**Họ và Tên: Trần Thị Ngọc Diệp**

**MSSV: 1827005**

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

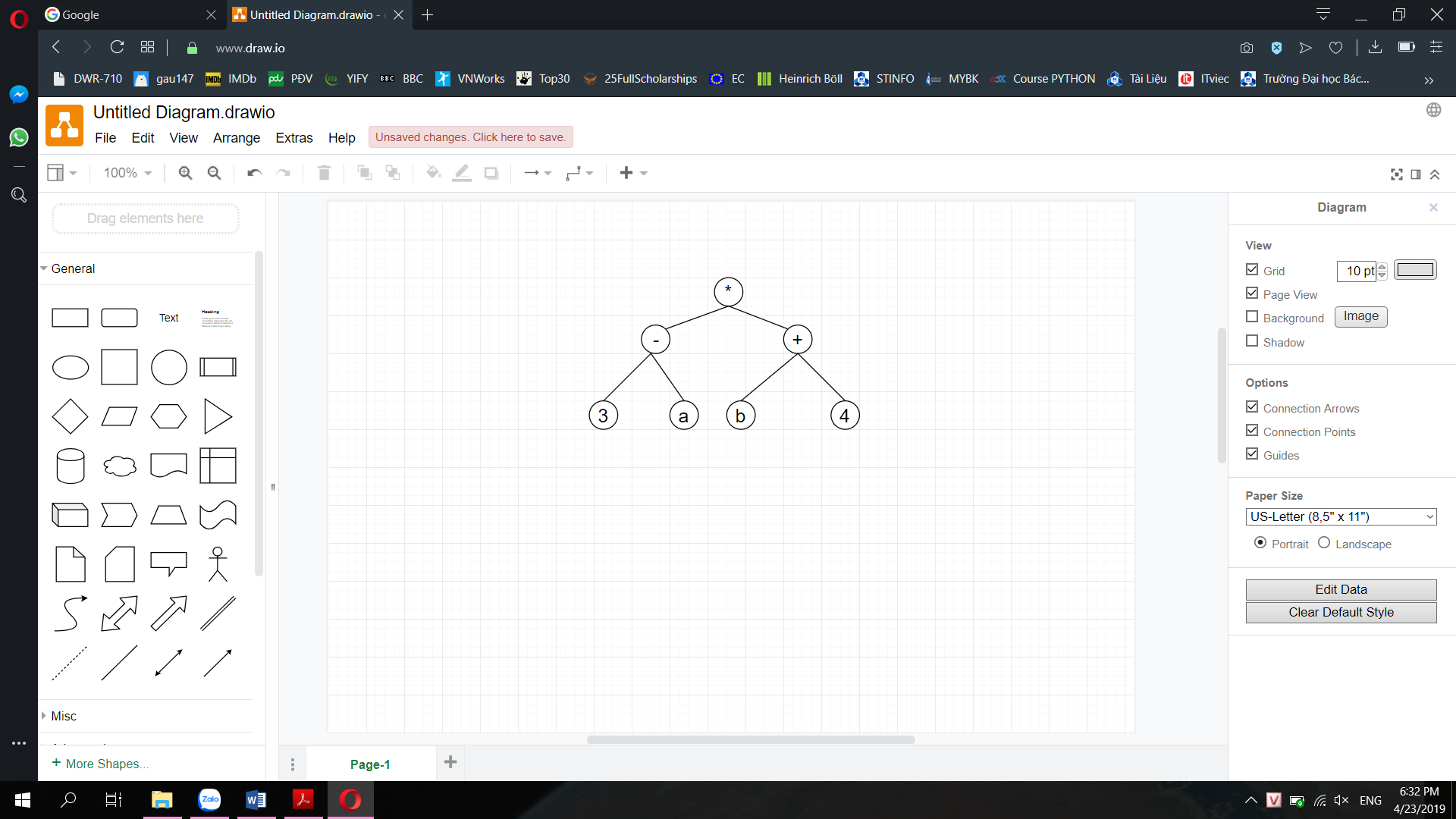
**(TUT 5 & LAB 5)**

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

