Student Name:- CHAUDHARY HAMDAN

Student Roll No.:- 1905387

Algorithm Lab. Class Assignment-9

CSE Group 1

Date: - 24th Sept. 2021

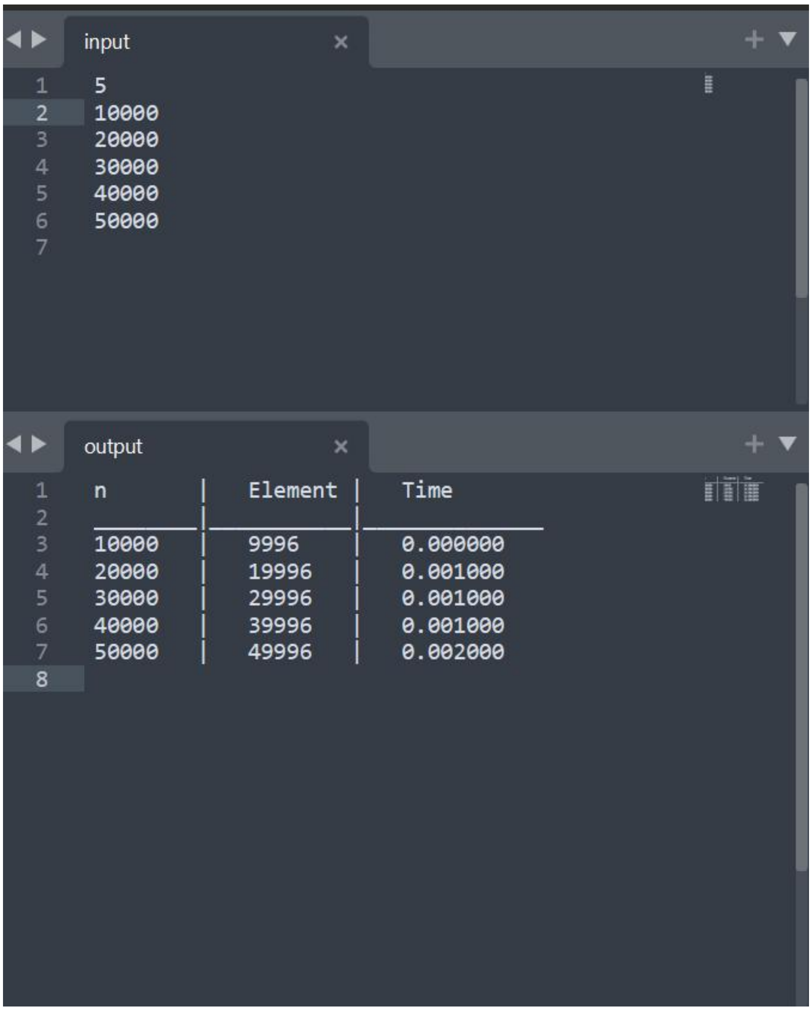
1. **Write a program to find the kth minimum and maximum element in Heap.**

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;
}
void heapify(int *arr, int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest])
            largest = l;

    if (r < n && arr[r] > arr[largest])
            largest = r;

    if (largest != i) {
            swap(arr + i, arr + largest);

            heapify(arr, n, largest);
    }
}
void buildHeap(int *arr, int n)
{
    int startIdx = (n / 2) - 1;
    for (int i = startIdx; i >= 0; i--) {
            heapify(arr, n, i);
    }
}
int extract_maximum(int *arr, int n) {

    int m = *arr;

    arr[0] = arr[n - 1];

    heapify(arr, n - 1, 0);

    return m;
}

int kthMax(int *arr, int n, int k) {

    int ans = 0;

    F(i, 0, k) {
            ans = extract_maximum(arr, n);
            n--;
    }
    return ans;
}
int main() {

    i_o_from_file();

    /* ******************************************** */
    pf("n\t\t|\tElement |\tTime\n________|___________|______________\n");

    int sizes;
    sf(sizes);

