# Overview

In this lab, you will write methods to work within a binary search tree (BST.) You will need to use recursion to navigate the nodes in the tree. For some methods you will need to rely on the properties of a BST: that from a given node, the left subtree will contain only nodes which are lower (by key) and the right subtree will contain only nodes which are higher (by key.) Each node in the tree will have both a key (used for ordering) and a value.

# Objectives

* Practice creating a class
* Practice with BSTs (binary search trees)
* Practice with recursion
* Apply test cases to your program

# Set-up

1. Create a folder on your local machine for your Java program, you can name it whatever you like
2. Start Visual Studio Code (VS Code)
3. In VS Code, Open that newly created folder.
4. Download the starter code from the course public folder ([public/15L](https://cs.unh.edu/~cs416/public/15L)) and save it in the src directory of your new project.

# **Implementation**

The starter code includes **BinarySearchTree.java**, **Data.java**, and **TreeApp.java**. You should not change anything in **Data.java**, or **TreeApp.java** but you can run the main method of **TreeApp.java** to test your work. Your task will be to complete several methods in **BinarySearchTree.java**. For each method, you will see that there is already a helper method which calls the method you will write with some starting values.

**public Data find( Node n, String key )**

Recursively navigate through the subtree starting at n to find the data with the given key. Remember that the properties of a BST mean that data to the left of n will have keys lower than n's key and the nodes to the right will have keys higher than n's. You can use compareTo on the node's key to determine whether you should recursively call on the left child or the right child (or if n has the key itself.) You will need two base cases: when n is null or n's key matches key. The helper method is find( String key ) which calls your method with the root node to start the recursion.

**public Data maxKey( Node n )**

Recursively find the maximum key in the subtree starting at n. Remember that the properties of a BST define that data to the left of n will have keys lower than n's key and the nodes to the right will have keys higher than n's. So if n has a right child, that node's key must be greater than n's key and you should call recursively on that right child node. You will need two base cases: when n is null and when n has no right child. The helper method is maxKey() which calls your method with the root node to start the recursion.

**public Data maxValue( Node n )**

Recursively find the maximum value in the subtree starting at n. Remember that the BST is ordered on the key, so you will not be able to use the BST properties for left and right to find the max value. You will need to recursively get the max value for the left child's subtree and the right child's subtree and return the maximum of n, left, and right. Your base case will be when n is null. Its helper method is maxValue() which calls your method with the root node to start the recursion.