

Algoritma dan Struktur Data

Selection Sort

Umi Sa'adah

Tita Karlita

Entin Martiana Kusumaningtyas

Arna Fariza

2021



Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

Sorting Algorithms

1. Selection
2. Insertion
3. Bubble
4. Merge
5. Quick
6. Shell

Pendahuluan

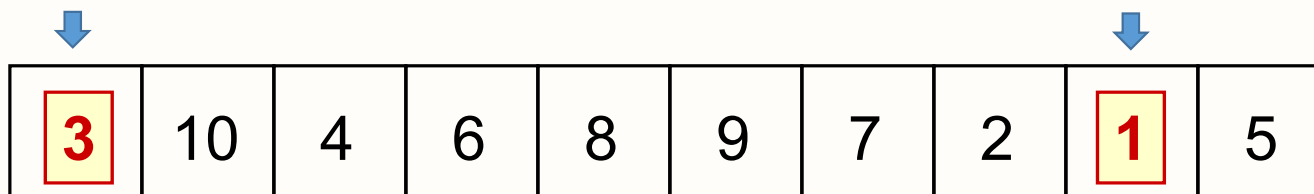
- *Sorting* merupakan salah satu hal penting di bidang pemrograman komputer.
- *Sorting* sering kali dimanfaatkan untuk mempermudah mendapatkan informasi tertentu secara cepat.
Contoh: sorting phone book berdasarkan nama
- *Selection sort* merupakan teknik *sorting* yang paling sederhana.

Selection Sort

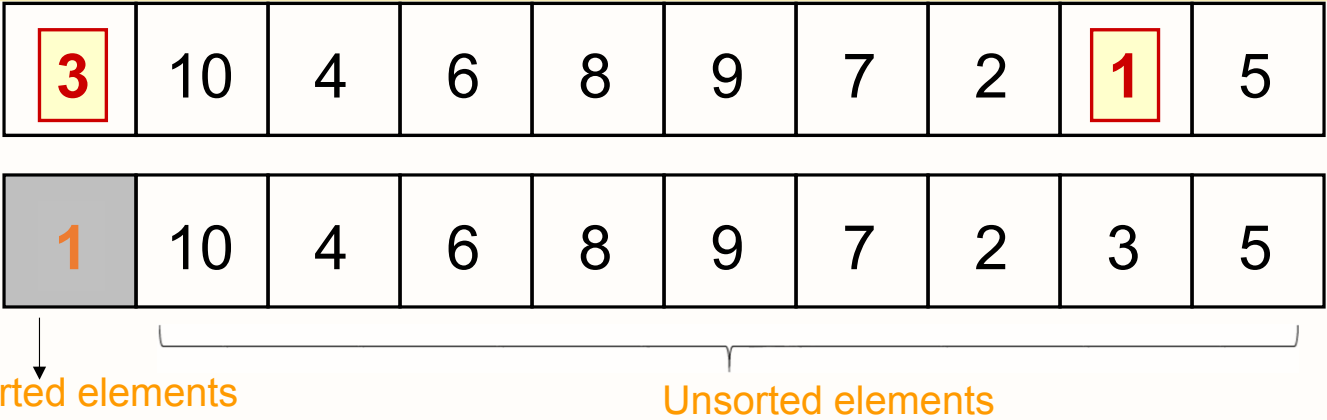
(one of the simplest sorting algorithms)

3	10	4	6	8	9	7	2	1	5
---	----	---	---	---	---	---	---	---	---

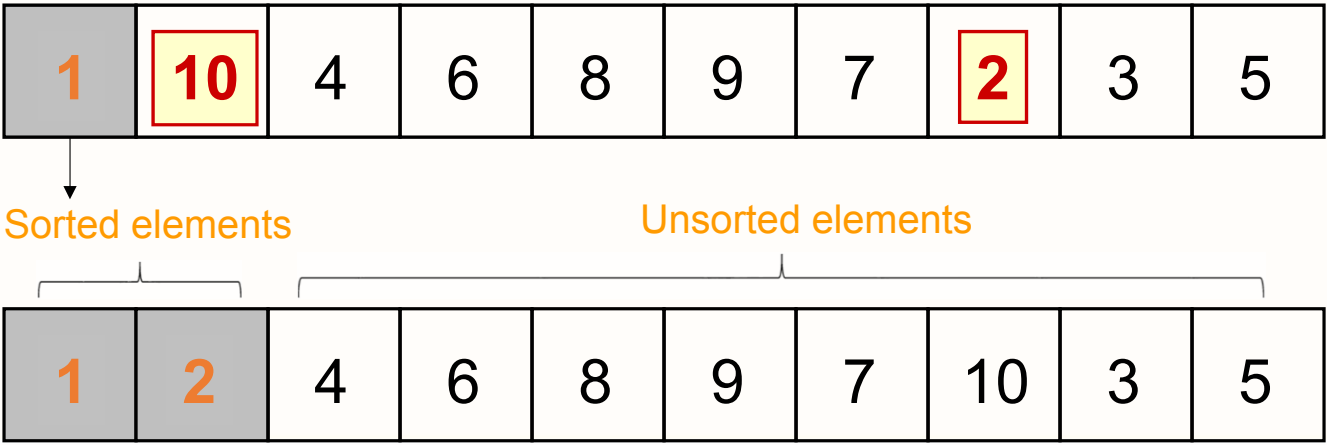
- Langkah pertama, cek seluruh elemen dalam array,
 - **Temukan** elemen dengan nilai terkecil (1)
 - Bandingkan apakah elemen dengan nilai terkecil memiliki nilai $<$ dari elemen pertama pada array ($1 < 3$) ?
 - Jika Ya, maka **tukarkan** elemen tersebut dengan elemen pertama dari array (3).
- **Ulangi** pola “**temukan**” dan “**tukar**” terhadap semua elemen, sampai semua elemen dalam array terurut.

