LAB TASK 13

Name: Abdul Moiz

Sap I'd: 54482

Question No:1

```
#include <iostream>
using namespace std;

// Function to partition the array around the pivot
int partition_desc(int arr[], int low, int high) {
   int pivot = arr[low]; // Take the first element as the pivot
   int left = low + 1;
   int right = high;

   while (true) {
      // Move the left pointer to the right until an element <= pivot is found
      while (left <= right && arr[left] >= pivot) {
         left++;
      }
      // Move the right pointer to the left until an element > pivot is found
      while (left <= right && arr[right] < pivot) {
```

```
            right--;
        }
        // If pointers cross, break the loop
        if (left > right) {
            break;
        }
        // Swap elements at left and right pointers
        swap(arr[left], arr[right]);
    }
    // Place the pivot in its correct position
    swap(arr[low], arr[right]);
    return right; // Return the pivot's final position
}

// Quick Sort function
void quick_sort_desc(int arr[], int low, int high) {
    if (low < high) {
        // Partition the array and get the pivot index
        int pivot_index = partition_desc(arr, low, high);
        // Recursively sort the left and right subarrays
        quick_sort_desc(arr, low, pivot_index - 1);  // Left subarray
        quick_sort_desc(arr, pivot_index + 1, high); // Right subarray
    }
}

// Driver code
int main() {
    int arr[] = {50, 23, 9, 18, 61, 32, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << "Original Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    quick_sort_desc(arr, 0, n - 1);

    cout << "Sorted Array (Descending): ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Dry Run:

Initial Array:

[50, 23, 9, 18, 61, 32, 90]

Step 1: First Partition (Pivot = 50)

- **Pivot = 50**, **Left = 1**, **Right = 6**:
  - Move **left** to find an element less than pivot: Stop at index 1 (23).
  - Move **right** to find an element greater than or equal to pivot: Stop at index 6 (90).
  - Swap arr[left] and arr[right]: **[50, 90, 9, 18, 61, 32, 23]**.
  - Continue:
    - Stop left at index 2 (9) and right at index 5 (32).
    - Swap arr[left] and arr[right]: **[50, 90, 32, 18, 61, 9, 23]**.
  - Stop left at index 3 and right at index 4.
  - Swap pivot (50) with arr[right] (61): **[61, 90, 50, 18, 32, 9, 23]**.

Pivot position: **Index 2**.

Step 2: Recursively Sort Left and Right Subarrays

1. **Left Subarray**: [90, 61].
    - Pivot = 90; no swaps needed.
2. **Right Subarray**: [18, 32, 9, 23].
    - Pivot = 18; sorted as [32, 23, 18, 9].

Final Sorted Array (Descending Order):

[90, 61, 50, 32, 23, 18, 9]

Question No: 2

```
#include <iostream>
using namespace std;

// Selection Sort in descending order
void selection_sort_desc(int arr[], int n) {
   for (int i = 0; i < n - 1; i++) {
      int max_idx = i; // Assume the first element of the unsorted part is the largest
      for (int j = i + 1; j < n; j++) {
         if (arr[j] > arr[max_idx]) {
            max_idx = j; // Update the index of the maximum element
         }
      }
      // Swap the largest element with the first element of the unsorted part
```

```
        swap(arr[i], arr[max_idx]);

        // Display the array after each iteration
        cout << "Iteration " << i + 1 << ": ";
        for (int k = 0; k < n; k++) {
            cout << arr[k] << " ";
        }
        cout << endl;
    }
}

int main() {
    int arr[] = {34, 8, 64, 51, 32};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    selection_sort_desc(arr, n);

    cout << "Sorted Array (Descending): ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
```

```
    return 0;
}
```

Dry Run of the Code

Initial Array:

[34, 8, 64, 51, 32]

Iteration 1:

- **Subarray:** [34, 8, 64, 51, 32]
- Find the largest element (64 at index 2).
- Swap 64 with the first element (34).
- **Array after iteration 1:** [64, 8, 34, 51, 32].

Iteration 2:

- **Subarray:** [8, 34, 51, 32]
- Find the largest element (51 at index 3).
- Swap 51 with the second element (8).
- **Array after iteration 2:** [64, 51, 34, 8, 32].

Iteration 3:

- **Subarray:** [34, 8, 32]
- Find the largest element (34 at index 2).
- Swap 34 with itself (no change).
- **Array after iteration 3:** [64, 51, 34, 8, 32].

Iteration 4:

- **Subarray:** [8, 32]
- Find the largest element (32 at index 4).
- Swap 32 with the fourth element (8).

- **Array after iteration 4:** [64, 51, 34, 32, 8].

Final Sorted Array:

[64, 51, 34, 32, 8]

Output When Compiled:

```
Original Array: 34 8 64 51 32
Iteration 1: 64 8 34 51 32
Iteration 2: 64 51 34 8 32
Iteration 3: 64 51 34 8 32
Iteration 4: 64 51 34 32 8
Sorted Array (Descending): 64 51 34 32 8
```