Modul 5

Pengurutan

Pengurutan (*sorting*) adalah proses menyusun atau menata koleksi item-item sedemikian rupa sehingga setiap item dengan item yang sesudahnya memenuhi persyaratan hubungan tertentu. Item-item ini bisa berupa nilai-nilai yang sederhana seperti bilangan bulat dan bilangan real, atau bisa juga berupa sesuatu yang lebih kompleks seperti data rekam mahasiswa, atau entri kamus. Yang manapun saja, pengurutan item-item itu didasarkan pada nilai ***kunci pengurutan*** (*sort key*). Kunci pengurutan ini bisa berupa nilai dirinya sendiri untuk tipe data primitif (int, float, str) atau bisa juga berupa komponen tertentu atau kombinasi komponen tertentu untuk pengurutan yang lebih kompleks.

Problem dasar pengurutan dapat diilustrasikan sebagai berikut. Diberikan sebuah array:

```
[10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
```

buatlah suatu fungsi untuk mengurutkan elemen-elemen di array itu dari yang paling kecil ke yang paling besar, menjadi:

```
[2, 4, 5, 10, 13, 18, 23, 29, 31, 51, 64]
```

Pengurutan merupakan salah satu algoritma yang paling sering dijalankan dan sampai sekarang masih merupakan bidang ilmu yang aktif diselidiki. Gambar 5.1 mendaftari berbagai macam algoritma pengurutan yang artikelnya terdapat di Wikipedia Bahasa Inggris.

Di modul ini kita akan mempelajari beberapa algoritma pengurutan sederhana: *bubble sort*, *selection sort*, dan *insertion sort*. Insya Allah di modul selanjutnya kita akan membahas sejumlah algoritma pengurutan yang lebih kompleks (yang juga lebih cepat).

Namun sebelum membahas algoritma pengurutan itu semua, kita tulis dulu sebuah *routine* yang akan sering dipakai, yakni fungsi tukar posisi, atau *swap*, untuk menukar posisi dua elemen di suatu list.

routine swap untuk menukar A[p] dan A[q]

```
1 def swap(A,p,q):
2     tmp  = A[p]
3     A[p] = A[q]
4     A[q] = tmp
```

Template:Sorting

From Wikipedia, the free encyclopedia

Sorting algorithms	
Theory	Computational complexity theory · Big O notation · Total order · Lists · Inplacement · Stability · Comparison sort · Adaptive sort · Sorting network · Integer sorting
Exchange sorts	Bubble sort · Cocktail sort · Odd–even sort · Comb sort · Gnome sort · Quicksort · Stooge sort · Bogosort
Selection sorts	Selection sort · Heapsort · Smoothsort · Cartesian tree sort · Tournament sort · Cycle sort
Insertion sorts	Insertion sort · Shellsort · Splaysort · Tree sort · Library sort · Patience sorting
Merge sorts	Merge sort · Cascade merge sort · Oscillating merge sort · Polyphase merge sort · Strand sort
Distribution sorts	American flag sort · Bead sort · Bucket sort · Burstsort · Counting sort · Pigeonhole sort · Proxmap sort · Radix sort · Flashsort
Concurrent sorts	Bitonic sorter · Batcher odd–even mergesort · Pairwise sorting network
Hybrid sorts	Block sort · Timsort · Introsort · Spreadsort · UnShuffle sort · JSort
Other	Topological sorting · Pancake sorting · Spaghetti sort

Gambar 5.1: Sejumlah algoritma pengurutan yang artikelnya terdapat di Wikipedia bahasa inggris. Kunjungi `en.wikipedia.org/wiki/Sorting_algorithm` untuk memulainya. Di modul ini kita akan mulai dengan tiga algoritma: *bubble sort*, *selection sort*, dan *insertion sort*.

