Struktur Pengurutan (Sorting) dalam bhs. Pemrograman Python

Pengurutan (*sorting*) adalah yang sangat penting saat memanipulasi data. Ketika kita mengurutkan data, data akan terlihat rapi dan mudah dibaca. Sehingga memudahkan juga dalam menganalisa.

Kenapa kita perlu algoritma untuk mengurutkan sesuatu? Saat pengurutan data dilakukan, program akan mengkalkulasi dan membandingkan setiap data agar dapat menemukan mana yang terbesar dan terkecil.

Oke, mungkin jika datanya cuman *"ratusan"* atau *"ribuan"* mungkin komputer kita masih bisa menanganinya. Tapi bagaimana jika datanya sampai jutaan atau miliaran seperti yang dilakukan Google?

Bisa kalian bayangkan kan? Maka dari itu pemilihan algoritma untuk pengurutan (*sorting*) juga mempengaruhi cepat atau lambatnya sistem dalam memanipulasi data.

Bubble Sort

Bubble Sort adalah algoritma sorting yang paling populer dan sederhana diantara algoritma lainnya. Proses pengurutan pada algoritma ini dengan membandingkan masing - masing elemen secara berpasangan lalu menukarnya dalam kondisi tertentu.

Proses ini akan terus diulang sampai elemen terakhir atau sampai tidak ada lagi elemen yang dapat ditukar. Inilah kenapa algoritma ini diberi nama "*Bubble*", dimana gelembung yang terbesar akan naik ke atas.

```
def bubble_sort(array):
    n = len(array) # jumlah list
    # perulangan pertama
    for i in range(n):
        # perulangan kedua
        for j in range(n - i - 1):
            # bandingkan masing" elemen
            if array[j] > array[j + 1]:
                # jika lebih besar, tukar.
                array[j], array[j + 1] = array[j + 1], array[j]
    return array
unordered = [5, 3, 4, 8, 1, 2, 9, 6]
print(bubble_sort(unordered))
```

Outputnya seperti ini :

```
[1, 2, 3, 4, 5, 6, 8, 9]
```

Apa yang terjadi pada kode diatas? Ini penjelasannya.

Alur Kode :

1. Kita hitung jumlah *list* menggunakan `len(array)` sebagai parameter perulangan.
2. Buat dua perulangan untuk membandingkan elemen pada *list*.
3. Bandingkan elemen pertama dengan elemen kedua menggunakan `if`.
4. Jika elemen pertama lebih besar daripada elemen kedua maka tukar posisinya.

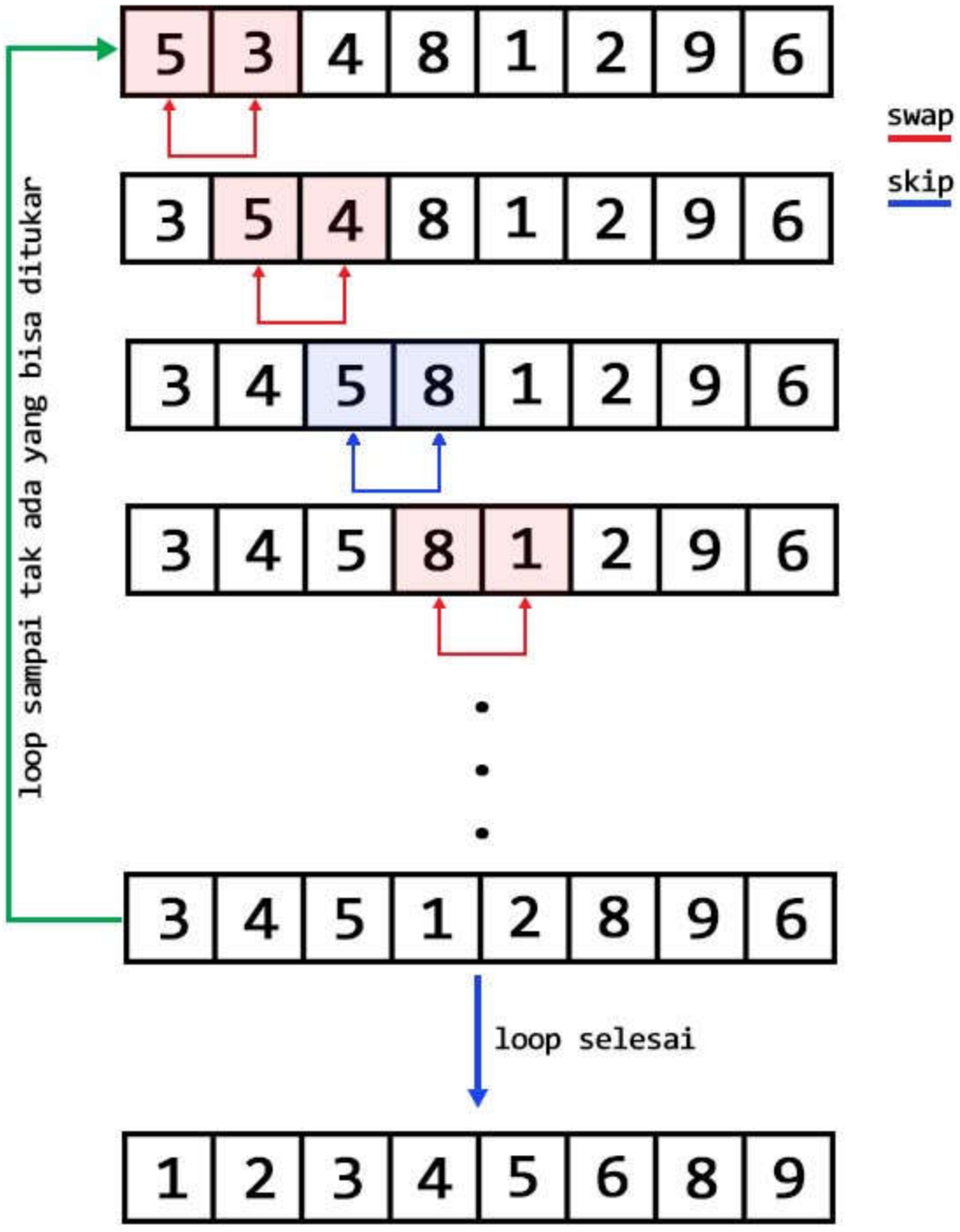
```
5. if array[j] > array[j + 1]:
6.     array[j], array[j + 1] = array[j + 1], array[j]
```

