

Quinn Roemer

CISP - 430

Assignment 6

3/7/2018

# Program 6.0 - Binary Search Tree Implementation

## Description:

The goal for this assignment was to implement a binary search tree. This tree is to support preorder, postorder, and inorder traversals. In addition, our search function must run in  $O(\log N)$  time. The following code is based on pseudocode provided by my professor.

## Source Code:

```
//Code written by Quinn Roemer, based on code by Professor Ross.
#include <iostream>
using namespace std;

//Node definition.
class node {
public:
    int data;
    node* left = NULL;
    node* right = NULL;
};

//Global node pointer.
node* root = NULL;

//Function Prototypes.
void insert(int);
void print_postorder(node*);
void print_preorder(node*);
void print_inorder(node*);
int search(int);
void process(node*);

//Main function to execute.
int main()
{
    char answer;

    cout << "Enter 1 for Sequence A or 2 for Sequence B." << endl;
    cin >> answer;
    cout << endl;

    while (answer != '1' && answer != '2')
    {
        cout << "Invalid input, please try again." << endl;
        cin >> answer;
        cout << endl;
    }
}
```

```

//Sequence A.
if (answer == '1')
{
    int dataArray[] = { 1,5,4,6,7,2,3 };

    for (int index = 0; index < sizeof(dataArray) / 4; index++)
    {
        insert(dataArray[index]);
    }
}

//Sequence B.
if (answer == '2')
{
    int dataArray[] = { 150,125,175,166,163,123,108,116,117,184,165,137,141,111,138,
122,109,194,143,183,178,173,139,126,170,190,140,188,120,195,113,104,193,181,
185,198,103,182,136,115,191,144,145,155,153,151,112,129,199,135,146,157,176,
159,196,121,105,131,154,107,110,158,187,134,132,179,133,102,172,106,177,171,
156,168,161,149,124,189,167,174,147,148,197,160,130,164,152,142,162,118,186,
169,127,114,192,180,101,119,128,100 };

    for (int index = 0; index < sizeof(dataArray) / 4; index++)
    {
        insert(dataArray[index]);
    }
}

//Demonstrating functionality.
node* temp = root;

cout << "Printing Preorder." << endl;
print_preorder(temp);
temp = root;
cout << endl << endl;

cout << "Printing Postorder." << endl;
print_postorder(temp);
temp = root;
cout << endl << endl;

cout << "Printing Inorder." << endl;
print_inorder(temp);
temp = root;
cout << endl << endl;

```

```

    cout << "Searching for 196." << endl;
    cout << search(196) << endl;

    cout << "\nSearching for 137." << endl;
    cout << search(137) << endl;

    cout << "\nSearching for 102." << endl;
    cout << search(102) << endl;

    cout << "\nSearching for 190." << endl;
    cout << search(190) << endl;

    cout << "\nSearching for 200." << endl;
    cout << "Nodes visited: " << search(200) << endl << endl;

}

//Insert function, using O(LogN) time.
void insert(int info)
{
    //Creating a new data element.
    node* temp = new node;
    temp->data = info;

    //Insertion point pointers.
    node* pc = NULL;
    node* c = root;

    //If no other nodes.
    if (root == NULL)
    {
        root = temp;
        return;
    }

    //Find insertion point.
    while (c != NULL)
    {
        pc = c;

        if (temp->data < c->data)
        {
            c = c->left;
        }

        else
        {
            c = c->right;
        }
    }
}

```

```

    }

    //Hook up node.
    if (temp->data < pc->data)
    {
        pc->left = temp;
    }
    else
    {
        pc->right = temp;
    }
}

//Print in the following fashion: Root, Left, Right.
void print_preorder(node* location)
{
    if (location != NULL)
    {
        process(location);
        print_preorder(location->left);
        print_preorder(location->right);
    }
}

//Print in the following fashion: Left, Right, Root.
void print_postorder(node* location)
{
    if (location != NULL)
    {
        print_postorder(location->left);
        print_postorder(location->right);
        process(location);
    }
}

//Print in the following fashion: Left, Root, Right.
void print_inorder(node* location)
{
    if (location != NULL)
    {
        print_inorder(location->left);
        process(location);
        print_inorder(location->right);
    }
}

//Prints the data in the passed node.
void process(node* info)
{
    cout << info->data << " ";
}

