1. Create some tests (at least one per function) that you want your Binary Search Tree (BST) to pass before you start coding.

A screenshot of a computer program

Description automatically generated with medium confidence

1. Implement a binary search tree that includes:
   1. nodes to store values,

A computer code on a black background

Description automatically generated with low confidence

* 1. an add function that adds a new value in the appropriate location based on our ordering rules,  
     (I likely used less than or equal to going to the left and greater than values going to the right)

A screen shot of a computer code

Description automatically generated with low confidence

* 1. a remove function that finds and removes a value and then picks an appropriate replacement node  
     (successor is a term often used for this)

A screen shot of a computer program

Description automatically generated with medium confidence

* 1. we have at least one tree traversal function (I recommend starting with an in-order traversal!)  
     **Bonus** if you implement the three common traversals (pre-order, post-order, in-order)  
     **More Bonus** if you also include a breadth-first traversal (sometimes called a level-order search)

A screen shot of a computer code

Description automatically generated with low confidence

1. Analyze and compare the complexity of insert and search as compared to a binary tree without any order in its nodes.

An Insert without any order in the nodes could have as bad as a O(n) runtime complexity where as a BST that organizes itself would be a O(log(N)) runtime complexity. The same thing goes for search because of its ordering.