Randomized Algorithms

What do you feel about making decisions by flipping a coin?

Surprisingly, randomness helps!

1. One can find the minimum spanning tree of a graph \( G = (V,E) \) in linear time \( \Theta(1V1 + 1E1) \) using a randomized algorithm.

2. Given polynomials \( p, q, r \) of degree \( n-1 \), one can check whether \( r = p \cdot q \) in linear time \( \Theta(n) \) using a randomized algorithm.

(For both, don't know of linear time deterministic algorithms).
Theme

- Randomness works when there are many possible choices, and "most" are good.
- Surprisingly, even when most decisions are good, it's not necessarily easy to find one that's good deterministically.

Example

The Middle Elements Problem

Given an array with n numbers, (unsorted) find an element in the array whose order is in \([\frac{n}{10}, \frac{9n}{10}]\).

middle elements if array is sorted.
Recall -

If you can solve the middle elements problem in $O(n)$ time, you can find the median in linear time.

A randomized algorithm to find a middle element - pick an element from the array at random!

```
1 2 3 4 ... n-1 n
```

$n$ choices; $\frac{8n}{10}$ of them yield middle elements.

$\Rightarrow$ With probability $\frac{8}{10}$, find middle element.

(Compare that to the complicated recursive deterministic algorithm).
Example - A Randomized Algorithm
With Many Random Choices

Randomized-Median-Finding (A)

1. Pick i \in \{4, \ldots, n\} at random.
2. Find the order of A[i].

3. If A[i] is the median, return A[i].
   - If A[i] smaller than median, remove from A all elements A[j] \leq A[i].
   - If A[i] larger than median, remove from A all elements A[j] \geq A[i].

[Note: algorithm always outputs median; running time depends on random choices]
Idea

- In each application, $n$ choices for $A[i]$.
- If $A[i]$ is a middle element (prob 0.8), the size of the array shrinks to at most $\frac{8n}{10}$.
- The size can shrink so much at most $\log_{\frac{1}{2}} n$ times:

$$n\left(\frac{8}{10}\right)^{\log_{\frac{1}{2}} n} = n\left(\frac{8}{10}\right)^{-\log_{2} \frac{1}{2}} = n \cdot \frac{1}{n} = 1$$

Note: In some applications, we may not get a middle element, but in most applications, we will probably get a middle element.
Analysis [CLRS 9.2 shows a different method of analysis]

Set a geometric random variable:

$T_i = \text{how many recursive applications until array shrunk to size } \leq \left(\frac{3}{10}\right)^i n.$

$$\text{Running time } \leq \sum_{i=0}^{\log_{10} n} O\left(\left(\frac{3}{10}\right)^i n\right) \cdot T_i$$

$$E[T_i] \leq \frac{10}{8}$$

Thus,

$$E[T] \leq E \sum_{i=0}^{\log_{10} n} O\left(\left(\frac{3}{10}\right)^i n\right) \cdot T_i$$

linearity of expectation $= \sum_{i=0}^{\log_{10} n} O\left(\left(\frac{3}{10}\right)^i n\right) \cdot E[T_i]$

$= O(n) \cdot \sum_{i=0}^{\log_{10} n} \left(\frac{3}{10}\right)^i$

geometric sum $= O(n)$
Randomness Saves Time

Polynomial multiplication verification

Given $p, q$ of deg $\leq n-1$

$\Rightarrow$ of deg $\leq 2n-2$

Is $p \cdot q = r$?

Using FFT, have $\Theta(n \log n)$ algorithm.

Randomized Algorithm

1. Set $X = \{0, \ldots, 100n\}$.

$\Theta(n)$ 2. Pick uniformly at random $x \in X$.

$\Theta(n)$ 3. Return whether $p(x) \cdot q(x) = r(x)$.

Fact A poly of deg $\leq d$ has at most $d$ roots.

Hence, if $p \cdot q \neq r$, then

$$P\left(\sum_{x \in X} p(x)q(x) = r(x)\right) = \frac{2n-2}{100n} \leq 0.02$$
Markov Inequality

If $X$ is a non-negative random variable, then $P(X \geq c \cdot E[X]) \leq \frac{1}{c}$.

Proof

If otherwise

$E[X] \geq c \cdot E[P(X \geq c \cdot E[X])]$

$> c \cdot E[X] - \frac{1}{c}$

$= E[X]$ (contradiction!)

Cor: Any algorithm with expected running time $T$ can be transformed to an algorithm whose running time is at most $c \cdot T$ always, but with probability $\leq \frac{1}{c}$ outputs "I don't know the answer."

Terminology:
- low expected running time, "Las Vegas" algorithm
- low running time always mostly correct, "Atlantic City" algorithm