Sorting

Teaching Team
Algorithm and Data Structure
2022/2023
Information Technology Department

Topics

- Bubble Sort
- Selection Sort
- Insertion Sort

Look for number 5!
Can you find it?

Difficult to find a book ????
Need a lot of time ????

Can you see the comparison ???

SORTING

- **Sorting**: is a very classic problem of reordering items (*that can be compared, e.g. integers, floating-point numbers, strings, etc*) of an array (or a list) in a certain order.
- **Ascending** order or **Descending** order
- **Lexicographical** order
- Sorting is commonly used as the introductory problem in various Computer Science classes to showcase a range of algorithmic ideas

Sorting

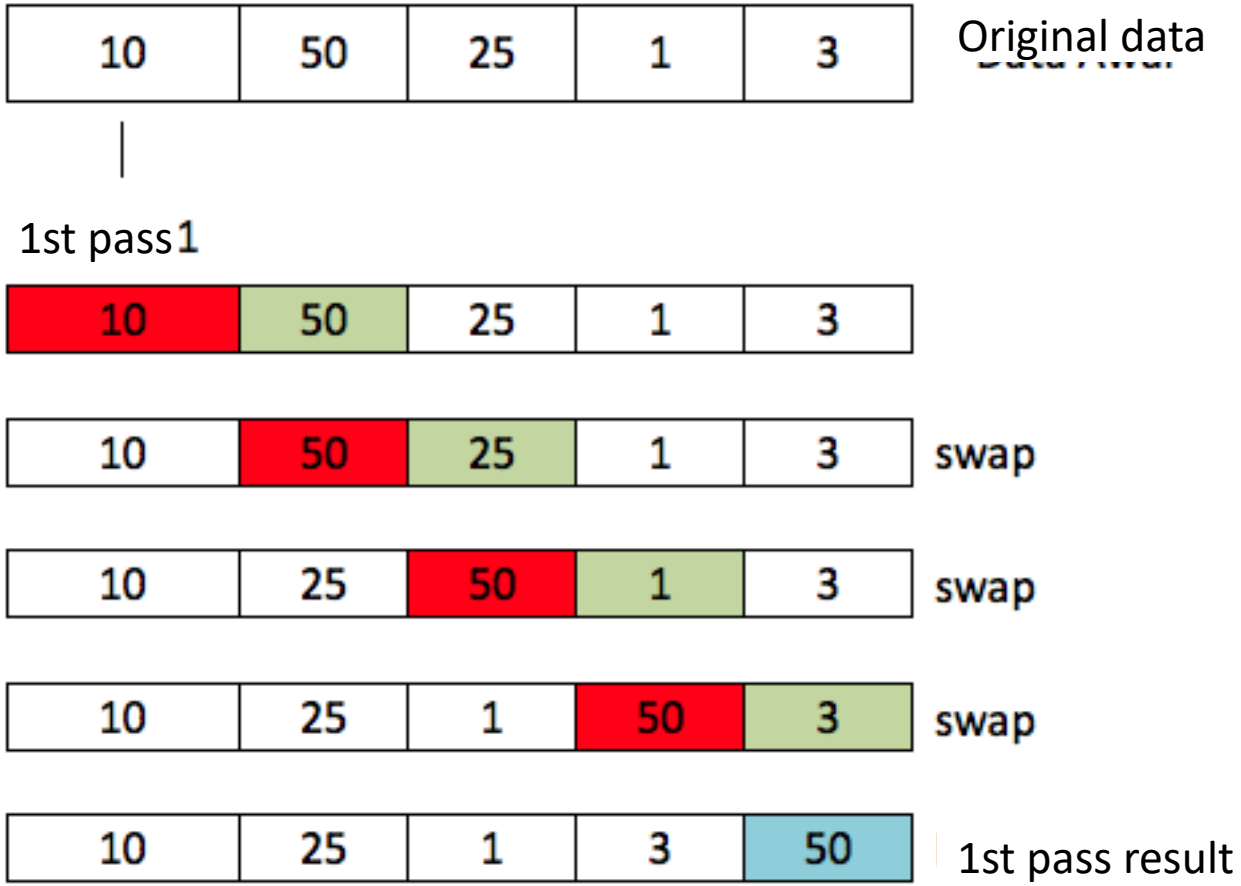
- Unsorted data:
  20, 1, 56, 30, 10, 15
- Sorted data in Ascending order
  1, 10, 15, 20, 30, 56
- Sorted data in Descending order
  56, 30, 20, 25, 10, 1