7. Jika elemen kedua lebih besar dari pada elemen pertama maka biarkan saja.
8. Langkah 3 - 5 diulang sampai elemen terakhir (atau perulangan selesai).

Kurang lebih gambarannya akan seperti ini :

@anbidev

Bubble Sort



>_ cd /AnbiDev

Lalu, untuk mengurutkan secara ***descending*** (dari terbesar ke terkecil) bagaimana? Mudah, tinggal kita ubah saja perbandingannya (`if`).

```
# descending
if array[j] < array[j + 1]:
    ...
[9, 8, 6, 5, 4, 3, 2, 1]
```

Dari yang tadinya elemen besar di tukar elemen kecil, sekarang elemen kecil ditukar elemen yang besar.

Selection Sort

Selection Sort adalah algoritma *sorting* yang mengurutkan data dengan cara mencari elemen paling kecil dari *list*, lalu menukar elemen tersebut ke urutan paling awal.

Dalam algoritma ini memiliki konsep yang sama dengan *bubble sort*, yaitu **membandingkan** dan **menukar**. Tetapi, dalam selection sort ia akan mencari `index` dengan elemen paling kecil, ketika sudah ketemu, elemen pada `index` itu akan ditukar dengan elemen pada `index` pertama.

Begitu seterusnya sampai perulangan selesai atau tidak ada lagi elemen yang bisa ditukar.

```
def selection_sort(arr):
    n = len(arr)
    # perulangan list elemen
    for i in range(n):
        # cari nilai elemen terendah
        # yang masih tersedia
        min_idx = i
        for j in range(i+1, n):
            if arr[min_idx] > arr[j]:
                min_idx = j

        # tukar dengan nilai elemen ke-i
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
listku = [64, 25, 12, 22, 11]
print(selection_sort(listku))
```

Lalu, outputnya seperti ini :

```
[11, 12, 22, 25, 64]
```

Apa yang terjadi pada kode diatas? Penjelasannya

Alur Kode :

1. Kita cari dulu jumlah elemen pada *list* dengan `len(arr)`.
2. Lalu lakukan perulangan pertama yang didalamnya terdapat kode untuk mencari nilai minimium dan menukar nilai.
3. Jika kalian perhatikan terdapat `min_idx` yang berperan untuk menampung index dengan nilai terendah.
4. Lalu pada perulangan kedua, setiap elemen akan terus dibandingkan menggunakan `if` untuk mendapatkan index dengan nilai terkecil.

```
5. for j in range(i+1, n):
6.     if arr[min_idx] > arr[j]:
7.         min_idx = j
```

8. Pada perulangan kedua, semua elemen setelah elemen ke-`i` atau `i+1`, saling dibandingkan untuk mencari nilai terkecil.
9. Setelah menemukan elemen dengan nilai terkecil, index tersebut ditukar dengan nilai ke-`i`.

