Advanced Algorithms

Lecture 5: Median selection, dynamic programming

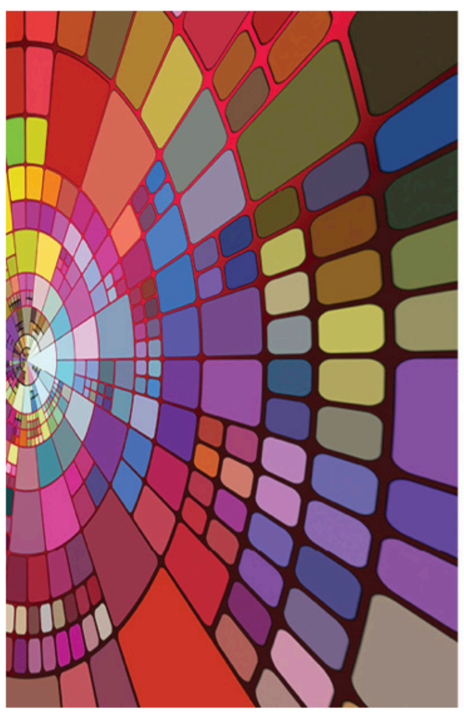
Announcements

- HW 1 due this Friday
- *Video for lecture 3* [hopefully today/tomorrow].

NSF GRFP Info Session

https://gradschool.utah.edu/registration-1/

REGISTRATION



NSF RESEARCH FELLOWSHIPS

FINAL TIPS FOR 2019 APPLICANTS

Thursday, September 5 • 11:30 - 1:00 pm
1110 Spencer Fox Eccles Business Building

NSF-GRFP Overview
Professor Tony Butterfield
College of Engineering Director of Fellowships

NSF-GRFP Success Panel
Current NSF Award Recipients

Pizza Lunch Provided

The Graduate School

NSF Research Fellowships: Final Tips for 2019 Applicants

Thursday, September 5 • 11:30 am - 1:00 pm

Spencer Fox Eccles Buisness Building, Room 1110 (http://bit.ly/2blbrBp)

Are you a first-or second-year graduate student working on a 2019 GRFP application? Deadlines are coming up fast, so right now is the time to polish your application materials. In this special event, Professor Tony Butterfield and winners of last year's NSF Research Fellowships will be on hand to explain the best steps you can take as you polish your applications. Pizza will be served!

Last two lectures

- Divide and conquer: divide into sub-problems, recurse, “combine”
- Analysis (correctness) via induction — need to show that combine step is right
  (Merge Sort example)
- Run time analysis using recurrences:
  $$f(n) = \text{function}\ (n, f(1), f(2), \ldots, f(n\text{-}1))$$
- Recurrence tree, plug-n-chug, guess-and-prove, master theorem, …
- Bring your own recurrence

$n^{\log n} \ll 2^n$.

