Algoritma dan Struktur Data

Insertion Sort

Umi Sa'adah
Tita Karlita
Entin Martiana Kusumaningtyas
Arna Fariza
2021

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

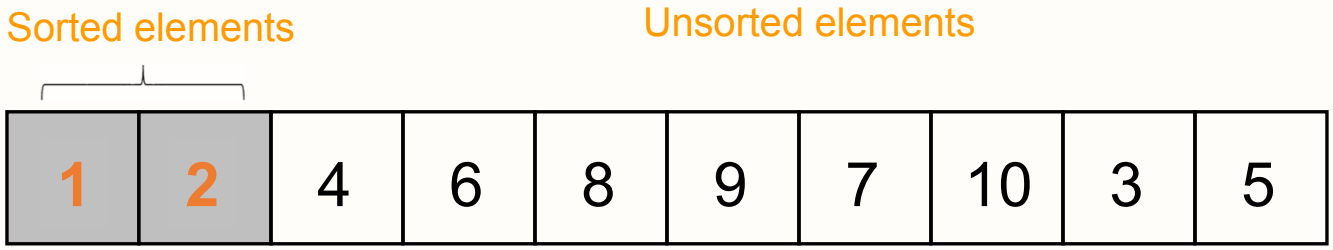
Sorting Algorithms

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

Insertion Sort

- Metode penyisipan (insertion sort) bertujuan untuk menjadikan bagian sisi kiri array terurutkan sampai dengan seluruh array berhasil diurutkan.

- Metode ini mengurutkan bilangan-bilangan yang telah dibaca; dan berikutnya secara berulang akan menyisipkan bilangan-bilangan dalam array yang belum terbaca ke sisi kiri array yang telah terurut.

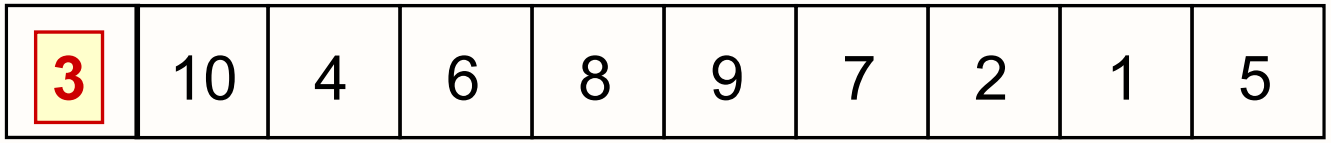


Insertion Sort

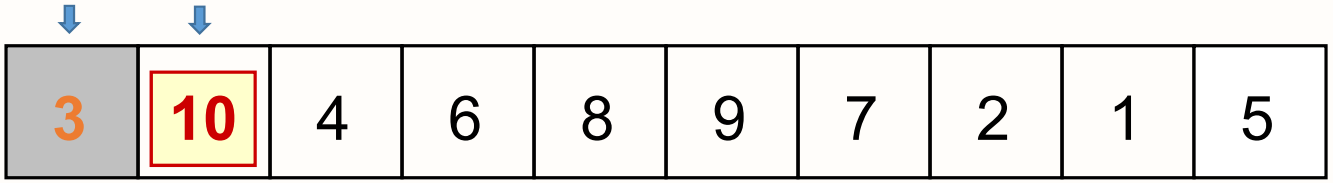
3	10	4	6	8	9	7	2	1	5

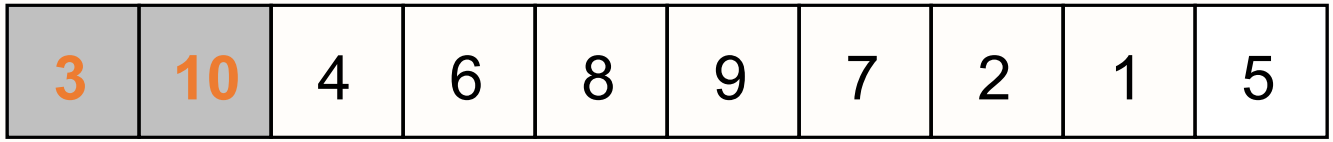
- Elemen paling kiri (3) bisa dikatakan telah terurut secara relatif terhadap dirinya sendiri.
- Thus, we don’t need to do anything.

- Elemen paling kiri (3) telah terurut secara relatif terhadap dirinya sendiri.

