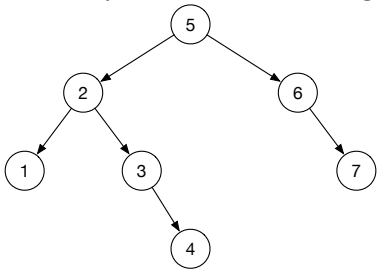
CS241
Homework 3

1. In class we studied the properties of selection sort, bubble sort, and insertion sort for various starting configurations of elements in the array. For instance, we examined how the algorithms would operate if given an array that was already sorted or an array sorted in reverse order.

   Suppose we want to sort an array of length *n* that has the following starting configuration: it always begins with a sequence of *n*-1 sorted numbers, but then ends with one that is less than all the previous numbers. For example, an array that looks like this would be [2 3 4 5 1].

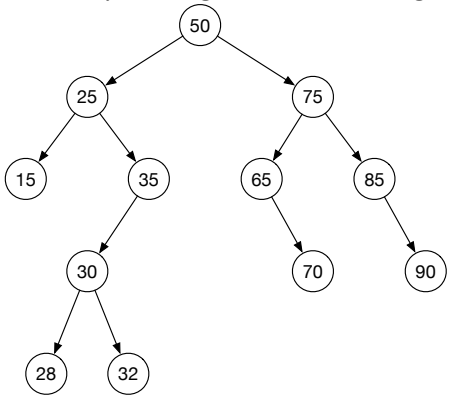
   Describe the performance of selection sort, bubble sort, and insertion sort on an array of this type. Tell me explicitly, in terms of big-oh, the overall running time, and then break this down into the big-oh for the number of comparisons each algorithm makes, and the number of swaps each algorithm makes. Explain your reasoning for each part (this will get you partial credit of your big-oh times are wrong).

2. Assume you have the following binary search tree:



   (a) List the order of the nodes for a preorder traversal.
   (b) List the order of the nodes for an inorder traversal.
   (c) List the order of the nodes for a postorder traversal.

3. Assume you are given the following binary search tree:



   Show what the tree looks like after each of the following sequences of operations (draw the tree only after the last operation in each sequence).
   RESET THE TREE BACK TO THE DRAWING ABOVE AT THE BEGINNING OF EACH SEQUENCE.
   For deleting a node with two children, replace with the *inorder successor*.

   (a) Insert 11, Insert 64, Insert 31, Insert 80, Insert 38, Insert 37, Insert 39
   (b) Delete 35
   (c) Delete 25
   (d) Delete 50

4. Suppose a friend tells you they have a binary search tree with the following preorder traversal: 14, 7, 3, 10, 20, 16, 17, 22. Reconstruct the entire BST from this traversal and draw it. Hint: given a preorder traversal, how do you identify the root?

5. Suppose you have a *sorted* array A with n integers in it, indexed from 0 to n-1.

   (a) Describe an algorithm that inserts the elements of array A into an empty binary search tree that is guaranteed to result in the tree being as "deep" as possible. (In other words, describe an algorithm that inserts the elements from A in such an order as to maximize the longest possible path from the root to any leaf node.) Note that this tree will be very unbalanced!

   (b) Describe an algorithm that inserts the elements of A into an empty binary tree that is guaranteed to result in the tree being as "shallow" or "bushy" as possible. That means that the longest path from the root to any leaf node should be minimized. This tree will be very balanced!

   Hint for (b): You may assume that the number of elements in array A is exactly a power of 2 minus 1 if that makes your life easier.

   Hint for (a) and (b): These algorithms are not complicated, and do not have to be written formally. Just describe an ordering of the elements in array A that makes the tree as unbalanced as possible (part (a)) and as balanced as possible (part (b)).

6. Suppose we wish to define a function to calculate the depth of a specific node in a binary tree (not necessarily a BST). Recall the depth of a node is the number of edges (connections between nodes) from the root to the node in question. This function will be very similar to the height function we wrote in class.

   Suppose we have our regular Node class:

```
public class Node {
    int key;
    Node left, right;
}
```

   And a BinaryTree class:

```
public class BinaryTree {
   private Node root;
}
```

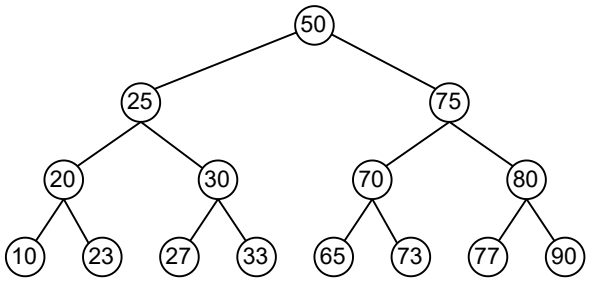
   (a) Fill in the space in the code below to complete the function. Assume both of these functions live inside the BinaryTree class so you have access to the root variable.

```
// return the depth of the node labeled with the searchKey variable.
// if searchKey doesn't exist, return -1 (as an error code).
public int depth(int searchKey) {
   return depth(root, searchKey)
}

private int depth(Node curr, int searchKey) {
   // your code here
}
```

   (b) What is the (worst-case) big-oh of your function? Briefly justify your answer.

7. Recall that we can use a binary search tree to represent a set of items. For instance, the set of integers {10, 20, 23, 25, 27, 30, 33, 50, 65, 70, 73, 75, 77, 80, 90} might be represented by the BST:



In class, we saw how to implement the common set operations "add" (add a new item to the set), "contains" (test if an item is in the set or not), and "remove" (delete an item from the set) as operations/methods on BSTs. Suppose we wish to add another set operation, that we will call "countBetween" that counts the number of items in the set that are between two specified endpoint numbers (inclusive of the two endpoints). For instance, calling countBetween(30, 68) on the set above would return 4 since there are 4 numbers between 30 and 68, namely {30, 33, 50, 65}.

Suppose we have our regular Node class:

```
public class Node {
   int key;
   Node left, right;
}
```

and a BST class:

```
public class BST {
   private Node root;
}
```

(a) Fill in the space in the code below to complete the function. Assume both of these functions live inside the BST class so you have access to the root variable.

```
// return the depth of the node labeled with the searchKey variable.
// if searchKey doesn't exist, return -1 (as an error code).
public int countBetween(int low, int high) {
   return depth(root, searchKey)
}

private int depth(Node curr, int low, int high) {
   // your code here
}
```

(b) What is the (worst-case) big-oh of your function? Briefly justify your answer. You may assume the BST is balanced.