CS 170
Efficient Algorithms and Intractable Problems

Lecture 3:
Divide and Conquer II

Nika Haghtalab and John Wright

EECS, UC Berkeley

Announcements

Discussion sections (yesterday and Tuesday)**.**

- Feeling you need a slower-paced section? Go to LOST section on Fridays.
- Starting next week: Tuesday 3-4pm and Thursday 10-11am discussion

Homework party:

- Tomorrow (Friday) and ~~Monday~~ (labor day!). **HW1 due on Tuesday.**

Short break:

- Seemed to work. Let's give it a second try today.
- Remember: at break time, please help close the lecture hall doors.

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

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

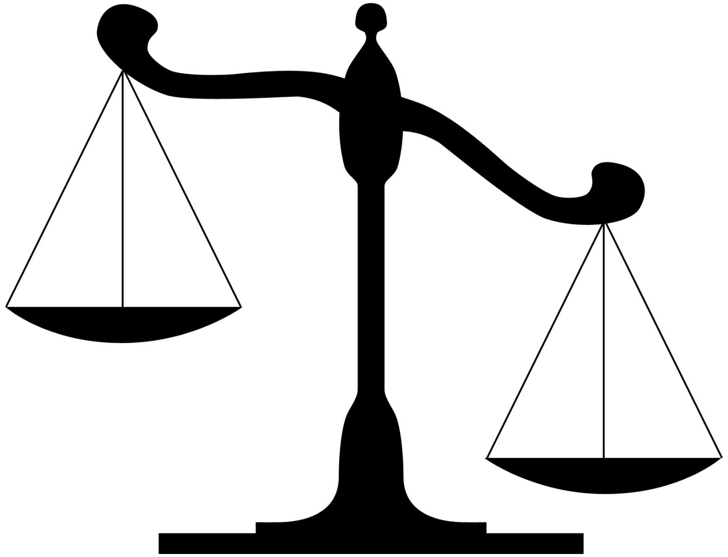
a: Number of sub-problems

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

Tall and narrow
$a < b^d$

Problem size shrinks fast, so **most work is at the top of the tree!**
$O\left(n^d\right)$

$a = b^d$
Branching perfectly balances total amount of work per layer.
All layers contribute equally.
$\boldsymbol{O}\left(n^d \log(n)\right)$

