**Họ và Tên: Trần Thị Ngọc Diệp**

**MSSV: 1827005**

**Lớp: B2HK182 – Cấu Trúc Dữ Liệu và Giải Thuật**

**(TUT 7 & LAB 7)**

------------------------

**TUT 7**

**Question 1: Building max heap for the 2 following arrays:**

**int maxHeap1[8] = {56, 45, 4, 77, 60, 34, 35, 22};**

|  |  |
| --- | --- |
| **Number** | **Max Heap** |
| 56 |  |
| 45 |  |
| 4 |  |
| 77 |  |
| 60 |  |
| 34 |  |
| 35 |  |
| 22 |  |

**int maxHeap2[9] = { 1, 3, 5, 7, 9, 2, 4, 6, 8};**

|  |  |
| --- | --- |
| **Number** | **Max Heap** |
| 1 |  |
| 3 |  |
| 5 |  |
| 7 |  |
| 9 |  |
| 2 |  |
| 4 |  |
| 6 |  |
| 8 |  |

**Question 2: Delete 2 numbers from the 2 Heap in Question 1**

|  |  |
| --- | --- |
| **Number** | **Max Heap** |
| 77 | **Before**    **After** |
| 35 |  |

|  |  |
| --- | --- |
| **Number** | **Max Heap** |
| 9 | Before    After |
| 7 |  |

**LAB 7**

#include <iostream>

#include <sstream>

#include <math.h>

#include <stdbool.h>

#define defaultSize 25

using namespace std;

class HeapNode {

public:

int data;

int index;

};

class Heap {

public:

HeapNode\* A; // The heap

int counter; // The number of items in the heap

int maxSize; // The max size of A.

Heap();

Heap(int newdata); // Insert new data to the heap

void reheapUp(int position);

void reheapDown(int position);

void insertHeap(int newdata); // Insert new data to the heap

void printHeap(); // Print Heap as a binary tree //Chi in duoc max heap theo thu tu, chua in ra theo dung chuan bai yeu cau

void printHeapLinear(); // Print Heap as an array

bool isMaxHeap(); // Check if an array is a heap

void deleteMax(); // Delete the maximum element of the Heap

void deleteHeapNode(int position); // Delete a specific node of the heap

};

Heap::Heap(){

this->counter = 0;

this->maxSize = defaultSize;

A = new HeapNode[defaultSize];

}

Heap::Heap(int newdata){

A[counter-1].data = newdata;

A[counter-1].index = this->counter;

this->counter++;

}

void Swap(HeapNode &A, HeapNode &B){

HeapNode t = A;

A = B;

B = t;

}

void Heap::reheapUp (int position){

if (position != 0){

int parent = (position - 1)/2;

if (A[position].data > A[parent].data){

Swap(A[position], A[parent]);

reheapUp(parent);

}

}

else {

return;

}

}

void Heap::reheapDown(int position){

int leftChild = position\*2 + 1;

int rightChild = position\*2 + 2;

int child;

if (leftChild<this->counter){

if (rightChild< this->counter && this->A[leftChild].data < this->A[rightChild].data){

child = rightChild;

}

else {

child = leftChild;

}

}

else {

return;

}

if (this->A[position].data < this->A[child].data){

Swap(A[position], A[child]);

reheapDown(child);

}

}

void Heap::insertHeap(int newdata){

if (this->counter == this->maxSize){

return;

}

else {

this->A[counter].data = newdata;

this->A[counter].index = counter;

reheapUp(counter);

counter++;

}

}

//Check later//////////////////////////////////////////////

void Heap::printHeap(){

int h = floor(log2(this->counter))+1;

if (this->counter == 0){

return;

}

else {

static int k = 0;

for (int i=0; i<h; i++){

int n = pow(2,i);

cout<<string((h-i-1),' ');

for (int m=0; m<n; m++){

if (k<counter){

cout<<A[k].data<<" ";

//cout<<string(n-m-1, ' ')<<A[k].data<<string(n-m-1, ' ');

k++;

}

else return;

}

cout<<"\n";

