1 Introduction: Randomization as Design Technique

Who doesn't love provably efficient deterministic algorithms! But they can frequently be hard to implement, requiring sophisticated data structures. They can also be disappointing in practice, the "Big-Oh" analysis often obscuring very large constants of proportionality. In addition, an application developer may be willing to tolerate occasionally slow performance as long as performance is very good "on average".

A number of fundamental problems admit algorithms having performance which is provably good "on average". Such algorithms employ *randomization* techniques and are often much easier to implement than their deterministic counterparts, as well as having lower constants of proportionality. This week we will learn some of the basic tools and techniques needed to design and analyze such algorithms. We'll also look at the well-known paper that presented the first linear-time median finding algorithm and compare that algorithm to a much simpler randomized algorithm.

2 Fundamentals of Randomized Algorithms

This week's work begins by having you read Sections 1–6, 9, 10, and 12 from Chapter 13 of *Algorithm Design* by Kleinberg and Tardos [1]. Section 13.12 provides some additional background material on discrete probability; read it in parallel with the other sections.

3 Exercising Your New Skills

After completing the reading, work through the following problems at the end of Chapter 13:

- 3.
- 4.
- 6. Hint: mimic the approach from Section 3.4 to get an algorithm that always produces a good ordering in expected polynomial time
- 7.
- 8. Question: why is $mk(k-1)/n(n-1)$ a meaningful goal? Also, I'd aim for (b)
- 13. Hint: Chernoff Bounds
- 14. Notes: If you show that the probability of failure is less than 1, then you get for free that a solution always exists. Also, if you want to guarantee termination, you can, after enough failure, try an exhaustive search. What will that do to the expected run time?
- 15. This one is perhaps more challenging. Here's a hint. Imagine dividing S into 20 subsets $Q_1, \ldots, Q_{20}$, where Q_i consists of every $x \in S$ such that x has at least $(i-1)n/20$ elements of S less than it, and at least $(20-i)n/20$ elements of S greater than it (in other words, Q_1 consists of the smallest $n/20$ elements, Q_2 consists of the next $n/20$ smallest, and so on). Then choosing a sample S' from S can be viewed as throwing balls into 20 bins labeled $Q_1, \ldots, Q_{20}$. Show that the probability that the balls are not nicely distributed is small—where nicely distributed intuitively means that any ball in bin Q10 or Q11 would work as an approximate median. Oh, and: Chernoff bounds....

[1] If you took CS 256 at Williams, this was the course text; one or two copies should be on reserve at Schow.

4 A Deterministic Linear-Time Algorithm for Computing the Median

In 1971, five (outstanding) computer scientists developed the first linear-time algorithm for computing the median. Manuel Blum, Robert Floyd, Vaughn Pratt, Ron Rivest, and Robert Tarjan first presented their result at STOC in 1972. The STOC proceedings contains the "conference version" of their paper. Read Section I and make a good faith effort to read Section II. A year later, the full journal version appeared as *Time Bounds for Selection* in the *Journal of Computer and System Sciences*. Read Sections 1 and 2 of this paper and browse through Section 3 to get a feel for tweaking of the PICK algorithm needed to produce PICK1 and for the the complexity of the analysis.

5 Presentation

Student B should come prepared to discuss the most interesting aspects of the problems assigned. Student A should come prepared to address the following questions:

- What were the key tools from discrete probability that were presented?
- Some randomized algorithms are guaranteed to halt, but won't necessarily return the correct solution; some are guaranteed to only return correct solutions but won't necessarily be fast (or even halt!). Give a good example of each. Describe how the first type of algorithm can be converted to the second. Do you think that the second can always be converted to the first?
- What are the essential aspects of the "simple" version of Blum et al's linear time selection algorithm?