```

```

}

//Search function, using O(LogN) time.
int search(int searchTerm)
{
    node* location = root;
    int nodesVisited = 0;

    while (location != NULL)
    {
        //Data found.
        if (location->data == searchTerm)
        {
            //cout << "Search for " << searchTerm << " successful!" << endl;
            return location->data;
        }

        else
        {
            if (location->data < searchTerm)
            {
                location = location->right;
                nodesVisited++;
            }

            else
            {
                location = location->left;
                nodesVisited++;
            }
        }
    }

    //cout << "Search for " << searchTerm << " failed." << endl;
    return nodesVisited;
}

```

## Output (1 of 2):

### Sequence A

```
C:\WINDOWS\system32\cmd.exe
Enter 1 for Sequence A or 2 for Sequence B.
1

Printing Preorder.
1 5 4 2 3 6 7

Printing Postorder.
3 2 4 7 6 5 1

Printing Inorder.
1 2 3 4 5 6 7

Searching for 196.
4

Searching for 137.
4

Searching for 102.
4

Searching for 190.
4

Searching for 200.
Nodes visited: 4

Press any key to continue . . .
```

## Output (2 of 2):

### Sequence B

```
C:\WINDOWS\system32\cmd.exe
Enter 1 for Sequence A or 2 for Sequence B.
2

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

Printing Postorder.
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
134 131 135 129 136 126 140 139 138 142 148 147 149 146 145 144 143 141 137 125 152 151 154 153 156 158 160 162 161 159
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

Printing Inorder.
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

Searching for 196.
196

Searching for 137.
137

Searching for 102.
102

Searching for 190.
190

Searching for 200.
Nodes visited: 7

Press any key to continue . . .
```

## BigO of Functions:

### **void insert(int data):**

This function has to have a time complexity of  $O(\log N)$ . This is because the function only runs until it finds a suitable insertion point. This never means traversing the entire tree. Only the necessary nodes are traversed before insertion begins.

### **void print\_inorder(node\* root):**

This function has to have a time complexity of  $O(N)$ . This is because every node must be processed in order to output it.

### **void print\_preorder(node\* root):**

This function has to have a time complexity of  $O(N)$ . This is because every node must be processed in order to output it.

### **void print\_postorder(node\* root):**

This function has to have a time complexity of  $O(N)$ . This is because every node must be processed in order to output it.

### **int search(int data):**

This function has to have a time complexity of  $O(\log N)$ . This is because the search function does not need to search every node individually. This function always chooses the best path to find where the data is.

