CS 5800 Problem Piazza 1
Fall 2013
Karl Lieberherr

Knowledge needed:
Algorithmic: Linear search, Binary Search, Binary Search Trees, Recurrence Relations, Dynamic Programming, Memoization, Pascal's Triangle.
Logic Game: Semantic game with Exists, ForAll , and, !.
Technical: JSON notation, programming.

Find a debate partner to solve the problem collaboratively. You should keep the same partner for several weeks. If you prefer, you can be a group of one but you deprive yourself of learning from others through the semantic games.

1. (See Kleinberg and Tardos, Addison Wesley, Chapter 2, ex. 8.) You are doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a jar and not have it break. We call this the *highest safe rung.*

   It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

   If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar – at the moment it breaks, you have the correct answer – but you may have to drop it n times (rather than $\lg n$ as in the binary search solution).

   So here is the trade-off: it seems you can perform fewer drops if you are willing to break more jars. To study this trade-off, let k be the number of jars you are given, and let n be the actual highest safe rung. Give the minimum number of drops needed to find the highest safe rung, as a function of k and n. When the minimum number of drops needed is x, we write HSRnk-min(n,k)=x. In other words, HSRnk-min(n,k) is the smallest number of questions needed in the worst-case for a ladder with rungs 0..n-1 and a jar budget of k. Your goal is to find an algorithm for HSRnk-min(n,k). Hint: consider the case $k = 2$, and then think about what happens when k increases.

   With HSR(n,k,q) we denote the claim: there exists an experimental plan for a ladder with n rungs, k jars and a maximum of q questions to determine the highest safe rung.

   Interesting small claims to start with:

   HSR(9,2,4), HSR(9,2,5),HSR(9,2,6),HSR(9,2,7)

2. Defining a common language for the scientific discourse about HSR. To represent an algorithm, i.e., an experimental plan, for finding the highest safe rung for fixed n,k, and q we use a restricted programming language that is powerful enough to express what we need. We use the programming language of binary decision trees which satisfy the rules of a binary search tree. The nodes represent questions such as 7 (representing the question: does the jar break at rung 7?). The edges represent yes/no answers. We use the following simple syntax for decision trees based on JSON. The reason we use JSON notation is that you can get parsers from the web and it is a widely used notation.

A decision tree is either a leaf or a compound decision tree represented by an array with exactly 3 elements.

```
// h = highest safe rung or leaf
{ "decision_tree" :
 [1,{"h":0},[2,{"h":1},[3,{"h":2},{"h":3}]]]
}
```

The grammar and object structure would be in an EBNF-like notation:

```
DTH = "{"  "\"decision_tree\""  ":" <dt> DT.
DT = Compound | Leaf.
Compound = "[" <q> int "," <yes> DT "," <no> DT "]".
Leaf = "{" "\"h\" " ":" <leaf> int "}".
```

This approach is useful for many algorithmic problems: define a simple computational model in which to define the algorithm. The decision trees must satisfy certain rules to be correct.

A decision tree in DT for HSR(n,k,q) must satisfy the following properties:
1) the BST (Binary Search Tree Property): For any left subtree: the root is one larger than the largest node in the subtree and for any right subtree the root is equal to the smallest (i.e., leftmost) node in the subtree.
2) there are at most k yes from the root to any leaf.
3) the longest root-leaf path has q edges.
4) each rung 1..n-1 appears exactly once as internal node of the tree.
5) each rung 0..n-1 appears exactly once as a leaf.