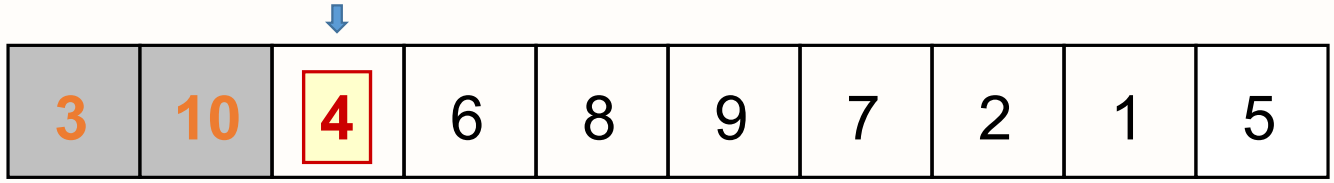


- Cek, untuk melihat apakah elemen kedua (10) lebih kecil dari pada yang pertama (3).
- Jika ya, tukarkan kedua bilangan ini.
- Namun, pada kasus ini kita tidak perlu melakukan penukaran, karena 10 lebih besar dari 3.





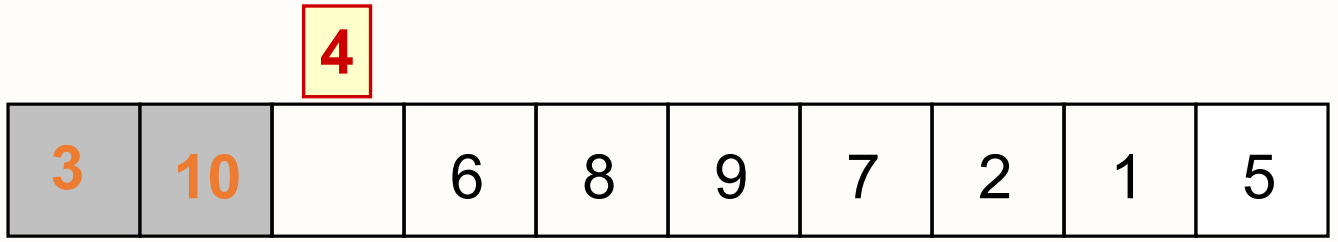
Bagian abu-abu (dua bilangan pertama) sekarang sudah dalam keadaan terurut secara relatif.



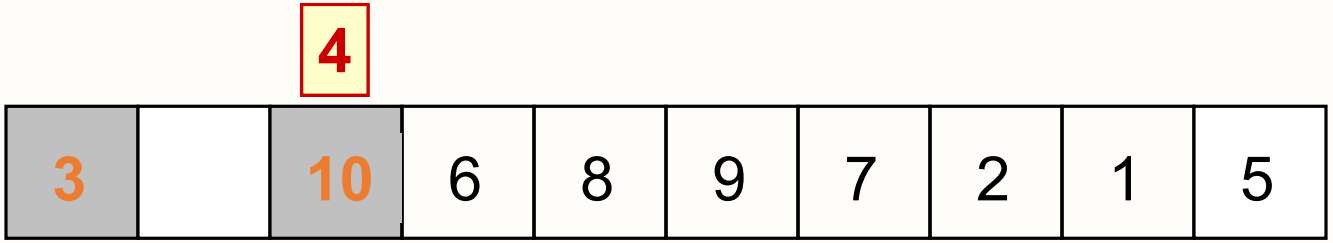
Berikutnya, kita bergeser ke elemen ketiga (4).
Kita perlu menyisipkan bilangan ketiga (4) ke dalam bagian abu-abu sehingga setelah penyisipan tersebut, bagian abu-abu tetap dalam keadaan terurut secara relatif;

CARANYA…

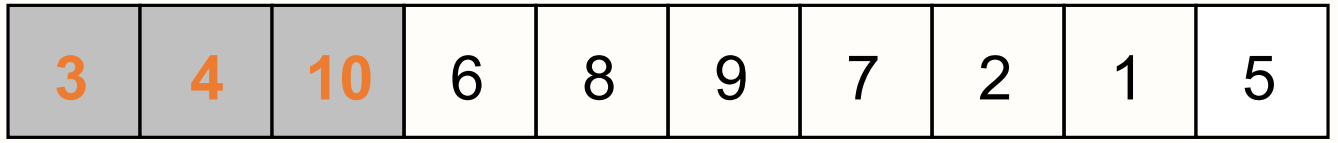
Pertama : Ambil bilangan ketiga (4) dan bandingkan dengan bilangan yang telah terurut relatif untuk menempatkan bilangan ketiga (4) pada tempatnya. Karena 4<10 dan 4>3, maka sisipkan 4 ke posisi 10.



