Penjelasan Algoritma Pengurutan (Sorting)

Alamanda Ardana - PPL
PeTIK Jombang 2022/2023 M

DAFTAR ISI

(PEMBAHASAN).....(HALAMAN)

DAFTAR ISI 2
BAGIAN 1 : BUBBLE SORT 3
BAGIAN 2 : SELECTION SORT 7
BAGIAN 3 : INSERTION SORT 11
BAGIAN 4 : QUICK SORT 15
BAGIAN 5 : MERGE SORT 17
REFERENSI 19

BAGIAN 1
BUBBLE SORT

Bubble Sort adalah algoritma *sorting* yang paling populer dan sederhana diantara algoritma lainnya. Proses pengurutan pada algoritma ini adalah 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]
```

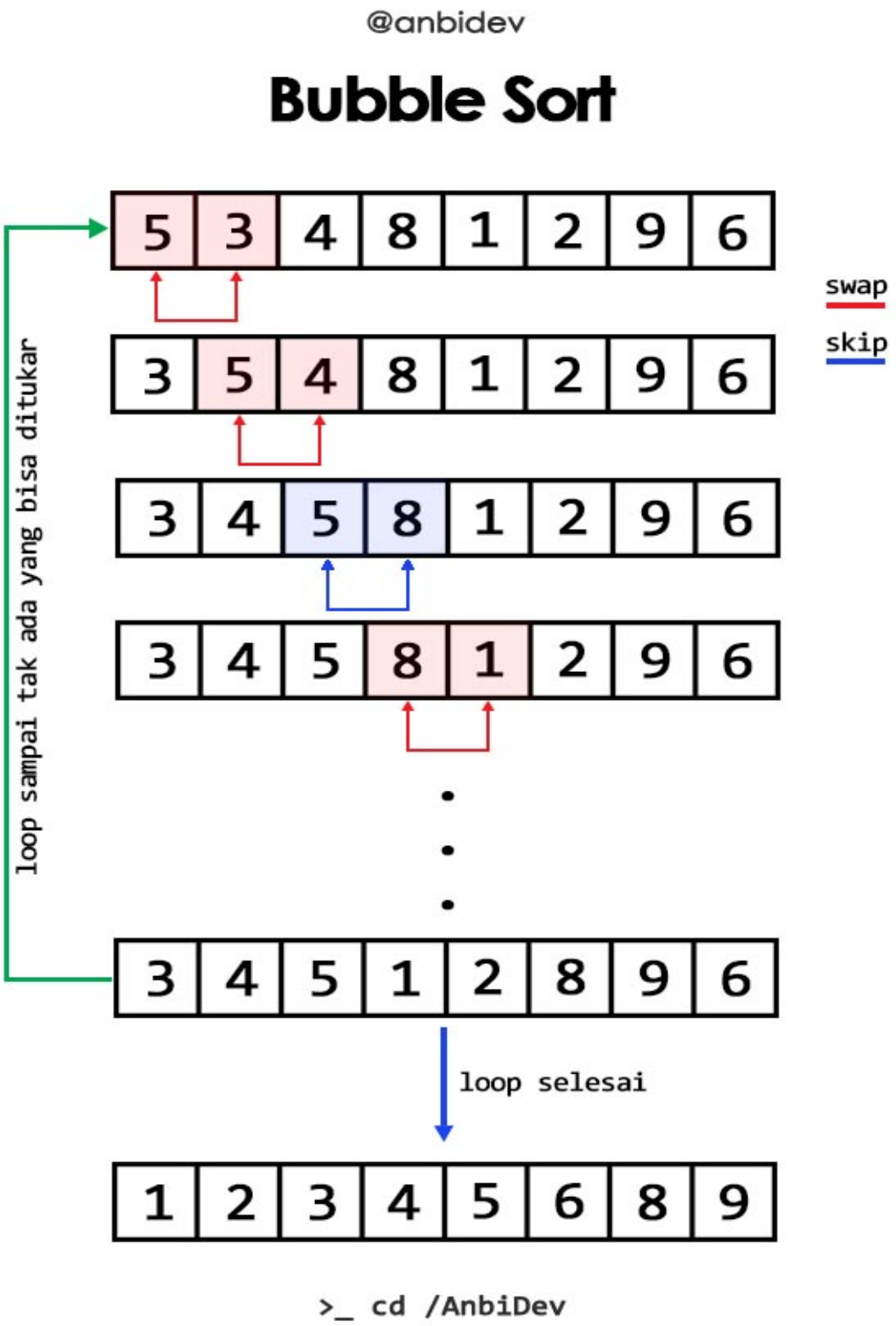
Alur Kode Bubble Sort (yang tadi) :

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.

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

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

Kurang lebih gambaran dari kode Bubble Sort tadi adalah sebagai berikut.



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]
```

