1. **(7 marks)** Consider the following binary tree:

```
      q
     / \  
m   / \  
 d  k   j
 / \  /  
b r p w
```

a) **(2 marks)** Is this tree a binary search tree? Explain your answer.

**Answer:** No, it is not a binary search tree – observe that the binary search tree property (for a tree rooted at node $x$ with value $v$, nodes in the left subtree of $x$ have values $\leq v$ and nodes in the right subtree of $x$ have values $\geq v$) is violated at nodes $m$ and $t$.

b) **(1 mark)** List the ancestor(s) of node $k$.

**Answer:** $m, q$

c) **(1 mark)** List the descendant(s) of node $m$.

**Answer:** $d, k, b, r, p, n$

d) **(1 mark)** What is the depth of the tree?

**Answer:** 4

e) **(2 marks)** Give the order in which the nodes are visited by a preorder traversal.

**Answer:** $q, m, d, b, r, k, p, n, t, j, y, w$
2. (4 marks) Draw a diagram of the binary search tree created by adding the following values in the order given to an initially empty tree:

```
26 10 30 27 8 19 18 21 25 6
```

![Binary Search Tree Diagram]

3. (9 marks) Write a class `BTNode` Java method

```java
public static double subTInv(BTNode root)
```

which, when called with a reference to the root of a binary tree, returns the quantity `1.0/(WL * 1.0)`, where `WL` is the number of nodes in the tree with null left subtrees. For example, the value returned for the tree in Question 1 is `1.0/6.0 = 0.16666...`

**Hint:** You may find it useful to break this into two methods, one of which does a recursive traversal of the tree.

**Answer:**

```java
public static double subTInv(BTNode root){
    return(1.0/(1.0 * WL(root)));
}

public static int WL(BTNode root){
    if (root == null)
        return(0);
    else {
        if (root.left == null)
            return(1 + WL(root.right));
        else
            return(WL(root.left) + WL(root.right));
    }
}
```