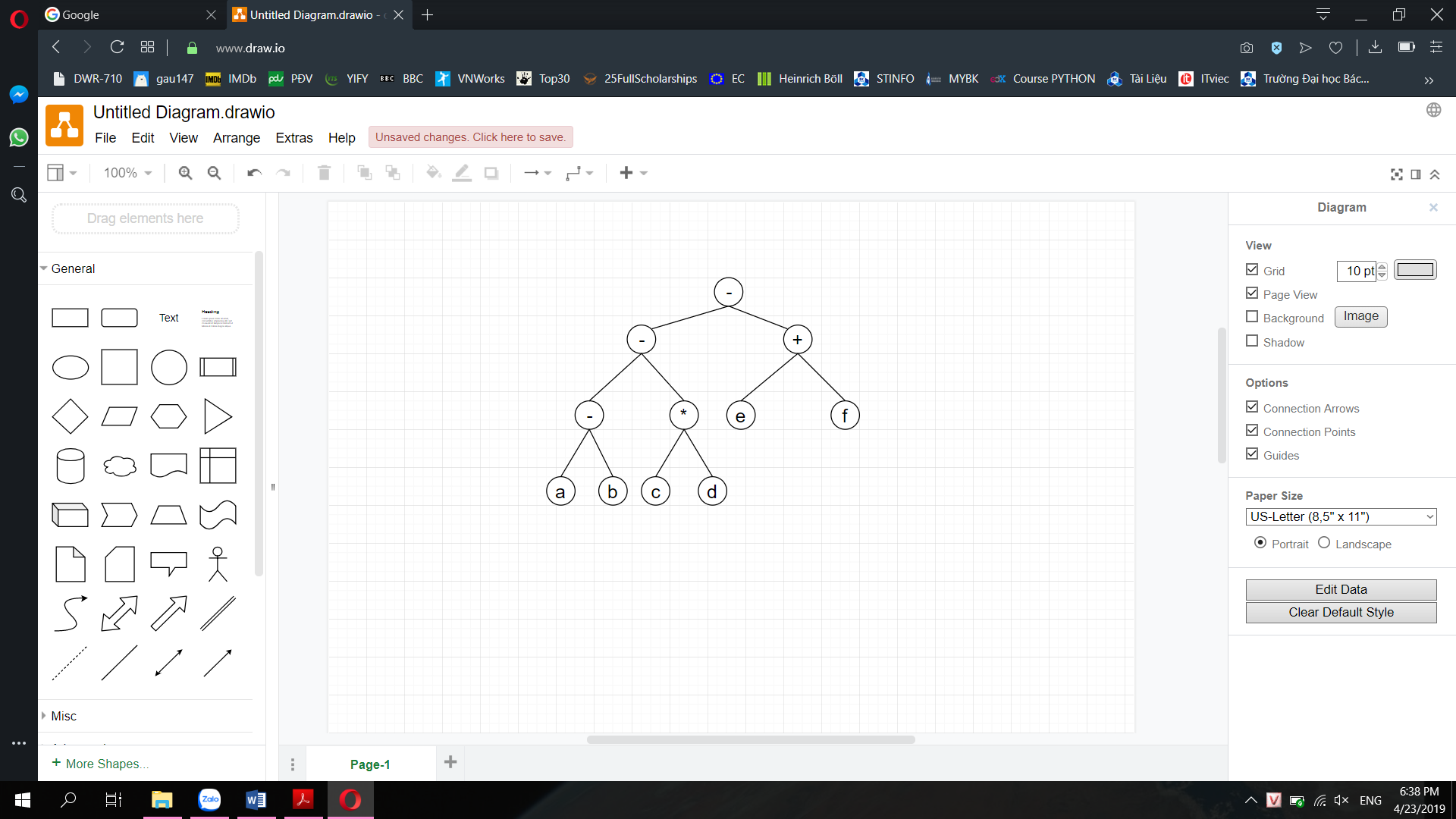
**TUT 5**

**Question 1: Expression Trees of the following expressions:**

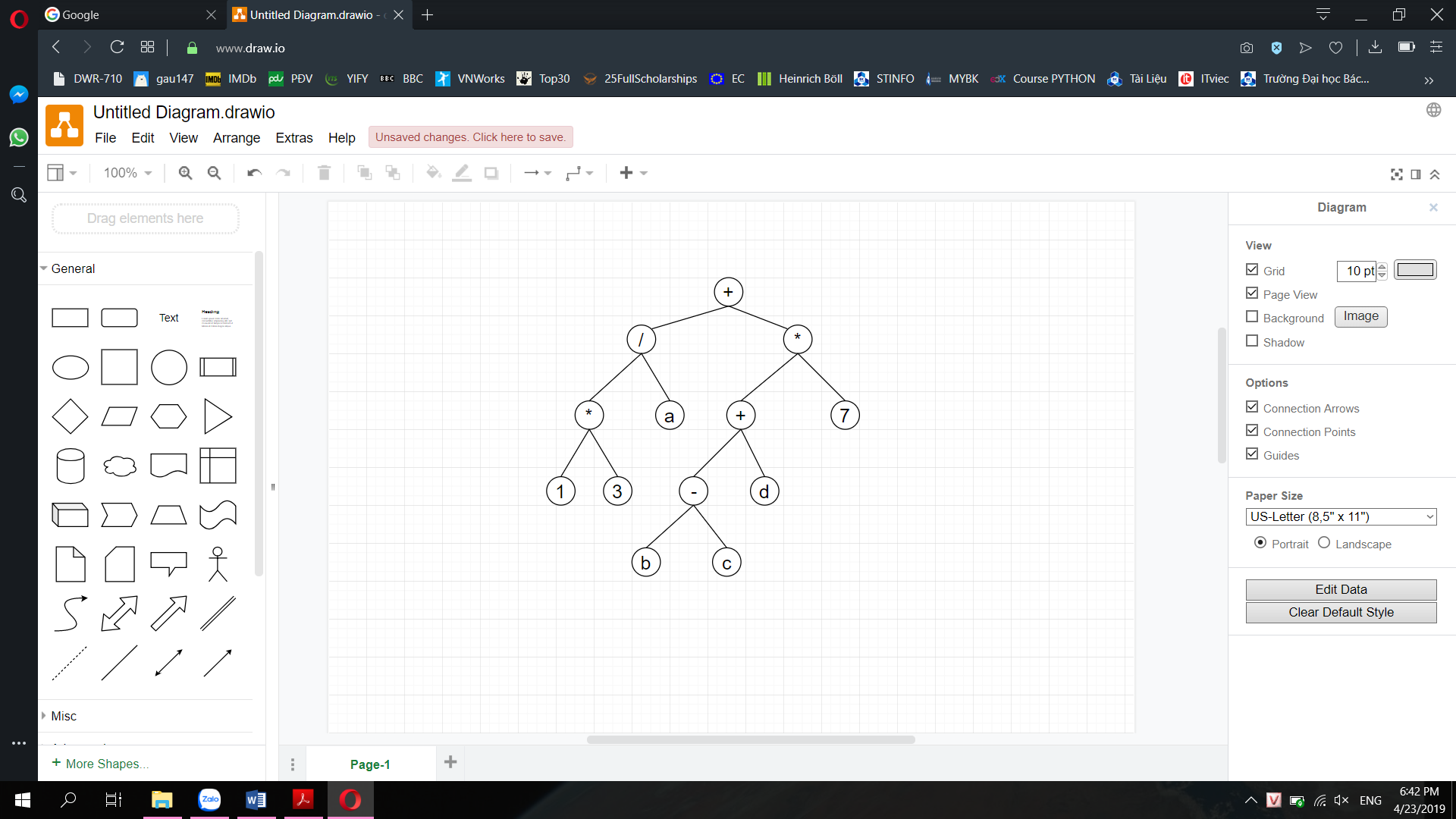
1. **(3 – a) \* (b + 4)**



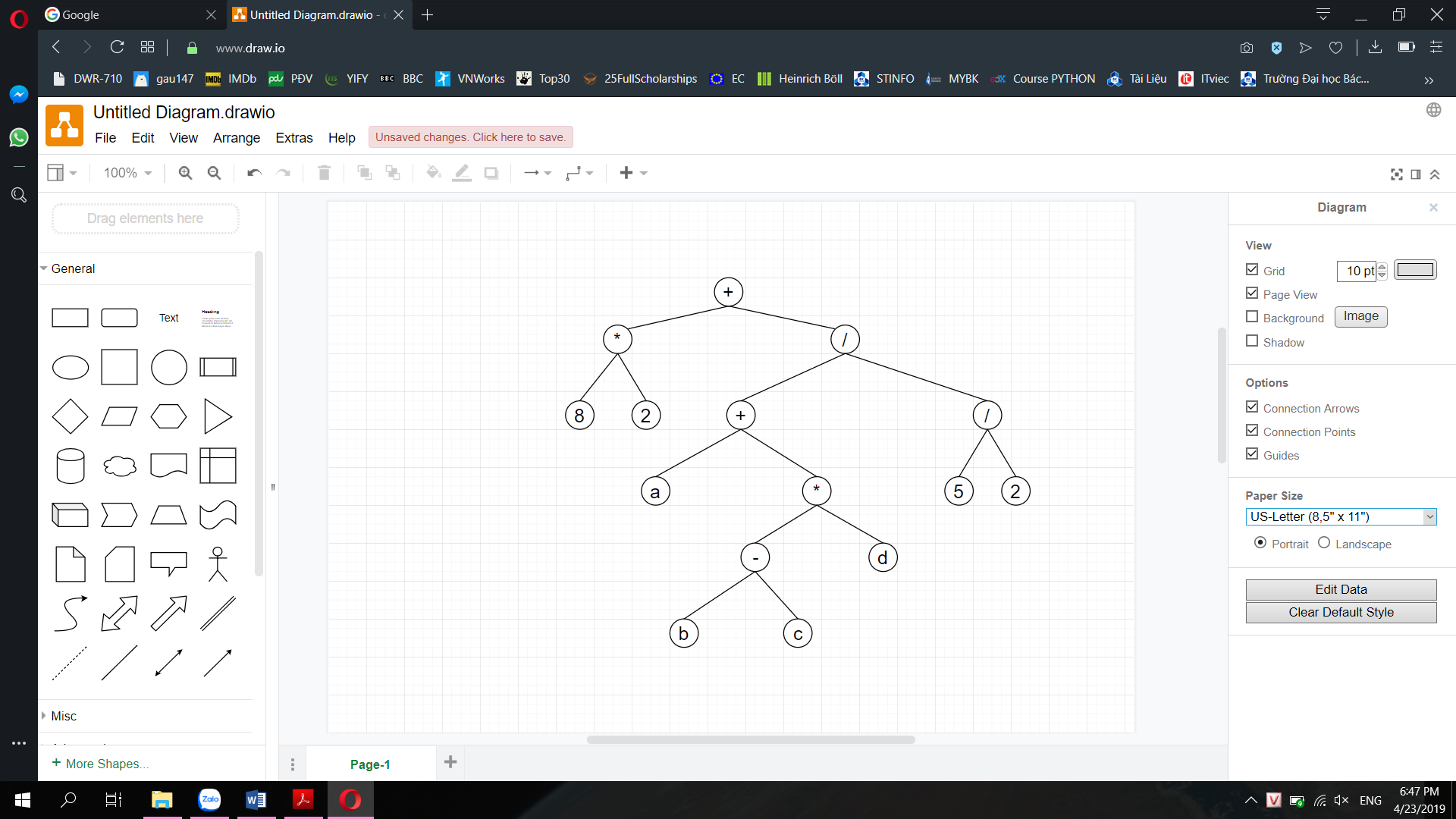
