ASSIGNMENT 3

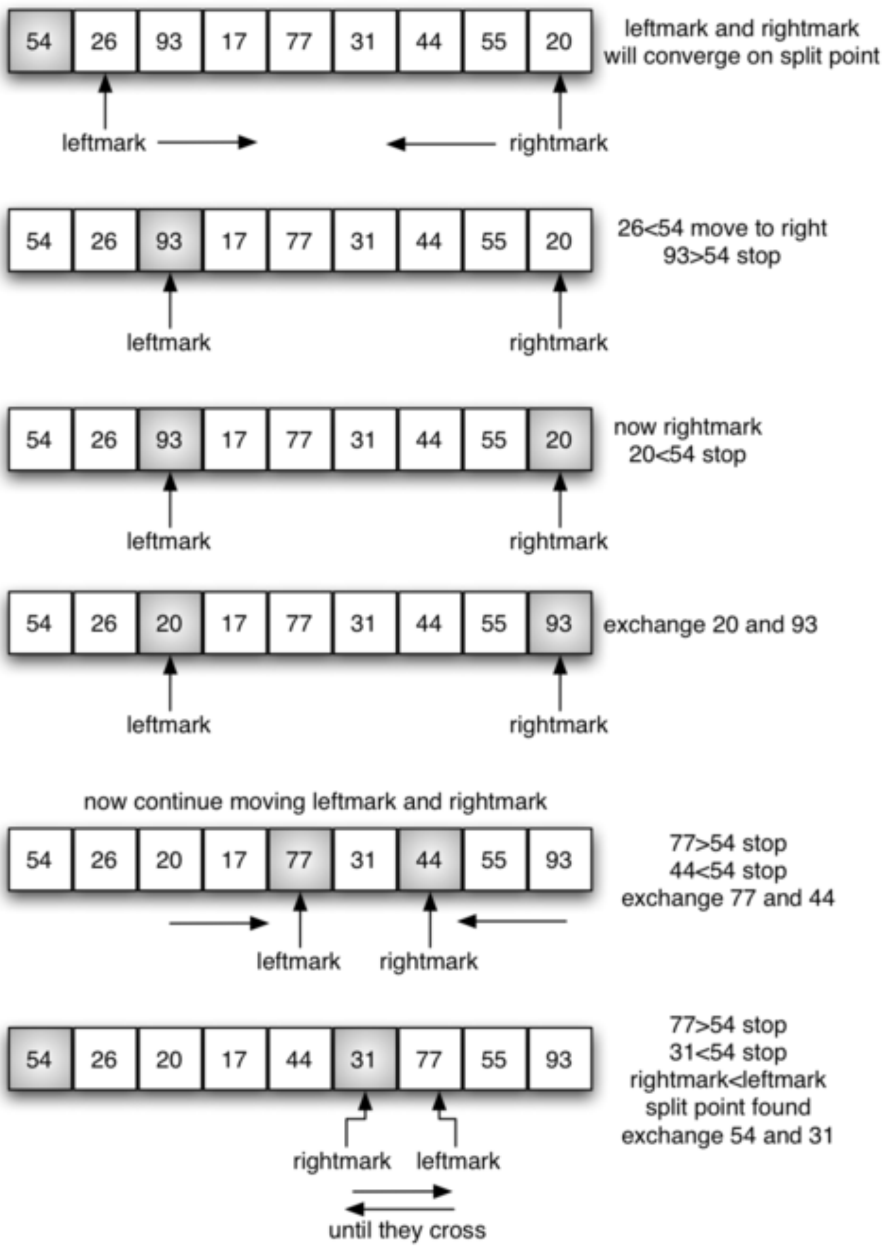
NAME - Keval Panwala
ROLL NO - 21BCP041
GROUP - G2

——Quick Sort——

- AIM -

To write a program in which a user populated an unsorted array of integers and then it was sorted using quick sort. Finally, the program printed out the sorted array to the console.

- INTRODUCTION -



- ALGORITHM -

STEP 1 - Choose the highest index values has pivot.
STEP 2 - Take two variables to point left and right of the list excluding pivot.
STEP 3 - left points to the low index.
STEP 4 - right points to the high.
STEP 5 - while value at left is less than pivot move right.
STEP 6 - while value at right is greater than pivot move left.
STEP 7 - if both step 5 and step 6 does not match swap left and right.
STEP 8 - if left ≥ right, the point where they met is new pivot.

- PROGRAM -

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int partition(int array[], int low, int high, int pivot)
{
    int i = low;
    int j = low;
    while (i <= high) {
        if (array[i]>pivot) {
            i++;
        } else {
            int temp = array[i];
            array[i]=array[j];
            array[j]=temp;
            i++;
            j++;
        }

    }
    return j-1;
}
void QuickSort(int array[], int low, int high)
{
    if(low<high) {
        int pivot = array[high];
        int pos = partition(array, low, high, pivot);
```

```
        QuickSort(array, low, pos - 1);
        QuickSort(array, pos + 1, high);
    }
}
int main(int argc, char const *argv[])
{
    clock_t t, t1, t2;
    srand(time(0));
    int n;
    scanf("%d", &n);
    int array[n];

    for (int i = 0; i < n; i++)
    {
        array[i] = rand();
    }
    t = clock();
    QuickSort(array, 0, n - 1);
    t = clock() - t;
    double time_t = ((double)t)/CLOCKS_PER_SEC;
    for (int i = 0; i < n; i++)
    {
        printf("%d ", array[i]);
    }
    printf("\n");
    printf("Average time is %f\n", time_t);

    t1 = clock();
    QuickSort(array, 0, n - 1);
    t1 = clock() - t1;
    double timet_1 = ((double)t1)/CLOCKS_PER_SEC;
    printf("Best time is %f\n", timet_1);

    t2 = clock();
    QuickSort(array, 0, n - 1);
    t2 = clock() - t2;
    double timet_2 = ((double)t2)/CLOCKS_PER_SEC;
    printf("Worst time is %f\n", timet_2);

    return 0;
}
```

- OUTPUT -

```
6
12 45 23 51 19 8
8 12 19 23 45 51
```

- ANALYSIS -

N	Best Case	Average Case	Worst Case
10	Time: 0.000002	Time: 0.000008	Time: 0.000004
100	Time: 0.000046	Time: 0.000018	Time: 0.000048
1000	Time: 0.003248	Time: 0.000179	Time: 0.003704
10000	Time: 0.103727	Time: 0.001804	Time: 0.124799
100000	Time: 10.617897	Time: 0.024996	Time: 10.583589

<u>Worst Case Time Complexity</u> -

It occurs when the pivot element picked is either the greatest or the smallest element.

This condition leads to the case in which the pivot element lies in an extreme end of the sorted array. One sub-array is always empty and another sub-array contains $n - 1$ elements. Thus, quick sort is called only on this sub-array.

However, the quick sort algorithm has better performance for scattered pivots.

<u>Best Case Time Complexity</u> -

It occurs when the pivot element is always the middle element or near to the middle element.

<u>Average Case Time Complexity</u> -

It occurs when the above conditions do not occur.

Space Complexity -

The space complexity for quick sort is `O(log n)`.

- APPLICATIONS -

Quick sort algorithm is used when:
- The programming language is good for recursion.
- Time complexity matters.
- Space complexity matters.

- REFERENCES -

https://www.programiz.com/dsa/quick-sort