This lecture

Two awesome uses of Divide and conquer

- Matrix Multiplication

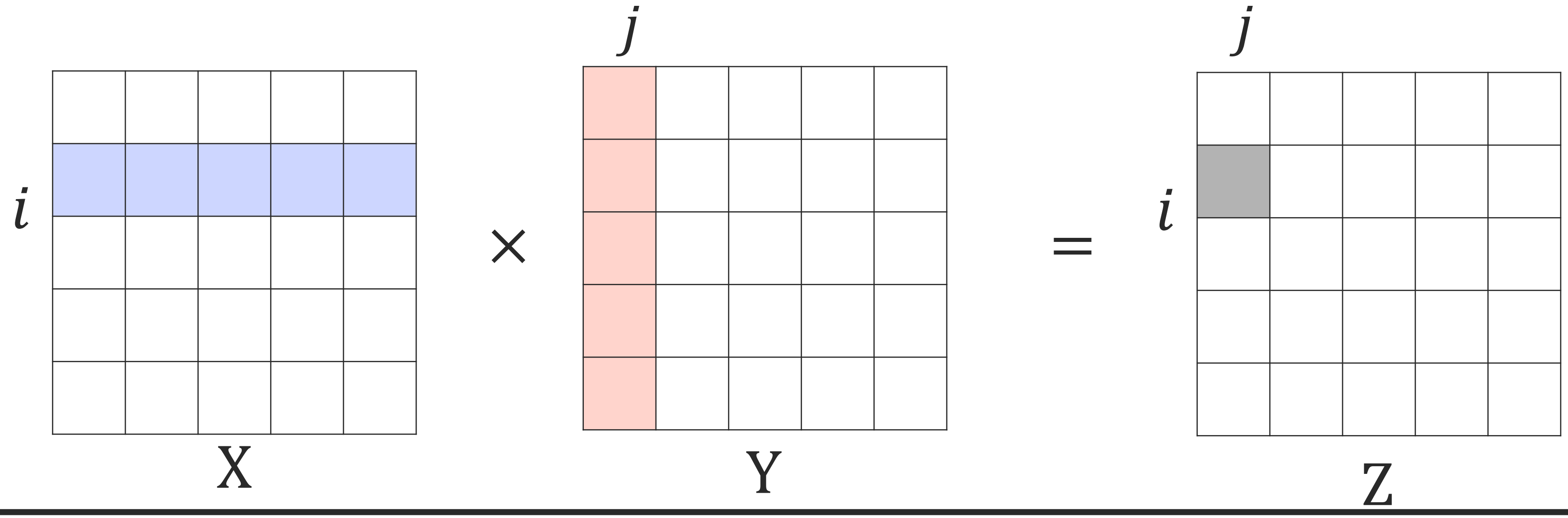
- Median Selection

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?

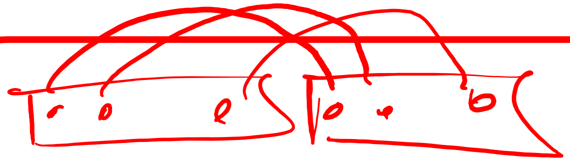
Product of two $n{\times}n$ matrices X and Y, is a $n{\times}n$ matrix Z: Entry $z_{i,j}$ is dot-product of X(row i) and Y(col j).



Matrix Operations

- For integer multiplication, “problem size” was the number of digits
- For matrix multiplication, it is the dimensionality.
  - →But we assume the integers have small number of bits, say 32-64.
  - →So, we can multiply/add two elements of the matrices in $O(1)$.
  - →Huge matrices in practice?

