Homework #2
Probabilistic Analysis, Randomized Algorithms and Lower Bounds
Presentation: 6-9 Nov, 2018

- This is an oral presentation assignment. You will present and discuss your answers with the professor using a white board (you may bring auxiliary written material to aid you).
- You should work in **groups of three**. At some point before the 2nd of November (friday) at 11:59pm, your group should sign up for a 30-minutes time slot on the signup sheet that you will receive by email.
- Each person in the group must be able to present every problem. The professor will select who presents which problem. The other group members may assist the presenter in case of need.
- You are not required to hand anything in at your presentation, but you may if you choose.

---

1. Quicksort variations

We will look at three versions of the Quicksort algorithm, each with a different rule for choosing the pivot.

1. Suppose we choose the pivot to be the *median* of the n numbers in the array. Assuming that it takes $O(n)$ time to compute the median (we will study an algorithm called *quickselect* later in class that achieves this), what is the worst-case running time of this version of Quicksort?

   Hint: try to write the recurrence that indicates the cost of this algorithm

2. Suppose we choose the pivot to be the *average* of the n numbers in the array (i.e. we add up all the numbers and divide by n). Assuming that it takes $O(n)$ time to compute the average, what is the worst-case running time of this version of Quicksort? Describe an array achieving this worst-case bound, giving an explicit formula for the i-th element in the array.

   Hint: how unbalanced can a split be using the average as the pivot?

3. Suppose you use the following algorithm to select the pivot:

   (a) look at the element in the middle of the array and count how many elements are bigger and how many are smaller.

   (b) if the elements are split evenly (i.e. half on each side), then choose that element as the pivot.

   (c) otherwise, do a random permutation of the array in $O(n)$ and go back to step (a).

   What is the average-case running time of this version of Quicksort?

   Hint: start by thinking what is the *expected* number of times you need to produce a random permutation before having a pivot that provides a balanced split. What happens next?

---

2. To the moon (Monte Carlo) and back (Las Vegas)!

1. ***From Monte Carlo to Las Vegas***: Imagine you have a Monte Carlo randomized algorithm that takes an unsorted array A, and outputs another array B in time $T(n)$. The output B is always some permuted version of the input A. We also know that:

$$Pr[B \text{ is in sorted order}] \geq p$$

   Prove that you can convert such an algorithm into a Las Vegas randomized sorting algorithm that has expected running time $O((T(n) + n) * 1/p)$.

   Hint: Given n, the Las Vegas algorithm should *always* output a sorted permutation, but its runtime should be a random variable (and not its probability of success, as before).

2. ***Back to Monte Carlo*:** Show that, if you have a Las Vegas algorithm with **expected** running time $T(n)$, then we can get a Monte Carlo algorithm with worst-case running time at most $2 \times T(n)$ and probability of success at least $1/2$.

   Hint: Use *Markov's Inequality*. For any non-negative random variable X, let $E[X] = a$. Then, for any $c > 0$,

$$Pr[X \geq ca] \leq 1/c$$

   This is a very useful inequality relating expected value and parts of a population using non-negative variables: e.g. it is impossible for more than half the population to be more than twice the average height, or for more than a tenth of the population to earn more than 10 times the average salary.

3. ***Stuck in Monte Carlo?*** In some cases the idea of part 1 can't be applied. For instance, imagine I have a Monte Carlo algorithm that finds the shortest path in a graph between 2 nodes in time $T(n)$ with probability at least p; why can't I construct a Las Vegas algorithm for this problem with expected running time $O((T(n) + n) * 1/p)$?

   Hint: What is different between the shortest path problem and the sorting problem?

---

3. Breaking light bulbs...

Imagine that I choose a number between 1 and N and you want to guess it in as few questions as possible (each time you make an incorrect guess, I'll tell you if it is too high or too low). As we all know, the strategy for this problem that minimizes the worst-case number of guesses is to do *binary search*. But, what if you are only **allowed at most one guess that is too high**? Another way to state this problem: you want to figure out how many centimeters high you can drop a light bulb without breaking it, but you only have two light bulbs...

1. Can you still solve this problem in $o(N)$ guesses? You should describe the algorithm you would use and give its asymptotic complexity (in terms of the number of guesses).

   Hint: If you were not allowed **any** guesses that are too high, the only option you would have would be to guess $1, 2, 3, \ldots$ (in that order).

2. Can you show a lower bound for any deterministic algorithm that matches the upper bound of your algorithm in the previous question?

   Hint: You are asked to prove that no deterministic algorithm can find the answer asymptotically faster than yours.
