Section 03: Asymptotic Analysis

Section Problems

1. Binary Search Trees

(a) Write a method `validate` to validate a BST. Although the basic algorithm can be converted to any data structure and work in any format, if it helps, you may write this method for the `IntTree` class:

```
public class IntTree {
    private IntTreeNode overallRoot;

    // constructors and other methods omitted for clarity

    private class IntTreeNode {
        public int data;
        public IntTreeNode left;
        public IntTreeNode right;

        // constructors omitted for clarity
    }
}
```

(b) Suppose we want to implement a method `findNode(V value)` that searches a binary search tree with n nodes for a given value.

(i) What is the worst case big-Θ runtime for `findNode`? Draw an example of a binary search tree with up to 4 nodes that would result in this worst-case runtime.

(ii) What is the best case big-Θ runtime for `findNode`? Draw an example of a binary search tree with up to 4 nodes that would result in this best-case runtime.

2. Code Analysis

For each of the following code blocks, what is the worst-case runtime? Give a big-Θ bound.

(a)
```
public List<String> repeat(List<String> list, int n) {
    List<String> result = new LinkedList<String>();
    for(String str : list) {
        for(int i = 0; i < n; i++) {
            result.add(str);
        }
    }
    return result;
}
```

(b)
```
public int num(int n){
    if (n < 10) {
        return n;
    } else if (n < 1000) {
        return num(n - 2);
    } else {
        return num(n / 2);
    }
}
```

(c)
```
public int foo(int n) {
    if (n <= 0) {
        return 3;
    }
    int x = foo(n - 1);
    System.out.println("hello");
    x += foo(n - 1);
    return x;
}
```

(d)
```
public boolean isPrime(int n) {
    int toTest = 2;
    while (toTest < n) {
        if (n % toTest == 0) {
            return false;
        } else {
            toTest++;
        }
     }
     return true;
}
```

3. "Tree method" walk-through

Consider the following method:

```java
public int A(int n) {
  if (n <= 1) {
    System.out.println("done!");
    return 10;
  }
  for (int i = 0; i < n; i++) {
    System.out.println("not done yet...");
  }
  return A(n/2) + A(n/2);
}
```

We want to find an *exact* closed form of this method by using the tree method. Suppose we draw out the total work done by this method as a tree, as discussed in lecture. Let n be the initial input to A.

(a) Draw a tree representing **the total number of calls** to A for $n = 8$.

(b) Consider the tree that would be generated representing A called with any arbitrary n.

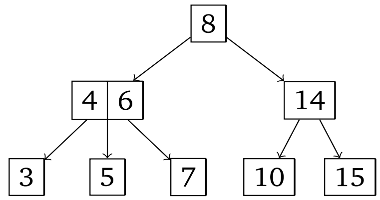
(i) What is the total amount of non-recursive work done at each **level** of the tree? Refer to the example you drew in the previous question to help you. Give your answer as an expression in terms of n.

(ii) What is the height of the tree? Give your answer as an expression in terms of n.

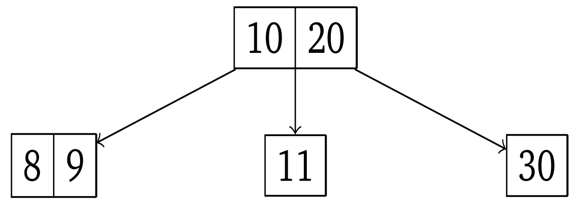
(iii) Give a big-Θ runtime for A given an arbitrary n. Hint: use your answers from the previous parts to help you solve this question.

4. B-Trees

(a) Draw what the following 2-3 tree would look like after inserting 18, 38, 12, 13, and 20.



(b) Given the following initial 2-3-4 tree, draw the result of performing each operation.



(i) Insert 5 into this tree.

(ii) Insert 7 into the resulting tree.

(iii) Insert 12 into the resulting tree.

(c) Suppose the keys 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 are inserted sequentially into an initially empty 2-3-4 tree. Which insertions cause a split to take place?