Binary Search Tree

1. BST definition: For every node x, x's _____ has smaller values than key in x, and x's _____ has greater values than key in x.

2. Does it say anything about being balanced?
In the worst case, BST is the same as a _____.

Let's talk about BST implementation.

1. Binary Search Tree with `BinaryNode<E>` What would be the requirement of type E?

2. _____ interface

```
public interface _____<T> {
    public int compareTo(T o);
}
```

```
public class BinarySearchTree<E extends _____> {

    private BinaryNode<E> overallRoot;

    public BinarySearchTree(){
        overallRoot = null;
    }
    …
}
```

3. Which one should we use?

```
public class BinarySearchTree<E extends Comparable<E>>
public class BinarySearchTree<E extends Comparable<? super E>>
```

4. Java Generics: <? `extends` E> vs <? `super` E>

1) <? `extends` E>: The type can be either _____ or any _____ of E.

Ex) Given, `List<? extends Number> foo;` and `Integer` and `Double` are subclasses of `Number`, then below are legal assignments

```
foo = new ArrayList<          >();
foo = new ArrayList<          >();
foo = new ArrayList<          >();
```

2) <? `super` E>: The type can be either _____ or any _____ of E.

Ex) Given, `List<? super Integer> bar;` and `Integer` is a subclass of `Number` and `Object`, below are legal assignments

```
bar = new ArrayList<        >();
bar = new ArrayList<        >();
bar = new ArrayList<        >();
```

5. Java Example
   LocalDate implements Comparable<ChronoLocalDate> and ChronoLocalDate

Class LocalDate

- java.lang.Object
  - java.time.LocalDate

All Implemented Interfaces:
Serializable, Comparable<ChronoLocalDate>, ChronoLocalDate, Temporal, TemporalAccessor, TemporalAdjuster
Note LocalDate does NOT implement Comparable<LocalDate>.

```
//sort collection c where each element T implements Comparable<T>
public static <T extends Comparable<T>> void sort(Collection<T> c){ … }

List<LocalDate> list = new ArrayList<>();
sort(list); //Will this compile? How to fix the problem?
```

6. Likewise we want to keep our code flexible.

```
public class BinarySearchTree<E extends Comparable<? super E>>
```

7. find

```
public boolean find(E targetData) {
    return find(targetData, overallRoot);
}
private boolean find(E targetData, BinaryNode<E> root) {
    //base case
    if (root == null) return false;
    //search hit
     if (targetData.compareTo(root.data) == 0){
      return true;
     }
    // search for the target in either left or right subtree
    //when to search left subtree? When to search right subtree?









}
```

Time complexity?

8. Insert

```
//insert: modifying existing tree: x = change(x); public
void insert(E targetData) {
    BinaryNode<E> newNode = new BinaryNode<>(targetData);
    overallRoot = insert(newNode, overallRoot);
}

//helper method that will insert target into the right location
//return the new subtree with the target node inserted
private BinaryNode<E> insert(BinaryNode<E> target, BinaryNode<E> root) {
     if (root == null) { //base case
        root = target; insert target to current root
    }
    //if not inserting to current root,
    //need to insert to either left/right subtree
    //when to insert left subtree? When to insert right subtree?









}
```

Time complexity?

9. Delete.

What are the possible 3 cases?

- Case 1: Delete is a _____ node ○ Set the node to null
- Case 2: Delete an internal node that has _____ left subtree or right subtree ○ Replace the node to delete with its non-null child
- Case 3: Delete an internal node that has _____ left subtree and right subtree ○ How to preserve the BST property?

```
public void remove(E targetData) {
    overallRoot = remove(targetData, overallRoot);
}

//removes a node that has targetData and returns the subtree
//by recursion
private BinaryNode<E> remove(E targetData, BinaryNode<E> node) {
    //base-case
    if (node == null) return null;
    if (targetData.compareTo(node.data) < 0) { //targetData is smaller than current data
        //remove from my left subtree
        node.left = remove(targetData, node.left);
```

```
    } else if (targetData.compareTo(node.data) > 0) {//target is greater
        //remove from my right subtree
        node.right = remove(targetData, node.right);
    } else { //data found!!! node's data is the target itself
        //case 1: node is leaf (remove node)
        if (node.left == null && node.right == null) {
            //implement me


        } else if (node.left == null) { //case 2: only left child is null
            //implement me


        } else if (node.right == null) { //case 3: only right child is null
            //implement me


        } else { //case 4: both are not null
            //option 1: pick the max from the left subtree
            //         (right subtree remains unchanged)
            //implement me









        }
    }
    return node;
}

//assumption: root is not null
private E findMax(BinaryNode<E> root){
    while(root.right!= null){
        root = root.right;
    }
    //root.right is null (current root is the right most node)
    return root.data;
}
```

Binary Tree Analysis

0. _____ of a node refers to the distance of the node from the root, measured in the number of e_____.

1. **_____ binary tree** a binary tree in which all the levels are completely filled except possibly the lowest (deepest), which is filled from the left.
[Draw some examples]

1) How many nodes at level k (where k is not the lowest level)?
2) Min number of nodes at lowest level?
3) Max number of nodes at lowest level, where k is the height of the tree?
4) Assume, all the levels are completely filled INCLUDING the lowest level What is the total number of nodes in this tree with k levels?

5) Given n is the number of nodes in a complete binary tree, what is the number of levels? (Hint: the closest power of 2 that is >= n)

2. **_____ binary tree** a binary tree in which every parent node/internal node has either two or no children. [examples]