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

            pf("%d\t|\t", n);
            int arr[n];
            F(j, 0, n) {
                arr[j] = 1 + j;
            }
            time_t start, end;
            double time;
            start = clock();

            buildHeap(arr, n);
            // Time
            pfs(kthMax(arr, n, 5));
            pf("\t|\t");
            end = clock();
            time = (end - start) * 1.0 / CLOCKS_PER_SEC;
            pf("%f\n", time);
    }
    return 0;
}
```

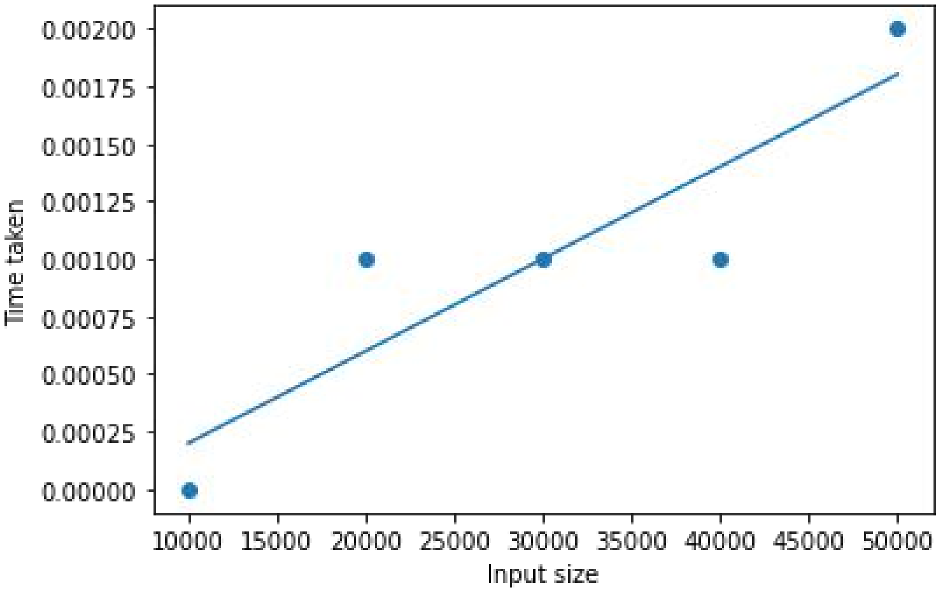
Output



```
5
10000
20000
30000
40000
50000
```

```
n        |   Element |    Time
         |           |
_________|___________|______________
10000    |   9996    |   0.000000
20000    |   19996   |   0.001000
30000    |   29996   |   0.001000
40000    |   39996   |   0.001000
50000    |   49996   |   0.002000
```

Graph



2. **Write a program to recursively implement Binary Search using divide and conquer method. Determine the time required to search an element in an array of n integers. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n. The n integers can be generated randomly.**

Program

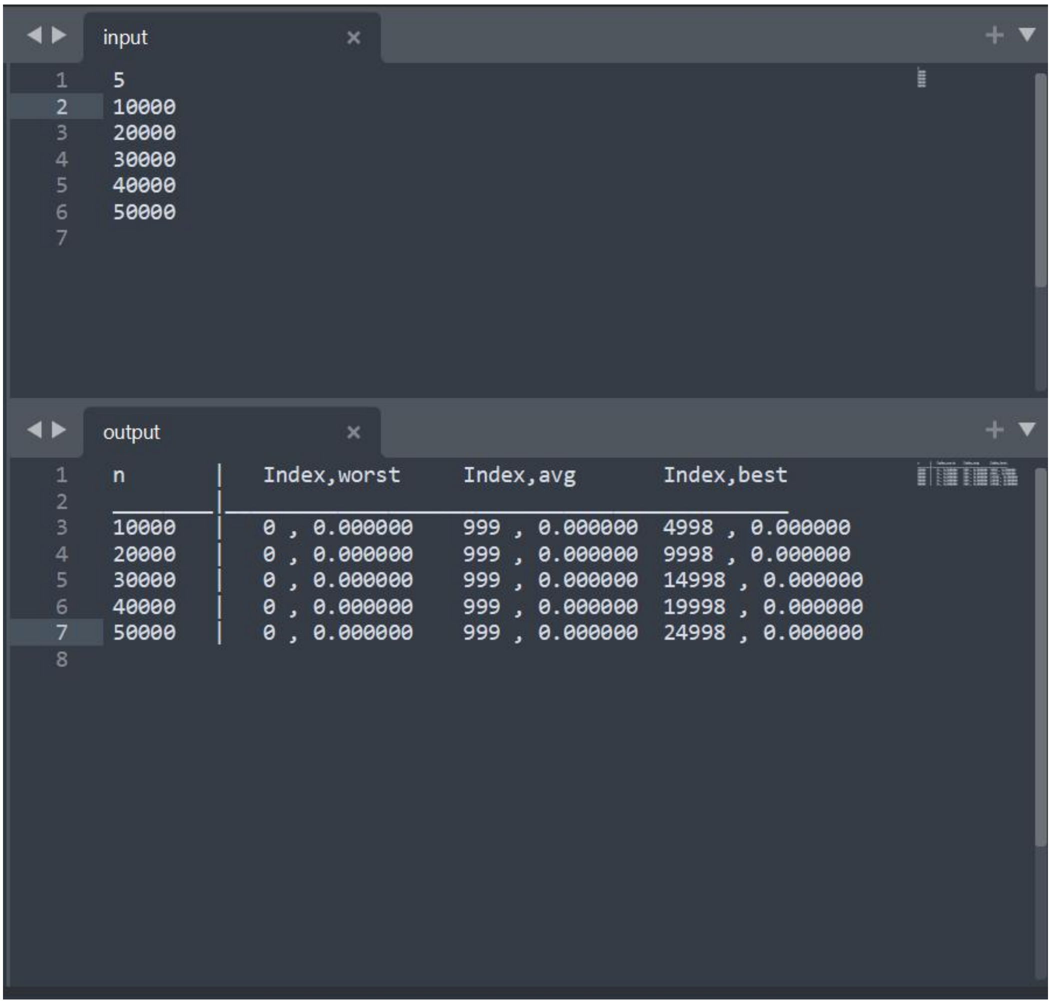
```
// Author: Chaudhary Hamdan


