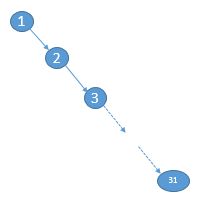
**Project 3: Optimizing a tree**

Due Date: 11:59 pm, May 4, 2018

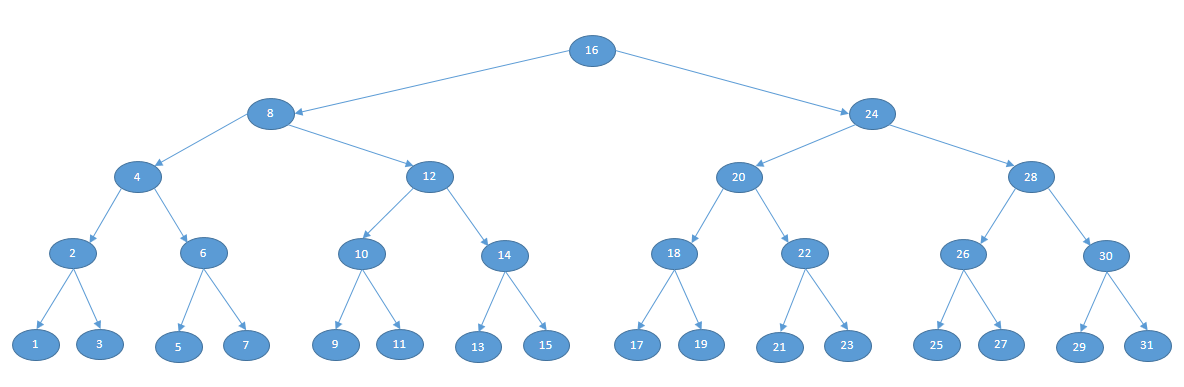
**Description**:

As you know, when you insert values to a tree, the values greater will go to the right side and the values smaller go to the left side. The tree will be extremely skewed if you insert values in the order from the smallest to the largest or from the largest to the smallest. For example, if you insert 31 values 1, 2, 3, …, 31 in the listed order, the tree will look like:



The tree becomes a list (with the depth of 30) and it is not optimized for the search.

Your task is to write a member function to optimize the tree to make the depth of the tree to be (Log2(n+1))-1 (n is the number of nodes). For example, the above tree will be optimized to:



You can see the depth of the tree becomes 4. The tree of this structure is the best for the search. Please keep it in mind: your optimizing algorithm should not be hardcoded and should be able to apply to all binary trees regardless the content of the trees.

**Instruction and Test**:

Use your assignment 8 as the base. Develop an algorithm and implement it in a public member function called Optimize() with void return. To test your code, insert following code into your main function:

Tree<int> optTree;

int depth;

for(int i=1; i<32; i++){ //insert values 1,2,3,..., 31

optTree.insertNode(i);

}

optTree.inOrderTraversal(); //you will output 1 2 3 ... 31

std::cout << std::endl;

std::cout << "The depth of the tree is: " << optTree.getDepth() << std::endl;

//you will output 30 as the depth

optTree.Optimize();

optTree.inOrderTraversal(); //you will output 1 2 3 ... 31

std::cout << std::endl;

std::cout << "The depth of the tree after optimized is: " << optTree.getDepth() << std::endl;

//the output depth will be 4