[] arr, int rendah, int tinggi) {
15
16    if (rendah < tinggi) {
17      int pi = partisi(arr, rendah, tinggi);
18      quickSort(arr, rendah, pi - 1);
19      quickSort(arr, pi + 1, tinggi);
20    }
21  }
```

Pada kode di atas, pivot yang dipilih yaitu pivot yang berada di akhir indeks array. Berdasarkan baris 5, apabila nilai indeks j lebih kecil dari pivot, maka terjadi penukaran dua indeks i dan j. Apabila pengurutan ingin dibuat *descending*, kode baris 5 diubah menjadi arr[j] > pivot.

Langkah 4, menjalankan *sorting* di program utama. Kode programnya yaitu sebagai berikut:

```
1  public static void main(String[] args) throws Exception {
2    int tabInt[] = {72, 259, 147, 568, 463, 309, 978, 640, 443, 71};
3      quickSort(tabInt, 0, tabInt.length - 1);
4      tampilArray(tabInt);
5      }
6  }
```

Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya yang telah terurut, yaitu sebagai berikut:

```
71 72 147 259 309 443 463 568 640 978
```

5.2. Metode Urut Gabung (*Merge Sort*)

Metode urut gabung yaitu metode pengurutan yang dilakukan dengan membagi dua *array* sama banyak secara terus menerus hingga menjadi satu elemen per bagian, kemudian digabung kembali dengan kondisi nilai terurut sebelum disatukan. Misalkan ada sebuah array dengan isian angka berikut:

72	259	147	568	463	309	978	640	443	71

Tahapan pengurutannya secara naik (*ascending*) yaitu sebagai berikut:

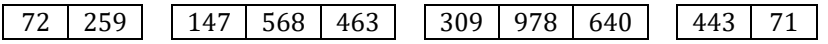
Langkah 1, bagilah *array* menjadi dua bagian sama banyak. Karena ada 10 elemen, maka tiap bagian dibagi menjadi 5 elemen sebagai berikut.

72	259	147	568	463

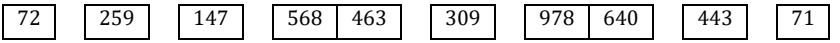
309	978	640	443	71

Pemisahan pertama

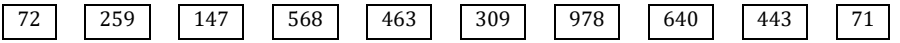
Langkah 2, bagilah *array* kembali menjadi dua bagian sama banyak. Karena ada 5 elemen, maka tiap bagian dibagi menjadi 2 elemen di satu sisi dan 3 elemen di sisi yang lain. Lalu ulangi hingga menjadi satu elemen per bagian.



Pemisahan kedua

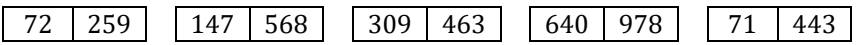


Pemisahan ketiga

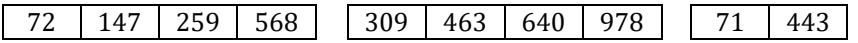


Pemisahan keempat

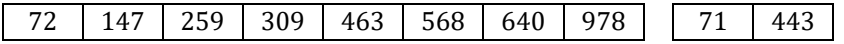
Langkah 3, gabungkan kembali array yang terpisah menjadi dua elemen per bagian, dengan diiringi pengurutan apabila belum terurut.



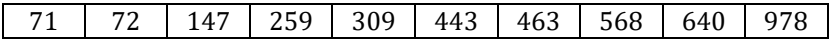
Penggabungan pertama



Penggabungan kedua



Penggabungan ketiga



Penggabungan keempat (proses pengurutan selesai)

Jumlah pemisahan elemen dan penggabungan elemen selalu sama, dimana bergantung pada banyaknya elemen yang diberikan. Jumlah elemen minimum dan maksimum apabila jumlah pemisahan dan penggabungan elemen diketahui, ditentukan dengan persamaan berikut:

Elemen minimum = $2^{(n-1)} + 1$, Elemen maksimum = 2^n
Dimana: n = jumlah pemisahan dan penggabungan elemen.
Berikut penulisan kode program untuk pengurutan metode *quick sort*:

Langkah 1, membuat prosedur untuk menampilkan isi array. Kode programnya yaitu sebagai berikut:

```
1  public class App {
2
3    public static void Array(int [] arr) {
4      int i = 0;
5
6      for (i = 0; i < arr.length; i++) {
7        System.out.print(arr[i] + " ");
8      }
9    }
```

Langkah 2, membuat prosedur merge untuk proses penggabungan dua subarray. Kode programnya yaitu sebagai berikut:

```
1  public static void merge(int arr[], int kiri, int
2  tengah, int kanan) {
3    //menentukan besaran dua subarray yang digabung
4    int n1 = tengah - kiri + 1;
5    int n2 = kanan - tengah;
6
7    //membuat array sementara
8    int L[] = new int[n1];
9    int R[] = new int[n2];
```

```
10
11    //salin data ke array sementara
12    for (int i = 0; i < n1; i++) {
13      L[i] = arr[kiri + i];
14      }
15    for (int j = 0; j < n2; j++) {
16      R[j] = arr[tengah + 1 + j];
17      }
18
19    //gabungkan array sementara
20
21    //inisialisasi subarray pertama dan kedua
22    int i = 0, j = 0;
23
24    //inisialisasi index dari array yang digabung
25    int k = kiri;
26    while (i < n1 && j < n2) {
27      if (L[i] <= R[j]) {
28        arr[k] = L[i];
29        i++;
30      }
31      else {
32        arr[k] = R[j];
33        j++;
34      }
35      k++;
36    }
37
38    //salin sisa elemen array L[] jika ada
39    while (i < n1) {
40      arr[k] = L[i];
41        i++;
42        k++;
43    }
44    //salin sisa elemen array R[] jika ada
45    while (j < n2) {
46      arr[k] = R[j];
47        j++;
48        k++;
49    }
50  }
```

Langkah 3, membuat fungsi pengurutan dengan merge sort. Kode programnya yaitu sebagai berikut:

```
1   public static void sort(int arr[], int kiri, int kanan) {
2     if (kiri < kanan) {
3
4       //mencari titik tengah
5       int tengah = kiri + (kanan - kiri) / 2;
6
7       //urutkan subbagian pertama dan kedua
8       sort(arr, kiri, tengah);
9       sort(arr, tengah + 1, kanan);
10
11      //gabungkan subbagian yang terurut
12      merge(arr, kiri, tengah, kanan);
13    }
14  }
```

Langkah 4, menjalankan *sorting* di program utama. Kode programnya yaitu sebagai berikut:

```
1   public static void main(String[] args) throws Exception {
2     int tabInt[] = {72, 259, 147, 568, 463, 309, 978, 640, 443, 71};
3       System.out.println("Diberikan Array: ");
4       Array(tabInt);
5
6       sort(tabInt, 0, tabInt.length - 1);
7
8       System.out.println("\nArray yang terurut: ");
9       Array(tabInt);
10    }
11  }
```

Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya yang telah terurut, yaitu sebagai berikut:

```
Diberikan Array:
72, 259, 147, 568, 463, 309, 978, 640, 443, 71

Array yang terurut: 71 72 147 259 309 443 463 568 640 978
```

5.3. Latihan Soal

1. Pisahkan NIM Anda menjadi satu digit angka per indeks *array*. Contoh, NIM 223140001, maka terdapat 9 buah indeks *array*, yaitu 2, 2, 3, 1, 4, 0, 0, 0, 1. Kemudian urutkan naik ataupun turun dengan algoritma *quick sort* (langkah pengurutan maupun kode programnya).
2. Pisahkan NIM Anda menjadi satu digit angka per indeks *array*. Contoh, NIM 223140001, maka terdapat 9 buah indeks *array*, yaitu 2, 2, 3, 1, 4, 0, 0, 0, 1. Kemudian urutkan naik ataupun turun dengan algoritma *merge sort* (langkah pengurutan maupun kode programnya).

BAB 6
PENCARIAN
(*SEARCHING*)

6.1. Pencarian (*Searching*)

Pencarian atau *searching* menjadi suatu hal yang penting dalam kehidupan sehari-hari untuk memudahkan pengelolaan data. Misalkan disediakan data-data berupa barang, harga, stok berikut:

Kode Barang	Nama Barang	Harga	Stok
13007501	Gula 500 gram	14900	144
13007502	Sabun mandi	8500	397
13007503	Mie instan goreng	3000	558
13007504	Minyak 2 L	45000	194
13007505	Tepung terigu 1 kg	19000	287

Dari data barang tersebut, akan dicari barang berupa tepung terigu 1 kg, dimana data yang dipunyai yaitu kode barang "13007505". Setelah dilakukan proses pencarian, akan diketahui bahwa tepung terigu 1 kg harganya 19000 dan memiliki stok sebanyak 287 pcs.

Pada bab ini, akan dibahas dua metode pencarian, yaitu:

- Pencarian beruntun (*sequential search*)
- Pencarian bagi dua (*binary search*)

6.2. Pencarian Beruntun (*Sequential Search*)

Pencarian beruntun yaitu metode yang dapat dilakukan baik untuk data yang terurut maupun yang belum terurut. Metode ini melakukan pencarian dengan menelusuri data satu per satu dan dicocokkan dengan data yang dicari. Jika tidak ada kecocokkan, maka penelusuran dilanjutkan, dan jika ada keococokkan, maka penelusuran dihentikan, yang artinya data telah berhasil ditemukan. Misalkan ada sebuah array dengan isian angka berikut:

72	259	147	568	463	309	978	640	443	71

Tahapan penulisan kode programnya yaitu:

Langkah 1, membuat fungsi cari yang memuat array dan variabel x sebagai nilai yang akan dicari. Kode programnya yaitu sebagai berikut:

```
1   public class App {
2
3     public static int cari(int arr[], int x) {
4       for (int i = 0; i < arr.length; i++) {
5         if (arr[i] == x) {
6           return i + 1;
7         }
8       }
9       return -1;
10    }
```

Pada langkah ini, proses pencarian terletak pada baris 4, dimana nilai yang akan dicari akan dicocokkan satu per satu sebanyak elemen array yang dimasukkan.

Langkah 2, menjalankan *searching* di program utama. Kode programnya yaitu sebagai berikut:

```
1   public static void main(String[] args) throws Exception {
2     int tabInt[] = {72, 259, 147, 568, 463, 309, 978, 640, 443, 71};
3     int x = 568;
4     int hasil = cari(tabInt, x);
5     if (hasil == -1) {
6       System.out.println("Pencarian tidak ditemukan!");
7     } else {
8       System.out.println("Pencarian ditemukan");
9       System.out.println(x + " terletak pada indeks ke " + hasil);
10    }
11    }
12  }
```

Pada langkah ini, nilai x sebagai nilai yang hendak dicari yaitu 568. Hasilnya, nilai 568 ditemukan pada *array* tabInt[] pada indeks ke-4. Setelah langkah ini, maka penulisan kode sudah

selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya, yaitu sebagai berikut:

```
Pencarian ditemukan
568 terletak pada indeks ke 4
```

6.3. Pencarian Bagi Dua (*Binary Search*)

Pencarian bagi dua yaitu metode pencarian yang datanya harus terurut terlebih dahulu. Proses pencariannya dilakukan dengan membagi dua *array* dan didasarkan pada kelompok dari *array* yang memungkinkan bilangan yang dicari berada sesuai urutan besar nilai pada masing-masing elemen *array*. Misalkan ada sebuah *array* dengan isian angka berikut:

72	259	147	568	463	309	978	640	443	71

Jika yang dicari adalah angka 568, maka prosesnya sebagai berikut:

Langkah 1, urutkan terlebih dahulu *array* yang masih teracak menjadi sebagai berikut (urut naik).

71	72	147	259	309	443	463	568	640	978

Langkah 2, bagilah *array* menjadi 2 bagian.

71	72	147	259	309

Subbagian 1

443	463	568	640	978

Subagian 2

Langkah 3, periksalah apablia angka yang dicari lebih besar atau sama dengan indeks terkiri subbagian 2 maka yang dicari adalah subbagian 2. Apabila angka yang dicari lebih kecil atau sama dengan indeks terkanan subbagian 1 maka yang dicari

adalah subbagian 1. Karena yang dicari angka 568, maka subbagian yang dicari yaitu subbagian 2.

443	463	568	640	978

Subbagian 2

Langkah 4, bagilah subbagian 2 menjadi 2 bagian. Karena jumlah subbagian 2 adalah 5 elemen, maka elemen di tengah ikut bergabung dengan element pada subbagian kanan (subbagian 2.2)

443	463

Subbagian 2.1

568	640	978

Subagian 2.2

Langkah 5, periksalah apablia angka yang dicari lebih besar atau sama dengan indeks terkiri subbagian 2.2 maka yang dicari adalah subbagian 2.2. Apabila angka yang dicari lebih kecil atau sama dengan indeks terkanan subbagian 2.1 maka yang dicari adalah subbagian 2.1. Karena yang dicari angka 568, maka subbagian yang dicari yaitu subbagian 2.2.

568	640	978

Subbagian 2.2

Langkah 6, bagilah subbagian 2.2 menjadi 2 bagian. Karena jumlah subbagian 2.2 adalah 3 elemen, maka elemen di tengah ikut bergabung dengan elemen pada subbagian kanan (subbagian 2.2.2)

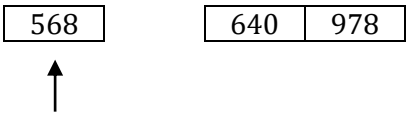
568

Subbagian 2.2.1

640	978

Subagian 2.2.2

Langkah 7, periksalah apablia angka yang dicari lebih besar atau sama dengan indeks terkiri subbagian 2.2.2 maka yang dicari adalah subbagian 2.2.2. Apabila angka yang dicari lebih kecil atau sama dengan indeks terkanan subbagian 2.2.1 maka yang dicari adalah subbagian 2.2.1. Karena yang dicari angka 568, maka subbagian yang dicari yaitu subbagian 2.2.1. Karena pada subbagian 2.2.1 hanya 1 elemen, maka nilai 568 otomatis ditemukan pada subbagian 2.2.1. Proses pencarian selesai, nilai 568 berada pada indeks ke-8.



Tahapan penulisan kode programnya yaitu:

Langkah 1, membuat prosedur bianrySearch yang memuat array, variabel kiri, kanan, dan variabel x sebagai nilai yang akan dicari. Kode programnya yaitu sebagai berikut:

```
1   public class App {
2   
3     public static void binarySearch(int arr[], int kiri, int
    kanan, int kunci){
4     int tengah = (kiri + kanan)/2;
5     while(kiri <= kanan) {
6       if (arr[tengah] < kunci) {
7         kiri = tengah + 1;
8       } else if (arr[tengah] == kunci) {
9         System.out.println("Pencarian ditemukan");
10        System.out.println(kunci + " berada pada indeks: "+
    (tengah + 1));
11        break;
12      } else {
13        kanan = tengah - 1;
14      }
15      tengah = (kiri + kanan)/2;
16    }
17    if (kiri > kanan) {
18    System.out.println("Pencarian tidak ditemukan!");
19    }
20  }
```

Pada langkah ini, proses pencarian terletak pada baris 4, dimana *array* akan dibagi menjadi dua bagian dan kemudian dibandingkan dengan nilai kuncinya untuk memilih bagian *array*nya dan akan dibagi lagi sampai menjadi bagian-bagian kecil. Apabila array diurutkan secara turun (*descending*), maka pada baris 6 diubah menjadi arr[tengah] > kunci.

Langkah 2, menjalankan *searching* di program utama. Kode programnya yaitu sebagai berikut:

```
1  public static void main(String[] args) throws Exception {
2    int tabInt[] = {71, 72, 147, 259, 309, 443, 463, 568, 640, 978};
3    int kunci = 568;
4    int kanan = tabInt.length-1;
5    binarySearch(tabInt, 0, kanan, kunci);
6  }
```

Pada langkah ini, nilai x sebagai nilai yang hendak dicari yaitu 568. Hasilnya, nilai 568 ditemukan pada *array* tabInt[] pada indeks ke-8. Setelah langkah ini, maka penulisan kode sudah selesai. Maka selanjutnya yaitu menampilkan hasil *output*nya, yaitu sebagai berikut:

```
Pencarian ditemukan
568 terletak pada indeks ke 8
```

6.4. Latihan Soal

1. Menurut Anda, mengapa untuk algoritma *binary search* nilai-nilai pada *array* harus diurutkan terlebih dahulu sebelum dilakukan eksekusi?
2. Berdasarkan tabel barang pada subbah 6.1, carilah nama barang, harga, dan stok untuk barang berkode 13007503 dengan algoritma *sequential search*, dengan menggunakan kode program *Java.*
3. Perhatikan *array* berikut ini!

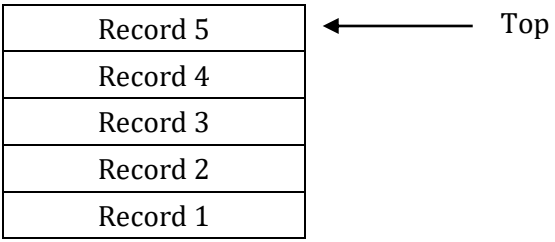
479	338	673	709	684	258	673	906	888	459

   Carilah angka 673 dengan algoritma *binary search* dengan terurut secara *descending*, dengan menggunakan kode program *Java*.
4. Menggunakan *array* pada soal nomor 3, carilah bilangan mana saja yang diawali dengan angka 6 dengan menggunakan algoritma *sequential search*. dengan menggunakan kode program *Java*.

BAB 7
STACK

7.1. *Stack*

Sesuai dengan katanya, *stack*, yang memiliki makna tumpukan. *Stack* merupakan suatu elemen yang ditumpuk dari bawah hingga atas, sedangkan dalam struktur data, *stack* diartikan sebagai penyimpanan dan pengambilan data yang diurutkan secara LIFO (*Last In First Out*). LIFO yaitu data terakhir yang disimpan akan diambil pertama kali. *Stack* dianalogikan sebagai sebuah tumpukkan koin, agar tumpukkan tidak ambruk, maka koin yang harus diambil adalah yang terletak paling atas terlebih dahulu. Ilustrasi mengenai stack ditunjukkan pada Gambar 7.1.



Gambar 7.1. Ilustrasi *Stack*

Berdasarkan Gambar 7.1. *stack* hanya dapat ditambah atau dikurangi *record*nya dari satu sisi. *Record* paling atas pada sebuah stack disebut *top of stack*. Semua operasi dalam *stack* dimulai dari record 5 yang berperan sebagai *top of stack*. Misalkan ingin mengambil satu *record* maka dilakukan satu per satu dari record 5, sehingga posisi top of stack berpindah ke record 4. Jika ingin menambahkan satu *record* baru (misalkan record 6), maka posisi puncak *stack* diganti menjadi record yang ditambahkan pada bagian atas *stack*, yaitu record 6.

7.2. Operator *Stack*

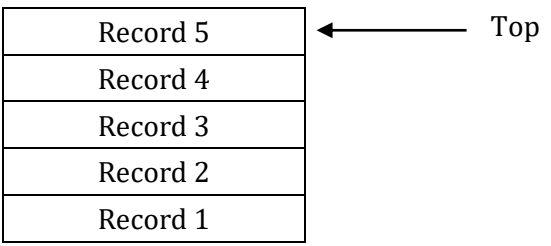
Secara umum, ada 4 macam operator dalam stack, diantaranya yaitu:

1. push()
2. pop()
3. peek()
4. search()

7.2.1. Operator Push()

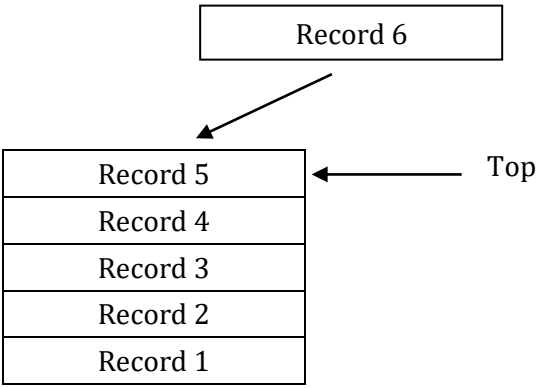
Operator push pada *stack* yaitu operator untuk menambahkan elemen baru pada *stack*. Ilustrasi untuk operator push yaitu sebagai berikut:

Sebagai kondisi awal ada sebuah stack yang memiliki beberapa elemen dengan elemen teratas disebut sebagai top.



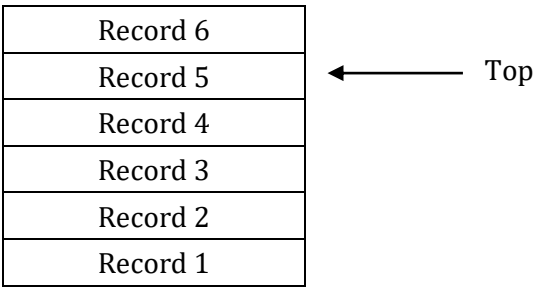
Gambar 7.2. Ilustrasi Awal *Stack*

Selanjutnya, ada satu buah elemen baru yang ditambahkan pada stack tersebut, yang dinamakan sebagai Record 6.



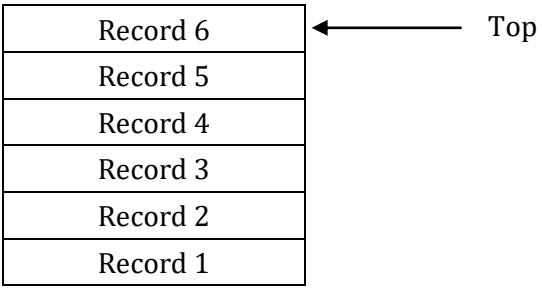
Gambar 7.3. Ilustrasi Penambahan Elemen pada *Stack*

Elemen "Record 6" dimasukkan ke dalam *stack*.



Gambar 7.4. Ilustrasi Record 6 Dimasukkan ke dalam *Stack*

Penunjuk top pada stack berganti ke elemen yang baru saja ditambahkan, yaitu Record 6.



Gambar 7.5. Ilustrasi Record 6 Sebagai *Top of Stack*

Untuk implementasi kode programnya yaitu sebagai berikut:

Langkah 1, siapkan beberapa deklarasi, seperti *scanner*, *array*, dan *stack* itu sendiri. Kode programnya yaitu sebagai berikut:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Scanner sc = new Scanner(System.in);
7       System.out.print("Masukkan banyak tumpukkan:");
8       int tumpukkan = sc.nextInt();
9       int a[] = new int[tumpukkan];
10      Stack<Integer> stack = new Stack<>();
```

Berdasarkan kode di atas, kita akan membuat sebuah *stack* dengan *array* yang bisa diatur jumlah kapasitas *stack*nya. Pada baris ke-10, deklarasi stack dibuat sesuai dengan tipe datanya, dengan format sebagai berikut:

```
Stack<tipe_data> stack = new Stack<>();
```

Namun perlu dicatat bahwa apabila ingin mengubah tipe data *stack*, maka tipe data untuk semua deklarasi perlu disesuaikan juga, kecuali untuk baris 8 dimana baris tersebut mendeklarasikan jumlah kapsitas *stack* yang diinginkan.