Perhatikan bahwa tidak ada kata `return` di sana. (Mengapa?) Mari kita uji routine di atas dengan program kecil seperti berikut.

```
>>> K = [50, 20, 70, 10]
>>> swap(K, 1, 3)
>>> K
[50, 10, 70, 20]
```

Berhasil! Elemen dengan index 1 dan 3 telah bertukar posisi. Kita juga akan memerlukan *routine* untuk mencari index yang nilainya terkecil, saat diberikan jangkauan indexnya[1].

routine untuk mencari index dari elemen yang terkecil

```
1 def cariPosisiYangTerkecil(A, dariSini, sampaiSini):
2     posisiYangTerkecil = dariSini           #-> anggap ini yang terkecil
3     for i in range(dariSini+1, sampaiSini): #-> cari di sisa list
4         if A[i] < A[posisiYangTerkecil]:    #->  kalau menemukan yang lebih kecil,
5             posisiYangTerkecil = i          #->  anggapan dirubah
6     return posisiYangTerkecil
```

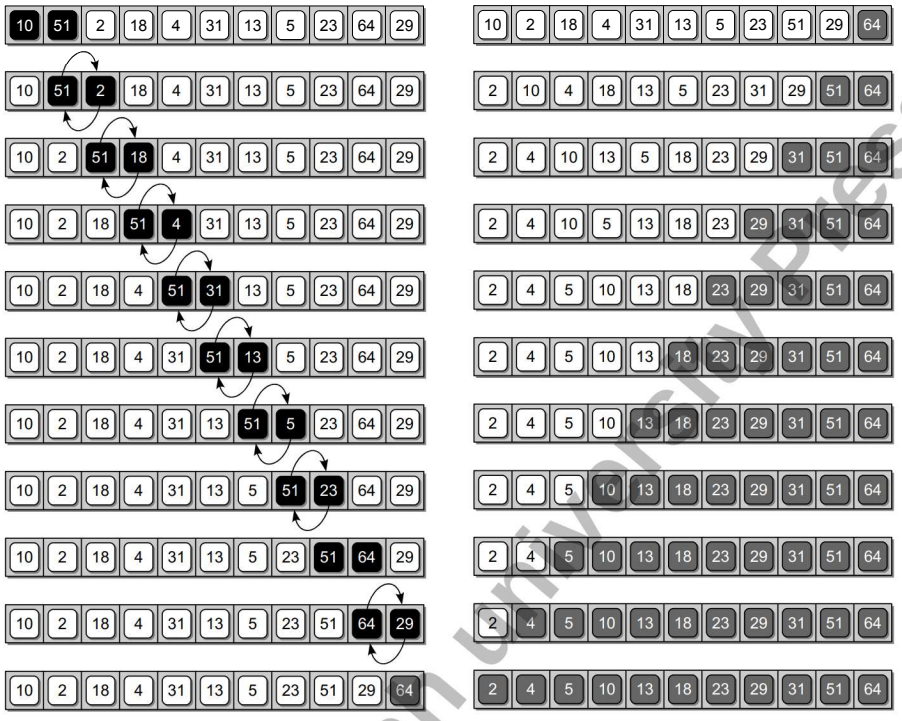
Untuk mengujinya, kita ketik yang berikut di Python Shell

```
>>> A = [18, 13, 44, 25, 66, 107, 78, 89]
>>> j = cariPosisiYangTerkecil(A, 2, len(A))
>>> j
3
```

Pada kode di atas, kita mencari index yang nilainya terkecil dalam jangkauan dari 2 sampai akhir list; hasilnya adalah index 3 (dengan nilai A[3]=25). Kalau jangkauannya dari awal sampai akhir list, hasilnya adalah index 1 (dengan nilai A[1]=13). Sekarang kita siap dengan semua algoritma pengurutan[2].

[1] Lihat juga algoritma pencarian nilai terkecil di halaman 42. Di sana kita mencari *nilai* yang terkecil. Di sini kita mencari *index* yang nilainya terkecil.

[2] Di sini kita tidak akan menggunakan fungsi dan metode bawaan Python seperti `A.sort()` atau `min(A)`. Juga, di sini kita akan tetap memakai array yang sama antara array asal dengan hasilnya. Maksudnya, array asal itulah satu-satunya array yang diproses. *Kita tidak membuat array baru.* Dalam beberapa aplikasi hal ini benar-benar



(a) (b)

Gambar 5.2: Contoh penerapan ***bubble sort***. (a) ***putaran-sisi-dalam pertama*** yang lengkap, menempatkan angka 64 pada posisi yang tepat. Kotak-kotak hitam mewakili nilai yang diperbandingkan. Anak panah menunjukkan pertukaran posisi. (b) Hasil penerapan bubble sort pada runtut contoh. Setiap satu baris pada gambar (b) menunjukkan satu putaran-sisi-dalam.

5.1 Bubble Sort

Pengurutan dengan bubble sort merupakan salah satu algoritma pengurutan yang paling sederhana. Algoritmanya bisa diterangkan seperti berikut.