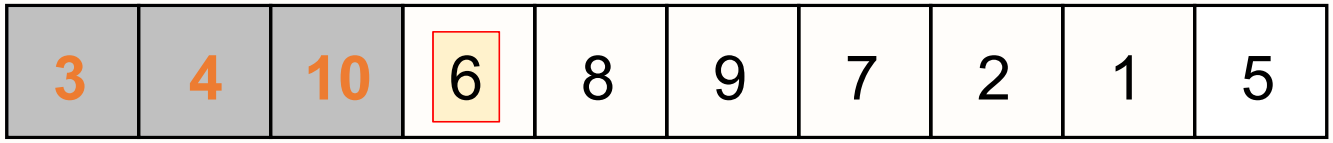
Kedua : Geser bilangan kedua (10) shg ada ruang untuk disisipi.



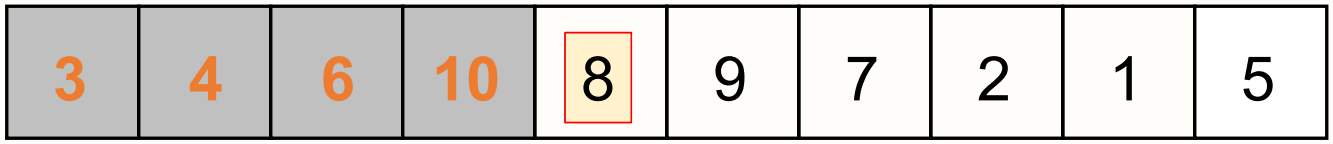
Ketiga : Sisipkan bilangan 4 ke posisi yang tepat



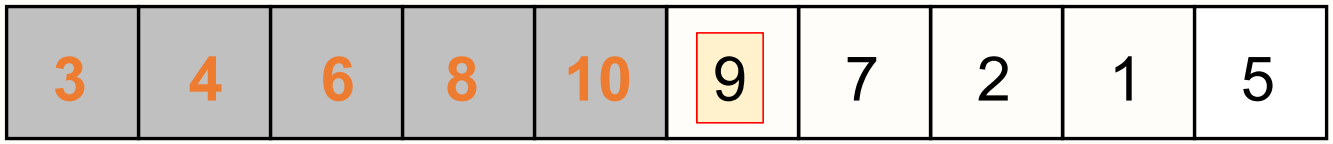
Sekarang, tiga bilangan pertama sudah terurut secara relatif