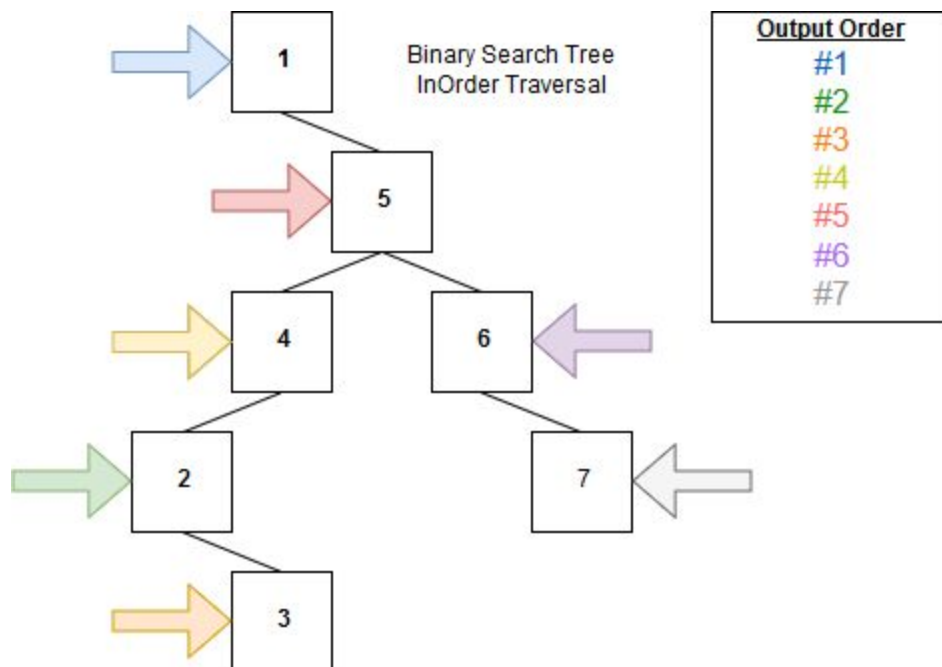


## Program 6.1 - Sequence A Hand Execution

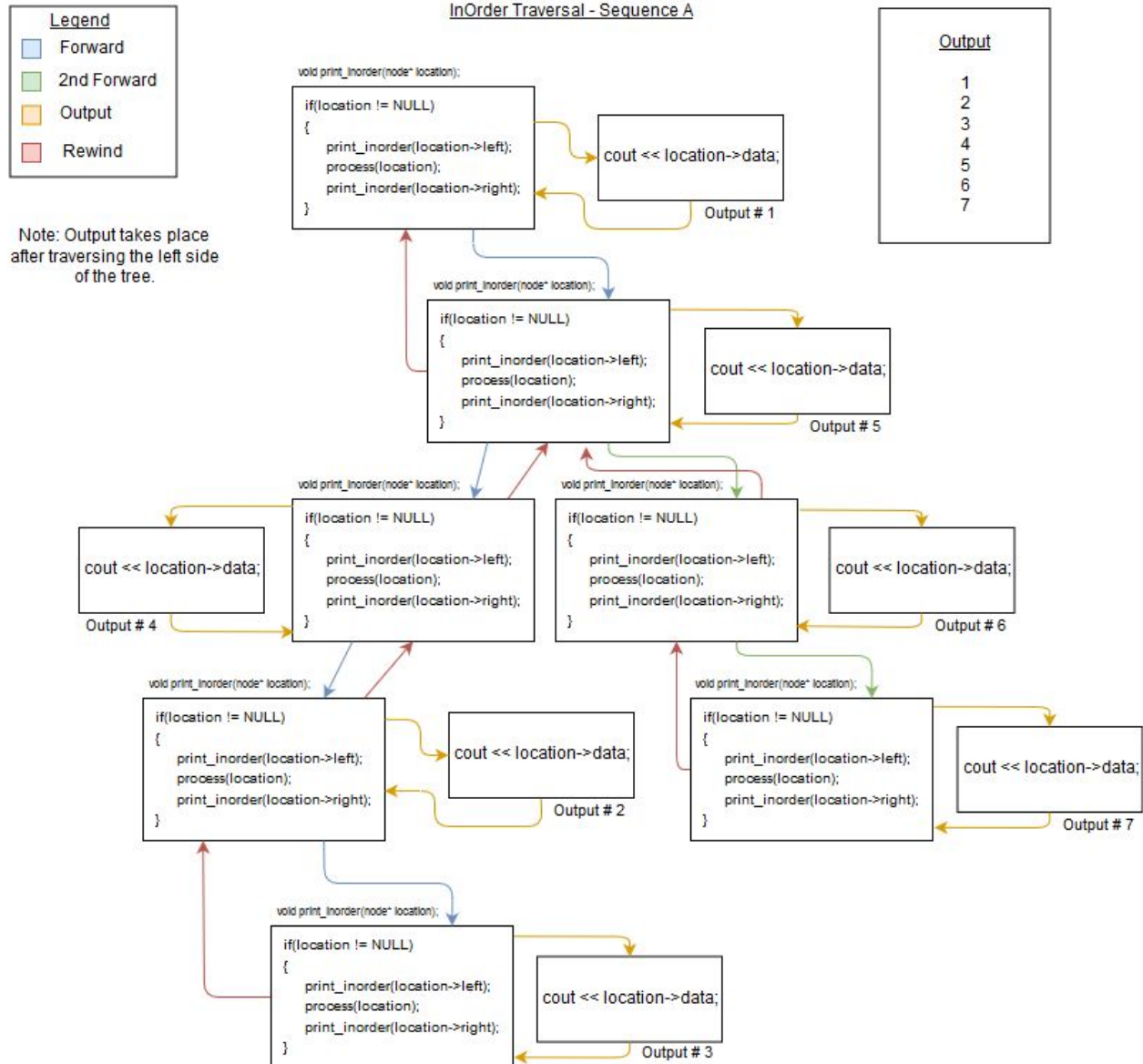
### Description:

The goal for this part of the assignment was to hand implement sequence A of the code. My diagrams are to indicate the path where the processing of each node occurs. To do this, I created two diagrams for each traversal type. One concentrates on showing the recursive nature of the function, while the other tries to show the processing order clearly.