1. Diberikan sebuah list yang kita diminta mengurutkan dari kecil ke besar:

   ```
   L = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
   ```

   - Bandingkan nilai elemen pertama dengan elemen kedua. Kalau elemen pertama lebih besar dari elemen kedua, tukar posisinya.
   - Sesudah itu bandingkan elemen kedua dengan elemen ketiga. Kalau elemen kedua lebih besar dari elemen ketiga, tukar posisinya.
   - Lakukan hal seperti di atas sampai elemen terakhir. Perulangan ini kita namakan

diperlukan.

putaran-dalam. Perhatikan bahwa dengan cara ini elemen yang paling besar akan 'menggelembung[3] ke kanan' dan akhirnya berakhir di paling kanan.

2. Ulangi lagi hal di atas dari awal sampai elemen yang terakhir kurang satu. (Karena elemen terbesarnya sudah berada di elemen terakhir.) Pada putaran ketiga: ulangi lagi hal di atas dari awal sampai elemen yang terakhir kurang dua. Demikian seterusnya.

Gambar 5.2(a) menunjukkan putaran-dalam pertama yang lengkap sebuah bubble sort. Sedangkan Gambar 5.2(b) menunjukkan hasil masing-masing putaran dalam dan akhirnya hasil akhir penerapan bubble sort pada runtut contoh. Berikut ini adalah program Python-nya.

```
1 def bubbleSort(A):
2     n = len(A)
3     for i in range(n-1):       #-> Lakukan operasi gelembung sebanyak n-1
4         for j in range(n-i-1): #->   Dorong elemen terbesar ke ujung kanan
5             if A[j] > A[j+1]:  #->     Jika di kiri lebih besar dari di kanannya,
6                 swap(A,j,j+1)  #->     tukar posisi elemen ke j dengan ke j+1
```

Pertanyaan: dengan elemen sebanyak n, berapa banyakkah operasi pembandingan dan pertukaran yang dilakukan oleh algoritma bubble sort ini? Selidiki nilainya untuk *worst-case*, *average-case*, dan *best-case scenario*[4].

Algoritma bubble sort ini sangat lambat dan merupakan contoh suatu *bad algorithm*. Bahkan ada beberapa pakar[5] yang menyarankan bahwa algoritma bubble sort ini *tidak usah diajarkan* di kuliah.

5.2 Selection Sort

Metode pengurutan lain yang cukup sederhana adalah metode *selection sort*. Berikut ini adalah penjelasannya

1. Mulai dari $i = 0$.
2. Pegang elemen ke i. Dari elemen i itu hingga terakhir, cari yang terkecil.
3. Kalau elemen ke i itu ternyata yang terkecil, berarti dia pada posisi yang tepat dan lanjutkan ke langkah berikutnya.

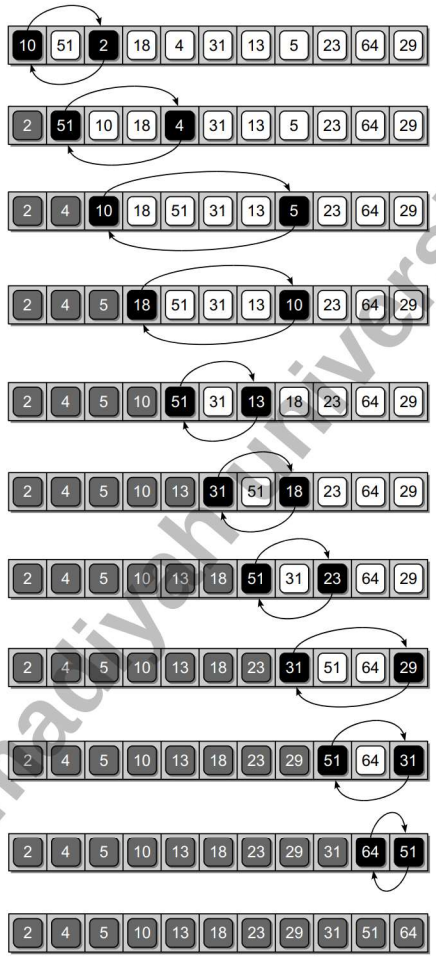
   Tapi kalau elemen terkecil itu bukan elemen ke i (berarti ada di sisi yang lebih kanan), tukar posisi elemen ke i dengan elemen terkecil itu tadi.
4. Lanjutkan untuk $i + 1$ dan ulangi proses 2 dan 3. Demikian seterusnya sampai elemen terakhir.

