**Họ và Tên: Trần Thị Ngọc Diệp**

**MSSV: 1827005**

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

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

**BÀI KIỂM TRA THỰC HÀNH 2**

**Question 1:**

#include <iostream>

#include <bits/stdc++.h>

#include <cstring>

#include <sstream>

#include <stdlib.h>

using namespace std;

struct Treenode {

char data;

Treenode \*left;

Treenode \*right;

};

// Implement the constructor of Treenode

Treenode \*newNode(char data) {

Treenode \*temp = new Treenode;

temp->left = temp->right = NULL;

temp->data = data;

return temp;

}

// Check if a character is operand

bool isOperator(char c) {

if (c == '+' || c == '-' || c == '\*' || c == '/')

{

return true;

}

return false;

}

// Implement the CreateExampleTree() function

Treenode \*CreateExampleTree(string s) {

stack<Treenode \*> temp;

Treenode \*tree1, \*tree2, \*tree3;

// Traverse through the string

for (int i = 0; i < s.length(); i++)

{

// if the char is operator, add it to stack

if (isOperator(s[i]) == false)

{

tree1 = newNode(s[i]);

temp.push(tree1);

}

// else: pop operator out of stack

else

{

tree1 = newNode(s[i]);

tree2 = temp.top();

temp.pop();

tree3 = temp.top();

temp.pop();

tree1->right = tree2;

tree1->left = tree3;

temp.push(tree1);

}

}

tree1 = temp.top();

temp.pop();

return tree1;

}

void printTree(Treenode \*tree) {

if (tree == NULL)

return;

printTree(tree->left);

cout << tree->data << " ";

printTree(tree->right);

return;

}

// Count number of nodes in the tree

int countNode(Treenode \*root) {

int count = 1;

if (root->left != NULL)

count += countNode(root->left);

if (root->right != NULL)

count += countNode(root->right);

return count;

}

// Calculate the value of the expression tree

float computeTree(Treenode \*root) {

if (root == NULL)

{

return 0;

}

if (root->left == NULL && root->right == NULL)

{

stringstream ss;

string stringData;

ss << root->data;

ss >> stringData;

return atof(stringData.c\_str());

}

float leftValue = computeTree(root->left);

float rightValue = computeTree(root->right);

if (root->data == '+')

{

return leftValue + rightValue;

}

if (root->data == '-')

{

return leftValue - rightValue;

}

if (root->data == '\*')

{

return leftValue \* rightValue;

}

if (root->data == '/')

{

return leftValue / rightValue;

}

}

int main() {

// Question 1a

// The print function only for test, it doesn't print parentheses

Treenode \*root = NULL;

root = CreateExampleTree("437\*+534+/-6+");

cout << "The expression tree: ";

printTree(root);

cout<<"\n";

// Question 1c

cout << "Number of nodes: " << countNode(root);