BAGIAN 2
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]
```

Alur Kode Selection Sort (yang tadi) :

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

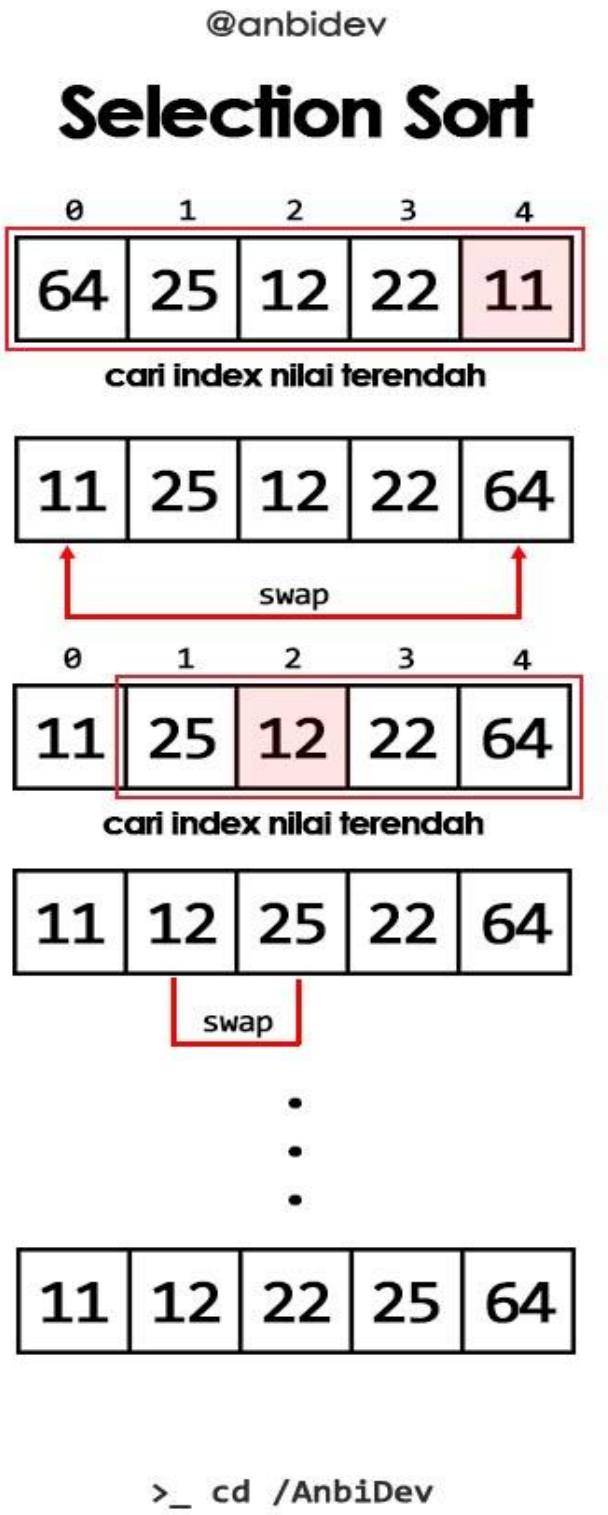
```
for j in range(i+1, n):
        if arr[min_idx] > arr[j]:
                min_idx = j
```

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

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

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

Kurang lebih gambaran dari kode Selection Sort tadi adalah sebagai berikut.



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]
```

