EXPERIMENT NO 2

Title: To implement a program for Selection Sort and find its complexity.

Aim: - Write a program to sort elements of an array in ascending order using a selection sort algorithm and find its complexity.

Lab Outcomes : CSL401.1: Ability to implement the algorithm using divide and conquer approach and also analyze the complexity of various sorting algorithms.

Theory:

The selection sort enhances the bubble sort by making only a single swap for each pass through the rundown. In order to do this, a selection sort searches for the biggest value as it makes a pass and, after finishing the pass, places it in the best possible area. Similarly, as with a bubble sort, after the first pass, the biggest item is in the right place. After the second pass, the following is set up. This procedure proceeds and requires n-1 goes to sort n items since the last item must be set up after the (n-1) th pass.

ALGORITHM: SELECTION SORT (A)

1. k ← length [A]
2. **for** j ←1 to n-1
3. smallest ← j
4. **for** I ← j + 1 to k
5. **if** A [i] < A [smallest]
6. then smallest ← i
7. exchange (A [j], A [smallest])

How Selection Sort works

- In the selection sort, first of all, we set the initial element as a **minimum**.]
- Now we will compare the minimum with the second element. If the second element turns out to be smaller than the minimum, we will swap them, followed by assigning a minimum to the third element.
- Else if the second element is greater than the minimum, which is our first element, then we will do nothing and move on to the third element and then compare it with the minimum.
  We will repeat this process until we reach the last element.
- After the completion of each iteration, we will notice that our minimum has reached the start of the unsorted list.
- For each iteration, we will start the indexing from the first element of the unsorted list. We will repeat the Steps from 1 to 4 until the list gets sorted or all the elements get correctly

Consider the following example of an unsorted array that we will sort with the help of the Selection Sort algorithm.
A[]=(7,4,3,6,5).
A [] =
1st Iteration:

Set minimum = 7

1. Compare a_0 and a_1

As, $a_0 > a_1$, set minimum = 4.

2. Compare a_1 and a_2

As, $a_1 > a_2$, set minimum = 3.

3. Compare a_2 and a_3

As, $a_2 < a_3$, set minimum= 3.

4. Compare a_2 and a_4

As, $a_2 < a_4$, set minimum =3.

Since 3 is the smallest element, so we will swap a_0 and a_2.

2nd Iteration:

Set minimum = 4

1. Compare a_1 and a_2

As, $a_1 < a_2$, set minimum = 4.

2. Compare a_1 and a_3

As, A[1] < A[3], set minimum = 4.

3. Compare a_1 and a_4

Again, $a_1 < a_4$, set minimum = 4.

Since the minimum is already placed in the correct position, so there will be no swapping.

3rd Iteration:

Set minimum = 7

1.Compare a_2 and a_3

As, $a_2 > a_3$, set minimum = 6.

2. Compare a_3 and a_4

As, $a_3 > a_4$, set minimum = 5.

Since 5 is the smallest element among the leftover unsorted elements, so we will swap 7 and 5.

4th Iteration:

Set minimum = 6

1. Compare a_3 and a_4

As $a_3 < a_4$, set minimum = 6.

Since the minimum is already placed in the correct position, so there will be no swapping.

Complexity Analysis of Selection Sort

Input: Given **n** input elements.

Output: Number of steps incurred to sort a list.

Logic: If we are given n elements, then in the first pass, it will do **n-1** comparisons; in the second pass, it will do **n-2**; in the third pass, it will do **n-3** and so on. Thus, the total number of comparisons can be found by;

Output;

(n-1) + (n-2) + (n-3) + (n-4) + ... + 1

$$Sum = \frac{n(n-1)}{2}$$

i.e., $O(n^2)$

Therefore, the selection sort algorithm encompasses a time complexity of **$O(n^2)$** and a space complexity of **O(1)** because it necessitates some extra memory space for temp variable for swapping.

Time Complexities:

Best Case Complexity: The selection sort algorithm has a best-case time complexity of **$O(n^2)$** for the already sorted array.

Average Case Complexity: The average-case time complexity for the selection sort algorithm is **$O(n^2)$**, in which the existing elements are in jumbled ordered, i.e., neither in the ascending order nor in the descending order.

Worst Case Complexity: The worst-case time complexity is also **$O(n^2)$**, which occurs when we sort the descending order of an array into the ascending order.

In the selection sort algorithm, the time complexity is **$O(n^2)$** in all three cases. This is because, in each step, we are required to find **minimum** elements so that it can be placed in the correct position. Once we trace the complete array, we will get our minimum element.

Conclusion: - Thus we have implemented the program for insertion sort algorithm. The program was successfully compiled & executed, and the output was verified. We have also analyzed the algorithm with its worst case and best case complexities.