CS 170
Efficient Algorithms and Intractable Problems

Lecture 3:
Divide and Conquer II

Nika Haghtalab and John Wright

EECS, UC Berkeley

Announcements

Homework

- HW1 released, due Saturday!
- Get started early! Go to OH and HW Party (Friday) for help
- Use Edstem thread on study group search if you like

Discussion sections

- Start today! Check the discussion tab on webpage.
- You can attend any discussion section, no RSVP needed!

My OH, right after class on Tuesdays

- Meet at the podium or the door

Recap of last time

- Karatsuba's algorithm with $O(n^{1.6})$

→Using divide and conquer with fewer subproblems!

- Reviewed $O(\cdot)$ and $\Omega(\cdot)$ notation formally.

- Recurrence relations and the Master theorem!

Recap: Master Theorem

The Master Theorem

Suppose that $a \geq 1, b > 1,$ and $d \geq 0$ are constants (independent of n).

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

a: Number of sub-problems

b: Factor by which the problem size shrinks at each layer

n^d: Amount of computation per node, before/after subproblems are done.

Recap: Master Theorem

The Master Theorem

Suppose that $a \geq 1, b > 1,$ and $d \geq 0$ are constants (independent of n).

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$c \cdot n^d$

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log(n)) & \text{if } a = b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

This is tight!

a: Number of sub-problems

b: Factor by which the problem size shrinks at each layer

n^d: Amount of computation per node, before/after subproblems are done.

Recap: Master Theorem's Interpretation

a vs. b^d

Wide tree

$a > b^d$

Branching causes the number of problems to explode!

Most work is at the bottom of the tree!

$O\left(n^{\log_b(a)}\right)$

$a = b^d$

Branching perfectly balances total amount of work per layer.

All layers contribute equally.

$\boldsymbol{O}\left(n^d \log(n)\right)$

Tall and narrow

$a < b^d$

Problem size shrinks fast, so **most work is at the top of the tree!**

$O\left(n^d\right)$

This lecture

More on uses of Divide and conquer

- More examples of solving recurrence relations

- Matrix Multiplication

- Median Selection (And next time)

Solving Recursion Example 1

Bound $T(n) = T(n-1) + \sqrt{n}$ in terms of $\Theta(.)$ notation

Solving Recursion Example 2

Bound $T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$ in terms of O(.) notation.