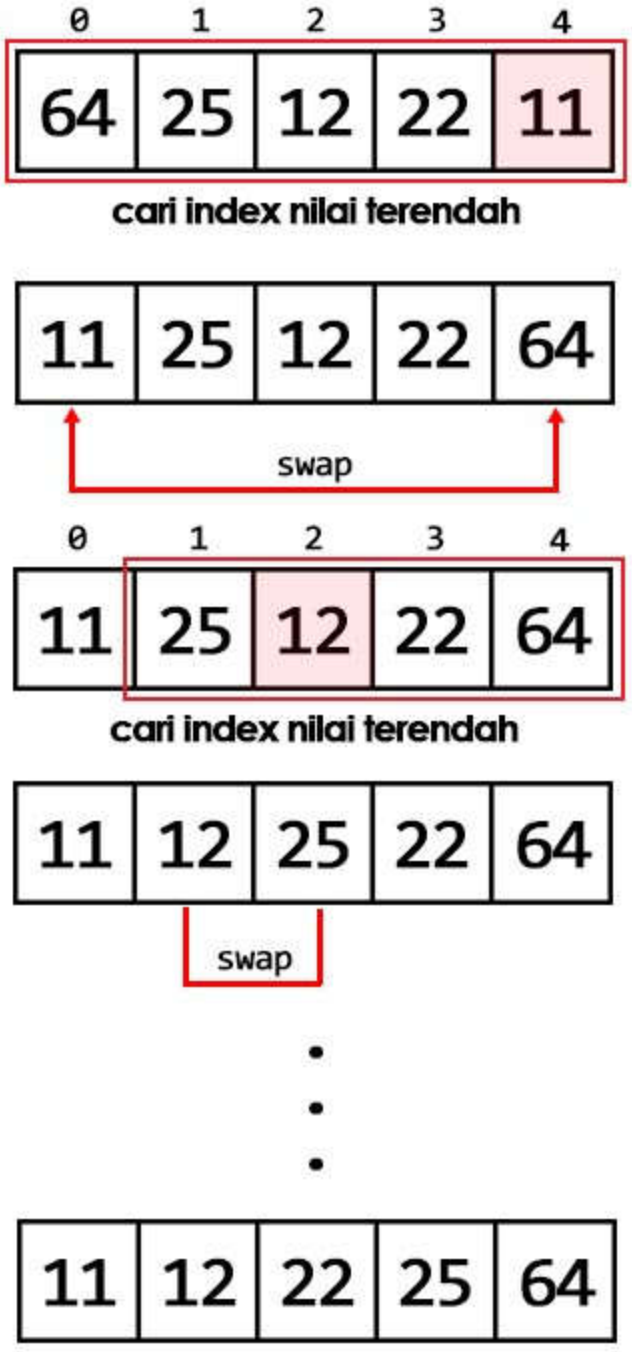
```
10.  arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

11. Hal tersebut diulang sampai perulangan pertama selesai (atau tidak elemen tuk diulang).

Kurang lebih gambarannya akan seperti ini :

@anbidev

Selection Sort



>_ cd /AnbiDev

Lalu bagaimana untuk ***descending*** order menggunakan *Selection Sort*? Sama seperti sebelumnya, kita tinggal ubah *pembandingnya* (`if`).

```
if arr[min_idx] < arr[j]:
    min_idx = j
[64, 25, 22, 12, 11]
```

Insertion Sort

Insertion Sort adalah algoritma yang melakukan pengurutan dengan membandingkan elemen satu dengan elemen lainnya dalam sebuah *list*. Elemen yang dibandingkan akan ditempatkan ke posisi yang sesuai (urut) pada *list*.

Analoginya seperti mengurutkan kumpulan kartu. Setiap kartu yang kalian ambil, kalian bandingkan terlebih dahulu ke kumpulan kartu yang sudah diurutkan. Dan ketika tahu urutan ke berapa, kalian selipkan kartu itu ke tumpukan kartu agar urut.