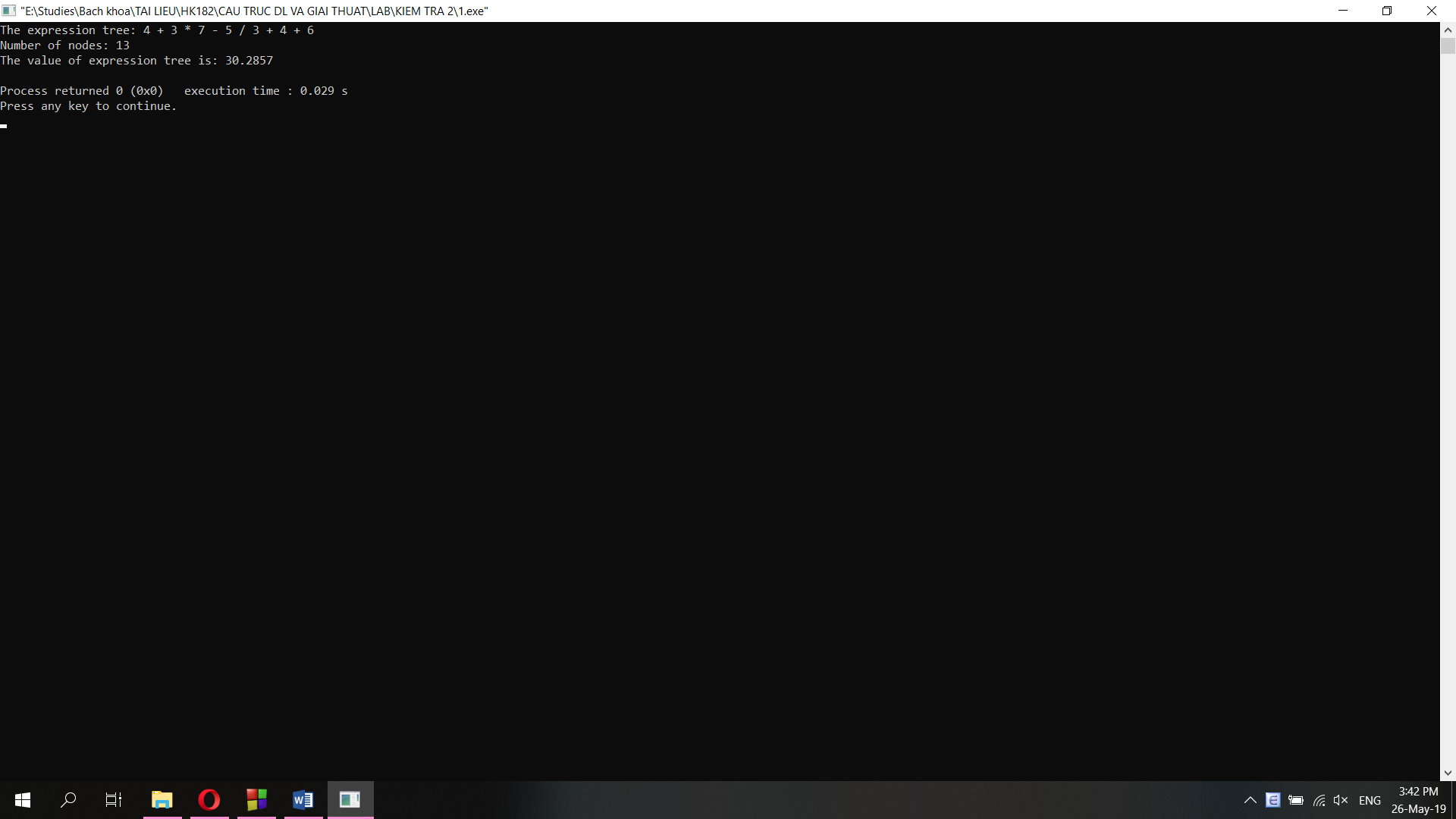
cout<<"\n";

// Question 1f

cout << "The value of expression tree is: " << computeTree(root);

cout<<"\n";

}



**Question 2:**

#include <iostream>

#include <math.h>

#include <cstdlib>

#include <ctime>

#include <algorithm>

#include <stdbool.h>

using namespace std;

void printArr(int \*arr, int arr\_size);

// Create a random array with n parts, each part contains 4 elements

int\* CreateRandomArray (int \*arr, int n, int a, int b)

{

int arr\_size = n;

arr = new int[arr\_size];

srand(time(0));

for (int i=0; i<arr\_size; i++)

{

bool dup = false; // a flag to check if there are duplicate elements

int temp = (rand() % (a-b+1));

for (int j=0; j<i; j++)

{

if (temp == arr[j])

dup = true;

}

// if there are no duplicate elements, add it to array

if (dup == false)

{

arr[i] = temp;

}

// else: decrease i then return to loop

else

{

i--;

}

}

return arr;

