**Question#1**

**(A)**

**Code:**

#include<iostream>

#include<string>

using namespace std;

void copyStr(string s1, string &s2, int index) {

if (index!=s1.length()) {

if (s1[index] >= 'A' && s1[index] <= 'Z') {

s2 = s2 + s1[index];

}

copyStr(s1, s2, ++index);

}

}

int main() {

string str = "AbCdEfG";

string str1 = "";

int index = 0;

cout << "Original String : " << str << endl;

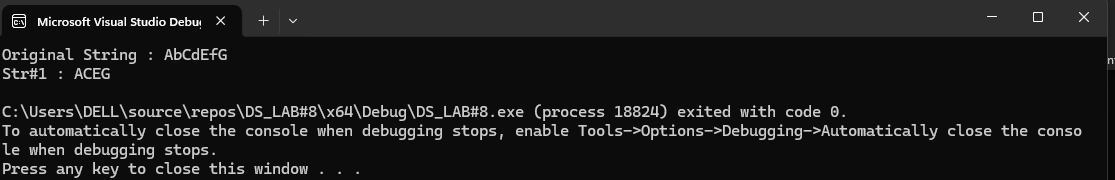
copyStr(str, str1,index);

cout << "Str#1 : " << str1 << endl;

return 0;

}

**Output:**

****

**(B)**

**Code:**

#include<iostream>

using namespace std;

int magicNumber(int arr[], int size) {

int sum = 0;

if (size != -1) {

if (size % 2 == 0) {

sum = sum + arr[size] \* 9;

}

else {

sum = sum + arr[size] \* 99;

}

sum = sum + magicNumber(arr, --size);

}

return sum;

}

int main() {

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

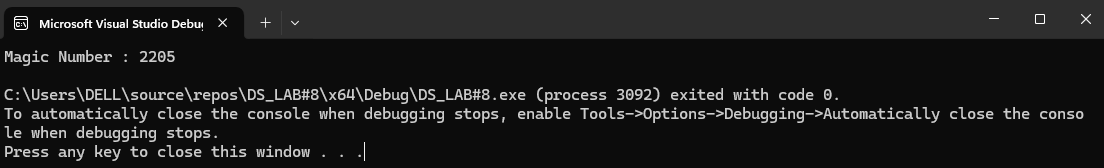
int size = 9;

cout << "Magic Number : " << magicNumber(arr, size – 1) << endl;

return 0;

}

**Output:**

****

**Question#2**

**(A)**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* left;

node\* right;

node(int value) {

data = value;

left = right = nullptr;

}

};

class BST {

public:

node\* root;

BST() {

root = nullptr;

}

void insert(node\* temp, int value) {

if (root == nullptr) {

root = new node(value);

return;

}

if (value < temp->data) {

if (temp->left != nullptr) {

insert(temp->left, value);

}

else {

temp->left = new node(value);

}

}

else if (value > temp->data) {

if (temp->right != nullptr) {

insert(temp->right, value);

}

else {

temp->right = new node(value);

}

}

}

void inOrder(node\* temp) {

if (temp != nullptr) {

inOrder(temp->left);

cout << temp->data << " ";

inOrder(temp->right);

}

}

bool Search(node\* temp, int val) {

bool found = false;

while (temp != nullptr) {

if (temp->data == val) {

found = true;

break;

}

if (val <= temp->data) {

temp = temp->left;

}

else if (val >= temp->data) {

temp = temp->right;

}

}

return found;

}

};

int main() {

BST tree;

tree.insert(tree.root, 50);

tree.insert(tree.root, 30);

tree.insert(tree.root, 70);

tree.insert(tree.root, 20);

tree.insert(tree.root, 40);

tree.insert(tree.root, 60);

tree.insert(tree.root, 80);

cout << "InOrder : ";

tree.inOrder(tree.root);

cout << endl;

cout << "Search(20) : " << tree.Search(tree.root, 20) << endl;

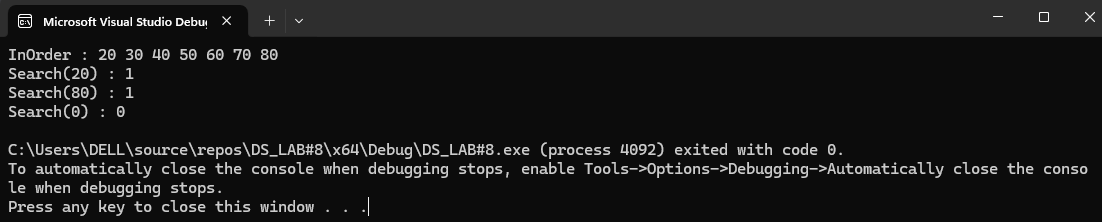
cout << "Search(80) : " << tree.Search(tree.root, 80) << endl;

cout << "Search(0) : " << tree.Search(tree.root, 0) << endl;

return 0;

