-RSM-

KUG1C3
DASAR ALGORITMA & PEMROGRAMAN

Pengurutan (Sorting)

Pengurutan

- Merupakan suatu proses untuk mengatur dimana posisi suatu data seharusnya.
- 2 macam pengurutan:
  - pengurutan internal, yaitu pengurutan terhadap sekumpulan data yang disimpan dalam media internal komputer yang dapat diakses setiap elemennya secara langsung.
  - pengurutan eksternal, yaitu pengurutan data yang disimpan dalam memori sekunder, biasanya data bervolume besar sehingga tidak mampu untuk dimuat semuanya dalam memori.

Macam Algoritma Pengurutan Internal

- Counting Sort
- Maximum Sort
- Insertion Sort
- Bubble sort
- Shaker sort
- Heap Sort
- Shell sort
- Quick Sort
- Radix sort
- dll

Kamus dan Persoalan Umum

```
Kamus   :
    constant    Nmax :integer   = 100
    type TabInt :    array   [1..Nmax] of   integer
    N : integer   {indeks efektif, maksimum tabel yang terdefinisi, N< Nmax}
{ jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
     T : TabInt { tabel integer }
     N : integer {indeks efektif, 1 ≤ N ≤ Nmax+1}
```

- Persoalan:

  Diberikan sebuah tabel integer TabInt [1..N] yang isinya sudah terdefinisi. Tuliskan sebuah algoritma yang mengurutkan elemen tabel sehingga tersusun membesar: TabInt[1] ≤ TabInt[2] ≤ TabInt[3] ..... ≤ TabInt[N]

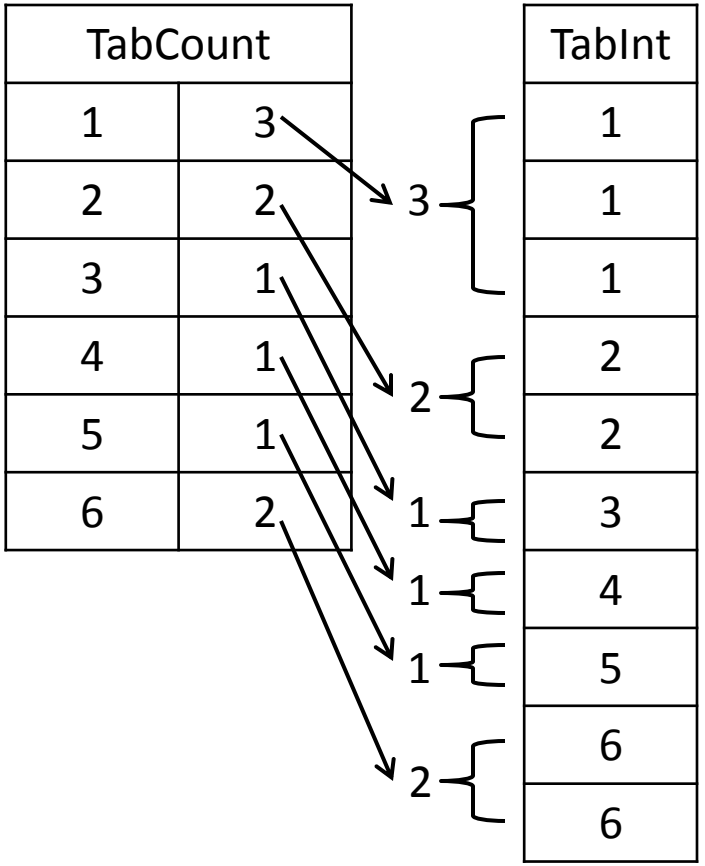
Counting Sort

- Merupakan pengurutan yang paling sederhana jika diketahui bahwa data yang akan diurut mempunyai daerah jelajah (*range) tertentu,* dan merupakan bilangan bulat, misalnya [Min..Max]
- Proses Umum:
  1. Sediakan array TabCount [Min..Max] yang diinisialisasi dengan nol, dan pada akhir proses TabCounti berisi banyaknya data pada tabel asal yang berharga i.
  2. Tabel dibentuk kembali dengan menuliskan kembali harga-harga yang ada.

