# Introduction

The problem we are faced with today is constructing an image with python of a BST (binary search tree), creating a new and iterative version of the search function provided to us, constructing a balanced BST from a sorted list, extracting elements from a BST into a sorted list, and printing out elements from a BST by ordered depth.

# Problem 1

This problem was the one that gave me the most trouble due to how different it was to what we have worked on recently. The route I chose to take was to use the previous code we had from lab 1, which we drew a BST. I learned 2 new things, a circle.plot function, and text plotting into the graph. Using those 2 new things, I modified my previous code to work with the T node we use to draw.

# Problem 2

This second problem was not to complicated, but it is a tricky to do if you do not understand trees a whole lot yet. The first thing I did was check if K was already in the tree, or if T was none. If either of those were true, the method would return reference to Node T. The next thing I did was create a while loop that would iterate as long as T was not None. The first case I used was an if statement that checked if K was in T.item, and would return the reference to T. The next case I did was check if K>t.item and that would cause us to iterate to the right side if true, t is set equal to t.right. If None of the if statements were true, the next thing that would happen would be an iteration to the left side that updates t to be equal to T.left.

# Problem 3

The way this problem was solved was very time consuming because it took a lot of debugging to finally get it to work. The first thing we did was check the length of the array, and made sure that the length was greater than 0. Once we check that, we begin by splitting the array in half. We had 2 if statements that checked for the len of the ordered list and checked if it was equal to 1 or to 2. In the first if statement(len = 1), we add the number to the node and we return the node. In the second if statement(len = 2), we add the number a[mid] to the node, and then we add a[0] to the node.left of the balance tree and then we return the node. Next, we update our node.item to be equal to the mid item of the sorted list. After that we we only have 2 recursive calls that update the node to go into the direction that we are updating, if we update .left, we recursively send .left, and we splice the list to send the left side of the list, excluding the mid point that we used to update our node, then we return.

# Problem 4

The first base case we have is to check if the node is none, and if it is, we return the list. The next case makes sure our node is not none and recursively allows us to iterate to the left side first, and to the right side after (this is done to get the smallest first and then the larger numbers) and we append to the list, and return.

# Problem 5