**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);**

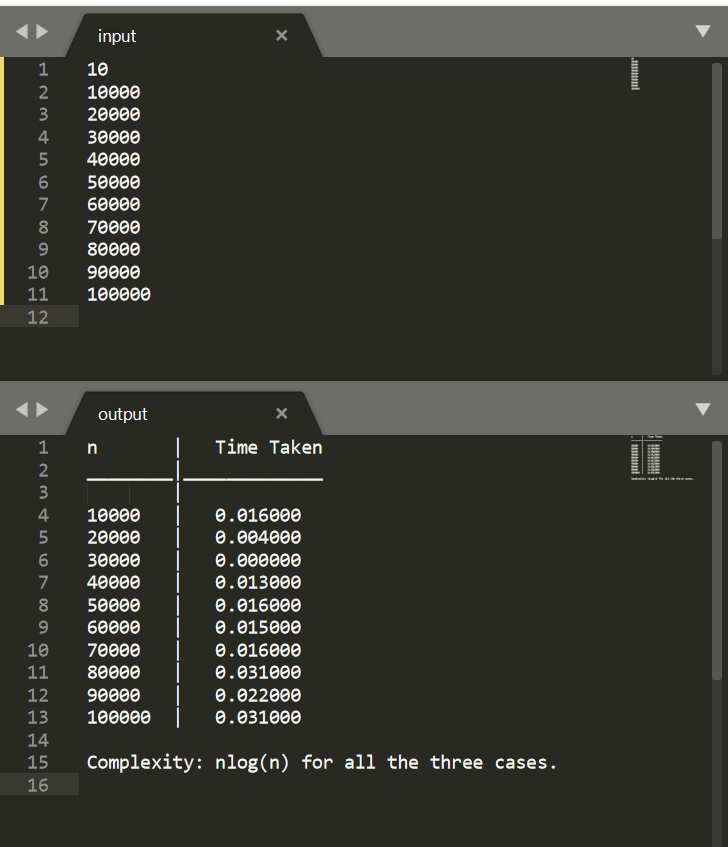
**}**

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

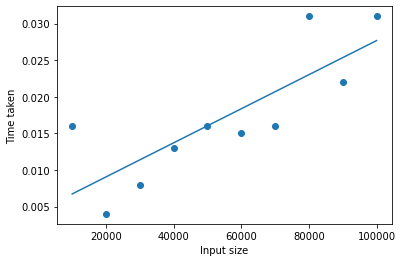
**return 0;**

**}**

**Output**

****

**Graph**

****

1. **Write a program to Perform following operations on MAX HEAP and find the time complexity for each of them.**
2. **maximum(Arr) : It returns maximum element from the heap.**
3. **extract\_maximum (Arr) - It removes and return the maximum element from the heap.**
4. **increase\_val (Arr, i , val) - It increases the key of element stored at index**

**i in heap to new value val.**

1. **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;**

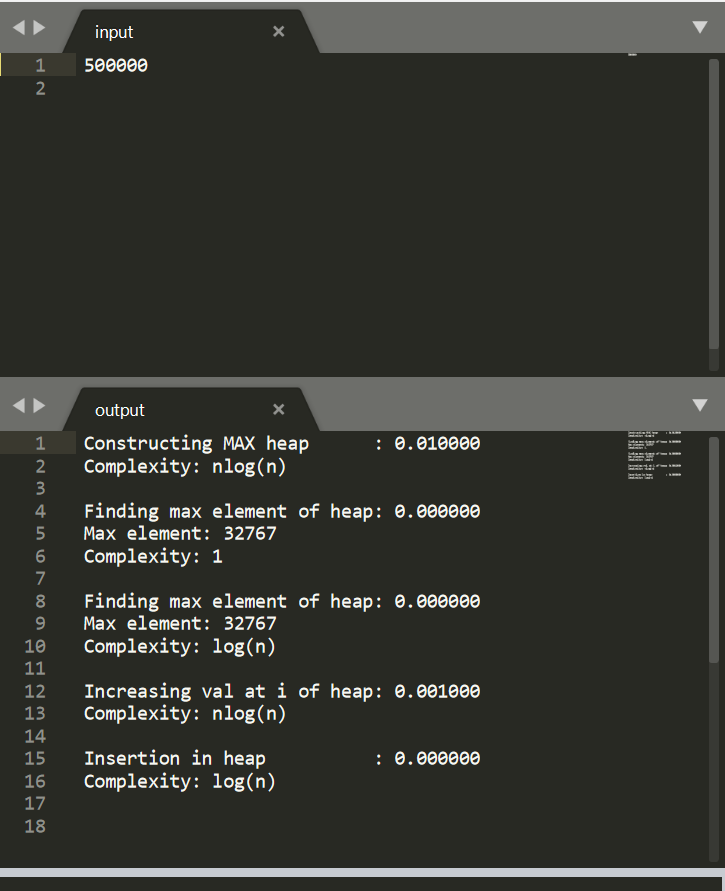
**pf("%f\n", time);**

**pf("Complexity: log(n)\n\n");**

**return 0;**

**}**

**Output**

****