1. **a – b – c \* d – e – f**



1. **1 \* 3 / a + (b – c + d) \* 7**



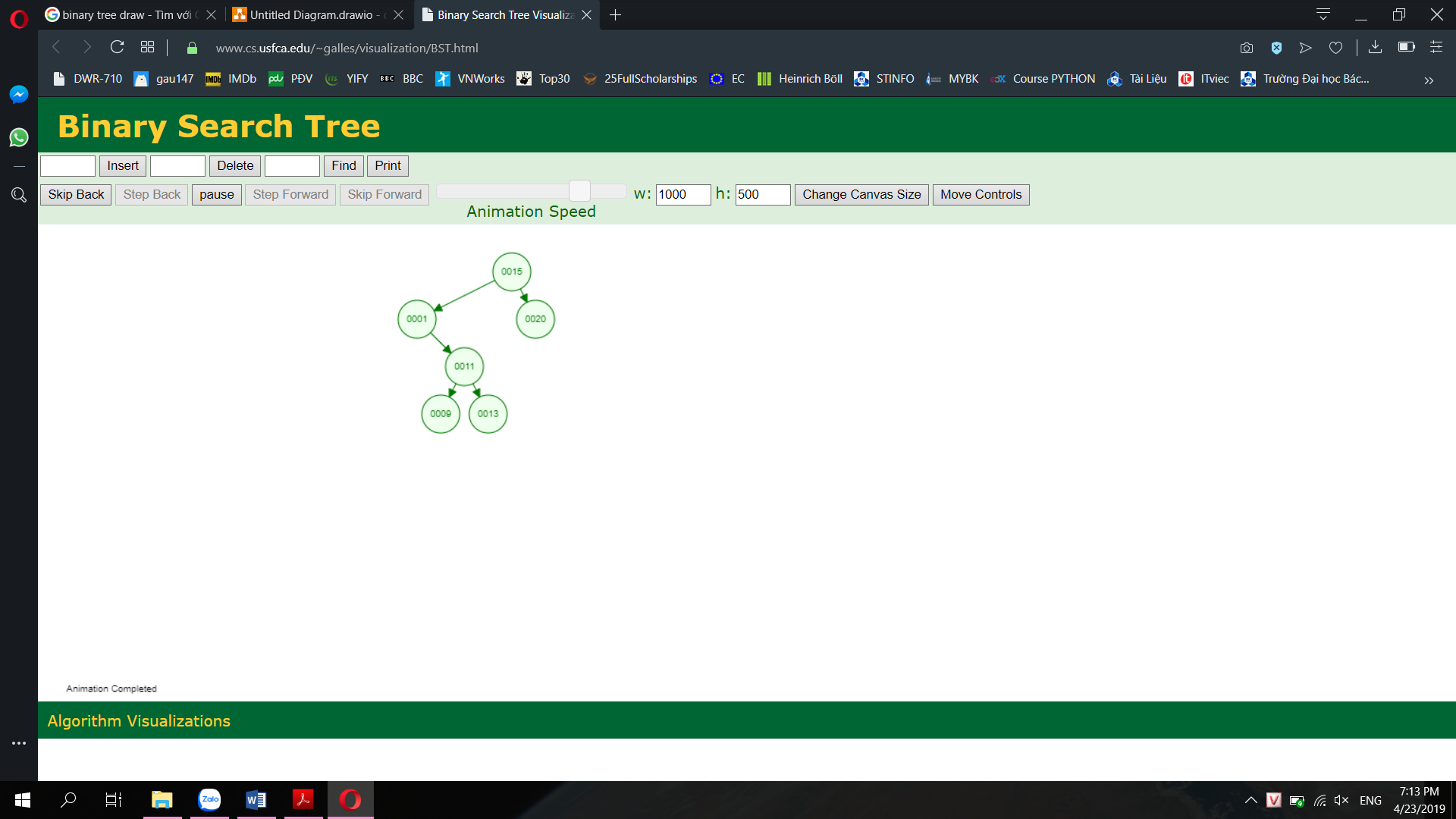
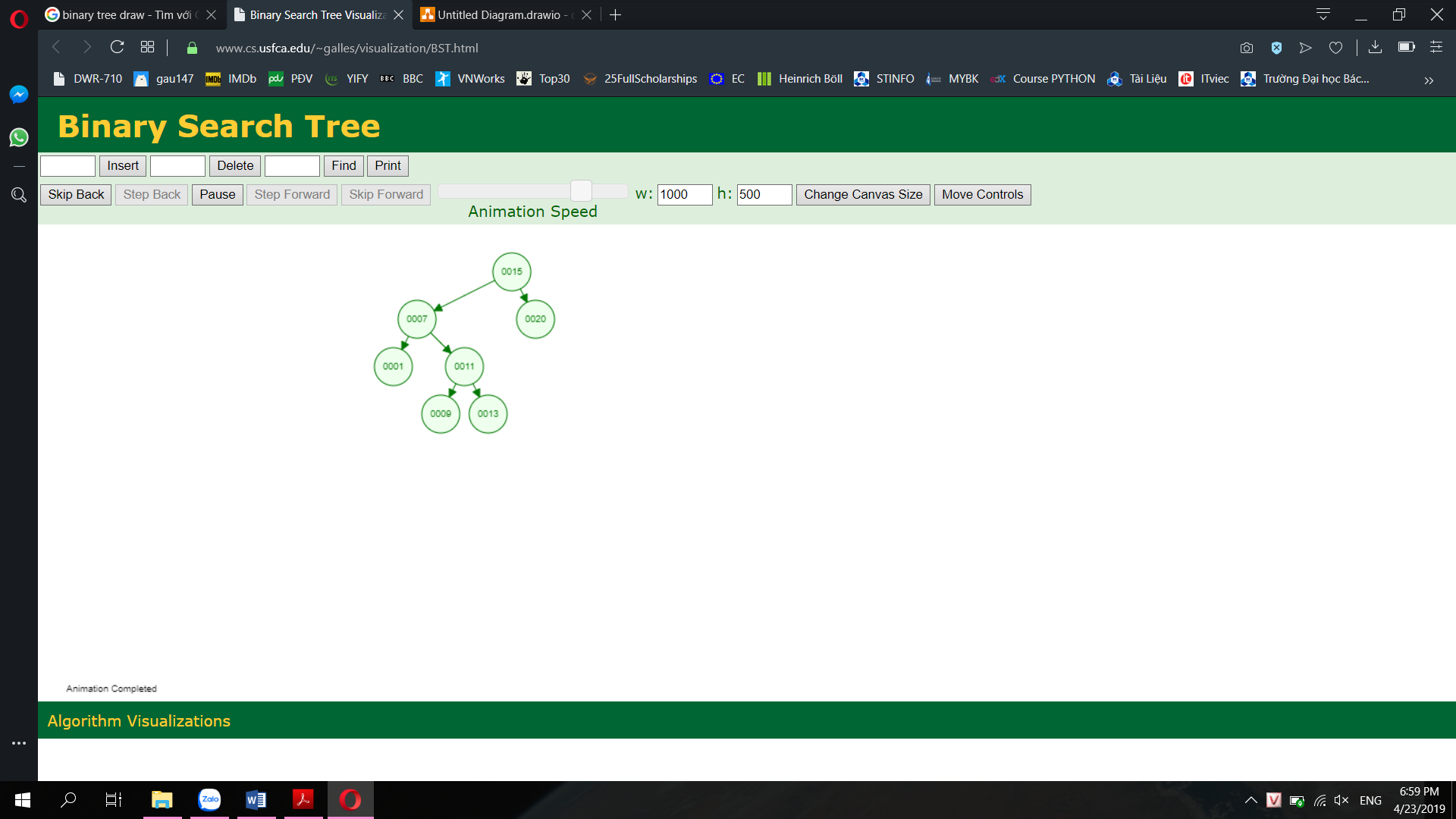
1. **(8 \* 2) + (a + (b – c) \* d) / (5 / 2)**



