1) What will be the height of a balanced full binary tree with 8 leaves?

a) 8

b) 5

c) 6

**d) 4**

2) The balance factor of a node in a binary tree is defined as \_\_\_\_\_

a) addition of heights of left and right subtrees

b) height of right subtree minus height of left subtree

**c) height of left subtree minus height of right subtree**

d) height of right subtree minus one

3) AVL trees are more balanced than Red-black trees.

**a) True**

b) False

4) A binary tree is balanced if the difference between left and right subtree of every node is not more than \_\_\_\_

**a) 1**

b) 3

c) 2

d) 0

5) Which of the following tree data structures is not a balanced binary tree?

a) AVL tree

b) Red-black tree

c) Splay tree

**d) B-tree**

6) Balanced binary tree with n items allows the lookup of an item in \_\_\_\_ worst-case time.

**a) O(log n)**

b) O(nlog 2)

c) O(n)

d) O(1)

7) Which of the following data structures can be efficiently implemented using height balanced binary search tree?

a) sets

b) priority queue

c) heap

**d) both sets and priority queue**

8) Two balanced binary trees are given with m and n elements respectively. They can be merged into a balanced binary search tree in \_\_\_\_ time.

**a) O(m+n)**

b) O(mn)

c) O(m)

d) O(mlog n)

9) Which of the following is an advantage of balanced binary search tree, like AVL tree, compared to binary heap?

**a) insertion takes less time**

b) deletion takes less time

c) searching takes less time

d) construction of the tree takes less time than binary heap

10) What is the output of an in-order traversal of the binary tree below?

a

/ \

b c

a) a b c

**b) b a c**

c) b c a

d) c b a

11) What is the output of a pre-order traversal of the binary tree below?

5

/ \

3 8

**a) 5 3 8**

b) 3 5 8

c) 8 5 3

d) 3 8 5

12) What is the output of a post-order traversal of the binary tree below?

2

/ \

1 3

a) 1 2 3

b) 3 1 2

**c) 1 3 2**

d) 3 2 1

13) What is the output of a level-order traversal of the binary tree below?

7

/ \

4 9

**a) 7 4 9**

b) 4 7 9

c) 7 9 4

d) 4 9 7

14) What is the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void printInorder(Node\* root) {

if (root == nullptr) return;

printInorder(root->left);

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

printInorder(root->right);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

printInorder(root);

return 0;

}

**a) 4 2 1 3**

b) 1 2 4 3

c) 4 2 3 1

d) 1 4 2 3

15) What is the function of following code snippet?

int fun(Node\* root) {

if (root == nullptr) return 0;

return 1 + fun(root->left) + fun(root->right);

}

a) Print inorder

b) Calculate height

**c) Calculate number of nodes**

d) Calculate

16) What is the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void printPostorder(Node\* root) {

if (root == nullptr) return;

printPostorder(root->left);

printPostorder(root->right);

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

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

printPostorder(root);

return 0;

}

**a) 4 5 2 3 1**

b) 5 4 2 3 1

c) 4 5 1 2 3

d) 5 4 1 3 2

17) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int sumOfNodes(Node\* root) {

if (root == nullptr) return 0;

return root->data + sumOfNodes(root->left) + sumOfNodes(root->right);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

cout << sumOfNodes(root);

return 0;

}

a) 10

**b) 15**

c) 16

d) 20

18) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

bool isLeaf(Node\* node) {

return node != nullptr && node->left == nullptr && node->right == nullptr;

}

int countLeafNodes(Node\* root) {

if (root == nullptr) return 0;

if (isLeaf(root)) return 1;

return countLeafNodes(root->left) + countLeafNodes(root->right);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->right = new Node(6);

cout << countLeafNodes(root);

return 0;

}

**a) 3**

b) 4

c) 5

d) 6

19) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int findMax(Node\* root) {

if (root == nullptr) return INT\_MIN;

int leftMax = findMax(root->left);

int rightMax = findMax(root->right);

return max(root->data, max(leftMax, rightMax));

}

int main() {

Node\* root = new Node(5);

root->left = new Node(3);

root->right = new Node(8);

root->left->left = new Node(9);

root->left->right = new Node(4);

root->right->left = new Node(6);

cout << findMax(root);

return 0;

}

a) 6

b) 8

**c) 9**

d) 5

20) What will the output be for the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int countFullNodes(Node\* root) {

if (root == nullptr) return 0;

int fullNodes = countFullNodes(root->left) + countFullNodes(root->right);

if (root->left && root->right) fullNodes++;

return fullNodes;

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->right = new Node(6);

cout << countFullNodes(root);

return 0;

}

**a) 2**

b) 3

c) 4

d) 5

21) What will the output be for the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

bool isMirror(Node\* root1, Node\* root2) {

if (root1 == nullptr && root2 == nullptr) return true;

if (root1 == nullptr || root2 == nullptr) return false;

return (root1->data == root2->data) &&

isMirror(root1->left, root2->right) &&

isMirror(root1->right, root2->left);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(2);

root->left->left = new Node(3);

root->left->right = new Node(4);

root->right->left = new Node(4);

root->right->right = new Node(3);

if (isMirror(root, root))

cout << "Yes";

else

cout << "No";

return 0;