The Master Theorem

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

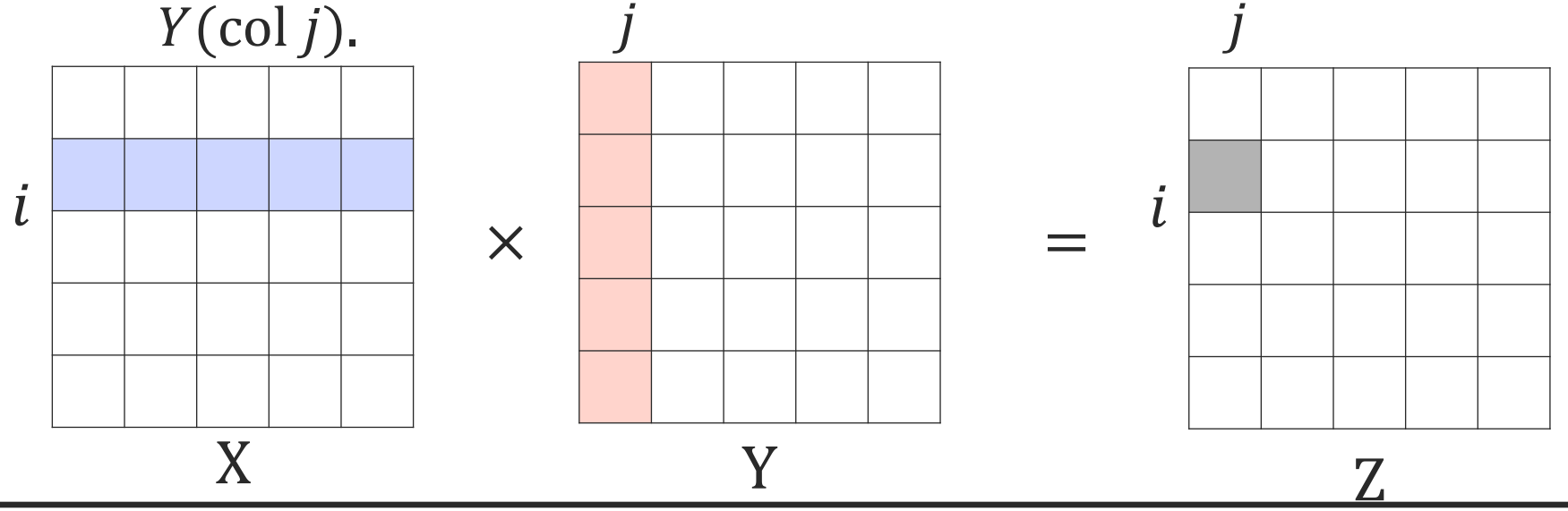
Matrix Multiplication

Matrix Operations

We showed that integer multiplication can be done faster than the grade school algorithm.

→Why stop there? Can we multiply Matrices faster than we did in high school?

Product of two $n \times n$ matrices X and Y, is a $n \times n$ matrix Z: Entry $z_{i,j}$ is dot-product of X(row i) and Y(col j).



Matrix Multiplication in Everyday Life!

Large language models rely on multiplying large (in dimensionality) matrices!

To generate text, we need to understand the relationship between these words.

This is done through a mechanism called "attention" that learns how each word pays attention to other words. There are lots of attentions and layers in each model.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

This slide is just for general knowledge. No need to learn about "attention" in this lecture. But if you are interested, take CS189!

Matrix Multiplication Everyday Life!

At the heart of attention is a large matrix multiplications.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

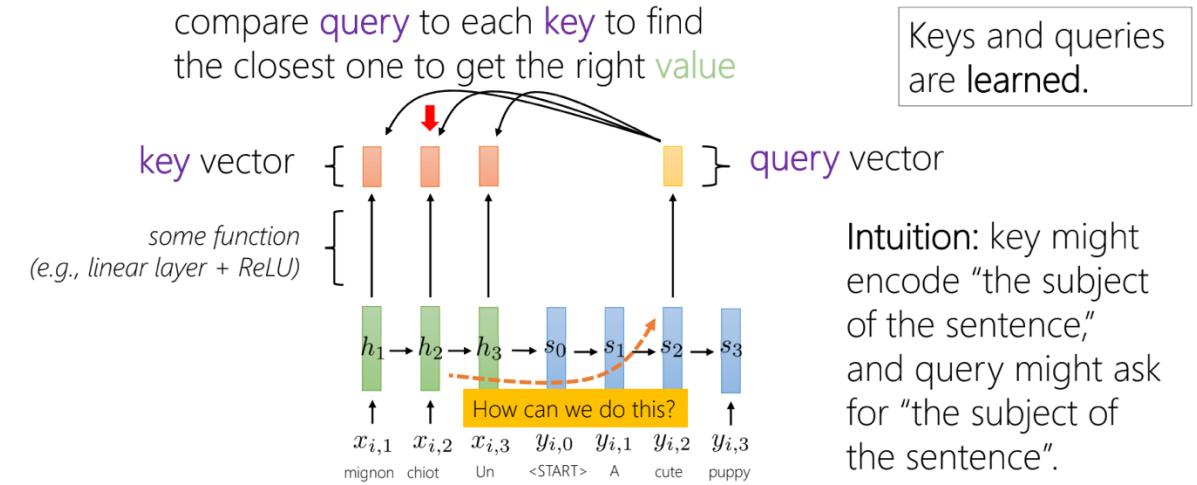
Older models: Q, K, and V are about $2^{13} \times 2^{13}$

Newer models: Not disclosed, likely higher.

