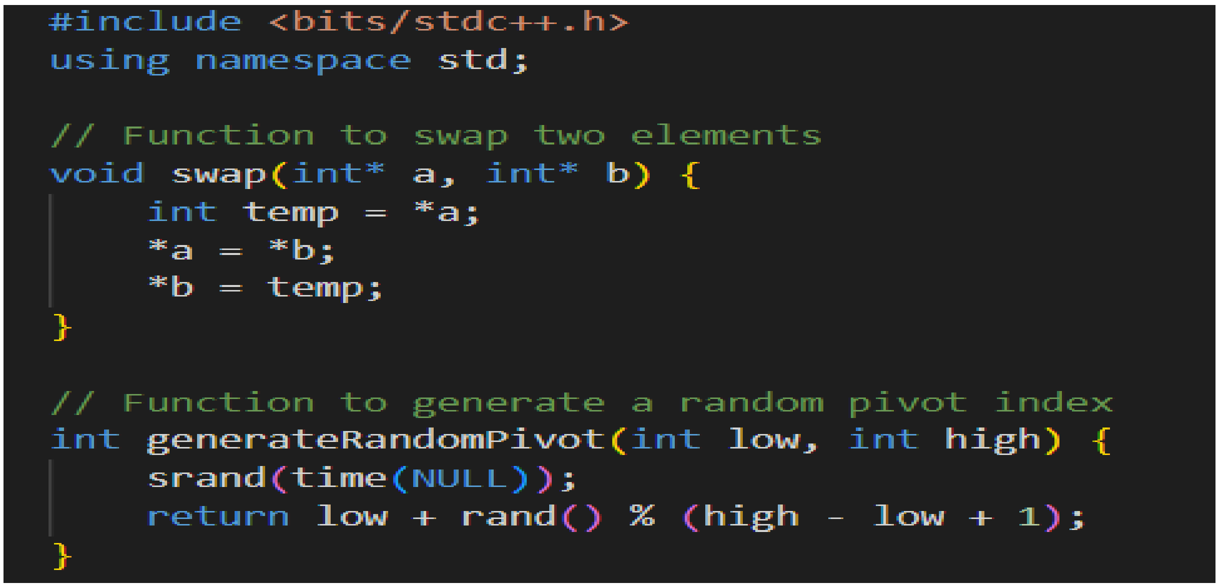
DAA Lab 5

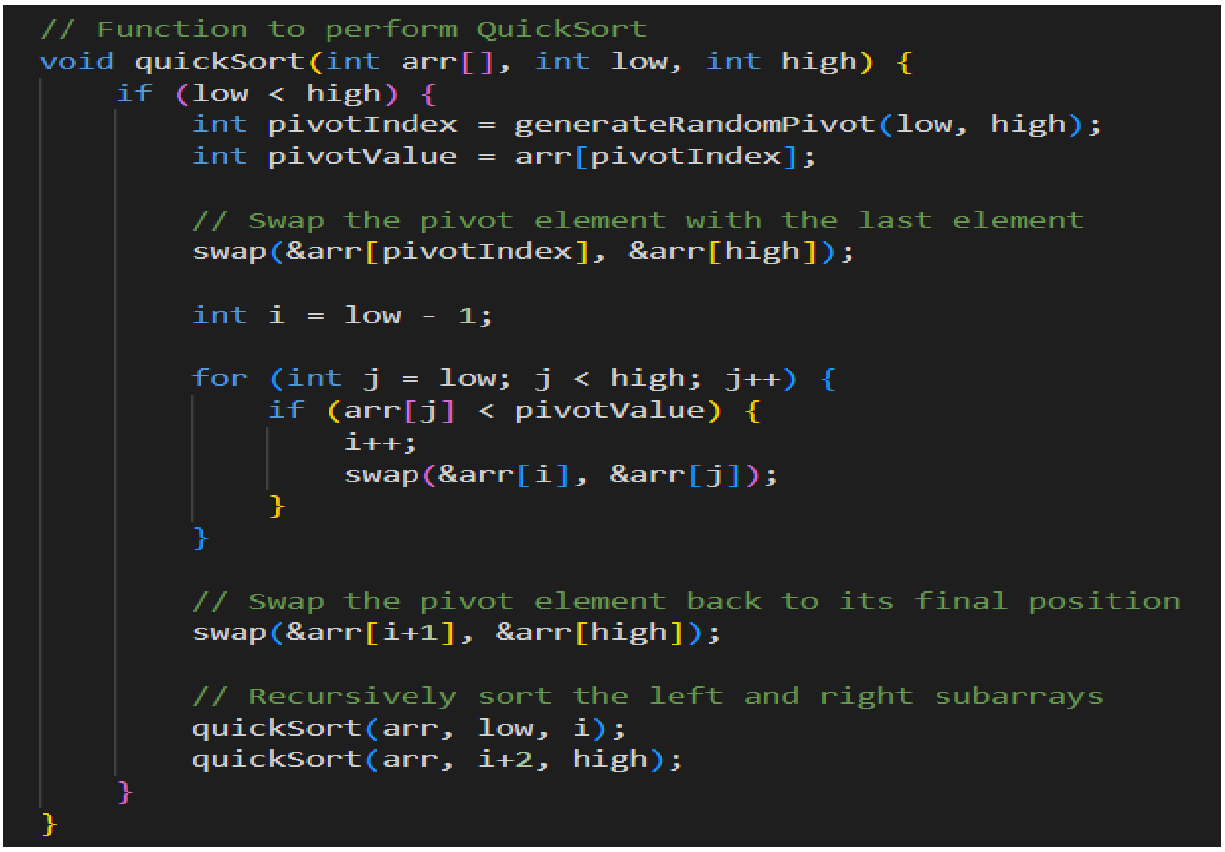
Name: Vatsal Bhuva

Roll No: IIT2022004

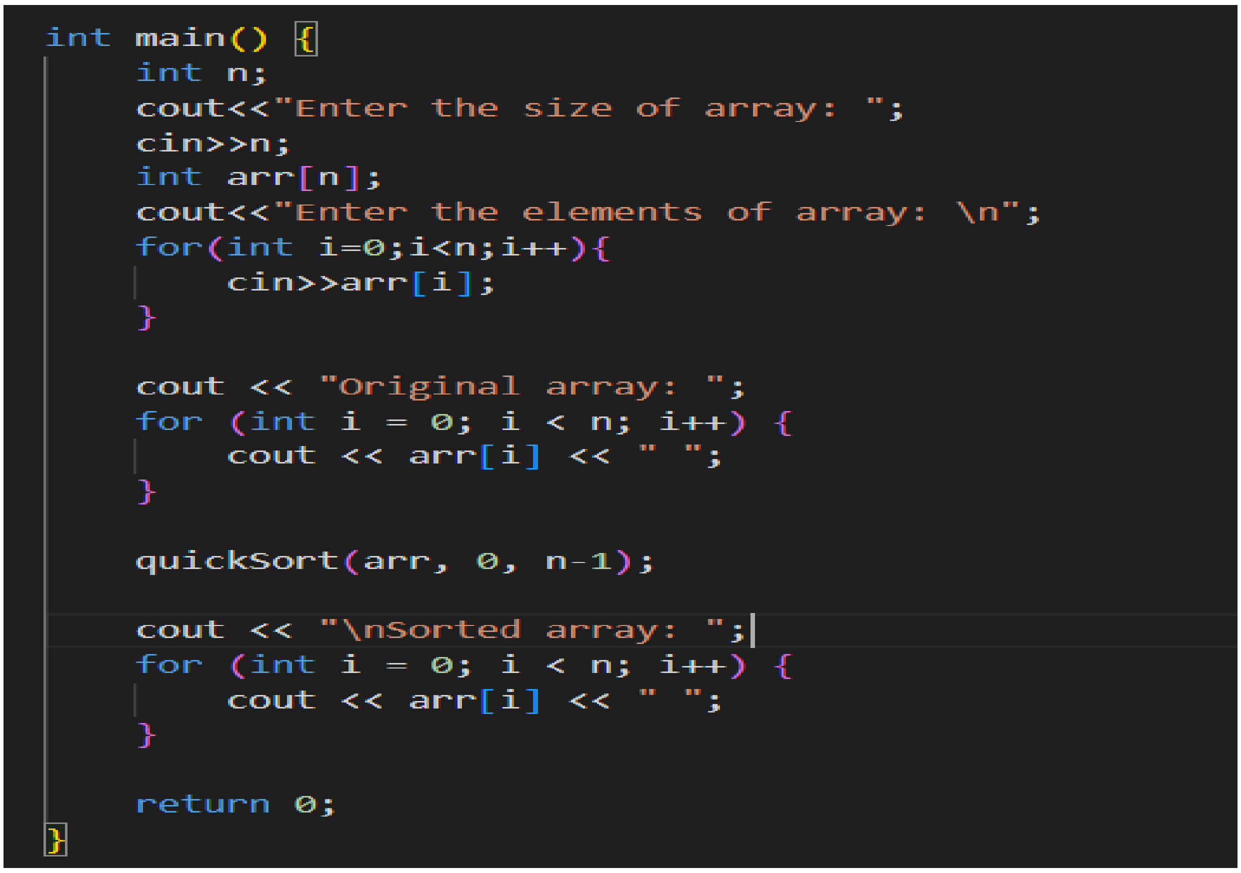
CODE:



```cpp
#include <bits/stdc++.h>
using namespace std;

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

// Function to generate a random pivot index
int generateRandomPivot(int low, int high) {
    srand(time(NULL));
    return low + rand() % (high - low + 1);
}
```



```cpp
// Function to perform QuickSort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivotIndex = generateRandomPivot(low, high);
        int pivotValue = arr[pivotIndex];

        // Swap the pivot element with the last element
        swap(&arr[pivotIndex], &arr[high]);

        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (arr[j] < pivotValue) {
                i++;
                swap(&arr[i], &arr[j]);
            }
        }

        // Swap the pivot element back to its final position
        swap(&arr[i+1], &arr[high]);

        // Recursively sort the left and right subarrays
        quickSort(arr, low, i);
        quickSort(arr, i+2, high);
    }
}
```



```
int main() {
    int n;
    cout<<"Enter the size of array: ";
    cin>>n;
    int arr[n];
    cout<<"Enter the elements of array: \n";
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }

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

    quickSort(arr, 0, n-1);

    cout << "\nSorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Apriori Analysis:

In this analysis, we will try to discuss the time complexity of this random quick sort algorithm for the best case, worst case and the average case.

