Roman Vasilyev

Prof Ross

CISP 430

3/3/2024

Assignment 6

Binary Trees

```
#include <iostream>
using namespace std;

struct Node {
	int data; Node* left; Node* right;

	Node(int data) {
		this->data = data;
		left = right = NULL;
	}
};

void print_preorder(Node* root) {
	if (root == NULL) return;
	cout << root->data << ", "; print_preorder(root->left); print_preorder(root-
>right);
}


void print_inorder(Node* root) {
	if (root == NULL) return; print_inorder(root->left);
	cout << root->data << ", "; print_inorder(root->right);
}


void print_postorder(Node* root) {
	if (root == NULL) return; print_postorder(root->left); print_postorder(root-
>right);
	cout << root->data << ", ";
}


Node* insert(Node* root, int data) {
	if (root == NULL) {
		return new Node(data);
	}
	else if (data <= root->data) {
		root->left = insert(root->left, data);
	}
	else {
		root->right = insert(root->right, data);
	}
```

```
	return root;
}


int search(Node* root, int data) {
	if (root == NULL) return 0;
	if (root->data == data) return 1;
	int left_search = search(root->left, data); int right_search = search(root-
>right, data); if (left_search > 0 || right_search > 0) {
		return 1 + left_search + right_search;
	}
	return 0;
}


int main() {
	Node* root = NULL;


	//SEQUENCE A

	root = insert(root, 1); root = insert(root, 5); root = insert(root, 4); root =
insert(root, 6); root = insert(root, 7); root = insert(root, 2); root = insert(root,
3);



	//SEQUENCE B
	/*
	root = insert(root, 150); root = insert(root, 125); root = insert(root, 175);
root = insert(root, 166); root = insert(root, 163); root = insert(root, 123); root =
insert(root, 108); root = insert(root, 116); root = insert(root, 117); root =
insert(root, 184); root = insert(root, 165); root = insert(root, 137); root =
insert(root, 141); root = insert(root, 111); root = insert(root, 138); root =
insert(root, 122); root = insert(root, 109); root = insert(root, 194); root =
insert(root, 143); root = insert(root, 183); root = insert(root, 178); root =
insert(root, 173); root = insert(root, 139); root = insert(root, 126); root =
insert(root, 170); root = insert(root, 190); root = insert(root, 140); root =
insert(root, 188); root = insert(root, 120);

	root = insert(root, 195); root = insert(root, 113); root = insert(root, 104);
root = insert(root, 193); root = insert(root, 181); root = insert(root, 185); root =
insert(root, 198); root = insert(root, 103); root = insert(root, 182); root =
insert(root, 136); root = insert(root, 115); root = insert(root, 191); root =
insert(root, 144); root = insert(root, 145); root = insert(root, 155); root =
insert(root, 153); root = insert(root, 151); root = insert(root, 112); root =
insert(root, 129); root = insert(root, 199); root = insert(root, 135); root =
insert(root, 146); root = insert(root, 157); root = insert(root, 176); root =
insert(root, 159); root = insert(root, 196); root = insert(root, 121); root =
insert(root, 105); root = insert(root, 131);

	root = insert(root, 154); root = insert(root, 107); root = insert(root, 110);
root = insert(root, 158); root = insert(root, 187); root = insert(root, 134); root =
insert(root, 132); root = insert(root, 179); root = insert(root, 133); root =
insert(root, 102); root = insert(root, 172); root = insert(root, 106); root =
insert(root, 177); root = insert(root, 171); root = insert(root, 156); root =
insert(root, 168); root = insert(root, 161); root = insert(root, 149); root =
insert(root, 124); root = insert(root, 189); root = insert(root, 167); root =
insert(root, 174); root = insert(root, 147); root = insert(root, 148); root =
insert(root, 197); root = insert(root, 160); root = insert(root, 130); root =
insert(root, 164); root = insert(root, 152);

	root = insert(root, 142); root = insert(root, 162); root = insert(root, 118);
root = insert(root, 186); root = insert(root, 169); root = insert(root, 127); root =
```

```
insert(root, 114); root = insert(root, 192); root = insert(root, 180); root =
insert(root, 101); root = insert(root, 119); root = insert(root, 128); root =
insert(root, 100);
	*/


	// Print traversal sequences cout << "Preorder traversal: ";
print_preorder(root);
	cout << endl;


	cout << "Inorder traversal: "; print_inorder(root);
	cout << endl;


	cout << "Postorder traversal: "; print_postorder(root);
	cout << endl;


	// Search for a node

	int search_val = 4;
	int num_occurrences = search(root, search_val);
	cout << "The value " << search_val << " occurs " << num_occurrences << " times
in the tree."
		<< endl;


	return 0;
}
```

Output:

Sequence A:

Microsoft Visual Studio Debug Console

```
Inorder traversal: 1, 2, 3, 4, 5, 6, 7,
Postorder traversal: 3, 2, 4, 7, 6, 5, 1,
The value 4 occurs 3 times in the tree.

C:\Users\roman\source\repos\Assignment 6\x64\Debug
To automatically close the console when debugging
Press any key to close this window . . .
```

Sequence B:

Microsoft Visual Studio Debug Console

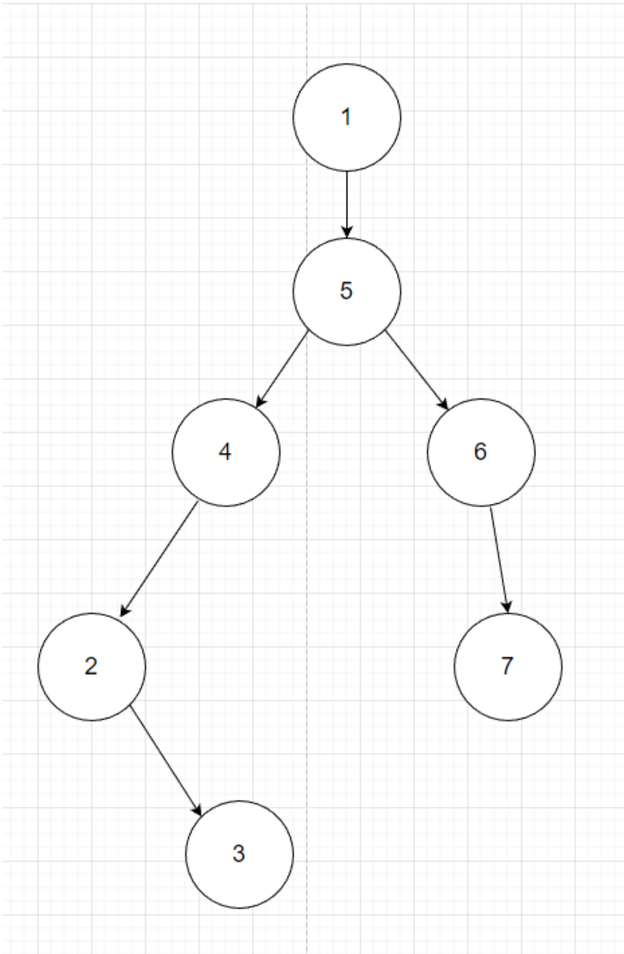
```
Inorder traversal: 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143
, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199,
Postorder traversal: 100, 101, 102, 103, 106, 107, 105, 104, 110, 109, 112, 114, 115, 113, 111, 119, 118, 121, 120, 122, 117, 116, 108, 124, 123, 128, 127, 130, 133, 132, 134, 131, 135, 129, 136, 126, 140, 139, 138, 142, 148, 147, 149, 1
46, 145, 144, 143, 141, 137, 125, 152, 151, 154, 153, 156, 158, 160, 162, 161, 159, 157, 155, 164, 165, 163, 167, 169, 168, 171, 172, 170, 174, 173, 166, 177, 176, 180, 179, 182, 181, 178, 183, 186, 187, 185, 189, 188, 192, 191, 193, 190
, 197, 196, 199, 198, 195, 194, 184, 175, 150,
The value 4 occurs 0 times in the tree.

C:\Users\roman\source\repos\Assignment 6\x64\Debug\Assignment 6.exe (process 28252) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

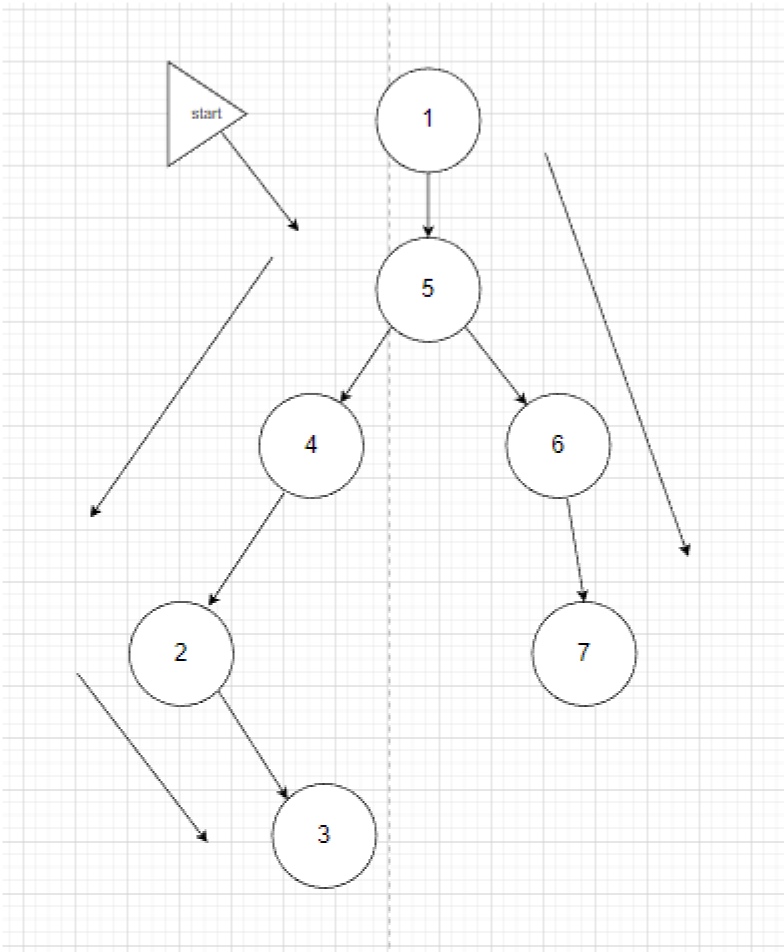
Sequence A

Binary Search Tree