Counting Sort: Ilustrasi

TabCount	
1	0
2	0
3	0
4	0
5	0
6	0

TabInt
1
3
2
4
5
6
1
6
2
1



Counting Sort: Algoritma

Procedure CountSORT (Input/Output TabInt: TabelInteger, Input N : integer)
{ mengurut tabel integer [1..N] dengan pencacahan}

```
Kamus   :
{ ValMin dan ValMax adalah batas minimum dan Maximum harga yg tersimpan
dalam TabInt, harus diketahui }
    TabCount :   array  [ValMin..Valmax] of  integer [0..NMax]
    i : integer   {indeks untuk traversal tabel }
    K : integer { jumlah elemen TabInt yang sudah diisi pada proses
pembentukan kembali }
Algoritma :
    {Inisialisasi TabCount }
    i traversal  [ValMin..ValMax]
        TabCount_i ← 0
    { Counting}
    i traversal  [1..N]
        TabCount_TabInti_i ←  TabCount_TabInt_i + 1
    { Pengisian kembali : TabInt_1 ≤ TabInt_2 ... ≤ TabInt_N }
    K ← 0
    i traversal  [ValMin..ValMax]
        if TabCount_i ≠ 0 then
            repeat TabCount_i times
                K ← K + 1
                TabInt_K ← i
```

Counting Sort: Catatan

- TabCount[TabInt[i]] dituliskan untuk menunjukkan bahwa indeks TabInt adalah i, dan TabInt[i] merupakan indeks dari TabCount.

Bubble Sort

- Merupakan salah satu bentuk pengurutan yang menerapkan pertukaran harga pada prosesnya.
- Idenya adalah gelembung air yang akan "mengapung": elemen berharga kecil akan "diapungkan", artinya diangkat ke atas melalui pertukaran.
- Proses dilakukan sebanyak N tahapan (yang dalam sorting disebut sebagai "pass"). Pada setiap Pass, tabel terdiri dari dua bagian: bagian yang sudah terurut yaitu [1..Pass-1] dan ide dasarnya adalah mengapungkan elemen ke “Pass" sampai pada tempatnya.

Bubble Sort: Proses

1. Untuk setiap dua elemen suksesif TabInt[K] dan TabInt[K-1], K [2..N], TabInt[K-1] ≤ TabInt[K], dengan demikian TabInt[K] harus ditukar dengan TabInt[K-1] jika sifat di atas tidak dipenuhi. Karena dilakukan secara berurutan, TabInt[1] berisi harga terkecil.
2. Untuk setiap dua elemen suksesif TabInt[K] dan TabInt[K-1], K [3..N], TabInt[K-1] ≤ TabInt[K], dengan demikian TabInt[K] harus ditukar dengan TabInt[K-1] jika sifat di atas tidak dipenuhi. Karena dilakukan secara berurutan, TabInt[1..2] terurut.
3. Untuk setiap dua elemen suksesif TabInt[K] dan TabInt[K-1], K [4..N], TabInt[K-1] ≤ TabInt[K], dengan demikian TabInt[K] harus ditukar dengan TabInt[K-1] jika sifat di atas tidak dipenuhi. Karena dilakukan secara berurutan, TabInt[1..3] terurut.

- .
- .

N-1 . Untuk setiap dua elemen suksesif TabInt[K] dan TabInt[K-1], K [N-1..N], TabInt[K-1] < TabInt[K], dengan demikian TabInt[K] harus ditukar dengan TabInt[K-1] jika sifat di atas tidak dipenuhi.

Karena dilakukan secara berurutan, TabInt[1..N-1] terurut.

TabInt [1..N] sudah terurut: TabInt[1] ≤ TabInt[2] ≤ TabInt[3] .... ≤ TabInt[N]

Bubble Sort: Ilustrasi

TabInt	6	5	6	3	1

Pass	K	TabInt					
1	5	6	5	6	3	1	TabInt[5] < TabInt[4]?
		6	5	6	1	3	Tukar
	4	6	5	6	1	3	TabInt[4] < TabInt[3]?
		6	5	1	6	3	Tukar
	3	6	5	1	6	3	TabInt[3] < TabInt[2]?
		6	1	5	6	3	Tukar
	2	6	1	5	6	3	TabInt[2] < TabInt[1]?
		1	6	5	6	3	Tukar
2	5	1	6	5	6	3	TabInt[5] < TabInt[4]?
		1	6	5	3	6	Tukar