}

// Question 2b: Shell Sort Implement

void ShellSort(int \*arr, int arr\_size)

{

int hold;

int incre = arr\_size/2;

int walker;

while (incre != 0)

{

for (int curr = incre; curr <= arr\_size-1; curr++)

{

// Hold arr[curr] value until we find its proper position

hold = arr[curr];

walker = curr - incre;

while (walker >= 0 && hold < arr[walker])

{

arr[walker + incre] = arr[walker];

walker = ( walker - incre );

}

// Insert hold in proper position

arr[walker + incre] = hold;

}

// print each step

printArr(arr, arr\_size);

// Move to next segment dividing

incre = incre / 2;

}

}

// Print the array

void printArr(int \*arr, int arr\_size)

{

for (int i=0; i<arr\_size; i++)

{

if (arr[i]<10)

{

cout<<" "<<arr[i];

}

else

{

cout<<" "<<arr[i];

}

if ((i+1)%10 == 0) cout<<"\n";

}

cout<<"\n";

}

// Question 2c

// Implement a sort method first, here I use insertion sort

void InsertionSort(int\* arr, int n)

{

long count = n, current;

current = 1;

while(current < count)

{

long temp = arr[current];

long walker = current - 1;

while(walker >= 0 && temp < arr[walker])

{

arr[walker + 1] = arr[walker];

walker--;

}

arr[walker+1] = temp;

current++;

}

}

// Sort the array in increasingly order then swap elements

void SortArr(int \*arr, int n)

{

InsertionSort(arr, n);

int walker1;

for (walker1 = 1; walker1 <n-2; walker1++)

{

int walker2 = walker1 + 1;

swap(arr[walker1], arr[walker2]);

walker1 = walker1 + 3;

}

}

// Implement Bubble sort for 2d

void bubbleSort (int\* arr, int n)

{

int temp;

// Each iteration is one sort pass

for (int current = 0, sorted = 0; current < n && !sorted; current++) // sorted flag

{

for (int walker = n-1, sorted = 1; walker > current; walker--)

{

if (arr[walker] < arr[walker-1])

// Any exchange means list is not sorted

{

sorted = 0;

swap(arr[walker], arr[walker-1]);

}

}

}

}

// Question 2d: O(n2)

// Use bubble sort so the complexity is O(n2)

// After sorting, get the element at index n-k

int SelectKth1(int \*arr, int n, int k)

{

bubbleSort(arr, n);

printArr(arr, n);

return arr[n-k];

}

// Question 2e: O(nlogn)

// Use simple sort so the complexity is O(nlogn)

// After sorting, get the element at index n-k

int SelectKth2(int \*arr, int n, int k)

{

sort(arr, arr+n);

printArr(arr, n);

return arr[n-k];

}

int main()

{

// Run each part once at a time, part 2a is always run

// 2a: Check CreateRandomArray() function

int \*pt;

int k;

int n = 40;

pt = CreateRandomArray(pt, n, 1, 100);

printArr(pt, n);

/\*

// 2b: Shell sort

ShellSort(pt, n);

printArr(pt, n);

\*/

/\*

// 2c: SortArr() function

SortArr(pt, n);

printArr(pt, n);

\*/

/\*

// 2d: SelectKth1() function

cout<<"Enter K: ";

cin>>k;

int i = SelectKth1(pt, n, k);

cout<<"The Kth largest element is (Complexity = O(n2)): "<<i;

\*/

/\*

// 2e: SelectKth2() function

cout<<"Enter K: ";

cin>>k;