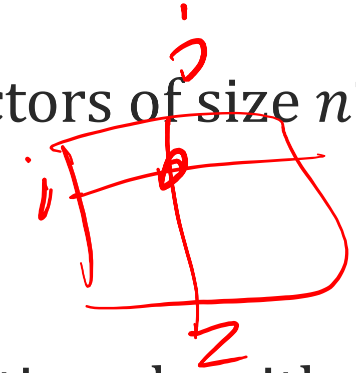
Discuss



Dot-product

$O(n)$

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



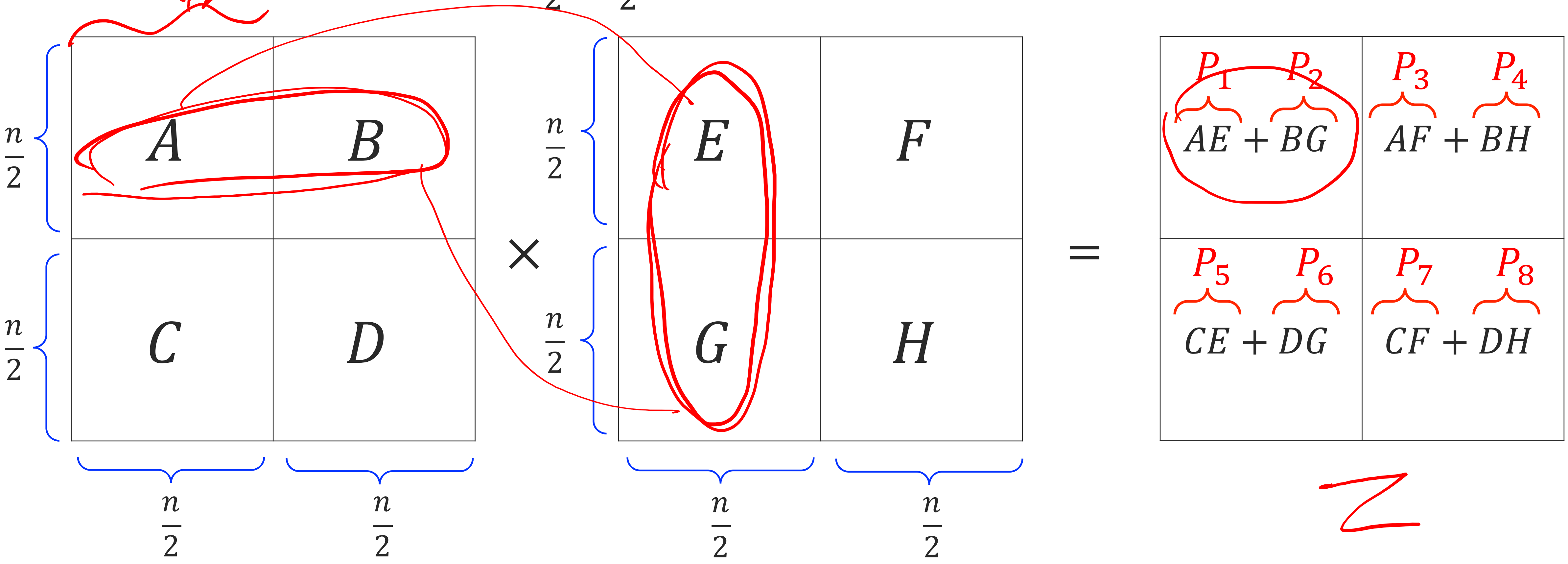
$n^2 \times O(n) = O(n^3)$

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 $\frac{n}{2} \times \frac{n}{2}$. AE+BG



$$\begin{array}{c|cc} \frac{n}{2} & A & B \\ \frac{n}{2} & C & D \\ \hline & \frac{n}{2} & \frac{n}{2} \end{array} \times \begin{array}{c|cc} \frac{n}{2} & E & F \\ \frac{n}{2} & G & H \\ \hline & \frac{n}{2} & \frac{n}{2} \end{array} = \begin{array}{|c|c|} \hline \overbrace{AE}^{P_1} + \overbrace{BG}^{P_2} & \overbrace{AF}^{P_3} + \overbrace{BH}^{P_4} \\ \hline \overbrace{CE}^{P_5} + \overbrace{DG}^{P_6} & \overbrace{CF}^{P_7} + \overbrace{DH}^{P_8} \\ \hline \end{array}$$

