Student Name:- Chaudhary Hamdan

Student Roll No.:- 1905387

Algorithm Lab. Class Assignment-6

CSE Group 1

Date: - 13th August 2021

1. **Write a program to sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted, and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

SL No.	Input (n)	Quick sort		
		Best Case (Time complexity)	Avg Case (Time complexity)	Worst Case (Time complexity)
1	10000	0.001000	0.193000	0.266000
2	15000	0.002000	0.419000	0.592000
3	20000	0.002000	0.740000	1.033000
4	25000	0.003000	1.156000	1.602000
5	30000	0.004000	1.658000	2.312000
6	35000	0.005000	2.251000	3.146000
7	40000	0.005000	2.964000	4.130000

Program

```
// Author: Chaudhary Hamdan

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

#define sf(x)           scanf("%d", &x)
#define pf              printf
#define pfs(x)          printf("%d ", x)
#define pfn(x)          printf("%d\n", x)
#define pfc(x)          printf("%d, ", x)
```

```
#define F(i,x,y)          for(int i = x; i < y; i++)
#define FI(i,x,y,inc)      for(int i = x; i < y; i += inc)
#define RF(i,x,y)         for(int i = x; i >= y; i--)
#define pfa(i,a,n)         for(int i = 0; i < n-1; i++) printf("%d ",a[i]); printf("%d\n",
a[n-1]);


void i_o_from_file() {

#ifndef ONLINE_JUDGE
    freopen("C:\\Users\\KIIT\\input", "r", stdin);
    freopen("C:\\Users\\KIIT\\output", "w", stdout);
#endif
}


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);
    }
}




int main() {

    i_o_from_file();

    /* ********************************************** */

    pf("n\t\t|\tworst\t\tavg\t\tbest\n________|_____________________________
_____\n");
```

```
int sizes;
sf(sizes);

F(i, 0, sizes) {
	int n;
	sf(n);

	pf("%d\t|\t", n);
	int arr[n];
	time_t start, end;
	double time;

	// Worst
	F(j, 0, n) {
		arr[j] = j + 1;
	}

	start = clock();
	quickSort(arr, 0, n - 1);
	end = clock();

	time = (end - start) * 1.0 / CLOCKS_PER_SEC;

	pf("%f\t", time);

	// Avg
	F(j, 0, n) {
		arr[j] = n - j;
	}
```

```
            start = clock();
            quickSort(arr, 0, n - 1);
            end = clock();

            time = (end - start) * 1.0 / CLOCKS_PER_SEC;

            pf("%f\t", time);


            // Best
            F(j, 0, n) {
                  arr[j] = rand() % 10000;
            }

            start = clock();
            quickSort(arr, 0, n - 1);
            end = clock();

            time = (end - start) * 1.0 / CLOCKS_PER_SEC;

            pf("%f\n", time);
      }
      return 0;
}
```

Output

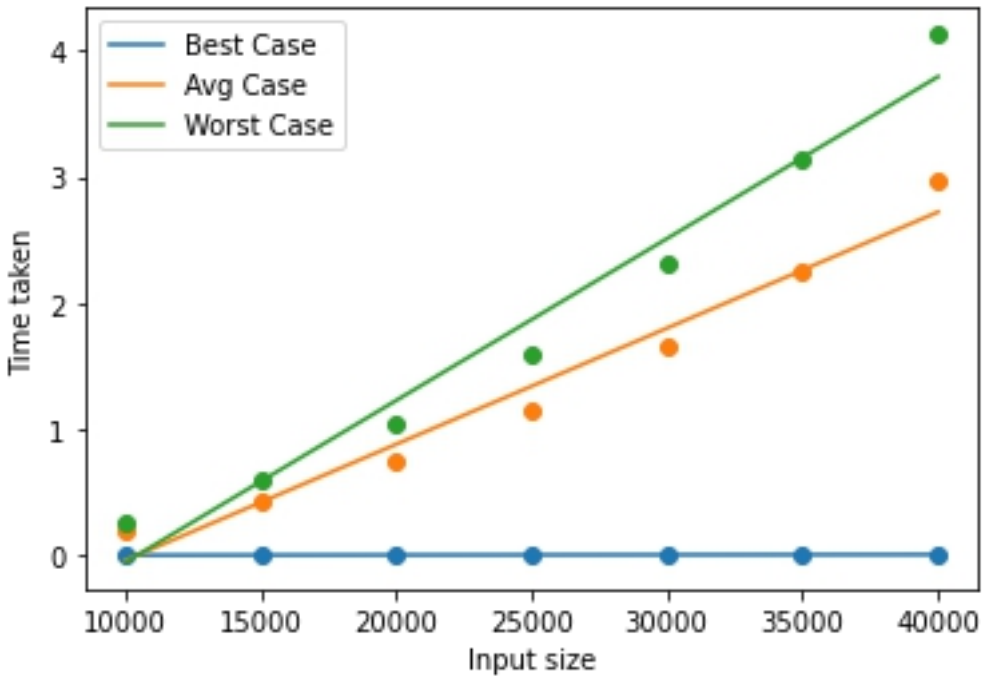
input

```
7
10000
15000
20000
25000
30000
35000
40000
```