}

}

}

void Heap::printHeapLinear(){

if (this->counter == 0){

return;

}

else {

int n=0;

while(n != counter){

cout<<A[n].data<<" ";

n++;

}

}

}

bool Heap::isMaxHeap(){

bool flag = true;

int h = floor(log2(this->counter))+1;

if (h == 0){

return flag;

}

else if (h == 1){

return flag;

}

else {

static int k = 0;

for (int i=0; i<h-1; i++){

for (int j=0; j<pow(2,i); j++){

if ((2\*k+1 < this->counter) && (A[k].data < A[2\*k+1].data)){

flag = false;

return flag;

}

if ((2\*k+2 < this->counter) && (A[k].data < A[2\*k+2].data)){

flag = false;

return flag;

}

k++;

}

}

return flag;

}

}

void Heap::deleteMax(){

if (this->counter == 0){

return;

}

else if (this->counter == 1){

A[0].data = 0;

this->counter--;

}

else {

Swap(A[0], A[counter-1]);

A[counter-1].data = 0;

reheapDown(0);

counter--;

}

}

void Heap::deleteHeapNode(int position){

if (this->counter == 0){

return;

}

else if (position >= this->counter){

return;

}

else if (position == 0){

A[0].data = 0;

this->counter--;

}

else if (position == this->counter-1){

A[position].data = 0;

this->counter--;

}

else {

Swap(A[position], A[counter-1]);

A[counter-1].data = 0;

reheapDown(position);

counter--;

}

}

// Check if an array is a Heap, function outside class

bool isMaxHeap(int arr[], int counter){

bool flag = true;

int h = floor(log2(counter))+1;

if (h == 0){

return flag;

}

else if (h == 1){

return flag;

}

else {

static int k = 0;

for (int i=0; i<h-1; i++){

for (int j=0; j<pow(2,i); j++){

if ((2\*k+1 < counter) && (arr[k] < arr[2\*k+1])){

flag = false;

return flag;

}

if ((2\*k+2 < counter) && (arr[k] < arr[2\*k+2])){

flag = false;

return flag;

}

k++;

}

}

return flag;

}

}

int main() {

// Test insertHeap()

Heap A;

int arr[20] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};

for (int i=0; i<10; i++){

A.insertHeap(arr[i]);

}

//Test printHeap

A.printHeapLinear();

cout<<"\n";

A.printHeap();

cout<<"\n";

// Test deleteMax()

A.deleteMax();

A.printHeapLinear();

cout<<"\n";

//Test deleteHeapNode()

A.deleteHeapNode(2);

A.printHeapLinear();

cout<<"\n";

// Test isMaxHeap()

cout<<"Array arr is a max-Heap: " << boolalpha << isMaxHeap(arr, 10) << "\n";

cout<<"Heap A is max-Heap: " << boolalpha << A.isMaxHeap();

}

**Results:**

**1. insertHeap() & printHeap()**

int main() {

// Test insertHeap()

Heap A;

int arr[20] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};

for (int i=0; i<10; i++){

A.insertHeap(arr[i]);

