CS 161 (Stanford, Winter 2024) Section 3

1 Expectation

1. *True or False:* Expected runtime averages the runtime over the outcomes of random events within the algorithm for a random input.
2. I have an algorithm that takes positive integers (n, i) where $1 \leq i \leq n$. The algorithm rolls a n-sided die repeatedly until the die returns any value $\leq i$. What is the expected runtime in n? Worst-case runtime? Rigorous proof not necessary :)

2 Light Bulbs and Sockets

You are given a collection of n differently sized light bulbs that have to be fit into n sockets in a dark room. You are guaranteed that there is exactly one appropriately-sized socket for each light bulb and vice versa; however, there is no way to compare two bulbs together or two sockets together as you are in the dark and can barely see! You can try and fit a light bulb into a chosen socket, from which you can determine whether the light bulb's base is too large, too small, or is an exact fit for the socket.

Suggest a (possibly randomized) algorithm to match each light bulb to its matching socket. Your algorithm should run strictly faster than quadratic time in expectation. Give an upper bound on the worst-case runtime, then prove your algorithm's correctness and expected runtime.

3 All On the Same Line

Suppose you're given n distinct ordered pairs of integers $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where for all i, j, $x_i \neq x_j$ and $y_i \neq y_j$. Recall that two points uniquely define a line, $y = mx + b$, with slope m and intercept b. (Note that choosing m and b also uniquely defines a line.) We say that a set of points S is $collinear$ if they all fall on the same line; that is, for all $(x_i, y_i) \in S$, $y_i = mx_i + b$ for fixed m and b. In this question, we want to find the maximum cardinality subset of the given points which are collinear – in less jargon, we're looking for the maximum integer N such that we can find N of the given points the same line. Assume that given two points, you can compute the corresponding m and b for the line passing through them in constant time, and you can compare two slopes or two intercepts in constant time.

This is a challenging problem – so we're only going to pseudocode at a high level!

1. Design an algorithm to find a maximum cardinality set of collinear points in $O(n^2 \log n)$ time. If there are several maximal sets, your algorithm can output any such set. *Some*

hints:

- $O(n^2 \log n) = O(n^2 \log n^2)$, which looks like sorting n^2 items.
- Start small; how would we verify that 3 points are on the same line?

2. It is not known whether we can solve the collinear points problem in better than $O(n^2)$ time. But suppose we know that our maximum cardinality set of collinear points consists of exactly n/k points for some constant k. Design a randomzed algorithm that reports the points in some maximum cardinality set in expected time $O(n)$. Prove the correctness and runtime of your algorithms.

   Some hints:

   - Your expected running time may also be expressed as $O(k^2 n)$.
   - You algorithm might not terminate!

For your own reflection: Imagine that you, an algorithm designer, had to pick one of the algorithms in part (a) or (b) to implement in the autopilot of an airplane, as part of the route-planning of a self driving car, or in any other scenario in which human lives are at stake. Given what you know about the performance and worst-case scenario of each of the algorithms, which algorithm would you choose and why?

4 Batch Statistics

Design an algorithm which takes as input array A consisting of n possibly very large integers as well as an array R that contains k ranks $r_0, ..., r_k$, which are integers in the range $\{1, ..., n\}$. (You may assume that $k < n$.) The algorithm should output an array B which contains the r_j-th smallest of the n integers, for every j in $1, ..., k$. So if an $r_j = 3$ in input array R, then we want to return the 3rd smallest element in the input array A as part of the output.

Input: A which is an unsorted array of n unbounded distinct integers; R which is an unsorted array of k distinct ranks.

Example:

- Input: $A = [11, 19, 13, 14, 16, 18, 17, 12, 15]$; R = [3, 7]
- Output: $[17, 13]$
- Explanation: 17 is the 7th smallest element of A and 13 is the 3rd smallest of A. [13, 17] is also an acceptable output.

Hint: we are looking for an $O(nlogk)$ runtime algorithm.