}

**Output:**

****

**(B)**

**Code:**

#include<iostream>

#include<queue>

using namespace std;

struct node {

int data;

node\* left;

node\* right;

node(int value) {

data = value;

left = right = nullptr;

}

};

class BST {

public:

node\* root;

BST() {

root = nullptr;

}

void insert(node\* temp, int value) {

if (root == nullptr) {

root = new node(value);

return;

}

if (value < temp->data) {

if (temp->left != nullptr) {

insert(temp->left, value);

}

else {

temp->left = new node(value);

}

}

else if (value > temp->data) {

if (temp->right != nullptr) {

insert(temp->right, value);

}

else {

temp->right = new node(value);

}

}

}

void inOrder(node\* temp) {

if (temp != nullptr) {

inOrder(temp->left);

cout << temp->data << " ";

inOrder(temp->right);

}

}

void levelOrderTraversal(node\* root) {

if (root == nullptr) {

return;

}

queue<node\*> q;

q.push(root);

while (!q.empty()) {

node\* currentNode = q.front();

q.pop();

cout << currentNode->data << " ";

if (currentNode->left != nullptr) {

q.push(currentNode->left);

}

if (currentNode->right != nullptr) {

q.push(currentNode->right);

}

}

}

};

int main() {

BST tree;

tree.insert(tree.root, 50);

tree.insert(tree.root, 30);

tree.insert(tree.root, 70);

tree.insert(tree.root, 20);

tree.insert(tree.root, 40);

tree.insert(tree.root, 60);

tree.insert(tree.root, 80);

cout << "Level Order Traversal : ";

tree.levelOrderTraversal(tree.root);

cout << endl;

return 0;

