Practical-6

Implement program for randomized version of quick sort and compare its performance with normal version of quick sort using steps count on various number of inputs.

Randomized Quicksort:

Code:

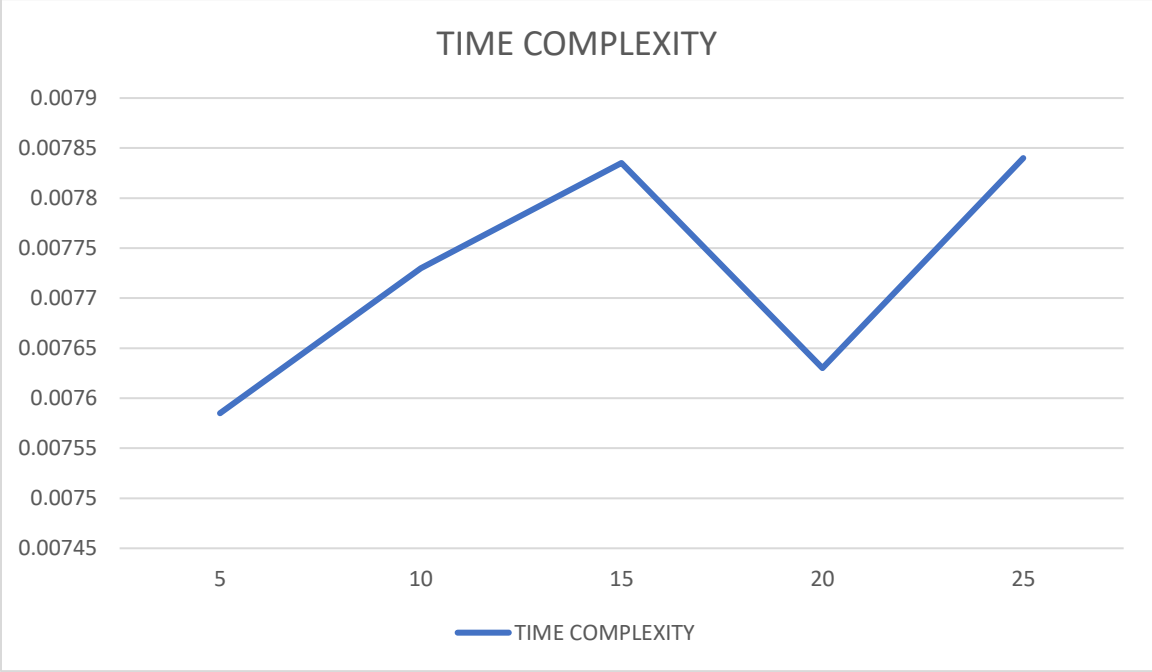
```
#include <cstdlib>
#include <time.h>
#include <iostream>
using namespace std;
int partition(int arr[], int low, int high)
{
int pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high - 1; j++)
{
if (arr[j] <= pivot) {
i++;
swap(arr[i], arr[j]);
}
}
swap(arr[i + 1], arr[high]);
return (i + 1);
}
int partition_r(int arr[], int low, int high)
{
srand(time(NULL));
int random = low + rand() % (high - low);
```

```
swap(arr[random], arr[high]);
return partition(arr, low, high);
}
void quickSort(int arr[], int low, int high)
{
if (low < high) {
int pi = partition_r(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}
void printArray(int arr[], int size)
{
int i;

for (i = 0; i < size; i++)
cout<<arr[i]<<" ";
}
int main()
{
int arr[] = { 10, 7, 8, 9, 1, 5 };
int n = sizeof(arr) / sizeof(arr[0]);
quickSort(arr, 0, n - 1);
printf("Sorted array: \n");
printArray(arr, n);
return 0;
}
```

Output:

Randomized Quicksort:

SIZE OF ARRAY	TIME COMPLEXITY
5	0.007585
10	0.007730
15	0.007835
20	0.007630
25	0.007840



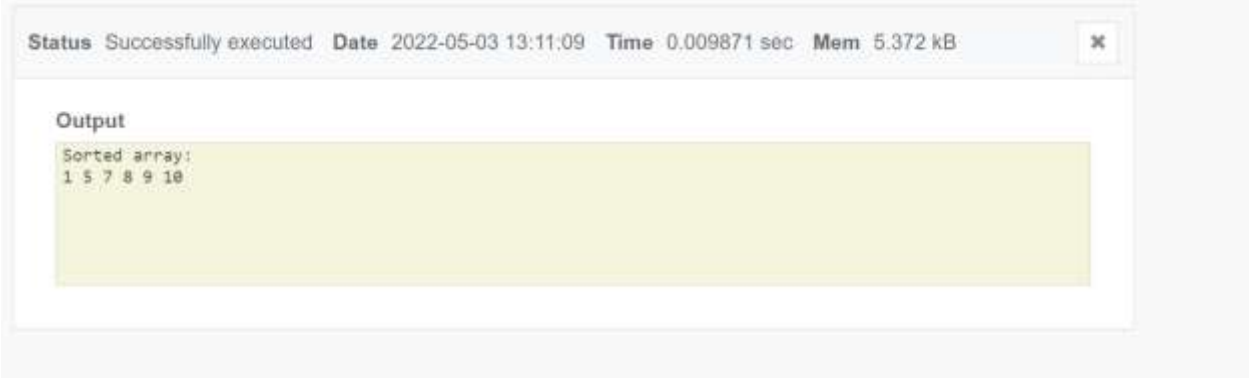
Quick sort:

Code:

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

void swap(int* a, int* b)

{

        int t = *a;
*a = *b;

        *b = t;

}

int partition (int arr[], int low, int high)

{

        int pivot = arr[high];
int i = (low - 1);

      for (int j = low; j <= high - 1; j++)

        {

              if (arr[j] < pivot)

               {
        i++;

                    swap(&arr[i], &arr[j]);

               }

        }

      swap(&arr[i + 1], &arr[high]);

        return (i + 1);

}


void quickSort(int arr[], int low, int high)

