Student Name:- CHAUDHARY HAMDAN

Student Roll No.:- 1905387

Algorithm Lab. Class Assignment-8

CSE Group 1

Date: - 3rd Sept. 2021

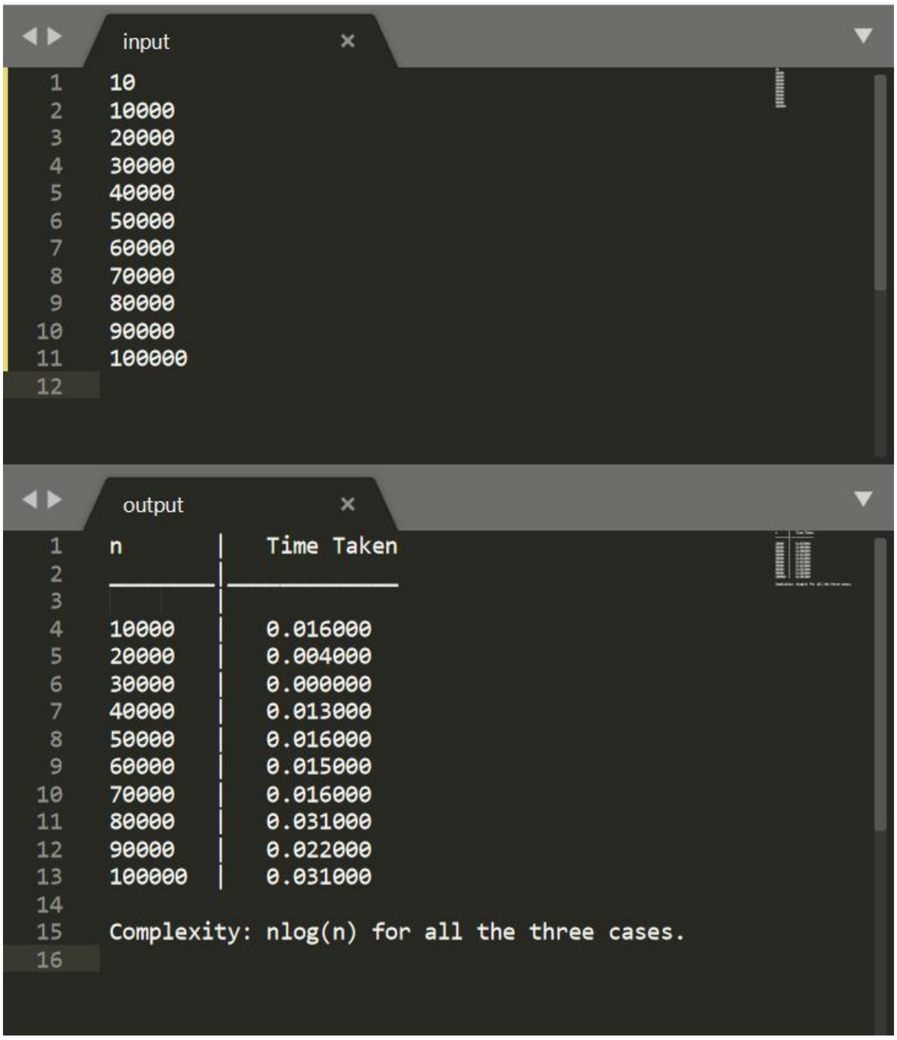
1. **Write a program to sort a given set of elements using the heap sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read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

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 heapSort(int *arr, int n) {

    for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {
            swap(arr + 0, arr + i);
            heapify(arr, i, 0);
    }
}


int main() {

    i_o_from_file();

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

    pf("n\t\t|\tTime Taken\n________|_____________\n\t\t|\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() % 100000;
            }

            start = clock();
            heapSort(arr, n);
            end = clock();

