CAS CS 535: Complexity Theory

Lecturer: Mark Bun — Fall 2023

**Lecture Notes 14:**

**Randomized Algorithms, Probabilistic TMs and Complexity Classes**

**Reading.**

- Arora-Barak § A.2, 7.1-7.2

**Last time:** Karp-Lipton, Circuit Lower Bounds, Restricted Depth
Today we'll start a unit on randomized computation.

**Motivation.** In the real world, various processes are (or, for all intents and purposes, can be modeled as) random: The diffusion of gas or heat, atmospheric noise, prices in financial markets, quantum mechanics...

If randomness is out there in the real world as a physical resource, it makes sense to leverage it when designing algorithms. This notion was studied implicitly for a long time in statistics, where the goal is to design algorithms to learn information about a population from a random sample. In computer science, there are many interesting examples of using randomness to improve (e.g., speed up) a computation even though the problem being solved has a deterministic answer.

# 1 Median Finding

**Problem 1.** Given an unordered list of $n$ numbers and a parameter $1 \leq k \leq n$, find the $k$'th smallest number.

For example, if the list is $\{8, 7, 12, 3, 2, 4, 9\}$ and $k = 3$, the solution to the problem is $4$. The problem of finding a median is the special case of this problem when $k = n/2$.

A natural deterministic algorithm for this problem is to sort the list and count to $k$. This takes roughly $O(n \log n)$ time (if implementing it using comparisons).

Here is an $O(n)$-time randomized algorithm. (When, e.g., analyzed in the word RAM model where comparison and other arithmetic operations on integers takes constant time.) We'll make the simplifying assumption that the list elements are all distinct.

Algorithm Find$k(\{a_1, \ldots, a_n\}, k)$:

1. Pick a random pivot $i \in [n]$, and let $x = a_i$.
2. Scan list to count $m = \#\{j \in [n] \mid a_j \leq x\}$.
3. If $m = k$, output $x$.
4. Else, if $m > k$, output Find$k(\{a_j \mid a_j \leq x\}, k)$.
5. Else, if $m < k$, output Find$k(\{a_j \mid a_j > x\}, k - m)$.

**Correctness:** If $m = k$ (and all elements are distinct), then $x$ is indeed the $k$th smallest element. Otherwise, the recursive call produces the $k$th smallest element by calling the algorithm on a strictly smaller sublist (so the algorithm must eventually terminate with a correct answer).

**Runtime:** Note that in the worst case, we could pick $k = n$ but get unlucky with obtaining $m = 1$ every time. This would force us to recurse $n$ times, with half of those runs dealing with a list of size at least $n/2$, giving us a runtime of $\Omega(n^2)$.

However, in expection over the random choice of the pivots, we do much better. Let $T(n)$ denote the expected runtime of the algorithm on a worst-case input with list size $n$. Then intuitively,

$$T(n) \leq O(n) + T(3n/4)$$

since the expected size of the new list in the recursive call is at most $3n/4$ (realized when $k = n/2$). This recurrence resolves to only $T(n) = O(n)$.

**Claim 1.** *If the non-recursive steps (i.e., steps 1 and 2) of algorithm Findk take $\leq cn$ steps on every input list of size $n$, then $T(n) \leq 10cn$.*

*Proof.* We prove this by induction. We have by the law of total expectation that

$$\begin{aligned}
T(n) &\leq cn + \sum_{m=1}^{n} \Pr[x = m\text{'th smallest number}] \cdot \begin{cases} T(m) & \text{if } m > k \\ T(n-m) & \text{if } m < k. \end{cases} \\
&\leq cn + \frac{1}{n}\left(\sum_{m>k} T(m) + \sum_{m<k} T(n-m)\right) \\
&\leq cn + \frac{1}{n}\left(\sum_{m>k} 10cm + \sum_{m<k} 10c(n-m)\right) && \text{by the inductive hypothesis} \\
&< cn + \frac{10c}{n}\left(\sum_{m>k} m + kn - \sum_{m<k} m\right) \\
&= cn + \frac{10c}{n}\left(\frac{n(n+1)}{2} - \frac{k(k+1)}{2} + k - \frac{k(k-1)}{2}\right) \\
&= cn + \frac{10c}{n}\left(\frac{n(n+1)}{2} + kn - k^2\right), && \text{a quadratic maximized at } k = n/2 \\
&\leq cn + \frac{10c}{n}\left(\frac{n(n+1)}{2} + \frac{n^2}{4}\right) \\
&\leq cn + \frac{10c}{n}\left(\frac{9n^2}{10}\right) = 10cn.
\end{aligned}$$

□

**Remarks:**

- This algorithm always succeeds ("zero error")
- Our runtime bound holds in expectation, and is noticeably lower than the worst-case runtime bound. This is very similar to the classic analysis of Quicksort, which uses $O(n)$ expected comparisons, but $O(n^2)$ comparisons in the worst case.