#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 binSearch(int *a, int s, int e, int x) {

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

    int m = (s + e) / 2;

    if (a[m] == x)
            return m;

    if (a[m] > x)
            return binSearch(a, s, m - 1, x);

    if (a[m] < x)
            return binSearch(a, m + 1, e, x);

}

int main() {

    i_o_from_file();

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


    pf("n\t\t|\tIndex,worst\t\tIndex,avg\t\tIndex,best\n________|________________
______________________________\n");
```

```
int sizes;
sf(sizes);

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

        pf("%d\t|\t", n);
        int arr[n];
        F(j, 0, n) {
                arr[j] = 1 + j;
        }
        time_t start, end;
        double time;

        // Worst

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

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

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

        // Avg

        start = clock();
        pfs(binSearch(arr, 0, n - 1, 1000));
```

```
            end = clock();

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

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

            // Best

            start = clock();
            pfs(binSearch(arr, 0, n - 1, (n - 1) / 2));
            end = clock();

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

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

    return 0;
}
```

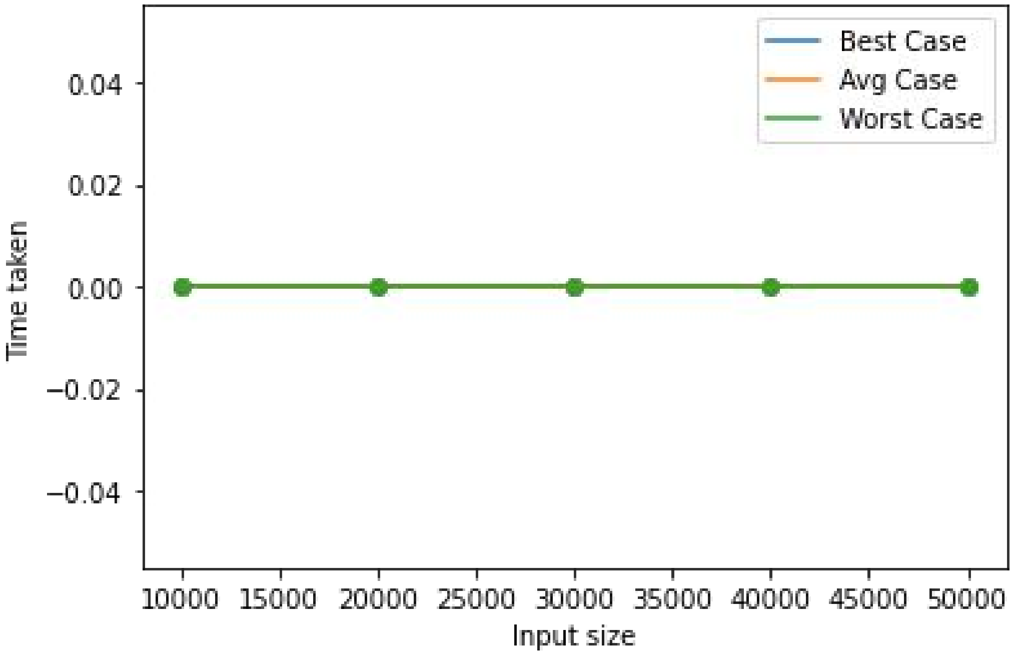
Output



```
5
10000
20000
30000
40000
50000
```

```
n        |    Index,worst     Index,avg       Index,best
_________|____________________________________________
10000    |   0 , 0.000000    999 , 0.000000  4998 , 0.000000
20000    |   0 , 0.000000    999 , 0.000000  9998 , 0.000000
30000    |   0 , 0.000000    999 , 0.000000  14998 , 0.000000
40000    |   0 , 0.000000    999 , 0.000000  19998 , 0.000000
50000    |   0 , 0.000000    999 , 0.000000  24998 , 0.000000
```

Graph



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

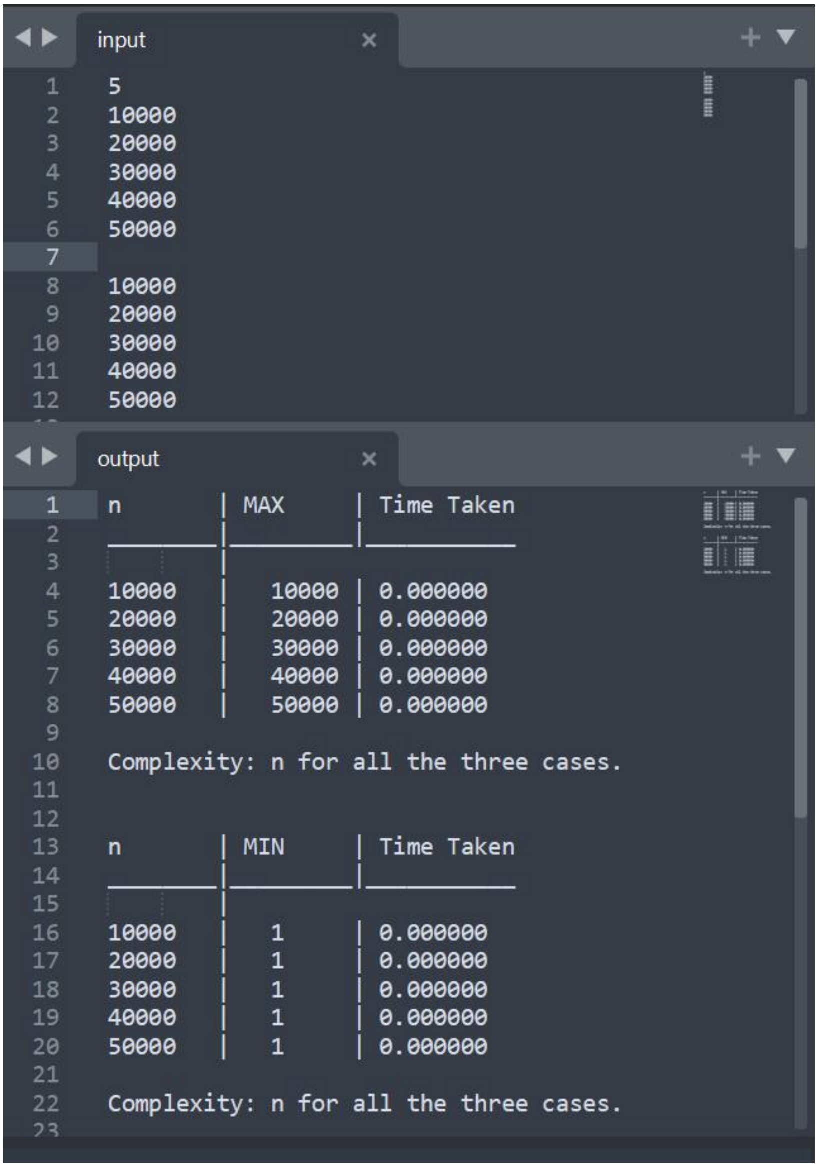
Program