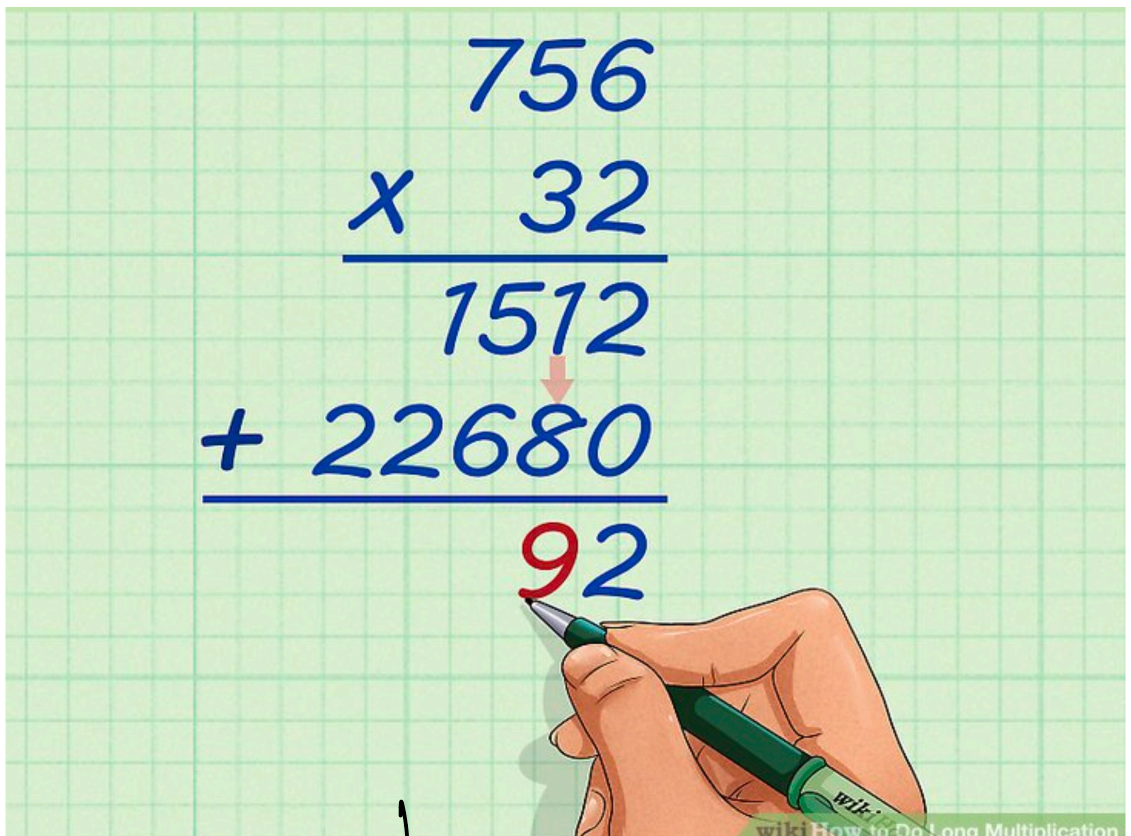
$(\log n)^2 \ll n$

$T(n) = T(n-1) + T\left(\frac{n}{2}\right) + 1.$

[Notes linked via Canvas].

Recap: integer multiplication

(n-digit numbers).



$O(n^2)$ run time

$T(n) = n^2$

- First attempt gave:
  $T(n) = 4\,T(n/2) + O(n)$
- Second attempt using a clever combination gave:
  $T(n) = 3\,T(n/2) + O(n)$
- Former solves to $O(n^2)$, while latter is $O(n^{(\log_2 3)})$

$n^{1.58\ldots}$

$n^{1+\varepsilon}$

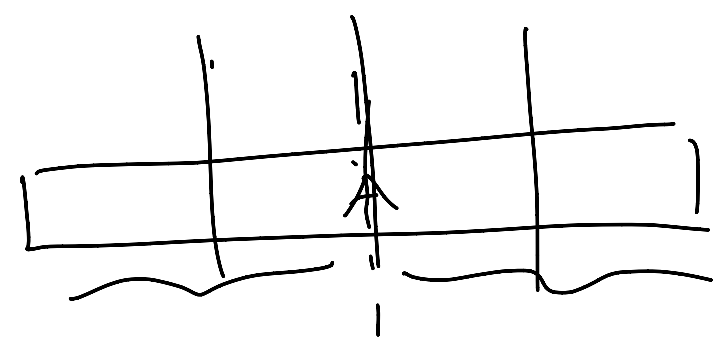
[Toom's algorithm].