}

**a) Yes**

b) No

c) Depends on the tree structure

d) Error

22) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void printLevelOrder(Node\* root) {

if (root == nullptr) return;

queue<Node\*> q;

q.push(root);

while (!q.empty()) {

Node\* current = q.front();

cout << current->data << " ";

q.pop();

if (current->left) q.push(current->left);

if (current->right) q.push(current->right);

}

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

printLevelOrder(root);

return 0;

}

**a) 1 2 3 4 5**

b) 4 2 5 1 3

c) 1 3 2 5 4

d) 1 2 4 3 5

23) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void printSpiralOrder(Node\* root) {

if (root == nullptr) return;

stack<Node\*> s1;

stack<Node\*> s2;

s1.push(root);

while (!s1.empty() || !s2.empty()) {

while (!s1.empty()) {

Node\* current = s1.top();

s1.pop();

cout << current->data << " ";

if (current->right) s2.push(current->right);

if (current->left) s2.push(current->left);

}

while (!s2.empty()) {

Node\* current = s2.top();

s2.pop();

cout << current->data << " ";

if (current->left) s1.push(current->left);

if (current->right) s1.push(current->right);

}

}

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

printSpiralOrder(root);

return 0;

}

a) 1 2 3 4 5 6 7

b) 4 2 5 1 6 3 7

**c) 1 2 3 7 6 5 4**

d) 1 2 4 3 6 5 7

24) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int findMinDepth(Node\* root) {

if (root == nullptr) return 0;

if (root->left == nullptr && root->right == nullptr) return 1;

if (root->left == nullptr) return findMinDepth(root->right) + 1;

if (root->right == nullptr) return findMinDepth(root->left) + 1;

return min(findMinDepth(root->left), findMinDepth(root->right)) + 1;

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

cout << findMinDepth(root);

return 0;

}

a) 2

**b) 3**

c) 4

d) 1

25) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

bool isBalanced(Node\* root, int& height) {

if (root == nullptr) {

height = 0;

return true;

}

int leftHeight = 0, rightHeight = 0;

bool leftBalanced = isBalanced(root->left, leftHeight);

bool rightBalanced = isBalanced(root->right, rightHeight);

height = max(leftHeight, rightHeight) + 1;

return abs(leftHeight - rightHeight) <= 1 && leftBalanced && rightBalanced;

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->right = new Node(6);

int height = 0;

if (isBalanced(root, height))

cout << "Balanced";

else

cout << "Not Balanced";

return 0;

}

**a) Balanced**

b) Not Balanced

c) Depends on the tree structure

d) Error

26) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

bool isSubtree(Node\* root1, Node\* root2) {

if (root1 == nullptr) return false;

if (root2 == nullptr) return true;

if (root1->data == root2->data) return isSubtree(root1->left, root2->left) && isSubtree(root1->right, root2->right);

return isSubtree(root1->left, root2) || isSubtree(root1->right, root2);

}

int main() {

Node\* root1 = new Node(5);

root1->left = new Node(3);

root1->right = new Node(8);

root1->left->left = new Node(9);

root1->left->right = new Node(4);

root1->right->left = new Node(6);

root1->right->right = new Node(1);

Node\* root2 = new Node(3);

root2->left = new Node(4);

root2->right = new Node(5);

if (isSubtree(root1, root2))

cout << "Yes";

else

cout << "No";

return 0;

}

a) Yes

**b) No**

c) Depends on the tree structure

d) Error

27) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int findDiameter(Node\* root, int& diameter) {

if (root == nullptr) return 0;

int leftHeight = findDiameter(root->left, diameter);

int rightHeight = findDiameter(root->right, diameter);

diameter = max(diameter, leftHeight + rightHeight + 1);

return max(leftHeight, rightHeight) + 1;

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->right = new Node(6);

int diameter = 0;

findDiameter(root, diameter);

cout << diameter;

return 0;

}

a) 3

b) 4

**c) 5**

d) 6

28) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void printTopView(Node\* root) {

if (root == nullptr) return;

map<int, int> m;

queue<pair<Node\*, int>> q;

q.push({root, 0});

while (!q.empty()) {

Node\* current = q.front().first;

int hd = q.front().second;

q.pop();

if (m.find(hd) == m.end()) m[hd] = current->data;

if (current->left) q.push({current->left, hd - 1});

if (current->right) q.push({current->right, hd + 1});

}

for (const auto& entry : m) {

cout << entry.second << " ";

}

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

printTopView(root);

return 0;

}

**a) 4 2 1 3 7**

b) 4 2 1 6 3 7

c) 4 2 1 3 7 6

d) 4 2 1 6 7 3

29) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int findLevel(Node\* root, int val, int level) {

if (root == nullptr) return 0;

if (root->data == val) return level;

int leftLevel = findLevel(root->left, val, level + 1);

if (leftLevel != 0) return leftLevel;

return findLevel(root->right, val, level + 1);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

int target = 5;

cout << findLevel(root, target, 1);

return 0;

}

a) 2

**b) 3**

c) 4

d) 5

30) What will be the output of the following code?

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

int findSumAtLevel(Node\* root, int level) {

if (root == nullptr) return 0;

if (level == 0) return root->data;

return findSumAtLevel(root->left, level - 1) + findSumAtLevel(root->right, level - 1);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

int targetLevel = 2;

cout << findSumAtLevel(root, targetLevel);

return 0;

}

a) 9

b) 12

**c) 22**

d) 10