CPE 349: Assignment– Destructive Testing

Suppose you are doing stress testing on a type of communication device that must survive long drops. To test the device you have a ladder with n-rungs and you want to find the highest rung from which the device will continue to operate after a drop from that rung. We will refer to this as the *highest safe rung*.

A natural approach is to do a binary search:

- drop a device from the middle rung, see if it breaks
- then do a recursive search using either n/4 or 3n/4 depending on the outcome.

The drawback is that this may break a lot of devices to find the answer.

#1: For a given ladder, what is the worst case number of devices you would break using this binary search approach?

If your goal were to use as only a single device, then you might just start at the bottom rung and work your way up the ladder until the device broke. But this has the drawback that it takes more (drops) time.

So it seems there is a basic tradeoff between expense of broken devices and the time (# drops) it takes to find the *highest safe rung*. Suppose you are given a budget of **2 devices** (the prototypes are very expensive.)
Design a strategy for finding the highest safe rung that requires you to drop the device at most (worst case) f(n) times for some function that grows **asymptotically more slowly than linearly**. (Note that "dropping the device" is the crucial step and is what you need to count. In other words f(n) should be better (grow more slowly) than functions in O(n). Then implement a **simulation** of how your algorithm would work in Java using the following specification. (In a simulation of this problem you will be given the highest safe rung. The algorithm will then need to find the highest safe rung by simulating the dropping of the devices.)

Input: Two positive integers

- "height of ladder" // height of the ladder can be any positive integer up to the largest positive integer expressible in Java
- "highest safe rung" // an integer ≥ 1 and ≤ height of ladder

Behavior: You program must simulate the test plan that your algorithm would perform in the "real" world. That is, it simulates the drops it would perform and **counts** the total number of drops that it performs (it does **not** calculate the number of drops it would perform, it actually counts them in the simulation) in order to determine the highest safe rung. Your program will be implemented in a class **DestTesterClass.java**. In this class implement a **static non-recursive** method **findHighestSafeRung** that returns an array list of integers in the following order.

height of the ladder (input parameter)
actual highest safe run (input parameter)
Highest safe rung determined by this algorithm
Rung where the first test device broke (-1 if this device never breaks)
Rung where the second test device broke (-1 if this device never breaks)
Total number of drops required to find the highest safe rung

findHighestSafeRung takes as input two positive integers

"height of ladder" , "highest safe rung" as described above

Before the due date and time you must submit three files to PolyLearn.

1. The source code for the java class that performs the simulation **DestTesterClass.java. this class must use the java file supplied as a starting point. In addition, the method `findHighestSafeRung` must not print any output and must return the values specified.**
   Make sure that your source code is well written. e.g. variable names should be meaningful but not too long, white space should make it easy to see program flow, and a few comments should explain how the algorithm works. Comments should **not** just be a rewording of the java code!
2. The source code for a driver java class uses **DestTesterClass.java** and prints out results so that you can test and demo your program in lab if desired
3. A pdf file, **DestTestAnalysis.pdf ,** that contains:
   - A derivation of **f(n)** where f(n) is the worst case number of drops needed by your program to find the highest safe rung on a ladder of n rungs. Clearly explain why this is the worst case and how you determined f(n).
   - **Three graphs** that show empirical results of the number of drops required for different ladder sizes and different highest safe rungs. Each graph should be followed by a table showing the points used to create the graph (ladder size, #drops to find highest safe rung).
   - All three graphs will show the number of drops required for the ladder sizes 10,000, 100,000, 1,000,000, 10,000,000 (ladder sizes represent the x coordinate and the x-axis should use a log scale.
     (in excel - http://grok.lsu.edu/article.aspx?articleid=8105)
     a) Graph 1 should show the number of drops if the highest safe rung is **100** for all four ladder sizes.
     b) Graph 2 should show the number of drops if the highest safe rung is the **ladder size - 1** for all four ladder sizes. I.e. ladder size = 100 then highest safe rung is 99.
     c) Graph 3 should show the number of drops if the highest safe rung is the **worst case** for all four ladder sizes. This will vary depending on your algorithm and how it is implemented. Thus you will need to determine this input parameter. Think carefully about the specifics how your algorithm is searching in each of the four ladder sizes.
   - Finally, interpret Graph 3. Does it agree with your theoretical analysis by comparing the number of drops calculated by your analysis with the actual number of drops performed by your simulation. If not, what is your explanation.

In lab you may be asked to demo your program. It must be the same program that you submit to PolyLearn.