Median/order finding

Problem: given *n* (distinct, unsorted) integers A[0], A[1], …, A[n-1] and parameter *k*, find the *k*'th smallest integer

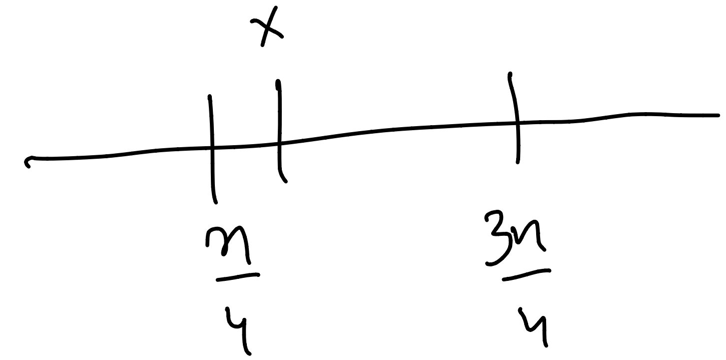
- An easy bound? → first sort, then take k^{th} smallest → $O(n \log n)$ time.

- Divide and conquer? → $O(n)$.



$A = \{ A[0], A[1], \dots, A[n-1] \}$.

q



Almost-median

What if… for any array of size *n,* we have a procedure that:

(a) runs in O(n) time, and

(b) returns an element that is an "almost median", i.e., it is the $\cancel{k}$ r'th largest in the array, for $n/4 < \cancel{k}\ r < 3n/4$

define median as some element of the array that is "in the middle" in sorted order.

Formally, it is the r^{th} smallest elt, where

$$\frac{n}{4} < r < \frac{3n}{4}.$$

$r = \frac{n}{4}$; $k = n$

Running time

Claim: Having access to an almost-median x allows us to reduce the problem size by a factor $< \frac{3}{4}$. (after $O(n)$ work)

Consider: form new array B[] by going over A[] & including only elements $\leq x$. Since x is an almost median, B[] will have $\leq \frac{3n}{4}$ elements.

Suppose size of $B = r$. If $k \leq r$, then the k^{th} smallest element of A is also the k^{th} smallest element of B[]. If $k > r$, idea: construct C[] with elements $> x$

Procedure shows that if we have access to an almost-median, then by doing $O(n)$ work, we can reduce to a problem of size $\le \frac{3n}{4}$.

$$T(n) \le \overbrace{O(n)}^{c \cdot n} + T\left(\frac{3n}{4}\right) + \cancel{\text{time}(\text{almost-median})}$$

$$T(n) \le cn + c \cdot \frac{3n}{4} + T\left(\frac{3}{4} \cdot \frac{3n}{4}\right)$$

$$\le cn + c \cdot \frac{3n}{4} + c \cdot \frac{3}{4} \cdot \frac{3n}{4} + T\left(\left(\frac{3}{4}\right)^3 \cdot n\right)$$

$$\le \ldots \quad \le \underbrace{cn + c \cdot \left(\frac{3}{4}\right) n + c\left(\frac{3}{4}\right)^2 n + \ldots + c\left(\frac{3}{4}\right)^{r-1} \cdot n + T\left(\left(\frac{3}{4}\right)^r \cdot n\right)}$$

We go on until $\left(\frac{3}{4}\right)^r \cdot n = 1 \iff \left(\frac{4}{3}\right)^r = n$

$$\iff r \cdot \log\left(\frac{4}{3}\right) = \log n$$

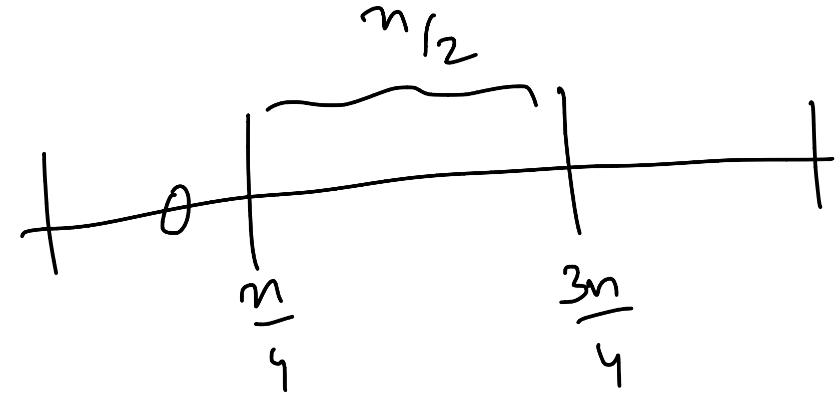
$$r = \frac{\log n}{\log (4/3)}$$

$$T(n) \leq cn\left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \cdots + \left(\frac{3}{4}\right)^{r-1}\right] + T(1)$$

