The “Not-To-Be-Turned-In Practice-For-The-Final Exam” Problem Set

Some of this material came from the text *Introduction to Algorithms* by Corman, Leiserson, Rivest, and Stein.

**Question 1.** Warm-up Consider the following game. I flip a coin repeatedly until it comes up tails, which ends the game. Sounds boring, right? To make it interesting I tell you that if it comes up heads the first time, I will give you a dollar and for each subsequent heads that comes up, I will give you double the amount I gave you for the previous heads. Determine your expected winnings, and given your answer, let me know how much you’d be willing to pay me to let you play this game with me!

**Question 2.** Imagine that you have a method `fairBit()` that will generate in constant time a random bit 0/1 with an equal probability of 1/2. You’d like to write a method `rng(d)` that will generate, for any \( d > 1 \), a random integer \( i \), where \( 0 \leq i \leq d \) with an equal probability of \( 1/(d+1) \). Note that this can be used to generate, for any \( a < b \), one of the values \( a, a+1, \ldots, b \) with equal probability by just computing \( a + \text{rng}(b-a) \), so this is a generally useful method.

You realize that randomization might be an option so you implement the following Las Vegas algorithm: Let \( k = 1 + \left\lceil \log_2 d \right\rceil \) (so, \( k \) is the number of bits required to represent \( d \)). Now use `fairBit()` \( k \) times to generate the bits of a random \( k \)-bit number \( n \). If \( n \leq d \) then return \( n \), otherwise repeatedly generate a new \( n \) until \( n \leq d \) and return the first such \( n \) constructed.

Give a convincing justification that `rng(d)` returns values in the range \([0, d]\) with equal probability and calculate the expected running time of this algorithm. Note: \( d \) requires \( k \) bits to represent it, so \( d \geq 2^k - 1 \)

**Question 3.** Imagine now that you have a method `biasedbBit()` that generates in constant time 0 with probability \( p \) (\( 0 < p < 1 \)) and 1 with probability \( 1 - p \). However, you don’t know what \( p \) is. You’d like to write a method `fairBit()` that will generate a random bit (0/1) with equal probability. Design a Las Vegas algorithm for implementing `fairBit()` using `biasedbBit()` and determine its expected running time.

*Hint: Consider repeatedly calling `biasedBit()` twice until an iteration produces two different bits....*

**Question 4.** Suppose you have an array \( A[1..n] \) containing \( n \) distinct numbers. Call a pair of indices \( i < j \) an inversion if \( A[i] > A[j] \). Note that \( A \) is sorted in increasing order exactly when there are no inversions and \( A \) is in reverse sorted order when all \( \binom{n}{2} \) pairs of indices form inversions. Assuming that all orderings of the elements of \( A \) are equally likely, compute the expected number of inversions in \( A \). What does this say about the average run-time of insertion sort and bubble sort?

*Hint: Consider, for each pair \( i < j \) the random variable \( X_{i,j} \) having value 1 if \( \{i, j\} \) form an inversion and 0 otherwise. What does the sum of all of these random variables represent?*