Every few sentences you generate likely needs 100s-1000s of these large matrix multiplications!

This slide is just for general knowledge. No need to learn about "attention" in this lecture. But if you are interested, take CS189!

CS189, Lecture 11



Matrix Operations

- For integer multiplication, "problem size" was the number of digits
- For matrix multiplication, it is the dimensionality.
  →We assume each cell has small number of bits, say 32-64.
  →So, we can multiply/add two elements of the matrices in $O(1)$.
  →But the sizes of matrices used in practice can be high.

Discuss

Dot-product

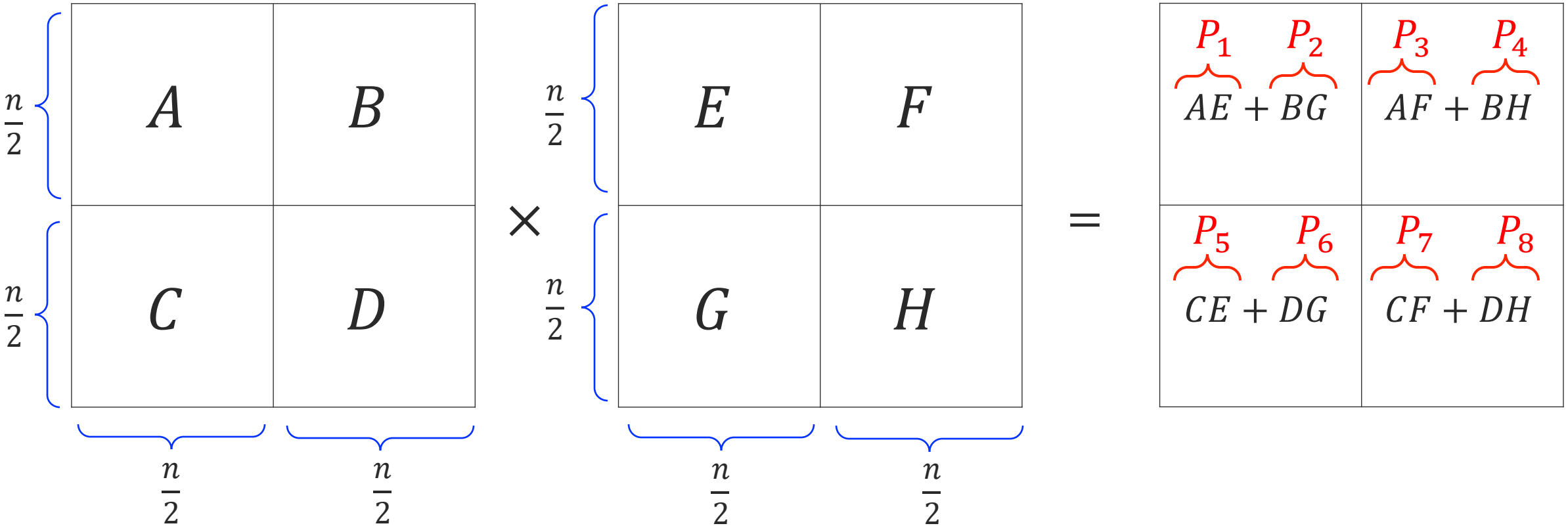
- What is the runtime of computing the dot-product of two vectors of size n?

Matrix Multiplication

- What is the runtime of the high-school $n \times n$ matrix multiplication algorithm?

Breaking Matrix Multiplication to Subproblems

Let's try the same trick we used in integer multiplication: Break the matrix to matrices of size $\frac{n}{2} \times \frac{n}{2}$.



Each subproblem P_i is a matrix multiplication of two $\frac{n}{2} \times \frac{n}{2}$ matrices

Recurrence Relationship

- At each layer, we have 8 problems

→ Each problem of size $\frac{n}{2}$.