}

//Test printHeap

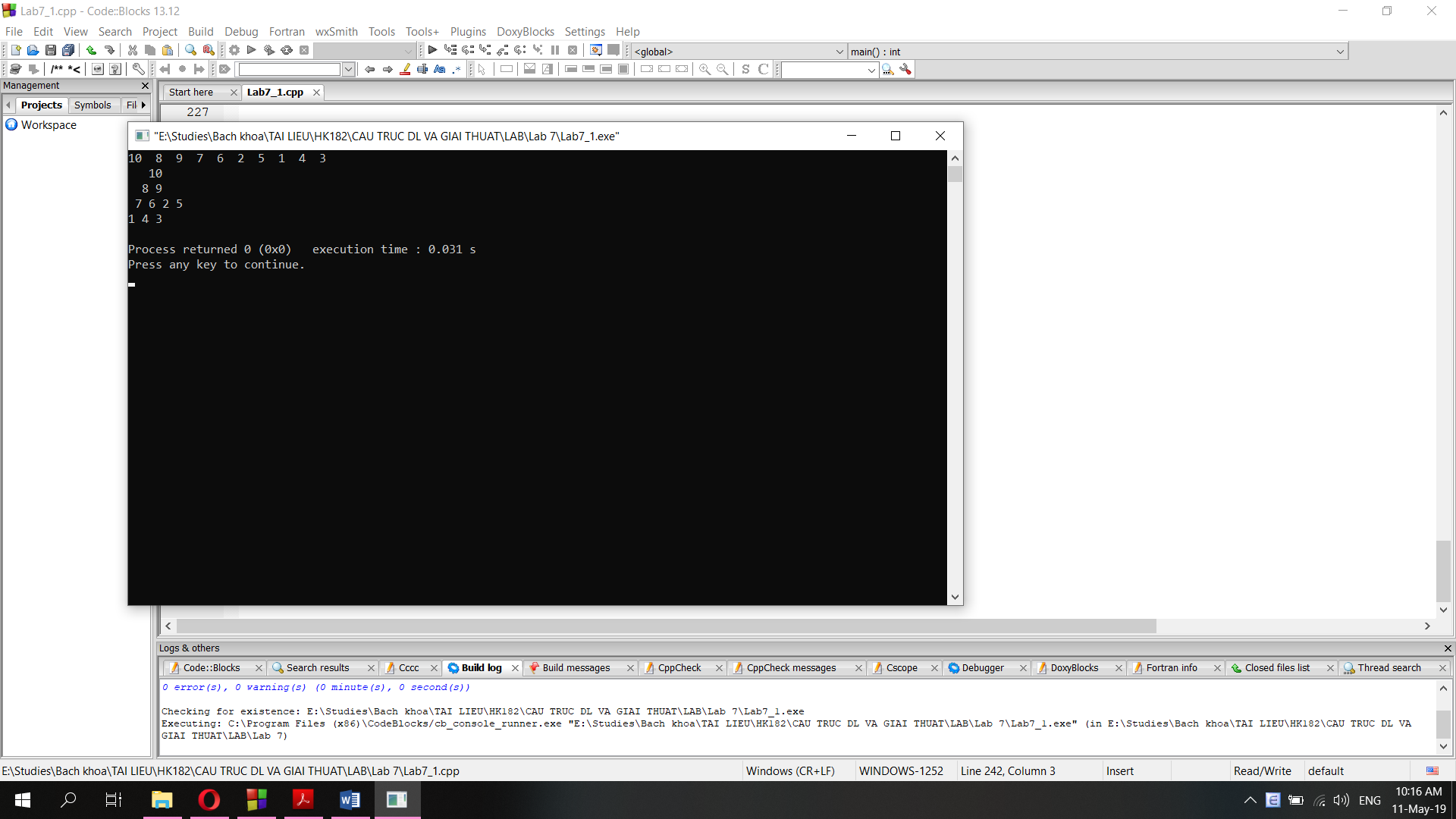
A.printHeapLinear();

cout<<"\n";

A.printHeap();

cout<<"\n";

}



**2. deleteMax() (delete the root), before and after deleting:**

int main() {

// Test insertHeap()

Heap A;

int arr[20] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};

for (int i=0; i<10; i++){

A.insertHeap(arr[i]);

}

//Test printHeap

A.printHeapLinear();

cout<<"\n";