```
def insertion_sort(array):
    # perulangan pertama
    for i in range(1, len(array)):
        # ini elemen yang akan kita posisikan
        key_item = array[i]
        # kunci index posisi
        j = i - 1
        # lakukan perulangan kedua
        while j >= 0 and array[j] > key_item:
            # menggeser elemen yang lain
            array[j + 1] = array[j]
            j -= 1
        # memposisikan elemen
        array[j + 1] = key_item

    return array
unordered = [91, 21, 37, 77, 82]
print(insertion_sort(unordered))
```

Outputnya akan seperti ini :

```
[21, 37, 77, 82, 91]
```

Apa yang terjadi pada kode diatas? berikut penjelasannya.

Alur Kode

1. Kita mulai dari perulangan dari elemen kedua (`1`) sampai elemen terakhir. Kenapa tidak dari elemen pertama? karena elemen pertama adalah elemen yang dibandingkan oleh elemen yang lain.
2. Variabel `key_item`, adalah tempat untuk menampung elemen yang akan diposisikan.
3. Sedangkan variabel `j` dengan nilai `i - 1` adalah tujuan index yang nantinya elemen `key_item` akan ditempatkan.
4. Menggunakan perulangan `while` yang memiliki kondisi selama `j >= 0` dan elemen index `j` lebih besar dari `key_item`, Maka elemen yang lain akan digeser dan index `j` diperbarui.

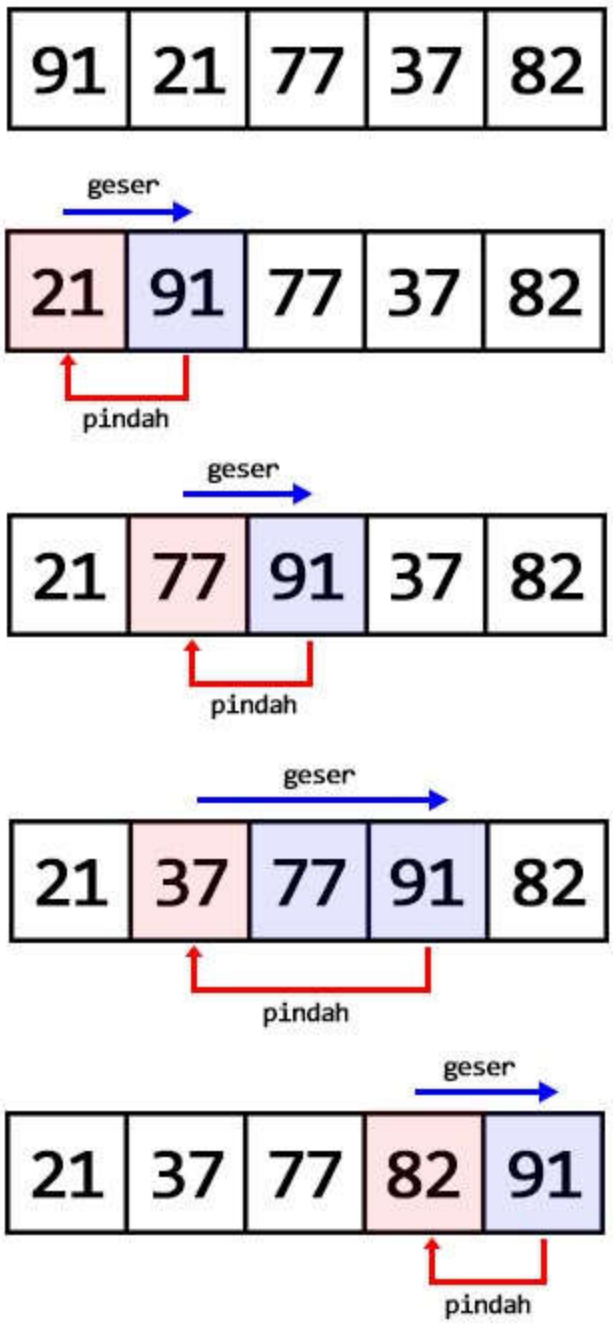
```
5. while j >= 0 and array[j] > key_item:
6.     array[j + 1] = array[j]
7.     j -= 1
```

8. Setelah tidak ada lagi elemen yang lebih besar dari `key_item`, maka perulangan `while` akan berhenti.
9. Lalu tempatkan elemen pada `key_item` ke index `j + 1`. Kenapa `j + 1`? karena sebelumnya kita memberikan nilai `i - 1` yang seharusnya tidak sesuai.

Kurang lebih gambarannya seperti ini.

@anbidev

Insertion Sort



>_ cd /AnbiDev

Insertion Sort

Lalu untuk ***descending*** order menggunakan *Insertion Sort*, kita tinggal ubah *pembanding* pada saat `while`.

```
while j >= 0 and array[j] < key_item:
    array[j + 1] = array[j]
    j -= 1
```

Outputnya akan seperti ini :

```
[91, 82, 77, 37, 21]
```

. . .

REFERENSI :

https://www.anbidev.com/python-sorting/