Gambar 5.3 menunjukkan hasil penerapan selection sort ini pada array contoh. Perhatikan bahwa algoritma ini lebih cepat daripada algoritma bubble sort yang kita bahas sebelumnya.

[3] Bahasa Inggris: *bubble* artinya gelembung (seperti gelembung sabun)

[4] *worst-case*: kejadian terburuk, misalnya array itu awalnya dalam keadaan urut terbalik; *average-case*: kejadian 'yang biasanya'; *best-case*: kejadian terbaik, misalnya array itu awalnya dalam keadaan sudah urut.

[5] misal Prof. Owen Astrachan dari *Duke University*



Gambar 5.3: Hasil penerapan algoritma pengurutan ***selection sort*** pada array contoh. Kotak-kotak berwarna abu-abu menunjukkan nilai-nilai yang sudah diurutkan. Kotak-kotak berwarna hitam menunjukkan nilai-nilai yang ditukar posisinya pada tiap iterasi di algoritma ini.

Kode berikut adalah salah satu implementasi selection sort dengan bahasa Python.

```
1 def selectionSort(A):
2     n = len(A)
3     for i in range(n - 1):
4         indexKecil = cariPosisiYangTerkecil(A, i, n)
5         if indexKecil != i :
6             swap(A, i, indexKecil)
```

Pada implementasi lain algoritma ini, pemeriksaan di baris 5 ditiadakan. Jadi setelah ditemukan posisi yang nilainya terkecil, langsung ditukar tanpa diperiksa terlebih dahulu. Cobalah untuk menyelidiki mana yang lebih cepat di antara dua implementasi ini.

Pertanyaan: dengan elemen sebanyak n, berapa banyakkah operasi pembandingan dan pertukaran yang dilakukan oleh algoritma selection sort ini? Selidiki nilainya untuk *worst-case*, *average-case*, dan *best-case scenario*

5.3 Insertion Sort

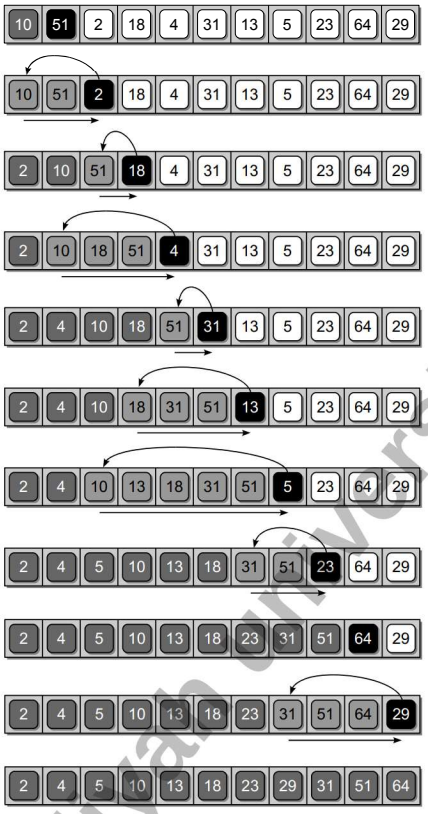
Algoritma pengurutan sederhana lain adalah algoritm *insertion sort*[6]. Algoritma ini mempunyai ide dasar seperti berikut (lihat pula Gambar 5.4):

1. Mulai dari elemen pertama. Anggap ini sebagai elemen dengan nilai terkecil.
2. Sekarang lihat elemen berikutnya (di sebelah kanannya). Tempatkan/sisipkan elemen ini ke kiri sesuai dengan nilainya.
3. Ulangi kembali proses nomer dua di atas untuk elemen di sebelah kanannya lagi. Demikian seterusnya sampai elemen terakhir diposisikan ke tempat yang tepat.

Sebagai contoh, perhatikan Gambar 5.4 di halaman 53.