## 2 Polynomial Identity Testing

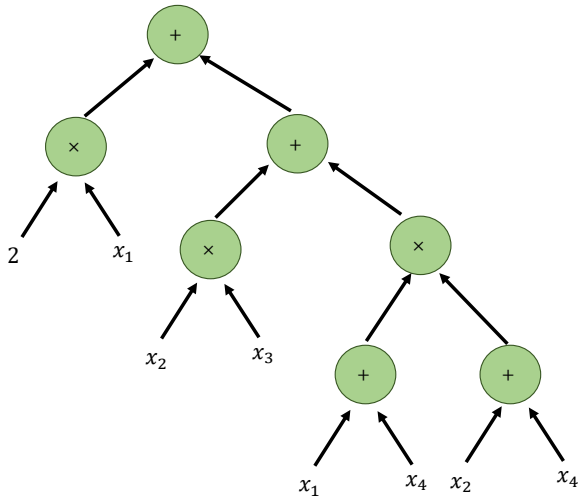
**Problem 2** (Polynomial Identity Testing)**.** Given two polynomials $p, q : \mathbb{Z}^n \to \mathbb{Z}$, are the polynomials identically equal?

For example, the polynomials $p(x, y) = x^2 - y^2$ and $q(x, y) = (x + y)(x - y)$ are identically equal.

Before studying algorithms for this problem, we need to say what we mean by "given a polynomial"; namely, how do we represent a polynomial? One way is through the list of all coefficients, but this doesn't make for a particularly interesting problem, as we could just check equality of the coefficients one-by-one. (Actually, equality of coefficients is the standard *definition* of identity. It coincides with pointwise equality for polynomials over infinite integral domains (like $\mathbb{Z}$), but not necessarily for finite fields.)

The question is most interesting when we have a succinct representation of each polynomial. One such representation is via an arithmetic formula, which is like a Boolean formula, but with gates corresponding to the arithmetic operations $+, \times$.

**Example 2.** The following arithmetic formula represents the polynomial $p(x) = 2x_1 + x_2x3 + (x_1 + x_4)(x_2 + x_4)$.




Note that polynomials $p$ and $q$ are identically zero iff $p - q \equiv 0$. Thus, identity testing is equivalent to zero testing, motivating the following definition of a language:

$$\mathsf{ZEROF}_{\mathbb{Z}} = \{\text{Arithmetic formulas } F \mid F(x) = 0 \quad \forall x \in \mathbb{Z}^n\}.$$

This is a very important problem in arithmetic circuit complexity, as well as interesting for studying the role of randomness in computation. It is not known to have a subexponential time deterministic algorithm, but does have an efficient randomized algorithm.

**Intuition:** A Boolean formula represents a low-degree polynomial. There are then two possibilities:

- If the polynomial is identically zero then (obviously) it evaluates to 0 everywhere.
- If the polynomial is not identically zero, then it evaluates to 0 on a small number of points.

Together, these mean that we can test whether a polynomial is zero by testing whether it is zero on a randomly chosen input.

So why is the second statement true? For univariate polynomials, for example, we know from the Fundamental Theorem of Algebra that every degree-$d$ polynomial over $\mathbb{Z}$ has at most $d$ real roots. The following important fact generalizes this to multivariate polynomials.

**Lemma 3** (Schwartz-Zippel)**.** *If $p(x_1, \ldots, x_n)$ is a polynomial of degree $d$, and $S \subseteq \mathbb{Z}$ is a finite set of integers, then*

$$\Pr[p(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|}$$

*where $a_1, \ldots, a_n \leftarrow S$ are chosen uniformly at random with replacement.*

*Proof idea.* See the Appendix of Arora-Barak for the full proof. It goes by induction on the number of variables $n$. The base case $n = 1$ follows directly from the Fundamental Theorem of Algebra. To extend this inductively to larger $n$, we write

$$p(x_1, \ldots, x_n) = \sum_{i=0}^{d} x_n^i p_i(x_1, \ldots, x_{n-1})$$

where each $p_i$ is a degree $\leq (d - i)$ polynomial. If $p$ is nonzero, then at least one of these $p_i$ is nonzero. We apply the inductive hypothesis to the largest such $i$, and note that whenever $p_i(a_1, \ldots, a_{n-1})$ is nonzero, the residual univariate polynomial in $x_n$ is nonzero. We can then apply the base case calculation to this univariate polynomial. □

## 2.1 Back to Formula Zero Testing

The Schwartz-Zippel Lemma suggests the following randomized algorithm for $\mathsf{ZEROF}_{\mathbb{Z}}$.