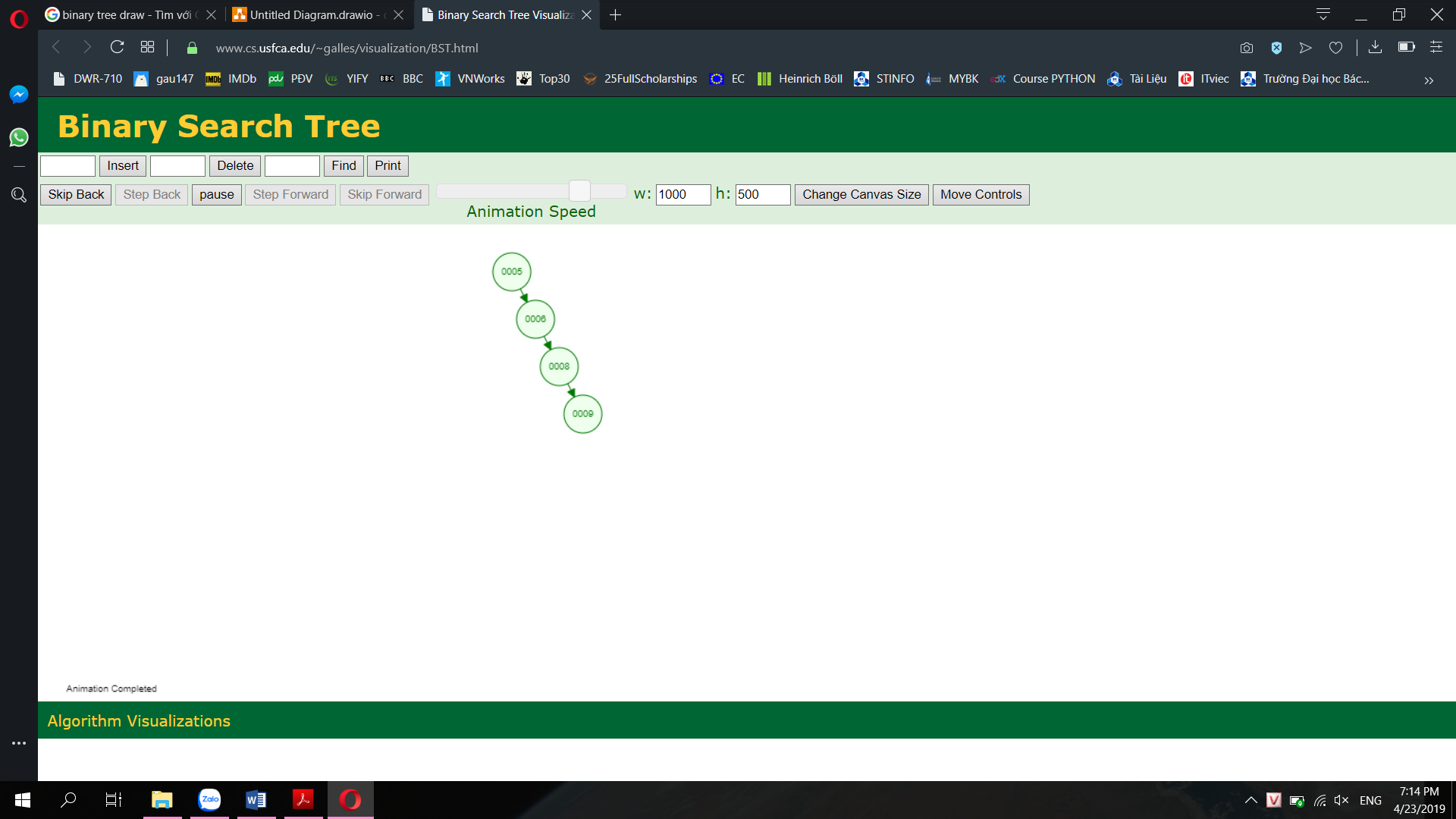
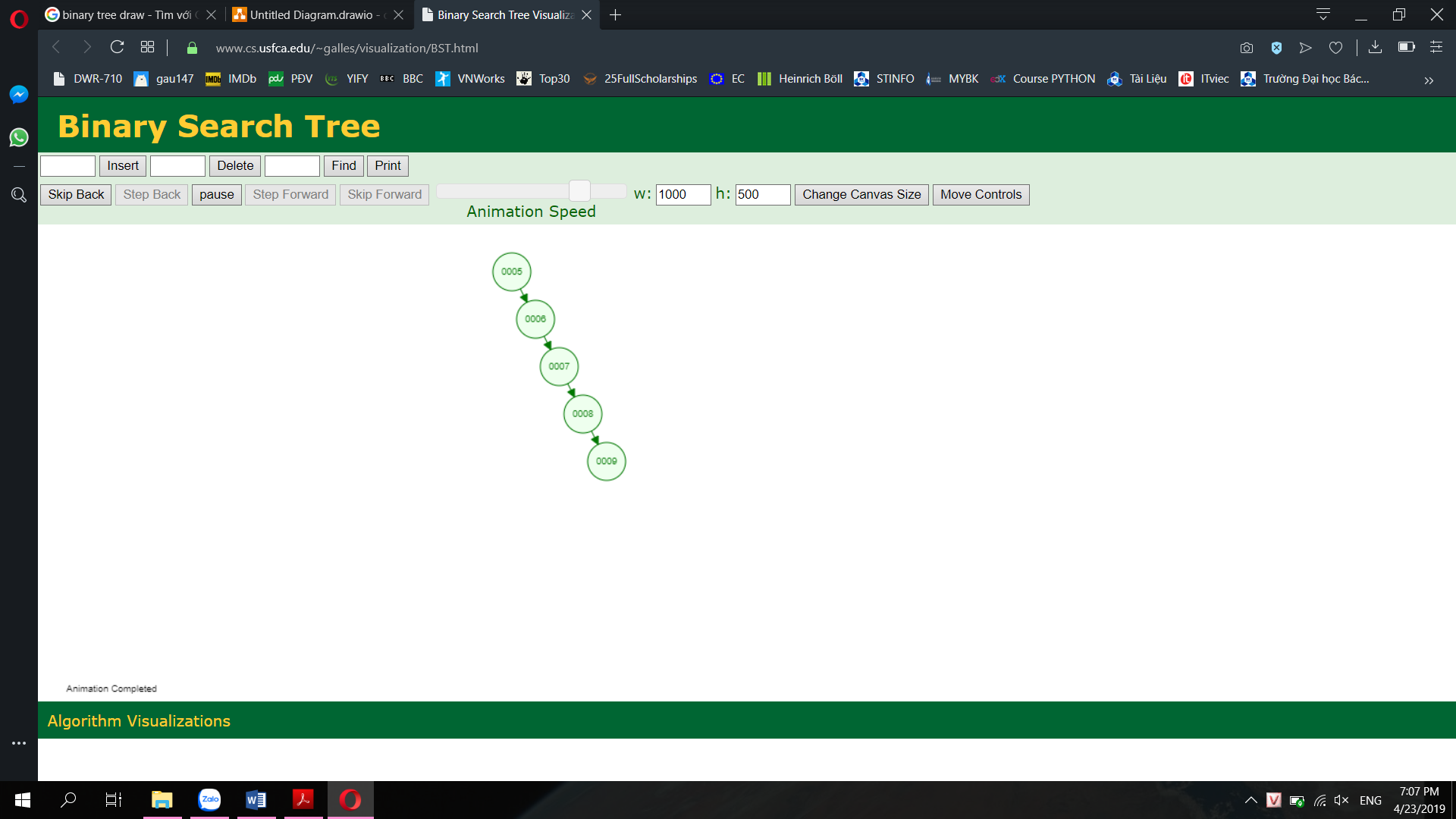
**Cả 4 cây đều là Cây Hoàn Chỉnh vì mỗi node đều có 0 hoặc 2 con.**

