SCJ2013 Data Structure & Algorithms

Insertion Sort

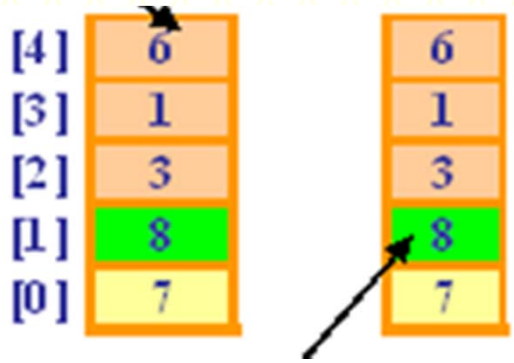
Nor Bahiah Hj Ahmad & Dayang
Norhayati A. Jawawi

Insertion Sort

- Strategy
  - Take multiple passes over the array
  - Partition the array into two regions: sorted and unsorted
    - Take each item from the unsorted region and insert it into its correct order in the sorted region
    - Find the next unsorted element and insert it in correct place, relative to the ones already sorted.
  - Appropriate for small arrays due to its simplicity

Insertion Sort Implementation [7 8 3 1 6]

```
void insertionSort(dataType data[])
 {  dataType item;
    int pass, insertIndex;
    for(pass=1;
        pass<n;pass++)
   {
       item = data[pass];
       insertIndex = pass;
     while((insertIndex >0) &&
            (data[insertIndex -1]>item))
     {
       //insert the right item
        data[insertIndex]=
                          data[insertIndex -1];
        insertIndex --;
     }
       data[insertIndex] = item;
     //insert item at the right place
   }
}
```



insertIndex = 1 Item = 8

Pass 1

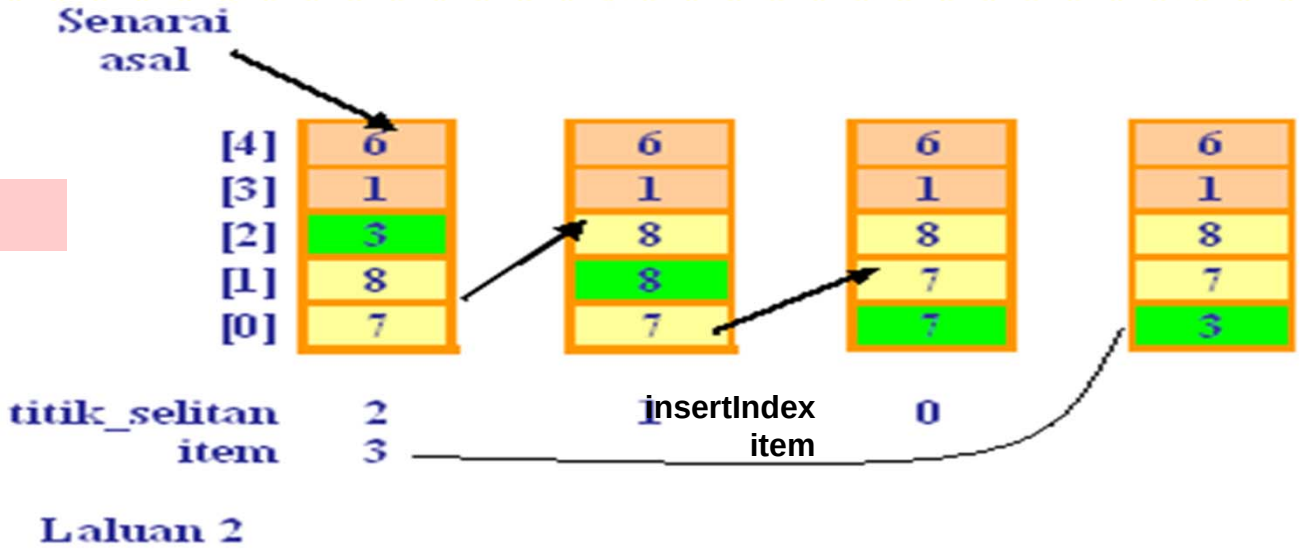
item=8 > data[0]=7. while loop condition is false, therefore data[1] will be assigned with

item = 8.