Bubble Sort: Ilustrasi

Pass	K	TabInt					
2	4	1	6	5	3	6	TabInt[4] < TabInt[3]?
		1	6	3	5	6	Tukar
	3	1	6	3	5	6	TabInt[3] < TabInt[2]?
		1	3	6	5	6	Tukar
3	5	1	3	6	5	6	TabInt[5] < TabInt[4]?
		1	3	6	5	6	Tidak Tukar
	4	1	3	6	5	6	TabInt[4] < TabInt[3]?
		1	3	5	6	6	Tukar
4	5	1	3	5	6	6	TabInt[5] < TabInt[4]?
		1	3	5	6	6	Tidak Tukar

TabInt	1	3	5	6	6

Bubble Sort: Algoritma

Procedure BubleSort (Input/Output TabInt: TabelInteger, Input N : integer)
{ mengurut tabel integer [1..N] dengan bubble sort }

```
Kamus   :
    i,K: integer  {indeks untuk traversal tabel }
    Pass : integer      { tahapan pengurutan }
    Temp : integer   { Memorisasi untuk pertukaran harga }
Algoritma :
    Pass traversal   [1..N-1]
        K traversal [N..Pass+1]
            if (TabInt_K < TabInt_K-1 then
                Temp ← TabInt_K
                TabInt_K ← TabInt_K-1
                TabInt_K-1← Temp
{ TabInt[1..Pass]  terurut  :TabInt_1 ≤ TabInt_2 ≤ TabInt_3 ...  TabInt_Pass}
{Seluruh tabel terurut, karena Pass = N TabInt_1 ≤ TabInt_2 ≤ TabInt_3 ... ≤
 TabInt_N}
```

Insertion Sort

- Idenya adalah mencari tempat yang "tepat" untuk setiap elemen tabel, dengan cara sequential search, kemudian sisipkan elemen tabel yang diproses ke tempat yang seharusnya.

Insertion Sort: Proses

Proses dilakukan sebanyak N tahapan (Pass):

1. TabInt[1] dianggap sudah tepat tempatnya
2. TabInt[2] harus dicarikan tempat yang tepat pada TabInt[1..2], yaitu i. Sisipkan TabInt[2] pada TabInt[i]. TabInt [1..2] terurut membesar.
3. TabInt[3] harus dicarikan tempat yang tepat pada TabInt[1..3], yaitu i. Sisipkan TabInt[3] pada TabInt[i]. TabInt [1..3] terurut membesar.

.

.

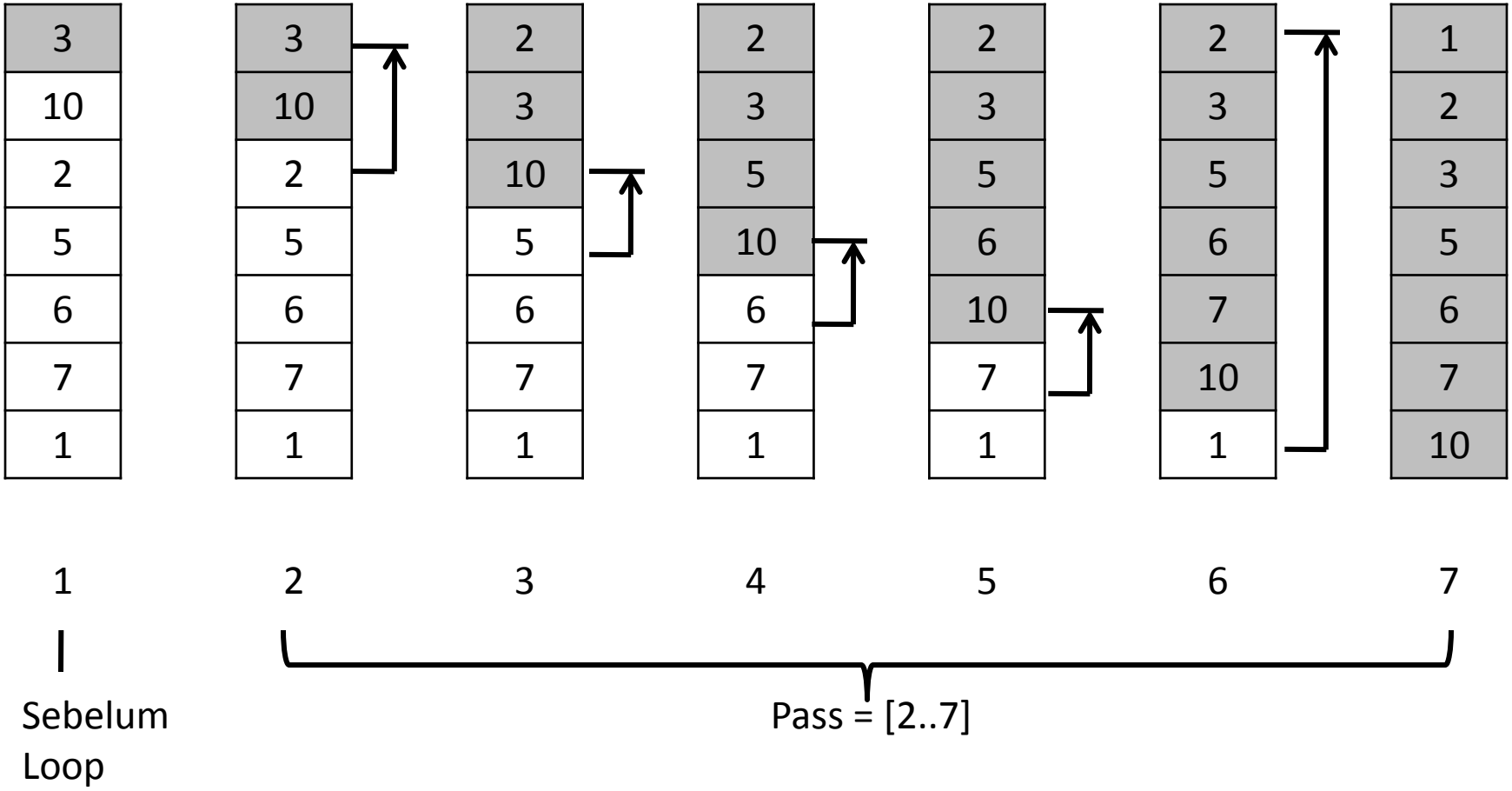
N-1 TabInt[N-1] harus dicarikan tempat yang tepat pada TabInt[1..N-1], yaitu i.
Sisipkan TabInt[N-1] pada TabInt[i]. TabInt [1..N-1] terurut membesar.

