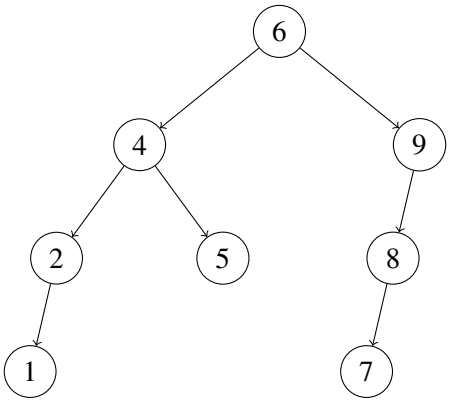
CS 61B
Summer 2020

Small Group Tutoring
Section 6 Trees, Stacks, and Queues

Worksheet 6

1 Tree-versal



a) What is the pre-order traversal of the tree?

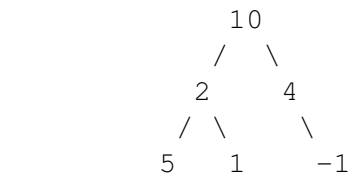
b) What is the post-order traversal of the tree?

c) What is the in-order traversal of the tree?

d) What is the breadth-first traversal of the tree?

2 Sum Paths

Define a root-to-leaf path as a sequence of nodes from the root of a tree to one of its leaves. Write a method `printSumPaths(TreeNode T, int k)` that prints out all root-to-leaf paths whose values sum to k. For example, if T is the binary tree in the diagram below and k is 13, then the program will print out `10 2 1` on one line and `10 4 -1` on another.



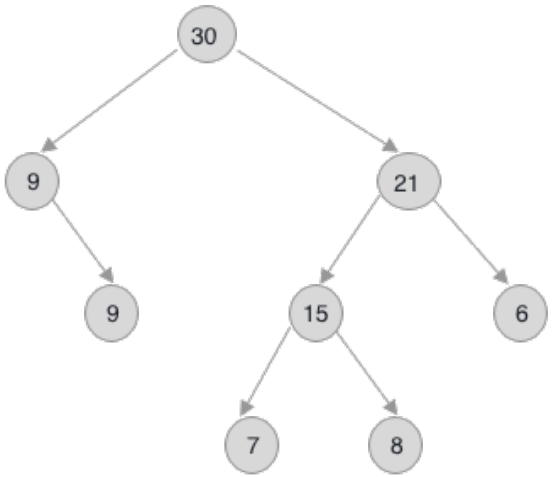
(a) Provide your solution by filling in the code below:

```
public static void printSumPaths(TreeNode T, int k) {
    if (T != null) {
        sumPaths(                                    );
    }
}

public static void sumPaths(TreeNode T, int k, String path) {









}
```

(b) What is the worst case runtime of `printSumPaths` in terms of N, the number of nodes in the tree? What is the worst case runtime in terms of h, the height of the tree?

3 Sum Tree

Given a binary tree, check if it is a sum tree or not. In a sum tree, value at each non-leaf node is equal to the sum of all elements presents in its left and right subtree. For example, the following binary tree is a sum tree -




```
public boolean isSumTree(TreeNode t) {

}
```

4 When am I Useful Senpai?

Based on the description, choose the data structure which would best suit our purposes. Choose from: **A - arrays, B - linkedlists, C - stacks, D - queues** (excluding dequeue's cause they're too OP).

1. Keeping track of which customer in a line came first.

2. We will expect many inserts and deletes on some dataset, but not too many searches and lookups.

3. We gather a lot of data of a fixed length that will remain relatively unchanged overtime, but we access its contents very frequently.

4. Maintaining a history of the last actions on Word in case I need to undo something.

5 Pseudo Stack

Implement a stack's `pop` and `push` methods using two Queues. Assume that we have a `MyIntQueue` class with API :

```
boolean isEmpty() //returns true if the queue is empty
void enqueue(int item) //adds item to the back of the queue
int dequeue() //removes the item at the front of the queue
int peek() //returns but doesn't remove the item at the front of the queue
int size() //returns the size of the queue
```

```
public class MyIntStack {
    MyIntQueue q1 = new MyIntQueue();
    MyIntQueue q2 = new MyIntQueue();

    public boolean isEmpty() {
        //Implementation not shown
    }
    public int size() {
        //Implementation not shown
    }
    public void push(int item) {









    }

    public int pop() {









    }

}
```

6 A Balancing Act

Given a string *str*, containing just the characters `(`, `)`, {, }, `[`, and `]`, implement a method `hasValidParens` which determines if the string is valid.

The brackets must close in the correct order so "`()`", "`()`{}", and "`[()]`" are all valid, but "`(`", "`({)`}", and "`[(`" are not.

You may use the `getRightParen` method provided below.

```
private static boolean hasValidParens(String str) {
    Stack s = new Stack();
    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        if (_____________________________________________________________) {
            _____________________;
        } else {
           if (_____________________________) {
               _____________________;
           }
           if (c != ____________________) {
               _____________________;
           }
        }
    }
    _____________________;
}


/**
    The method getRightParen takes in the left parenthesis
    and returns the corresponding right parenthesis.
**/
private static char getRightParen(char leftParen) {
    if (leftParen == '(') {
        return ')';
    } else if (leftParen == '{') {
        return '}';
    } else if (leftParen == '[') {
        return ']';
    } else {
        //not one of the valid parenthesis characters
        throw new IllegalArgumentException();
    }
}
```