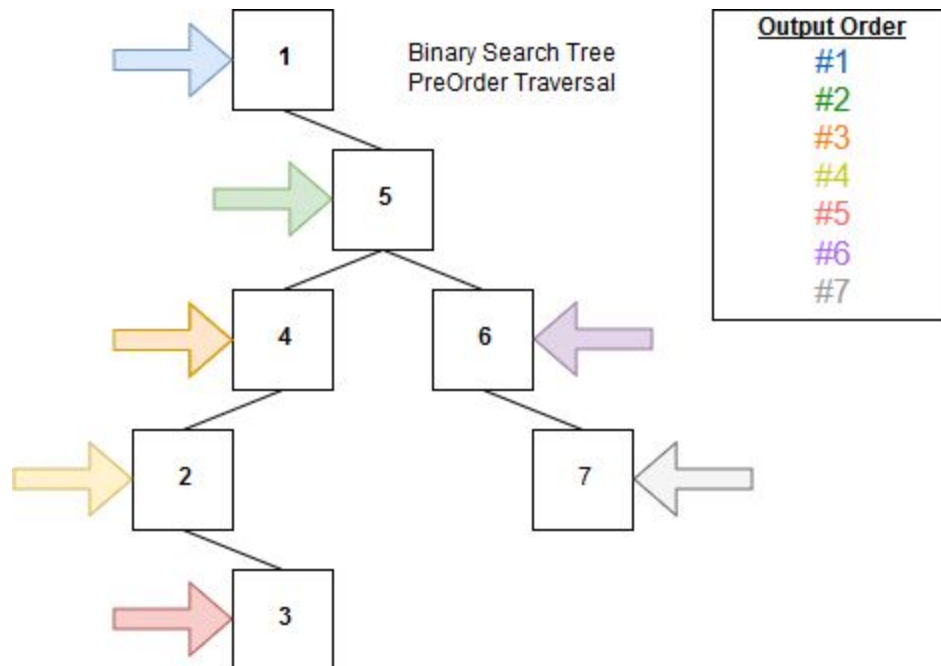
### InOrder Diagram (1 of 2):