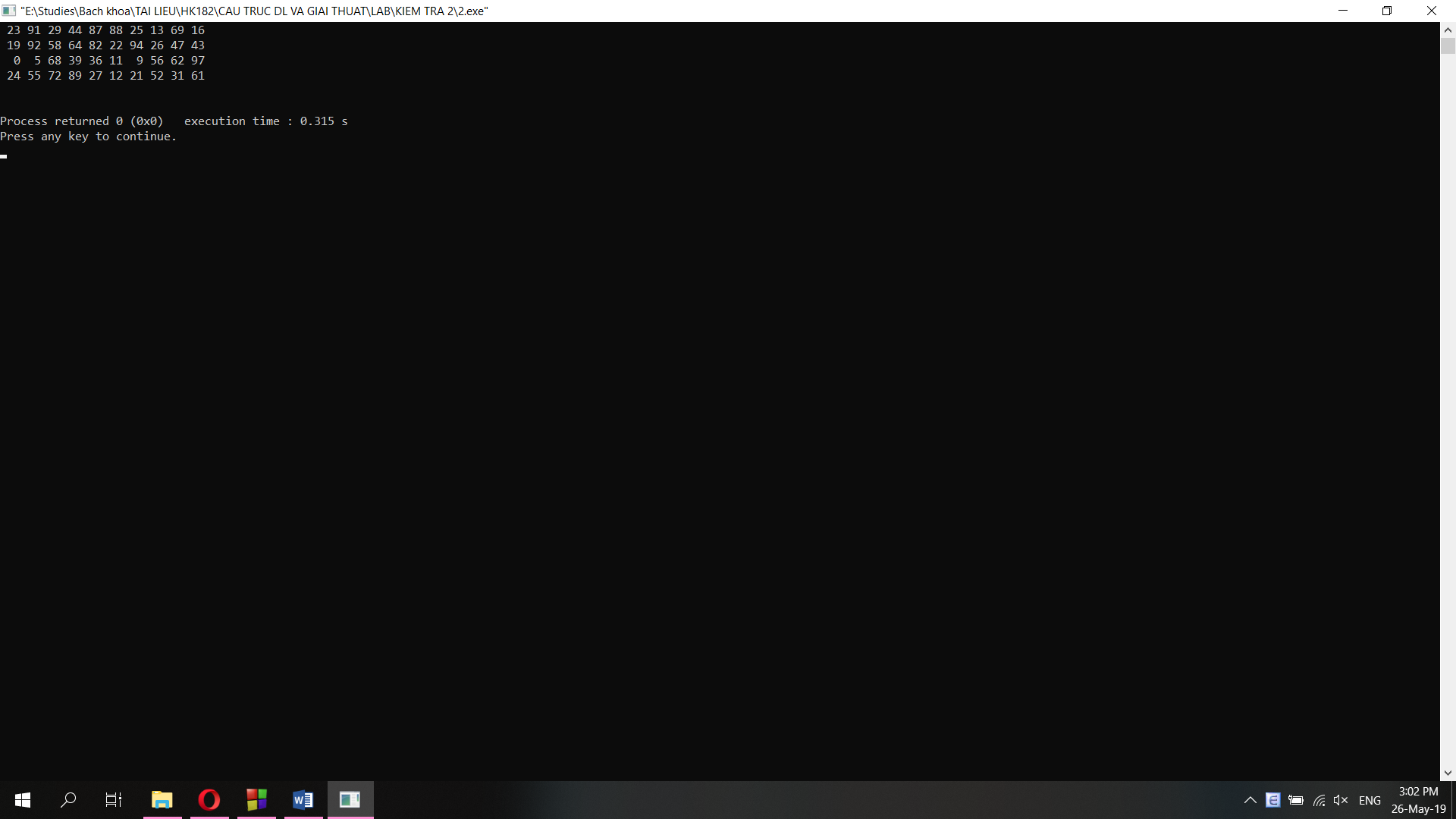
int i = SelectKth2(pt, n, k);

cout<<"The Kth largest element is (Complexity = O(nlogn)): "<<i;