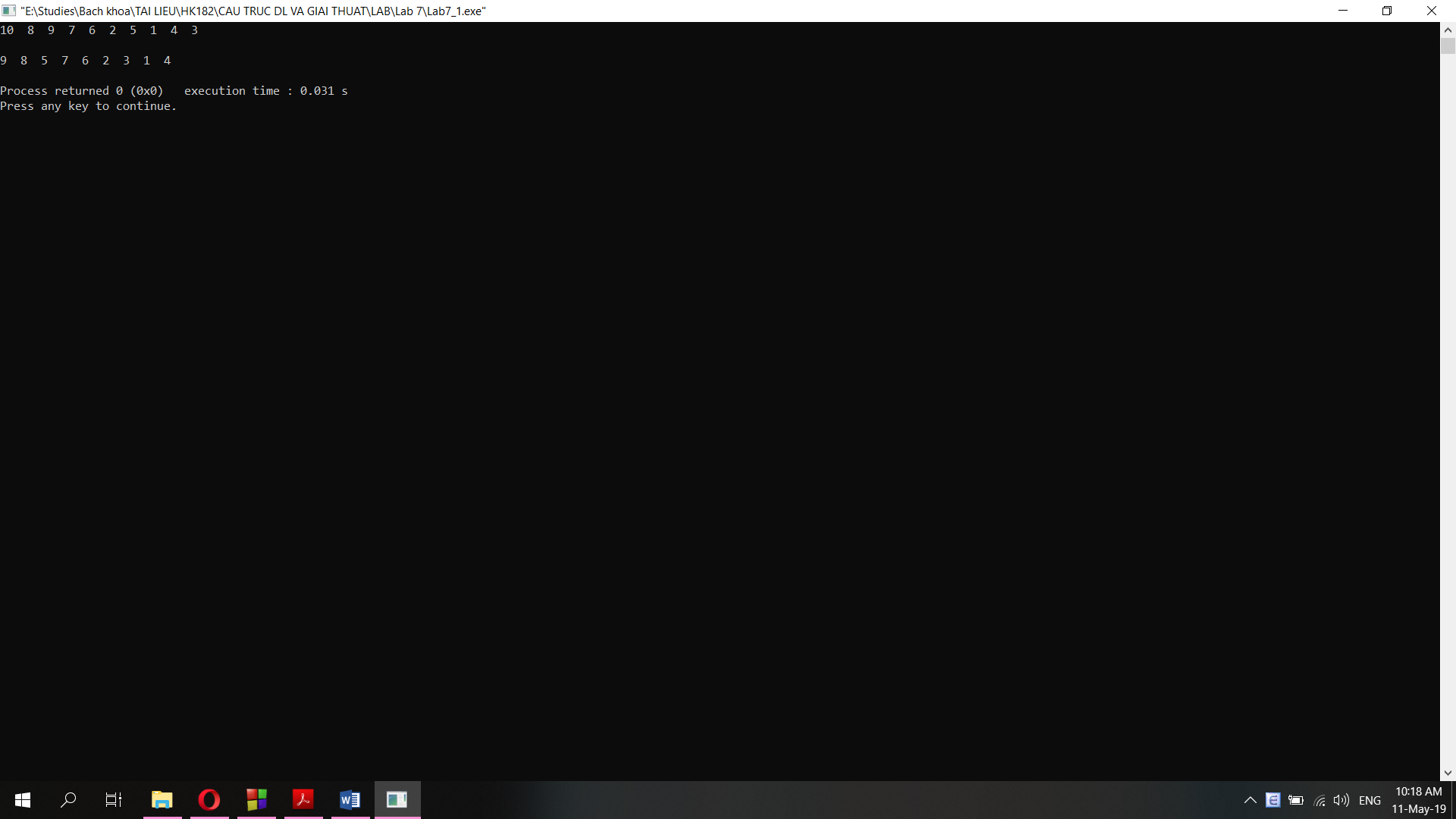
// Test deleteMax()

A.deleteMax();

A.printHeapLinear();

cout<<"\n";

}



**3. deleteHeapNode(): delete a specific node of the max-Heap**

int main() {

// Test insertHeap()

Heap A;

int arr[20] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};

for (int i=0; i<10; i++){

A.insertHeap(arr[i]);

}

//Test printHeap

A.printHeapLinear();

cout<<"\n";

//A.printHeap();

cout<<"\n";