The Master Theorem

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

We have to do a bunch of other operations

- Finding A, B, …, H by shifting n-digit arrays.
- Adding $\frac{n}{2} \times \frac{n}{2}$ matrices.
- Appending matrices to make one $n \times n$ matrix

} $O(?)$

$\underbrace{P_5}_{CE} + \underbrace{P_6}_{DG}$	$\underbrace{P_7}_{CF} + \underbrace{P_8}_{DH}$

Recurrence $T(n) = ?$

Runtime $T(n) = ?$

Strassen's Algorithm

No need to memorize this!

Like Karatsuba's algorithm, but this time for matrices.

Express the answer with fewer than 8 subproblems of size $\frac{n}{2} \times \frac{n}{2}$.

→ Subtlety: Matrix multiplication is not "commutative" → order matters!

Strassen's trick:

$$Q_1 = A(F - H)$$
$$Q_2 = (A + B)H$$
$$Q_3 = (C + D)E$$
$$Q_4 = D(G - E)$$
$$Q_5 = (A + D)(E + H)$$
$$Q_6 = (B - D)(G + H)$$
$$Q_7 = (A - C)(E + F)$$

$$X \times Y = \begin{array}{|c|c|} \hline Q_5 + Q_4 - Q_2 + Q_6 & Q_1 + Q_2 \\ \hline Q_3 + Q_4 & Q_1 + Q_5 - Q_3 - Q_7 \\ \hline \end{array}$$

Recurrence Relationship

- At each layer, we have 7 problems

→ Each problem of size $\frac{n}{2}$.

All other extra operations, additions, subtractions, …

- Finding A, B, …, H by shifting n-digit arrays
- Additions $(F - H)$, $(A + B)$, …
- Appending matrices back together.
- All together at most $O(n^2)$

Runtime $T(n) =$

$Q_5 + Q_4 - Q_2 + Q_6$	$Q_1 + Q_2$
$Q_3 + Q_4$	$Q_1 + Q_5 - Q_3 - Q_7$

$$Q_1 = A(F - H)$$
$$Q_2 = (A + B)H$$
$$Q_3 = (C + D)E$$
$$Q_4 = D(G - E)$$
$$Q_5 = (A + D)(E + H)$$
$$Q_6 = (B - D)(G + H)$$
$$Q_7 = (A - C)(E + F)$$

Recurrence Relationship

- At each layer, we have 7 problems

→ Each problem of size $\frac{n}{2}$.

All other extra operations, additions, subtract

- Finding A, B, …, H by shifting n-digit arrays
- Additions $(F - H)$, $(A + B)$, …
- Appending matrices back together.
- All together at most $O(n^2)$

Runtime $T(n) = 7\,T\left(\frac{n}{2}\right) + O(n^2)$

Using the master theorem $T(n) = ?$

The Master Theorem

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

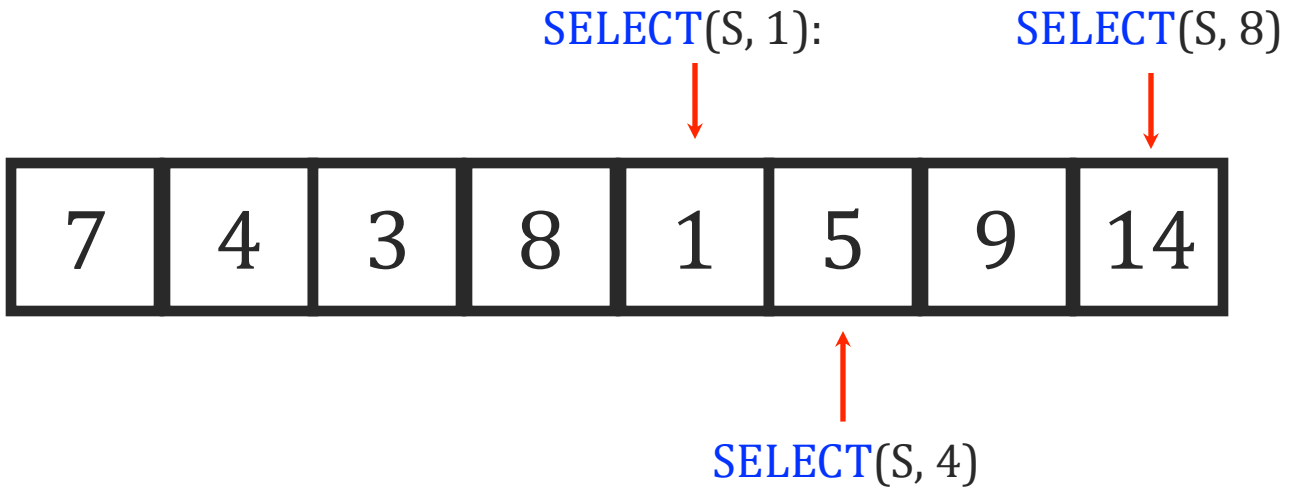
$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