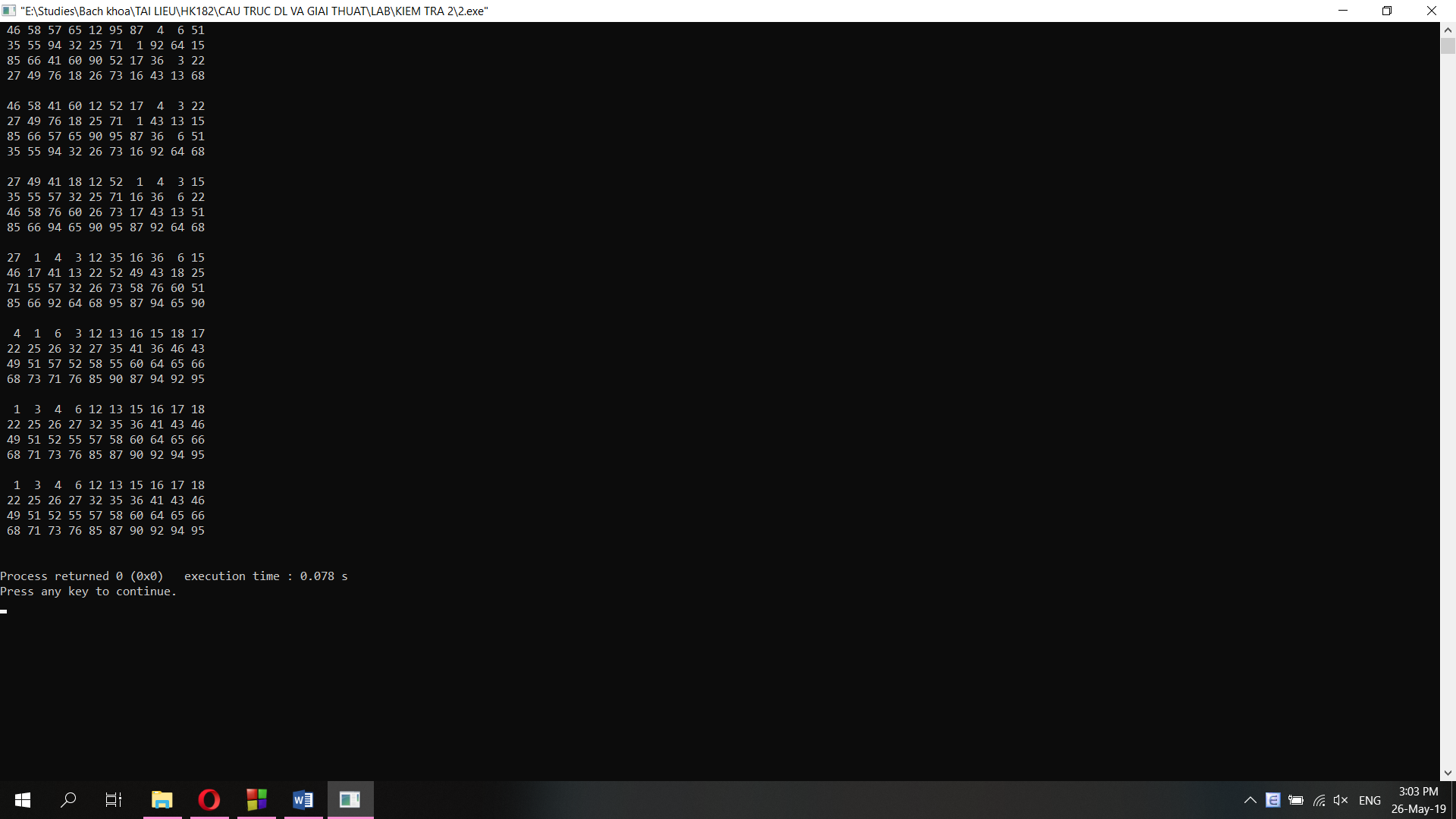
\*/}

Result:

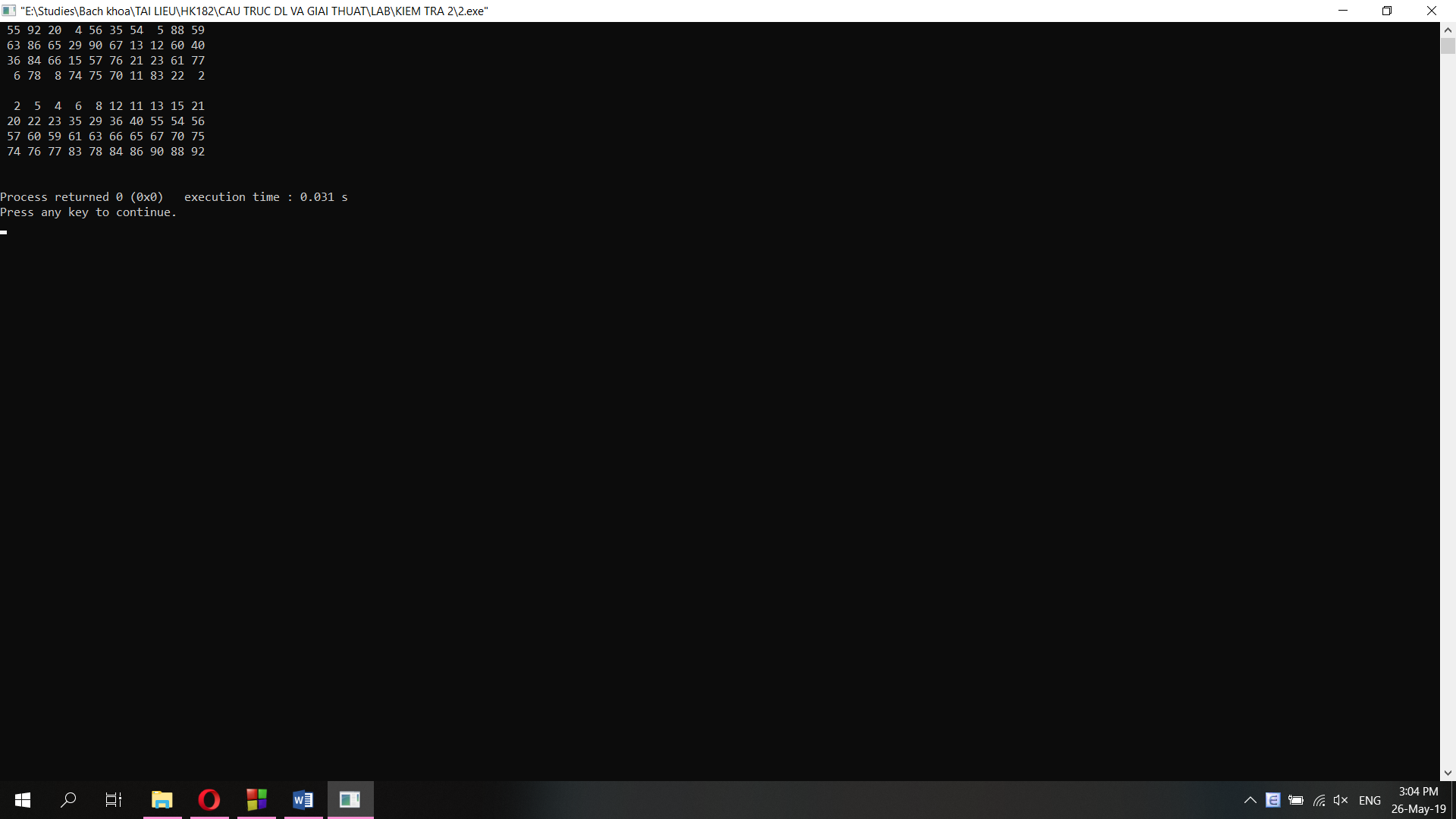
a.



