CS 473 ✧ Spring 2017
Homework 5

Due Wednesday, February 15, 2017 at 8pm

0. **[Warmup only; do not submit solutions]**

   After sending his loyal friends Rosencrantz and Guildenstern off to Norway, Hamlet decides to amuse himself by repeatedly flipping a fair coin until the sequence of flips satisfies some condition. For each of the following conditions, compute the *exact* expected number of flips until that condition is met.

   (a) Hamlet flips heads.

   (b) Hamlet flips both heads and tails (in different flips, of course).

   (c) Hamlet flips heads twice.

   (d) Hamlet flips heads twice in a row.

   (e) Hamlet flips heads followed immediately by tails.

   (f) Hamlet flips more heads than tails.

   (g) Hamlet flips the same positive number of heads and tails.

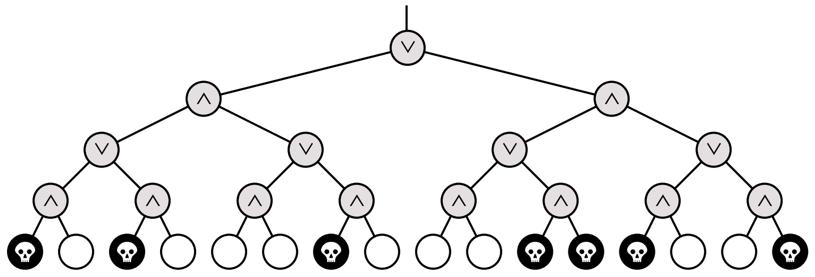
   [Hint: Be careful! If you're relying on intuition instead of a proof, you're probably wrong.]

1. Consider the following non-standard algorithm for shuffling a deck of n cards, initially numbered in order from 1 on the top to n on the bottom. At each step, we remove the top card from the deck and ***insert*** it randomly back into in the deck, choosing one of the n possible positions uniformly at random. The algorithm ends immediately after we pick up card $n-1$ and insert it randomly into the deck.

   (a) Prove that this algorithm uniformly shuffles the deck, meaning each permutation of the deck has equal probability. *[Hint: Prove that at all times, the cards below card $n-1$ are uniformly shuffled.]*

   (b) What is the *exact* expected number of steps executed by the algorithm? *[Hint: Split the algorithm into phases that end when card $n-1$ changes position.]*

2. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.



You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are Or gates, the nodes at odd levels (where it's Death's turn) are And gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for True and False. If the output at the top of the tree is True, then you can win and live forever! If the output at the top of the tree is False, you should challenge Death to a game of Twister instead.

(a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

(b) Unfortunately, Death won't give you enough time to look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $O(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*

*(c) **[Extra credit]** Describe and analyze a randomized algorithm that determines whether you can win in $O(c^n)$ expected time, for some constant $c < 3$. *[Hint: You may not need to change your algorithm from part (b) at all!]*

3. The following randomized variant of "one-armed quicksort" selects the kth smallest element in an unsorted array $A[1..n]$. As usual, PARTITION($A[1..n], p$) partitions the array A into three parts by comparing the pivot element $A[p]$ to every other element, using $n-1$ comparisons, and returns the new index of the pivot element.

```
QUICKSELECT($A[1..n], k$):
    $r \leftarrow$ PARTITION($A[1..n]$, RANDOM($n$))
    if $k < r$
        return QUICKSELECT($A[1..r-1], k$)
    else if $k > r$
        return QUICKSELECT($A[r+1..n], k-r$)
    else
        return $A[k]$
```

(a) State a recurrence for the expected running time of QUICKSELECT, as a function of n and k.

(b) What is the *exact* probability that QUICKSELECT compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and k. *[Hint: Check your answer by trying a few small examples.]*

(c) What is the *exact* probability that in one of the recursive calls to QUICKSELECT, the first argument is the subarray $A[i..j]$? The correct answer is a simple function of i, j, and k. *[Hint: Check your answer by trying a few small examples.]*

(d) Show that for any n and k, the expected running time of QUICKSELECT is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b) or (c).