$$Q_1 = A(F - H)$$
$$Q_2 = (A + B)H$$
$$Q_3 = (C + D)E$$
$$Q_4 = D(G - E)$$
$$Q_5 = (A + D)(E + H)$$
$$Q_6 = (B - D)(G + H)$$
$$Q_7 = (A - C)(E + F)$$

(Median) Selection

The k-select Problem

Given an array S of n numbers and $k \in \{1, 2, \ldots, n\}$, find the kth smallest element of it.



Some special cases:

SELECT(S, 1): Minimum element of the array

SELECT(S, n): Maximum element of the array

SELECT(S, $\left\lceil \frac{n}{2} \right\rceil$): Median element of the array

Simple Algorithms for k-Select

An $O(n \log(n))$ algorithm

→ Sort the array, using merge-sort (or another $O(n \log(n))$ sort).

→ Then go through the array and return the k-th element.

Technicality: Arrays are 0-index, so you should return $S[k-1]$ after sorting!

Remainder of the lecture

Can we do better than $O(n \log(n))$?

Can we do $O(n)$?

Simple Algorithms for k-Select

Can you think of $O(n)$ algorithm for SELECT(S, 1)?

- FOR loop through the array. **Store the minimum so far**: If the current element is less than the stored value, store the current value as min instead.

Can you think of $O(n)$ algorithm for SELECT(S, 2)?

- Run SELECT(S, 1) and let $S \leftarrow S \backslash$SELECT(S, 1). (remove that element) $O(n)$
- Return SELECT(S, 1) $O(n)$
- Total of $O(n)$ runtime.

Does this trick produce an $O(n)$ algorithm for SELECT(S, $n/2$)?

- No. We would be running $\frac{n}{2}$ SELECTs each $O(n)$.

Technically: Array S is shrinking, so SELECT(S, 1) is getting faster, but not that much faster $\text{len}(S) > \frac{n}{2}$.

Big Question

Can we perform Median selection
(or any other k-select generally)
in $O(n)$?

Idea: Divide and Conquer

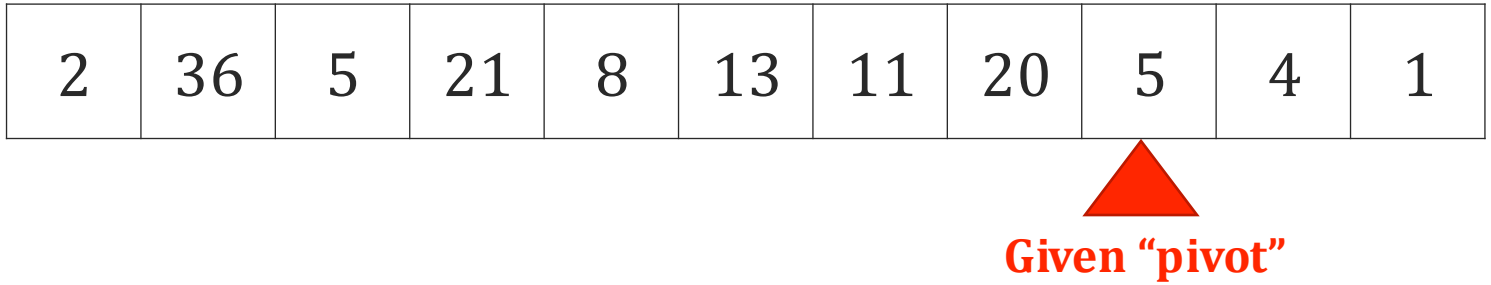
We want to divide the problem to subproblems. How?