Bubble Sort

Bubble Sort

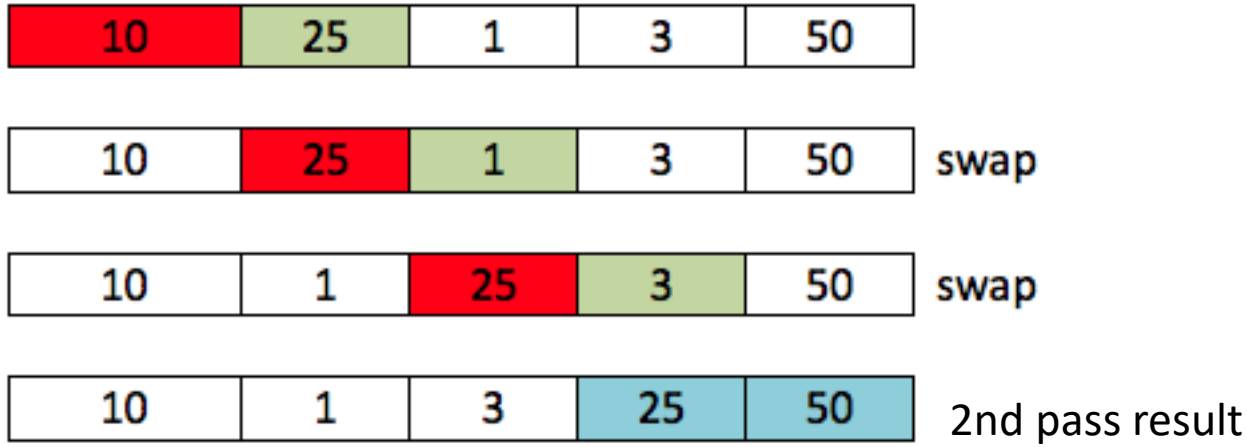
- The most simple algorithm to implement
- Due to its simplicity, bubble sort is often used to introduce the concept of a sorting algorithm
- Less effective than other algorithms
- The bubble sort makes **multiple passes** through a list. It will be **n-1 passes**, where **n** is number of data in list
- In each pass, It compares adjacent items and exchanges (**SWAP**) those that are out of order. It will be n-i comparations, where **n** is number of data in list and **i** is **i**$_{th}$ pass.
- Each pass through the list places the next largest value in its proper place.
- In essence, each item “bubbles” up to the location where it belongs.

Bubble Sort

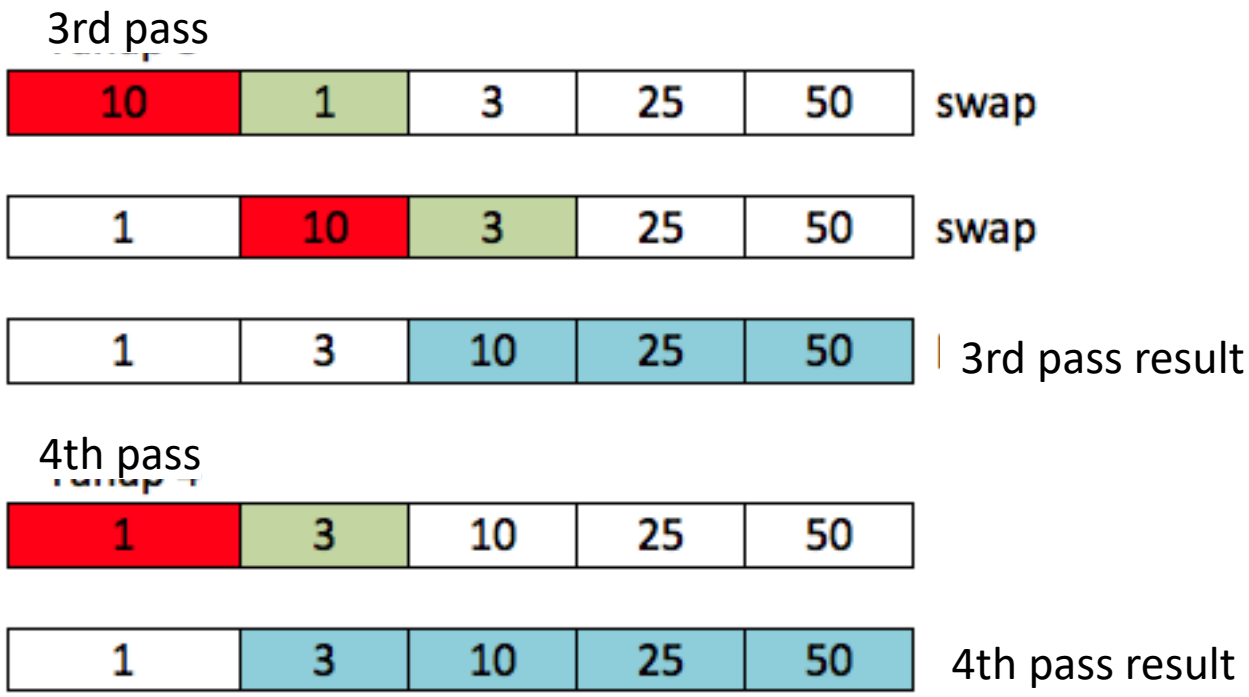

10 50 25 1 3
Original data
1st pass 1
10 50 25 1 3
10 50 25 1 3
swap
10 25 50 1 3
swap
10 25 1 50 3
swap
10 25 1 3 50
1st pass result

Bubble Sort

2nd pass



Bubble Sort


3rd pass
10 1 3 25 50
swap
1 10 3 25 50
swap
1 3 10 25 50
3rd pass result
4th pass
1 3 10 25 50
1 3 10 25 50
4th pass result

Bubble Sort Example (8 data)

```
[6, 5, 3, 1, 8, 7, 2, 4]
```

1st pass
7 steps

```
> 6 -> 5 swap : [5, 6, 3, 1, 8, 7, 2, 4]
> 6 -> 3 swap : [5, 3, 6, 1, 8, 7, 2, 4]
> 6 -> 1 swap : [5, 3, 1, 6, 8, 7, 2, 4]
> 6 -> 8 no swap : [5, 3, 1, 6, 8, 7, 2, 4]
> 8 -> 7 swap : [5, 3, 1, 6, 7, 8, 2, 4]
> 8 -> 2 swap : [5, 3, 1, 6, 7, 2, 8, 4]
> 8 -> 4 swap : [5, 3, 1, 6, 7, 2, 4, 8]
```

2nd pass
6 steps

```
> 5 -> 3 swap : [3, 5, 1, 6, 7, 2, 4, 8]
> 5 -> 1 swap : [3, 1, 5, 6, 7, 2, 4, 8]
> 5 -> 6 no swap : [3, 1, 5, 6, 7, 2, 4, 8]
> 6 -> 7 no swap : [3, 1, 5, 6, 7, 2, 4, 8]
> 7 -> 2 swap : [3, 1, 5, 6, 2, 7, 4, 8]
> 7 -> 4 swap : [3, 1, 5, 6, 2, 4, 7, 8]
```