Langkah 2, buat perulangan untuk mengisikan *stack*. Kode programnya yaitu sebagai berikut:

```
1       for (int i = 0; i < tumpukkan; i++) {
2         System.out.print("Masukkan stack ke-" + (i+1) + ": ");
3         a[i] = sc.nextInt();
4         stack.push(a[i]);
5       }
6
7       System.out.println("Elemen pada stack yaitu: " + stack);
8     }
9   }
```

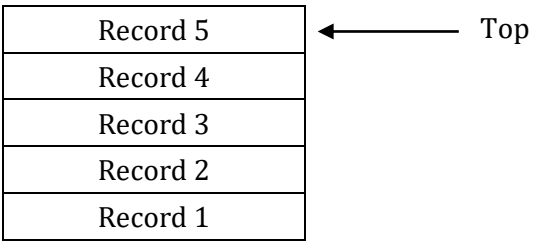
Pada kode di atas, pengisian *stack* dilakukan dengan perulangan sampai dengan sejumlah kapasitas *stack* yang telah ditentukan. Nilai pada *stack* ditentukan sesuai keinginan, kemudian operator *push* akan bekerja sebagai pengisian *stack*. Setelah langkah ini, penulisan kode program selesai. Berikut contoh hasil *output* dari program ini:

```
banyak tumpukan: 5
Masukkan stack ke-1: 10
Masukkan stack ke-2: 20
Masukkan stack ke-3: 30
Masukkan stack ke-4: 40
Masukkan stack ke-5: 50
Elemen pada stack yaitu: [10, 20, 30, 40, 50]
```

7.2.2. Operator Pop()

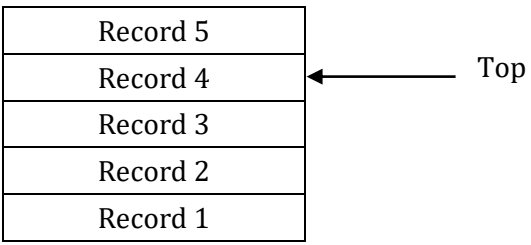
Operator pop pada *stack* yaitu operator untuk menghapus elemen teratas pada *stack*. Ilustrasi untuk operator push yaitu sebagai berikut:

Sebagai kondisi awal ada sebuah stack yang memiliki beberapa elemen dengan elemen teratas disebut sebagai top.



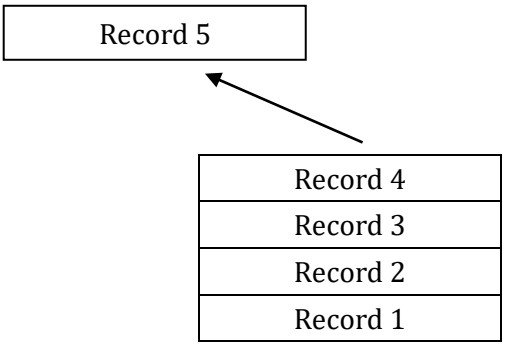
Gambar 7.6. Ilustrasi Awal *Stack*

Penunjuk top akan dipindah ke “Record 4”.



Gambar 7.7. Ilustrasi Record 4 Sebagai *Top of Stack*

Selanjutnya, Record 5 akan dihapus dari stack.



Gambar 7.8. Ilustrasi Penghapusan Stack 5

Untuk implementasi kode programnya yaitu sebagai berikut:

Langkah 1, siapkan beberapa deklarasi, seperti *scanner*, *array*, dan *stack* itu sendiri. Kode programnya yaitu sebagai berikut:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Scanner sc = new Scanner(System.in);
7       System.out.print("Masukkan banyak tumpukkan:");
8       int tumpukkan = sc.nextInt();
9       int a[] = new int[tumpukkan];
10      Stack<Integer> stack = new Stack<>();
```

Langkah 2, buat perulangan untuk mengisikan *stack*. Kode programnya yaitu sebagai berikut:

```
    for (int i = 0; i < tumpukkan; i++) {
      System.out.print("Masukkan stack ke-" + (i+1) + ": ");
      a[i] = sc.nextInt();
      stack.push(a[i]);
    }

    System.out.println("Elemen pada stack yaitu: " + stack);
```

Langkah 3, buatlah proses penghapusan elemen *stack*. Kode programnya yaitu sebagai berikut:

```
    System.out.println();
    System.out.print("Hapus berapa stack? ");
    int hapus_stack = sc.nextInt();

    for (int i = 0; i < hapus_stack; i++) {
      System.out.println("Stack yang dihapus: " + stack.pop());
    }

    System.out.println();
    System.out.println("Elemen stack saat ini yaitu: " + stack);
  }
}
```

Berdasarkan kode program di atas, proses penghapusan elemen *stack* ditentukan sesuai keinginan. Misalkan di awal mengisikan 5 elemen stack dan kemudian ingin menghapus 3 elemen, maka *output* dari program tersebut yaitu:

```
Masukkan banyak tumpukan: 5
Masukkan stack ke-1: 10
Masukkan stack ke-2: 20
Masukkan stack ke-3: 30
Masukkan stack ke-4: 40
Masukkan stack ke-5: 50
Elemen pada stack yaitu: [10, 20, 30, 40, 50]

Hapus berapa stack? 3
Stack yang dihapus: 50
Stack yang dihapus: 40
Stack yang dihapus: 30

Elemen stack saat ini yaitu: [10, 20]
```

Ketiga elemen yang dihapus yaitu 3 elemen teratas pada stack, yaitu 50, 40, dan 30, sehingga elemen yang tersisa pada stack yaitu 10 dan 20.

Dari ilustrasi tersebut, apabila penghapusan elemen sejumlah lebih dari kapasitas awal stack, maka akan terjadi *exception* yaitu *EmptyStackException*. Misal di awal akan membuat stack dengan kapasitas sebanyak 3 elemen, kemudian akan dilakukan penghapusan sebanyak 4 elemen, maka program tidak akan menghapus elemen *stack* karena setelah semua elemen terhapus maka *stack* akan kosong dan satu elemen sisanya tidak diketahui elemen mana yang dihapus, sehingga akan mentrigger *exception* tersebut.

7.2.3. Operator Peek()

Operator peek pada *stack* yaitu operator untuk mengidentifikasikan elemen yang paling atas pada stack. Dengan memodifikasi kode program pada subbab 7.2.2, maka kode program untuk operator peek yaitu sebagai berikut:

```
1  import java.util.*;
2
3  public class App {
4
5    public static void main(String[] args) throws Exception {
6      Scanner sc = new Scanner(System.in);
7      System.out.print("Masukkan banyak tumpukkan:");
8      int tumpukkan = sc.nextInt();
9      int a[] = new int[tumpukkan];
10     Stack<Integer> stack = new Stack<>();
11
12     for (int i = 0; i < tumpukkan; i++) {
13       System.out.print("Masukkan stack ke-" + (i+1) + ": ");
14       a[i] = sc.nextInt();
15       stack.push(a[i]);
16     }
17
18     System.out.println("Elemen pada stack yaitu: " + stack);
19     System.out.println("Elemen teratas stack yaitu: " + stack.peek());
20
21     System.out.println();
22     System.out.print("Hapus berapa stack? ");
23     int hapus_stack = sc.nextInt();
24
25     for (int i = 0; i < hapus_stack; i++) {
26       System.out.println("Stack yang dihapus: " + stack.pop());
27     }
28
29     System.out.println();
30     System.out.println("Elemen stack saat ini yaitu: " + stack);
31     System.out.println("Elemen teratas saat ini yaitu: " + stack.peek());
32   }
33 }
```

Hasil outputnya yaitu sebagai berikut:

```
Masukkan banyak tumpukan: 5
Masukkan stack ke-1: 10
Masukkan stack ke-2: 20
Masukkan stack ke-3: 30
Masukkan stack ke-4: 40
Masukkan stack ke-5: 50
Elemen pada stack yaitu: [10, 20, 30, 40, 50]
Elemen teratas stack yaitu: 50

Hapus berapa stack? 3
Stack yang dihapus: 50
Stack yang dihapus: 40
Stack yang dihapus: 30
```

```
Elemen stack saat ini yaitu: [10, 20]
Elemen teratas saat ini yaitu: 20
```

7.2.4. Operator Search()

Operator search pada *stack* yaitu operator untuk mencari elemen yang paling atas pada stack. Operator jenis ini menerapkan algoritma *sequential search*, dimana elemen tidak diurutkan terlebih dahulu sebelum dicari. Dengan memodifikasi kode program pada subbab 7.2.2, maka kode program untuk operator search yaitu sebagai berikut:

```
1  import java.util.*;
2
3  public class App {
4
5    public static void main(String[] args) throws Exception {
6      Scanner sc = new Scanner(System.in);
7      System.out.print("Masukkan banyak tumpukkan:");
8      int tumpukkan = sc.nextInt();
9      int a[] = new int[tumpukkan];
10     Stack<Integer> stack = new Stack<>();
11
12     for (int i = 0; i < tumpukkan; i++) {
13       System.out.print("Masukkan stack ke-" + (i+1) + ": ");
14       a[i] = sc.nextInt();
15       stack.push(a[i]);
16     }
17
18     System.out.println("Elemen pada stack yaitu: " + stack);
19
20     System.out.println();
21     System.out.print("Elemen yang ingin dicari: ");
22     int cari = sc.nextInt();
23
24     if (stack.search(cari) == -1) {
25       System.out.println("Elemen tidak ditemukan!");
26     } else {
27       System.out.println("Elemen " + cari + " ditemukan!");
28     }
29   }
30 }
```

Hasil outputnya yaitu sebagai berikut:

```
Masukkan banyak tumpukan: 5
Masukkan stack ke-1: 10
Masukkan stack ke-2: 20
Masukkan stack ke-3: 30
Masukkan stack ke-4: 40
Masukkan stack ke-5: 50
Elemen pada stack yaitu: [10, 20, 30, 40, 50]

