1. Which of the following statements is true regarding AVL trees?

a) AVL trees are unbalanced binary search trees.

**b) AVL trees are a type of self-balancing binary search trees.**

c) AVL trees are best suited for small datasets.

d) AVL trees have O(n) worst-case complexity for insertion and deletion.

2. What will be the output of the following C++ program?

#include <iostream>

struct Node {

int key;

Node\* left;

Node\* right;

int height;

};

Node\* insert(Node\* root, int key) {

// AVL tree insertion code

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

std::cout << root->key;

return 0;

}

a) 10

b) 20

**c) 30**

d) 40

3. What is the height difference allowed between the left and right subtrees of any node in an AVL tree?

a) 0

**b) 1**

c) 2

d) 3

4. What will be the output of the following C++ program?

#include <iostream>

struct Node {

int key;

Node\* left;

Node\* right;

int height;

};

Node\* insert(Node\* root, int key) {

// AVL tree insertion code

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 15);

root = insert(root, 5);

std::cout << root->left->key;

return 0;

}

a) 5

b) 10

**c) 15**

d) 20

5. An AVL tree with n nodes has a height of \_\_\_\_\_\_ at most.

a) n - 1

b) n

c) log(n)

**d) 2 \* log(n)**

6. What will be the output of the following C++ program?

#include <iostream>

struct Node {

int key;

Node\* left;

Node\* right;

int height;

};

Node\* insert(Node\* root, int key) {

// AVL tree insertion code

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

root = insert(root, 5);

root = insert(root, 15);

root = insert(root, 12);

std::cout << root->right->left->key;

return 0;

}

a) 5

b) 10

**c) 12**

d) 15

7. In an AVL tree, what is the maximum number of rotations required during an insertion to maintain the balance factor?

a) 1

**b) 2**

c) 3

d) 4

8. What will be the output of the following C++ program?

#include <iostream>

struct Node {

int key;

Node\* left;

Node\* right;

int height;

};

Node\* insert(Node\* root, int key) {

// AVL tree insertion code

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 15);

root = insert(root, 30);

root = insert(root, 25);

std::cout << root->right->right->key;

return 0;

}

a) 15

b) 20

**c) 25**

d) 30

9. AVL tree is an example of a \_\_\_\_\_.

a) Search tree

b) Traversal tree

c) Priority tree

**d) Balanced tree**

10. What will be the output of the following C++ program?

#include <iostream>

struct Node {

int key;

Node\* left;

Node\* right;

int height;

};

Node\* insert(Node\* root, int key) {

// AVL tree insertion code

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 5);

root = insert(root, 15);

std::cout << root->left->right->key;

return 0;

}

a) 5

b) 10

**c) 15**

d) 20

11.What is the time complexity of searching an element in an AVL tree with n nodes?

a) O(1)

**b) O(log n)**

c) O(n)

d) O(n log n)

12. What will be the output of the following C++ program?

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

int main() {

Node\* root = nullptr;

root = new Node{10, nullptr, nullptr};

root->left = new Node{5, nullptr, nullptr};

root->right = new Node{15, nullptr, nullptr};

cout << "Root key: " << root->key << endl;

cout << "Left child key: " << root->left->key << endl;

cout << "Right child key: " << root->right->key << endl;

return 0;

}

**a) Root key: 10, Left child key: 5, Right child key: 15**

b) Root key: 5, Left child key: 10, Right child key: 15

c) Root key: 15, Left child key: 10, Right child key: 5

d) Root key: 10, Left child key: 15, Right child key: 5

13. For a binary search tree to be an AVL tree, the balance factor of each node must be in the range of \_\_\_\_\_\_.

**a) -1 to 1**

b) -2 to 2

c) 0 to 1

d) -1 to 0

14. What will be the output of the following C++ program?

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

int main() {

Node\* root = nullptr;

root = new Node{10, nullptr, nullptr};

root->left = new Node{5, nullptr, nullptr};

root->right = new Node{15, nullptr, nullptr};

root->left->left = new Node{3, nullptr, nullptr};

root->left->right = new Node{8, nullptr, nullptr};

cout << "Root key: " << root->key << endl;

cout << "Left child key: " << root->left->key << endl;

cout << "Right child key: " << root->right->key << endl;

cout << "Left grandchild key: " << root->left->left->key << endl;

cout << "Right grandchild key: " << root->left->right->key << endl;

return 0;