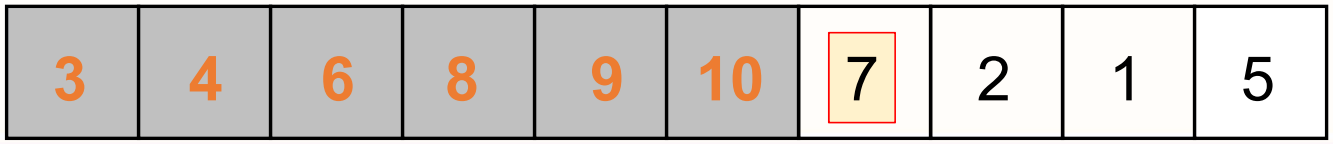


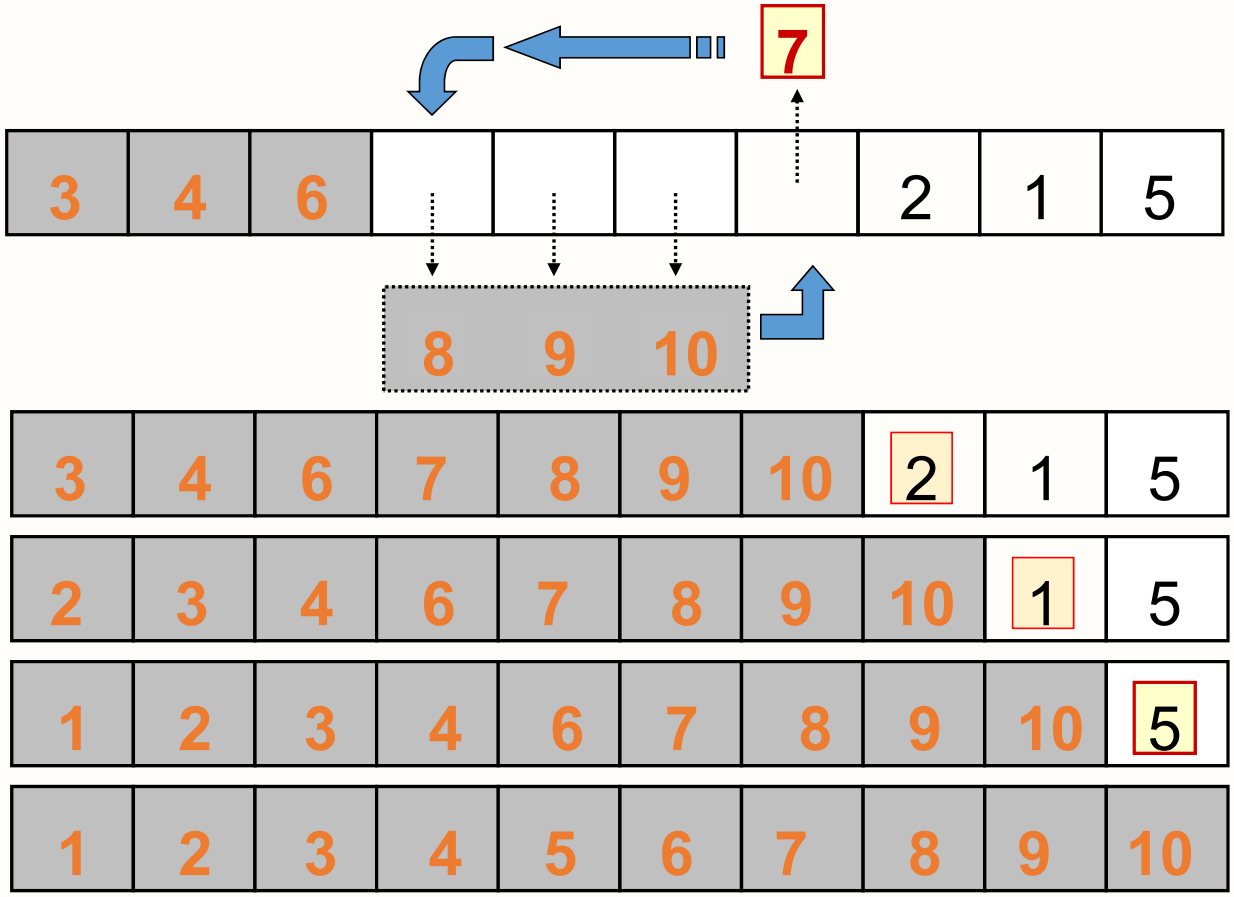
- Selanjutnya sisipkan bilangan keempat (6) ke tiga bilangan pertama yang telah terurut relatif.
- Setelah penyisipan, empat bilangan pertama haruslah dalam keadaan terurut secara relatif.



- Ulangi proses sampai bilangan terakhir disisipkan.






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

Algoritma Metode Penyisipan

1. **i ← 1**
2. selama **(i < n)** kerjakan baris 3 sampai dengan 9
3. **key ← A[i]**
4. **j ← i – 1**
5. selama **j >= 0** dan **(A[j] > key)** kerjakan baris 6 & 7
6. **A[j + 1] ← A[j]**
7. **j ← j – 1**
8. **A[j+1] ← key**
9. **i ← i + 1**

Pseudo Code

```
InsertionSort(A, n) {
  FOR i = 1 TO n {
        key = A[i]              //Assign elemen array indeks i ke key
        j = i - 1               //Inisialisasi j utk pembandingan
        //bandingkan elemen array pd indeks j dgn key
        //if j >= 0 dan elemen indeks j > key
        WHILE (j >= 0) and (A[j] > key) {
            A[j+1] = A[j] //pindahkan elemen tsb ke 1 posisi berikutnya
            j = j – 1           //go to next lower element
        }                       //Lanjutkan sampai A[j] not > key
        A[j+1] = key            //assign temp kembali ke array
  }
}
```

Insertion Sort Analysis

- Running time bukan hanya bergantung pada ukuran array, namun juga pada susunan isi nya