Z

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

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

$O(n^2)$

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

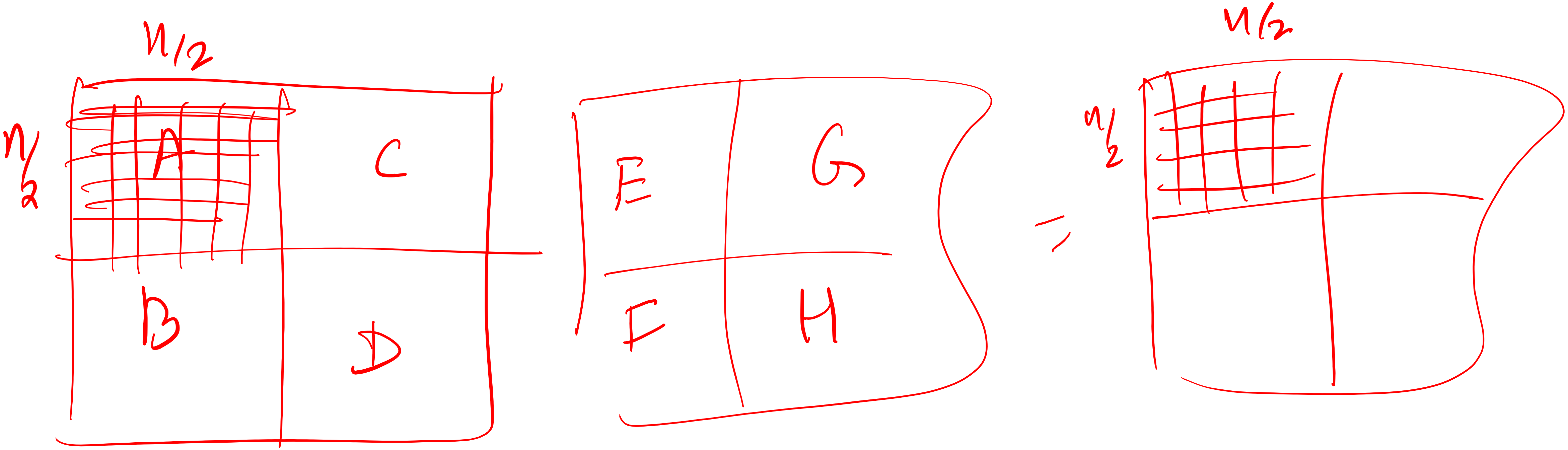
$\Big\}\ O(n^2)$

$\overbrace{CE}^{P_5} + \overbrace{DG}^{P_6}$	$\overbrace{CF}^{P_7} + \overbrace{DH}^{P_8}$

$8 > b^d = 4$

Recurrence $T(n) = ?$ $\quad \overset{a}{8} \times T\left(\frac{n}{\underset{b}{2}}\right) + O\left(n^{\overset{d}{2}}\right)$

Runtime $T(n) = ?$ $\quad T(n) = O\left(n^{\log_2(8)}\right) = O(n^3)$


n/2
n/2
A
C
B
D
E
G
F
H
=
n/2
n/2

Strassen's Algorithm

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

No need to memorize this!

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!

$$\underbrace{AF - AH}_{2\ \text{multiplications}} = \underbrace{A\times(F-H)}_{1\ \text{multiplication}}$$

Strassen's trick:

$$\left.\begin{aligned} Q_1 &= A(F - \cancel{H}) \\ Q_2 &= (A+B)\cancel{H} \end{aligned}\right\} AF + BH$$

$$Q_3 = (C+D)E$$

$$Q_4 = D(G-E)$$

$$Q_5 = (A+D)(E+H)$$

$$Q_6 = (B-D)(G+H)$$

$$Q_7 = (A-C)(E+F)$$

$$X\times Y = \begin{bmatrix} Q_5+Q_4-Q_2+Q_6 & \underline{Q_1+Q_2} \\ Q_3+Q_4 & Q_1+Q_5-Q_3-Q_7 \end{bmatrix}$$

Recurrence Relationship

- At each layer, we have 7 problems

→ Each problem of size $\frac{n}{2}$.

$\overbrace{AE}^{P_1} + \overbrace{BG}^{P_2}$	$\overbrace{AF}^{P_3} + \overbrace{BH}^{P_4}$
$\overbrace{CE}^{P_5} + \overbrace{DG}^{P_6}$	$\overbrace{CF}^{P_7} + \overbrace{DH}^{P_8}$

All other extra operations, additions, subtractions, …

- At most $O(n^2)$

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

Recurrence Relationship

- At each layer, we have 7 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

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

All other extra operations, additions, subtract

- At most $O(n^2)$

$\overbrace{CE}^{P_5} + \overbrace{DG}^{P_6}$	$\overbrace{CF}^{P_7} + \overbrace{DH}^{P_8}$