```
// 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 max(int a, int b) {
    if (a > b) {
            return a;
    }
    return b;

}



int getMax(int *a, int i, int n) {

    if (i == n - 2) {

            return max(a[i], a[i + 1]);

    }

    return max(a[i], getMax(a, i + 1, n));

}

int min(int a, int b) {

    if (a < b) {
            return a;
    }
    return b;

}
```

```
int getMin(int *a, int i, int n) {

    if (i == n - 2) {

            return min(a[i], a[i + 1]);

    }

    return min(a[i], getMax(a, i + 1, n));

}



int main() {

    i_o_from_file();

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

    pf("n\t\t| MAX\t  |\tTime Taken\n_______|________|_________\n\t\t|\n");

    int sizes;
    sf(sizes);

    int n;
    int arr[50005];

    F(i, 0, sizes) {
```

```
    sf(n);

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

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

    start = clock();
    pf("%d | ", getMax(arr, 0, n));
    end = clock();

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

    pf("%f\n", time);
    // pfa(i, arr, n);

}

pf("\nComplexity: n for all the three cases.\n");



pf("\n\n");

pf("n\t\t| MIN\t  |\tTime Taken\n________|_________|___________\n\t\t|\n");

F(i, 0, sizes) {
        pf("%d\t|\t", n);
        int arr[n];
        time_t start, end;
        double time;

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

        start = clock();
        pf("%d | ", getMin(arr, 0, n));
        end = clock();

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

        pf("%f\n", time);
        // pfa(i, arr, n);

    }

    pf("\nComplexity: n for all the three cases.\n");



    return 0;
}
```

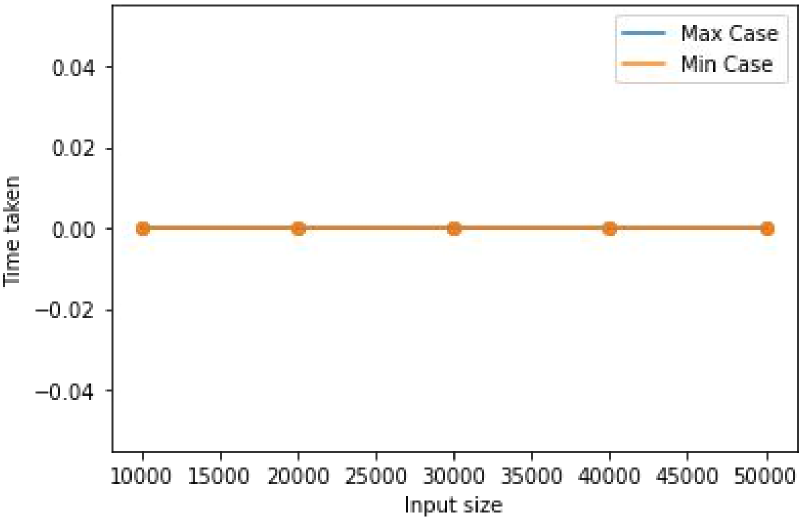
Output



```
5
10000
20000
30000
40000
50000

10000
20000
30000
40000
50000
```

```
n       | MAX     | Time Taken
________|_________|___________
        |
10000   |   10000 | 0.000000
20000   |   20000 | 0.000000
30000   |   30000 | 0.000000
40000   |   40000 | 0.000000
50000   |   50000 | 0.000000

Complexity: n for all the three cases.


n       | MIN     | Time Taken
________|_________|___________
        |
10000   |   1     | 0.000000
20000   |   1     | 0.000000
30000   |   1     | 0.000000
40000   |   1     | 0.000000
50000   |   1     | 0.000000

Complexity: n for all the three cases.
```

Graph



Max Case
Min Case
Time taken
Input size