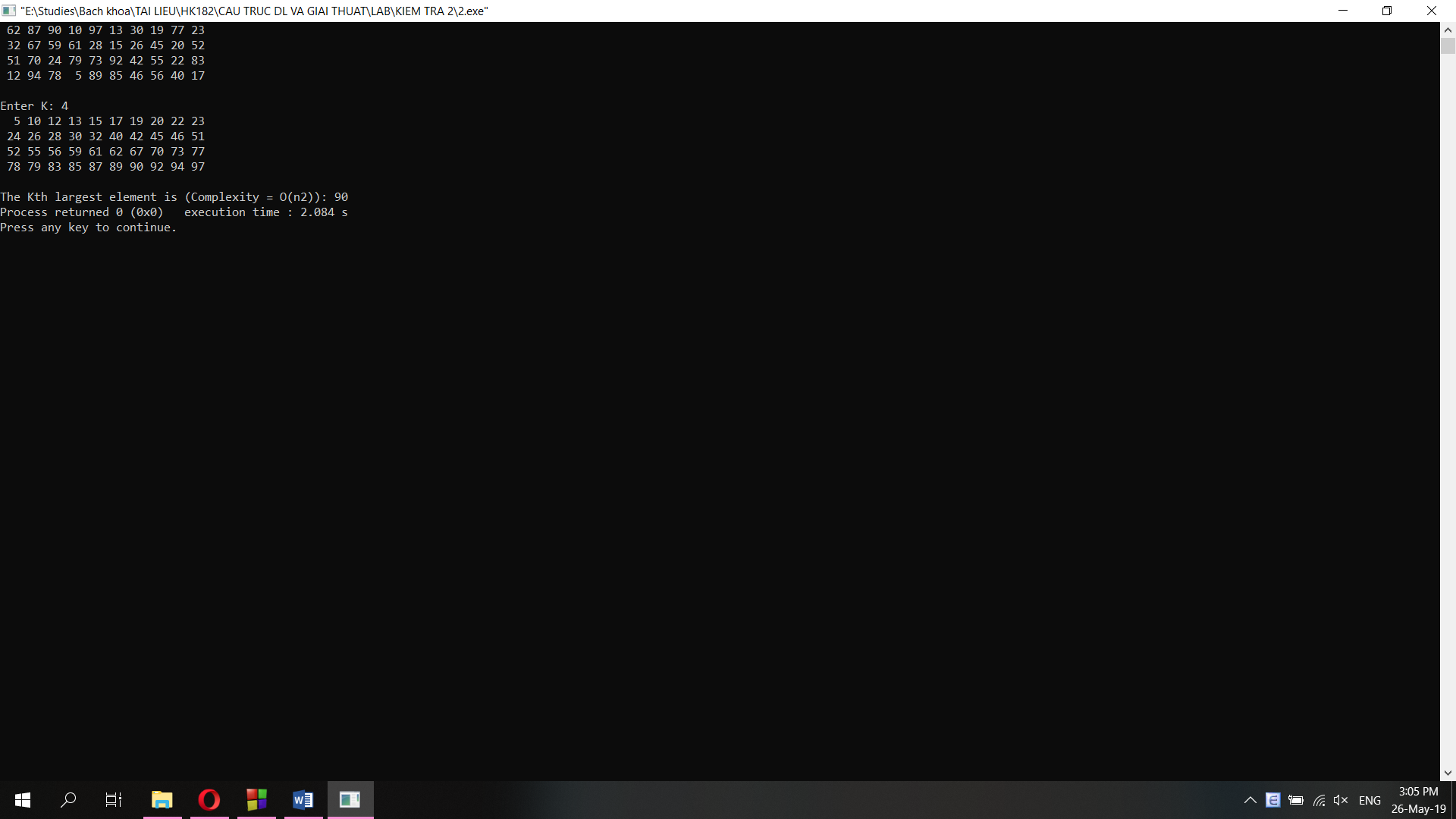
b.



c.



d.



e.