{

        if (low < high)
```

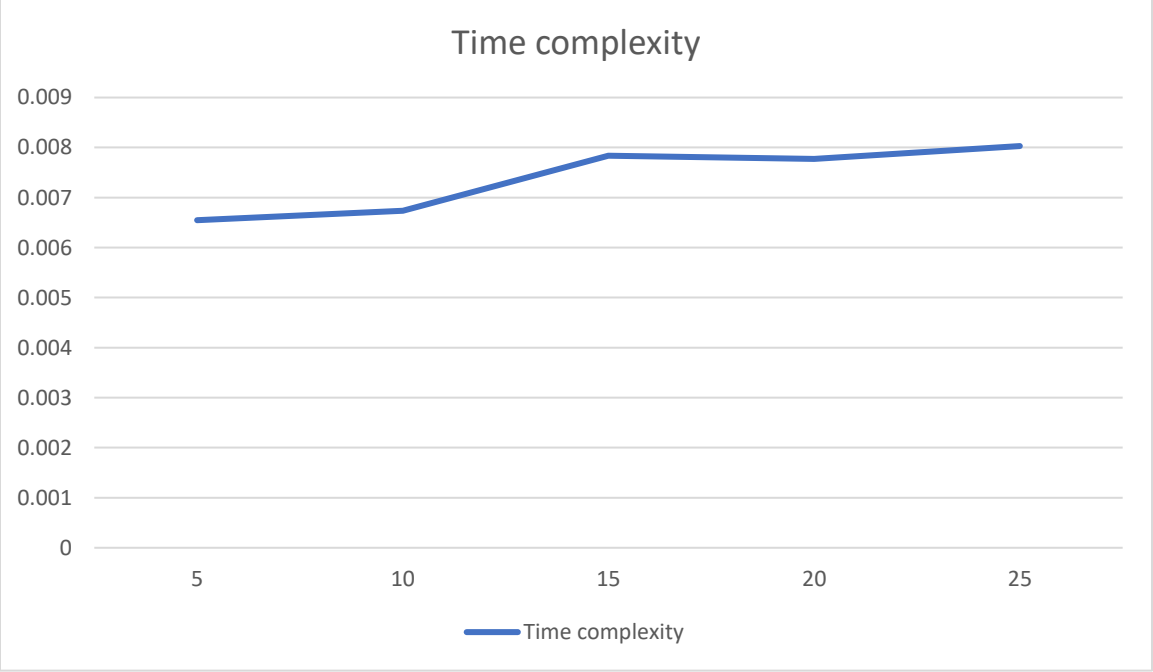
```
        {
                int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high);
        }
}
void printArray(int arr[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                cout << arr[i] << " ";
cout << endl;
}
int main()
{
        int arr[] = {10, 7, 8, 9, 1, 5};
 int n = sizeof(arr) / sizeof(arr[0]);
quickSort(arr, 0, n - 1);   cout << "Sorted
array: \n";
        printArray(arr, n);
        return 0;
}
```

Output:



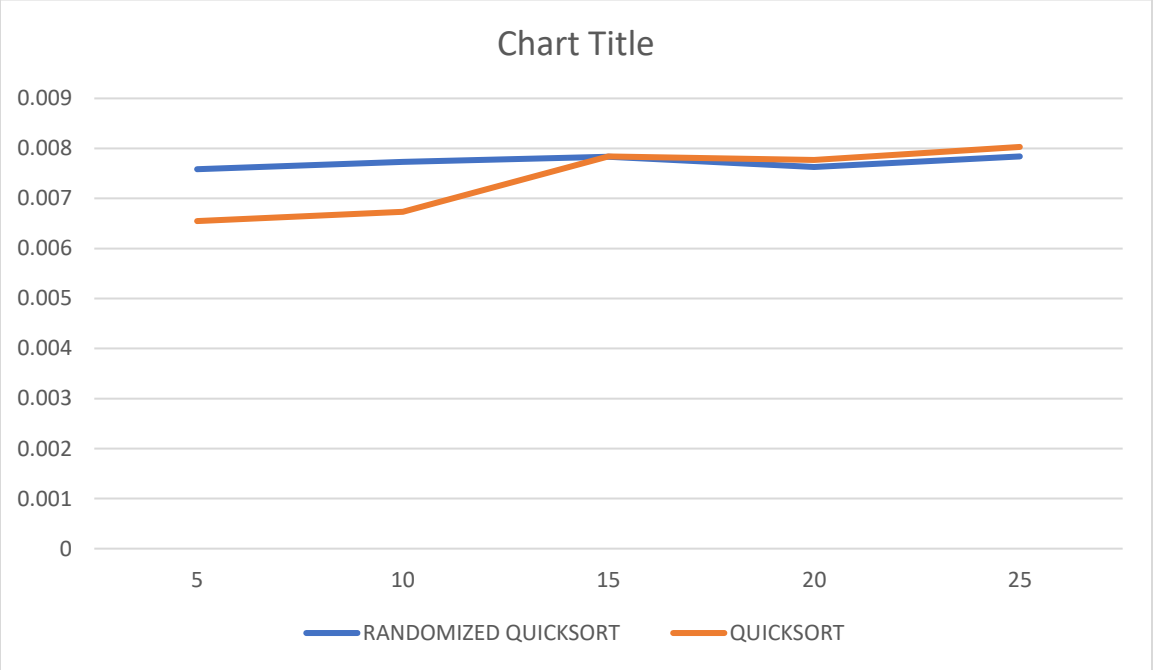
Quick sort:

Size of Array	Time complexity
5	0.006548
10	0.006736
15	0.007840
20	0.007770
25	0.008030



Comparison:

Size of Array	RANDOMIZED QUICKSORT	QUICKSORT
5	0.007585	0.006548
10	0.007730	0.006736
15	0.007835	0.007840
20	0.007630	0.007770
25	0.007840	0.008030



Conclusion:

In this practical we analyzed randomized quicksort and it's time complexity and later on we compared with simple quick sort by entering various input values.