5.6. Sortasi

Sortasi adalah proses penempatan elemen dari koleksi dalam beberapa jenis pesanan. Misalnya, daftar kata dapat diurutkan berdasarkan abjad atau panjangnya. Daftar kota dapat diurutkan berdasarkan populasi, berdasarkan wilayah, atau dengan kode pos. Kami telah melihat sejumlah algoritma yang dapat mengambil manfaat dari memiliki daftar yang diurutkan (ingat contoh anagram terakhir dan pencarian biner).

Ada banyak, banyak algoritma penyortiran yang telah dikembangkan dan dianalisis. Ini menunjukkan bahwa menyortir adalah bidang studi penting dalam ilmu komputer. Menyortir sejumlah besar item dapat mengambil banyak sumber daya komputasi. Seperti halnya pencarian, efisiensi algoritme pengurutan terkait dengan jumlah item yang sedang diproses. Untuk koleksi kecil, metode pemilahan yang kompleks mungkin lebih merepotkan daripada nilainya. Overhead mungkin terlalu tinggi. Di sisi lain, untuk koleksi yang lebih besar, kami ingin memanfaatkan sebanyak mungkin peningkatan. Pada bagian ini kita akan membahas beberapa teknik penyortiran dan membandingkannya dengan waktu berjalannya.

Sebelum masuk ke algoritma spesifik, kita harus memikirkan operasi yang dapat digunakan untuk menganalisis proses penyortiran. Pertama, perlu membandingkan dua nilai untuk melihat mana yang lebih kecil (atau lebih besar). Untuk mengurutkan koleksi, perlu memiliki beberapa cara sistematis untuk membandingkan nilai untuk melihat apakah mereka rusak. Jumlah total perbandingan akan menjadi cara paling umum untuk mengukur prosedur penyortiran. Kedua, ketika nilai tidak dalam posisi yang benar sehubungan dengan satu sama lain, mungkin perlu untuk menukar mereka. Pertukaran ini adalah operasi yang mahal dan jumlah total pertukaran juga akan menjadi penting untuk mengevaluasi efisiensi keseluruhan algoritma.

5.7. The Bubble Sort

Jenis **gelembung** membuat beberapa melewati seluruh daftar. Ini membandingkan barang-barang yang berdekatan dan menukar barang-barang yang rusak. Setiap melewati daftar menempatkan nilai terbesar berikutnya di tempat yang tepat. Intinya, setiap item "menggelembung" ke lokasi di mana ia berada.

[Gambar 1](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheBubbleSort.html#fig-bubblepass) menunjukkan lintasan pertama semacam gelembung. Item yang diarsir sedang dibandingkan untuk melihat apakah mereka rusak. Jika ada *n* item dalam daftar, maka adan−1n−1pasang item yang perlu dibandingkan pada pass pertama. Penting untuk dicatat bahwa sekali nilai terbesar dalam daftar adalah bagian dari suatu pasangan, itu akan terus dipindahkan sampai pass selesai.



Gambar 1:: bubbleSortPass Pertama

Pada awal pass kedua, nilai terbesar sekarang ada di tempatnya. Adan−1n−1 item yang tersisa untuk disortir, artinya akan ada n−2n−2berpasangan. Karena setiap pass menempatkan nilai terbesar berikutnya, jumlah total pass yang diperlukan adalahn−1n−1. Setelah menyelesaikann−1n−1berlalu, barang terkecil harus berada di posisi yang benar tanpa perlu proses lebih lanjut. [ActiveCode 1](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheBubbleSort.html#lst-bubble) menunjukkan bubbleSortfungsi lengkap . Dibutuhkan daftar sebagai parameter, dan memodifikasinya dengan menukar item yang diperlukan.

Operasi pertukaran, kadang-kadang disebut "swap," sedikit berbeda dalam Python daripada di sebagian besar bahasa pemrograman lainnya. Biasanya, menukar dua elemen dalam daftar memerlukan lokasi penyimpanan sementara (lokasi memori tambahan). Fragmen kode seperti

temp = alist[i]

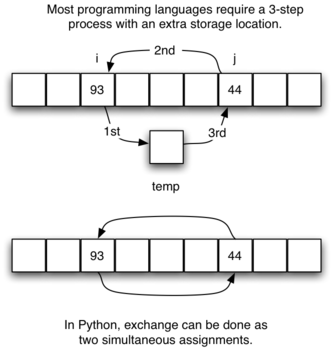
alist[i] = alist[j]

alist[j] = temp

akan menukar item engan dan j dalam daftar. Tanpa penyimpanan sementara, salah satu nilai akan ditimpa.

