Divide & Conquer

Examples

- Median Finding
- Integer Multiplication

} TODAY

- Matrix Multiplication } see notes at the end
- Polynomial "
- Fast Fourier Transform

} Lec 9

MAIN IDEA

Given problem of size n:

(subproblems can overlap)

Divide it into a subproblems of size n/b (divide)

Solve each subproblem recursively (conquer)

Combine solutions of subproblems to get solution of original problem (rule)

Resulting runtime:

$$T(n) = a \cdot T(n/b) + (\text{time to combine})$$

Analyze using master theorem or ad-hoc (unravel the recursion)

APPLICATION 1: Median Finding

Given a set S of n numbers, define

$\text{rank}(x) = \#$ of elements of $S \leq x$

upper median = element of rank $\left\lceil \frac{n+1}{2} \right\rceil$

lower median = element of rank $\left\lfloor \frac{n+1}{2} \right\rfloor$

If n is odd, these are equal

Eg: Median $(\{2, -5, 3, 10, 1, -1, 8\}) = 2$

We will solve a more general problem:

> Given a set S of n numbers and a number $i \in \{1, 2, 3, \ldots, n\}$
>
> Find the element $x \in S$ s.t. $\text{rank}(x) = i$
> (that is, the i-th smallest element)

Naïve algorithm: Sort and return i^{th} element of sorted list

Running Time: $O(n \lg n)$

e.g., using mergesort

Challenge: Can we do better? $O(n)$?

$\boxed{\text{[Blum, Floyd, Pratt, Rivest, Tarjan 1973] } O(n) \text{ using divide and conquer !!!}}$

Idea:

1. Pick some $x \in S$ cleverly *(will see how)

2. Compute $L = \{ y \in S \mid y < x \}$ (no repeats, w.l.o.g.)

   $G = \{ y \in S \mid y > x \}$

$S = [\ L \mid x \mid G\ ]$

$\boxed{\Rightarrow \text{rank}(x) = |L| + 1}$

3. If rank$(x) = i$, DONE ☺

   If rank$(x) > i$, find element of rank i in the subset L

   If rank$(x) < i$, find element of rank $i - \text{rank}(x)$ in the subset G

Why do we need to pick x cleverly?

e.g., execution for input $S = \{8, 4, 2, 1, 10, 9\}$
$i = 5$

ADVERSARY
(i.e., the BAD guy)

choose $x = 1$
$L = \emptyset$
$G = S \setminus \{1\}$

choose $x = 2$
$L = \emptyset$
$G = S \setminus \{1, 2\}$

⋮

VERY unbalanced recursion

$\Rightarrow$ Reduction in size only 1 per level
$\Rightarrow$ $\Theta(n)$ levels ($\Theta(n)$ work on each level)
$\boxed{\Rightarrow \Theta(n^2) \text{ total running time}}$

How to pick x cleverly?

- Need to pick x so rank(x) is not extreme.
- x = median(S) is best, but that's what we are trying to solve!
- ... but we don't need the median!

  Just an x such that $\underbrace{\max\{\text{rank}(x), n - \text{rank}(x)\} \le c \cdot n}$ for const. $c < 1$
  is good enough.

c-balanced x

$\Rightarrow T(n) = T(\underbrace{c \cdot n}) + O(n)$

$\Rightarrow T(n) = O(n)$

size reduction by a factor of $\frac{1}{c}$ ← const!

THE ALGORITHM.

(Assume $|S| = n$ is a power of 10. If not, add enough small numbers to make it so. This increases the size of S by a factor of 10, at most)

$O(n)$ time

1. Divide the n elements into $\frac{n}{5}$ groups of 5 elements each.

2. Sort each group of 5 elements
$\Rightarrow$ Find the median of each group

(Note: Each group has $O(1)$ size $\Rightarrow$ sorting takes $O(1)$ time)

$T(\frac{n}{5})$

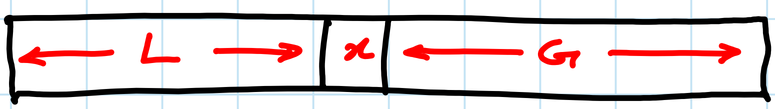
3. Recursively find the median x of the $n/5$ group medians.