Elemen yang dicari: 40
Elemen 40 ditemukan!
```

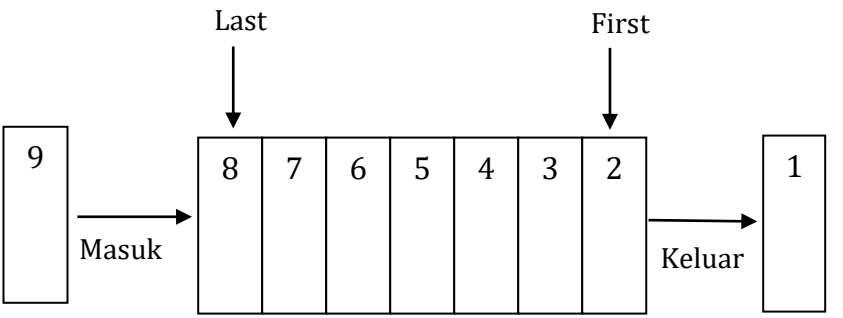
7.3. Latihan Soal

1. Modifikasilah kode porgram pada subbab 7.2.1 dengan memberikan trigger saat mengisi *stack*. Trigger tersebut terjadi apabila stack yang diisi melebihi batas kapasitas *stack* yang telah ditentukan.
2. Buatlah *stack* yang berisi elemen berupa 10 orang nama mahasiswa di kelas Anda secara acak, kemudian hapus 4 orang dan masukkan kembali 2 orang nama mahasiswa lainnya. Tampilkan *list* awal *stack* beserta elemen teratasnya, dimulai dari pengisian 10 elemen awal, penghapusan 4 elemen, sampai dengan 2 elemen tambahan.
3. Diberikan sejumlah elemen integer ke dalam sebuah *stack*, kemudian *stack* itu perlu dibagi menjadi 4 bagian (jumlah isi stack selalu kelipatan 4, beri trigger apabila isi tidak kelipatan 4 maka program tidak akan dijalankan). Masukkan semua integer ke dalam *stack*, kemudian bagi 4 isi stack tersebut ke dalam 4 *stack* lain dan tampilkan keempat stack tersebut mulai dari top masing-masing *stack*. Misal diberikan 12 elemen maka dibagi 4 *stack*, masing-masing berisi 3 elemen.

BAB 8
QUEUE

8.1. *Queue*

Sesuai dengan katanya, *queue*, yang memiliki makna antrean. Berkebalikan dengan *stack*, *queue* diartikan sebagai penyimpanan dan pengambilan data yang diurutkan secara FIFO (*First In First Out*). FIFO yaitu data pertama yang disimpan akan diambil pertama kali. Ilustrasi mengenai queue ditunjukkan pada Gambar 8.1.

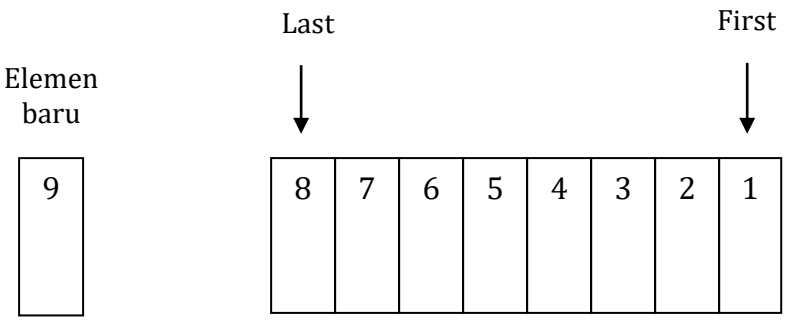


Gambar 8.1. Ilustrasi *Queue*

Berdasarkan Gambar 8.1, elemen pada *queue* hanya dapat ditambahkan dari sisi belakang dan dikeluarkan dari sisi depan, sehingga ada dua penunjuk elemen pada *queue*, yaitu *first* sebagai penunjuk elemen paling depan dan *last* sebagai penunjuk elemen paling belakang.

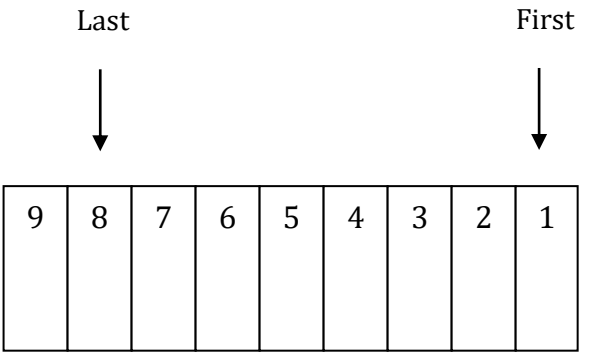
8.2. Operator *Queue*

Operator pada queue yaitu menambahkan elemen dari sisi belakang dan mengeluarkan elemen dari sisi depan. Aturan penambahan elemen *queue* diawali dengan penambahan elemen baru yang akan dimasukkan ke dalam *queue* dengan ilustrasi sebagai berikut:



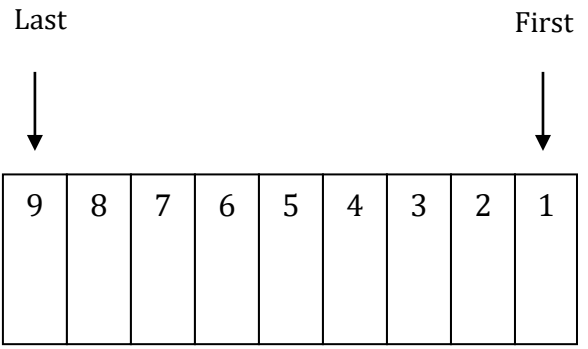
Gambar 8.2. Ilustrasi Penambahan Elemen Baru

Kemudian elemen baru dimasukkan ke dalam queue dengan ilustrasi sebagai berikut:



Gambar 8.3. Ilustrasi Memasukkan Elemen Baru ke dalam *Queue*

Karena terdapat elemen baru yaitu elemen bernomor 9, maka posisi last diubah menunjuk ke elemen 9 dengan ilustrasi sebagai berikut:



Gambar 8.4. Ilustrasi Pemindahan Petunjuk *Last*

Implementasi kode programnya yaitu sebagai berikut:

Langkah 1, siapkan beberapa deklarasi, seperti *scanner*, *array*, dan *queue* itu sendiri. Kode programnya yaitu sebagai berikut:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Scanner sc = new Scanner(System.in);
7       System.out.print("Masukkan banyak antrean:");
8       int antrean = sc.nextInt();
9       int a[] = new int[antrean];
10      Queue<Integer> queue = new LinkedList<Integer>();
```

Berdasarkan kode di atas, kita akan membuat sebuah *queue* dengan *array* yang bisa diatur jumlah kapasitas *queue*nya. Pada baris ke-10, deklarasi *queue* dibuat sesuai dengan tipe datanya, dengan format sebagai berikut:

```
Queue<tipe_data> queue = new LinkedList<Integer>();
```

Namun perlu dicatat bahwa apabila ingin mengubah tipe data *queue*, maka tipe data untuk semua deklarasi perlu disesuaikan juga, kecuali untuk baris 8 dimana baris tersebut mendeklarasikan jumlah kapsitas *queue* yang diinginkan.

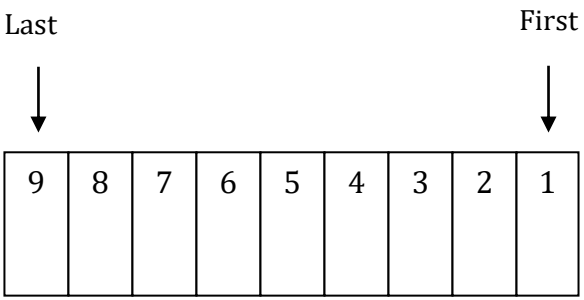
Langkah 2, buat perulangan untuk mengisikan *queue*. Kode programnya yaitu sebagai berikut:

```
    for (int i = 0; i < antrean; i++) {
      System.out.print("Masukkan antrean ke-" + (i+1) + ": ");
      a[i] = sc.nextInt();
      queue.add(a[i]);
    }

    System.out.println("Elemen pada antrean yaitu: " + queue);
    System.out.println("Antrean pertama yaitu: " + queue.peek());

  }
}
```

Pada kode di atas, pengisian *queue* dilakukan dengan perulangan sampai dengan sejumlah kapasitas *queue* yang telah ditentukan. Nilai pada *queue* ditentukan sesuai keinginan, kemudian operator *add* akan bekerja sebagai pengisian *queue*. Setelah langkah ini, penulisan kode program selesai. Berikut contoh hasil *output* dari program ini:

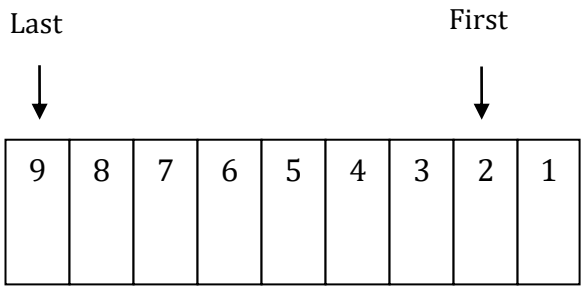
```
Masukkan banyak antrean: 5
Masukkan antrean ke-1: 35
Masukkan antrean ke-2: 20
Masukkan antrean ke-3: 25
Masukkan antrean ke-4: 10
Masukkan antrean ke-5: 40
Elemen pada stack yaitu: [35, 20, 25, 10, 40]
Antrean pertama yaitu: 35
```

Aturan pengeluaran elemen *queue* diawali dengan kondisi awal *queue* yang berisi elemen dengan ilustrasi sebagai berikut:



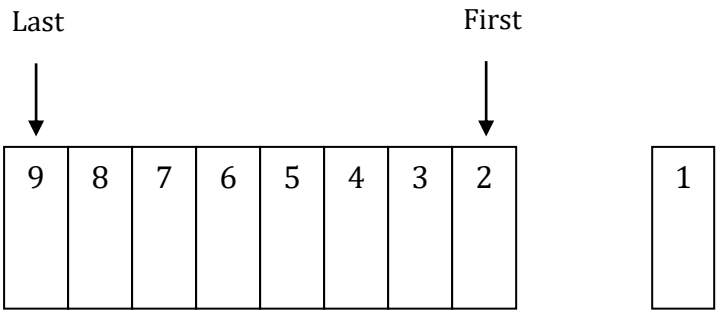
Gambar 8.5. Ilustrasi Kondisi Awal *Queue*

Kemudian penunjuk awal pada queue dipindah ke elemen di belakang elemen terdepan, yaitu elemen 2 dengan ilustrasi sebagai berikut:



Gambar 8.6. Ilustrasi Memindahkan posisi First Pada *Queue*

Kemudian elemen paling depan dikeluarkan dengan ilustrasi sebagai berikut:



Gambar 8.7. Ilustrasi Pengeluaran Elemen Paling Depan

Implementasi kode programnya yaitu sebagai berikut:

Langkah 1, siapkan beberapa deklarasi, seperti *scanner*, *array*, dan *queue* itu sendiri. Kode programnya yaitu sebagai berikut:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Scanner sc = new Scanner(System.in);
7       System.out.print("Masukkan banyak antrean:");
8       int antrean = sc.nextInt();
9       int a[] = new int[antrean];
10      Queue<Integer> queue = new LinkedList<Integer>();
```

Langkah 2, buat perulangan untuk mengisikan *queue*. Kode programnya yaitu sebagai berikut:

```
1       for (int i = 0; i < antrean; i++) {
2         System.out.print("Masukkan antrean ke-" + (i+1) + ": ");
3         a[i] = sc.nextInt();
4         queue.add(a[i]);
5       }
6
7       System.out.println("Elemen pada antrean yaitu: " + queue);
8       System.out.println("Antrean pertama yaitu: " + queue.peek());
```

Langkah 3, buatlah proses pengeluaran elemen *queue*. Kode programnya yaitu sebagai berikut:

```
1       System.out.println();
2       System.out.print("Keluarkan berapa elemen? ");
3       int keluar = sc.nextInt();
4
5       for (int i = 0; i < keluar; i++) {
6         System.out.println("Elemen yang dikeluarkan: " + queue.poll());
7       }
8
9       System.out.println("Elemen antrean setelah dikeluarkan yaitu: " + queue);
10      System.out.println("Antrean terdepan yaitu: " + queue.peek());
11    }
12  }
```

Berdasarkan kode program di atas, proses penghapusan elemen *queue* ditentukan sesuai keinginan. Misalkan di awal mengisikan 5 elemen *queue* dan kemudian ingin menghapus 2 elemen, maka *output* dari program tersebut yaitu:

```
Masukkan banyak antrean: 5
Masukkan antrean ke-1: 35
Masukkan antrean ke-2: 20
Masukkan antrean ke-3: 25
Masukkan antrean ke-4: 10
Masukkan antrean ke-5: 40
Elemen pada stack yaitu: [35, 20, 25, 10, 40]
Antrean pertama yaitu: 35