$$1 + \alpha + \alpha^2 + \cdots = \frac{1}{1-\alpha} \quad \text{for any } 0 < \alpha < 1$$

$$\leq 4cn + T(1)$$

Finding an almost-median

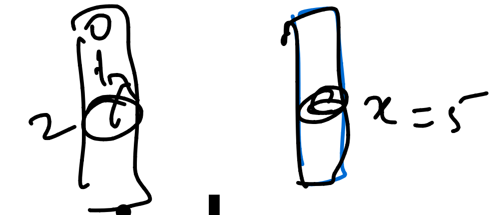


→ In practice → very simple:

- pick a few random elements, go with one that is an almost-median.

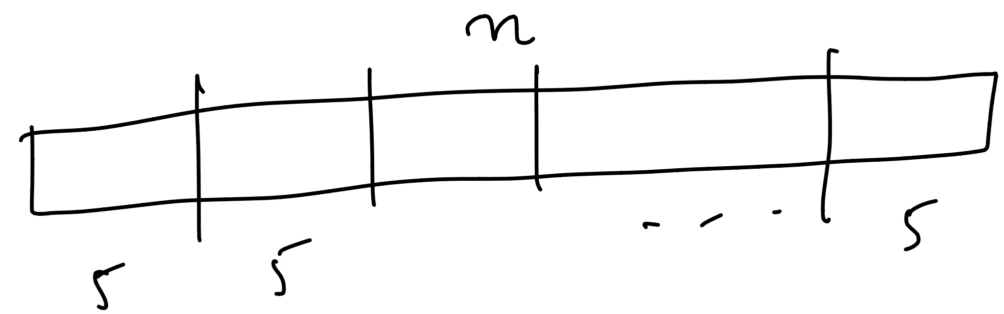
(has non-zero failure prob.)