Dengan Python, dimungkinkan untuk melakukan tugas simultan. Pernyataan ini a,b=b,aakan menghasilkan dua pernyataan penugasan yang dilakukan pada saat yang sama (lihat [Gambar 2](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheBubbleSort.html#fig-pythonswap) ). Menggunakan penugasan simultan, operasi pertukaran dapat dilakukan dalam satu pernyataan.

Baris 5-7 di [ActiveCode 1](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheBubbleSort.html#lst-bubble) melakukan pertukaranii dan (i+1)th(i+1)thitem menggunakan prosedur tiga langkah yang dijelaskan sebelumnya. Perhatikan bahwa kami juga bisa menggunakan penugasan simultan untuk menukar item.



Gambar 2: Bertukar Dua Nilai dengan Python

Contoh kode aktivasi berikut menunjukkan bubbleSortfungsi lengkap yang bekerja pada daftar yang ditunjukkan di atas.

Simpan & JalankanMemuat RiwayatTampilkan CodeLens



1

def bubblesort ( alist ):

2

untuk passnum di kisaran ( len ( alist ) - 1 , 0 , - 1 ):

3

untuk saya dalam jangkauan ( passnum ):

4

jika alist [ i ] > alist [ i + 1 ]:

5

temp = alist [ i ]

6

alist [ i ] = alist [ i + 1 ]

7

alist [ i + 1 ] = temp

8

​

9

alist = [ 54 , 26 , 93 , 17 , 77 , 31 , 44 , 55 , 20 ]

10

bubblesort ( alist )

11

print ( alist )

12

​

Bubble Sort (lst\_bubble)

Animasi berikut ditampilkan bubbleSortdalam aksi.

Inisialisasi Menjalankan Berhenti   
Awal Maju Langkah Mundur Akhir

Untuk menganalisis semacam gelembung, kita harus perhatikan bahwa terlepas dari bagaimana item tersebut diatur dalam daftar awal, n−1n−1pass akan dibuat untuk mengurutkan daftar ukuran *n* . [Tabel 1](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheBubbleSort.html#tbl-bubbleanalysis)menunjukkan jumlah perbandingan untuk setiap pass. Jumlah total perbandingan adalah jumlah yang pertaman−1n−1bilangan bulat. Ingat bahwa jumlah bilangan bulat *n* pertama adalah12n2+12n12n2+12n. Jumlah yang pertaman−1n−1 bilangan bulat adalah 12n2+12n−n12n2+12n−n, yang mana 12n2−12n12n2−12n. Ini masih O(n2)O(n2)perbandingan. Dalam kasus terbaik, jika daftar sudah dipesan, tidak ada pertukaran yang akan dilakukan. Namun, dalam kasus terburuk, setiap perbandingan akan menyebabkan pertukaran. Rata-rata, kami bertukar setengah dari waktu.

| **Tabel 1: Perbandingan untuk Setiap Pass dari Bubble Sort** | |
| --- | --- |
| **Lulus** | **Perbandingan** |
| 1 | n−1n−1 |
| 2 | n−2n−2 |
| 3 | n−3n−3 |
| ... | ... |
| n−1n−1 | 11 |

Bubble sort sering dianggap metode penyortiran yang paling tidak efisien karena harus bertukar item sebelum lokasi akhir diketahui. Operasi pertukaran "sia-sia" ini sangat mahal. Namun, karena bubble sort membuat melewati seluruh bagian daftar yang tidak disortir, ia memiliki kemampuan untuk melakukan sesuatu yang kebanyakan algoritma sorting tidak dapat lakukan. Secara khusus, jika selama pass tidak ada pertukaran, maka kita tahu bahwa daftar harus disortir. Sortir gelembung dapat dimodifikasi untuk berhenti lebih awal jika menemukan bahwa daftar telah diurutkan. Ini berarti bahwa untuk daftar yang hanya memerlukan beberapa lintasan, semacam gelembung mungkin memiliki keuntungan karena ia akan mengenali daftar yang diurutkan dan berhenti. [ActiveCode 2](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheBubbleSort.html#lst-shortbubble) menunjukkan modifikasi ini, yang sering disebut sebagai **gelembung pendek** .

