

Name: Vivek Vikram Pundkar

Roll no: 77

GR no: 11910860

Division: C

Batch: 3

Quick Sort using Randomized method

Code:

// Random selection of pivot

```
int RandomPivotPartition(int a[], int low, int high)
{
    int pvt, n, temp;
    n = rand();
    pvt = low + n % (high - low + 1);
    swap(&a[high], &a[pvt]);
    return Partition(a, low, high);
}
```

// Quick Sort algorithm.

```
int QuickSort(int a[], int low, int high)
{
    int pindex;
    if (low < high)
    {
        pindex = RandomPivotPartition(a, low, high);
        QuickSort(a, low, pindex - 1);
        QuickSort(a, pindex + 1, high);
    }
}
```

```

    }
    return 0;
}

```

Output:

The screenshot shows a Visual Studio Code editor with a C++ file named `QuicksortUsingRandomizedMethod.cpp`. The code implements a QuickSort algorithm using a randomized pivot. The `main` function initializes an array `x` with values `{9, 5, 0, 1, 15, 2, 6, 3, 99, 12}`, prints the number of elements (10), the unsorted array, and then the sorted array in ascending order. The output in the terminal confirms these steps, showing the unsorted array `9 5 0 1 15 2 6 3 99 12` and the sorted array `0 1 2 3 5 6 9 12 15 99`. The program exited with code 0 in 1.188 seconds.

```

58     {
59         pindex = RandomPivotPartition(a, low, high);
60         QuickSort(a, low, pindex - 1);
61         QuickSort(a, pindex + 1, high);
62     }
63     return 0;
64 }
65
66 int main()
67 {
68     int x[] = {9, 5, 0, 1, 15, 2, 6, 3, 99, 12};
69     int n = sizeof(x) / sizeof(x[0]);
70     int i;
71
72     cout << "Number of array elements: \n"
73         << n << "\n";
74
75     cout << "Unsorted Array: \n";
76     printArray(x, n);
77
78     QuickSort(x, 0, n - 1);
79
80     cout << "Sorted Array in ascending order: \n";
81     printArray(x, n);
82     return 0;
83 }

```

```

[Running] cd "d:\DAAOS\" && g++ QuicksortUsingRandomizedMethod.cpp -o QuicksortUsingRandomizedMethod && "d:\DAAOS\
Number of array elements:
10
Unsorted Array:
9 5 0 1 15 2 6 3 99 12
Sorted Array in ascending order:
0 1 2 3 5 6 9 12 15 99

[Done] exited with code=0 in 1.188 seconds

```

Analysis:

Time Complexity of Quick Sort

Best case:

The partition is evenly balanced, here the pivot element is close to middle number or same as middle number. The best-case complexity of the quick sort algorithm is **$O(n \log n)$** .

Worst case:

The partition is unbalanced. The worst-case time complexity of Quick Sort is **$O(n^2)$** .

Average case:

Here the number of chances to get a pivot element is equal to the number of items. The average case complexity of the quick sort algorithm is **$O(n \log n)$** .

Space Complexity of Quick sort

The space complexity is determined on basis of the space used in the recursion stack. The space complexity of quicksort is **$O(n \log n)$** .