GREEEDY: SELECTION SORT

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        # Assume the current element is the minimum
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        # Swap the found minimum with the first unsorted element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
        print(f"Step {i+1}: {arr}")


# Example usage
arr = [64, 25, 12, 22, 11]
print("Original array:", arr)
selection_sort(arr)
print("Sorted array:", arr)
```

Output (step-by-step greedy choices):

less Copy Edit

```
Original array: [64, 25, 12, 22, 11]
Step 1: [11, 25, 12, 22, 64]
Step 2: [11, 12, 25, 22, 64]
Step 3: [11, 12, 22, 25, 64]
Step 4: [11, 12, 22, 25, 64]
Step 5: [11, 12, 22, 25, 64]
Sorted array: [11, 12, 22, 25, 64]
```

Each step shows the **greedy selection of the smallest element** in the remaining unsorted list.

Viva Questions for This Practical

1. **What is Selection Sort, and how does it work?**
    - Selection Sort is a comparison-based sorting algorithm where the minimum element from the unsorted part of the array is selected and swapped with the first unsorted element.
2. **What is the time complexity of Selection Sort?**
    - The time complexity of Selection Sort is **O(n^2)** because we have two nested loops: one to select the element and the other to find the minimum.
3. **What is the space complexity of Selection Sort?**
    - The space complexity of Selection Sort is **O(1)** because it is an in-place sorting algorithm, meaning it doesn't require extra space except for the input array.
4. **How does the greedy approach apply in Selection Sort?**
    - In each step, Selection Sort greedily picks the smallest element from the unsorted part and places it in its correct position. This is a greedy approach because it makes the optimal choice at each step.
5. **Is Selection Sort stable?**
    - No, Selection Sort is **not stable**. Stability in sorting algorithms means that equal elements retain their relative order. In Selection Sort, equal elements might not retain their original order after sorting.
6. **What are the advantages and disadvantages of Selection Sort?**
    - **Advantages:** Simple to implement, works well with small datasets.
    - **Disadvantages:** Inefficient for large datasets due to its **O(n^2)** time complexity.
7. **How does the Selection Sort algorithm compare to other sorting algorithms like Bubble Sort or Insertion Sort?**
    - Selection Sort is less efficient than both Bubble Sort and Insertion Sort in terms of number of swaps. However, it does fewer swaps, which might be beneficial when memory writes are expensive.
8. **What happens if you start with a sorted array?**
    - If the array is already sorted, Selection Sort still performs all the comparisons but makes no swaps. It will still take **O(n^2)** time but won't modify the array.
9. **How does the algorithm behave when all elements are the same?**

- If all elements are the same, Selection Sort will still perform all the comparisons, but no swaps will be made. The algorithm will still run in **O(n^2)** time.

10. **Can Selection Sort be used for sorting large datasets?**
    - No, Selection Sort is inefficient for large datasets because of its quadratic time complexity. More efficient algorithms like Merge Sort or Quick Sort (with **O(n log n)** complexity) should be used for large datasets.

11. **How would you modify this algorithm to sort in descending order?**
    - To sort the array in descending order, instead of selecting the minimum element, you would select the maximum element in each iteration and place it at the beginning of the unsorted portion.

12. **Why do we need the inner loop in Selection Sort?**
    - The inner loop is used to find the smallest (or largest) element in the unsorted part of the array and helps in selecting the correct element to swap.

13. **Can Selection Sort be implemented in a non-iterative way (recursively)?**
    - Yes, Selection Sort can also be implemented recursively. The outer loop's iteration can be replaced by recursive function calls.