Randomized Algorithms

Why randomized?
- Checking Matrix multiply
- Quicksort

Randomized or Probabilistic Algorithms

- Algorithm that generates a random number $r \in \{1, \dots R\}$ 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	Las Vegas
• runs in poly time always	• always produces correct output
• prob(output is correct) > high	• runs in expected poly time

variation due to r

Matrix Product

$C = A \times B$

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

Strassen: Multiply two 2×2 matrices using 7 multiplications: $O(n^{2.81})$ $\log_2 7$

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:

if $A \times B = C$, then prob[output = YES] = 1

if $A \times B \neq C$, then prob[output = YES] $\leq 1/2$

We will assume entries in matrices $\in \{0,1\}$.

assume mod 2 arithmetic

Frievald's algorithm

Choose a random binary vector $r[1 \ldots n]$ such that $Pr[r_i = 1] = 1/2$ 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 $Prob[ABr \neq Cr] \geq 1/2$

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

We need to show that there are many r such that $Dr \neq 0$.

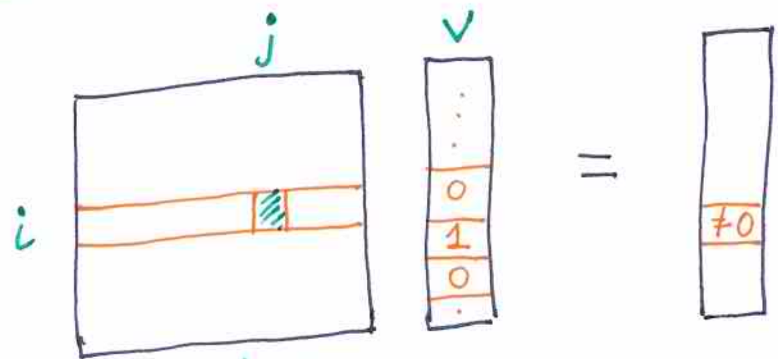
Specifically, $Prob[Dr \neq 0] \geq 1/2$ for a randomly chosen r

Analyzing Correctness (contd.)

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

$Dr = 0$ case

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



$d_{ij} \neq 0$

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

$(Dv)_i = d_{ij} \neq 0$ implying $Dv \neq 0$

Take any r that can be chosen by our algo.
We are looking at the case where $Dr = 0$.

$r' = r + V$ ← vector addition

r' same as r except $r_j' = (r_j + v_j) \bmod 2$

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

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

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

$\Rightarrow \Pr[Dr \neq 0] \geq 1/2$

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 ← requires $O(n)$ auxiliary space

Different variants:

Basic: good in average case (for a random input)

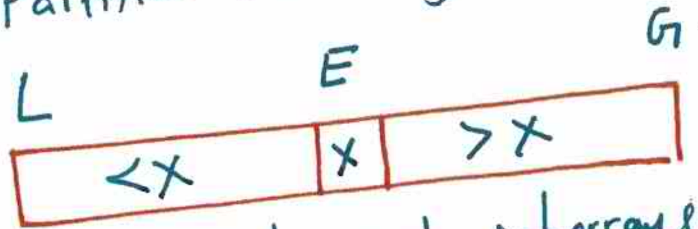
Median-based pivoting: uses median finding

Randomized: good for all inputs in expectation
Las Vegas algorithm

Quicksort

n-element array A

Divide:
1. Pick a pivot element x in A
   Partition the array into sub-arrays



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

$$T(n) = T(0) + T(n-1) + \theta(n)$$ divide step

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

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

$$T(n) = 2\,T\left(\frac{n}{2}\right) + \theta(n) + \theta(n)$$

recursive median selection divide

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

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

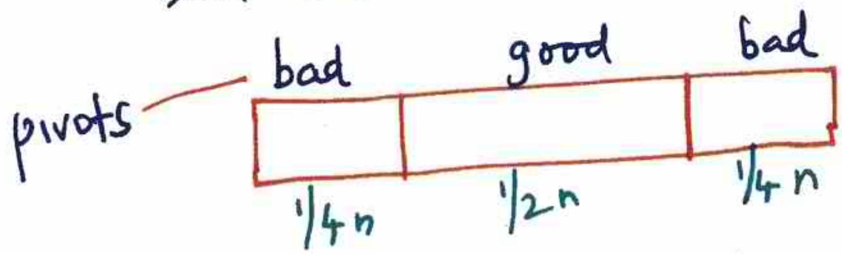
"Paranoid" Quicksort

Repeat
choose pivot to be random element of A
Perform Partition
Until resulting partition is such that
$|L| \leq \frac{3}{4}|A|$ and $|G| \leq \frac{3}{4}|A|$

