Problem Set 1: Recurrences and Proofs

Due: Thursday, July 5th at 5 p.m.[1]

Welcome to CS161. This first problem set will encourage you to think a little more critically than you might for a programming class, and will also give you the opportunity to present clear and formal derivations to theoretical problems. The problem set is short, but in spite of its being short, you be revisiting many of the key concepts introduced in the previous discrete math classes you've presumably taken. Have fun with it!

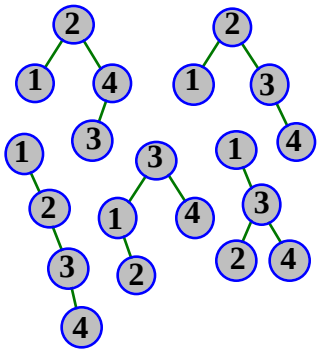
Problem 1: Crash Course In Treaps

In CS106B or CS106X, you were introduced to a recursive data structure called the **binary search tree**, most likely using a data structure like this:

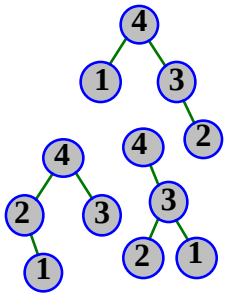
```
typedef struct node {
   int key;
   struct node *left;
   struct node *right;
} node, *tree;
```

The placement of nodes within a binary search tree were subject to the following constraints:

1. The key stored at the root of any subtree must be strictly greater than all keys stored in the left subtree.
2. The key stored at the root of any subtree must be less than or equal to all those keys stored in the right subtree.



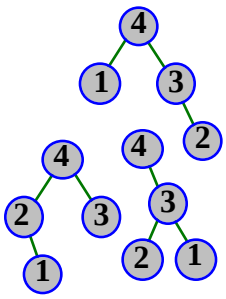
For example, consider all of the binary trees which store the keys 1, 2, 3, and 4. Among the fourteen different legal binary search trees which store these four keys are those five trees drawn on the left. Some examples of binary trees which are not binary **search** trees are drawn on the right.

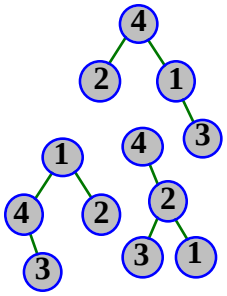


[1] Understand that problem sets will generally fall due on Wednesdays, but because of the Wednesday holiday next week, I'm making the problem set due one day later.

A related data structure—specifically, the **heap** data structure—is also a binary tree data structure, but instead maintains a **heap** property—the property that the key of every node in the binary tree is greater than or equal to those of its children.

Consider once more those binary trees which store the keys 1, 2, 3, and 4. Drawn on the left are three of the legal heaps storing these keys, and drawn on the right are three binary trees which are **not** heaps.





So we have binary search trees, and we have heaps. A **treap** is a binary tree data structure where each node, in addition to storing its key, stores a second value called a **priority**. A treap also maintains the binary search tree property with respect to its keys, while **simultaneously** maintaining the heap property with respect to its priorities. A key and its priority must never be separated; that is, a key and its priority are always stored in the same node. However, the location of the node within the treap depends on its relationship to other nodes.

a) Draw the treap which stores the following sets of keys and priorities:

$$(\text{"Jerry"},0.73),(\text{"Bob"},0.53),(\text{"Julie"},0.99),(\text{"Nick"},0.03),(\text{"Maggie"},0.28)$$

In this case, the keys are strings and the priorities are real numbers.

b) Consider a collection of n ordered pairs. Assuming that all keys and priorities are distinct, prove that there always exists **exactly one** treap which accommodates the collection of n ordered pairs while meeting the treap constraints.

Problem 2: Fibonacci Bases

One of the most important properties of the Fibonacci numbers is the special way in which they can be used to represent integers. Using the notation $j >> k$ to mean that $j \geq k+2$, prove that every positive integer has a unique representation of the form

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}, \quad k_1 >> k_2 >> \cdots >> k_r >> 0.$$[2]

Problem 3: Circular Towers

Let Q_n be the minimum number of moves needed to transfer a tower of n disks from A to B, if all moves must be clockwise—that is, from A to B, or from B to the other peg,

[2] For instance, $1000000 = 832040 + 121393 + 46368 + 144 + 55 = F_{30} + F_{26} + F_{24} + F_{12} + F_{10}$ and $1000001 = 832040 + 121393 + 46368 + 144 + 55 + 1 = F_{30} + F_{26} + F_{24} + F_{12} + F_{10} + F_2$

or from the other peg to A. Also, let R_n be the minimum number of moves needed to go from B back to A under this restriction. Justify why:

$$Q_n = \begin{cases} 0; & \text{if } n = 0; \\ 2R_{n-1} + 1; & \text{if } n > 0; \end{cases} \qquad R_n = \begin{cases} 0; & \text{if } n = 0; \\ Q_n + Q_{n-1} + 1; & \text{if } n > 0; \end{cases}$$

You needn't solve these recurrences.