**Question 2: BST before and after remove node 7**

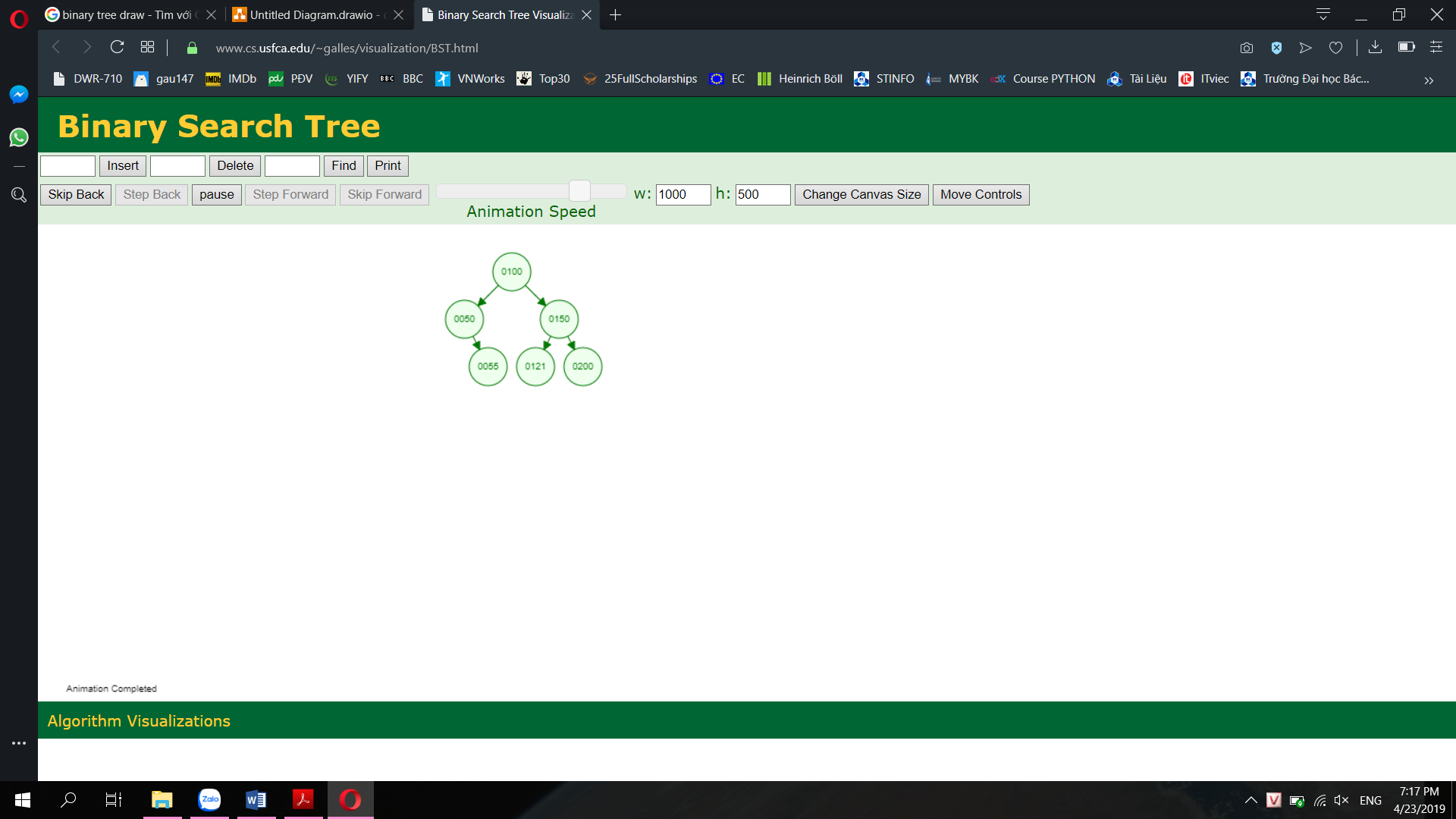
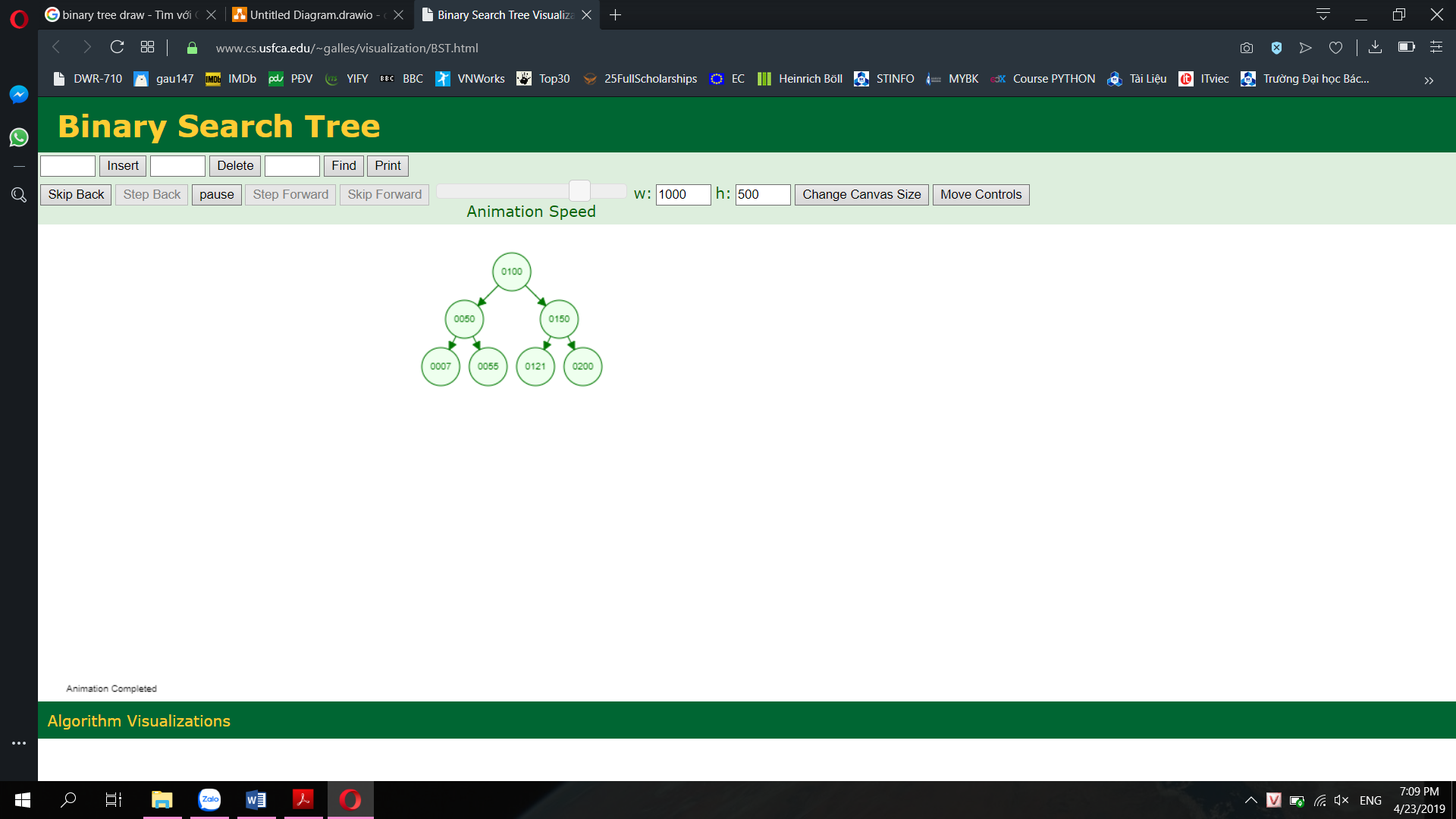
1. **15, 7, 1, 11, 9, 13, 20**



1. **5, 6, 7, 8, 9**



1. **100, 50, 150, 7, 55, 121, 200**



**Question 3:**

#include<iostream>

#include<sstream>

#define COUNT 10

using namespace std;

class treeNode {

public:

int data;

treeNode\* left = NULL;

treeNode\* right = NULL;

treeNode(int data);

};

treeNode::treeNode(int data){

this->data = data;

this->left = NULL;

this->right = NULL;

}

void print2D(treeNode \*root, int space){

if (root == NULL){

return;

}

space += COUNT;

//Call a recursive function to print each node on the right

print2D(root->right, space);

cout<<endl;

for (int i = COUNT; i < space; i++){

cout<<" ";

}

cout<<root->data<<"\n";

//Call a recursive function to print each node on the left

print2D(root->left, space);

}

void printTree(treeNode \*root){

print2D(root, 0);

}

//recursive function insert a new node to BST

treeNode \*recursiveInsert(treeNode\*subroot, treeNode\*newNode ) {

//return subroot when we reach the end

if (subroot==NULL){

subroot = newNode;

return subroot;