3rd pass
5 steps

```
> 3 -> 1 swap : [1, 3, 5, 6, 2, 4, 7, 8]
> 3 -> 5 no swap : [1, 3, 5, 6, 2, 4, 7, 8]
> 5 -> 6 no swap : [1, 3, 5, 6, 2, 4, 7, 8]
> 6 -> 2 swap : [1, 3, 5, 2, 6, 4, 7, 8]
> 6 -> 4 swap : [1, 3, 5, 2, 4, 6, 7, 8]
```

4th pass
4 steps

```
> 1 -> 3 no swap : [1, 3, 5, 2, 4, 6, 7, 8]
> 3 -> 5 no swap : [1, 3, 5, 2, 4, 6, 7, 8]
> 5 -> 2 swap : [1, 3, 2, 5, 4, 6, 7, 8]
> 5 -> 4 swap : [1, 3, 2, 4, 5, 6, 7, 8]
```

5th pass
3 steps

```
> 1 -> 3 no swap : [1, 3, 2, 4, 5, 6, 7, 8]
> 3 -> 2 swap : [1, 2, 3, 4, 5, 6, 7, 8]
> 3 -> 4 no swap : [1, 2, 3, 4, 5, 6, 7, 8]
```

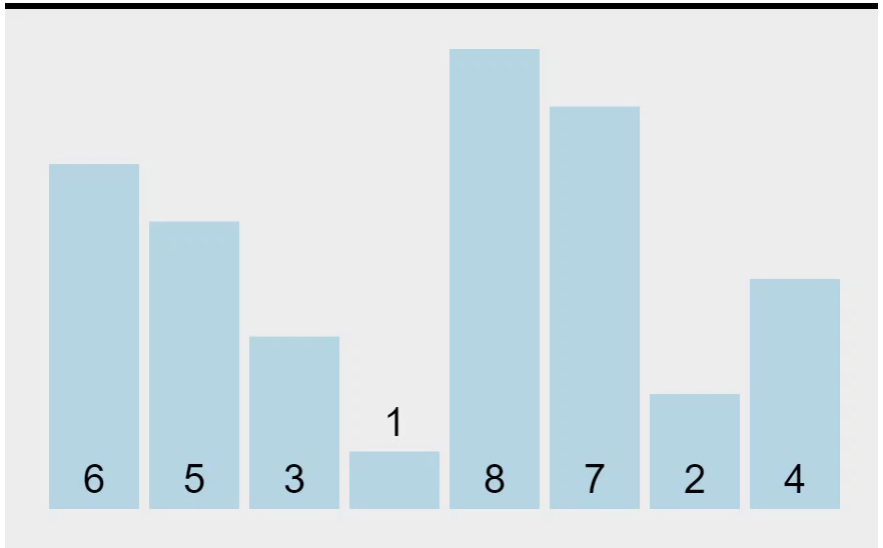
6th pass
2 steps

```
> 1 -> 2 no swap : [1, 2, 3, 4, 5, 6, 7, 8]
> 2 -> 3 no swap : [1, 2, 3, 4, 5, 6, 7, 8]
```

7th pass
1 step

```
> 1 -> 2 no swap : [1, 2, 3, 4, 5, 6, 7, 8]
```

Bubble Sort Visualisation



Bubble Sort

```
Bubble Sort(arr, size)
for i←0 to size-1
      for j←0 to size-i-1
            if arr[j]>arr[j+1]
                  swap arr[j] and arr[j+1]

return (arr)
```

Bubble Sort

```
38      static void bubbleSort(int[] data){
39          //perulangan sejumlah n-1 tahap pengurutan
40          for(int i=0;i<data.length-1;i++){
41              //perulangan sejumlah n-i-1 langkah pembandingan
42              for(int j=0;j<data.length-i-1;j++){
43                  if(data[j]>data[j+1]){
44                      int tmp = data[j];
45                      data[j] = data[j+1];
46                      data[j+1] = tmp;
47                  }
48              }
49          }
50      }
```

Advantages of Bubble Sort

1. Bubble sort calculation process is the simplest method
2. The Bubble Sort algorithm is easy to understand
3. The steps or stages in data sorting are very simple.

Disadvantages Bubble Sort

1. The Bubble Sort calculation process using the sorting method is among the least efficient even though it is considered simple. Because the data sorting process is done in stages one - by one, starting from the earliest data to the left, to the last data
2. When we have a lot of data or in large numbers, the calculation process will be longer and slower. Because the process of sorting data singly (one by one).
3. The number of repetitions will remain the same until the last data, even though some of the existing data has been sorted.

