Randomized Algorithms

Why randomized?
- Checking Matrix multiply
- Quicksort

Randomized or Probabilistic Algorithms

- Algorithm that generates a random number \( r \in \{1, \ldots, K\} \) and makes decisions based on \( r \)'s value.
- On the same input on different executions, randomized algorithm may:
  - run for a different number of steps
  - produce different outputs

Monte Carlo
- runs in poly time always
- prob (output is correct) > high

Las Vegas
- always produces correct output
- runs in expected poly time

Variation due to \( r \)
Matrix Product

\[ C = A \times B \]

Simple algorithm: \( O(n^3) \) multiplications

Strassen: Multiply two \( 2 \times 2 \) matrices using 7 multiplications: \( O(n^{\log_2 7}) \)

Coppersmith-Winograd: \( O(n^{2.376}) \)

Matrix Product Checker

Given \( n \times n \) matrices \( A, B, C \)
Goal: check if \( A \times B = C \) or not?

Question: Can we do better than multiply?

We will see an \( O(n^2) \) algorithm that:

\[
\begin{align*}
\text{if } A \times B = C, \text{ then } \text{prob}[\text{output = YES}] &= 1 \\
\text{if } A \times B \neq C, \text{ then } \text{prob}[\text{output = YES}] &\leq \frac{1}{2}
\end{align*}
\]

We will assume entries in matrices \( E \{0, 1\} \)
assume mod 2 arithmetic
Friivald's algorithm

Choose a random binary vector $r[1 \ldots n]$ such that $Pr[r_i=1] = \frac{1}{2}$ independently for $i = 1, \ldots n$.
If $A(Br) = Cr$, then output 'YES'; else output 'NO'.

Observations:
$O(n^2)$ time, since 3 matrix vector multiplications for $Br$, $A(Br)$, $Cr$.

If $AB = C$, then $A(Br) = (AB)r = Cr$ and algorithm always outputs YES.

Analyzing correctness if $AB \neq C$

Claim: If $AB \neq C$, then $Pr[ABr \neq Cr] > \frac{1}{2}$

Let $D = AB - C$. Our hypothesis is thus that $D \neq 0$. Clearly, there exists $r$ such that $Dr \neq 0$.

Let $D \neq 0$. We need to show that there are many $r$ such that $Dr \neq 0$. Specifically, $Pr[Dr \neq 0] > \frac{1}{2}$ for a randomly chosen $r$. 
Analyzing Correctness (cont'd.)

If $Dr \neq 0$, we would output 'No', done

$Dr = 0$ case

$D = AB - C \neq 0 \Rightarrow \exists i,j \text{ s.t. } dij \neq 0$

Fix vector $v$ which is 0 in all coordinates except for $v_j = 1$

$(Dv)_i = dij \neq 0$ implying $Dv \neq 0$

Take any $r$ that can be chosen by our algo.

We are looking at the case where $Dr = 0$.

$r' = r + v$

$r'$ same as $r$ except $r'_j = (vj + v_j) \mod 2$

$Dr' = D(r + v) = 0 + Dv \neq 0$

$r$ to $r'$ is 1 to 1 because if $r'' = r + v$ then $r' = r'' + v$

Number of $r'$ for which $Dr' \neq 0 \succ$ Number of $r$ for which $Dr = 0$

$\Rightarrow P[r | Dr \neq 0] \succ \frac{1}{2}$
Quicksort

C.A.R. Hoare (1962)

Divide & conquer algorithm but work mostly in divide step rather than combine
Sorts "in place" like insertion sort and unlike merge sort

Different variants:
Basic: good in average case (for a random input)
Median-based pivoting: uses median finding
Randomized: good for all inputs in expectation
Las Vegas algorithm

requires o(n)
auxiliary space
Quicksort

n-element array A

Divide:
1. Pick a pivot element $x$ in A
   Partition the array into sub-arrays
   L E G
   $<x \quad x \quad >x$

Conquer: Recursively sort subarrays $L$ and $G$
Combine: Trivial

Basic Quicksort

pivot $x = A[1]$ or $A[n]$, first or last element
- Remove, in turn, each element $y$ from A and
- Insert $y$ into $L$, $E$ or $G$ depending on
  the comparison with pivot $x$
- Each insertion and removal takes $O(1)$ time
- Partition step takes $O(n)$ time
- To do this in place: see code in CLRS p 171
Basic Quicksort analysis

- Input sorted or reverse sorted
- Partition around min or max elements
- One side $L$ or $G$ has $n-1$ elements, other 0

$$ T(n) = T(0) + T(n-1) + \Theta(n) $$
$$ = \Theta(1) + T(n-1) + \Theta(n) $$
$$ = T(n-1) + \Theta(n) $$
$$ = \Theta(n^2) \text{ (arithmetic series)} $$

Does well on random inputs in practice

Pivot Selection Using Median Finding

Can guarantee balanced $L$ and $G$ using rank/median selection algorithm that runs in $\Theta(n)$ time

$$ T(n) = 2 T(n/2) + \Theta(n) + \Theta(n) $$

recursive median selection

$$ T(n) = \Theta(n \log n) $$

This algorithm is slow in practice and loses to mergesort.
Randomized Quicksort

X is chosen at random from array A (at each recursion, a random choice is made)

Expected time is O(n \log n) for all input arrays A

See CLRS p181-4 for analysis; we will analyze here a variant quicksort

"Paranoid" Quicksort

Repeat

choose pivot to be random element of A

Perform Partition

Until resulting partition is such that

\[ |L| \leq \frac{3}{4} |A| \quad \text{and} \quad |A| \leq \frac{3}{4} |A| \]

Recurse on L and G
"Paranoid" Quicksort Analysis

Good call: sizes of L & G ≤ \( \frac{3}{4} n \) each
Bad call: one of L or G is \( > \frac{3n}{4} \)

Pivots: bad good bad

\( \frac{1}{4} n \) \( \frac{1}{2} n \) \( \frac{1}{4} n \)

A call is good with probability \( \frac{1}{2} \)

Let \( T(n) \) be an upper bound on the expected running time on any array of \( n \) size \( T(n) \) comprises:

- Time needed to sort left subarray
- Time needed to sort right subarray
- The number of iterations to get a good call * \( c \cdot n \)
  
  Cost of partition
Expectations

\[ T(n) \leq \max_{n/4 \leq i \leq 3n/4} (T(i) + T(n-i)) + E(\text{# iterations}) \cdot cn \]

\[ E(\text{# iterations}) \leq 2 \quad \text{since prob of good call} > 1/2 \]

\[ = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 2cn \]

\[ \log_4(2cn) \quad \text{levels} \]

\[ \frac{1}{4} \rightarrow \frac{3}{4} \rightarrow \frac{9}{16} \rightarrow \ldots \rightarrow \frac{\log_4(2cn)}{3} \text{ levels} \]

\[ \Theta(1) \quad \Theta(1) \]

2cn work at each level

\[ \max \log_4(2cn) \quad \text{levels} \]

\[ \Theta(n \log n) \quad \text{expected runtime} \]