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

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

    }

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

    return 0;
}
```

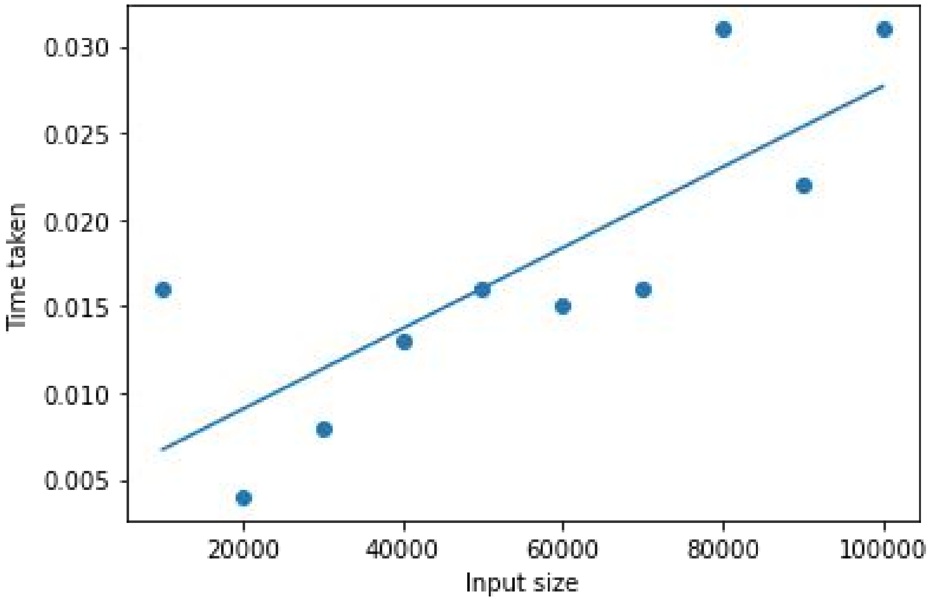
Output



```
10
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
```

```
n       |   Time Taken
________|_____________
        |
10000   |   0.016000
20000   |   0.004000
30000   |   0.000000
40000   |   0.013000
50000   |   0.016000
60000   |   0.015000
70000   |   0.016000
80000   |   0.031000
90000   |   0.022000
100000  |   0.031000

Complexity: nlog(n) for all the three cases.
```

Graph



2. Write a program to Perform following operations on MAX HEAP and find the time complexity for each of them.

   A. maximum(Arr) : It returns maximum element from the heap.

   B. extract_maximum (Arr) - It removes and return the maximum element from the heap.

   C. increase_val (Arr, i , val) - It increases the key of element stored at index i in heap to new value val.

   D. insert_val (Arr, val) - It inserts the element with value val 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() {
```