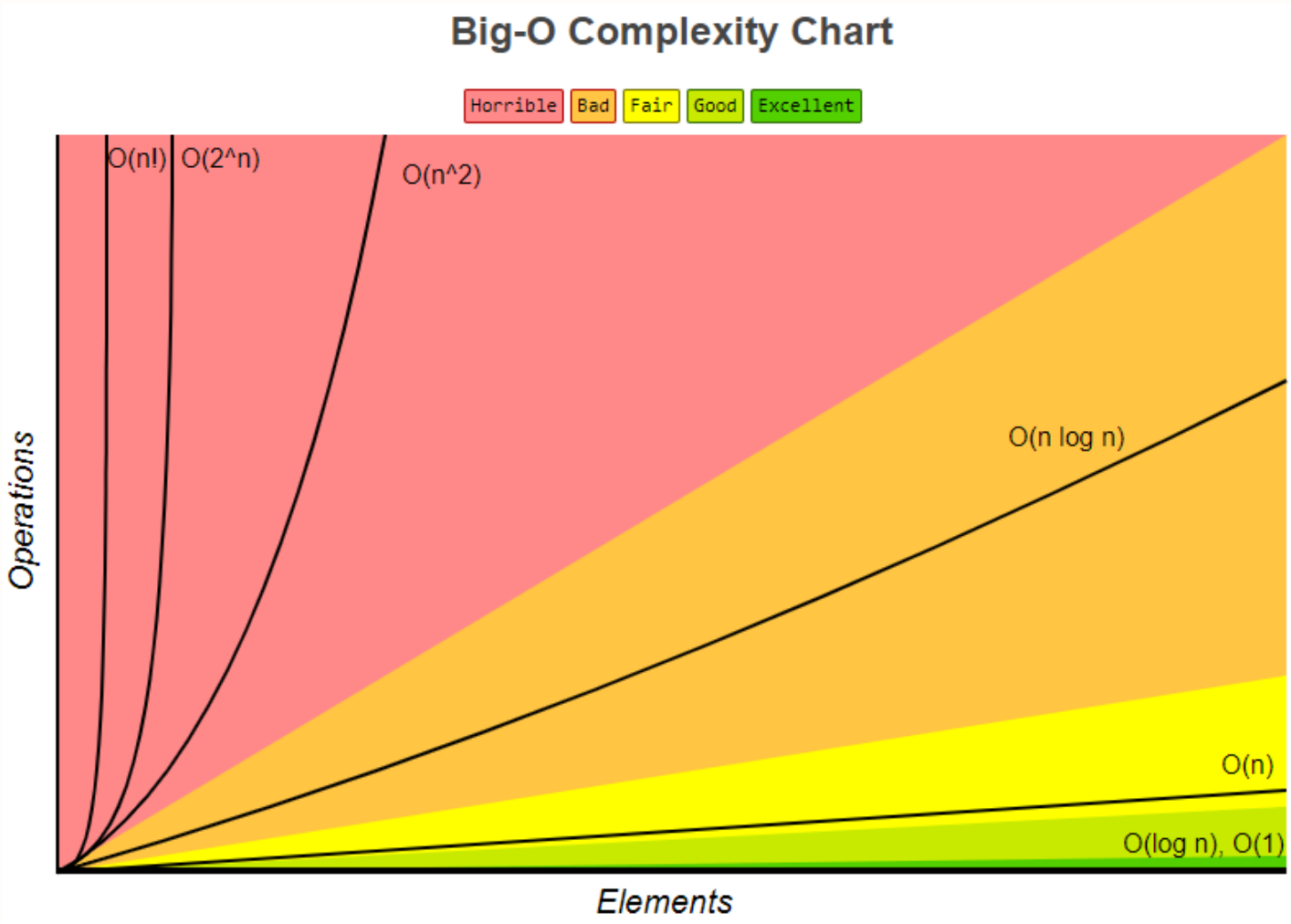
- BEST CASE:
  - Array sudah dalam keadaan terurut naik
  - Loop terdalam tidak pernah dieksekusi
  - Jumlah pergeseran : 2(n-1)
  - Jumlah pembandingan key (C) : (n-1)

Insertion Sort Analysis

- WORST CASE
  - Array dalam urutan kebalikannya
  - Loop terdalam dieksekusi sebanyak p-1 kali,
    untuk p = 2,3,..,n
  - Jumlah pergeseran :
    $2(n-1) + (1 + 2 + .. + n-1) = 2(n-1) + n * (n-1) / 2$
  - Jumlah pembandingan key :
    $(1 + 2 + .. + n-1) = n * (n-1) / 2$

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements