Lecture 6: Randomized Algorithms

- Check matrix multiplication
- Quicksort

Randomized or Probablistic Algorithms

What is a randomized algorithm?

- Algorithm that generates a random number $r \in \{1, ..., R\}$ and makes decisions based on r's value.
- On the same input on different executions, a randomized algorithm may
  - Run a different number of steps
  - Produce a different output

Randomized algorithms can be broadly classified into two types- Monte Carlo and Las Vegas.

Monte Carlo	Las Vegas
runs in polynomial time always	runs in expected polynomial time
output is correct with high probability	output always correct

Matrix Product

$$C = A \times B$$

Simple algorithm: $O(n^3)$ multiplications.
Strassen: multiply two 2×2 matrices in 7 multiplications: $O(n^{\log_2 7}) = O(n^{2.81})$
Coppersmith-Winograd: $O(n^{2.376})$

Matrix Product Checker

Given $n \times n$ matrices A, B, C, the goal is to check if $A \times B = C$ or not.

Question. Can we do better than carrying out the full multiplication?

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

- if $A \times B = C$, then $Pr[\text{output=YES}] = 1$.
- if $A \times B \neq C$, then $Pr[\text{output=YES}] \leq \frac{1}{2}$.

We will assume entries in matrices $\in \{0, 1\}$ and also that the arithmetic is mod 2.

Frievald's Algorithm

Choose a random binary vector $r[1...n]$ such that $Pr[r_i = 1] = 1/2$ independently for $r = 1, ..., n$. The algorithm will output 'YES' if $A(Br) = Cr$ and 'NO' otherwise.

Observation

The algorithm will take $O(n^2)$ time, since there are 3 matrix multiplications Br, $A(Br)$ and Cr of a $n \times n$ matrix by a $n \times 1$ matrix.

Analysis of Correctness if $AB \neq C$

Claim. If $AB \neq C$, then $Pr[ABr \neq Cr] \geq 1/2$.

Let $D = AB - C$. Our hypothesis is thus that $D \neq 0$. Clearly, there exists r such that $Dr \neq 0$. Our goal is to show that there are many r such that $Dr \neq 0$. Specifically, $Pr[Dr \neq 0] \geq 1/2$ for randomly chosen r.

$D = AB - C \neq 0 \implies \exists\ i, j$ s.t. $d_{ij} \neq 0$. 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 algorithm. We are looking at the case where $Dr = 0$. Let

$$r' = r + v$$

Since v is 0 everywhere except v_j, r' is the same as r exept $r'_j = (r_j + v_j) \mod 2$. Thus, $Dr' = D(r + v) = 0 + Dv \neq 0$. We see that there is a 1 to 1 correspondence between r and r', as if $r' = r + V = r'' + V$ then $r = r''$. This implies that

$$\text{number of } r' \text{ for which } Dr' \neq 0 \geq \text{number of } r \text{ for which } Dr = 0$$

From this we conclude that $Pr[Dr \neq 0] \geq 1/2$

Quicksort

Divide and conquer algorithm but work mostly in the divide step rather than combine. Sorts "in place" like insertion sort and unlike mergesort (which requires $O(n)$ auxiliary space).

Different variants:

- Basic: good in average case
- Median-based pivoting: uses median finding
- Random: good for all inputs in expectation (Las Vegas algorithm)

Steps of quicksort:

- Divide: pick a pivot element x in A, partition the array into sub-arrays L, consisting of all elements $< x$, G consisting of all elements $> x$ and E consisting of all elements $= x$.
- Conquer: recursively sort subarrays L and G
- Combine: trivial

Basic Quicksort

Pivot around $x = A[1]$ or $A[n]$ (first or last element)

- Remove, in turn, each element y from A
- 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 CLRS p. 171

Basic Quicksort Analysis

If input is sorted or reverse sorted, we are partitioning around the min or max element each time. This means one of L or G has $n-1$ elements, and the other 0. This gives:

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ &= \Theta(1) + T(n-1) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

However, this algorithm 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. The first $\Theta(n)$ below is for the pivot selection and the second for the partition step.

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(n) \\ T(n) &= \Theta(n \log n) \end{aligned}$$

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 p.181-184 for the analysis of this algorithm; we will analyze a variant of this.

"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

Let's define a "good pivot" and a "bad pivot"-
  Good pivot: sizes of L and $G \leq \frac{3}{4}n$ each
  Bad pivot: one of L and G is $\leq \frac{3}{4}n$ each

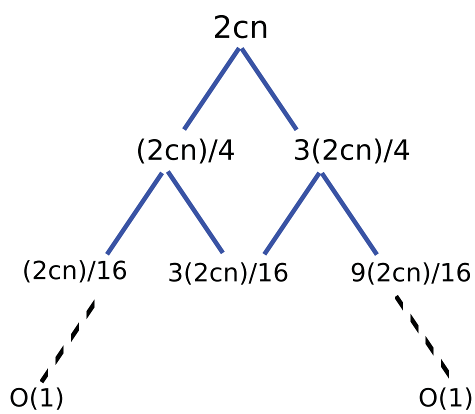
bad pivots	good pivots	bad pivots
$\frac{n}{4}$	$\frac{n}{2}$	$\frac{n}{4}$

We see that a pivot 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. Denote as $c \cdot n$ the cost of the partition step

Expectations



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

Now, since probability of good pivot $> \frac{1}{2}$,

$$E(\#iterations) \leq 2$$

$$T(n) \leq T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + 2cn$$

We see in the figure that the height of the tree can be at most $\log_{\frac{4}{3}}(2cn)$ no matter what branch we follow to the bottom. At each level, we do a total of $2cn$ work. Thus, expected runtime is $T(n) = \Theta(n \log n)$