Selection Sort

Selection Sort

- Selection sort is the combination of sorting and searching algorithm
- The selection sort improves on the bubble sort by making only **one exchange (swap)** for **every pass** through the list
- In order to do this, a selection sort looks for the largest/smallest value as it makes a pass and, after completing the pass, places it in the proper location
- It will be **n-1** passes as well

Selection Sort Example

Data = {10,14,27,35,42,19,33,29}

1st pass

10	14	27	35	42	19	33	29

Index = 0 ;
minIndex = 0; minValue = 10

- 14 < 10
- 27 < 10
- 35 < 10
- 42 < 10
- 19 < 10
- 33 < 10
- 29 < 10

Swap index 0 with minIndex 0 {**10**,14,27,35,42,19,33,29}

2nd pass

10	14	27	35	42	19	33	29

Index = 1 ;
minIndex = 1; minValue = 14

- 27 < 14
- 35 < 14
- 42 < 14
- 19 < 14
- 33 < 14
- 29 < 14

Swap index 1 with minIndex 1 {**10**,**14**,27,35,42,19,33,29}

Selection Sort Example

3rd pass

10	14	**27**	**35**	**42**	19	**33**	**29**

Index = 2 ;
minIndex = 2; minValue = 27

- 35 < 27
- 42 < 27
- 19 < 27 (minValue = 19, minIndex = 5)
- 33 < 19
- 29 < 19

Swap index 2 with minIndex 5 {**10**,**14**,**19**,35,42,27,33,29}

4th pass

10	14	19	**35**	**42**	27	**33**	**29**

Index = 3 ;
minIndex = 3; minValue = 35

- 42 < 35
- 27 < 35 (minValue = 27, minIndex = 5)
- 33 < 27
- 29 < 27

Swap index 4 with minIndex 5 {**10**,**14**,**19**,**27**,42,35,33,29}

Selection Sort Example

5th pass

10	14	19	27	**42**	**35**	**33**	29

Index = 4 ;
minIndex = 4; minValue = 42
- 35 < 42 (minValue = 35, minIndex = 5)
- 33 < 35 (minValue = 33, minIndex = 6)
- 29 < 33 (minValue = 29, minIndex = 7)

Swap index 4 with minIndex 7 {**10**,**14**,**19**,**27**,**29**,35,33,42}

6th pass

10	14	19	27	29	**35**	33	**42**

Index = 5 ;
minIndex = 5; minValue = 35
- 33 < 35 (minValue = 33, minIndex = 6)
- 42 < 33

Swap index 5 with minIndex 6 {**10**,**14**,**19**,**27**,**29**,**33**,35,42}

7th pass

10	14	19	27	29	33	**35**	**42**

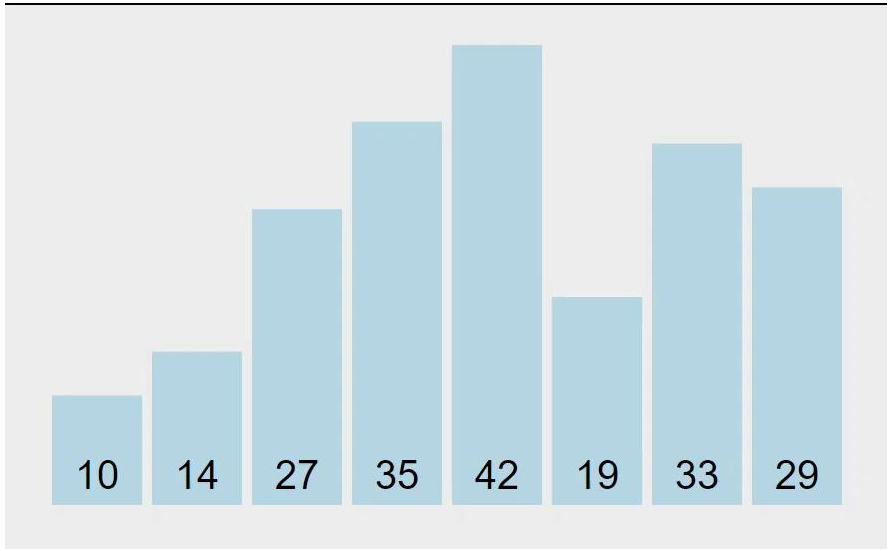
Index = 6 ;
minIndex = 6; minValue= 35
- 42 < 35

Swap index 6 with minIndex 6 {**10**,**14**,**19**,**27**,**29**,**33**,**35**,**42**}

Hasil Akhir Pengurutan

10	14	19	27	29	33	35	42

Visualisasi SelectionSort



Selection Sort

```
Selection Sort(arr, size)
for i←0 to size-1
      minIndex ← i
      minValue ← arr[i]
      for j←i+1 to size-1
            if arr[j]< minValue
                  minIndex ← j
                  minValue ← arr[j]
      swap arr[i] and arr[minIndex]

return (arr)
```

Selection Sort

```
51      static void selectionSort(int[] data){
52          //perulangan tahap pengurutan n-1 kali
53          for(int i=0;i<data.length-1;i++){
54              //perulangan cari nilai min
55              int minValue = data[i];
56              int idxMin = i;
57              for(int j=i+1;j<data.length;j++){
58                  if(data[j]<minValue){
59                      minValue = data[j];
60                      idxMin = j;
61                  }
62              }
63              //swap
64              int tmp = data[i];
65              data[i] = data[idxMin];
66              data[idxMin] = tmp;
67          }
68      }
```

Advantages of Selection Sort

1. This algorithm is more efficient than bubble sort and easy to implement.
2. The exchange (SWAP) operation is performed only once.
3. Sorting time can be reduced.
4. The complexity of selection sort is relatively smaller.

Disadvantages of Selection Sort

1. Inefficient: as with bubble sort, selection sort requires as many steps as n to sort the data.
2. Selection sort performance is affected by the initial sequence of data before the sorting process.

