CSC 212 MT2	Name:	ID:

Question 1

a) Write a static method **replace** (user of ADT) that takes as input a stack **st** and two elements **e1** and **e2**. The method replaces **all** the occurrences of the element e1 in st with e2. Use the method **equals** to test for equality.

```
public static <T> void replace(LinkedStack<T> st, T e1, T e2)
{









}
```

b) Write the method **replace**, member of the class LinkedStack, which takes as input two elements **e1, e2** and replaces **all** the occurrences of the element e1 in the stack with e2. Use the method **equals** totest for equality. (**Do not call any method when writing this method.)**

```
public class LinkedStack<T>{
    private Node<T>top;

    public void replace(T e1, T e2)
    {

    }
}
```

Question 2

a) Write a recursive method that returns the **maximum key** value in a **BST**.

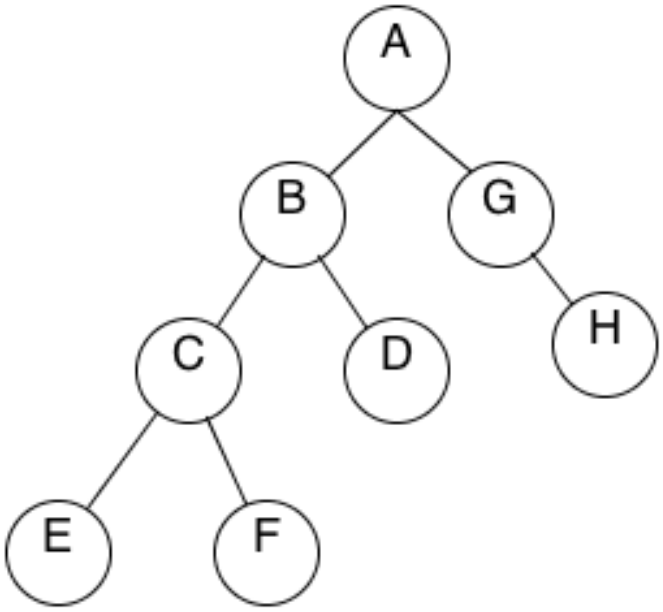
```
private int maxKey(BSTNode<T> n)
{









}
```

b) Write a recursive method that counts the number of **leaf nodes** in a **Binary Tree**.

```
private int countLeafs(BTNode<T> n)
{









}
```

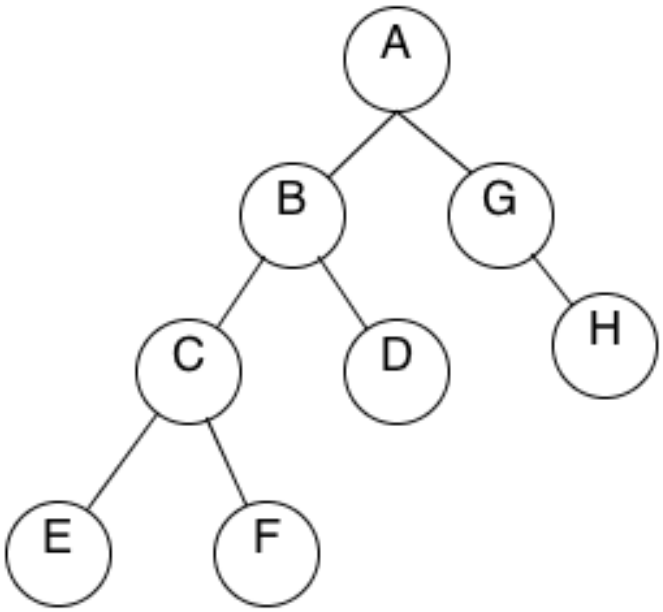
Question 3

a) Write a main method that uses the class **BT** to create the **Binary Tree** represented below. **The values must be inserted in the following order: A, B, C, D, E, F, G, H**




```
public static void main(String[] args){
      BT<String> tree = new BT<String>();









}
```

b) Using the **Binary Tree** represented below, write the **order of traversal** using **Preorder** and **Inorder** methods

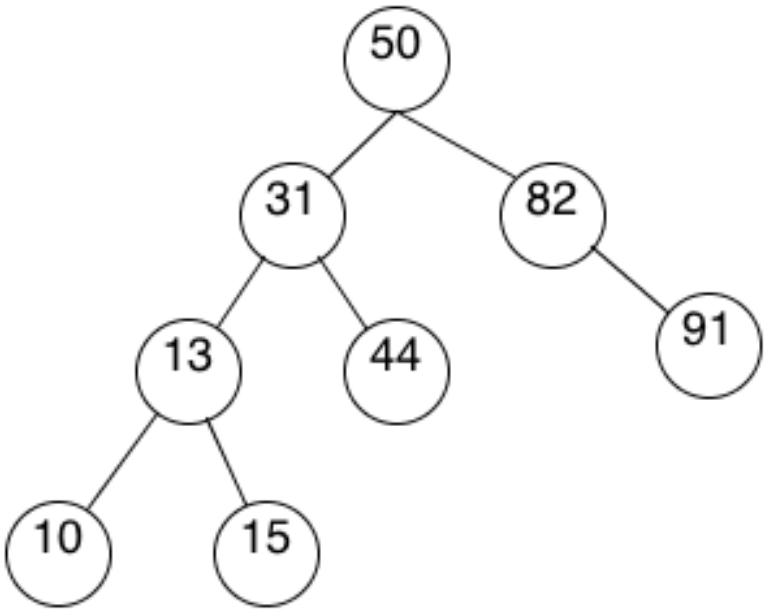


Preorder

Inorder

Question 4

a) Insert the following keys into the **Binary Search Tree (BST)** below, assuming they **arrive from left to right: 84, 70, 85, 35, 49, 20, 10**

