**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;

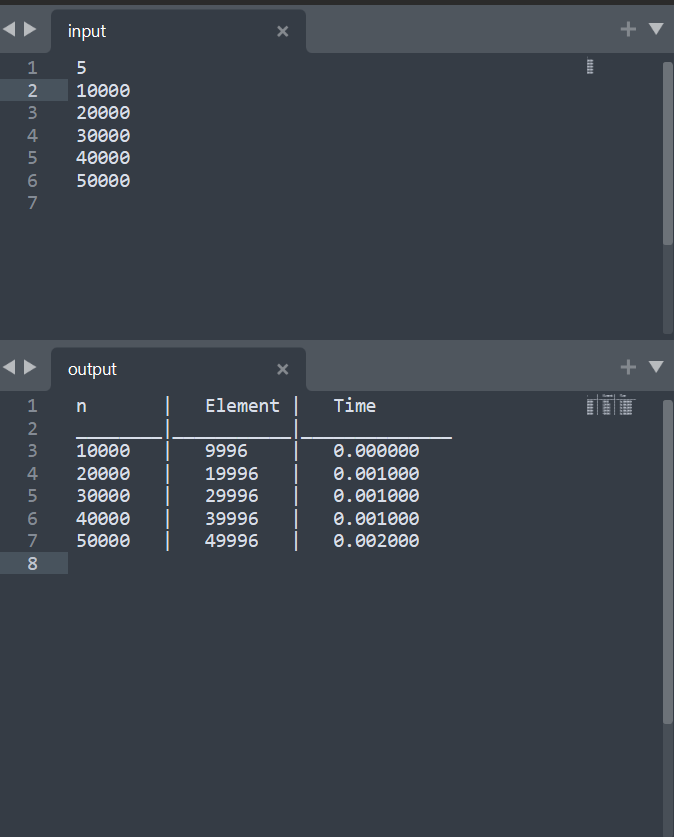
pf("%f\n", time);

}

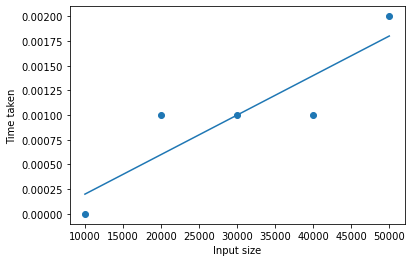
return 0;

}

**Output**

****

**Graph**

****

1. **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;

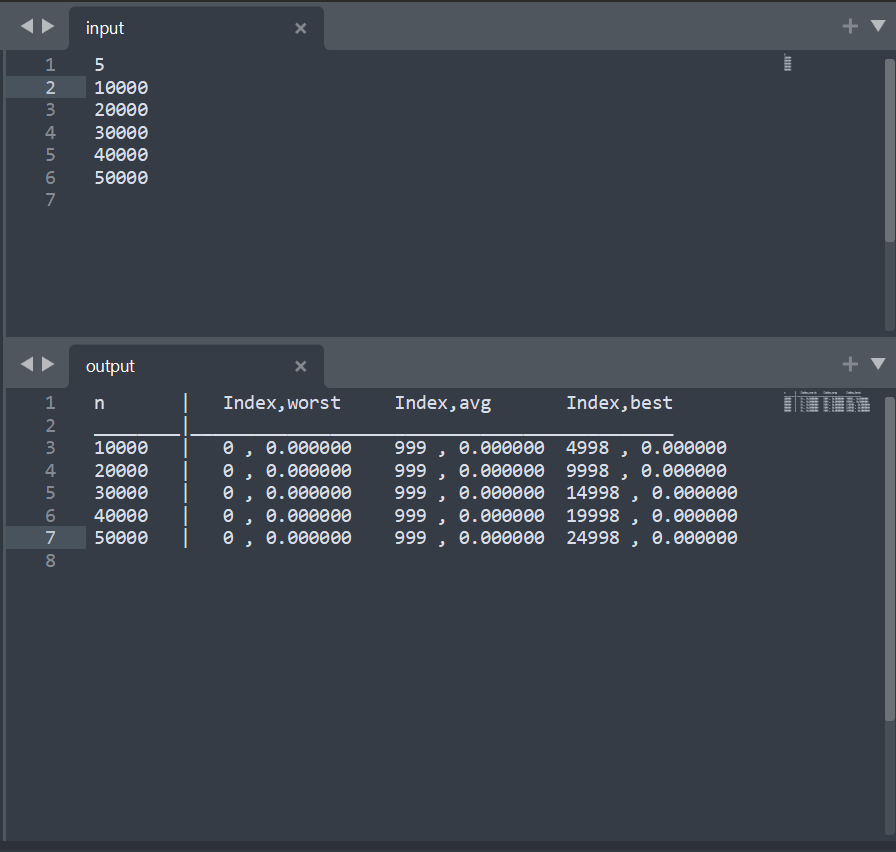
pf(", %f\n", time);

