2.2 SELECTION SORTING

Question:

Describe the Selection Sort algorithm's process of sorting an array. Selection Sort works by dividing the array into a sorted and an unsorted region. Initially, the sorted region is empty, and the unsorted region contains all elements. The algorithm repeatedly selects the smallest element from the unsorted region and swaps it with the leftmost unsorted element, then moves the boundary of the sorted region one element to the right. Explain why Selection Sort is simple to understand and implement but is inefficient for large datasets. Provide examples to illustrate step-by-step how Selection Sort rearranges the elements into ascending order, ensuring clarity in your explanation of the algorithm's mechanics and effectiveness.

AIM

To describe the Selection Sort algorithm's process of sorting an array.

ALGORITHM

1.For each index i from 0 to n-2:

2. Assume the element at i is the smallest.

3.Compare it with the rest of the unsorted region.

4.Find the index of the smallest element.

5.Swap the smallest element with arr[i].

PROGRAM

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
        print(f"Step {i+1}: {arr}")
    return arr

def run_selection_sort():
    arr = list(map(int, input("Enter array elements: ").split()))
    print("Sorted array:", selection_sort(arr))

run_selection_sort()
```

Input:

[5, 2, 9, 1, 5, 6]

Output:

```
Enter array elements: 5 2 9 1 5 6
Step 1: [1, 2, 9, 5, 5, 6]
Step 2: [1, 2, 9, 5, 5, 6]
Step 3: [1, 2, 5, 9, 5, 6]
Step 4: [1, 2, 5, 5, 9, 6]
Step 5: [1, 2, 5, 5, 6, 9]
Step 6: [1, 2, 5, 5, 6, 9]
Sorted array: [1, 2, 5, 5, 6, 9]
>>>
```

RESULT:

Thus the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity:
  - Best Case (Already Sorted): Still scans entire array → $O(n^2)$ comparisons, O(n) swaps.
  - Worst Case (Reverse Sorted): $O(n^2)$ comparisons, O(n) swaps.
  - Average Case: $O(n^2)$ comparisons.
- Space Complexity:
  - O(1) (in-place sorting).