1. (5 minutes) Consider the following tree:

```
A
/|
B D
/|
E G
```

a. What is the in-order traversal of this tree? B G D A E C H F J I
b. What is the pre-order traversal of this tree? A B D G C E F H I J
c. What is the post-order traversal of this tree? G D B E H J I F C A

2. (5 Minutes) Given the following traversals, construct the general binary tree and list the post-order traversal.

In-order: XLKMA
Pre-order: KXLAM
Answer:
Post-order: LXMAK
3. **(5 Minutes) Complete vs. Full Binary Trees**
   a. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes at the bottom-most level are as far left as possible. What is the minimum and maximum number of nodes in a complete binary tree with a height of 5?
      \[ \text{Min} = 2^\text{height} = 32 \text{ nodes} \]
      \[ \text{Max} = 2^{\text{height}+1} - 1 = 63 \text{ nodes} \]
   b. A full binary tree is a binary tree in which every node has either two or none children.
      What is the minimum and maximum number of nodes in a binary tree with a height of 4?
      \[ \text{Min} = 2 \times \text{height} + 1 = 9 \text{ nodes} \]
      \[ \text{Max} = 2^{\text{height}+1} - 1 = 31 \text{ nodes} \]

4. **(20 Minutes) Binary Search Trees:**
   A binary search tree is a tree in which for each node, all the nodes in the left subtree are smaller than the node and all the elements in the right subtree are larger than the node. Using a binary search tree of integers:
   a. **(5 Minutes)** Using the algorithm mentioned in class, create a binary search tree by inserting integers in the following order: \([6, 7, 8, 2, 4, 3, 5, 1]\). After creating the tree, delete element 8, then 4, then 6. What does the new tree look like? Is this new tree complete? Is it full?

   **Original tree**
   ```plaintext```
   6
   /   \
  2     7
 / |  \
1  4  8
 |   |
3   5
```

   **Tree after Deletion**
   ```plaintext```
   5
   /   \
  2     7
 / |  \
1  3   
```

   After deletion, the tree became a complete binary tree that isn’t full.

   b. **(5 Minutes)** What if you inserted elements in the following order:
[8, 7, 6, 5, 4, 3, 2, 1]? What happens to the tree? What is the order of complexity for inserting an element into this type of tree? What is the order of complexity for finding and removing an element?

The tree would be degenerated and perform like a linked list.
8->7->6->5->4->3->2->1

Inserting: O(n)
Since the numbers are sorted in decreasing order, you will have to visit every node if an element smaller than 1 were to be inserted.

Removing: O(n)
Finding the element to remove will take O(n) time to search, and removing will also take O(n) since the binary search tree needs to retain a valid structure.

c. (10 Minutes) Given an array arr[] of n integers, find the maximum height of the binary tree created from the array. You are given a node class with its left and right references alongside a method insert(node n, int value) which you can assume properly inserts a node into the tree and returns the original node. Hint: Traverse the array from both ends and compare the heights.

```java
public static int maxDepth(node n) {
    if (n == null)
        return 0;
    else {
        int leftDepth = maxDepth(n.left);
        int rightDepth = maxDepth(n.right);

        if (leftDepth > rightDepth) {
            return (leftDepth + 1);
        } else {
            return (rightDepth + 1);
        }
    }
}

public static int maxHeight(int arr[], int n) {
    node root1 = null;
    root1 = insert(root1, arr[0]);
    for (int i = 1; i < n; i++)
        root1 = insert(root1, arr[i]);
```
5. **(10 Minutes)** Write methods for the `BinarySearchTree` class that find the lowest common ancestor (LCA) of two numbers within the tree. Find the smallest integer that both given integers descend from. For example, given this tree:

```
20
/  \
10 30
/  /  \
5 15 25 35
```

- The LCA of 5 and 15 is 10.
- The LCA of 5 and 10 is 10.
- The LCA of 10 and 30 is 20.
- The LCA of 20 and 20 is 20.

You have access to the tree’s root, and can use the `BTNode` methods; `getData()`, `getRight()`, and `getLeft()`.

```java
public static BTNode returnLCA(BTNode root, int i, int j){
    if(root==null){
        return null;
    }
    if(root.getData()>Math.max(i, j)){
        return returnLCA(root.getLeft(),i,j);
    }
    else if(root.getData()<Double.min(i,j)){
        return returnLCA(root.getRight(),i,j);
    }
    return root;
}
```
6. (10 Minutes) Given a Node Class with references to its left and right Nodes, write a method (mirror(Node root)) and a helper method (swap(Node root)) to convert a binary tree to the mirror of itself.

```java
public static void swap(Node root)
{
    if(root == null)
        return;

    Node temp = root.left;
    root.left = root.right;
    root.right = temp;
}

public static void mirror(Node root)
{
    if(root == null)
        return;

    mirror(root.left);
    mirror(root.right);
    swap(root);
}
```