This Random Quick sort algorithm randomly generates a pivot using the generateRandomPivot() function. It takes use of a mathematical function to generate a random number which is between the starting index and the ending index. It then swaps the pivot element with the element present at the last index and applies standard QuickSort algorithm after that.

Best Case Time Complexity = O(NlogN)

Average Case Time Complexity = O(NlogN)

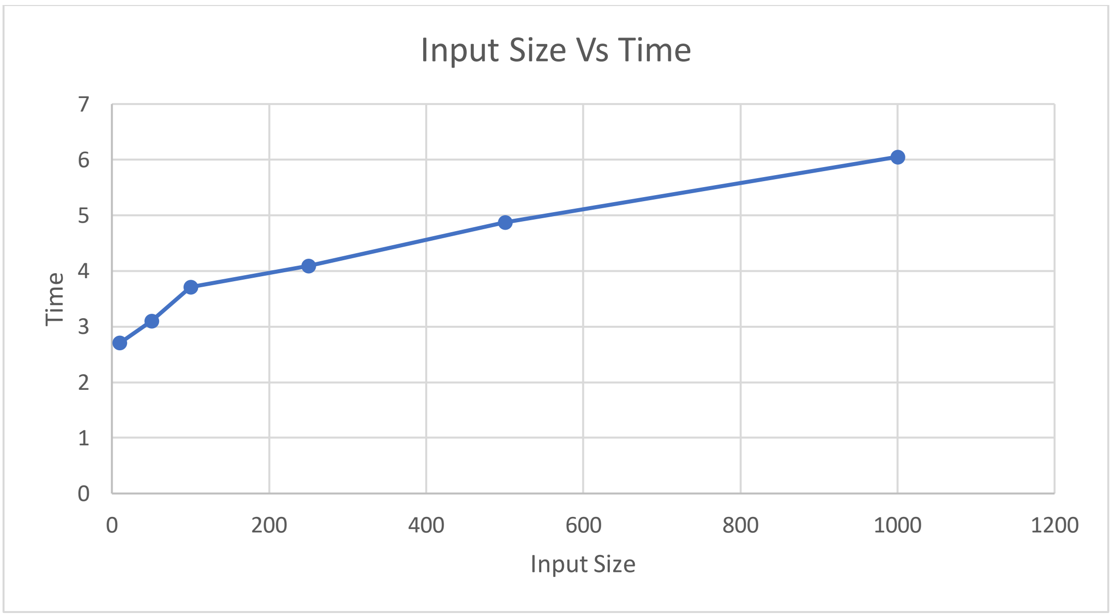
Worst Case Time Complexity = O(n^2)

Aposteriori Analysis:

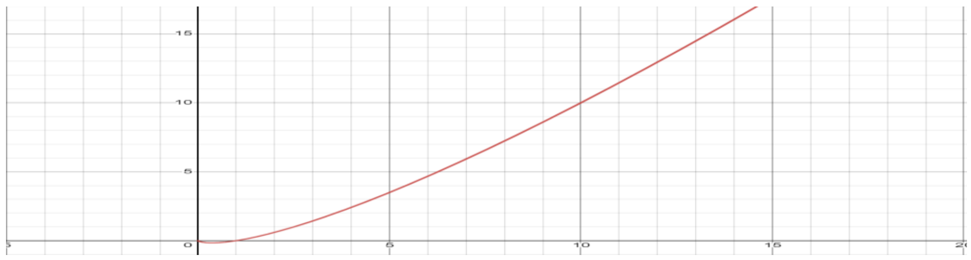
Variation of algorithm with different input sizes:

INPUT SIZE	TIME TAKEN FOR OUTPUT
10	**2.704 seconds**
50	**3.098 seconds**
100	**3.712 seconds**
250	**4.096 seconds**
500	**4.872 seconds**
1000	**6.051 seconds**

Actual Graph:



Expected Time Complexity Graph: (Graph of nlogn)



REPORT:

- Here we can see that the actual graph differs from the expected graph. This might be accounted to the fact that the size of data used for the calculations is very less. As we increase the size of dataset used the graph tends to get more similar to the actual graph expected.

- It has been observed that this algorithm performs in the worst case when the array is completely sorted. The swaps made in this case are maximum and the whole array is traversed 2 times. Hence giving it the worst case complexity of 0(n^2).

- The best case time complexity occurs when the chosen pivot happens to be the median element every time. This provides a way for both the halves to get sorted easily.