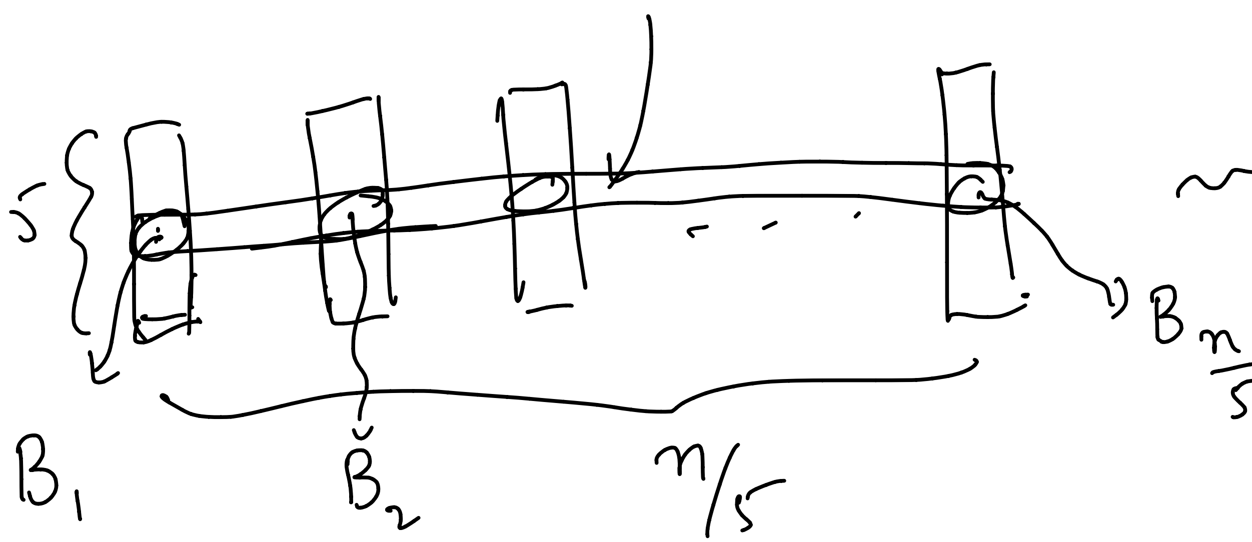
"Median of medians" algorithm

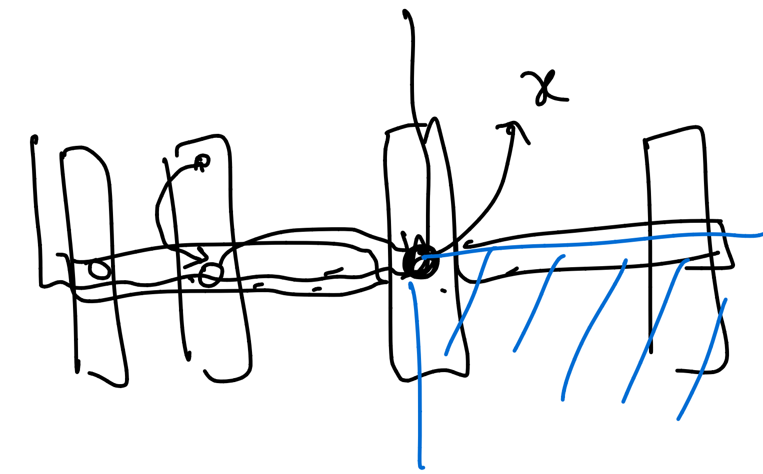


Blum, Floyd, Pratt, Rivest, Tarjan 73

A =







Claim: median (i.e., $\frac{n}{10}$th smallest elt of B[]) is an ⇝ x.

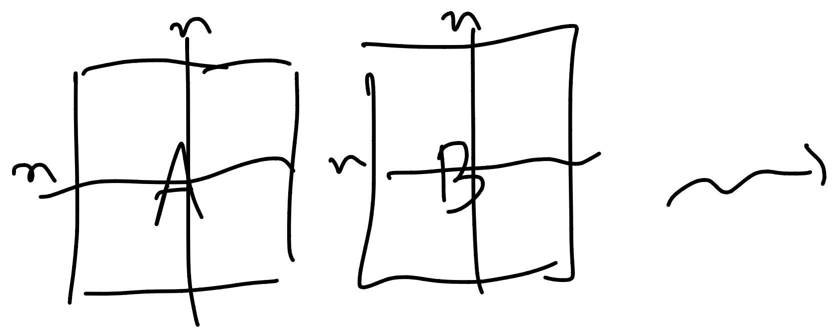
almost-median of A[].

$\frac{4n}{5}$

$$T(n) = cn + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$$ ⇝ ends up being $O(n)$.

More examples

$n^{2+\varepsilon}$??



Standard way $= n^3$.

$n^{\log 7} \approx n^{2.7\ldots}$

Book chapter of D, P, V..

Best known $\approx n^{2.36\ldots}$

- Matrix multiplication (Strassen's algorithm)
- Fast Fourier Transform
- Graph partitioning
- ….

Dynamic programming

Toy problem: subset sum

Problem: given *n* **non-negative** integers A[0], A[1], …, A[n-1] and a "target" *S*, find if there is a subset of the A[i] that add up to *S*

- Brute force?
- Divide and conquer?

Simple example of "sequential decision making"…

Divide and conquer

Recursive procedure

Recursive procedure

What are sub-problems?

What is running time?

Looking closer – example

Suppose A = [1, 2, 3, 5, 7, 9, 10, 11] and S = 20

Avoiding duplicate solves

- Can we “store answers”?
- How many sub-problems are there in total?

Modified procedure

Running time

Is this polynomial?