No of comparison = 1

Insertion Sort Implementation [7 8 3 1 6]

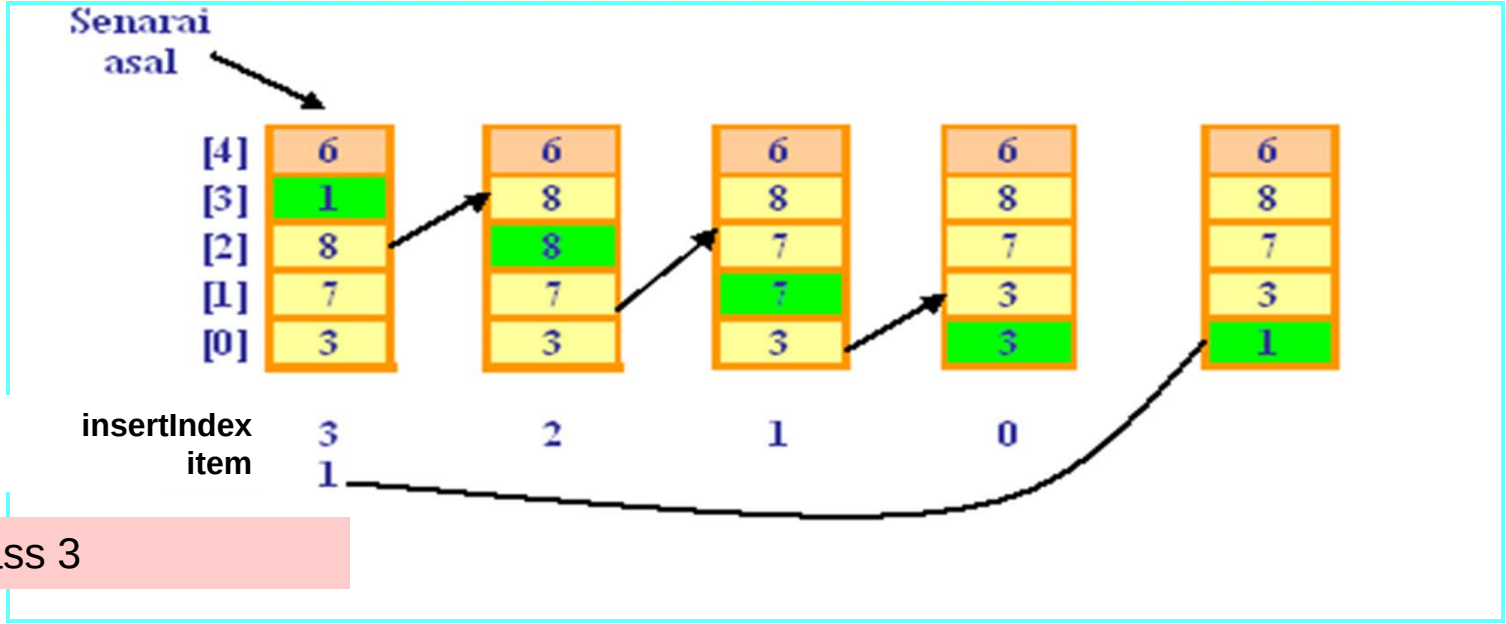
Pass 2



Item to be insert is 3. Insertion point is from indeks 0-2, which is between 7 and 8.

Number of comparison = 2

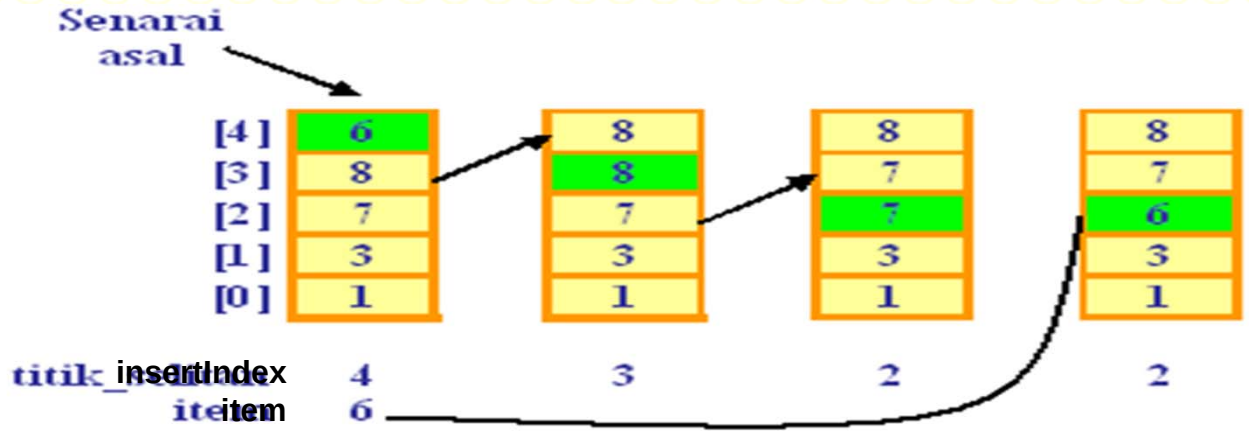
Insertion Sort Implementation [7 8 3 1 6]



Item to be insert is 1. Insertion point is from indeks 0-3, which is between 3, 7 and 8.