On input formula $F(x_1, \ldots, x_n)$:

1. Sample $a_1, \ldots, a_n$ uniformly at random from $\{1, \ldots, 10|F|\}$ where $|F|$ is the number of gates in $F$.
2. Evaluate $F(a_1, \ldots, a_n)$. If 0, accept, else reject.

**Correctness:** There are two cases to consider: If $F \equiv 0$ and if $F \not\equiv 0$.

If $F \equiv 0$, then the algorithm accepts with probability 1.

**Claim 4.** *If $F \not\equiv 0$, then the algorithm rejects with probability $\geq 9/10$.*

*Proof.* First, observe that if $F$ is an arithmetic formula, then $\deg(F) \leq |F|$. This holds by induction on the size of $F$.

Now if $F \not\equiv 0$, then by Schwartz-Zippel, we have

$$\Pr[F(a_1, \ldots, a_n) \neq 0] \geq 1 - \frac{\deg(F)}{10|F|} \geq 1 - \frac{|F|}{10|F|} = \frac{9}{10}.$$

So the algorithm indeed (correctly) rejects with probability $9/10$. □

**Runtime:** The algorithm requires sampling $n$ points, each with bit complexity $O(\log |F|)$. Evaluating the formula gate-by-gate involves polynomially many arithmetic operations on poly-bit integers, so it takes poly time.

**Remarks:**

- Always runs in polynomial time.
- Always correctly accepts inputs in $\mathsf{ZERO}_{\mathbb{Z}}$ ("one-sided error").
- We can make the constant $9/10$ in the rejection probability to be an arbitrarily large constant by increasing $10|F|$. We'll see how to generically amplify the success probability of randomized algorithms next week.
- The algorithm as stated only works for formulas because of the degree bound $\deg(F) \leq |F|$. The issue with generalizing it to arithmetic circuits is that the degree of the polynomial computed by a circuit can be exponential in its size. This is fine for how we apply the Schwartz-Zippel Lemma, since even if we take $|S|$ to be exponentially large, the bit complexity of each element in $S$ is still polynomial. The real issue is that the values arising in the gate-by-gate evaluation of the circuit may become doubly exponentially large. Arora-Barak describes a way to overcome this by evaluating the circuit modulo a random prime.

# 3 Formalizing Randomized Computation

**Definition 5.** A probabilistic TM has two transition functions $\delta_0, \delta_1$. It computes by applying either $\delta_0$ or $\delta_1$ independently in each time step, each with probability $1/2$.

This is not the only way to define a probabilistic TM. Some other ways to define it are:

- As a determinsistic TM that has an additional functionality of being able to write a random bit in one step of computation.
- As a deterministic TM with an additional read-only/read-once tape initialized with random bits.

If $M$ is a probabilistic TM and $x$ is an input, let $M(x)$ denote the random variable denoting whether $M$ accepts on input $x$.

Let $T_{M,x}$ be the random variable denoting the number of steps $M$ takes on $x$ before halting.

**Definition 6.**

- $M$ runs in (worst-case) time $T(n)$ if $T_{M,x} \leq T(|x|)$ with probability 1 for all $x \in \{0,1\}^*$.
- $M$ runs in expected time $T(n)$ if $\mathbb{E}[T_{M,x}] \leq T(|x|)$ for all $x \in \{0,1\}^*$.

**Example 7.** Find$k$ runs in expected time $O(n)$.

**Some Probabilistic Time Classes**

**Definition 8.** $L \in \mathbf{RP}$ if there exists a probabilistic TM running in time $\mathrm{poly}(n)$ such that

$$x \in L \implies \Pr[M(x) = 1] \geq 2/3$$
$$x \notin L \implies \Pr[M(x) = 1] = 0.$$

**Definition 9.** $L \in \mathbf{coRP}$ if there exists a probabilistic TM running in time $\mathrm{poly}(n)$ such that

$$x \in L \implies \Pr[M(x) = 1] = 1$$
$$x \notin L \implies \Pr[M(x) = 1] \leq 1/3.$$

**Example 10.** $\mathsf{ZEROF}_{\mathbb{Z}} \in \mathbf{coRP}$.

**Two-sided error:**

**Definition 11.** $L \in \mathbf{BPP}$ if there exists a probabilistic TM running in time $\mathrm{poly}(n)$ such that

$$x \in L \implies \Pr[M(x) = 1] \geq 2/3$$
$$x \notin L \implies \Pr[M(x) = 1] \leq 1/3.$$

**Zero error:**

**Definition 12.** $L \in \mathbf{ZPP}$ if there exists a probabilistic TM running in expected time $\mathrm{poly}(n)$ such that

$$x \in L \implies \Pr[M(x) = 1] = 1$$
$$x \notin L \implies \Pr[M(x) = 1] = 0.$$