}

//traverse through the BST to find the location for new node

else {

if (subroot->data < newNode->data){

subroot->right = recursiveInsert(subroot->right,newNode);

}

else if (subroot->data > newNode->data) {

subroot->left = recursiveInsert(subroot->left, newNode);

}

//If the node is already exist, return

else {

return subroot;

}

}

}

//A function prints out the path from root to the node having searchedData

void printPath (treeNode\* subroot, int searchedData ) {

if (subroot == NULL){

return;

}

else{

while (subroot != NULL){

if (subroot->data > searchedData){

cout<<subroot->data<<" -> ";

subroot = subroot->left;

}

else if (subroot->data < searchedData){

cout<<subroot->data<<" -> ";

subroot = subroot->right;

}

else {

cout<<subroot->data<<"\n";

return;

}

}

}

}

//A function that print out all leaves of the tree via Breadth First Traverse

void printLeavesBFT (treeNode\* root) {

if (root == NULL){

return;

}

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

cout<<root->data<<" ";

}

else if (root->left == NULL && root->right != NULL){

printLeavesBFT(root->right);

}

else if (root->left != NULL && root->right == NULL){

printLeavesBFT(root->left);

}

else {

printLeavesBFT(root->left);

printLeavesBFT(root->right);

}

}

//A function that print out all leaves of the tree via LNR

void printLeavesLNR (treeNode\* subroot) {

// YOUR CODE HERE

}

//A function that print out all leaves of the tree via NLR

void printLeavesNLR (treeNode\* subroot) {

// YOUR CODE HERE

}

int main(){

treeNode \*Head, \*Temp1, \*Temp2, \*left, \*right;

//Initialize the BTS

Temp1 = new treeNode(5);

Head = Temp1;

left = new treeNode(2);

right = new treeNode(7);

Head->left = left;

Head->right = right;

Temp1 = left;

Temp2 = right;

left = new treeNode(1);

right = new treeNode(3);

Temp1->left = left;

Temp1->right = right;

left = new treeNode(6);

right = new treeNode(8);

Temp2->left = left;

Temp2->right = right;

printTree(Head);

//a. Insert new node

Temp1 = new treeNode(9);

cout<<"\n\n\n\n\n";

recursiveInsert(Head, Temp1);

printTree(Head);

//b. Print the path to the node which has search data

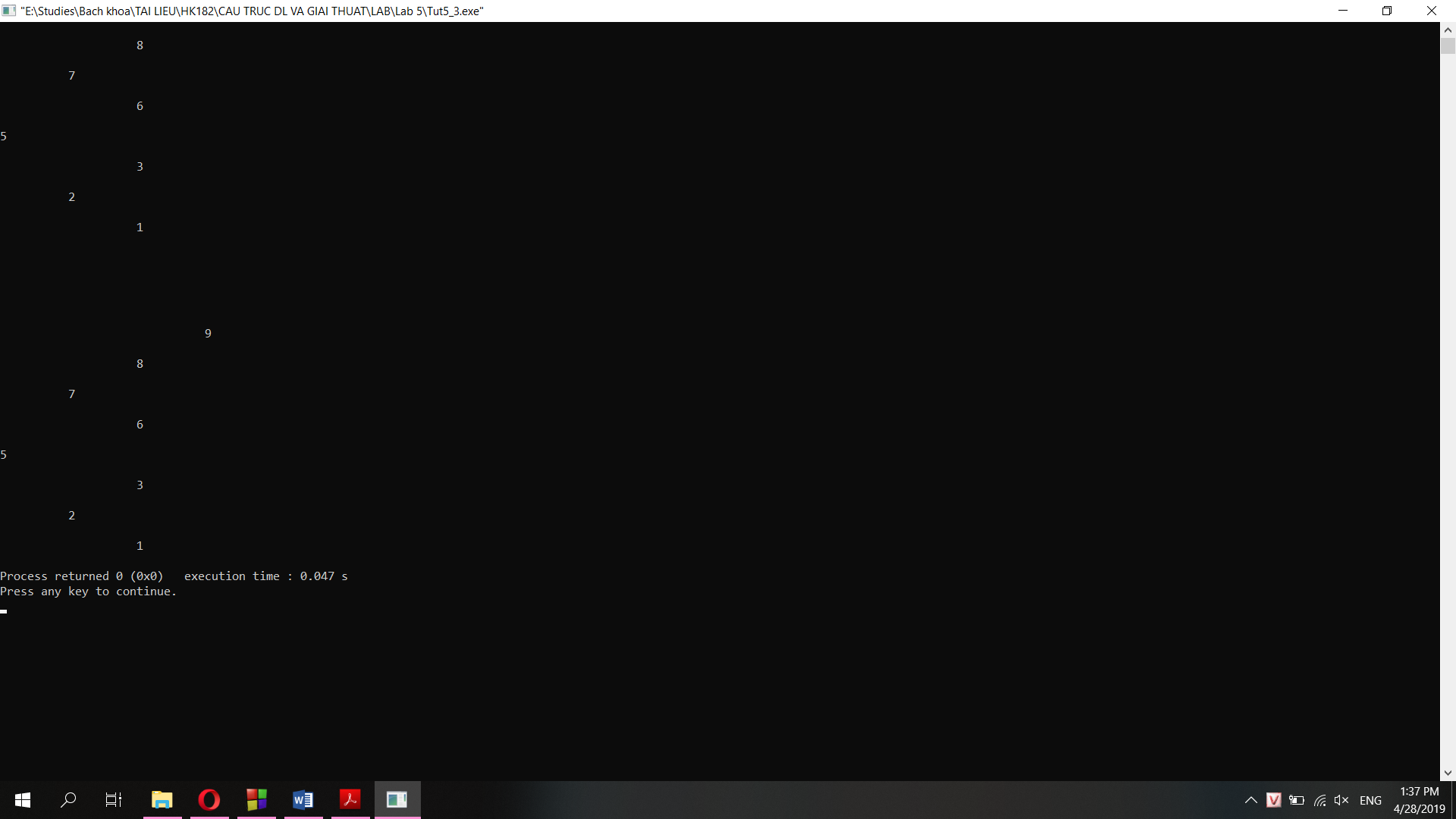
printPath(Head, 9);