Number of comparison = 3

Insertion Sort Implementation [7 8 3 1 6]



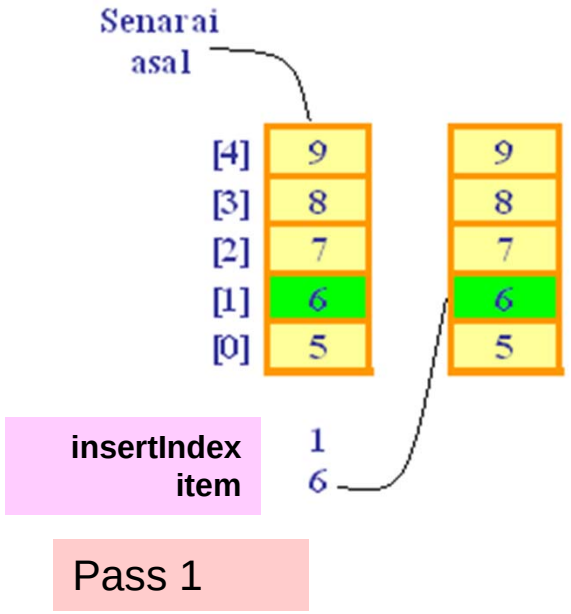
Pass 4

Item to be insert is 6. Insertion point is from indeks 0-4, which is between 1,3, 7 and 8. at index, item (6) > data[1]=3, while loop condition is false and therefore data[2] is assigned with value for item = 6.

Number of comparison = 3

Insertion Sort for Best Case [5 6 7 8 9]

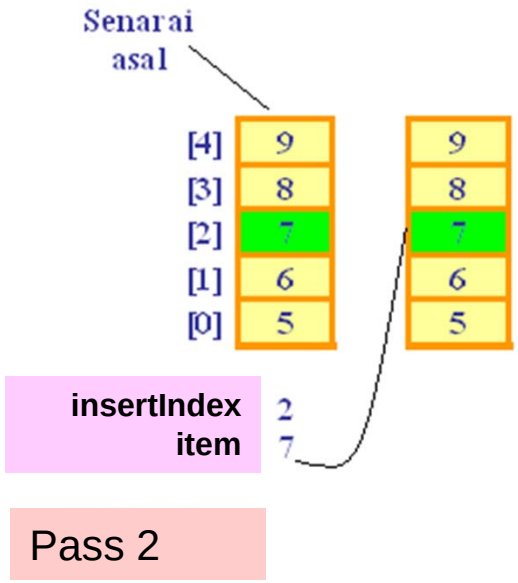
Best case for Insertion Sort can be achieved when data is almost sorted or totally sorted. Each pass will have 1 comparison only.



item=6 > data[0]=1. while condition is false and data[1] is assigned with item=6.

Number of Comparison= 1

Insertion Sort for Best Case [5 6 7 8 9]

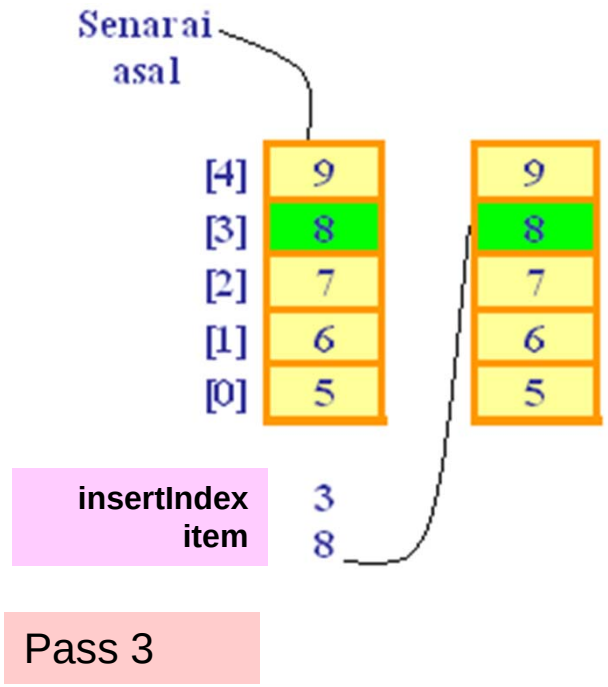


Item=7 > data[1]=1.

while condition become false and data[2] is assigned with item=7.