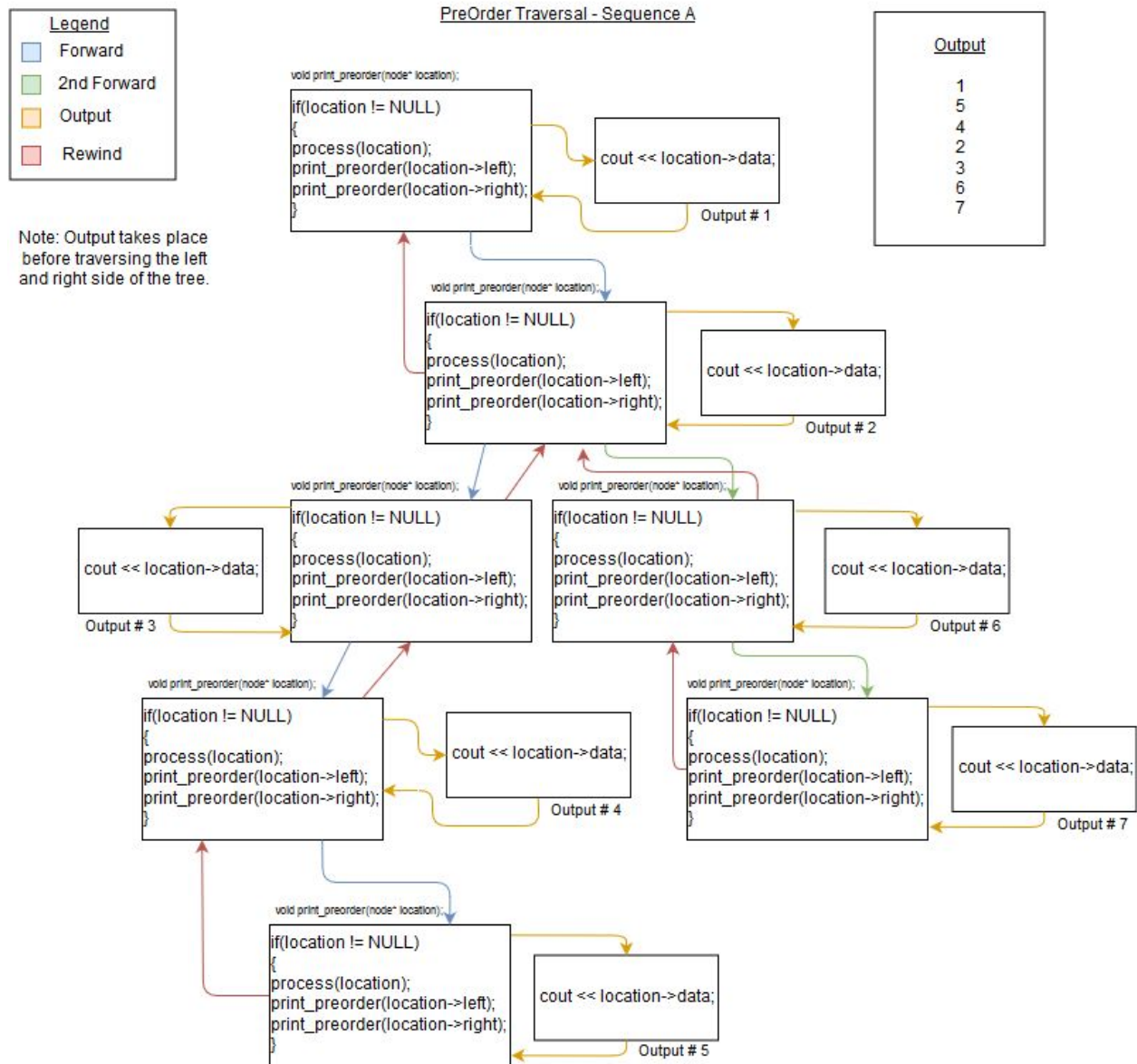
## InOrder Diagram (2 of 2):



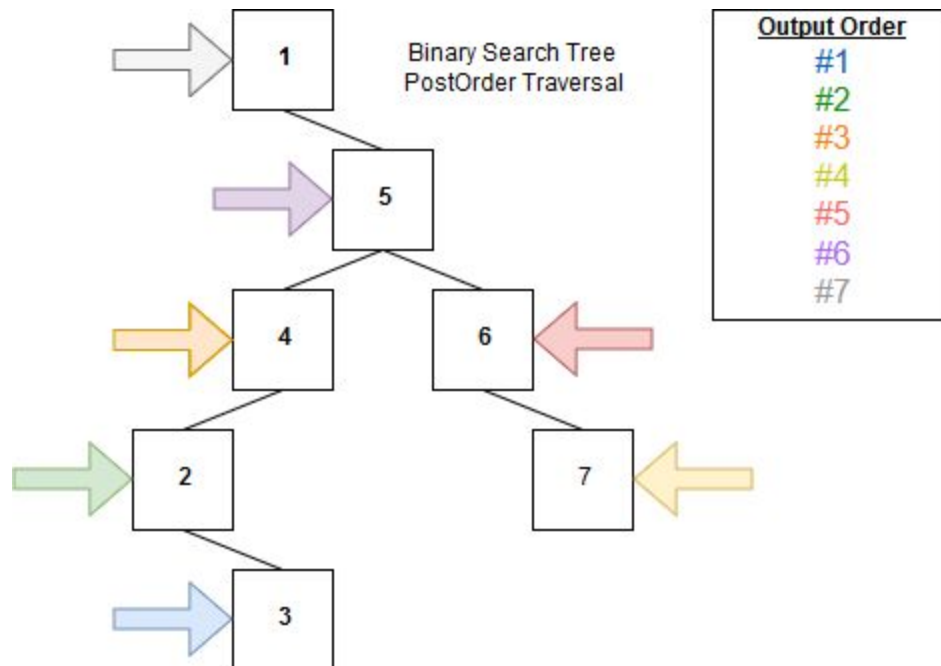
## PreOrder Diagram (1 of 2):



