| Lecture 02 Randomized Algorithms: Matrix Product Verification & QuickSort | L02.1<br>6.046<br>02/06/2020 |
|---|---|

Today: Why, in our precise and well specified algorithms, would we admit randomization?

Reading:
CLRS 5.1, 5.2, 5.3
CLRS Chpt 7

---

## Randomized (or Probabilistic) Algorithms

- Algorithm that generates a random number $r \in \{1, \ldots, R\}$ and makes decisions based on value of $r$.
- Given the same input, different executions of the same randomized algorithm may
  - have different running time, by carrying out different sequence of operations
  - produce different output

### Monte Carlo

- runs in polynomial time, always
- prob (output is correct) → "high"

variation in each case due to $r$

### Las Vegas

- runs in expected polynomial time
- always gives correct output

---

## Computing Matrix Products

$C = A \times B$ (nxn matrices)

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

Strassen: A pair of 2x2 matrices requires 7 multiplications; $O(n^{2.81})$ $\log_2 7$

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

## Matrix Product Verification

Given: nxn matrices $A, B, \& C$

Goal: Check whether or not $C = A \times B$

## Can we do better than matrix multiply?

Imagine checking by multiplying both sides of the equation by the same random vector and checking agreement. [Matrix-vector product is $O(n^2)$]

Here, to simplify analysis, assume entries of matrices are only elements of $\{0, 1\}$ and arithmetic is mod 2.

## Consider Frievald's algorithm

Choose a random binary vector $r[1 \ldots n]$ s.t.

$Pr[r_i = 1] = \frac{1}{2}$ $\forall i \in \{1, \ldots, n\}$ independently.

If $\begin{cases} A(Br) = Cr & \text{output "YES"} \\ \text{Otherwise} & \text{output "No"} \end{cases}$

Monte Carlo because always fast, efficient but can be incorrect.

## Observations:

Running time: Three matrix-vector multiplications
$[Br, A(Br), Cr] \Rightarrow O(n^2)$

Always correct for case $C = A \times B$ because $A(Br) = (AB)r = Cr$ and algorithm always says "YES" $\forall r$.

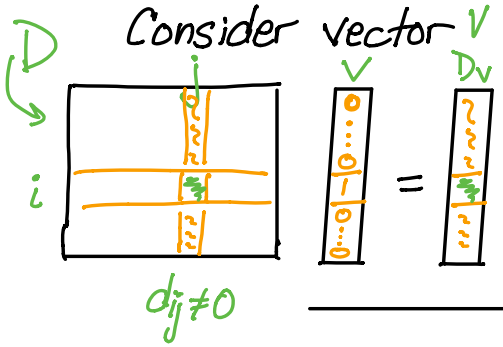
So algorithm has excellent "sensitivity" (true positive rate = 100%), but how is its "specificity" (true negative rate)?

## Analyzing correctness for $C \neq A \times B$

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

**Proof:** Let $D = C - AB$. We are looking at the case where $D \neq 0$. There exists some $r$ s.t. $Dr \neq 0$, but we want to show there are many such $r$.

Consider vector $v$ which is all zeros except $v_j = 1$




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

For any $r$ chosen by our algorithm, if $Dr = 0$ then

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

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

But $r$-to-$r'$ is a one-to-one mapping because if $r' = r + v$ and $r' = r'' + v$, then $r = r''$

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

$\Rightarrow \Pr[Dr \neq 0] \geq 1/2$ when $C \neq A \times B$ ∎

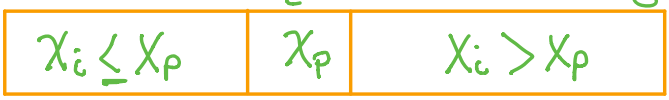
## QuickSort [Hoare, 1962]

- Divide & Conquer algorithm, with work mostly in divide rather than combine.
- Sorts in place, like Insertion sort and unlike Merge sort.

## "Core" QuickSort

Given an $n$-element array $A$, output the sorted array.

Divide: Pick a pivot element $x_p$ in $A$, and partition the array into subarrays.




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

Combine: Trivial

## Basic QuickSort

$x_p = A[1]$ or $A[n]$ (pivot is first or last element)

- Compare pivot to each element in turn & optionally swap
  - Each comparison (w/swap) takes $\Theta(1)$ time
- Partition takes $\Theta(n)$ time
- Can be done in place (CLRS, pp. 171ff)

## Analysis:

- Consider case of input already sorted or reverse sorted (worst-case)
- Partition occurs around min or max element
- One side (L or G) has $n-1$ elements & other 0

$$T(n) = T(0) + T(n-1) + \Theta(n)$$ ← Divide step of partitioning

$$= \Theta(1) + T(n-1) + \Theta(n)$$

$$= T(n-1) + \Theta(n) = \underline{\Theta(n^2)}$$

- (In practice, does well on random inputs)

## Median-based Pivoting

If we could guarantee balanced L and G by using the median as the pivot, would this improve running time?

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \underline{\Theta(n \log n)}$$

- Significant improvement
- L03 introduces median-finding (Algorithm is slow in practice; loses to mergesort)

**Exercise:** Any finite ratio split (90%:10% or 99%:1%) also leads to $\Theta(n \log n)$ running time.

# Randomized QuickSort

- $x_p$ chosen at random from array $A$
  (and at each recursion, a new random choice is made)
- Expected running time is $\Theta(n \log n)$ for all input arrays $A$
  - Las Vegas algorithm – always correct result

In RANDOMIZED algorithm analysis, WORST-CASE behavior is still WORST-CASE for input, but averaged over random number possibilities. (The adversary can choose the input, but without knowledge or control of the random number behavior.)

NOTE DISTINCTION IN TERMINOLOGY:

AVERAGE-CASE analysis – averages over inputs

EXPECTED-CASE analysis – average over random choices

- CLRS (pp. 181-184) provides full analysis; we will analyze special case of Paranoid QuickSort

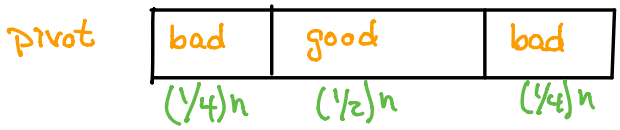
# Paranoid QuickSort

Repeat
- Choose pivot as random element of $A$
- Perform Partition
- Exit if $|L| \leq \frac{3}{4}|A|$ and $|G| \leq \frac{3}{4}|A|$

Recurse on $L$ and $G$

## Analysis:

- Good call: sizes of $L$ and $G$ each $\leq \frac{3}{4}|A|$
- Bad call: size of either $L$ or $G$ $> \frac{3}{4}|A|$



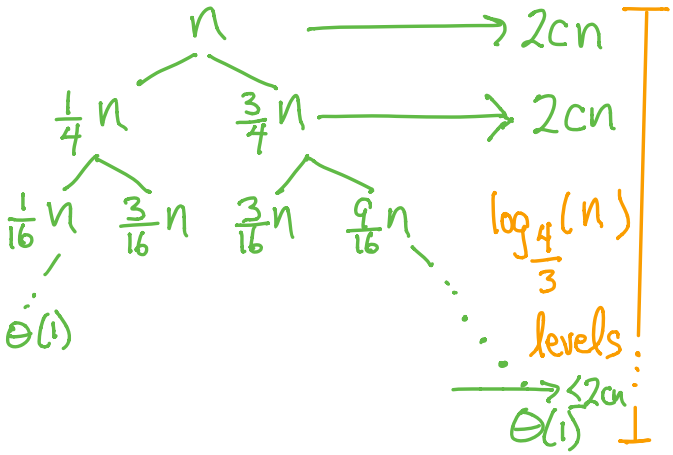

- A call is good with probability $\geq \frac{1}{2}$
- Let $T(n)$ be upper bound on expected running time on any array of size $n$

$T(n)$ includes:

- Time to sort left subarray
- Time to sort right subarray
- Number of iterations to find good call $\times c \cdot n$ (Time for each partition)

$$T(n) \leq \max_{\frac{n}{4} \leq i \leq \frac{3n}{4}} [T(i) + T(n-i)] + \underbrace{E(\#\text{iterations})}_{\leq 2 \text{ because prob. of good call} \geq \frac{1}{2}} \cdot cn$$

$$= T(n/4) + T(3n/4) + 2cn$$




- $2cn$ work at each level
- max $\log_{\frac{4}{3}}(n)$ levels

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

# Useful Probability Theory



## Markov Inequality:

For discrete, nonnegative random variable $Z$ with finite expectation value,

$$\Pr[Z \geq a] \leq \frac{E[Z]}{a} \quad \forall \text{ constant } a > 0$$

Bounds the probability that a random variable exceeds some value.