$O(n)$ time

4. (As before) Find sets L and G s.t

$$L = \{y \in S \mid y < x\}$$

$$G = \{y \in S \mid y > x\}$$



$$\text{rank}(x) = |L| + 1$$

5. (As before) If $\text{rank}(x) = i$, return x

either $T(|L|)$ $\left\{ \text{If } \text{rank}(x) > i, \text{ find element of rank } i \text{ in } L \right.$

OR

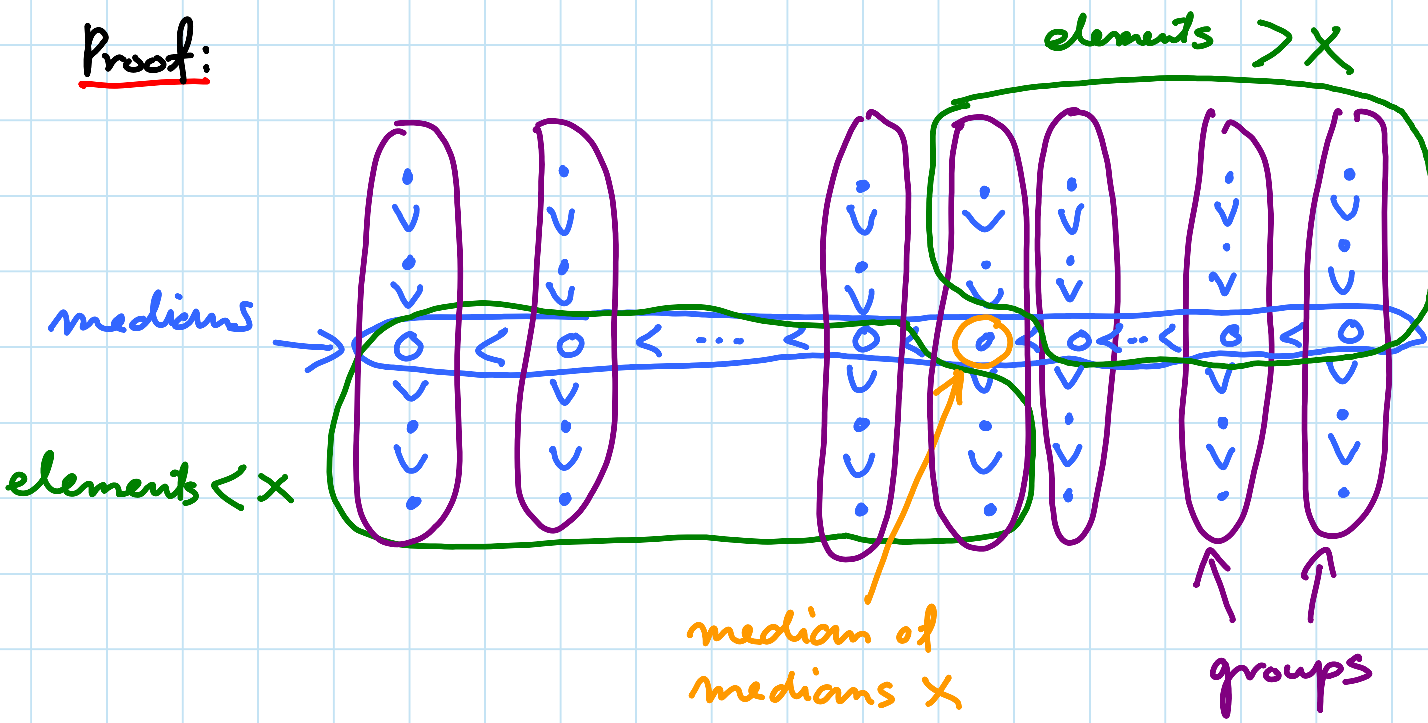
$T(|G|)$ $\left\{ \text{If } \text{rank}(x) < i, \text{ find element of rank } i - \text{rank}(x) \text{ in } G \right.$

ANALYSIS:

Key observation:

The "median of medians" element x is $\frac{3}{4}$-balanced, i.e., $\max\{|L|, |G|\} \leq \frac{3}{4} \cdot n$

Proof:



→ How many elements $\leq x$?

→ at least $\frac{1}{2} \cdot \frac{n}{5} = \frac{n}{10}$ group medians $\leq x$

$\Rightarrow$ at least $3 \cdot \frac{n}{10}$ elements $\leq x$ (3 per each of $\frac{n}{10}$ groups)

$\Rightarrow |L| \geq \frac{3n}{10} \geq \frac{n}{4}$

$\Rightarrow \boxed{n - \text{rank}(x) \leq \frac{3}{4} n}$

→ Similarly, at least $\frac{n}{4}$ elements $\geq x$

$\Rightarrow \boxed{\text{rank}(x) \leq \frac{3}{4} \cdot n}$

$\Rightarrow$ x is indeed $\frac{3}{4}$-balanced

∎

Running time?

$$T(n) = \underbrace{T\left(\tfrac{3}{4} \cdot n\right)}_{\text{standard "conquer" step}} + \underbrace{T\left(\tfrac{n}{5}\right)}_{\text{Additional time needed to find } x\,!} + \underbrace{O(n)}_{\text{Everything else}}$$

Claim: $T(n) \leq c_1 \cdot n$ for some constant c_1

c_2 = the constant from $O(n)$ term

Proof: by induction

Suppose our claim true for $< n$.