N TabInt [1..N] sudah terurut: TabInt[1] ≤ TabInt[2] ≤ TabInt[3] ... ≤ TabInt[N]

Insertion Sort: Proses

- Pada setiap Pass, tabel terdiri dari dua bagian: yang sudah terurut yaitu [1..Pass – 1] dan sisanya [Pass..Nmax] yang belum terurut.
- Ambil elemen TabInt[Pass], sisipkan ke dalam TabInt[1..Pass-1] dengan tetap menjaga keterurutan.
- Untuk menyisipkan TabInt[Pass] ke TabInt[i], harus terjadi "pergeseran" elemen tabel TabInt [i..Pass].
- Pergeseran ini dapat dilakukan sekaligus dengan pencarian i.
- Pencarian dapat dihentikan segera dengan memanfaatkan sifat keterurutan TabInt[1..Pass].
- Metoda pengurutan ini cukup efisien (≅ N) terutama untuk N yang "kecil".
- Terdapat 2 varian: tanpa sentinel dan dengan sentinel

Insertion Sort: Ilustrasi



Insertion Sort: Algoritma

```
Procedure InsertionSORT(Input/Output TabInt: TabelInteger, Input N : integer)

{ mengurut tabel integer [1..N] dengan insertion }

Kamus  :
   i: integer   {indeks untuk traversal tabel }
   Pass : integer     { tahapan pengurutan }
   Temp : integer

Algoritma :
   { TabInt1 adalah terurut}
   Pass traversal  [2..N]
      Temp ← TabIntPass    {Simpan  harga TabIntPass
                              spy tidak tertimpa krn pergeseran  }
      { Sisipkan elemen ke Pass dalam TabInt[1..Pass-1] sambil menggeser:}
      i ← Pass-1
      { Cari i, Temp ≤ TabInti and i > 1 }
      while (Temp < TabInti)  and (i > 1) do
         TabInti+1 ← TabInti            { Geser}
         i   ← i - 1             { Berikutnya }
      {Temp ≤ TabInti(tempat yg tepat)or i= 1 (sisipkan sbg elmt pertama)}
      depend on TabInt, I, Temp
         Temp ≤ TabInti:  TabInti+1 ←  Temp  {Menemukan tempat yg tepat}
         Temp < TabInti : TabInti+1 ← TabInti
                          TabInti ←  Temp {sbg elemen pertama }
      {TabInt[1..Pass] terurut membesar: TabInt≤ TabInt2≤ .. ≤ TabIntPass}
{Seluruh tabel terurut, karena Pass = N : TabInt1≤ TabInt2 ≤ TabInt3... ≤
TabIntN}
```

