# Lecture

**NOTE:** FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

## Sorting

Sorting is nothing but arranging the data in ascending or descending order. The term ***sorting*** came into picture, as humans realized the importance of searching quickly.

### Selection Sort

Selection sort is conceptually the simplest sorting algorithm of them all. The algorithm works by selecting the smallest unsorted item and then swapping it with the item in the position to be filled.

The selection sort works as you look through the entire array for the smallest element, once you find it you swap it (the smallest element) with the first element of the array. Then you look for the smallest element in the remaining array (an array without the first element) and swap it with the second element. Then you look for the smallest element in the remaining array (an array without first and second elements) and swap it with the third element, and so on. The worst-case runtime complexity is ***O(n²).***

**Example:** In each pass, ***red*** is where the value will go and ***green*** is the found smallest value.

***29***, 64, 73, 34, ***20***,
20, ***64***, 73, 34, ***29***,
20, 29, ***73***, ***34***, 64
20, 29, 34, ***73***, ***64***
20, 29, 34, 64, 73

### Code for Selection Sort:

Below is the code for a method of Selection Sort.

```
public void selectionSort(int[] arr)
{
    int n = arr.length;
    for (int i = 0; i < n - 1; i++)
    {
        int min = i;

        for (int j = i; j < n; j++)
        {
            if (arr[j] < arr[min])
            {
                min = j;
            }
```

```
            }
            int temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
}
```

## Insertion Sort

Consider I have 10 cards out of a deck. You have the rest of the deck in your hand in sorted form. If I give you a card from my hand and tell you to insert the card in its right position, so that the cards in your hand are still remain sorted. What will you do?

Well, you will have to go through each card from the start or the back and find the right position for the new card, comparing it with each card. Once you find the right position, you will insert the card there. Similarly, if more new cards are provided to you, you can easily repeat the same process and insert the new cards and keep the cards sorted too.

This is exactly how insertion sort works. To sort unordered list of elements, we remove its entries one at a time and then insert each of them into a sorted part (left of the array).

**Example:** We color a sorted part in ***green***, and an unsorted part in ***red***. Here is an insertion sort step by step. We take an element from unsorted part and compare it with elements in sorted part, moving from right to left.

29, 20, 73, 34, 64
**29**, 20, 73, 34, 64
**20**, **29**, 73, 34, 64
**20**, **29**, **73**, 34, 64
**20**, **29**, **34**, **73**, 64
**20**, **29**, **34**, **64**, **73**

Let us compute the worst-time complexity of the insertion sort. In sorting the most expensive part is a comparison of two elements. Surely that is a dominant factor in the running time. We will calculate the number of comparisons of an array of N elements:

we need 0 comparisons to insert the first element
we need 1 comparison to insert the second element
we need 2 comparisons to insert the third element
and son on…
we need (N-1) comparisons (at most) to insert the last element
Totally,

$$1 + 2 + 3 + \ldots + (N-1) = O(n^2)$$

The worst-case runtime complexity is $O(n^2)$. What is the best-case runtime complexity? I comparison each time for $n$ elements so $O(n)$. The advantage of insertion sort comparing it

to the previous two sorting algorithm is that insertion sort runs in linear time on nearly sorted data.

### Code for Insertion Sort:

Below is the code for a method of Insertion Sort.

```
public void insertionSort(int[] arr)
{
    for (int i = 1; i < arr.length; i++)
    {
        int value = arr[i];
        int j = i;
        while (j > 0 && arr[j-1] > value)
        {
            arr[j] = arr[j-1];
            j--;
        }
        arr[j] = value;
    }
}
```