RIPHAH INTERNATIONAL UNIVERSITY

LAB#12

NAME: AHMED AZIZ
SAP ID: 55223

TASK 1

```cpp
#include <iostream>
#include <vector>
#include <ctime> // For measuring execution time
using namespace std;

// Function to swap two elements
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Partition function that selects the first element as the pivot and
arranges elements around the pivot
int partition(vector<int> &arr, int low, int high)
{
    int pivot = arr[low]; // Choose the first element as the pivot
    int i = low + 1;      // Start from the element right after the pivot

    for (int j = low + 1; j <= high; j++)
    {
        // If the current element is smaller than the pivot, swap it with
the element at i
        if (arr[j] < pivot)
        {
            swap(arr[i], arr[j]);
            i++;
        }
    }
    // Place the pivot element in its correct position
    swap(arr[low], arr[i - 1]);
    return i - 1; // Return the pivot index
}

// Recursive Quick Sort function
void quickSort(vector<int> &arr, int low, int high)
{
    if (low < high)
    {
        // Find the pivot index where the array is partitioned
        int pivotIndex = partition(arr, low, high);
```

```
        // Recursively sort elements before and after the partition
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

// Utility function to print the array
void printArray(const vector<int> &arr)
{
    for (int num : arr)
    {
        cout << num << " ";
    }
    cout << endl;
}

int main()
{
    // Initialize an array to be sorted
    vector<int> arr = {33, 12, 52, 3, 75, 29, 41, 6, 19};
    int n = arr.size();

    cout << "Original Array: " << endl;
    printArray(arr);

    // Measure the time taken to perform Quick Sort
    clock_t start = clock();
    quickSort(arr, 0, n - 1);
    clock_t end = clock();

    // Display the sorted array
    cout << "Sorted Array: " << endl;
    printArray(arr);

    // Calculate and display the time taken for sorting
    double time_taken = double(end - start) / CLOCKS_PER_SEC;
    cout << "Time taken by Quick Sort: " << time_taken << " sec" << endl;

    return 0;
}
```

TASK2:

```cpp
#include <iostream>
#include <vector>
#include <cstdlib>   // For rand() and srand()
#include <ctime>     // For time()
#include <chrono>    // For measuring execution time

using namespace std;
using namespace std::chrono;

// Function to swap two elements
void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

// Partition function with fixed pivot (last element)
int partitionFixedPivot(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

// Partition function with random pivot
int partitionRandomPivot(int arr[], int low, int high) {
    // Select a random pivot index between low and high
    int randomIndex = low + rand() % (high - low + 1);
```

```
    swap(arr[randomIndex], arr[high]); // Move random pivot to end
    return partitionFixedPivot(arr, low, high);
}

// QuickSort using fixed pivot (last element)
void quickSortFixedPivot(int arr[], int low, int high) {
    if (low < high) {
        int pi = partitionFixedPivot(arr, low, high);
        quickSortFixedPivot(arr, low, pi - 1);
        quickSortFixedPivot(arr, pi + 1, high);
    }
}

// QuickSort using random pivot
void quickSortRandomPivot(int arr[], int low, int high) {
    if (low < high) {
        int pi = partitionRandomPivot(arr, low, high);
        quickSortRandomPivot(arr, low, pi - 1);
        quickSortRandomPivot(arr, pi + 1, high);
    }
}

// Function to copy an array
void copyArray(int src[], int dest[], int size) {
    for (int i = 0; i < size; i++) {
        dest[i] = src[i];
    }
}

int main() {
    // Seed for random number generation
    srand(time(0));

    const int SIZE = 10000; // Size of the array
    int arr[SIZE];
    int arrCopy[SIZE];

    // Fill the array with random numbers
    for (int i = 0; i < SIZE; i++) {
        arr[i] = rand() % 10000;
    }

    // Copy the array for testing both versions on the same data
    copyArray(arr, arrCopy, SIZE);
```

```
    // Measure execution time for QuickSort with fixed pivot
    auto startFixed = high_resolution_clock::now();
    quickSortFixedPivot(arr, 0, SIZE - 1);
    auto endFixed = high_resolution_clock::now();
    auto durationFixed = duration_cast<milliseconds>(endFixed - startFixed);
    cout << "Execution time with fixed pivot: " << durationFixed.count() <<
" ms" << endl;

    // Measure execution time for QuickSort with random pivot
    auto startRandom = high_resolution_clock::now();
    quickSortRandomPivot(arrCopy, 0, SIZE - 1);
    auto endRandom = high_resolution_clock::now();
    auto durationRandom = duration_cast<milliseconds>(endRandom -
startRandom);
    cout << "Execution time with random pivot: " << durationRandom.count()
<< " ms" << endl;

    return 0;
}
```

OUTPUT:

```
PS C:\Users\lenovo\Downloads\LAB_12AHmedAziz55223> cd "c:\Users\lenovo\Downloads\LAB_12AHmedAziz55223\"
p -o task2 } ; if ($?) { .\task2 }
Execution time with fixed pivot: 0 ms
Execution time with random pivot: 0 ms
PS C:\Users\lenovo\Downloads\LAB_12AHmedAziz55223>
```