Insertion Sort: Algoritma dengan Sentinel

```
Procedure InsertionSORTWithSentinel(Input/Output TabInt: TabelInteger, Input N : integer)
{ mengurut tabel integer [1..N] dgn insertion sort, pencarian dengan sentinel}
Kamus :
   i : integer  {indeks untuk traversal tabel }
   Pass : integer    { tahapan pengurutan }
   Temp : integer  { Menyimpan harga Tab_Pass spy tidak tertimpa krn pergeseran }
Algoritma :
   { TabInt_1 adalah terurut}
   Pass traversal  [2..N]
      {Simpan dulu harga TabInt(Pass) spy tidak tertimpa krn pergeseran  }
      Temp ← TabInt_Pass
      TabInt_0 ← Temp
      { Sisipkan elemen ke Pass dalam TabInt[1..Pass-1] sambil menggeser }
      i ← Pass-1
      { Cari i, TabInt_I ≤ Temp and i > 1 }
      while (Temp < TabInt_i) do
         TabInt_i+1 ← TabInt_i          { Geser}
         i   ← i - 1            { Berikutnya }
      {Temp ≥ TabInt_i}
      TabInt_i+1 ← Temp               { Sisipkan }
      { TabInt[1..Pass] terurut:TabInt≤TabInt_2≤TabInt_3 ..... ≤TabInt_Pass }
{Seluruh tabel terurut, karena Pass = N : TabInt_1≤TabInt_2 ≤ TabInt_3 .... ≤ TabInt_N}
```

Selection Sort

- Idenya adalah menghasilkan nilai maksimum tabel (untuk efisiensi, hanya indeks dimana harga maksimum ditemukan yang dicatat), kemudian menukarnya dengan elemen terujung.
- Elemen terujung ini "diisolasi" dan tidak diikut sertakan pada proses berikutnya.
- Proses diulang untuk sisa tabel.
- Karena elemen terujung berharga maksimum adalah indeks pertama tabel, maka tabel terurut **mengecil:**

  TabInt[1] ≥ TabInt[2] ≥ TabInt[3] ... ≥ TabInt[N]

Selection Sort: Proses

Proses dilakukan sebanyak N tahapan (Pass) :