- Imagine we are given a **"pivot" v.** Split the array into three pieces

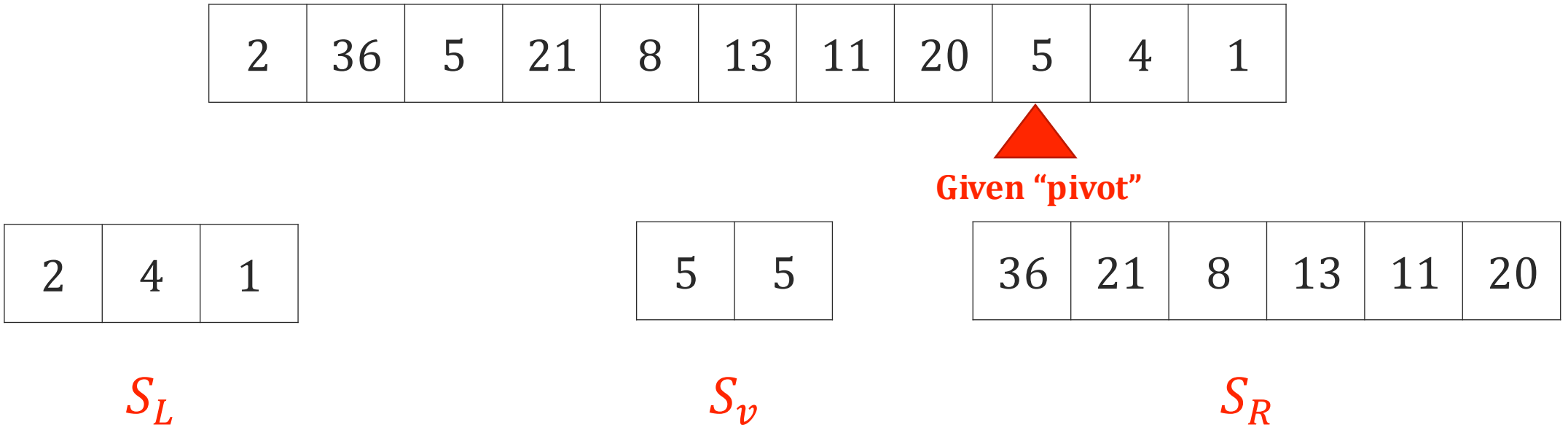
→ S_L: Elements less than the pivot

→ S_v: Elements equal to the pivot

→ S_R: Elements larger than the pivot



The subproblems



We want to compute SELECT(S, k):

- If $k \leq len(S_L)$: We should return SELECT(S_L, k)
- If $len(S_L) < k \leq len(S_L) + len(S_v)$: We should return v.
- If $len(S_L) + len(S_v) < k$: We should return SELECT(S_R, $k - len(S_L) - len(S_v)$)

The Recurrence Relation

We want to compute SELECT(S, k):

- If $k \leq len(S_L)$: We should return SELECT(S_L, k)
- If $len(S_L) < k \leq len(S_L) + len(S_v)$: We should return v.
- If $len(S_L) + len(S_v) < k$: We should return SELECT$(S_R, k - len(S_L) - len(S_v))$

$$T(n) = \begin{cases} T\big(len(S_L)\big) + O(n) & \text{if } k \leq len(S_L) \\ T\big(len(S_R)\big) + O(n) & \text{if } len(S_L) + len(S_v) < k \\ O(n) & \text{if } len(S_L) < k \leq len(S_L) + len(S_v) \end{cases}$$

The lengths of S_L and S_R depend on the choice of the pivot.