}

**a) Root key: 10, Left child key: 5, Right child key: 15, Left grandchild key: 3, Right grandchild key: 8**

b) Root key: 5, Left child key: 10, Right child key: 15, Left grandchild key: 3, Right grandchild key: 8

c) Root key: 15, Left child key: 10, Right child key: 5, Left grandchild key: 8, Right grandchild key: 3

d) Root key: 10, Left child key: 15, Right child key: 5, Left grandchild key: 3, Right grandchild key: 8

15. Which of the following operations can unbalance an AVL tree?

a) Insertion

b) Deletion

**c) Both insertion and deletion**

d) Searching

16. What will be the output of the following C++ program?

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

void insert(Node\*& root, int key) {

if (root == nullptr) {

root = new Node{key, nullptr, nullptr};

return;

}

if (key < root->key)

insert(root->left, key);

else

insert(root->right, key);

}

int main() {

Node\* root = nullptr;

insert(root, 12);

insert(root, 7);

insert(root, 17);

cout << "Root key: " << root->key << endl;

cout << "Left child key: " << root->left->key << endl;

cout << "Right child key: " << root->right->key << endl;

return 0;

}

**a) Root key: 12, Left child key: 7, Right child key: 17**

b) Root key: 7, Left child key: 17, Right child key: 12

c) Root key: 12, Left child key: 17, Right child key: 7

d) Root key: 17, Left child key: 12, Right child key: 7

17. What is the worst-case time complexity for inserting a node into an AVL tree with n nodes?

**a) O(log n)**

b) O(n)

c) O(n log n)

d) O(1)

18. What will be the output of the following C++ program?

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

void insert(Node\*& root, int key) {

if (root == nullptr) {

root = new Node{key, nullptr, nullptr};

return;

}

if (key < root->key)

insert(root->left, key);

else

insert(root->right, key);

}

int getHeight(Node\* root) {

if (root == nullptr)

return -1;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

return max(leftHeight, rightHeight) + 1;

}

int main() {

Node\* root = nullptr;

insert(root, 10);

insert(root, 5);

insert(root, 15);

insert(root, 3);

insert(root, 8);

cout << "Height of the AVL tree: " << getHeight(root) << endl;

return 0;

}

a) Height of the AVL tree: 2

**b) Height of the AVL tree: 3**

c) Height of the AVL tree: 4

d) Height of the AVL tree: 5

19. An AVL tree becomes a degenerate tree if \_\_\_\_\_\_.

a) All nodes have the same value

**b) The height difference between left and right subtrees is greater than 1**

c) It has only one node

d) It has no nodes

20. What will be the output of the following C++ program?

#include <iostream>

using namespace std;

struct Node {

int key;

Node\* left;

Node\* right;

};

void insert(Node\*& root, int key) {

if (root == nullptr) {

root = new Node{key, nullptr, nullptr};

return;

}

if (key < root->key)

insert(root->left, key);

else

insert(root->right, key);

}

bool isAVL(Node\* root) {

if (root == nullptr)

return true;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

int balanceFactor = abs(leftHeight - rightHeight);

return (balanceFactor <= 1) && isAVL(root->left) && isAVL(root->right);

}

int getHeight(Node\* root) {

if (root == nullptr)

return -1;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->

right);

return max(leftHeight, rightHeight) + 1;

}

int main() {

Node\* root = nullptr;

insert(root, 10);

insert(root, 5);

insert(root, 15);

insert(root, 3);

insert(root, 8);

if (isAVL(root))

cout << "The tree is an AVL tree." << endl;

else

cout << "The tree is not an AVL tree." << endl;

return 0;

}

**a) The tree is an AVL tree.**

b) The tree is not an AVL tree.

c) The program will not compile due to an error.

d) The program will run into an infinite loop.