output

```
n       |   worst       avg         best
________|_____________________________________
10000   |   0.266000    0.193000    0.001000
15000   |   0.592000    0.419000    0.002000
20000   |   1.033000    0.740000    0.002000
25000   |   1.602000    1.156000    0.003000
30000   |   2.312000    1.658000    0.004000
35000   |   3.146000    2.251000    0.005000
40000   |   4.130000    2.964000    0.005000
```

Graph



2. **Write a program to use the divide and conquer method to recursively implement and to find the maximum and minimum in a given list of n elements.**

SL No.	Input (n)	(Time complexity)
1	10000	0.001000
2	15000	0.002000
3	20000	0.003000
4	25000	0.004000
5	30000	0.004500
6	35000	0.005000
7	40000	0.005000

Program

```
// Author: Chaudhary Hamdan

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

#define sf(x)            scanf("%d", &x)
#define pf               printf
#define pfs(x)           printf("%d ", x)
#define pfn(x)           printf("%d\n", x)
#define pfc(x)           printf("%d, ", x)
#define F(i,x,y)         for(int i = x; i < y; i++)
#define FI(i,x,y,inc)    for(int i = x; i < y; i += inc)
#define RF(i,x,y)        for(int i = x; i >= y; i--)
#define pfa(i,a,n)       for(int i = 0; i < n-1; i++) printf("%d ",a[i]); printf("%d\n",
a[n-1]);


void i_o_from_file() {
```

```
#ifndef ONLINE_JUDGE
    freopen("C:\\Users\\KIIT\\input", "r", stdin);
    freopen("C:\\Users\\KIIT\\output", "w", stdout);
#endif
}

int find_max(int arr[], int s, int e) {

    if (s > e) {
            return -(1 << 30);
    }

    if (s == e) {
            return arr[s];
    }

    int rem = find_max(arr, s + 1, e);

    return ((arr[s] > rem) ? arr[s] : rem);

}


int find_min(int arr[], int s, int e) {

    if (s > e) {
            return -(1 << 30);
    }

    if (s == e) {
            return arr[s];
```

```
    }

    int rem = find_max(arr, s + 1, e);

    return ((arr[s] < rem) ? arr[s] : rem);


}


int main() {

    i_o_from_file();

    /* ****************************************** */

    pf("n\t\t|\tTime\t\tMax\t\tMin\n");

    int sizes;
    sf(sizes);

    F(i, 0, sizes) {
        int n;
        sf(n);

        pf("%d\t|\t", n);
        int arr[n];
        time_t start, end;
        double time;

        F(j, 0, n) {
```

```
            arr[j] = rand() % 5000 + 1;
        }

        start = clock();
        int ans1 = find_max(arr, 0, n - 1);
        int ans2 = find_min(arr, 0, n - 1);
        end = clock();

        time = (end - start) * 1.0 / CLOCKS_PER_SEC;

        pf("%f\t%d\t%d\n", time, ans1, ans2);

    }


    return 0;
}
```

Output

input

```
7
10000
15000
20000
25000
30000
35000
40000
```

output

```
n       |   Time          Max     Min
10000   |   0.010000      5000    42
15000   |   0.020000      5000    4485
20000   |   0.030000      5000    694
25000   |   0.040000      5000    1016
30000   |   0.045000      5000    4804
35000   |   0.050000      5000    2651
40000   |   0.050000      5000    1939
```

Graph