Keluarkan berapa elemen? 2
Elemen yang dikeluarkan: 35
Elemen yang dikeluarkan: 20
Elemen antrean setelah dikeluarkan yaitu: [25, 10, 40]
Antrean terdepan yaitu: 25
```

Ketiga elemen yang dihapus yaitu 2 elemen terdepan pada queue, yaitu 35 dan 20, sehingga tersisa 3 elemen pada antrean yaitu 25, 10, dan 40.

Dari ilustrasi tersebut, apabila pengeluaran elemen sejumlah lebih dari kapasitas awal *queue*, semisal di awal diatur 5 kapasitas elemen *queue* lalu mengeluarkan 6 elemen, maka outputnya menjadi sebagai berikut:

```
Masukkan banyak antrean: 5
Masukkan antrean ke-1: 35
Masukkan antrean ke-2: 20
Masukkan antrean ke-3: 25
Masukkan antrean ke-4: 10
Masukkan antrean ke-5: 40
Elemen pada stack yaitu: [35, 20, 25, 10, 40]
Antrean pertama yaitu: 35

Keluarkan berapa elemen? 6
Elemen yang dikeluarkan: 35
Elemen yang dikeluarkan: 20
Elemen yang dikeluarkan: 25
Elemen yang dikeluarkan: 10
Elemen yang dikeluarkan: 40
Elemen yang dikeluarkan: null
Elemen antrean setelah dikeluarkan yaitu: []
Antrean terdepan yaitu: null
```

Berbeda dengan *stack* yang apabila akan menghasilkan *exception*, pada *queue* yang terjadi program tetap dapat berjalan, akan tetapi setelah kelima elemen dikeluarkan, *queue* akan menjadi elemen kosong dan elemen yang dikeluarkan selanjutnya menghasilkan nilai null.

Selain menggunakan operator poll, pengeluaran elemen queue juga dapat dilakukan dengan operator remove, dimana operator remove bergantung pada elemen mana yang hendak dikeluarkan, tanpa memerdulikan urutan *queue*. Untuk langkah-langkah pembuatan kodenya, secara umum langkah 1 dan 2 masih sama, hanya saja cukup memodifikasi langkah 3 menjadi sebagai berikut:

```
1       System.out.println();
2       System.out.print("Elemen yang ingin dikeluarkan? ");
3       int keluar = sc.nextInt();
4       queue.remove(keluar);
5
6       System.out.println("Elemen antrean setelah dikeluarkan yaitu: " + queue);
7       System.out.println("Antrean terdepan yaitu: " + queue.peek());
8      }
9   }
```

Output yang dihasilkan yaitu:

```
Masukkan banyak antrean: 5
Masukkan antrean ke-1: 35
Masukkan antrean ke-2: 20
Masukkan antrean ke-3: 25
Masukkan antrean ke-4: 10
Masukkan antrean ke-5: 40
Elemen pada stack yaitu: [35, 20, 25, 10, 40]
Antrean pertama yaitu: 35

Elemen yang ingin dikeluarkan? 25
Elemen antrean setelah dikeluarkan yaitu: [35, 20, 10, 40]
Antrean terdepan yaitu: 35
```

Jika terdapat dua atau lebih elemen yang sama dan yang hendak dikeluarkan adalah elemen yang sama tersebut, maka yang dikeluarkan yaitu yang terletak pada posisi paling depan. Contoh outputnya sebagai berikut:

```
Masukkan banyak antrean: 5
Masukkan antrean ke-1: 40
Masukkan antrean ke-2: 30
Masukkan antrean ke-3: 25
Masukkan antrean ke-4: 40
Masukkan antrean ke-5: 25
Elemen pada stack yaitu: [40, 30, 25, 40, 25]
Antrean pertama yaitu: 40

Elemen yang ingin dikeluarkan? 25
Elemen antrean setelah dikeluarkan yaitu: [40, 30, 40, 25]
Antrean terdepan yaitu: 40
```

Dari contoh di atas, elemen 25 ada di antrean ke-3 dan ke-5, sementara yang hendak dikeluarkan yaitu elemen 25. Maka elemen 25 yang dikeluarkan yaitu elemen 25 di antrean ke-3, dimana posisinya lebih depan daripada elemen 25 di antrean ke-5.

8.3. *Priority Queue*

Pada subbab 8.2, kode program menggunakan kelas *LinkedList*, dimana urutan elemen sesuai dengan antrean yang mengikuti aturan FIFO. Sementara *Priority Queue* atau antrean prioritas yaitu kelas pada *library Queue*, dimana elemen dari queue yang dimasukkan akan diurutkan secara naik, dan posisi antrean pertama yaitu elemen dengan nilai terkecil (untuk tipe *integer* dan riil) ataupun elemen dengan huruf alfabetis paling awal yaitu "A" (untuk tipe *string*). Dengan sedikit modifkasi kode program pada subbab 8.2, berikut kode untuk *priority queue*:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Scanner sc = new Scanner(System.in);
7       System.out.print("Masukkan banyak antrean:");
8       int antrean = sc.nextInt();
9       int a[] = new int[antrean];
10      Queue<Integer> queue = new PriorityQueue<Integer>();
11
12     for (int i = 0; i < antrean; i++) {
13        System.out.print("Masukkan antrean ke-" + (i+1) + ": ");
14        a[i] = sc.nextInt();
15        queue.add(a[i]);
16      }
17
18      System.out.println("Elemen pada antrean yaitu: " + queue);
19      System.out.println("Antrean pertama yaitu: " + queue.peek());
20
21      System.out.println();
22      System.out.print("Keluarkan berapa elemen? ");
23      int keluar = sc.nextInt();
24
25      for (int i = 0; i < keluar; i++) {
26        System.out.println("Elemen yang dikeluarkan: " + queue.poll());
27      }
28
29      System.out.println("Elemen antrean setelah dikeluarkan yaitu: " + queue);
30      System.out.println("Antrean terdepan yaitu: " + queue.peek());
31    }
32  }
```

Contoh *output* yang dihasilkan yaitu:

```
Masukkan banyak antrean: 5
Masukkan antrean ke-1: 35
Masukkan antrean ke-2: 20
Masukkan antrean ke-3: 25
Masukkan antrean ke-4: 10
Masukkan antrean ke-5: 40
Elemen pada stack yaitu: [10, 20, 25, 35, 40]
Antrean pertama yaitu: 10

Keluarkan berapa elemen? 2
Elemen yang dikeluarkan: 10
Elemen yang dikeluarkan: 20
Elemen antrean setelah dikeluarkan yaitu: [25, 35, 40]
Antrean terdepan yaitu: 25
```

8.4. Latihan Soal

1. Modifikasilah kode porgram pada subbab 8.2 dengan memberikan trigger saat mengisi *queue*. Trigger tersebut terjadi apabila stack yang diisi melebihi batas kapasitas *queue* yang telah ditentukan.
2. Buatlah *queue* yang berisi elemen berupa 10 orang nama mahasiswa di kelas Anda secara acak, kemudian hapus 4 orang sesuai urutan antrean dan masukkan kembali 2 orang nama mahasiswa lainnya. Tampilkan daftar *queue* nya beserta nama yang terdapat pada antrean pertama.
3. Buatlah *queue* yang berisi elemen berupa 10 orang nama mahasiswa di kelas Anda secara acak, kemudian hapus 4 orang dengan memilih nama yang akan dikeluarkan dari antrean. Tampilkan daftar *queue* nya beserta nama yang terdapat pada antrean pertama.
4. Ulangi soal nomor 2 dan 3 dengan menggunakan *Priority Queue.*

BAB 9
TUGAS AKHIR STRUKTUR DATA

9.1. Penilaian Formatif

Penllaian formatif pada matakuliah ini dilakukan setelah menempuh serangkaian materi Struktur Data pada buku ini. Penilaian dilakukan dengan membuat sebuah projek berupa sebuah program yang menerapkan struktur data, seperti *sort, search, stack, queue*, dan sebagainya.

Petunjuk pelaksanaan proyek akhir yaitu sebagai berikut:

1. Bagilah kelas Anda menjadi kelompok beranggotakan 2-3 orang.
2. Silakan membuat sebuah program apa saja yang menerapkan struktur data, setidaknya terdiri atas dua fitur struktur data.
3. Ada nilai tambahan apabila struktur data yang dipakai selain struktur data yang dibahas di buku ini.

Perencanaan jadwal proyek akhir:

1. Pekan ke-12 perkuliahan: Menentukan tim proyek beserta judul proyek yang diusulkan.
2. Pekan ke-13 perkuliahan: Mempresentasikan usulan judul proyek, dengan format: Judul proyek, deskripsi singkat proyek, dan jenis tipe data apa yang digunakan.
3. Pekan ke-14 perkuliahan: Pembimbingan proyek kepada dosen pengampu matakuliah (opsional).
4. Pekan ke-15 perkuliahan: Mempresentasikan kemajuan proyek. Usahakan progres minimum 60%.
5. Pekan ke-16 (UAS): Mempresentasikan hasil akhir proyek.

Jadwal tersebut dilakukan pada saat kegiatan tatap muka. Sementara kegiatan praktikum ditiadakan agar mahasiswa lebih memfokuskan penyelesaian proyek akhir tersebut.

Selain mempresentasikan hasil akhir proyek, Anda diminta untuk mengumpulkan laporan proyek sebagai tanda bukti (*hard copy* jilid biasa), beserta lembar progres mingguan yang dicantumkan pada halaman terakhir laporan proyek. Berikut ini format laporan dan lembar progres mingguan.

Format cover laporan proyek

LAPORAN PROYEK AKHIR
STRUKTUR DATA

<Judul Proyek>

<Logo Institusi>

diajukan untuk penilaian tugas akhir
matakuliah Struktur Data

NAMA ANGGOTA:

<NAMA dan NIM anggota kelompok>

Dosen Pengampu:
<Nama Dosen Pengampu Matakuliah beserta gelar>

<NAMA FAKULTAS/PRODI>
<NAMA INSTITUSI>
TAHUN …

Format isi Laporan

BAB 1: Pendahuluan

Bab ini berisi mengenai deskripsi dari proyek yang Anda buat, fitur apa saja yang ada dalam proyek tersebut, beserta batasan dari proyek tersebut.

BAB 2: Hasil

Bab ini berisi hasil akhir dari proyek yang Anda buat, yaitu berupa kode program dan output programnya.

BAB 3: Penutup

Bab ini berisi kesimpulan dari proyek yang Anda buat, beserta rencana keberlanjutan dari proyek tersebut (jika ada).

Format lembar progres proyek

LEMBAR PROGRES PROYEK

Nama Kelompok: 1. …… (NIM……)
2. …… (NIM……)
3. …… (NIM……)
Kelas : ……
Judul Proyek : ……
……
Dosen Pengampu: ……

No	Tanggal	Kegiatan	TTD Dosen Pengampu
1			
2			
3			
4			
5			

Keterangan: Tabel bisa ditambahkan sesuai dengan pengerjaan proyek
masing-masing tim.

9.2. Penilaian Sumatif

Penilaian sumatif pada matakuliah ini yaitu penilaian individu berupa soal-soal objektif sebanyak 128 butir soal untuk mengukur pemahaman materi yang telah diberikan selama 1 semester yang tertuang dalam buku ini. Silakan untuk mengerjakan soal tersebut paling lambat hingga pekan UAS.

