Introduction to Sorting

Sorting is arranging the data in ascending or descending order.
There are so many things in our real life that we need to search for, like a particular record in database, roll numbers in merit list, a particular telephone number in telephone directory, a particular page in a book etc. All this would have been a mess if the data was kept unordered and unsorted, but fortunately the concept of **sorting** came into existence, making it easier for everyone to arrange data in an order, hence making it easier to search.
Sorting arranges data in a sequence which makes searching easier.

Sorting Efficiency

The two main criterias to judge which algorithm is better than the other have been:

1. Time taken to sort the given data.
2. Memory Space required to do so.

Different Sorting Algorithms

There are many different techniques available for sorting, differentiated by their efficiency and space requirements.

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

Bubble Sort Algorithm

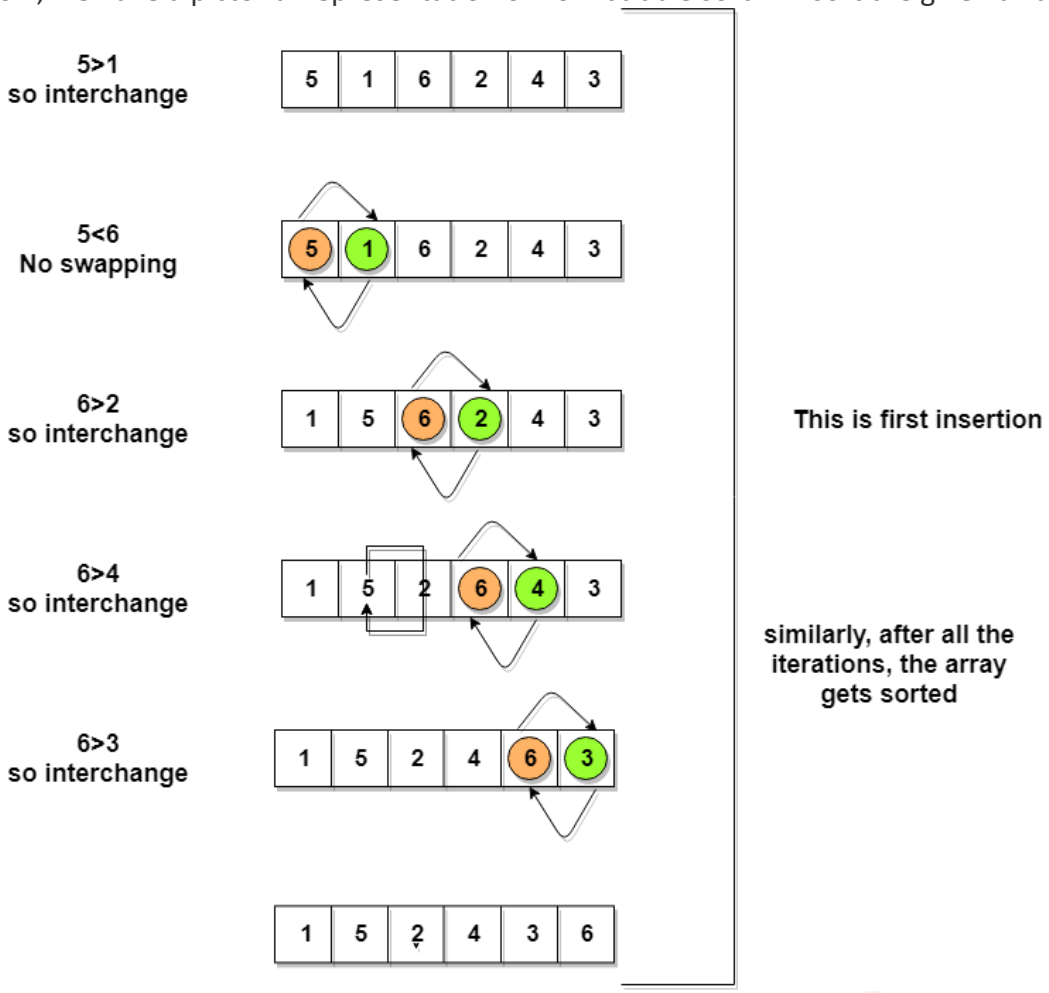
Bubble Sort is a simple algorithm which is used to sort a given set of `n` elements provided in form of an array with `n` number of elements. Bubble Sort compares all the element one by one and sort them based on their values.
If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements, and then move on to compare the second and the third element, and so on.
If we have total `n` elements, then we need to repeat this process for `n-1` times.
It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.
Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

Implementing Bubble Sort Algorithm

Following are the steps involved in bubble sort (for sorting a given array in ascending order):

1. Starting with the first element (index = 0), compare the current element with the next element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. **Repeat Step 1**.

Let's consider an array with values {5, 1, 6, 2, 4, 3}
Below, we have a pictorial representation of how bubble sort will sort the given array.



So as we can see in the representation above, after the first iteration, 6 is placed at the last index, which is the correct position for it.
Similarly after the second iteration, 5 will be at the second last index, and so on.

```c
/*Bubble Sort - C program to sort an Array
in Ascending and Descending Order.*/

#include <stdio.h>

#define MAX 100

int main()
{
    int arr[MAX],limit;
    int i,j,temp;

    printf("Enter total number of elements: ");
    scanf("%d",&limit);

    /*Read array*/
    printf("Enter array elements: \n");
    for(i=0; i<limit; i++)
    {
        printf("Enter element %3d: ",i+1);
```

```
        scanf("%d",&arr[i]);
    }

    /*sort elements in Ascending Order*/
    for(i=0; i<(limit-1); i++)
    {
        for(j=0; j<(limit-i-1); j++)
        {
            if(arr[j]>arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }

    printf("Array elements in Ascending Order:\n");
    for(i=0; i<limit; i++)
        printf("%d ",arr[i]);

    printf("\n");

    /*sort elements in Descending Order*/
    for(i=0; i<(limit-1); i++)
    {
        for(j=0; j<(limit-i-1); j++)
        {
            if(arr[j]<arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }

    printf("Array elements in Descending Order:\n");
    for(i=0; i<limit; i++)
        printf("%d ",arr[i]);

    printf("\n");

    return 0;
}
```

Output

```
    Enter total number of elements: 10
    Enter array elements:
    Enter element 1: 12
    Enter element 2: 34
    Enter element 3: 43
    Enter element 4: 32
    Enter element 5: 21
    Enter element 6: 1
```

```
Enter element 7: 11
Enter element 8: 2
Enter element 9: 3
Enter element10: 100
Array elements in Ascending Order:
1 2 3 11 12 21 32 34 43 100
Array elements in Descending Order:
100 43 34 32 21 12 11 3 2 1
```

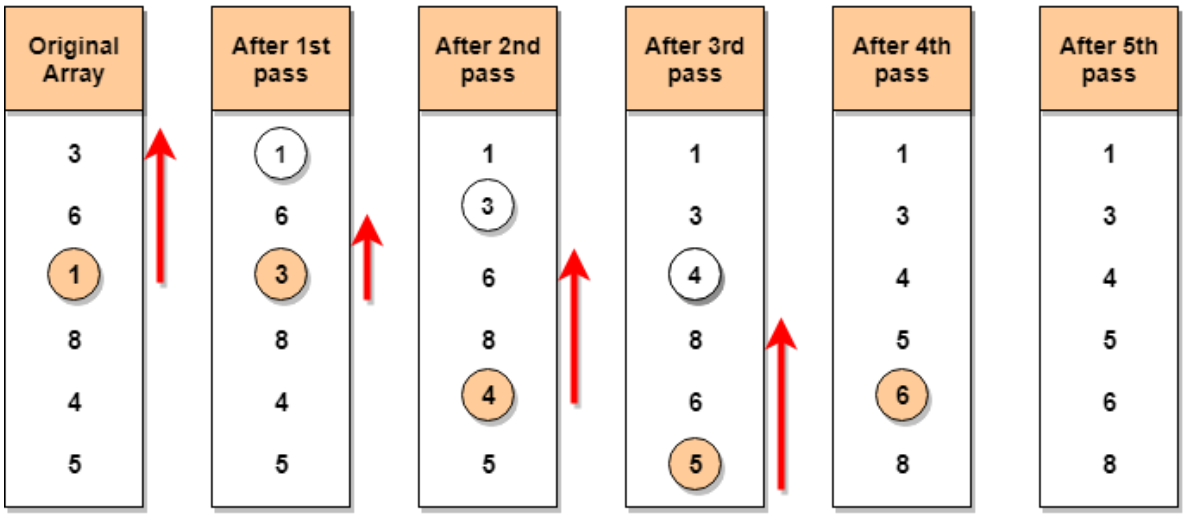
Selection Sort Algorithm

Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted.
It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.

Following are the steps involved in selection sort (Ascending order)

1. Starting from the first element, we search the smallest element in the array, and replace it with the element in the first position.
2. We then move on to the second position, and look for smallest element present in the subarray, starting from index 1, till the last index.
3. We replace the element at the **second** position in the original array, or we can say at the first position in the subarray, with the second smallest element.
4. This is repeated, until the array is completely sorted.

Let's consider an array with values {3, 6, 1, 8, 4, 5}



In the **first** pass, the smallest element will be 1, so it will be placed at the first position. Then leaving the first element, **next smallest** element will be searched, from the remaining elements. We will get 3 as the smallest, so it will be then placed at the second position. Then leaving 1 and 3(because they are at the correct position), we will search for the next smallest element from the rest of the elements and put it at third position and keep doing this until array is sorted.

Algorithm for Selection Sort:

Step 1 − Set min to the first location

Step 2 − Search the minimum element in the array

Step 3 – swap the first location with the minimum value in the array

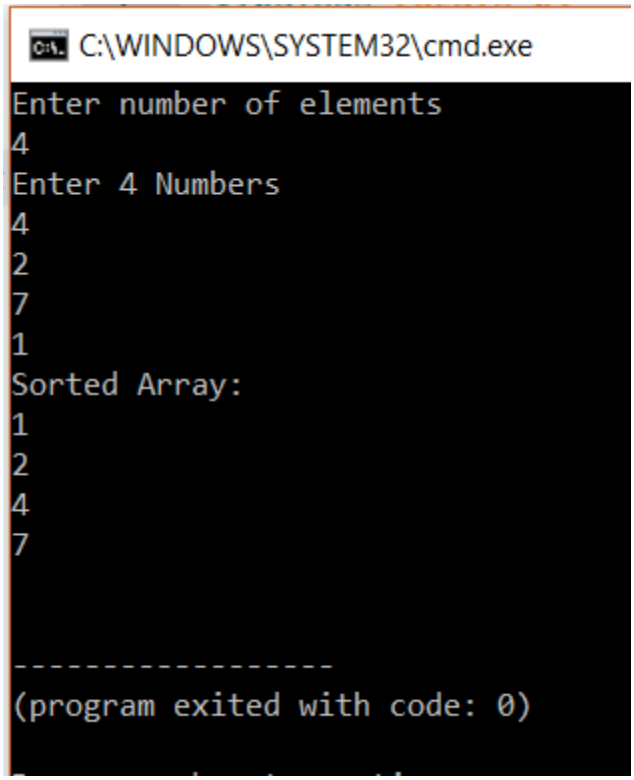
Step 4 – assign the second element as min.

Step 5 − Repeat the process until we get a sorted array.

Code for Selection Sort:

```
#include <stdio.h>
int main()
{
int a[100], n, i, j, position, swap;
printf("Enter number of elementsn");
scanf("%d", &n);
printf("Enter %d Numbersn", n);
for (i = 0; i < n; i++)
  scanf("%d", &a[i]);
for(i = 0; i < n - 1; i++)
{
 position=i;
 for(j = i + 1; j < n; j++)
 {
  if(a[position] > a[j])
    position=j;
 }
 if(position != i)
 {
   swap=a[i];
   a[i]=a[position];
   a[position=swap;
 }
}
printf("Sorted Array:n");
for(i = 0; i < n; i++)
  printf("%dn", a[i]);
return 0;
}
```

Output:


C:\WINDOWS\SYSTEM32\cmd.exe
Enter number of elements
4
Enter 4 Numbers
4
2
7
1
Sorted Array:
1
2
4
7

(program exited with code: 0)