$7 > 2^2$

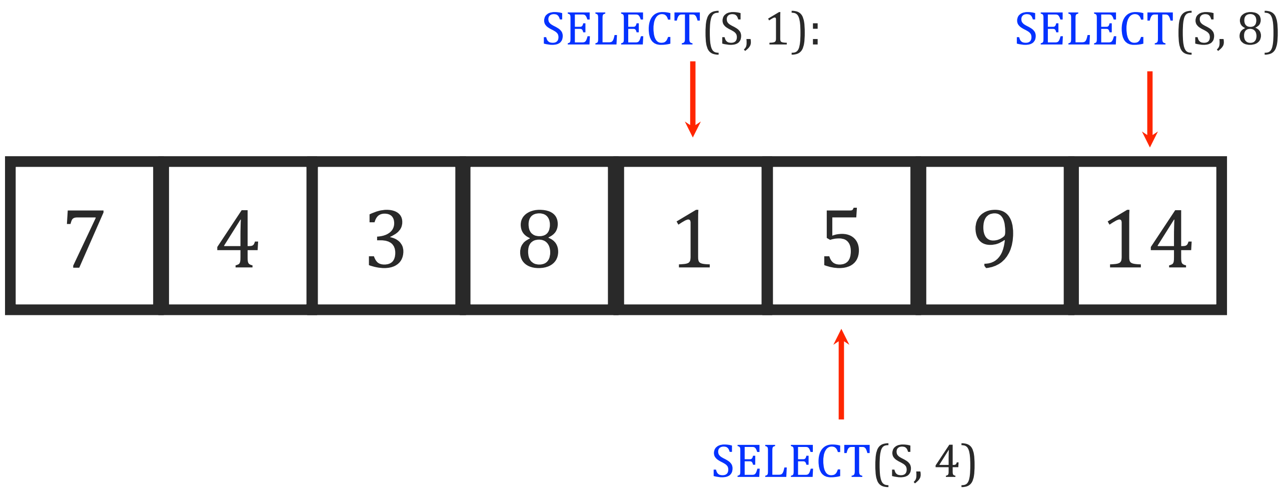
Runtime $T(n) = 7\,T\left(\frac{n}{2}\right) + O(n^2)$

Using the master theorem $T(n) = ?$ $O\left(n^{\log_2(7)}\right) \approx O\left(n^{2.8}\right)$

(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.



SELECT(S, 1):

SELECT(S, 8)

7	4	3	8	1	5	9	14

SELECT(S, 4)

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). (<u>remove that element</u>) $O(n)$
- 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 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?

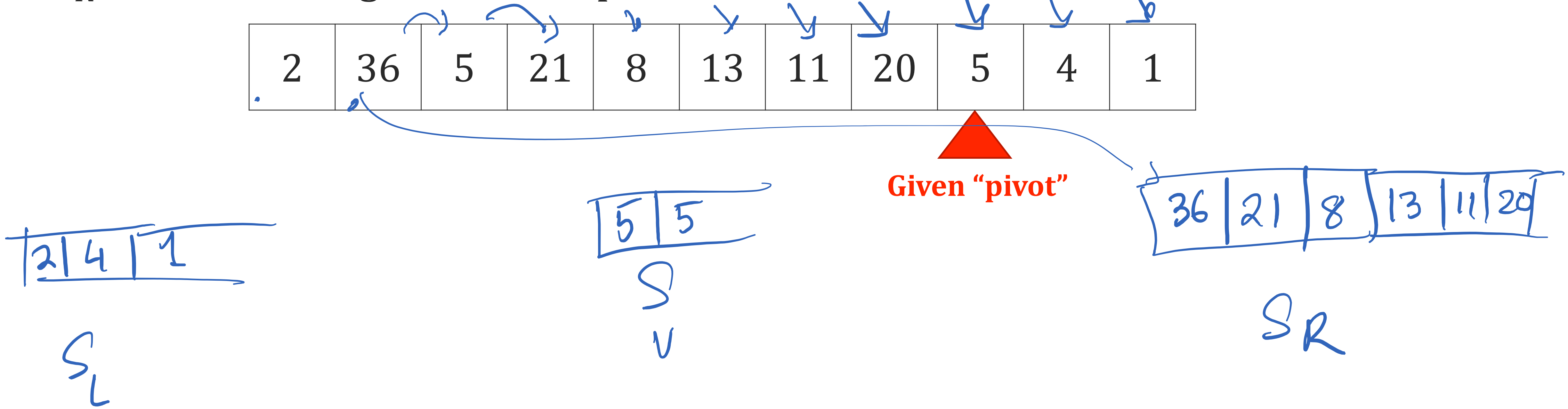
- Imagine we are given a **"pivot" v.** Split the array into three pieces

