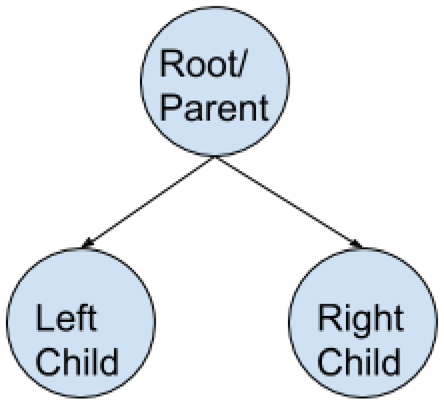
Project 3

We have been discussing binary trees, particularly the Binary Search Tree, over the last two weeks. Trees are a common class of data structures with many useful generalizations, such as the Heap data structure we began to develop in the last lecture. Complete the following practice questions to build a better intuitive understanding of binary trees and how to access the data they store. Then complete the implementation questions given in the next section.

Submission:
Non-linear data structures are often easier to visualize as diagrams rather than code. For the questions that ask for a diagram, you may draw the diagram in any medium you feel is most convenient. Submit **one PDF document** containing your answers, including diagrams, written responses, and Java code.

Concepts and Operations

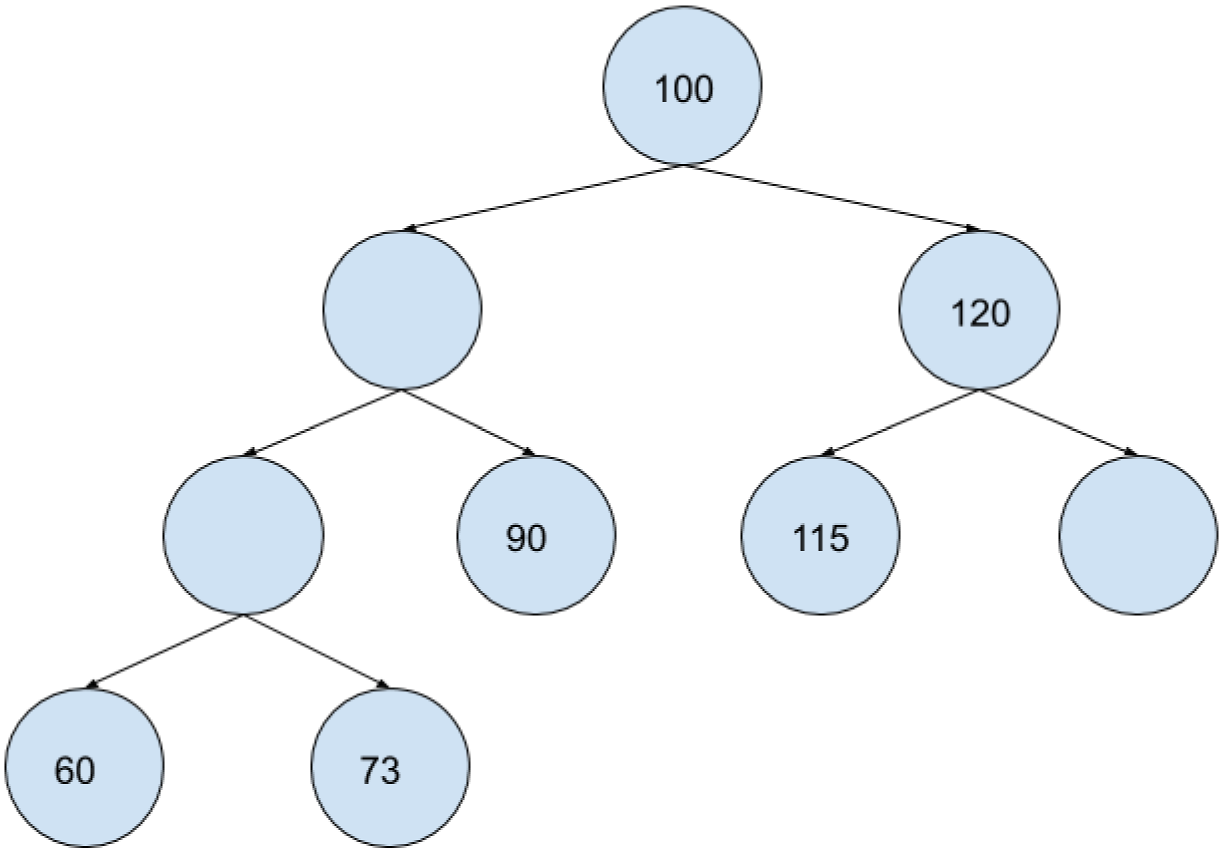
A binary tree is a collection of nodes assembled into a hierarchy. The "root" node at the top of the tree is created first, and is the only node in the hierarchy without a "parent". The "root" and all other nodes in the tree may have up to two "child" nodes connected to it. These references define the structure of the tree.