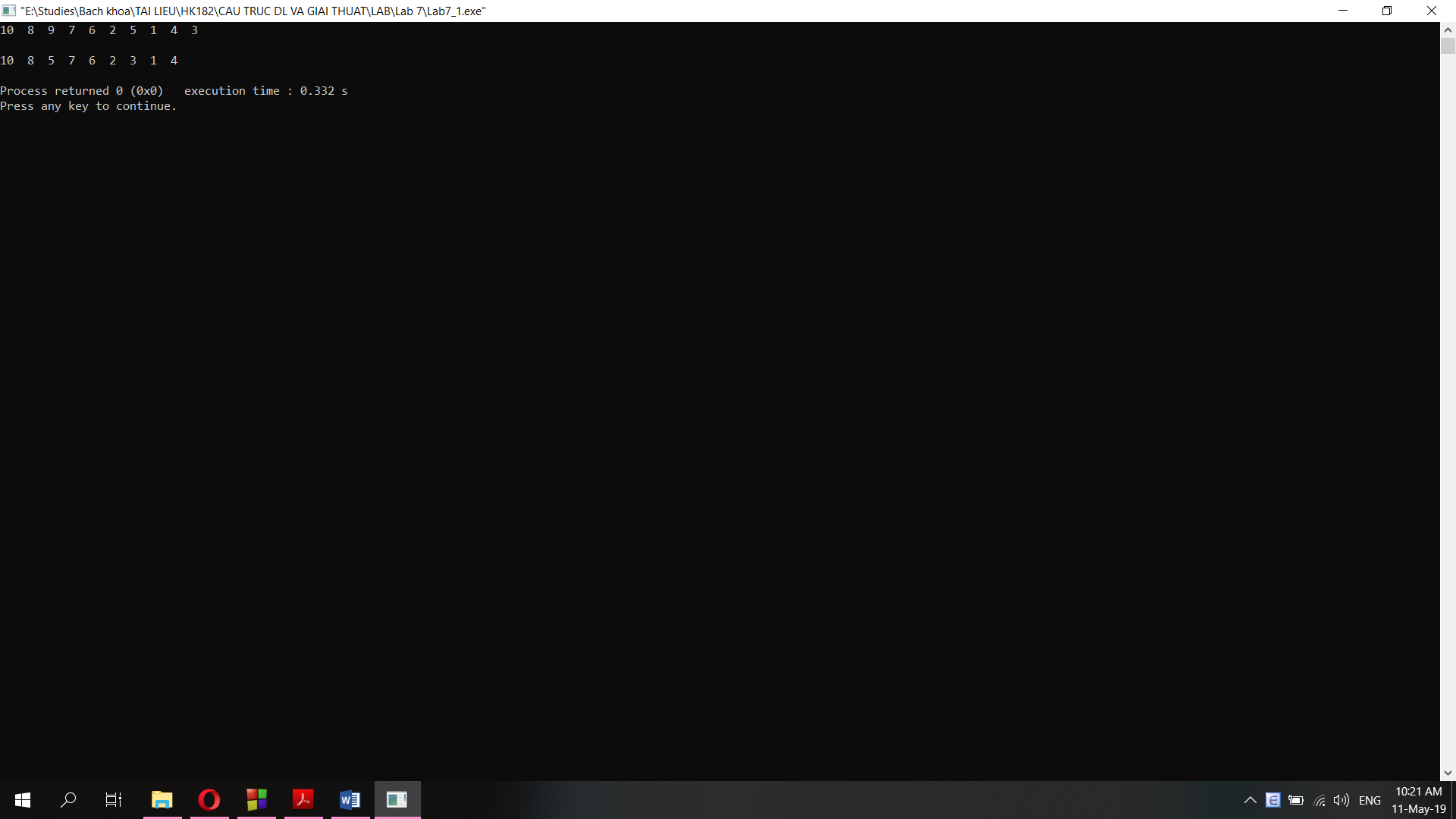
//Test deleteHeapNode()

A.deleteHeapNode(2);

A.printHeapLinear();

cout<<"\n";

}



**4. isMaxHeap(): check if a heap is max-Heap**

int main() {

// Test insertHeap()

Heap A;

int arr[20] = {1, 2, 3, 4, 5, 10, 9, 8, 7, 6};

for (int i=0; i<10; i++){

A.insertHeap(arr[i]);

}

// Test isMaxHeap()

cout<<"Array arr is a max-Heap: " << boolalpha << isMaxHeap(arr, 10) << "\n";

cout<<"Heap A is max-Heap: " << boolalpha << A.isMaxHeap();

}