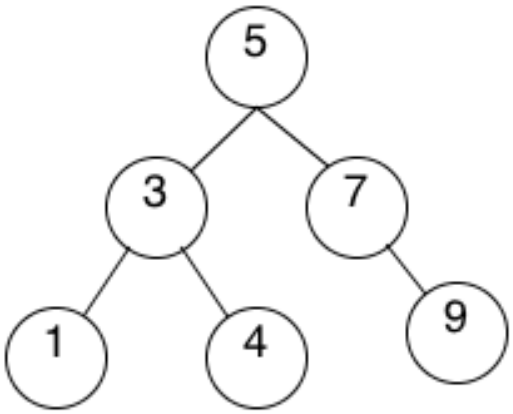
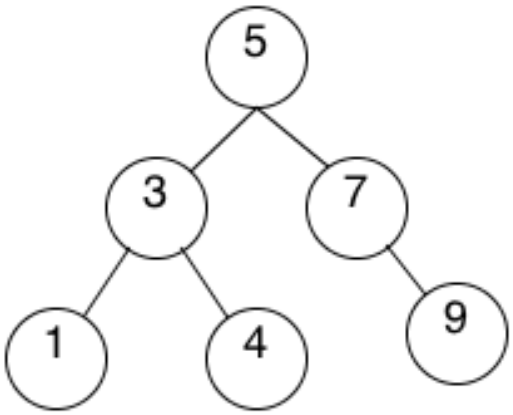
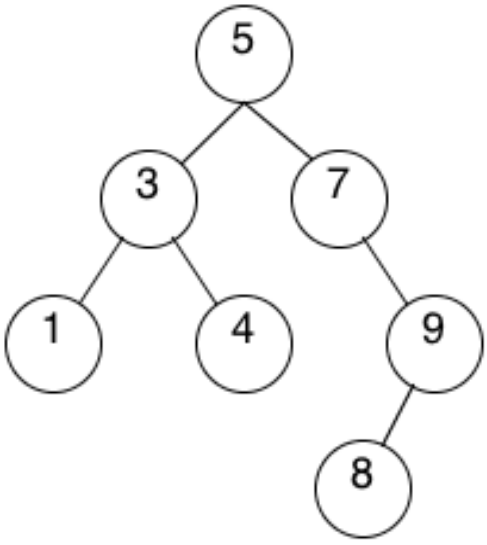


Insert 84	**Insert 70**

Insert 85	**Insert 35**
Insert 49	**Insert 20**

Insert 10

b) From each one of the **Binary Search Tree (BST)** below, delete the specified key and redraw the tree next to it after the deletion.

Delete 4 	
Delete 5 	
Delete 7 	

ADT Stack Specification

All operations operate on a stack S.

1. Method **Push** (Type e)
   requires: Stack S is not full.
   input: Type e.
   results: Element e is added to the stack as its most recently added elements.
   output: none.
2. Method **Pop** (Type e)
   requires: Stack S is not empty.
   input: none.
   results: the most recently arrived element in S is removed and its value assigned to e.
   output: Type e.
3. Method **Empty** (boolean flag)
   requires: none
   input: none.
   results: If Stack S is empty then flag is true, otherwise false.
   output: flag.
4. Method **Full** (boolean flag).
   requires: none
   input: none.
   results: If S is full then Full is true, otherwise Full is false.
   output: flag.

ADT Binary Tree Specification

1. Method **Insert** (Type e, Relative rel, boolean inserted)
   (Relative = {leftchild, rightchild, root, parent})
   requires: (1) Full () is false and (2) either (a) rel = root and Empty() is true or (b) rel <> root and rel <> parent and Empty( ) is false.
   input: e, rel.
   results: if case (1) rel = leftChild, current node has a left child, or (2) rel = rightChild, current node has a right child, then inserted is false. Else a node containing e is added as rel of the current node in the tree and becomes the current node and inserted is true.
   output: inserted.
2. Method **DeleteSub** ()
   requires: Binary tree is not empty.
   input: none

results: The subtree whose root node was the current node is deleted from the tree. If the resulting tree is not empty, then the root node is the current node.
output: none.

3. **Method** Update (Type e).
   requires: Binary tree is not empty.
   input: e.
   results: the element in e is copied into the current node.
   output: none.
4. Method **Retrieve** (Type e)
   requires: Binary tree is not empty.
   input: none
   results: element in the current node is copied into e.
   output: e.
5. Method **Find** (Relative rel, boolean found)
   requires: Binary tree is not empty.
   input: rel.
   results: The current node of the tree is determined by the value of rel and previous current node..
   output: found.
6. Method **Empty** (boolean empty).
   requires: None.
   input: none
   results: If Binary tree is empty then empty is true; otherwise empty is false.
   output: empty.
7. Method **Full** (boolean full)
   requires: None.
   input: None.
   results: if the binary tree is full then full is true otherwise false.
   output: full.