The rules we use to organize data in this hierarchy determine the properties of the data structure. In the case of the Binary Search Tree (BST) we have covered so far, we are usually storing numerical data. When data is stored in a BST, the value of the Left Child must be less than than the value of the Parent node. The value of the Right Child must be greater than or equal to the value of the Parent node.

1. Draw a binary tree with 10 nodes. Only consider the structure of the tree, using the diagram above as a reference. Disregard the values stored in the tree for this question.

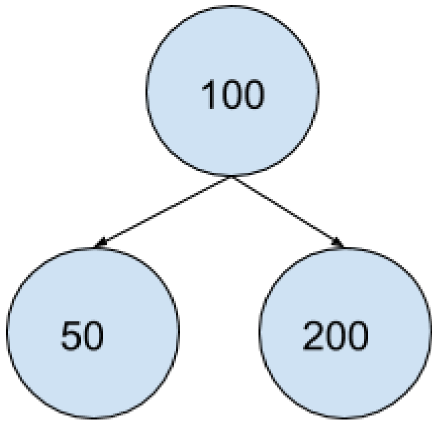
2. Complete the following Binary Search Tree, adding the values 126, 80, and 70. Use the rules described in the section above to determine where each value should be placed in the hierarchy.



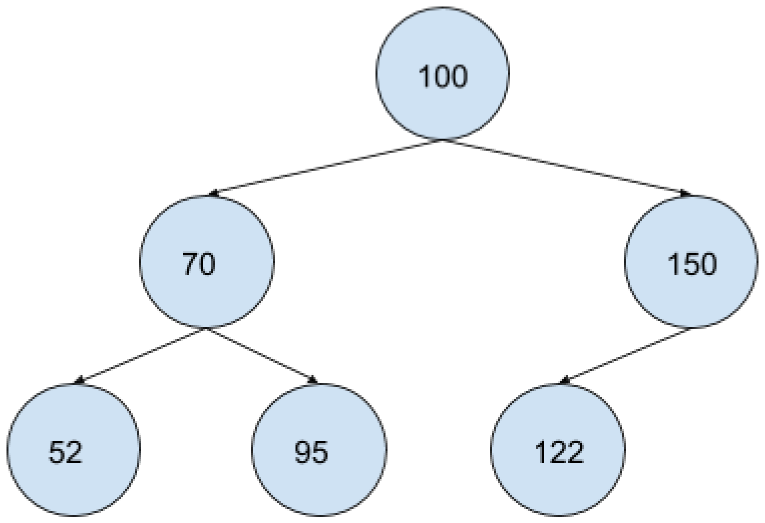
3. Draw a Binary Search Tree which stores the values 5, 8, 2, 10, and 7. Insert the values into the tree in that order. Use the rules described in the section above to determine where each value should be placed in the hierarchy.

4. Show the result of a Right Rotation performed on the following tree. In other words, rotate the following tree to the right. Remember that we spent approximately two hours

doing examples of these rotations in class, and an example implementation of this operation is given in the tree demo posted on iLearn.