1. Bagian dari ilmu pemrograman dasar untuk menyusun sebuah struktur penyimpanan data yang digunakan saat program berjalan yaitu ...
   A. algoritma
   B. basis data
   C. struktur data
   D. pemrograman

2. Ilmu untuk mengakses dan memelihara data yang disimpan di dalam *storage* yaitu ...
   A. algoritma
   B. basis data
   C. struktur data
   D. pemrograman

3. Struktur data meliputi *array* (larik) dan *record* (rekaman) pada berkas beruntun (*sequential file*).
   A. Benar
   B. Salah

4. Perbedaan antara struktur data dengan basis data yaitu pada tempat penyimpanan datanya, dimana struktur data disimpan di dalam memori dan basis data disimpan di dalam *storage*.
   A. Benar
   B. Salah

5. *Array* termasuk ke dalam struktur data non-linier.
   A. Benar
   B. Salah

6. Struktur data yang dapat diubah ukuran dan isi elemennya disebut struktur data linier dinamis.
   A. Benar B. Salah

7. Jenis struktur data linier dinamis yaitu sebagai berikut, kecuali ...
   A. *linked list* C. *queue*
   B. *sort* D. *tree*

8. Jenis struktur data non-linier yaitu ...
   A. *graph* C. A dan B benar
   B. *tree* D. A dan B salah

9. *Overflow* dan *underflow* merupakan kelemahan dari struktur data linier statis.
   A. Benar B. Salah

10. *Sort* dan *Search* merupakan jenis struktur data linier dinamis.
    A. Benar B. Salah

11. Berikut ini contoh penerapan struktur data *tree* yaitu ...
    A. penerapan klasifikasi analisis sentimen dengan metode C4.5
    B. sistem pakar pada penentuan gejala hama tanaman
    C. susunan struktur organisasi pada suatu perusahaan
    D. semua benar

12. Tatanan koin dari bawah ke atas merupakan penerapan struktur data ...
    A. *graph* C. *stack*
    B. *tree* D. *queue*

13. Barisan pembeli makanan di warteg merupakan penerapan struktur data ...
    A. *graph*
    B. *tree*
    C. *stack*
    D. *queue*

14. Strategi pengiriman pesanan barang ke alamat masing-masing dengan jarak tempuh tersingkat merupakan penerapan struktur data ...
    A. *graph*
    B. *tree*
    C. *stack*
    D. *queue*

15. Presensi secara alfabetis merupakan penerapan struktur data queue.
    A. Benar
    B. Salah

16. Strategi untuk melewati semua jalan di kelurahan tertentu tanpa melewati jalan yang sama merupakan penerapan struktur data graph.
    A. Benar
    B. Salah

17. Penomoran pada *array* sebagai pelabelan disebut ...
    A. indeks
    B. elemen
    C. variabel
    D. cell

18. *Array* yang hanya terdiri atas 1 baris dan beberapa kolom atau 1 kolom dan beberapa baris disebut *array* satu dimensi.
    A. Benar
    B. Salah

19. Penomoran indeks *array* untuk bahasa pemrograman Java diawali dengan 1.
    A. Benar
    B. Salah

20. Penulisan kode program yang tepat untuk membuat *array* bertipe integer sejumlah 8 elemen yaitu ...
    A. int [] array;
       array = new int [8];
    B. int [] array = new int [8];
    C. int array [];
       array = new int [8];
    D. semua benar

Perhatikan kode di bawah ini untuk soal nomor 21 sampai 23.

```
1   int nomor_undian[] = new int[6];
2
3   nomor_undian[0] = 205;
4   nomor_undian[1] = 194;
5   nomor_undian[3] = 247;
6   nomor_undian[5] = 300;
7
8   System.out.println(nomor_undian[1]);
9   System.out.println(nomor_undian[4]);
10  System.out.println(nomor_undian[6]);
```

21. *Output* untuk baris 8 yaitu ...
    A. 205
    B. 194
    C. 0
    D. ArrayIndexOutOfBoundsException

22. *Output* untuk baris 10 yaitu ...
    A. 205
    B. 194
    C. 0
    D. ArrayIndexOutOfBoundsException

23. *Output* untuk baris 9 yaitu null.

A. Benar B. Salah

24. Perhatikan kode di bawah ini:

```
1 String kota[] = new String[4];
2
3 kota[0] = “Lumajang”;
4 kota[1] = “Jember”;
5 kota[3] = “Banyuwangi”;
6
7 System.out.println(kota[2]);
```

Output untuk baris 7 yaitu ...

A. Lumajang C. null

B. Banyuwangi D. Jember

25. Perhatikan kode di bawah ini:

```
1 String ibukota_prov[] = {“Yogyakarta”,
  “Serang”, “Pekanbaru”, “Kendari”, “Kupang”};
2
3 System.out.println(ibukota_prov[3]);
```

Output untuk baris 3 yaitu Pekanbaru.

A. Benar B. Salah

26. Perhatikan kode di bawah ini:

```
1 String ibukota_prov[] = {“Yogyakarta”,
  “Serang”, “Pekanbaru”, “Kendari”, “Kupang”};
2
3 System.out.println(ibukota_prov[5]);
```

Output untuk baris 5 yaitu ArrayIndexOutOfBoundsException.

A. Benar B. Salah

27. *Array* multi dimensi yaitu array yang berisi beberapa baris dan beberapa kolom.
    A. Benar B. Salah

28. *Array* yang memiliki jumlah kolom berbeda-beda di tiap barisnya yaitu ...
    A. *array* satu dimensi C. *jagged array*
    B. *array* dua dimensi D. *array* multi dimensi

29. Perhatikan kode di bawah ini:

```
1  import java.util.Scanner;
2
3  public class App{
4  public static void main(String[] args) throws Exeption {
5
6    int matriks [][] = new int[7][10];
```

    Jumlah elemen yang dapat diisi pada *array* matriks yaitu ...
    A. 70 C. 10
    B. 17 D. 7

30. Perhatikan kode di bawah ini:

```
1  import java.util.Scanner;
2
3  public class App{
4  public static void main(String[] args) throws Exeption {
5
6    int matriks [][][][][] = new int[3][2][5][4][1];
```

    Jumlah elemen yang dapat diisi pada *array* matriks yaitu 60.
    A. Benar B. Salah

31. Perhatikan kode di bawah ini:

```
1   public class App{
2   public static void main(String[] args) throws Exeption {
3
4     int [][] angka = new int [3][];
5     angka[0] = new int [2];
6     angka[1] = new int [6];
7     angka[2] = new int [4];
8
9     for (int baris = 0; baris < angka.length; baris++) {
10      int nilaiBaris = (baris+1)*3;
11      for (int kolom = 0; kolom < angka[baris].length; kolom++) {
12        angka[baris][kolom] = nilaiBaris;
13      }
14    }
15
16    for (int baris = 0; baris < angka.length; baris++) {
17      for (int kolom = 0; kolom < angka[baris].length; kolom++) {
18        System.out.print(angka[baris][kolom]);
19      }
20      System.out.print("\n");
21    }
22  }
```

Output dari program di atas yaitu:

3 3
6 6 6 6 6 6
9 9 9 9

A. Benar B. Salah

32. *Array* empat dimensi berikut yang dapat diisi sebanyak 100 elemen yaitu ...
    A. int matriks = new matriks [2][5][2][5]
    B. int matriks = new matriks [10][1][2][5]
    C. int matriks = new matriks [1][1][1][100]
    D. semua benar

33. Proses untuk memanggil diri sendiri yang dilakukan oleh fungsi atau prosedur yaitu ...
    A. basis
    B. array
    C. perulangan
    D. rekursif

34. Kode yang berperan sebagai titik henti dari proses rekursif yaitu ...
    A. basis
    B. array
    C. perulangan
    D. rekursif

35. Perintah public static int variabel (int a, int b) merupakan perintah untuk membuat fungsi.
    A. Benar
    B. Salah

36. Dalam pembuatan prosedur, diperlukan perintah *return* di dalamnya.
    A. Benar
    B. Salah

37. Perhatikan langkah-langkah berikut:
    a. Menentukan kondisi basis
    b. Menentukan kondisi rekursif
    c. Membuat fungsi / prosedur
    d. Memanggil fungsi / prosedur rekursif

    Urutan langkah yang tepat untuk membuat kode program rekursif yaitu ...
    A. c – b – a – d
    B. a – b – c – d
    C. c – a – b – d
    D. d – c – a – b

Perhatikan kode rekursif berikut untuk soal nomor 38 sampai 40.

```
1   public class App {
2
3     public static int sum (int n) {
4
5       if (n <= 1) {
6         return 1;
7       }
8
9       else {
10        return (n + sum(n-1));
11      }
12    }
13
14    public static void main(String[] args) throws Exception {
15      int hasil = sum(10);
16      System.out.println(hasil);
17    }
18  }
```

38. Fungsi if pada baris 5 sampai 7 berfungsi sebagai blok basis rekursif.
    A. Benar B. Salah

39. Pembuatan rekursif pada kode di atas yaitu dengan menggunakan prosedur.
    A. Benar B. Salah

40. Hasil *output* yang muncul apabila program dijalankan yaitu ...
    A. 10 C. 55
    B. 11 D. 110

    Perhatikan kode rekursif berikut untuk soal nomor 41 sampai 44.

```
1   public class App {
2
3     public static void rekursif(int [] arr, int a, int b, int n, int i) {
4       if (i == 0) {
5         arr[i] = a;
6         sum(arr, a, b, n, i+1);
7       }
8
9       else {
10        if (i < n) {
11          arr[i] = arr[i-1] + b;
12          sum(arr, a, b, n, i+1);
13        }
14      }
15    }
16
17    public static void main(String[] args) throws Exception {
18      int i = 0;
19      int arr[] = new int[10];
20      rekursif(arr, 2, 6, 10, 0);
21
22      for (i = 0; i < arr.length; i++) {
23        System.out.print(arr[i] + " ");
24      }
25    }
26  }
```

41. Nama rekursif yang tepat untuk kode di atas yaitu ...
    A. membuat barisan deret aritmatika
    B. membuat barisan Fibonacci
    C. membuat barisan deret geometri
    D. membuat barisan deret segitiga

42. Kode di atas blok rekursif dimulai dari baris 4 sampai 7.
    A. Benar B. Salah

43. *Output* yang dihasilkan untuk kode di atas yaitu 2 8 14 20 26 32 38 44 50 56.
    A. Benar B. Salah

44. Kondisi basis dari kode rekursif tersebut yaitu ...
    A. jika i != 0
    B. jika i < n
    C. jika i == 0
    D. jika i >= n

Perhatikan kode rekursif berikut untuk soal nomor 45 sampai 48.

```
1   public class App {
2
3     public static void rekursif(int [] arr, int a, int b, int n, int i) {
4       if (i == 0) {
5         arr[i] = a;
6         sum(arr, a, b, n, i+1);
7       }
8
9       else {
10        if (i < n) {
11          arr[i] = arr[i-1] * b;
12          sum(arr, a, b, n, i+1);
13        }
14      }
15    }
16
17    public static void main(String[] args) throws Exception {
18      int i = 0;
19      int arr[] = new int[10];
20      rekursif(arr, 3, 4, 5, 0);
21
22      for (i = 0; i < arr.length; i++) {
23        System.out.print(arr[i] + " ");
24      }
25    }
26  }
```

45. Nama rekursif yang tepat untuk kode di atas yaitu ...
    A. membuat barisan deret aritmatika
    B. membuat barisan Fibonacci
    C. membuat barisan deret geometri
    D. membuat barisan deret segitiga

46. Variabel yang berperan sebagai nilai awal yaitu variabel b.
    A. Benar
    B. Salah

47. Variabel n berperan sebagai penentu banyaknya elemen pada array.
    A. Benar B. Salah

48. *Output* yang dihasilkan dari kode rekursif tersebut yaitu ...
    A. 3 12 48 192 C. 3 9 81 6561 43046721
    B. 3 12 48 192 768 D. 3 9 27 81 243

49. Pengurutan / *sort* yang dilakukan secara menurun yaitu *ascending*.
    A. Benar B. Salah

50. Algoritma pengurutan yang menggunakan elemen kunci untuk membandingkan elemen sebelah dan menggeser elemen yang lebih besar dari kunci yaitu ...
    A. *Bubble Sort* C. *Insertion Sort*
    B. *Selection Sort* D. *Merge Sort*

51. Algoritma pengurutan yang mencari nilai terkecil atau terbesar untuk selanjutnya ditempatkan ke tempat terdepan yaitu ...
    A. *Bubble Sort* C. *Insertion Sort*
    B. *Selection Sort* D. *Merge Sort*

52. Algoritma pengurutan yang dilakukan dengan membandingkan dua elemen secara terus menerus hingga pengurutan selesai yaitu *Bubble Sort*.
    A. Benar B. Salah

53. Secara umum, algoritma *insertion sort* lebih efisien jika dibandingkan dengan *bubble sort*.
    A. Benar B. Salah

54. Jumlah proses penelusuran pada algoritma *selection sort* sama dengan *insertion sort*.
    A. Benar B. Salah

55. Diketahui ada 20 elemen integer yang akan diurutkan. Jika menggunakan *selection sort*, maka jumlah proses penelusuran hingga terurut yaitu ...
    A. 210 C. 20
    B. 190 D. 361

56. Diketahui ada 17 elemen integer yang akan diurutkan. Jika menggunakan *bubble sort*, maka jumlah proses penelusuran hingga terurut yaitu ...
    A. 256 C. 17
    B. 289 D. 153

Perhatikan *array* berikut untuk soal nomor 57 sampai 60.