//c. Print all leaves using BFT

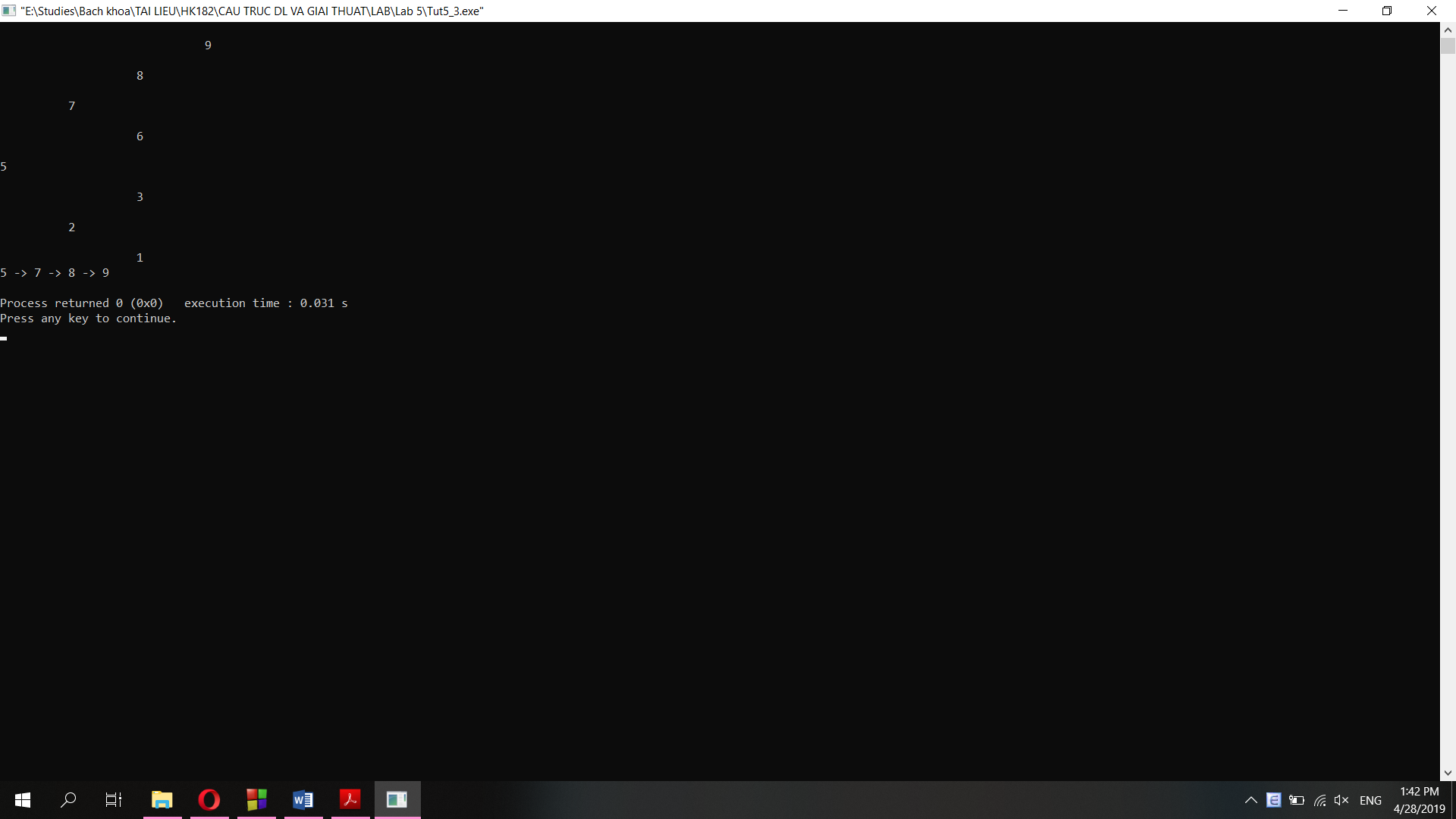
printLeavesBFT(Head);

}

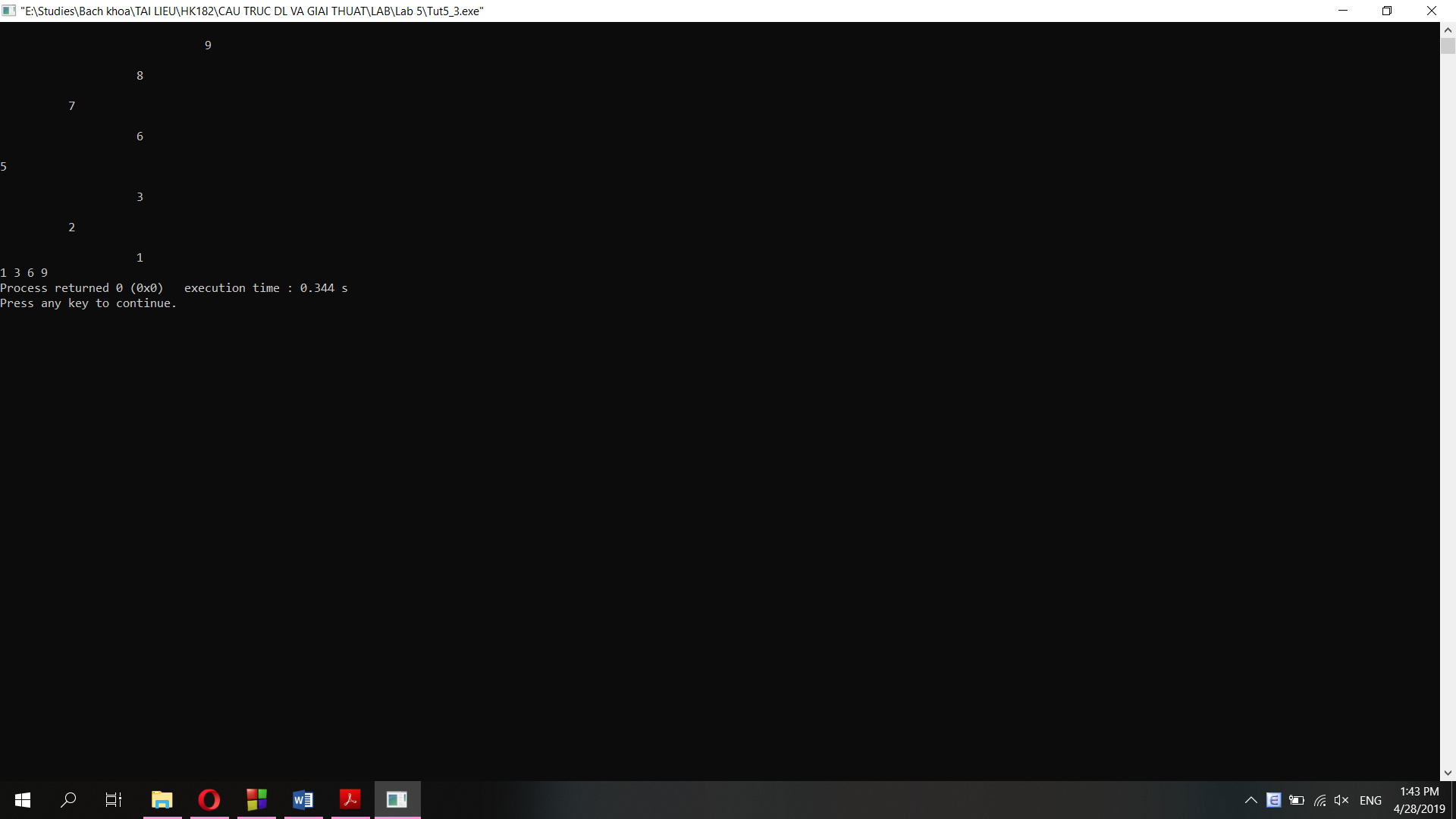
* **Insert a new node:**



* **Print the path from root to the node which has searched data (9)**



* **Print out all leaves of the tree via Breadth First Traverse**



**LAB 5**

#include <iostream>

#include <string>

#include <string.h>

#define COUNT 10

using namespace std;

//Question 1

class TreeNode {

private:

string character;

int count;

TreeNode\* left = NULL;

TreeNode\* right = NULL;

public:

TreeNode(string character);

TreeNode(char character);

void DestructorRecur(TreeNode \*T); //Add new

~TreeNode();

void increaseCount();

int getCount();

void setCount(int newCount);

string getChar();

void setChar(string newChar);

TreeNode\* getLeft();

void setLeft(TreeNode\* newLeft);

TreeNode\* getRight();

void setRight(TreeNode\* newRight);

void print2D(TreeNode \*root, int space); //Add new

void printTree(TreeNode \*root); //Add new

};

TreeNode::TreeNode(string character){

this->character = character;

count = 1;

this->left = NULL;

this->right = NULL;

}

TreeNode::TreeNode(char character){

this->character = character;

count = 1;

this->left = NULL;

this->right = NULL;

}

TreeNode::~TreeNode(){

DestructorRecur(this);

}

void TreeNode::DestructorRecur(TreeNode \*T) {

if (T != NULL) {

DestructorRecur(T->left);

DestructorRecur(T->right);

delete T;

}

}

void TreeNode::increaseCount(){

this->count++;

}

int TreeNode::getCount(){

return this->count;

}

void TreeNode::setCount(int newCount){

this->count = newCount;

}

string TreeNode::getChar(){

return this->character;

}

void TreeNode::setChar(string newChar){

this->character = newChar;

}

TreeNode\* TreeNode::getLeft(){

return this->left;

}

void TreeNode::setLeft(TreeNode\* newLeft){

this->left = newLeft;

}

TreeNode\* TreeNode::getRight(){

return this->right;

}

void TreeNode::setRight(TreeNode\* newRight){

this->right = newRight;

}

void TreeNode::print2D(TreeNode \*root, int space){

if (root == NULL){

return;

}

space += COUNT;

//Call a recursive function to print each node on the right

print2D(root->right, space);

cout<<endl;

for (int i = COUNT; i < space; i++){

cout<<" ";