- Anggap angka 10 sebagai nilai terkecil. Lalu, lihat baris pertama, angka 51 coba diposisikan ke tempat yang tepat. Tidak ada pergeseran karena 51 lebih besar dari 10.
- Baris kedua. Angka 2 coba diposisikan ke tempat yang tepat. Dia harus menggeser 10 dan 51 ke kanan agar angka 2 ini menempati posisi yang tepat.
- Baris ketiga. Angka 18 datang. Dia harus disisipkan ke tempat yang tepat. Untuk itu dia harus menggeser 51 ke kanan.
- Baris keempat. Angka 4 datang. Dia harus menggeser 10, 18, 51 ke kanan sampai dia menempati posisi yang tepat.
- ... demikian seterusnya.

[6] *Insertion* dapat diartikan *penyisipan*.



Gambar 5.4: Hasil penerapan algoritma pengurutan ***insertion sort*** pada array contoh. Kotak-kotak berwarna abu-abu menunjukkan nilai-nilai yang sudah diurutkan. Kotak-kotak berwarna hitam menunjukkan nilai berikutnya yang akan diposisikan. Kotak-kotak abu-abu yang lebih terang dengan tulisan hitam adalah nilai yang sudah urut yang harus digeser ke kanan untuk memberi tempat pada nilai yang sedang disisipkan.

Program di bawah ini adalah salah satu bentuk implementasi insertion sort.

```
def insertionSort(A):
    n = len(A)
    for i in range(1, n):
        nilai = A[i]
        pos = i
        while pos > 0 and nilai < A[pos - 1]: # -> Cari posisi yang tepat
            A[pos] = A[pos - 1]               #    dan geser ke kanan terus
            pos = pos - 1                     #    nilai-nilai yang lebih besar
        A[pos] = nilai # -> Pada posisi ini tempatkan nilai elemen ke i.
```

5.4 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal di bawah, kerjakan dulu latihan-latihan di atas.

1. Buatlah suatu program untuk mengurutkan array mahasiswa berdasarkan NIM, yang elemennya terbuat dari class `MhsTIF`, yang telah kamu buat sebelumnya.

2. Misal terdapat dua buah array *yang sudah urut* A dan B. Buatlah suatu program untuk menggabungkan, secara efisien, kedua array itu menjadi suatu array C yang urut.

3. Kamu mungkin sudah menduga, *bubble sort* lebih lambat dari *selection sort* dan juga *insertion sort*. Tapi manakah yang lebih cepat antara *selection sort* dan *insertion sort*?[7] Untuk memulai menyelidikinya, kamu bisa membandingkan waktu yang diperlukan untuk mengurutkan sebuah array yang besar, misal sepanjang 6000 (enam ribu) elemen.

```
from time import time as detak
from random import shuffle as kocok
k = range(1,6001)
kocok(k)
u_bub = k[:] ## \\
u_sel = k[:] ##   -- Jangan lupa simbol [:]-nya!.
u_ins = k[:] ## //

aw=detak();bubbleSort(u_bub);ak=detak();print('bubble: %g detik' %(ak-aw) );
aw=detak();selectionSort(u_sel);ak=detak();print('selection: %g detik' %(ak-aw) );
aw=detak();insertionSort(u_ins);ak=detak();print('insertion: %g detik' % (ak-aw) );
```

   Bandingkan hasil percobaan kamu dengan hasil teman-temanmu. Jika waktu untuk pengurutan dirasa terlalu cepat, kamu bisa memperbesar ukuran array itu.

   Perlu diingat bahwa hasil yang kamu dapat belum dapat menyimpulkan apa-apa. Selain strategi implementasinya, banyak kondisi yang harus diperhatikan dalam penerapan suatu algoritma. Untuk keadaan tertentu, dipakai selection sort. Untuk keadaan tertentu yang lain, dipakai insertion sort. Untuk keadaan lain lagi, dipakai algoritma yang berbeda[8].

Ketiga algoritma pengurutan yang dibahas di modul ini termasuk algoritma awal yang relatif sederhana. Para ahli telah menyelidiki dan membuat berbagai algoritma pengurutan yang lebih cepat, yang insya Allah beberapa di antaranya akan kita bahas di modul berikutnya.

[7] Setidaknya untuk versi implementasi di modul ini. Versi yang lebih optimal bisa jadi membuahkan hasil yang berbeda.

[8] Python sendiri, ketika kita memanggil `A.sort()`, memakai algoritma yang dinamai *Timsort*, yang merupakan gabungan beberapa algoritma. Lihat `http://en.wikipedia.org/wiki/Timsort` . Cobalah membandingkan perintah pengurutan di Python (misal: `k.sort()`) dengan algortima yang telah kamu buat di atas.