BAGIAN 3
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 kita ambil, kita bandingkan terlebih dahulu ke kumpulan kartu yang sudah diurutkan. Dan ketika tahu urutan ke berapa, kita 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]
```

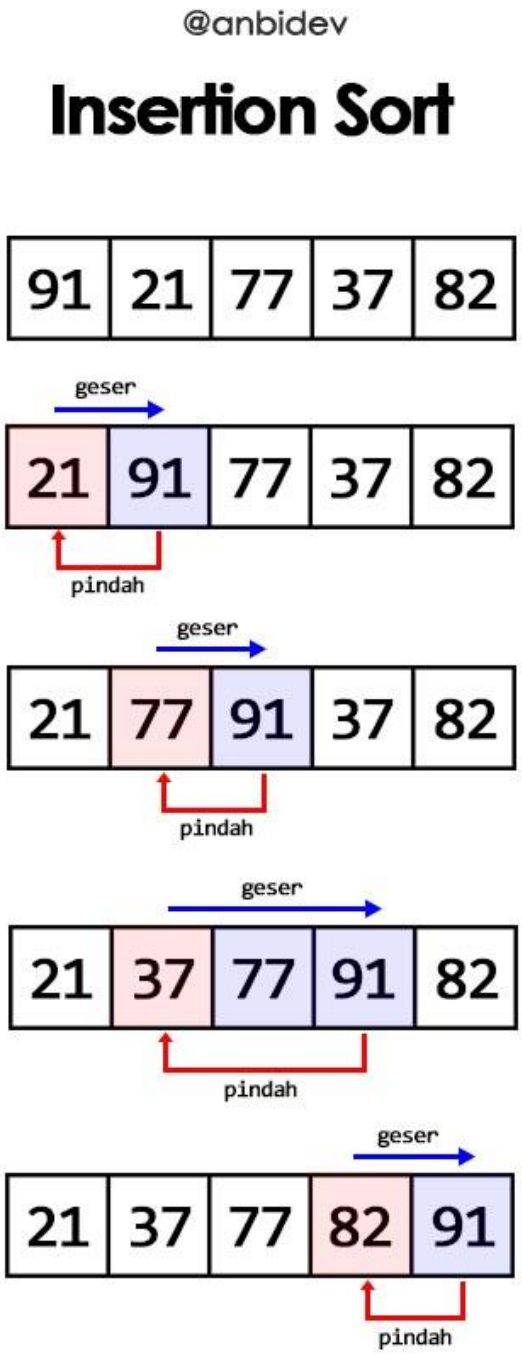
Alur Kode Insertion Sort (yang tadi) :

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, sehingga elemen yang lain akan digeser dan index j diperbarui.

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

5. Setelah tidak ada lagi elemen yang lebih besar dari key_item, maka perulangan while akan berhenti.
6. 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 gambaran dari kode Insertion Sort tadi adalah sebagai berikut.



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]
```

BAGIAN 4
QUICK SORT

Cara kerja algoritma Quick Sort berprinsip pada pendekatan **divide and conquer,** yakni memilih satu elemen sebagai elemen pivot dan mempartisi array sehingga sisi kiri pada pivot mempunyai semua elemen dengan nilai yang lebih kecil dibandingkan dengan elemen pivot, dan sisi kanan mempunyai semua elemen dengan nilai yang lebih besar daripada nilai elemen pivot.

Analogi algoritma Quick Sort :

1. Mempunyai data A yang memiliki N elemen. Pilih sembarang elemen dari data tersebut, dan biasanya elemen pertama dianggap sebagai elemen x.
2. Kemudian, semua elemen tersebut disusun dengan menempatkan x pada posisi j sedemikian rupa sehingga elemen kesatu sampai pada j-1 memiliki nilai yang lebih besar dari x.
3. Begitu seterusnya setiap sub data

Contoh Quick Sort :

Code

```
def qs(list,awal,akhir):
    if awal < akhir:
        pindex = partisi(list,awal,akhir)
        qs(list,awal,pindex-1)
        qs(list,pindex+1,akhir)

def partisi(list,awal,akhir):
    tengah = int(akhir/2)
    pivot = list[tengah]
    pindex = awal
    for i in range(awal,tengah):
        if list[i]>=pivot:
            list[i],list[pindex]=list[pindex],list[i]
            pindex = pindex + 1
    list[pindex],list[tengah]=list[tengah],list[pindex]
    print(list)
    return pindex

list = [67,91,87,33,49,10,16,43,65,3]
print('Data yang akan di sort :', list)
print('Quick Sort :')
qs(list,0,len(list)-1)
```