}

cout<<root->character<<"\n";

//Call a recursive function to print each node on the left

print2D(root->left, space);

}

void TreeNode::printTree(TreeNode \*root){

print2D(root, 0);

}

//Question 2

class BinarySearchTree {

public:

TreeNode\* root = NULL;

void insert(TreeNode\* node); // Insert a node, if the "character" of "node" does exist, then increase the "count"

void remove(string character); // Remove/delete a node having the same "character"

int search(string character); // return "count"

//void print();

void print2D(TreeNode \*root, int space);

void printTree(); // print out the whole tree on the console

};

void BinarySearchTree::insert(TreeNode \*node){

//If the BST is null, add the node as the root

if (this->root == NULL) {

this->root = node;

}

//If the BST is not null

else {

TreeNode \*prev, \*cur;

cur = this->root;

while (cur != NULL) {

prev = cur;

//if there is a node has the same character already, increase count then break

if (node->getChar() == cur->getChar()){

cur->increaseCount();

break;

}

//else, keep looking for the node

else if (node->getChar() > cur->getChar()){

cur = cur->getRight();

}

else {

cur = cur->getLeft();

}

}

//if we reach the end of the BST but found no node has the same data, add a new node

if (cur == NULL){

if (node->getChar() > prev->getChar()){

prev->setRight(node);

}

else {

prev->setLeft(node);

}

}

}

}

void BinarySearchTree::remove(string character){

// TO DO

}

int BinarySearchTree::search(string character){

//If the BST is null, return 0

if (this->root == NULL){

return 0;

}

else {

TreeNode \*pTemp = this->root;

//Traverse through the BST to find the node

while (pTemp != NULL){

if (character == pTemp->getChar()){

return pTemp->getCount();

}

else if (character > pTemp->getChar()){

pTemp = pTemp->getRight();

}

else {

pTemp = pTemp->getLeft();

}

}

//if the node is not in BST, return 0

if (pTemp == NULL){

return 0;

}

}

}

void BinarySearchTree::print2D(TreeNode \*root, int space){

if (root == NULL){

return;

}

space += COUNT;

//Call a recursive function to print each node on the right

print2D(root->getRight(), space);

cout<<endl;

for (int i = COUNT; i < space; i++){

cout<<" ";

}

cout<<root->getChar()<<"\n";

//Call a recursive function to print each node on the left

print2D(root->getLeft(), space);

}

void BinarySearchTree::printTree(){

print2D(this->root, 0);

}

//Question 3

BinarySearchTree\* buildTreeFromString(string str){

BinarySearchTree \*B = new BinarySearchTree();

TreeNode \*pTemp;

pTemp = new TreeNode(str[0]);

B->root = pTemp; //Add the root first

int n = str.length(); //Get the string length

//traverse through the string and add characters (alphabet only)

for (int i=1; i<n; i++){

if ((str[i]>=97 && str[i]<=122) || (str[i]>=65 && str[i]<=90)){

pTemp = new TreeNode(str[i]);

B->insert(pTemp);

}

}

return B;

}

int main(){

/\*

//Question 1 & 2, except for remove() function

TreeNode \*T1 = new TreeNode("B");

TreeNode \*T2 = new TreeNode("A");

TreeNode \*T3 = new TreeNode("C");

TreeNode \*T4 = new TreeNode("D");

TreeNode \*T5 = new TreeNode("D");

TreeNode \*T6 = new TreeNode("D");

T1->setLeft(T2);

T1->setRight(T3);

BinarySearchTree B;

B.root = T1;

B.insert(T4);

B.insert(T5);

B.insert(T6);

B.printTree();

cout<<"T4->getCount() = "<<T4->getCount();

\*/

//Question 3

string str = "A binary search tree is a binary tree with the following properties: All items in the left subtree are less than the root. All items in the right subtree are greater than or equal to the root. Each subtree is itself a binary search tree.";

BinarySearchTree\* bst = buildTreeFromString(str);

bst->printTree();

cout << endl;

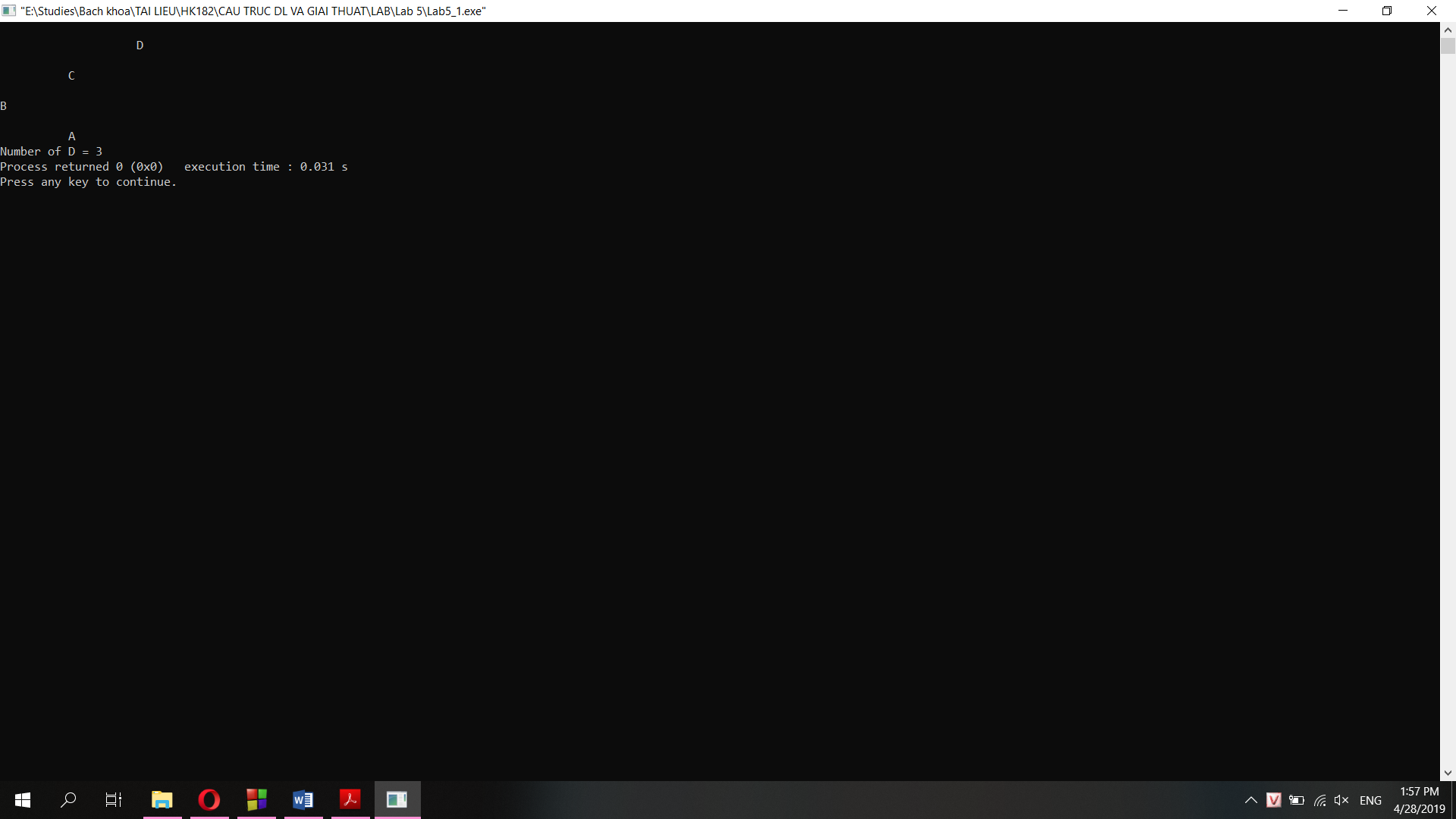
cout << "b = " << bst->search("b") << endl; // 6 times

cout << "s = " << bst->search("s") << endl; // 13 times

cout << "t = " << bst->search("t") << endl; // 24 times

}

**Question 1&2: (except the remove() function)**



**Question 3:**