Insertion Sort

Insertion Sort

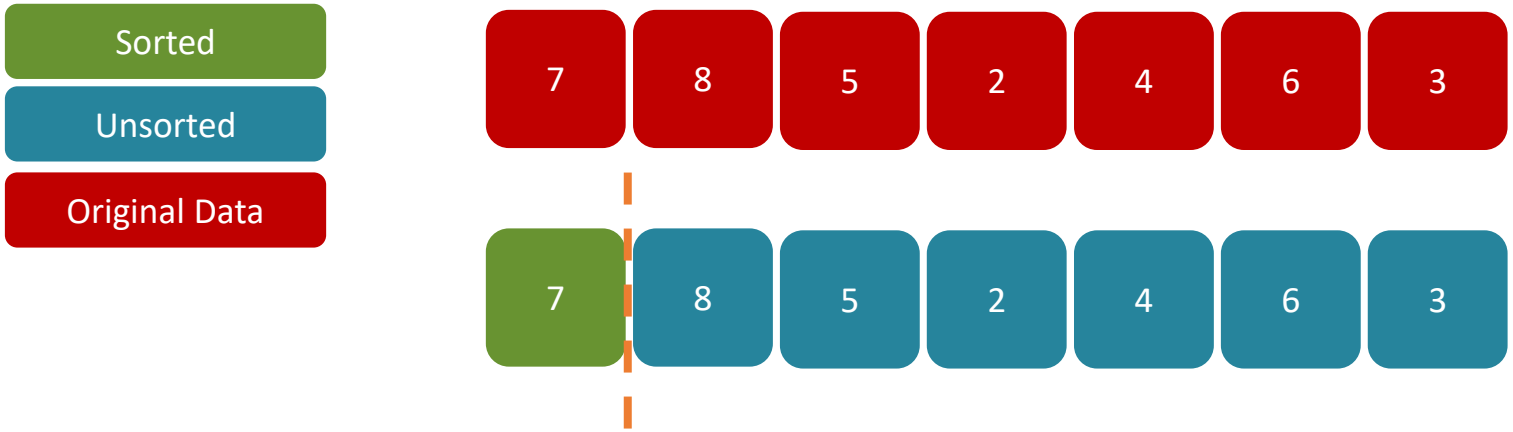
- It always maintains a sorted sublist in the lower positions of the list.
- Each new item is then “inserted” back into the previous sorted sublist so that the sorted sublist keeps in sorted mode

Insertion Sort

It inserts the data in the proper position, by the following steps:

1. Take **i-th** data and save it as **temp**
2. Compare the data in **temp** with the left adjacent data one by one (sorted sublist)
3. Check if **temp** is smaller than the left adjacent data.
4. If step 3 return "**true**" then shift one by one until data that is smaller than the **temp** is found, and then at that position **temp** will be inserted
5. Repeat step 1 to 4, until **i** is equal with **n** (until the last data)

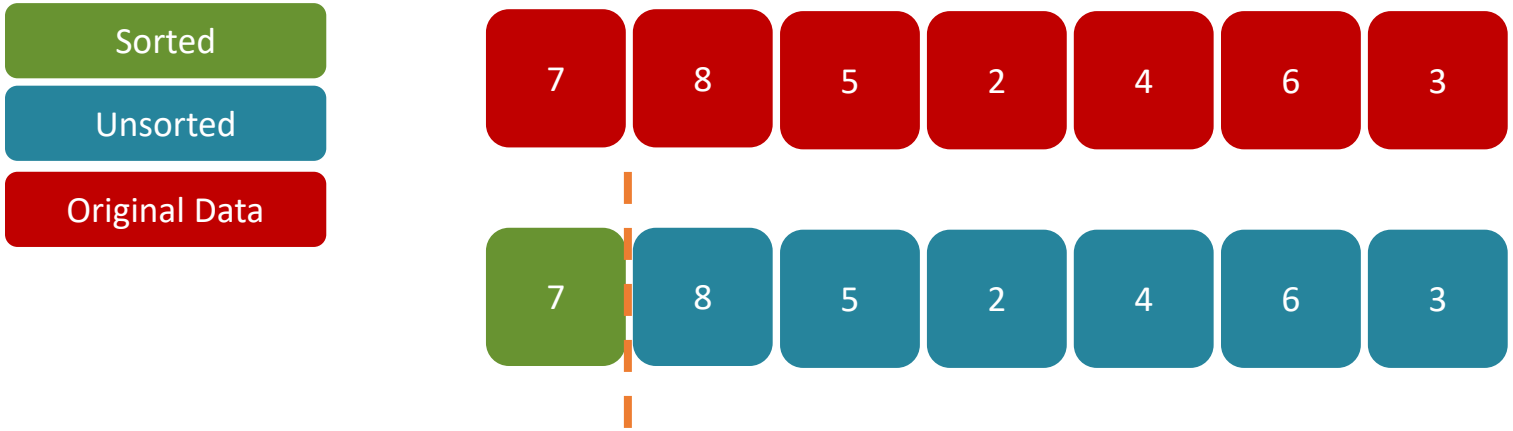
Insertion Sort Example



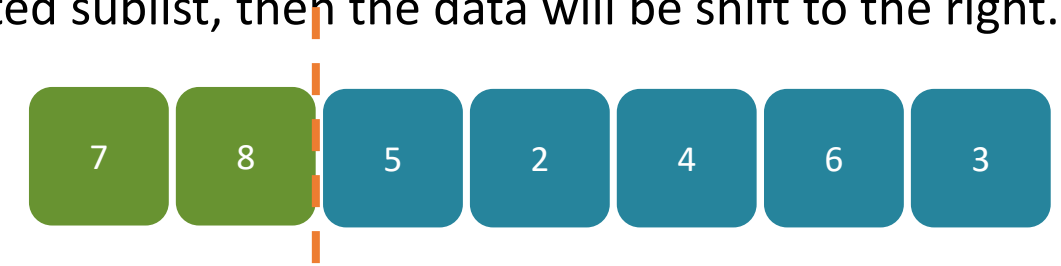
1st Pass :

First data will be initial data for sorted sublist. And the remaining data data will be unsorted sublist