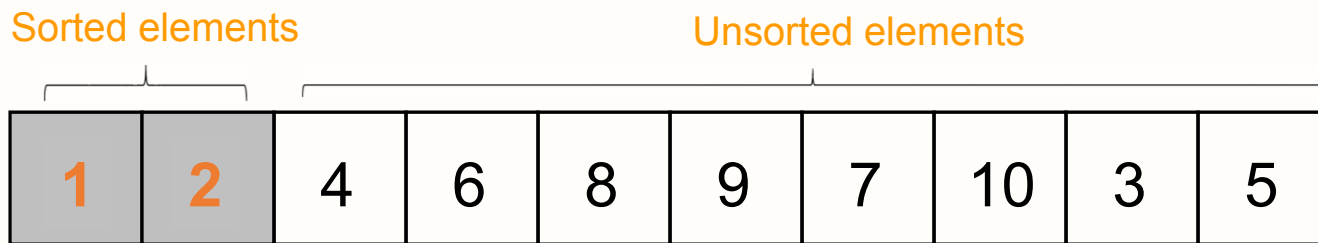


3	10	4	6	8	9	7	2	1	5
----------	----	---	---	---	---	---	---	----------	---



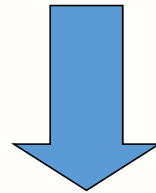
Temukan nilai terkecil kedua (2), dan tukarkan posisinya dengan nilai yang berada pada posisi kedua (10).





Dua elemen pertama dengan background abu-abu (1 dan 2) tidak akan berubah tempat lagi sebab mereka sudah merupakan nilai terkecil pertama dan kedua dalam array tersebut.

Sekarang, ulangi proses “**pilih dan tukar**” ...



1	2	4	6	8	9	7	10	3	5
1	2	3	6	8	9	7	10	4	5
1	2	3	6	8	9	7	10	4	5
1	2	3	4	8	9	7	10	6	5
1	2	3	4	8	9	7	10	6	5
1	2	3	4	5	9	7	10	6	8



1	2	3	4	5	9	7	10	6	8
---	---	---	---	---	---	---	----	---	---

1	2	3	4	5	6	7	10	9	8
---	---	---	---	---	---	---	----	---	---

1	2	3	4	5	6	7	10	9	8
---	---	---	---	---	---	---	----	---	---

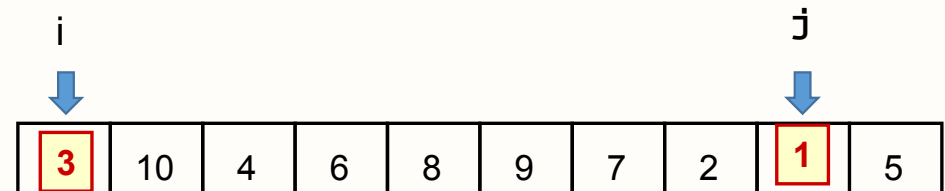
1	2	3	4	5	6	7	10	9	8
---	---	---	---	---	---	---	----	---	---

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Algoritma Metode Selection Sort

1. $i \leftarrow 0$
2. selama ($i < N$) kerjakan baris 3 sd 12
3. $min \leftarrow i$
4. $j \leftarrow i + 1$
5. Selama ($j < N$) kerjakan baris 6 sd 8
6. Jika ($A[j] < A[min]$) kerjakan 7
7. $min \leftarrow j$
8. $j \leftarrow j + 1$
9. $temp \leftarrow A[i]$
10. $A[i] \leftarrow A[min]$
11. $A[min] \leftarrow temp$
12. $i \leftarrow i + 1$



Pseudo Code

```
SelectionSort(A, n) {  
  for i = 0 to n-1 {  
    min = i                                //Assign indeks i sebagai min  
    //bandingkan elemen pd indeks j dgn indeks min  
    for j = i +1 to n-1  
      if A[j] < A[min]                    //jika elemen j lbh kecil dr elemen min  
        min = j                          //update nilai min menjadi j  
      j = j+1                            //Ulangi sampai nilai j sama dgn n  
    temp = A[i]                          //menukarkan 2 elemen :  
    A[i] = A[min]                        // A[i] dengan A[min]  
    A[min] = temp  
  }                                     //Ulangi sampai nilai i sama dgn n-1  
}
```



Selection Sort & Analysis

- Berdasarkan algoritma selection sort, loop for terluar dilakukan sebanyak $n-1$ kali
- Pada setiap iterasi, dilakukan satu kali penukaran elemen, sehingga :
 - Total penukaran/swap = $n-1$
 - Total pergeseran = $3 * n-1$
(pada setiap penukaran terjadi 3 x pergeseran)
- Jumlah perbandingan pada metode ini adalah
 - $= 1 + 2 + \dots + n-1$
 - $= n * (n-1) / 2$



Selection Sort & Analysis

- Secara umum, yang dilakukan dalam metode seleksi adalah **pembandingan** key (elemen pada posisi min) serta **penukaran** elemen.
- Sehingga untuk menganalisis metode ini harus dihitung jumlah pembandingannya serta jumlah penukaran elemennya.
- Dalam metode ini, jumlah pembandingan untuk **best case & worst casenya sama**.
- Memindahkan dari kanan ke kiri, meletakkan elemen ke posisi finalnya tanpa merevisi lagi posisi tersebut.
- Menghabiskan sebagian besar waktu untuk mencari elemen terkecil pada sisi array yang belum terurut.



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
<u>Quicksort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$



Big-O Complexity Chart

