

## LAB NO. 4

# Sorting Algorithms: Bubble Sort and Selection Sort

---

### OBJECTIVE:

- To understand & implement the working of sorting algorithms using arrays in C++.

### SORTING:

It is often necessary to arrange the elements in an array in numerical order from highest to lowest values i.e.; from ascending to descending and vice versa. If an array contains string values or alphabetical order then arrays are need to be sorted. The process of sorting an array requires the exchanging of values. While this seems to be a simple process, a computer must be careful that no values are lost during this exchange.

### BUBBLE SORT:

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of  $O(n^2)$  where  $n$  is the number of items.

### How Bubble Sort Works?

We take an unsorted array for our example. Bubble sort takes  $O(n^2)$  time so we're keeping it short and precise.



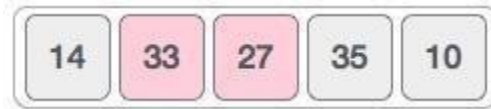
Bubble sort starts with very first two elements, comparing them to check which one is greater.



In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.



We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this –



Next we compare 33 and 35. We find that both are in already sorted positions.



Then we move to the next two values, 35 and 10.



We know then that 10 is smaller 35. Hence they are not sorted.



We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –



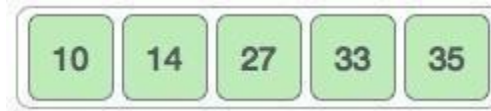
To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this –



Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sort learns that an array is completely sorted.



Now we should look into some practical aspects of bubble sort.

### ALGORITHM

We assume **list** is an array of **n** elements. We further assume that **swap** function swaps the values of the given array elements.

```
begin BubbleSort(list)
  for all elements of list
    if list[i] > list[i+1]
      swap(list[i], list[i+1])
    end if
  end for
  return list
end BubbleSort
```

Example: C++ Program to sort 10 number using bubble sort technique.

```
#include<iostream>
#define MAX 10
using
namespace std;

int list[MAX] = {1,8,4,6,0,3,5,2,7,9};
void display() {
  int i;
  cout<<"[";
  // navigate through all items
  for(i=0; i < MAX; i++)
  {   cout<<list[i]<<" ";
```

```

        }
        cout<<"]\n";
    }
    void bubbleSort() {
        int temp;  int i,j;  bool swapped = false;

        // loop through all numbers
        for(i = 0; i < MAX-1; i++)
        {
            swapped = false;
            // loop through numbers falling ahead
            for(j = 0; j < MAX-1-i; j++)
            {
                cout<<"Items compared: ["<<list[j]<<","<<list[j+1]<<"]";
                // check if next number is lesser than current no      // swap the numbers.
                // (Bubble up the highest number)
                if(list[j] > list[j+1]) {
                    temp = list[j];
                    list[j] = list[j+1];
                    list[j+1] = temp;

                    swapped = true;
                    cout<<"=>swapped["<<list[j]<<","<<list[j+1]<<"]\n";
                }

            }
            else {
                cout<<"=> not swapped\n";
            }

        }

        // if no number was swapped that means array is sorted now, break the loop.
        if(!swapped) {
            break;
        }
        cout<<"Iteration "<<i+1<<" #";
        display();
    }

}

int main() {
    cout<<"Input Array: ";
    display();
    cout<<"\n";
}

```

```

    bubbleSort();
    cout<<"\nOutput Array: ";
    display();
    return 0;
}

```

## SELECTION SORT:

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparisonbased algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexities are of  $O(n^2)$ , where **n** is the number of items.

### How Selection Sort Works?

Consider the following depicted array as an example.



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.



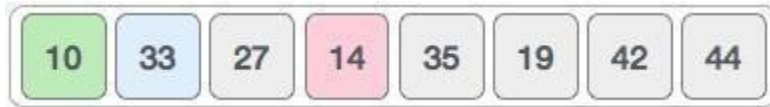
So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.



For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.