Simpan & JalankanMemuat RiwayatTampilkan CodeLens



1

def shortBubbleSort ( alist ):

2

pertukaran = Benar

3

passnum = len ( alist ) - 1

4

sementara passnum > 0 dan bertukar :

5

pertukaran = Salah

6

untuk saya dalam jangkauan ( passnum ):

7

jika alist [ i ] > alist [ i + 1 ]:

8

pertukaran = Benar

9

temp = alist [ i ]

10

alist [ i ] = alist [ i + 1 ]

11

alist [ i + 1 ] = temp

12

passnum = passnum - 1

13

​

14

alist = [ 20 , 30 , 40 , 90 , 50 , 60 , 70 , 80 , 100 , 110 ]

15

shortBubbleSort ( alist )

16

print ( alist )

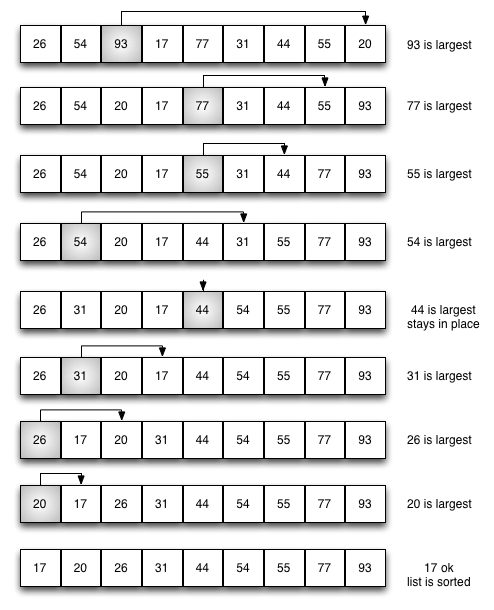
17

​

5.8. Pilihan Seleksi

Sortir **seleksi** meningkat pada sortir gelembung dengan membuat hanya satu pertukaran untuk setiap melewati daftar. Untuk melakukan ini, jenis seleksi mencari nilai terbesar saat membuat pass dan, setelah menyelesaikan pass, menempatkannya di lokasi yang tepat. Seperti halnya bubble sort, setelah pass pertama, item terbesar ada di tempat yang benar. Setelah umpan kedua, yang terbesar berikutnya adalah di tempat. Proses ini berlanjut dan membutuhkann−1n−1berlalu untuk mengurutkan *n* item, karena item terakhir harus ada di tempat setelah(n−1)(n−1) st pass.

[Gambar 3](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheSelectionSort.html#fig-selectionsort) menunjukkan seluruh proses penyortiran. Pada setiap pass, item yang tersisa terbesar dipilih dan kemudian ditempatkan di lokasi yang tepat. Pass pertama menempatkan 93, pass kedua menempatkan 77, yang ketiga menempatkan 55, dan seterusnya. Fungsi ini ditampilkan di [ActiveCode 1](https://runestone.academy/runestone/static/PraktikumAlprogIF18/SortSearch/TheSelectionSort.html#lst-selectionsortcode) .



Gambar 3: selectionSort

Simpan & JalankanMemuat RiwayatTampilkan CodeLens



1

def selectionSort ( alist ):

2

untuk fillslot di kisaran ( len ( alist ) - 1 , 0 , - 1 ):

3

positionOfMax = 0

4

untuk lokasi dalam jangkauan ( 1 , fillslot + 1 ):

5

jika alist [ location ] > alist [ positionOfMax ]:

6

positionOfMax = lokasi

7

​

8

temp = alist [ fillslot ]

9

alist [ fillslot ] = alist [ positionOfMax ]

10

alist [ positionOfMax ] = temp

11

​

12

alist = [ 54 , 26 , 93 , 17 , 77 , 31 , 44 , 55 , 20 ]

13

selectionSort ( alist )

14

print ( alist )

15

​

Sortir Pilihan (lst\_selectionsortcode)

Inisialisasi Menjalankan Berhenti   
Awal Maju Langkah Mundur Akhir

Anda dapat melihat bahwa sort seleksi membuat jumlah perbandingan yang sama dengan sort bubble dan karenanya juga O(n2)O(n2). Namun, karena pengurangan jumlah pertukaran, jenis seleksi biasanya dieksekusi lebih cepat dalam studi benchmark. Bahkan, untuk daftar kami, jenis gelembung membuat 20 pertukaran, sedangkan jenis seleksi hanya menghasilkan 8.