Output dari penerapan quick sort di atas seperti pada gambar di bawah ini :

```
C:\sort\py>python qs.py
Data yang akan di sort : [67, 91, 87, 33, 49, 10, 16, 43, 65, 3]
Quick Sort :
[67, 91, 87, 49, 33, 10, 16, 43, 65, 3]
[91, 67, 87, 49, 33, 10, 16, 43, 65, 3]
[91, 67, 87, 49, 33, 10, 16, 43, 65, 3]
[91, 67, 87, 49, 33, 10, 16, 43, 65, 3]
[91, 67, 87, 49, 10, 33, 16, 43, 65, 3]
[91, 67, 87, 49, 16, 33, 10, 43, 65, 3]
[91, 67, 87, 49, 43, 33, 10, 16, 65, 3]
[91, 67, 87, 49, 65, 33, 10, 16, 43, 3]
```

Output Quick Sort

BAGIAN 5
MERGE SORT

Algoritma Merge Sort merupakan salah satu pengurutan dengan metode memecah data, kemudian mengolah untuk diselesaikan pada setiap bagian dan menggabungkan kembali sehingga data tersebut berhasil tersusun. Merge Sort dalam menyelesaikan pengurutan membutuhkan fungsi rekursif.

Analogi algoritma Merge Sort :

1. Data dipecah menjadi dua kelompok, dimana kelompok pertama adalah setengah apabila data genap atau setengah kurang satu apabila data ganjil dari seluruh data.
2. Kemudian dilakukan pemecahan kembali pada masing-masing kelompok hingga hanya terdapat satu data pada satu kelompok.
3. Gabungkan data kembali dengan membandingkannya pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah ditambah satu. Jika iya, maka data ke tengah ditambah satu dipindah menjadi data pertama.
4. Kemudian data pertama tadi hingga data ke tengah dipindah menjadi data kedua sampai data ke tengah ditambah satu.
5. Begitu seterusnya sehingga membentuk sebuah data yang tersusun dalam satu kelompok yang utuh.

Contoh Merge Sort :

Code

```
def ms(list):
    print('Memecah List', list)
    n = len(list)
    if n < 2:
        return list
    else:
        mid=n//2
        left=list[:mid]
        right=list[mid:]

        ms(left)
        ms(right)
        i=0
        j=0
        k=0
        while i < len (left) and j < len (right):
            if left[i]>right[j]:
                list[k]=left[i]
                i=i+1
            else:
                list[k]=right[j]
                j=j+1
            k=k+1
        while i < len (left):
            list[k]=left[i]
            i=i+1
            k=k+1
        while j < len (right):
            list[k]=right[j]
            j=j+1
            k=k+1
    print('Menggabungkan List', list)

list = [2,5,60,43,27,10,89,17]
ms(list)
```

Output dari penerapan merge sort di atas seperti pada gambar di bawah ini :

```
C:\sort\py>python ms.py
Memecah List [2, 5, 60, 43, 27, 10, 89, 17]
Memecah List [2, 5, 60, 43]
Memecah List [2, 5]
Memecah List [2]
Memecah List [5]
Menggabungkan List [5, 2]
Memecah List [60, 43]
Memecah List [60]
Memecah List [43]
Menggabungkan List [60, 43]
Menggabungkan List [60, 43, 5, 2]
Memecah List [27, 10, 89, 17]
Memecah List [27, 10]
Memecah List [27]
Memecah List [10]
Menggabungkan List [27, 10]
Memecah List [89, 17]
Memecah List [89]
Memecah List [17]
Menggabungkan List [89, 17]
Menggabungkan List [89, 27, 17, 10]
Menggabungkan List [89, 60, 43, 27, 17, 10, 5, 2]
```

Output Merge Sort

REFERENSI

- https://www.anbidev.com/python-sorting/
- https://blogbugabagi.blogspot.com/2019/12/algoritma-sorting-qoutput-quicj-sortt.html