Number of Comparison is 1

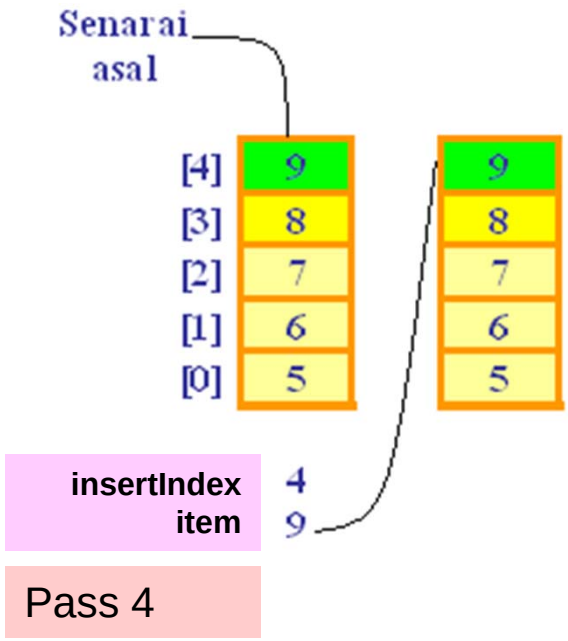
Insertion Sort for Best Case [5 6 7 8 9]



Item=8 > data[2]=7. while condition become false and data[3] is assigned with item=8.

Number of Comparison is 1

Insertion Sort for Best Case [5 6 7 8 9]



Item=9 > data[3]=8. while condition become false and data[4] is assigned with item=9.

Number of Comparison is 1

Insertion Sort Analysis – Best Case

There are 4 passes to sort array with elements [5 6 7 8 9].

In each pass there is only 1 comparison.

Example,

Pass 1, 1 comparison

Pass 2, 1 comparison

Pass 3, 1 comparison

Pass 4, 1 comparison

In this example, the total comparisons for an array with size 5 is 4. Therefore, for best case, the number of comparison is $n-1$ which gives linear time complexity - linear O(n).

Insertion Sort Analysis – Worse Case

Worse case for insertion sort is when we have totally unsorted data. In each pass, the number of iteration for while loop is maximum.

Pass 4, 4 comparison - (n-1)
Pass 3, 3 comparison -(n-2)
Pass 2, 2 comparison -(n-3)
Pass 1, 1 comparison - (n-4)

The number of comparisons between elements in Insertion Sort can be stated as follows:

$$\sum_{i=1}^{n-1} i = (n-1)+(n-2)+\ldots\ldots+2+1 = \frac{n(n-1)}{2} = O(n^2)$$

Insertion Sort Analysis

The number of comparisons is as follows:

$$\sum_{i=1}^{n-1} i = (n-1)+(n-2)+\ldots\ldots+2+1 = \frac{n(n-1)}{2} = O(n^2)$$

InsertionSort – Algorithm Complexity

Insertion	Comparisons:	Swaps
Best Case	O(*n*)	0
Average Case	O(n^2)	O(n^2)
Worst Case	O(n^2)	O(n^2)

- Number of comparisons
  - worst case : 1+2+…+(n-1), O(n^2)
  - best case : (n-1)* 1 , O(n)
- Number of swaps
  - worst case : 1+2+...+(n-1), O(n^2)
  - best case : 0 , O(1)

Summary and Conclusion

	Insertion	Bubble	Selection
Comparisons:			
Best Case	O(n)	O(n^2)	O(n^2)
Average Case	O(n^2)	O(n^2)	O(n^2)
Worst Case	O(n^2)	O(n^2)	O(n^2)
Swaps			
Best Case	0	0	O(n)
Average Case	O(n^2)	O(n^2)	O(n)
Worst Case	O(n^2)	O(n^2)	O(n)

Both Bubble sort and Selection sort performance do not depend on the initial arrangement of data, however, insertion sort performance is better for the best case.

References

1. Nor Bahiah et al. *Struktur data & algoritma menggunakan C++. Penerbit UTM, 2005*
2. Richrd F. Gilberg and Behrouz A. Forouzan, "*Data Structures A Pseudocode Approach With C++*", Brooks/Cole Thomson Learning, 2001.