305	127	440	671	109	500	998	305	839	276

57. Jika diurutkan secara *ascending*, yang terjadi pada langkah awal algoritma *insertion sort* yaitu ...
    A. menukar elemen 305 dan 109
    B. menukar elemen 127 dan 440
    C. menukar elemen 305 dan 127
    D. menukar elemen 671 dan 109

58. Jika elemen kunci terletak pada 305, maka langkah yang terjadi untuk pengurutan *ascending* pada *insertion sort* adalah tidak terjadi pertukaran elemen dengan 998.
    A. Benar B. Salah

59. Untuk pengurutan *ascending* dengan metode *selection sort*, maka langkah awalnya yaitu menukar elemen 305 dengan 127.
    A. Benar B. Salah

60. Untuk pengurutan *ascending* dengan metode *bubble sort*, susunan elemen setelah 9 kali penelusuran yaitu ...
    A. 127 – 305 – 440 – 671 – 109 – 500 – 998 – 305 – 839 – 276
    B. 127 – 305 – 440 – 109 – 500 – 671 – 998 – 305 – 839 – 276
    C. 127 – 305 – 440 – 109 – 500 – 671 – 305 – 839 – 276 – 998
    D. 109 – 127 – 276 – 305 – 305 – 440 – 500 – 671 – 839 – 998

Perhatikan *array* berikut untuk soal nomor 61 sampai 64.

475	302	775	160	350	567	499	302	667	380

61. Untuk pengurutan *descending* dengan metode *insertion sort*, maka langkah awalnya yaitu tidak terjadi penukaran antara elemen 475 dengan 302.
    A. Benar B. Salah

62. Elemen yang diletakkan paling depan untuk pengurutan *descending* yaitu 775. Pada algoritma *selection sort*, itu terjadi pada saat penelusuran yang ke ...
    A. 1 C. 3
    B. 2 D. 4

63. Pada algoritma *insertion sort* secara *descending*, penukaran elemen terjadi ketika elemen kunci terletak pada angka ...

A. 775, 160, 350, 567, 499, 302, 380
B. 775, 160, 350, 567, 499, 302, 667
C. C. 775, 350, 567, 499, 302, 667, 380
D. 775, 160, 567, 499, 302, 667, 380

64. Pada algoritma *bubble sort* secara *descending*, proses penelusuran ke-2 terjadi pertukaran antara elemen 302 dengan 775.
A. Benar B. Salah

65. Pengurutan yang menggunakan titik pivot sebagai acuannya yaitu metode *merge sort*.
A. Benar B. Salah

66. Pertukaran elemen pada metode *quick sort* secara *ascending* akan terjadi apabila memenuhi syarat berikut, yaitu ...
A. i > pivot, dan j >= pivot, atau i dan j saling bertemu sehingga pivot berada di antara i dan j
B. i < pivot, dan j >= pivot, atau dan j saling bertemu sehingga pivot berada di antara i dan j
C. i < pivot, dan j < pivot, atau i dan j saling bertemu sehingga pivot berada di antara i dan j
D. i > pivot, dan j < pivot, atau i dan j saling bertemu sehingga pivot berada di antara i dan j

Perhatikan *array* berikut untuk soal nomor 67 sampai 69.

384	631	273	400	926	184	708	419	857	774

67. Jika pivot ada di titik elemen 774 dan pengurutan dilakukan secara *ascending*, maka titik i akan berhenti bergeser bilamana i ada di elemen ...
A. 631 C. 400
B. 273 D. tidak ada yang benar

68. Jika pivot ada di titik elemen 774 dan pengurutan dilakukan secara *ascending*, maka titik j akan berhenti bergeser bilamana j ada di elemen ...
    A. 857 C. 184
    B. 419 D. 708

69. Jika pivot ada di titik elemen 774 dan pengurutan dilakukan secara *ascending*, pertukaran elemen pertama kali terjadi jika titik i berada pada 926 dan j berada pada 419.
    A. Benar B. Salah

Perhatikan *array* berikut untuk soal nomor 70 sampai 72.

384	631	273	400	926	184	708	419	857	774

70. Jika pivot ada di titik elemen 774 dan pengurutan dilakukan secara *descending*, maka titik i akan berhenti bergeser bilamana i ada di elemen 631.
    A. Benar B. Salah

71. Jika pivot ada di titik elemen 774 dan pengurutan dilakukan secara *descending*, maka titik j akan berhenti bergeser bilamana j ada di elemen 857.
    A. Benar B. Salah

72. Jika pivot ada di titik elemen 774 dan pengurutan dilakukan secara *descending*, maka pertukaran elemen pertama kali terjadi jika ...
    A. i = 631 dan j = 857 C. i = 631 dan j = 419
    B. i = 384 dan j = 857 D. i = 384 dan j = 419

73. Metode pengurutan yang dilakukan dengan membagi dua elemen *array* sama banyak secara terus menerus hingga menjadi satu elemen per bagian, kemudian

digabung kembali dengan kondisi nilai terurut sebelum disatukan yaitu ...

A. *bubble sort*
B. *insertion sort*
C. *quick sort*
D. *merge sort*

74. Jika terdapat 20 elemen bilangan acak dan ingin diurutkan dengan metode *merge sort*, maka pemisahan elemen hingga menjadi satu elemen per bagian dilakukan sebanyak 5 kali pemisahan.

A. Benar
B. Salah

75. Jumlah pemisahan elemen pada algoritma *merge sort* sama dengan jumlah penggabungan elemen.

A. Benar
B. Salah

76. Pada algoritma *merge sort*, apabila elemen dipidah sampai 7 kali pemisahan hingga membentuk satu elemen per bagian, maka jumlah elemen maksimum yang memungkinkan yaitu ...

A. 64
B. 65
C. 127
D. 128

Perhatikan *array* berikut untuk soal nomor 77 sampai 80.

768	215	380	489	663	280	489	573	378

77. Jumlah pemisahan elemen pada algoritma *merge sort* untuk array di atas dilakukan sebanyak 3 kali hingga menjadi satu elemen per bagian.

A. Benar
B. Salah

78. Hasil pemisahan pertama elemen pada algoritma *merge sort* untuk array di atas yaitu ...

A. [768, 215, 380, 489], [663, 280, 489, 573] [378]

B. [768, 215, 380], [489, 663, 280] [489, 573, 378]
C. [768, 215, 380, 489], [663, 280, 489, 573, 378]
D. [768, 215, 380, 489], [663, 280, 489], [573, 378]

79. Hasil penggabungan pertama elemen pada algoritma *merge sort* secara *ascending* untuk array di atas setelah dipisah satu elemen per bagian yaitu ...
A. [215, 768], [380, 489], [280, 663], [489, 573], [378]
B. [215, 280], [378, 380], [489, 489], [573, 663], [768]
C. [768], [663, 573], [489, 489], [380, 378], [280, 215]
D. [768], [215, 380], [489, 663], [280, 489], [573, 378]

80. Hasil penggabungan pertama elemen pada algoritma *merge sort* secara *descending* untuk array di atas setelah dipisah satu elemen per bagian, salah satunya yaitu [768, 215], [489, 380], [663, 280], [573, 489], [378].
A. Benar B. Salah

81. Metode pencarian dengan memeriksa kecocokan antara elemen yang dicari dengan elemen pada *array* yaitu ...
A. *binary search* C. *depth first search*
B. *sequential search* D. *jump search*

82. Pada algoritma *sequential search*, algoritma akan berhenti apabila elemen yang dicari lebih besar dari salah satu elemen pada *array*.
A. Benar B. Salah

83. Pada algoritma *sequential search*, elemen pada array boleh terurut maupun tidak terurut sebelum dilakukan pencarian.
A. Benar B. Salah

84. Metode *sequential search* serupa dengan algoritma pengurutan berikut, yaitu ...
    A. *merge sort*
    B. *quick sort*
    C. *insertion sort*
    D. *selection sort*

85. Metode pencarian yang dilakukan dengan dengan membagi *array* menjadi dua bagian secara terus-menerus, kemudian memeriksa elemen tengahnya yaitu ...
    A. *binary search*
    B. *sequential search*
    C. *depth first search*
    D. *jump search*

86. Metode *binary search* serupa dengan algoritma pengurutan berikut, yaitu ...
    A. *merge sort*
    B. *quick sort*
    C. *insertion sort*
    D. *selection sort*

87. Ciri dari algoritma *binary search* yaitu ...
    A. membagi *array* menjadi dua bagian
    B. elemen harus terurut
    C. serupa dengan algoritma *merge sort*
    D. semua benar

88. Pada algoritma *binary search*, elemen pada *array* harus terurut sebelum dilakukan pencarian, untuk menghindari proses pencarian yang tidak ditemukan.
    A. Benar
    B. Salah

Perhatikan *array* berikut untuk soal nomor 89 sampai 90.

379	289	449	529	509	839	779	699

89. Apabila nomor yang dicari yaitu 509, maka pada algoritma *sequential search*, dilakukan pembandingan sebanyak ... kali agar elemen dapat ditemukan.

A. 4
B. 5
C. 6
D. 7

90. Apabila nomor yang dicari yaitu 509, maka pada algoritma *binary search*, elemen dapat ditemukan pada elemen ke-5.
    A. Benar
    B. Salah

Perhatikan *array* berikut untuk soal nomor 91 sampai 93.

104	475	497	673	728	882	901	993

91. Apabila nomor yang dicari yaitu 475, maka pada algoritma *sequential search*, dilakukan pembandingan sebanyak 2 kali agar elemen dapat ditemukan.
    A. Benar
    B. Salah

92. Langkah pertama beserta pemilihan *subarray* yang tepat apabila nomor yang dicari yaitu 497 yaitu.
    A. Subarray 1: [104, 475, 497, 673]
       Subarray 2: [728, 882, 901, 993]
       Yang dipilh yaitu subarray 2
    B. Subarray 1: [104, 475, 497, 673]
       Subarray 2: [728, 882, 901, 993]
       Yang dipilh yaitu subarray 1
    C. Subarray 1: [728, 882, 901, 993]
       Subarray 2: [104, 475, 497, 673]
       Yang dipilh yaitu subarray 2
    D. Subarray 1: [728, 882, 901, 993]
       Subarray 2: [104, 475, 497, 673]
       Yang dipilih yaitu subarray 1

93. Apabila nomor yang dicari yaitu 728, maka pada algoritma *binary search*, *subarray* yang dipilih hingga langkah kedua yaitu subarray 2 dan subarray 2.2.

