LAB on SORTING ALGORITHMS

SELECTION SORT

AIM:

To sort the given data using selection sort.

ALGORITHM:

Step 1: Initialize the integer variables.
Step 2: Get the total number of values to be sorted from the user (Size of the array)
Step 3: Get the numbers to be sorted.
Let: a = array containing the values
Let: n = # of elements
Step 4: Initialize i to current
Step 5: If current is less than j ,then initialize j to current.
Step 6: Assign j to temp, current to i and temp to current for swapping
Step 7: print the sorted data

Convert the below Pseudocode to C++ code

```
Selection Sort(A, n)
For i = 1 to n-1 do:
        iMin = i
        For J = i + 1 to n-1 do:
          If A(J) < A(iMin)
            iMin = J
          End-If
        End-For
        Temp = A(i)
        A(i) = A(iMin)
        A(iMin) = Temp
      End-For
```

Run It
Input
Enter the number of elements: 5
Enter the Numbers to be sorted: 33 21 75 52 45

OUTPUT
Sorted Data: 21 33 45 52 75

RESULT:
The given data is sorted and printed using selection sort.

BUBBLE SORT

Bubble Sort is a simple-minded algorithm based on the idea that we look at the list, and wherever we find two consecutive elements out of order, we swap them. Repeatedly traverse the unsorted part of the array, comparing consecutive elements, and we interchange them when they are out of order.

AIM:
To sort the given data using Bubble sort.

ALGORITHM:
Step 1: Initialize the variables i,j,n, and temp.
Step 2: Get the number of elements.
Step 3: Get the numbers to be sorted.
Step 4; Assign a[i] to temp
Step 5: Assign a[i+1] to a[i]
Step 6: Assign Temp to a[i+1]
Step 7: The data is sorted.

Convert the below Pseudocode to C++ code

```
Bubble Sort(A, n)

For i = 1 to n - 1
       For j = 1 to n - 1
         If (A(j) > A(j + 1)
           Temp = A(j)
           A(j) = A(j + 1)
           A(j + 1) = Temp
         End-If
       End-For
     End-For
```

INPUT
Enter the number of elements: 6
Enter the numbers to be sorted: 22 66 11 99 33 77

OUTPUT
Sorted Data: 11 22 33 66 77 99
RESULT:
The given data is sorted and printed using Bubble sort.

INSERTION SORT

Insertion Sort is an algorithm to traverse a given array and insert each element into the sorted part of the list where it belongs. This usually involves pushing down the larger elements in the sorted part.

AIM:
To sort the given data using Insertion sort.

Convert the below Pseudocode to C++ code

```
Insertion Sort(A, n)
     For i = 2 to n
       hole = i
       Do while (hole > 1) and (A(hole) < A(hole - 1)
         Temp = A(hole)
         A(hole) = A(hole - 1)
         A(hole - 1) = Temp
         hole = hole - 1
       End-Do
     End-For
```

ALGORITHM:
Step 1: Initialize the variables i,hole,n and temp.
Step 2: Get the number of elements.
Step 3: Get the numbers to be sorted.
Step 4: If the second element is less than the first element then assign it to temp.
Step 5: Assign the first element to second element.
Step 6: Assign Temp to first element.
Step 7: The swapping is done
Step 8: The data is sorted.

INPUT
Enter the number of entries: 5
Enter the list of no to be sorted: 25 15 35 05 45
OUTPUT
The sorted list: 05 15 25 35 45
RESULT:
The given data is sorted and printed using Insertion Sort.

MERGE SORT

AIM:

To sort the given data using Merge sort.

1) Break the list in two

2) Assume we can sort the smaller lists (recursion)

3) Merge the results together

Convert the below Pseudocode to C++ code

```
Merge Sort(A[])
n = length (A)
{
If (n<2)
Return;
else
Merge_Sort(A[1…n/2])
Merge_Sort(A[n/2+1 …n ])
Merge(A[1…n])
```

QUICK SORT

AIM:

To sort the given data using Quick Sort.

Convert the below Pseudocode to C++ code

```
QuickSort( double[] a )
   {
      if ( a.length ≤ 1 )
         return;    // Don't need sorting

      Select a pivot;  // It's usually the last elem in a[]

      Partition a[] in 2 halves:
         left[]: elements ≤ pivot
        right[]: elements > pivot;

      Sort left[];
      Sort right[];

      Concatenate: left[] pivot right[]
   }
```