Then, $T(n) \le T(\frac{n}{5}) + T(\frac{3n}{4}) + c_2 n$

$$\le \frac{c_1 n}{5} + \frac{3c_1 n}{4} + c_2 n$$

$$= \frac{19}{20} c_1 n + c_2 n$$

$$= c_1 n + \left(c_2 - \frac{c_1}{20}\right) n$$

$$< c_1 n \quad \text{once we set } c_1 > 20 c_2.$$

$\blacksquare$

Why linear time (and not, say, $n \log n$)?

→ Because $1/5 + 3/4 < 1$

→ Significant (constant factor) reduction in problem size per step

→ geometric series

⇒ overall time ≈ time in the first step of recursion

Ex.: what if groups had 7 elements? 3?

APPLICATION 2: INTEGER MULTIPLICATION

INPUT : Two n-bit numbers a, b
GOAL : Compute $a \cdot b$

- grade-school algorithm

$$
\begin{array}{r}
0010 \times \\
1101 \\
\hline
0010 \\
0000 \\
0010 \\
0010 \\
\hline
0011010
\end{array}
$$

\# bit ops $= O(n^2)$
as I need to add n integers, n bit each

- better algorithm?

Use divide and conquer!

Key question: What are the subproblems?

$\boxed{\text{IDEA 1}}$

View $a = 2^{n/2} \cdot X + Y$
$b = 2^{n/2} \cdot W + Z$

all $n/2$-bit numbers

Then, $a \cdot b = 2^n \cdot XW + YZ +$

$2^{n/2} \cdot (XZ + YW)$

product of 2 $n/2$-bit numbers

$$T(n) = 4T(n/2) + \theta(n)$$
$$= \theta(n^{\log_2 4}) \text{ by Master theorem}$$
$$= \theta(n^2)$$

:(back to $\square_1$

$\boxed{\text{IDEA 2 : [Anatoli Karatsuba 1962]}}$

→ Same way as before to partition a, b

→ Compute $X \cdot W$ and $Y \cdot Z$

→ but DO NOT compute $X \cdot Z$ and $Y \cdot W$ separately.

- Compute instead $(X+Y) \cdot (Z+W)$

KEY "MAGIC" IDENTITY :)

$$(X+Y) \cdot (Z+W) = (XZ+YW) + XW + YZ$$

→ We know

$$(x+y)(z+w),\ xw,\ yz$$

⇒ We can compute $xz+yw$

$$= (x+y)(z+w) - xw - yz$$

→ Now $T(n) = 3T(n/2) + \theta(n)$

$$= \theta(n^{\log_2 3}) = \theta(n^{1.58})\ !$$

Is this the best possible?

[Schönage & Strassen 1971] $\theta(n \cdot \lg n \cdot \lg\lg n)$

[Fürer 2007] $n \cdot \lg n \cdot 2^{\theta(\lg^* n)}$

$\lg^* n$ = min number of times you take iterated logs starting with n until you reach ≤ 1.

<u>Note:</u> $\lg^*(\underbrace{2^{65536}}_{\text{\# of atoms in observable universe}}) = 5$

:)

Additional Material (NOT REQUIRED)

① Matrix Multiplication:

- Given two $n \times n$ matrices A and B
  Compute $A \cdot B$.

- Trivial : $O(n^3)$
- Best Possible : $O(n^2)$
  { n^2 elements in each matrix
  need to look at each one at least once

- Subproblems ? Blockwise multiplication

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

each of these is $(n/2) \times (n/2)$
& same for B

$$= \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

8 subproblems ?

$$T(n) = 8T(n/2) + O(n^2) = O(n^3) \quad :($$

Strassen 1969 : 7 subproblems

$$\Rightarrow O(n^{\lg_2 7})$$

!

We did not say what these 7 subproblems are : see CLRS.

(2) A Combinatorial Application of Matrix Mult :

Counting # triangles in a graph

- Given undirected Graph $G = (V, E)$ with n nodes
  output # triangles
  $= \{ (i,j,k) :$ there are edges $(i,j), (j,k), (i,k) \in E \}$

- Trivial: $O(n^3)$

- **Claim** : Time to count # triangles
  $\leq$ Time to multiply two $n \times n$ matrices

Algorithm

- Let A be the adjacency matrix of G

= Matrix Mult

- Compute A^2

Claim : $(i,j)^{th}$ entry of A^2 is the # length-2 paths between i and j

- Initialize counter = 0
- For each i and j:

- if $A[i,j] = 1$, then
  counter $+= A^2[i,j]$
  { if there is an (i,j) edge, each (i,j) path of length 2 defines a triangle }
- else do nothing

• output counter/6
  { since we count each triangle six times }

Total time = Time for matrix-mult + $O(n^2)$

- Improved matrix mult algorithms automatically translate to improved triangle-counting

- We just reduced problem A to problem B
  counting triangles (A), matrix mult (B)
- Will see more in Lecture 19-20