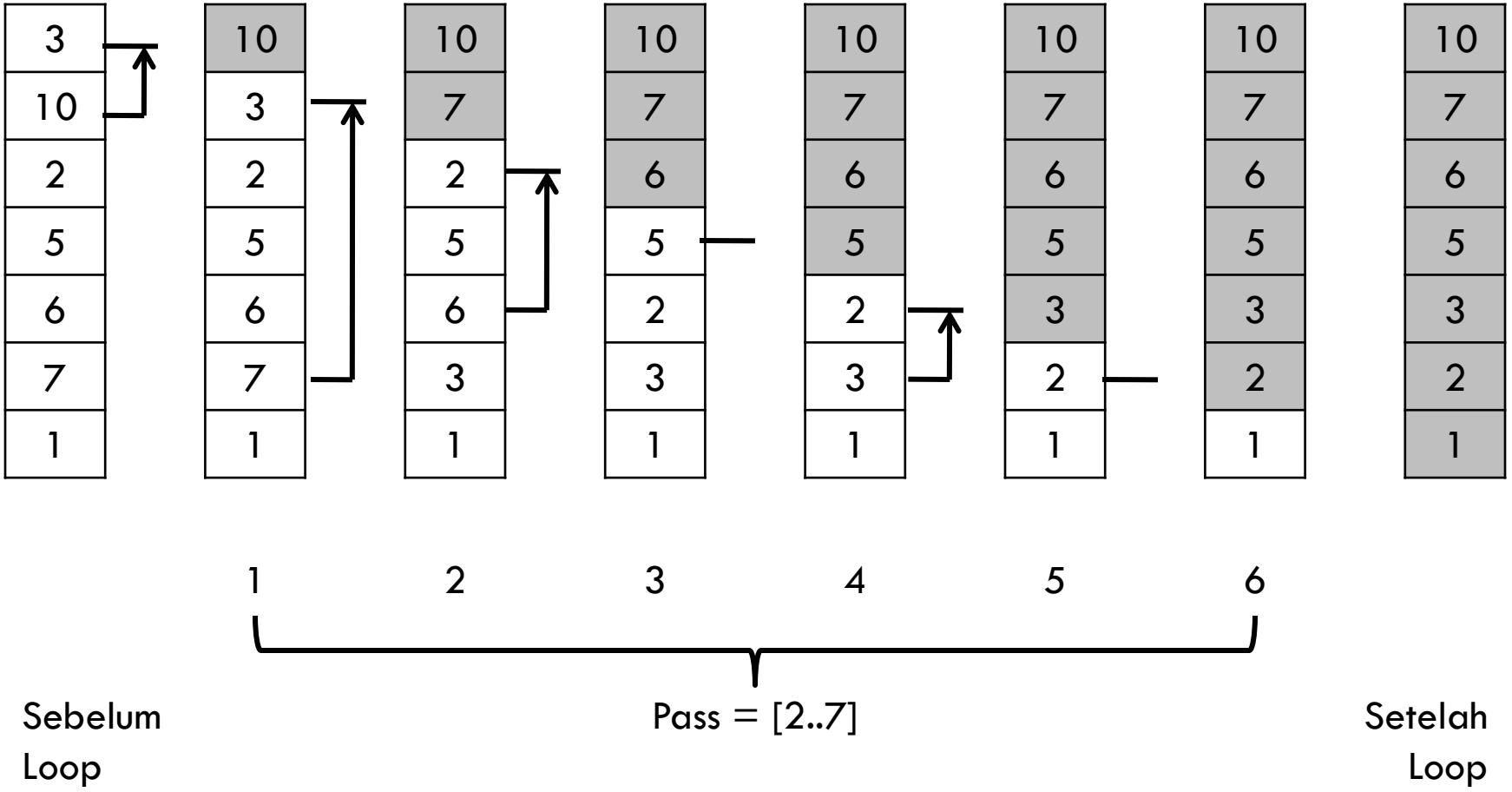
1. Tentukan IMAX [1..N], TabInt[IMAX] adalah maksimum dari TabInt[1..N]
   Tukar TabInt[IMAX] dengan TabInt[1]
2. Tentukan IMAX [2..N], TabInt[IMAX] adalah maksimum dari TabInt[2..N]
   Tukar TabInt[IMAX] dengan TabInt[2]
3. Tentukan IMAX [3..N], TabInt[IMAX] adalah maksimum dari TabInt[3..N]
   Tukar TabInt[IMAX] dengan TabInt[3]

.

.

N-1 Tentukan IMAX [N-1..N], TabInt[IMAX] adalah maksimum dari TabInt[N-1..N]
Tukar TabInt[IMAX] dengan TabInt[N-1]
TabInt [1..N] sudah terurut : TabInt[1] ≥ TabInt[2] ≥ TabInt[3] ... ≥ TabInt[N]

Selection Sort: Ilustrasi



Selection Sort: Algoritma

Program MAXSORT(Input/Output TabInt: TabelInteger, Input N : integer)
{ mengurut tabel integer [1..N] terurut mengecil dengan maksimum suksesif }

```
Kamus
    i : integer   {indeks untuk traversal tabel }
    Pass : integer     { tahapan pengurutan }
    Temp : integer  {memorisasi harga untuk penukaran }
    IMax : integer  {indeks, dimana TabInt [1..pass] berharga maksimum }
Algoritma
    Pass traversal  [1..N-1]
        { Tentukan Maximum [Pass..N] }
        Imax ←  pass
        i traversal [pass+1..N]
            if (TabInt_Imax < TabInt_i ) then
                Imax ←  i
        { TabInt_Imax  adalah maximum TabInt[pass..N]  }
        {Tukar TabInt_Imax  dengan TabInt_Pass  }
        Temp ← TabInt_Imax
        TabInt_Imax  ← TabInt_Pass
        TabInt_Pass ←  Temp
        { TabInt[1..Pass]  terurut: TabInt_1≥TabInt_2≥TabInt_3 .. ≥TabInt_Pass}
 {Seluruh tabel terurut, TabInt_1 ≥ TabInt_2 ≥ TabInt_3 ..... ≥ TabInt_N }
```

Referensi

- Liem, Inggriani. 2007. Diktat Kuliah Dasar Pemrograman (Bagian Pemrograman Prosedural). Bandung: Institut Teknologi Bandung