Recurse on L and G

"Paranoid" Quicksort Analysis

Good call : sizes of L & G $\leq \frac{3}{4}n$ each
Bad call : One of L or G is $> \frac{3n}{4}$



A call is good with probability $> 1/2$

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

$T(n)$ comprises :

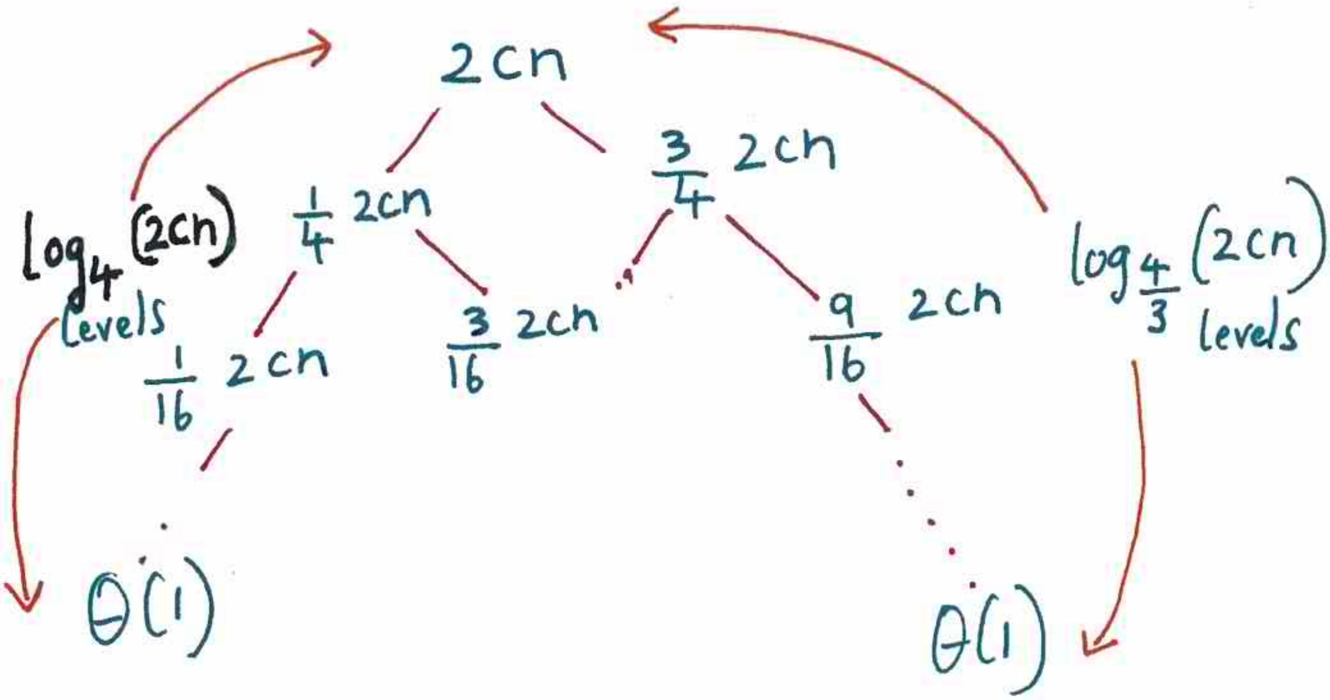
- Time needed to sort left subarray
- Time needed to sort right subarray
- The number of iterations to get a good call * $\underbrace{c \cdot n}_{\text{cost of partition}}$

Expectations

$$T(n) \le \max_{n/4 \le i \le 3/4 n} \left(T(i) + T(n-i)\right) + E(\#\text{iterations}) \cdot cn$$

$E(\#\text{ iterations}) \le 2$ since prob of good call $> 1/2$

$$= T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 2c.n$$



2cn work at each level

max $\log_{\frac{4}{3}} (2cn)$ levels

$\Theta(n \log n)$ expected runtime