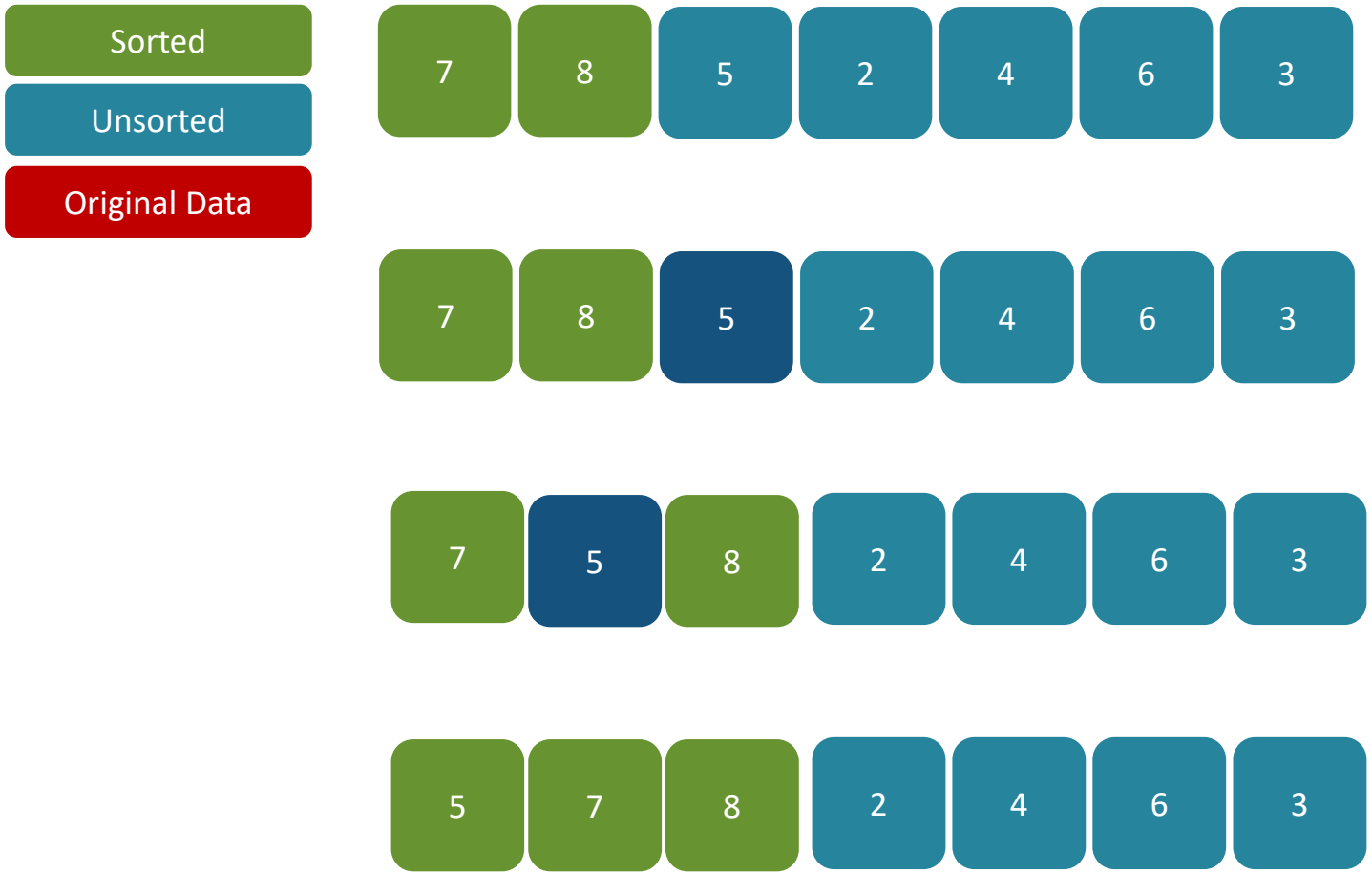
Insertion Sort Example (2)



2nd Pass : the second data is saved temporarily in **temp**, then it will be compared to the sorted sublist data, starting from the most right data in the sorted sublist data. If the temp is smaller than data in the sorted sublist, then the data will be shift to the right.



Insertion Sort Example (3)



3rd Pass :

Save the 3rd data to **temp**, then compare it to each value in the sorted sublist starting from the most right data, if the value in the sorted sublist is greater, it will be shifted to the right. It will continue until finding the smaller data in the sorted sublist or no more data in the sublist that will be compared with, the **temp** will be inserted in that position

Insertion Sort Example (4)

