18.9a Here I used recursion to check for the right and left section. If they were both null we returned the leaveCounter++

```
public static <AnyType> int numberOfLeaves(BinaryNode<AnyType> t) {
    if (t.getRight()!=null) {
        numberOfLeaves(t.getRight());
    } else if (t.getLeft()!=null){
        numberOfLeaves(t.getLeft());
    }

    if (t.getRight()==null && t.getLeft()==null) {
        leaveCounter++;
    }

    return leaveCounter;
}
```

18.9b Here I did the exact same thing as 18.9a but I just changed to have it check if either the left one was null or the right one was null

```
35      public static <AnyType> int numberOfNodesWithOneNonNullChild(BinaryNode<AnyType> t) {
36          if (t.getRight()!=null && t.getLeft()==null){
37              oneNonNullCounter++;
38              return numberOfNodesWithOneNonNullChild(t.getRight());
39          } else if (t.getRight()==null && t.getLeft()!=null){
40              oneNonNullCounter++;
41              return numberOfNodesWithOneNonNullChild(t.getLeft());
42          } else if (t.getRight()==null && t.getLeft()==null) {
43              return 0;
44          } else {
45              numberOfNodesWithOneNonNullChild(t.getLeft());
46              numberOfNodesWithOneNonNullChild(t.getRight());
47              return oneNonNullCounter;
48          }
49      }
```

18.9c Here also I just modified my code to check if both of the children were null and then added one to the counter if they were.

```
    public static <AnyType> int numberOfNodesWithTwoNonNullChildren(BinaryNode<AnyType> t) {
        if (t.getRight()!=null && t.getLeft()==null){
            oneNonNullCounter++;
            return numberOfNodesWithOneNonNullChild(t.getRight());
        } else if (t.getRight()==null && t.getLeft()!=null){
            oneNonNullCounter++;
            return numberOfNodesWithOneNonNullChild(t.getLeft());
        } else if (t.getRight()==null && t.getLeft()==null) {
            return 0;
        } else {
            numberOfNodesWithOneNonNullChild(t.getLeft());
            numberOfNodesWithOneNonNullChild(t.getRight());
            return oneNonNullCounter;
        }
```

18.10a I utitilized very similar code once again but this time I checked to see if the data in the node was evenly divisible by 2. If it was we added one to the counter. If it wasn't we moved on.

```java
public static int numberOfNodesWithEvenDataItems(BinaryNode<Integer> t) {
    if (t.getRight()!=null) {
        sumOfAllItems(t.getRight());
    }
    if (t.getLeft()!=null) {
        sumOfAllItems(t.getLeft());
    }
    if (t.getElement()%2==0) {
        evenDataCount++;
    }
    return evenDataCount;
}
```

18.10b Here we just recursively went through each node and added its element to the runningSum variable. At the end we returned the total amount.

```java
public static int sumOfAllItems(BinaryNode<Integer> t) {
    if (t.getRight()!=null) {
        sumOfAllItems(t.getRight());
    }
    if (t.getLeft()!=null) {
        sumOfAllItems(t.getLeft());
    }
    runningSum = runningSum + t.getElement();
    return runningSum;
}
```

Because I utilized all the same code, all these should run in just n time.

Also, I seemed to run into an issue with the tester? I cant quite figure out what is going on so I may message the professor and see what's wrong with my methods. I believe the thought process behind all of them should work so I am not sure what is wrong.