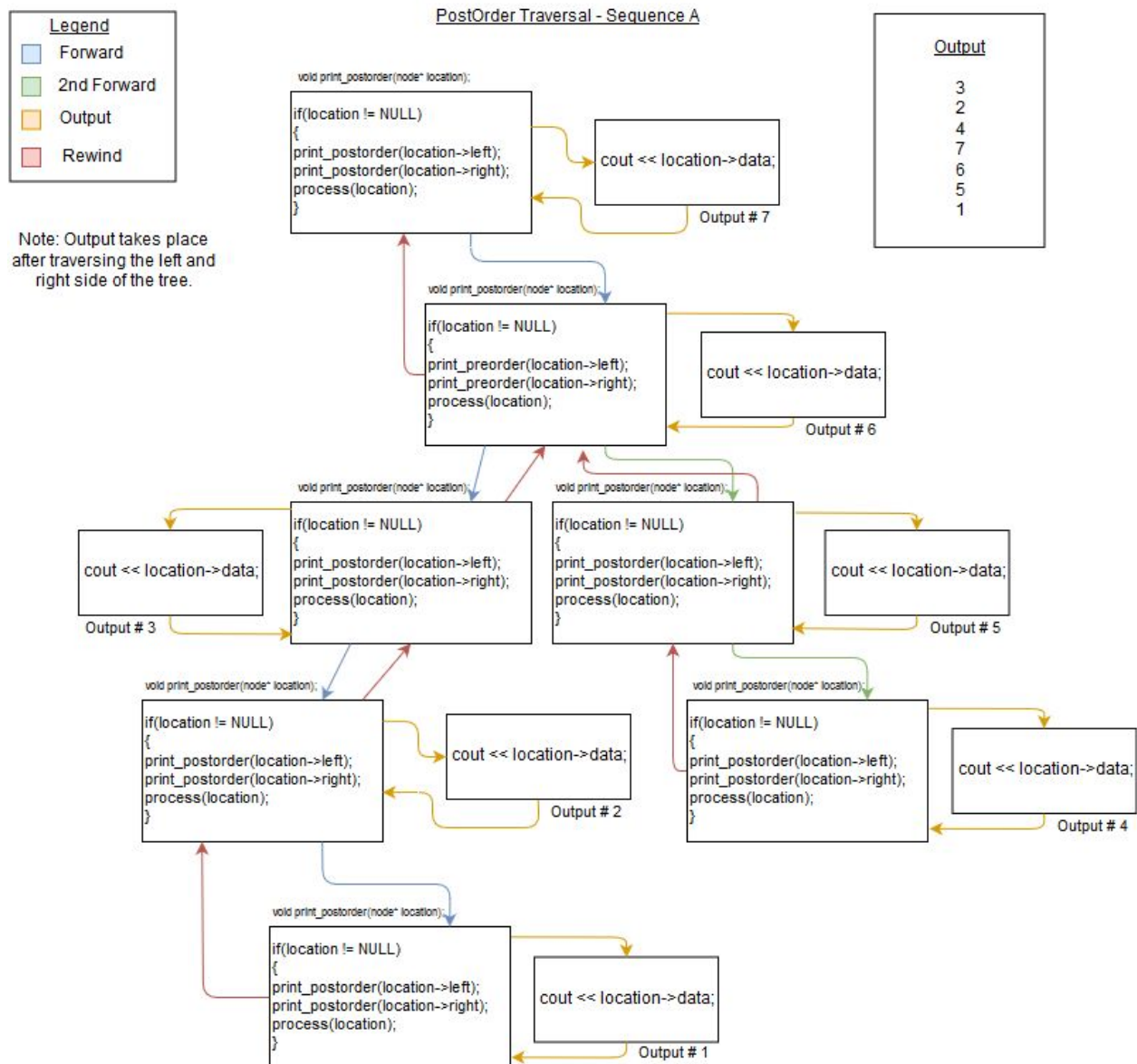
## PreOrder Diagram (2 of 2):



## PostOrder Diagram (1 of 2):



## PostOrder Diagram (2 of 2):



## Conclusion

This assignment was awesome! Binary search trees is something that we just barely talked about in my previous programming class, where they seemed quite complex. However, after implementing one I realize that all it is, is a linked list where each node has two forward links instead of one, and where the data is sorted in some manner. Looking forward to what we are going to learn next!