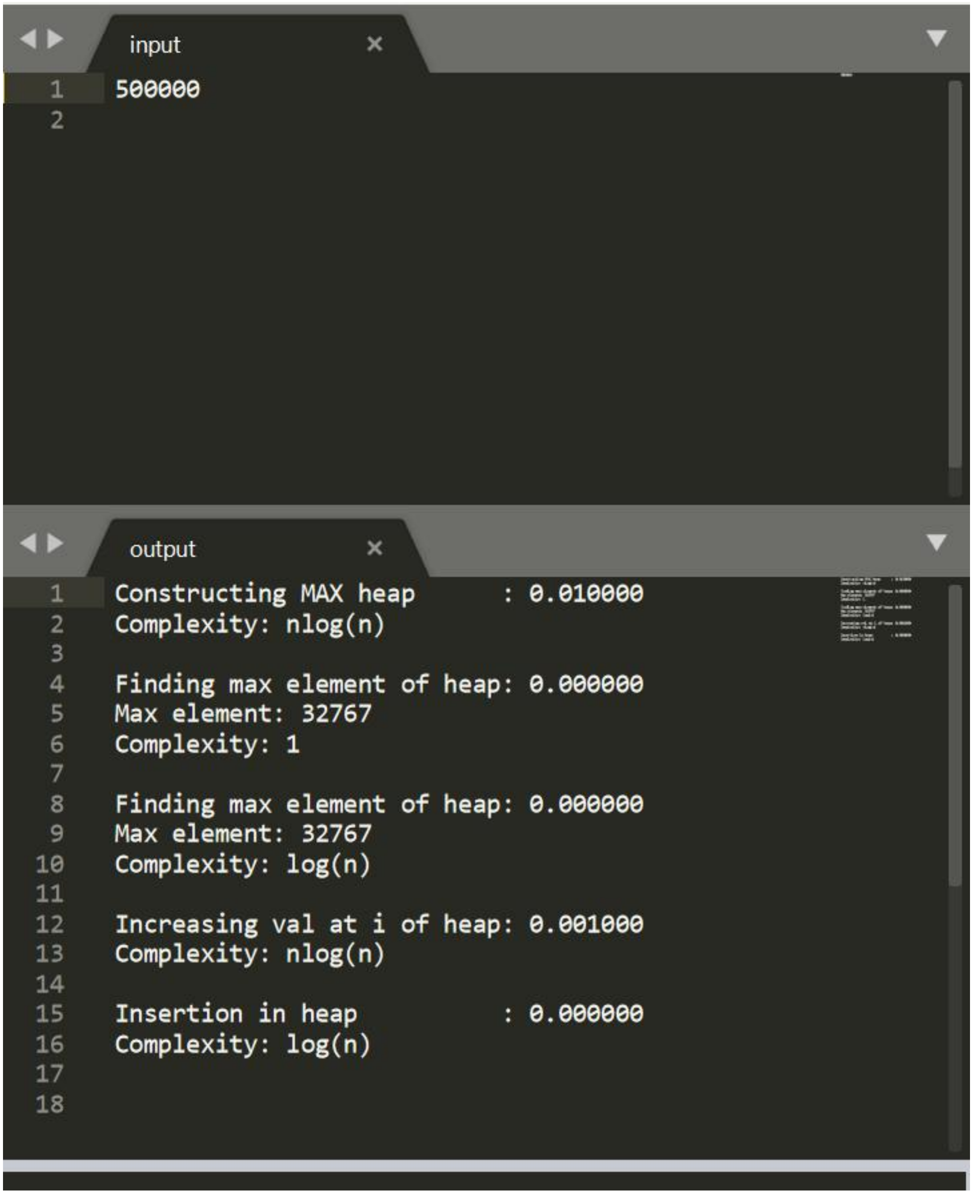
```
#ifndef ONLINE_JUDGE
    freopen("C:\\Users\\KIIT\\input", "r", stdin);
    freopen("C:\\Users\\KIIT\\output", "w", stdout);
#endif
}


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 maximum(int *arr, int n) {
    return *arr;
}

int extract_maximum(int *arr, int n) {

    int m = *arr;

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

    heapify(arr, n - 1, 0);

    return m;
}

void increase_val(int *arr, int i , int val, int n) {
    arr[i] = val;
    buildHeap(arr, n);

}

void insert_val(int *arr, int n, int val)
{
    n++;
    arr[n - 1] = val;

    heapify(arr, n, n - 1);
}




int main() {

    i_o_from_file();

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

    // pf("n\t\t|\tTime Taken\n_________|______________\n\t\t|\n");


    int n;
    sf(n);

    // Constructing
```

```
pf("Constructing MAX heap      : ");
int arr[n];
time_t start, end;
double time;

F(j, 0, n) {
        arr[j] = rand() % 100000;
}

start = clock();
buildHeap(arr, n);
end = clock();

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

pf("%f\n", time);
pf("Complexity: nlog(n)\n\n");



// Max of heap
pf("Finding max element of heap: ");
start = clock();
int m = maximum(arr, n);
end = clock();
time = (end - start) * 1.0 / CLOCKS_PER_SEC;

pf("%f\n", time);
pf("Max element: %d\n", m);
pf("Complexity: 1\n\n");
```

```
// Max of heap
pf("Finding max element of heap: ");
start = clock();
m = extract_maximum(arr, n);
n--;
end = clock();
time = (end - start) * 1.0 / CLOCKS_PER_SEC;

pf("%f\n", time);
pf("Max element: %d\n", m);
pf("Complexity: log(n)\n\n");



// Increase val at i of heap
pf("Increasing val at i of heap: ");
start = clock();
increase_val(arr, 5, 9999999, n);
end = clock();
time = (end - start) * 1.0 / CLOCKS_PER_SEC;

pf("%f\n", time);
pf("Complexity: nlog(n)\n\n");

// Insert val in heap
pf("Insertion in heap          : ");
start = clock();
insert_val(arr, n, 9999998);
n++;
```

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

    pf("%f\n", time);
    pf("Complexity: log(n)\n\n");

    return 0;
}
```

Output



```
500000
```

```
Constructing MAX heap       : 0.010000
Complexity: nlog(n)

Finding max element of heap: 0.000000
Max element: 32767
Complexity: 1

Finding max element of heap: 0.000000
Max element: 32767
Complexity: log(n)

Increasing val at i of heap: 0.001000
Complexity: nlog(n)

Insertion in heap           : 0.000000
Complexity: log(n)
```