Proof: Let $Z' = \begin{cases} a & \text{if } Z \geq a \\ 0 & \text{otherwise} \end{cases}$

Note $Z, Z' \geq 0$, and $Z' \leq Z$ with probability 1

$$E[Z'] = a \cdot \Pr[Z \geq a] + 0 \cdot \Pr[Z < a] = a \cdot \Pr[Z \geq a]$$

$E[Z'] \leq E[Z]$ Thus, $\Pr[Z \geq a] \leq \frac{E[Z]}{a}$

The Markov inequality provides a way to convert a Las Vegas algorithm into a Monte Carlo one: Run the Las Vegas algorithm for a time $cT$, where $T$ is the expected running time. The new algorithm will complete in an efficient time but may not give a correct answer (because it may not have finished).

## Union Bound

The probability of a finite or countably infinite union of events $E_i$ is at most the sum of their individual probabilities.

the probability that at least one occurs

Intuitively, the bound is achieved for disjoint events; any overlap diminishes this value.

$$\Pr[E_1 \text{ OR } E_2 \text{ OR } E_3 \text{ OR } \cdots \text{ OR } E_n] \leq \sum_{i=1}^{n} \Pr[E_i]$$

## Chebyshev Inequality

If $Z$ has expectation value $\mu$ and strictly positive variance $\sigma^2$, then for all real $k > 0$

$$\Pr[|Z-\mu| \geq k] \leq \frac{\sigma^2}{k^2}$$

**Proof:** Let $X = (Z-\mu)^2 \Rightarrow X \geq 0$ and $E[X] = \sigma^2$

Apply Markov inequality to $X$:

$$\Pr[X \geq k^2] \leq \frac{\sigma^2}{k^2}$$

$$\Pr[(Z-\mu)^2 \geq k^2] = \Pr[|Z-\mu| \geq k] \leq \frac{\sigma^2}{k^2}$$

## Chernoff Bound

Let $\{X_1, \ldots, X_n\}$ be independent variables taking values in $\{0,1\}$. Let $X$ be their sum and $\mu = E[X]$ be the expectation value of their sum.

Then for any $\beta > 0$,

- $\Pr[X > (1+\beta)\mu] < e^{-\beta^2\mu/3}$ for $0 < \beta < 1$
- $\Pr[X > (1+\beta)\mu] < e^{-\beta\mu/3}$ for $\beta > 1$
- $\Pr[X < (1-\beta)\mu] < e^{-\beta^2\mu/2}$ for $0 < \beta < 1$

You are not responsible for the proof, which can be found in internet resources.

We can use the Chernoff bound to estimate how likely it is for randomized QuickSort to take substantially longer than the expected case. Consider the case where a bad pivot is outside the 1/10 — 9/10 region, and let $Y_{i,k}$ be the indicator random variable for a bad pivot for the subarray containing element $a_i$ at recursion depth $k$, so $E[Y_{i,k}] = 0.2$.

For recursion depth $K$, the probability of exceeding the expected number of bad pivots by at least 50% is

$$\Pr\left[\sum_{k=1}^{K} Y_{i,k} \geq 0.2K + 0.1K\right] \leq e^{\frac{-(0.1)^2 K}{2}} = e^{-0.005K}$$

Let $K = 400 \ln n$, so $\Pr\left[\sum_{k=1}^{K} Y_{i,k} \geq 0.3K\right] \leq \frac{1}{n^2}$.

If, instead, $\sum_{k=1}^{K} Y_{i,k} < 0.3K$ and $0.7K$ good pivots were selected, what is the size of the subarray containing $a_i$?

$$\text{Size} \le n \cdot \left(\frac{9}{10}\right)^{0.7K} = n\left(\frac{9}{10}\right)^{280 \ln n} = n \cdot n^{280 \ln(9/10)} \approx n^{-28.5} < 1$$

That is, the size of $a_i$'s subarray is at most one. For randomized QuickSort to complete in $400 \ln n$ recurrence levels, we need this to be true for all $n$ $a_i$'s. By the Union Bound, this probability is $\le n\left(\frac{1}{n^2}\right) = \frac{1}{n}$.