A. Benar B. Salah

Perhatikan *array* berikut untuk soal nomor 94 sampai 96.

790	714	592	314	314	314	314	116

94. Setelah *array* dibagi menjadi 2, yaitu *subarray* 1 untuk 4 elemen pertama dan *subarray* 2 untuk 4 elemen kedua, maka *subarray* yang dipilih dari *array* tersebut yaitu *subarray* 1 jika elemen yang dicari lebih besar atau sama dengan elemen terkanan, dan *subarray* 2 jika elemen yang dicari lebih kecil atau sama dengan elemen terkiri.
    A. Benar B. Salah

95. Apabila nomor yang dicari yaitu 314, maka pada algoritma *sequential search*, elemen yang dipilih yaitu elemen ke ...
    A. 4 C. 6
    B. 5 D. 7

96. Apabila nomor yang dicari yaitu 314, maka pada algoritma *binary search*, elemen *subarray* yang dipilih hingga akhir pencarian yaitu ...
    A. *subarray* 2, *subarray* 2.1, *subarray* 2.1.1
    B. *subarray* 2, *subarray* 2.2, *subarray* 2.2.1
    C. *subarray* 1, *subarray* 1.2, *subarray* 1.2.2
    D. semua benar

97. Struktur data *stack* menerapkan FIFO, yaitu elemen pertama yang masuk sekaligus elemen pertama yang keluar.
    A. Benar B. Salah

98. Diberikan array sebagai berikut: [92, 17, 30, 47, 66, 29, 57]. Yang berperan sebagai *top of stack* yaitu ...
    A. 92
    B. 47
    C. 30
    D. 57

99. Penambahan atau pengurangan elemen *stack* mengacu pada *top of stack*.
    A. Benar
    B. Salah

100. Diberikan *array* sebagai berikut: [47, 39, 81, 24]. Apabila ditambahkan elemen [37, 43], maka alur posisi *top of stack* yang tepat yaitu ...
    A. *Top* awal: 24, penambahan elemen [37, 43], *top* terbaru: 37
    B. *Top* awal: 47, penambahan elemen [37, 43], *top* terbaru: 37
    C. *Top* awal: 24, penambahan elemen [37, 43], *top* terbaru: 43
    D. *Top* awal: 47, penambahan elemen [37, 43], *top* terbaru: 43

101. Diberikan *array* sebagai berikut: [47, 39, 81, 24]. Apabila dua elemen dihapus, maka alur posisi *top of stack* yang tepat yaitu ...
    A. *Top* awal: 24, pemindahan *top*: 81, penghapusan elemen [24, 81]
    B. *Top* awal: 24, pemindahan *top*: 39, penghapusan elemen [24, 81]
    C. *Top* awal: 24, penghapusan elemen [24, 81], pemindahan *top*: 81
    D. *Top* awal: 24, penghapusan elemen [24, 81], pemindahan *top*: 39

102. Operator *stack* yang digunakan untuk menambah elemen *stack* yaitu push().
   A. Benar B. Salah

103. Operator *stack* yang digunakan untuk menampilkan *top of stack* yaitu ...
   A. push() C. peek()
   B. pop() D. search()

104. Operator search() digunakan untuk menghapus elemen *stack*.
   A. Benar B. Salah

105. *Library stack* pada Java terletak pada *library* java.util.
   A. Benar B. Salah

106. Format peulisan kode *stack* apabila isi *stack* [7.75, 0.032, 2.4, 9.0] yaitu ...
   A. stack<Integer> stack = new Stack<>();
   B. stack<Double> stack = new Stack<>();
   C. stack<Char> stack = new Stack<>();
   D. stack<String> stack = new Stack<>();

Perhatikan kode berikut untuk soal nomor 107 sampai 109:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Stack<Integer> stack = new Stack<>();
7       stack.push(35);
8       stack.push(20);
9       stack.push(40);
10      stack.push(40);
11      System.out.println(stack);
12      System.out.println(stack.peek());
13      System.out.println(stack.search(40));
14    }
15  }
```

107. *Output* yang dihasilkan pada baris 11 yaitu [40, 40, 20, 35].
   A. Benar
   B. Salah

108. *Output* yang dihasilkan pada baris 12 yaitu ...
   A. 35
   B. 20
   C. 40 (baris ke-9)
   D. 40 (baris ke-10)

109. *Output* yang dihasilkan pada baris 13 yaitu 1.
   A. Benar
   B. Salah

Perhatikan kode berikut untuk soal nomor 110 sampai 112:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Stack<Integer> stack = new Stack<>();
7       stack.push(75);
8       stack.push(60);
9       stack.push(10);
10      stack.push(55);
11      stack.push(55);
12      stack.push(95);
13
14      System.out.println(stack);
15      System.out.println(stack.peek());
16
17      int pop = 2;
18      for(int i = 0; i < pop; i++) {
19        stack.pop();
20      }
21
22      System.out.println(stack);
23      System.out.println(stack.peek());
24      System.out.println(stack.search(95));
25    }
26  }
```

110. *Output* yang dihasilkan pada baris 15 dan 23 yaitu ...
   A. 55 dan 95
   B. 95 dan 95

C. 95 dan 55
D. 55 dan 55

111. Apabila baris 17 diubah menjad int pop = 7, maka proses penghapusan tidak terjadi dan akan menghasilkan *exception* berupa *EmptyStackException*.
A. Benar B. Salah

112. *Output* yang dihasilkan pada baris 14, 22, dan 24 yaitu ...
A. [75, 60, 10, 55, 55, 95], [75, 60, 10, 55], 0
B. [95, 55, 55, 10, 60, 75], [55, 10, 60, 75], -1
C. [75, 60, 10, 55, 55, 95], [10, 55, 55, 95], -1
D. [75, 60, 10, 55, 55, 95], [75, 60, 10, 55], -1

113. Struktur data *queue* menerapkan ...
A. LIFO C. FILO
B. FIFO D. LILO

114. Diberikan *array* sebagai berikut: [38, 47, 91, 67, 77]. Yang berperan sebagai *first* dan *last* pada *queue* yaitu 77 untuk *first* dan 38 untuk *last*.
A. Benar B. Salah

115. Apabila elemen pada *queue* ditambahkan, maka terjadi pergeseran posisi *last*, sedangkan apabila elemen dikeluarkan, maka terjadi pergeseran posisi *first*.
A. Benar B. Salah

116. Diberikan *array* sebagai berikut: [69, 59, 10, 37]. Apabila ditambahkan elemen [37, 10], maka alur posisi *first* dan *last* pada *queue* yang tepat yaitu ...
A. *First* awal: 37, *Last* awal: 69, penambahan elemen [37, 10], *First* akhir: 37, *Last* akhir: 10
B. *First* awal: 37, *Last* awal: 69, penambahan elemen [37, 10], *First* akhir: 10, *Last* akhir: 37

C. *First* awal: 69, *Last* awal: 37, penambahan elemen [37, 10], *First* akhir: 37, *Last* akhir: 10
D. *First* awal: 69, *Last* awal: 37, penambahan elemen [37, 10], *First* akhir: 69, *Last* akhir: 10

117. Diberikan *array* sebagai berikut: [69, 59, 10, 37]. Apabila satu elemen dikeluarkan, maka alur posisi *first* dan *last* pada *queue* yang tepat yaitu ...
A. *First* awal: 69, *Last* awal: 37, pergeseran *First*: 59, elemen dikeluarkan [69], *Last* akhir: 10
B. *First* awal: 69, *Last* awal: 37, pergeseran *First*: 59, elemen dikeluarkan [69], *Last* akhir: 37
C. *First* awal: 69, *Last* awal: 37, elemen dikeluarkan [69], pergeseran *First*: 59, *Last* akhir: 10
D. *First* awal: 69, *Last* awal: 37, elemen dikeluarkan [69], pergeseran *First*: 59, *Last* akhir: 37

118. Operator untuk menambah elemen pada queue yaitu pull().
A. Benar B. Salah

119. Operator untuk mengeluarkan elemen pada queue yang sesuai dengan urutan FIFO yaitu ...
A. pull() C. remove()
B. add() D. peek()

120. Jika menggunakan format *queue PriorityQueue*, maka elemen *queue* akan terurut secara naik.
A. Benar B. Salah

121. Perhatikan kode berikut:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Queue<Integer> queue = new LinkedList<Integer>();
7       queue.add(65);
8       queue.add(90);
9       queue.add(30);
10      queue.add(45);
11      System.out.println(queue.peek());
12    }
13  }
```

Output yang dihasilkan pada baris 11 yaitu posisi *first* pada *queue*, yakni 45.

A. Benar B. Salah

Perhatikan kode berikut untuk soal nomor 122 sampai 124:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Queue<Integer> queue = new LinkedList<Integer>();
7       queue.add(65);
8       queue.add(90);
9       queue.add(30);
10      queue.add(45);
11      queue.add(50);
12      System.out.println(queue);
13      System.out.println(queue.peek());
14
15      int keluar = 2;
16      for(int i = 0; i < keluar; i++) {
17        queue.pull();
18      }
19      System.out.println(queue);
20      System.out.println(queue.peek());
21    }
22  }
```

122. Elemen yang dikeluarkan dari kode di atas yaitu ...
    A. [0, 65]
    B. [65, 90]
    C. [30, 50]
    D. [45, 50]

123. *Output* yang dihasilkan pada baris 20 yaitu ...
    A. 65
    B. 90
    C. 30
    D. 45

124. Jika baris 15 diubah menjadi int keluar = 6, maka *output* baris 20 yaitu null.
    A. Benar
    B. Salah

Perhatikan kode berikut untuk soal nomor 125 dan 126:

```
1   import java.util.*;
2
3   public class App {
4
5     public static void main(String[] args) throws Exception {
6       Queue<Integer> queue = new LinkedList<Integer>();
7       queue.add(65);
8       queue.add(90);
9       queue.add(75);
10      queue.add(70);
11      queue.add(65);
12
13      System.out.println(queue);
14      System.out.println(queue.peek());
15
16      int keluar = 90;
17      queue.remove(keluar);
18      System.out.println(queue);
19      System.out.println(queue.peek());
20    }
21  }
```

125. *Output* yang dihasilkan pada baris 18 yaitu ...
    A. [65, 70, 75, 65]
    B. [65, 75, 70, 65]
    C. [65, 90, 75, 70]
    D. [65, 70, 75, 90]

126. Jika baris 16 diubah menjadi int keluar = 65, maka hasil *output* untuk baris 19 yaitu 65.
    A. Benar B. Salah

Perhatikan kode berikut untuk soal nomor 127 dan 128:

```
1  import java.util.*;
2
3  public class App {
4
5    public static void main(String[] args) throws Exception {
6      Queue<Integer> queue = new PriorityQueue<Integer>();
7      queue.add(15);
8      queue.add(40);
9      queue.add(95);
10     queue.add(20);
11     queue.add(60);
12     System.out.println(queue);
13     System.out.println(queue.peek());
14
15     int keluar = 2;
16     for(int i = 0; i < keluar; i++) {
17       queue.pull();
18     }
19     System.out.println(queue);
20     System.out.println(queue.peek());
21   }
22 }
```

127. *Output* yang dihasilkan untuk baris 12 yaitu [15, 20, 40, 60, 95].
    A. Benar B. Salah

128. *Output* yang dihasilkan pada baris 20 yaitu ...
    A. 95 C. 60
    B. 15 D. 40

Referensi

Rosa, A. S. (2018a). *Logika Algoritma dan Pemrograman Dasar*. Modula.

Rosa, A. S. (2018b). *Struktur Data: Terapan Dalam Berbagai Bahasa Pemrograman: Pascal, C, C++, Dan Java*. Modula.

CATATAN: