**ASSIGNMENT NO. :- 11**

**TITLE** : - SORTING ALGORITHMS

**PROBLEM STATEMENT** :-

Write C++ program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using a) Selection Sort

b) Bubble sort and display top five scores.

**PRE - REQUISITES** :-

Concept of arrays , passing arrays to functions . Basics of sorting techniques .

**LEARNING OBJECTIVES** :-

To implement and learn program for bubble sort and selection sort for percentage of students in ascending order .

**THEORY** :-

**Sorting** :

Sorting is nothing but storage of data in sorted order , it can be in ascending order or descending order .

It is the process of ordering a list of elements in either ascending or descending order .

**TYPES OF SORTING TECHNIQUES** :-

There are many types of sorting techniques differentiated by their efficiency and space requirements.

Following are some sorting techniques

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

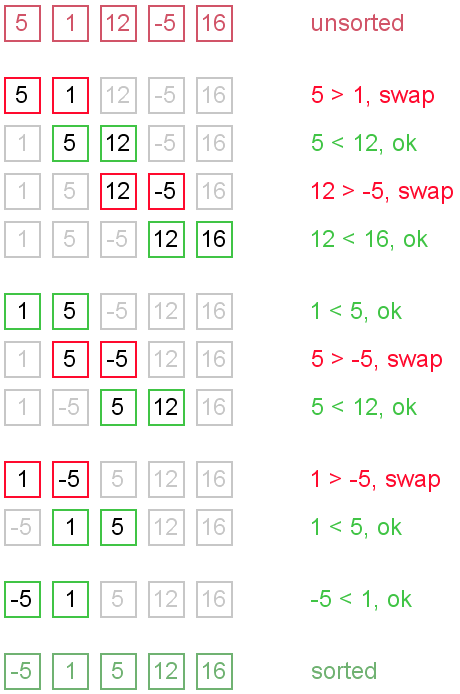
**BUBBLE SORT** :-

Bubble sort is one of the simplest and the most popular sorting method. The basic idea behind bubble sort is as a bubble rises up in water , the smallest element goes to the beginning . This method is based on successive selecting through exchange of adjacent element .

Instead of searching the array as a whole , the bubble sort works by comparing adjacent pairs of objects in the array . If the objects are not in correct order they are swapped so that the largest of two moves up . This process continues until the largest element of objects eventually bubbles up to the highest position in the array .

For example :

Sort {5, 1, 12, -5, 16} using bubble sort.



**Advantages of Bubble sort** :-

1. The primary advantage is it is popular and easy to implement .
2. In Bubble sort elements are swapped directly so space requirement is minimum.
3. It is memory efficient as it requires only O(1) space , as the only extra memory it require is that of temporary value created when swapping adjacent elements.
4. It is stable because the order of identical items does not change while the array is being sorted.

**Applications of Bubble sort** :-

It is very simple algorithm that can be used to introduce basic techniques for proving an algorithm correct and analyzing its run time.

**Time complexity for bubble sort** :-

The complexity of sorting algorithm depends upon the no. of comparisons that are made .

Total no. of comparisons in buble sort is : n(n-1)/ 2

=n2-n

Time complexity = O(n2)

**Space complexity for bubble sort** :-

Space complexity =O(1)

**Algorithm for bubble sort** :-

Step 1: Repeat Steps 2 and 3 for i=1 to n

Step 2: Set j=1

Step 3: Repeat while j<=n

         (A) if  a[i] < a[j]

             Then interchange a[i] and a[j]

             [End of if]

         (B) Set j = j+1

        [End of Inner Loop]

    [End of Step 1 Outer Loop]

Step 4: Exit

**SELECTION SORT** :-

In selection sort list is divided into two sublists sorted and unsorted . These two list are divided by imaginary wall. We find a smallest element from unsorted sublist and swap it to the beginning , and wall move one element ahead as the sorted list increases and unsorted list decreases.

For example

9 2 5 7 4 8 on pass 1 look for smallest in 1st to 6th

swap 2nd with first giving

2 9 5 7 4 8 on pass 2 look for smallest in 2nd to 6th

swap 5th with second giving

2 4 5 7 9 8 on pass 3 look for smallest in 3rd to 6th

swap 3rd with third giving

2 4 5 7 9 8 on pass 4 look for smallest in 4th to 6th

Swap 4th with fourth giving

2 4 5 7 9 8 on pass 5 look for smallest in 5th to 6th

swap 5th with 6th giving

2 4 5 7 8 9 sorted.

Selection sort is very simple sorting method . In the Ith pass , we select the element with lowest value among a[i],a[i+1],…,a[n-1] and we swap it with a[i] . As a result after Ith passes (pass number 0 to i-1) first I elements will be in sorted order .

**Time complexity for selection sort** :-

Selection sort is not data sensitive . In Ith pass , n-1 comparisons will be needed to select the smallest element.

Thus , the number of comparisons needed to sort an array having m elements

=(n-1) + (n-2)+…+2+1 = n(n-1)/2

=O(n2)

**Space complexity for selection sort** :-

Space complexity =O(1)

**Algorithm for selection sort** :-

**Step 1** − Set MIN to location 0

**Step 2** − Search the minimum element in the list

**Step 3** − Swap with value at location MIN

**Step 4** − Increment MIN to point to next element

**Step 5** − Repeat until list is sorted.

OR

**Step 1** –

for (i = 0; i < n-1; i++)

{

// find smallest entry in ith to n-1 th place

// p is subscript of smallest entry yet found

p = i;

for (j = i+1; j < n; j++)

if (a[j]<a[p])

p = j;

// exchange pth element with ith element

t = a[p];

a[p] = a[i];

a[i] = t;

}

* **Flow Chart:**
* **Test Cases:**