What are good/bad choices of pivot

Discuss

Order the following pivots from worst pivot (slowest runtime) to the best pivot (fastest). For intuition, imagine **no element is repeated**.

1. smallest element (min)
2. $n/4\ th$ smallest element
3. $n/2\ th$ smallest element (median)
4. $3n/4\ th$ smallest element
5. $(n-1)th$ smallest element

Intuitively, we want a pivot such that $\max(len(S_L), len(S_R))$ is small.

Thought Exercise: Runtime for Bad Pivot

Let's pretend that we are unlucky, and we always get a pivot that is the smallest element of the array. What is the runtime of SELECT(S, k)?

Thought Exercise: Runtime for the Ideal Pivot

Let's pretend that the pivot we picked is indeed the median!

What is the runtime of SELECT(S, k)?

Uhhh! Wasn't the whole point that we don't know how to find the median in $O(n)$?

Yes! This is just a thought exercise to know the ideal situation.

Thought Exercise: Runtime for the Ideal Pivot

Let's pretend that the pivot we picked is indeed the median!

Then $len(S_L) \leq n/2$ and $len(S_R) \leq n/2$.

$$T(n) \leq T\left(\frac{n}{2}\right) + O(n)$$

What's the runtime?
$a = 1, b = 2, d = 1$, so $a < b^d$
$O(n)$ runtime.

Uhhh! Wasn't the whole point that we don't know how to find the median in $O(n)$?

Yes! This is just a thought exercise to know the ideal situation.

The Master Theorem

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Thought Exercise: "Good" Enough Pivot

Let's pretend that the pivot we picked is always between the $\frac{n}{4}th$ smallest and $\frac{3n}{4}$ th smallest element! What is the runtime of SELECT(S, k)?

Discuss

Assuming that the pivot is between the $\frac{n}{4}th$ smallest and $\frac{3n}{4}$ th smallest element, what's the maximum $\text{len}(S_L)$ and $\text{len}(S_R)$?

Write down the recurrence relationship in this case:

Thought Exercise: "Good" Enough Pivot

Let's pretend that the pivot we picked is always between the $\frac{n}{4}th$ smallest and $\frac{3n}{4}$ th smallest element! What is the runtime of SELECT(S, k)?

So the runtime can be expressed by $T(n) \leq T\left(\frac{3n}{4}\right) + O(n)$

What's the runtime?

- $a = 1, b = 4/3, d = 1, a < b^d$
- $O(n)$ runtime.

The Master Theorem

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

One last thought exercise!

Let's pretend that the pivot we picked is always between the $\frac{n}{10}\,th$ smallest and $\frac{9n}{10}$ th smallest element! What is the runtime of SELECT(S, k)?

Then, $len(S_R) \leq \frac{9n}{10}$ and $len(S_L) \leq \frac{9n}{10}$. So, the recurrence is $T(n) \leq T\left(\frac{9n}{10}\right) + O(n)$

In this case as well,

- $a = 1, b = 10/9, d = 1, a < b^d$
- $O(n)$ runtime!

The Master Theorem

Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Summarizing Our Thoughts

Any pivot that's kind of in the middle is a good enough pivot

→ Call Pivots between the $\frac{n}{4}th$ smallest and $\frac{3n}{4}$ th smallest elements "good" pivots

→ There are more than half of the elements in the array are "good" pivots

→If we pick a random element, we have 50% chance of getting a "good" pivot

We don't need a "good" pivot at every round.

→We just need to get "good" pivots often enough

→ Every time we have a "good" pivot, the problem size shrinks to $\frac{3}{4}$ of the last one.

If we pick a random element as pivot, the expected runtime is fast!

→ We will formalize this next time

Wrap up

Matrix Multiplication:

- Strassen's algorithm
- Similar to Karatsuba, we reduce the number of subproblems from 8 to 7.

k-Select

- We saw that a good pivot selection gives us O(n)
- We discussed that a random selection, will likely give a good pivot

Next time

- Continue with Median selection
- Formalize the discussion of pivots and prove O(n) runtime