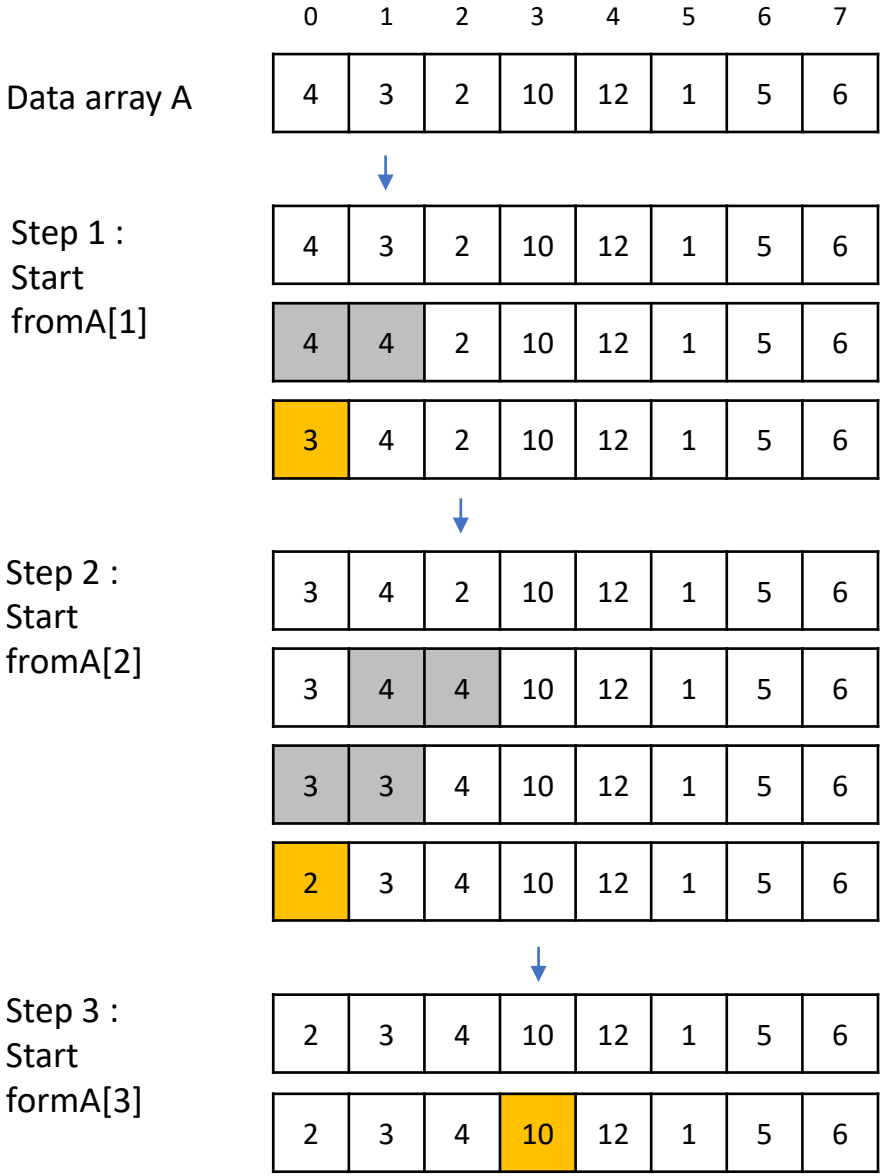

```
> 8 -> 2 : [5, 7, 8, 8, 4, 6, 3]
> 7 -> 8 : [5, 7, 7, 8, 4, 6, 3]
> 5 -> 8 : [5, 5, 7, 8, 4, 6, 3]
       : [2, 5, 7, 8, 4, 6, 3]
```


```
> 8 -> 4 : [2, 5, 7, 8, 8, 6, 3]
> 7 -> 8 : [2, 5, 7, 7, 8, 6, 3]
> 5 -> 8 : [2, 5, 5, 7, 8, 6, 3]
       : [2, 4, 5, 7, 8, 6, 3]
```


```
> 8 -> 6 : [2, 4, 5, 7, 8, 8, 3]
> 7 -> 8 : [2, 4, 5, 7, 7, 8, 3]
       : [2, 4, 5, 6, 7, 8, 3]
```


```
> 8 -> 3 : [2, 4, 5, 6, 7, 8, 8]
> 7 -> 8 : [2, 4, 5, 6, 7, 7, 8]
> 6 -> 8 : [2, 4, 5, 6, 6, 7, 8]
> 5 -> 8 : [2, 4, 5, 5, 6, 7, 8]
> 4 -> 8 : [2, 4, 4, 5, 6, 7, 8]
       : [2, 3, 4, 5, 6, 7, 8]
```

Insertion Sort Another Ilustration



Temp

3

temp

2

temp

10

Start 1 : Start formA[1]
Temp = A[1] = 3

4 is greater than temp (3) then shift 4 to the right
because it's in column 0, insert temp to replace 4

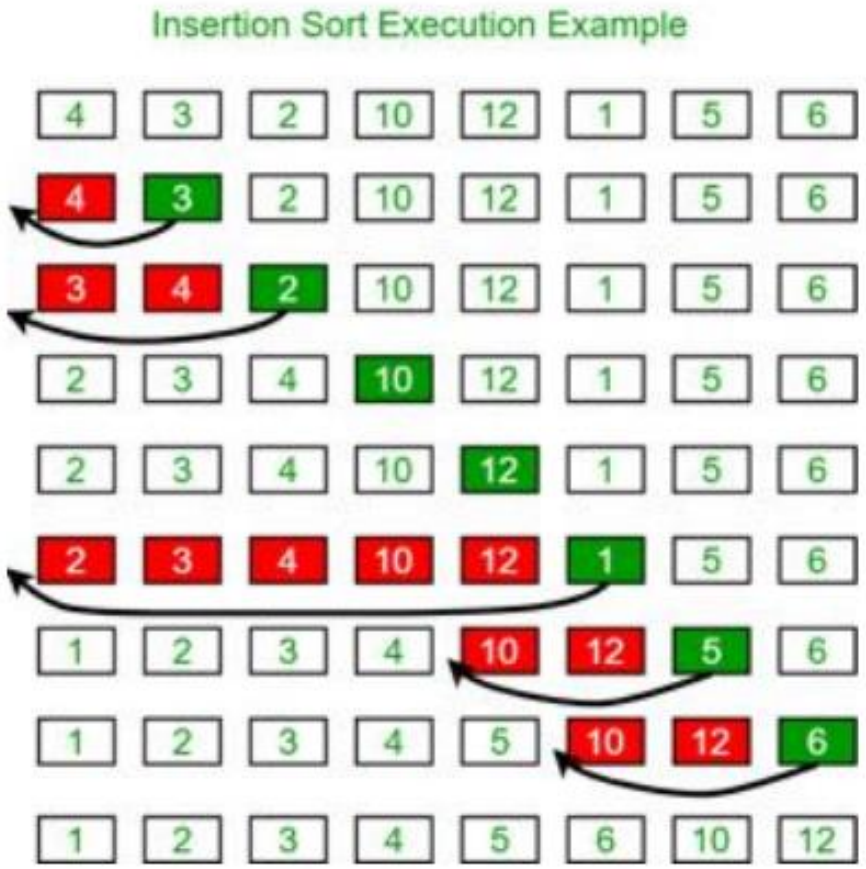
Step 2 : Start fromA[2]
Temp = A[2] = 2

4 is greater than temp (2) then shift 4 to the right
3 is greater than temp (2) then move 3 to the right
because it's already in column 0, insert temp to replace 3