We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

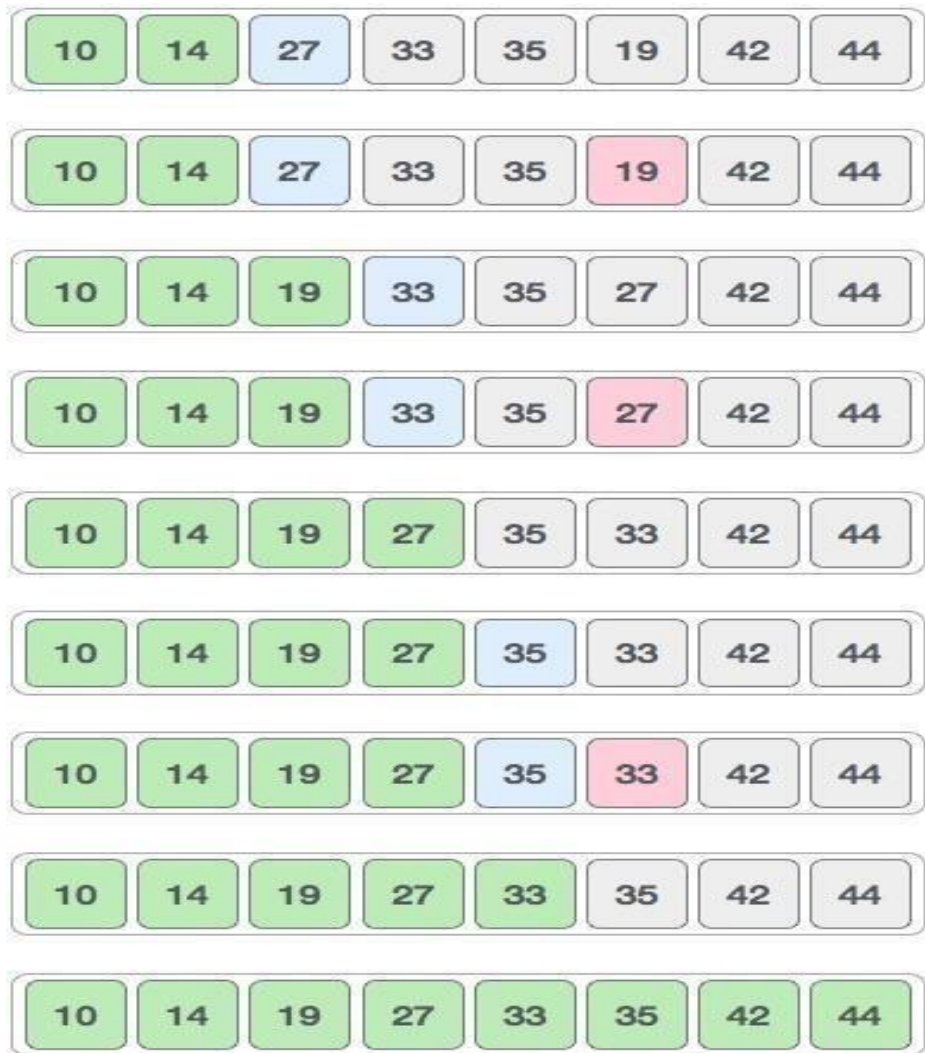


After two iterations, two least values are positioned at the beginning in a sorted manner.



The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process –



Now, let us learn some programming aspects of selection sort.

## Algorithm

```
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
```

### Example: Program to sort 10 numbers using selection sort technique

```
#include <iostream>
using namespace std;
#define MAX 10

int intArray[MAX] = {4,6,3,2,1,9,7};

void printline(int count)
{
    int i;
    for(i = 0; i < count-1; i++) {
        cout<<"=";
    }
    cout<<"=\n";
}

void display() {
    int i;
    cout<<"[ ";

    // navigate through all items
    for(i = 0; i < MAX; i++)
    {
        cout<< intArray[i]<<" ";
    }
    cout<<"]\n";
}

void selectionSort() {
    int indexMin, i, j;

    // loop through all numbers
    for(i = 0; i < MAX-1; i++) {
        // set current element as minimum
        indexMin = i;
```

```

        // check the element to be minimum
for(j = i+1; j < MAX; j++) {
    if(intArray[j] < intArray[indexMin])
    {
        indexMin = j;
    }
}
if(indexMin != i)
{
    cout<<"Items swapped: ["<<intArray[i]<<","<<intArray[indexMin]<<" ]\n";

        // swap the numbers
        int temp = intArray[indexMin];
intArray[indexMin] = intArray[i];
intArray[i] = temp;
    }

    cout<<"Iteration "<<i+1<<" #:";
display();
}
}
int main()
{
    cout<<"Input Array: ";
display();
prntline(50);
selectionSort();
cout<<"Output Array: ";
display();
prntline(50);
return 0; }

```



## Lab Tasks:

**Task 1:** Write a code to sort the following arrays using selection sort method.

[10, 34, 2, 56, 7, 67, 88, 42]

**Task 1:** Write a code to sort the following arrays using bubble sort method.

[10, 34, 2, 56, 7, 67, 88, 42]