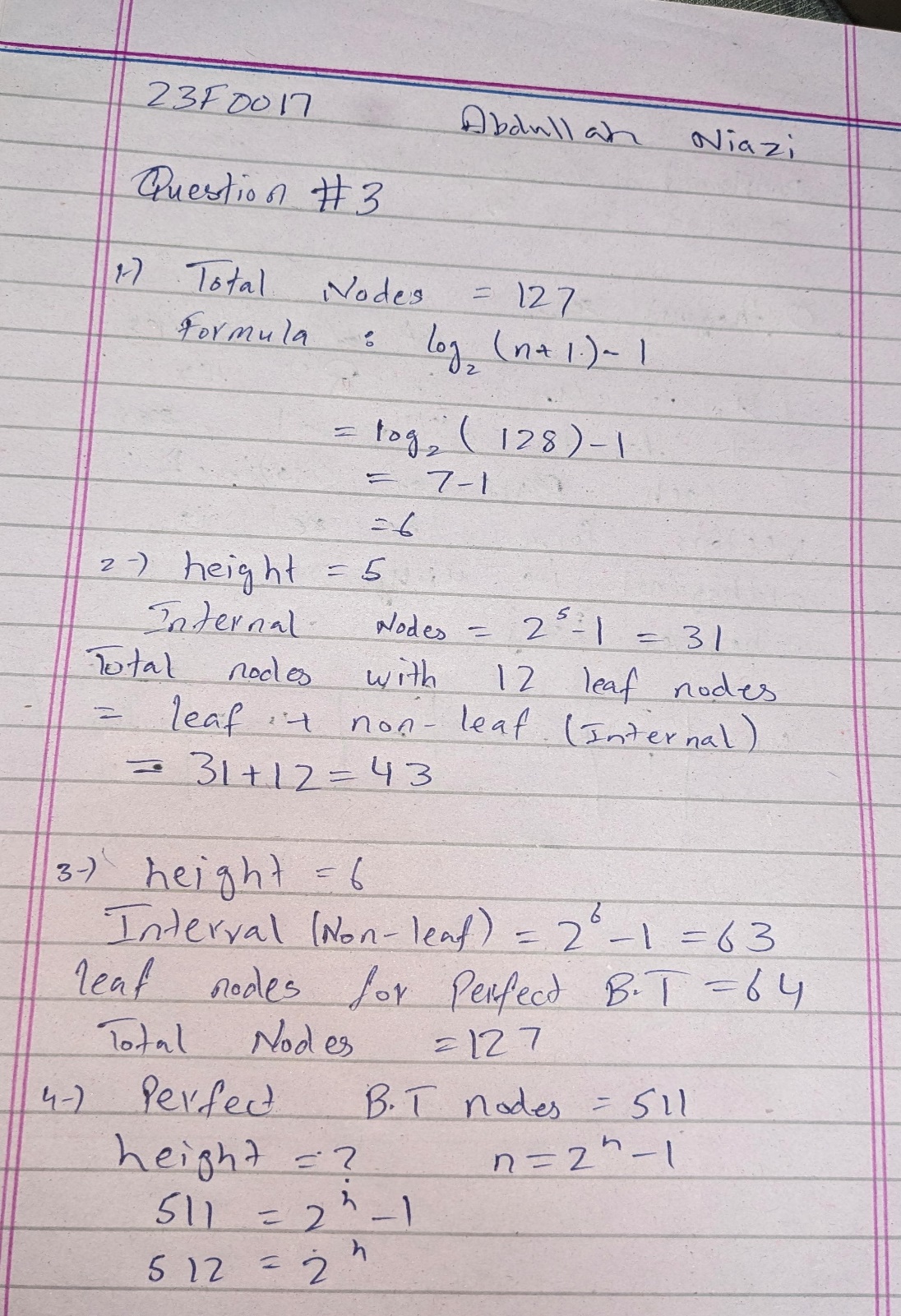
}

**Output:**

**A screen shot of a computer

Description automatically generated**

**Question#3**



A close-up of a math problem

Description automatically generated

**Question#4**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* left;

node\* right;

//constructor of node

node(int value) {

data = value;

left = right = nullptr;

}

};

class BST {

public:

node\* root;

// Constructor to initialize the BST

BST() {

root = nullptr;

}

void insert(node \*temp, int value) {

// if the tree is empty

if (root == nullptr) {

root = new node(value);

return;

}

if (value < temp->data) {

if (temp->left != nullptr) {

insert(temp->left, value);

}

else {

temp->left = new node(value);

}

}

else if (value > temp->data) {

if (temp->right != nullptr) {

insert(temp -> right, value);

}

else {

temp -> right = new node(value);

}

}

}

void inOrder(node\* temp) {

if (temp != nullptr) {

inOrder(temp->left);

cout << temp->data << " ";

inOrder(temp->right);

}

}

void Search(node\* temp,int val) {

if (temp != nullptr) {

if (temp->data == val) {

cout << ";)" << endl;

return;

}

Search(temp->left,val);

Search(temp->right,val);

}

}

};

int main() {

BST tree;

tree.insert(tree.root, 50);

tree.insert(tree.root, 30);

tree.insert(tree.root, 70);

tree.insert(tree.root, 20);

tree.insert(tree.root, 40);

tree.insert(tree.root, 60);

tree.insert(tree.root, 80);

cout << "InOrder : ";

tree.inOrder(tree.root);

cout << endl;

cout << "Search(60) :";

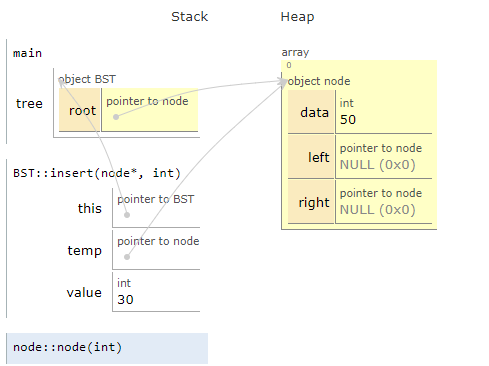
tree.Search(tree.root, 60);

return 0;

}

**Output:**

**Insertion:**

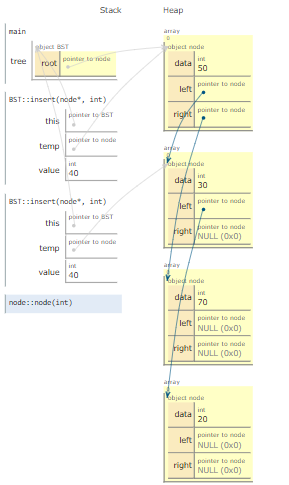
****

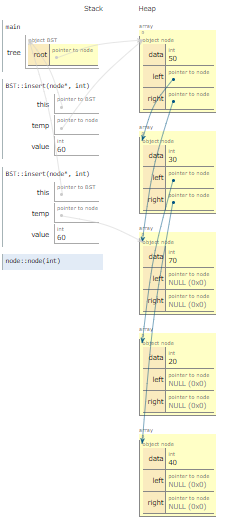
**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

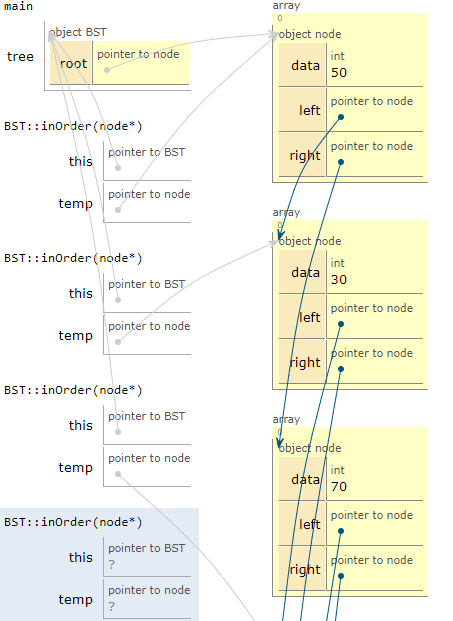
Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**InsOrder:**

****

**A screenshot of a computer

Description automatically generated**