**Q. COMPARE BUBBLE SORT AND SELECTION SORT?**

Ans. **BUBBLE SORT**

Bubble sort sends the most significant element at the end of the unsorted array.

Below is the code for the implementation of bubble sort in c++:-

**void bubbleSort(vector<int> vec, int n){**

**int i;**

**int j;**

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

**// Last i elements are already placed at their place.**

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

**if (arr[j] > arr[j + 1]){**

**swap(arr[j], arr[j + 1]);**

**}**

**}**

**}**

**}**

1. ***Comparison made based on a number of comparasions:-***

In each pass, the bubble sort frequently compares and switches adjacent components (if necessary). The last (i-1) elements in the i-th pass of the bubble sort are already sorted, and the i-th most significant element is put in the (N-i)-th position, or the i-th last position, as indicated in the code.

Comparisons; — n-1

An inner loop and an outer loop are used in bubble sort.

Deterministic O(n) comparisons are carried out in the inner loop.

1. ***Comparison made based on a number of swaps:-***

**Best Case:** Sorted input array. Or almost all of the elements are where they should be. [ O(N) ]. swaps in O(1).

**Worst case scenario:** Reversed sorting/Very few elements are in the correct position [ O(N2) ]. swaps in O(N2).

**Average Case:** The typical case is [O(N2)]. swaps in O(N2).

It is a straightforward sorting method that is also reliable.

1. An in-place sorting algorithm changes the items of the starting array to sort the supplied array.

**SELECTION SORT**

Below is the code for the implementation of selection sort in c++:-

**void selectionSort(int arr[], int n)**

**{**

**int i, j, min\_idx;**

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

**// To find the minimum element in an unsorted array.**

**min\_idx = i;**

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

**if (arr[j] < arr[min\_idx])**

**min\_idx = j;**

**// Swapping the minimum element with the first element.**

**if(min\_idx!=i)**

**swap(&arr[min\_idx], &arr[i]);**

**}**

**}**

1. ***Comparison made based on a number of comparasions:-***

The i-th smallest element is chosen and placed in position i by selection sort. The array is split into a sorted (left) and an unsorted (right) subarray via this procedure. In an unsorted subarray, the smallest element is chosen and placed in the first position (ascending order).

It chooses the next smallest element repeatedly.

Comparisons: n- i (as shown in the code 2nd for loop).

1. ***Comparison made based on a number of swaps:-***

**Best Case:** [O(N2)]. include O(1) swaps.

**Worst Case:** When the inner loop makes a maximum comparison, the data is reverse-sorted. [ O(N2) ] . Moreover, O(N) swaps.

**Average case:** A typical case is [O(N2)]. Moreover, O(N) swaps.

It can also be applied to list structures like linked lists that efficiently add and remove operations. Eliminate the tiniest component of the unsorted section until you reach the end of the sorted part.

The number of swaps decreased. Swaps are always O(N).

1. ***It can be in place as well as out place algorithm.***

* *In-Place sort.*

When elements are moved rather than swapped (for example, setting temp=a[min], moving items from ar[i] to ar[min-1], and then putting a[i]=temp).

* *Out-Place sort.*

If swapping is not chosen, the algorithm is not in place.

**SUBMITTED BY:-**

**LAKSHAY BATRA**

**ELECTRICAL**

**21104031**