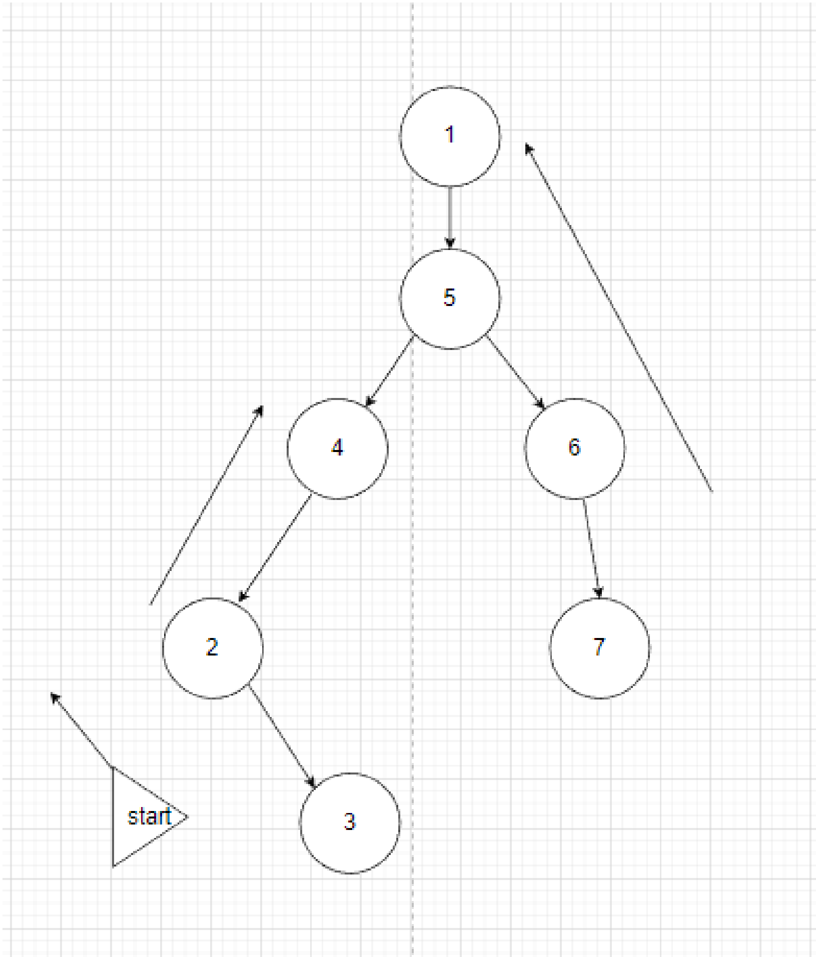
In Order



Pre Order



Post Order



Big O Analysis

Insert: When the tree is balanced, the height of the tree is log(n), where n is the number of nodes in the tree. The Insert function will have to traverse the height of the tree, which takes O(log n) time. As a result, the big O notation of the Insert function is O(h).

Print Pre Order:
The print_preorder function traverses each node of the binary tree exactly once in a pre-order fashion. Since each node is visited once, the time complexity of this function is proportional to the number of nodes in the tree is O(n).

Print Post Order:

The big O notation for the print post order function is also O(n). This is because the function traverses every node in the binary tree exactly once, visiting the left subtree, then the right subtree, and finally the root node.

Print In Order:
The time complexity of the print in-order function is O(n), where n is the number of nodes in the binary tree. This is because the function visits every node in the tree exactly once and performs a constant amount of work for each node.

Search:

The Big O notation varies depending on the binary tree. It can be O(n) in the worst case scenario but in a balanced scenario, it is O(log n) because it can discard half of the nodes.