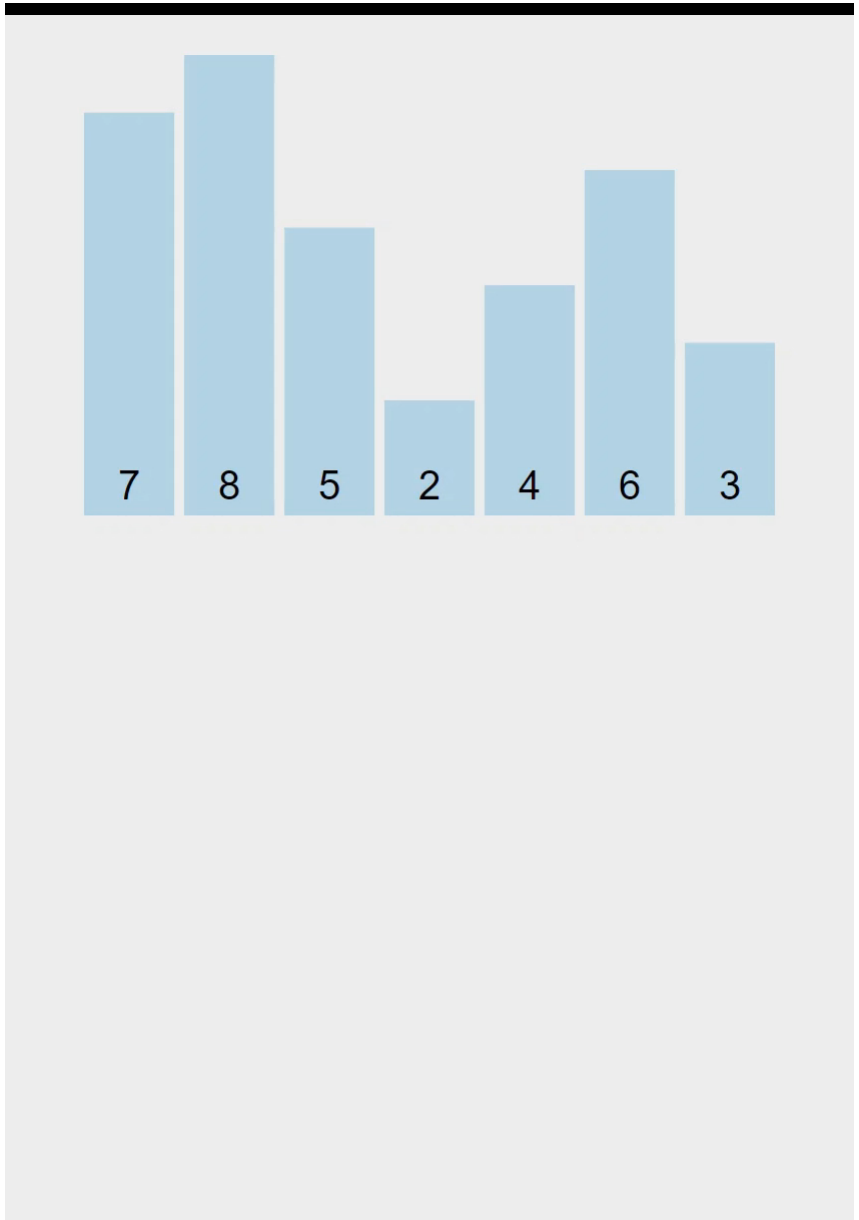
Step 3 : Start fromA[3]
Temp = A[3] = 10
4 is not greater than 10 then the process is complete. Then insert temp replaces 10 (for algorithm consistency)

Illustration of Insertion Sort (Ascending)

The results of each stage can be described as follows



Visualisasi InsertionSort



Algoritma Insertion Sort

```
Insertion Sort(arr, size)
for i←0 to size-1
        temp ← arr[i];
        j ← i;
        while (j>0 and arr[j-1]>temp)
                arr[j] ← arr[j-1]
                j--
        arr[j] ← temp

return (arr)
```

Insertion Sort

```
27      static void insertionSort(int[] data){
28          for(int i=0;i<data.length;i++){
29              int temp = data[i];
30              int j = i;
31              while(j>0 && data[j-1]>temp){
32                  data[j] = data[j-1];
33                  j--;
34              }
35              data[j] = temp;
36          }
37      }
```

Assignments

1. Illustrate the process of resolving data sorting cases using Bubble Sort for data = {20,35,14,7,67,89,23,46}
2. Illustrate the process of resolving data sorting cases using Selection Sort for data ={39,14,67,29,65,25,88,17}!
3. Illustrate the process of resolving data sorting cases using Insertion Sort for data = {11,13,0,91,11,23,111,19}!
4. Explain the actions performed on the Bubble Sort and Selection Sort algorithm if you find data elements with the same value! Example = {22,33,45,17,33}