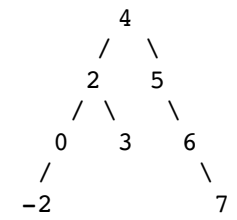
Lab 5

Andrew Crowell (05454826)

I tested my tree algorithms on the following example tree:



When I run my program, I get following results:

```
4: -2
3: 0
2: 2
3: 3
1: 4
2: 5
3: 6
4: 7
Tree height = 4
Shortest path: 2
Internal node count: 5
Internal length: 14
```

All of the algorithms take O(n) time. This is because the shortest path algorithm traverses all nodes on each subtree until it finds a leaf, the internal node count traverses all nodes on each subtree until it finds all leaf nodes, and the internal path length traverses all nodes unconditionally. These algorithms may not take any shorter time than this because it is not cutting down the region visited by each recursive call with these algorithms.

And here is my code for each of the operations. For the two algorithms that take a height parameter, height will initially be zero.

```
int BinaryTreeNodeFindShortestPath(BinaryTreeNode* this, int height)
{
	int l, r;
	if(this == NULL || (this->left == NULL && this->right == NULL))
	{
		return height;
	}
	l = BinaryTreeNodeFindShortestPath(this->left, height + 1);
	r = BinaryTreeNodeFindShortestPath(this->right, height + 1);
	if(l < r)
	{
		return l;
	}
	else
	{
		return r;
	}
}


int BinaryTreeNodeCountInternalNodes(BinaryTreeNode* this)
```

```
{
	if(this == NULL || (this->left == NULL && this->right == NULL))
	{
		return 0;
	}
	else
	{
		return BinaryTreeNodeCountInternalNodes(this->left) +
BinaryTreeNodeCountInternalNodes(this->right) + 1;
	}
}

int BinaryTreeNodeCountInternalLength(BinaryTreeNode* this, int height)
{
	if(this == NULL)
	{
		return 0;
	}
	else
	{
		return BinaryTreeNodeCountInternalLength(this->left, height + 1) +
BinaryTreeNodeCountInternalLength(this->right, height + 1) + height;
	}
}
```