4. Show the result of a Left Rotation performed on the following tree.



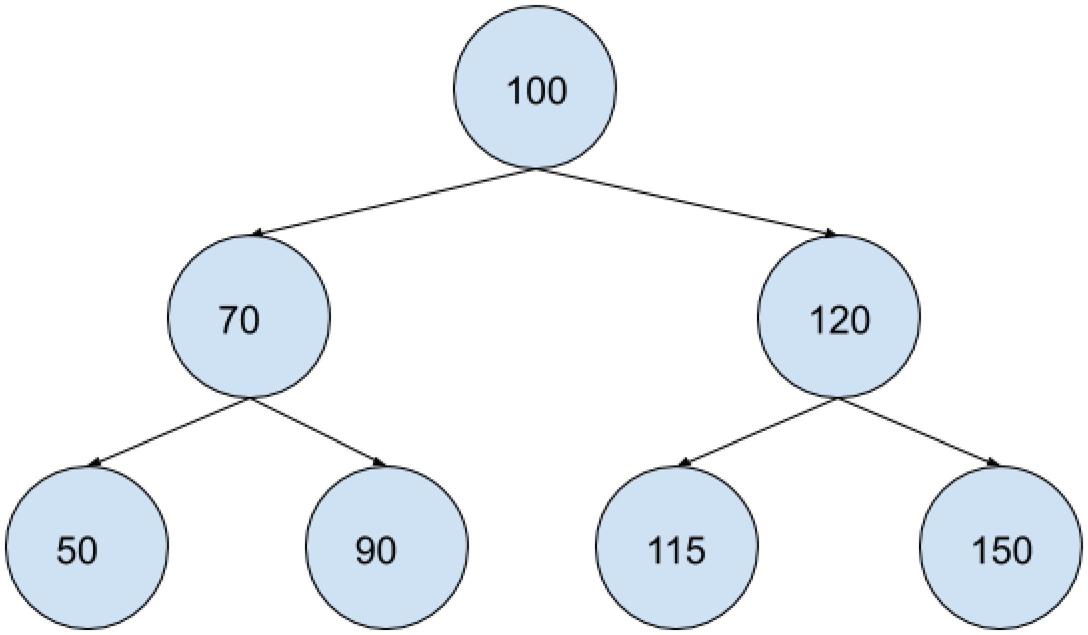
5. Show the result of a Right Rotation performed on the tree shown in question 4.

6.

a) Assume the tree shown in question 4 is rotated Left, then rotated Right. How will the resulting tree differ from the original?

b) Will the order and position of the nodes change after performing a Left rotation followed by a Right rotation?

7. Write the value of each node visited when performing a **pre-order** traversal of the following tree:



8. Write the value of each node visited when performing an **in-order** traversal of the tree above.

Implementation

Use this Node class when completing the following questions.

```
class Node {
     int data;
     Node left;
     Node right;

     Node(int value) {
          data = value;
          left = null;
          right = null;
     }
}
```

1. Using the Node class defined above, write an algorithm to insert a new Node into a Binary Search Tree. Assume the root of the tree is stored in a variable called “root”. Use the following function signature when writing your code:

```
void insert(Node newNode) {

     // your code here

}
```

2. Write a function which takes the root node of a Binary Search Tree as a parameter, and prints the values stored in the tree in order from smallest to largest.

3. Write a function which takes the Root node of a Binary Search Tree and an integer value called “target” as parameters. The function should search the tree for the node storing the “target” value and return the Node object. If the given value is not found in the tree, the function should return null.

4. Write pseudocode to perform a **left** rotation and **right** rotation on a Binary Search Tree. Assume the root node of the tree is stored in a variable named “root”. Remember that “pseudocode” is simply an less formal description of each operation needed to perform an algorithm. Each step should be clearly defined. An example of pseudocode for finding the average value in an array is given below:

```
int[] arrayOfNumbers = {5, 7, 9, 10, 12}
int sum, average = 0

For each number in arrayOfNumbers:
     sum = sum + number

average = sum / arrayOfNumbers.length
```