}

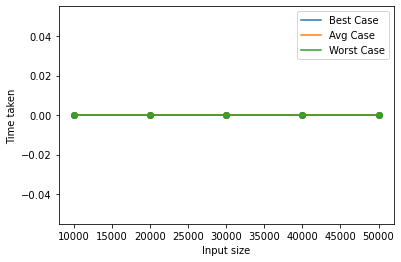
return 0;

}

**Output**

****

**Graph**

****

1. **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);

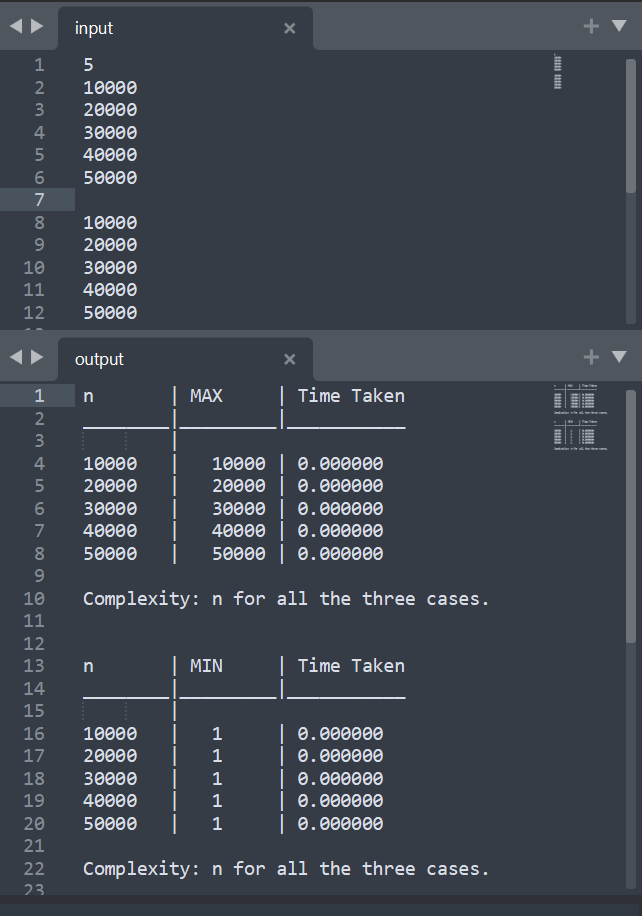
}

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

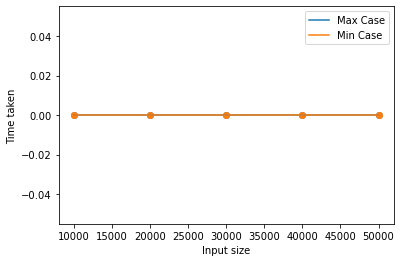
return 0;

}

**Output**

****

**Graph**

****