Problem Set #4

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn these problems in. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Algorithms and Decision Trees

You are given 9 identical looking balls and told that one of them is slightly heavier than the others. Your task is to identify the defective ball. All you have is a balanced scale that can tell you which of the two sides is heavier or if the two sides weigh the same.

Give a decision tree lower bound showing that it is not possible to determine the defective ball in fewer than 2 weighings.

Problem 2: Decision Tree Lower Bounds

Alan's toolbox is a mess. He has lots and lots of bolts of all different sizes. He keeps them in bins, and they are a little bit sorted within the bins. For example, we know that all of the bolts in the first bin are smaller than any of the rest of the bolts. We know that all of the bolts in the second bin are smaller than all of the rest of the bolts, except those in the first bin. And so on.

Let's think about a way to help him get organized. Let's say he has n bolts. They are divided into bins as described above such that each bin contains exactly k bolts (i.e., there are n/k bins), and the bolts in a given bin are all smaller than the bolts in the succeeding bin and larger than the bolts in the preceding bin.

(a) Show an $\Omega(n \log k)$ lower bound on the number of comparisons needed to put all n bolts in total order.

(b) Design an optimal algorithm for me to use to sort Alan's bolts.

Problem 3: Glass Jars

You have been asked to do some testing of a model of new glass jars to determine the maximum height at which they can be dropped without breaking. The setup for your experiment is as follows. You have a ladder with n rungs. You need to find the highest rung from which you can drop one of the jars without it breaking. We'll call this the *highest safe rung*.

Intuitively, you might try a binary search. First, drop the jar from the middle rung and see if it breaks. If it does, try rung $n/4$; if not, try rung $3n/4$. But this process can potentially break a lot of jars. If your primary goal were to break as few jars as possible, you might start at rung 1. If the jar doesn't break, you move on to rung 2. You're guaranteed to break only one jar, but you may have to do a lot of dropping if n is very large.

To summarize, you have to trade off the number of broken jars for the number of drops. To understand this tradeoff better, consider how to run the experiment given a budget of $k \geq 1$ jars. Your goal is to determine the correct answer (i.e., the *highest safe rung*) using at most k jars.

(a) Suppose your budget is $k = 2$ jars. Describe an approach for finding the *highest safe rung* that requires at most $f(n)$ drops for some function $f(n)$ that grows slower than linearly. (In other words, it must be true that $\lim_{n\to\infty} f(n)/n = 0$.) This means you cannot just start at the bottom rung and work up.

(b) Now suppose your budget is $k > 2$ jars, for some given k. Describe an approach for finding the *highest safe rung* using at most k jars. If $f_k(n)$ is the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, \ldots$ should have the property that each grows asymptotically slower than the previous one, i.e., that it is true that $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ for each k.

Problem 4: Heaps

Solve the following problems related to Heaps.

(a) Devise an algorithm for finding the k smallest elements of an unsorted set of n integers in $O(n + k \log n)$.

(b) Give an $O(n \log k)$ time algorithm that merges k sorted lists with a total of n elements into one sorted list.

Problem 5: Priority Queues

One of your classmates claims to have developed a new data structure for priority queues (other than using a heap) that supports the standard priority queue operations INSERT, MAXIMUM, and EXTRACT-MAX all in constant ($O(1)$) time in the worst case. Prove that he is mistaken. (Hint: this is not a lengthy proof; think about the implications of such a claim on the efficiency of sorting.)