Experiment No. 2

Aim:

To write a C program to store the first-year percentage of students in an array and sort the array in ascending order using Selection Sort and Bubble Sort. The program should also display the top five scores after sorting.

Objective:

- To understand and implement sorting algorithms (Selection Sort and Bubble Sort) in C.
- To store and sort percentage data using arrays.
- To extract and display the top five scores from sorted data.

Problem Statement

Write a program to store the first year percentage of students in an array. Write function for sorting array of floating point numbers in ascending order using - Selection Sort -Bubble sort and display top five scores

Theory:

Selection Sort:

Selection Sort works by selecting the smallest element from the unsorted part of the array and placing it in the correct position. It continues this process until the array is sorted.

- Time Complexity: $O(n^2)$
- Space Complexity: O(1)

Advantages:

- Simple and easy to understand.
- Performs well on small datasets.
- Does not require extra memory (in-place sorting).
- Makes at most (n-1) swaps.

Disadvantages:

- Inefficient for large datasets.
- Not adaptive (does not perform better on partially sorted arrays).
- Performs unnecessary comparisons even if the array is already sorted.

Example:

Selection Sort Example

Input: 45, 78, 23, 56, 89
Passes:
Pass 1: 23, 78, 45, 56, 89
Pass 2: 23, 45, 78, 56, 89
Pass 3: 23, 45, 56, 78, 89
Pass 4: 23, 45, 56, 78, 89

Sorted Output: 23, 45, 56, 78, 89

Top Five Scores: 89, 78, 56, 45, 23

Algorithm for Selection Sort:

1. Start
2. Input number of students n
3. Input n percentage values into array arr[]
4. Repeat for i = 0 to n-2
   a. Set minIndex = i
   b. Repeat for j = i+1 to n-1
      • If arr[j] < arr[minIndex], set minIndex = j
   c. Swap arr[i] and arr[minIndex]
5. Display sorted array
6. Display top 5 scores (last 5 elements in reverse)
7. Stop

Bubble Sort:

Bubble Sort works by repeatedly comparing adjacent elements and swapping them if they are in the wrong order. This process continues until no more swaps are needed.

- Time Complexity: $O(n^2)$
- Space Complexity: O(1)

Advantages:

- Easy to implement and understand.
- Adaptive: stops early if the array becomes sorted before all passes are completed.
- Good for nearly sorted data.

Disadvantages:

- Very inefficient on large datasets.
- Requires many unnecessary comparisons and swaps.
- Not suitable for performance-critical applications.

Bubble Sort Example

Input: 55, 42, 90, 33, 67

Passes:
Pass 1: 42, 55, 33, 67, 90
Pass 2: 42, 33, 55, 67, 90
Pass 3: 33, 42, 55, 67, 90
Pass 4: 33, 42, 55, 67, 90

Sorted Output: 33, 42, 55, 67, 90

Top Five Scores: 90, 67, 55, 42, 33

Algorithm for Bubble Sort:

1. Start
2. Input number of students n
3. Input n percentage values into array arr[]
4. Repeat for i = 0 to n-2
   a. Set swapped = false
   b. Repeat for j = 0 to n-i-2
      • If arr[j] > arr[j+1], swap them
      • Set swapped = true
   c. If swapped == false, break
5. Display sorted array
6. Display top 5 scores (last 5 elements in reverse)
7. Stop

Input:

- **Number of students n**
- **Percentage values (as float)**

Note: Write input you are giving to your program

Output:

Note: Write output of your program

Conclusion:

Hence, we have implemented basic sorting techniques such as Selection Sort and Bubble Sort. This helped us understand how to organize data, compare sorting methods, and extract top scores using arrays in C.