→ S_L: Elements less than the pivot

→ S_v: Elements equal to the pivot

→ S_R: Elements larger than the pivot

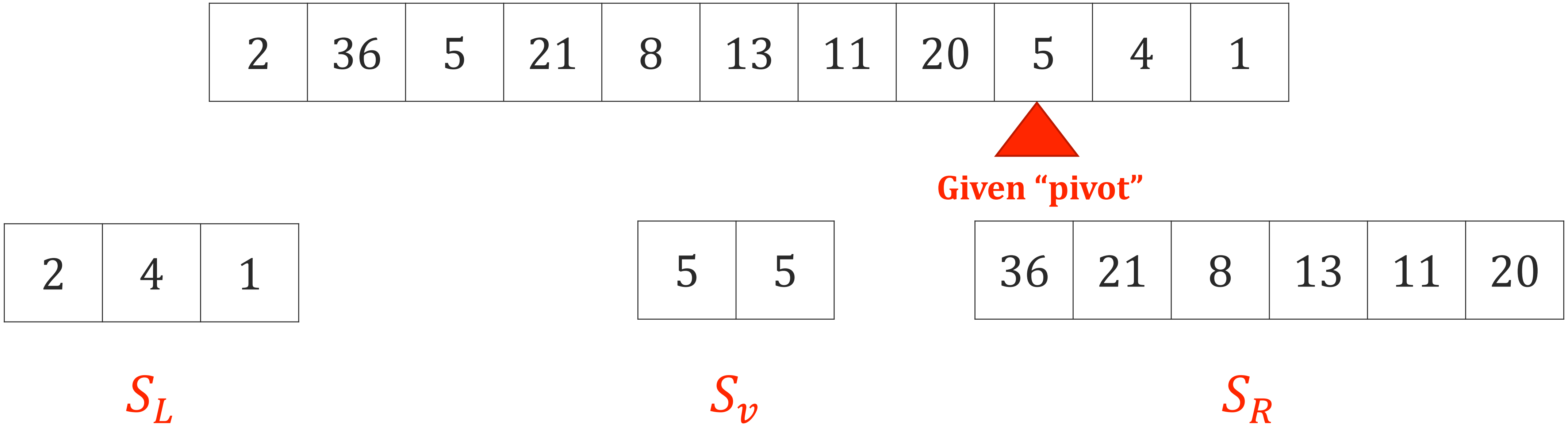
→ $O(n)$



2	36	5	21	8	13	11	20	5	4	1

Given "pivot"

The subproblems



2	36	5	21	8	13	11	20	5	4	1

Given “pivot”

2	4	1

S_L

5	5

S_v

36	21	8	13	11	20

S_R

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

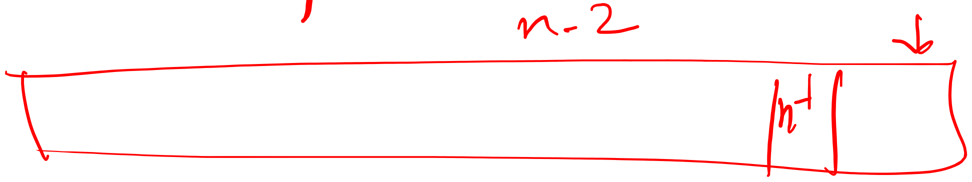
Intuitively, we want a pivot such that $\max(len(S_L), len(S_R))$ is small.

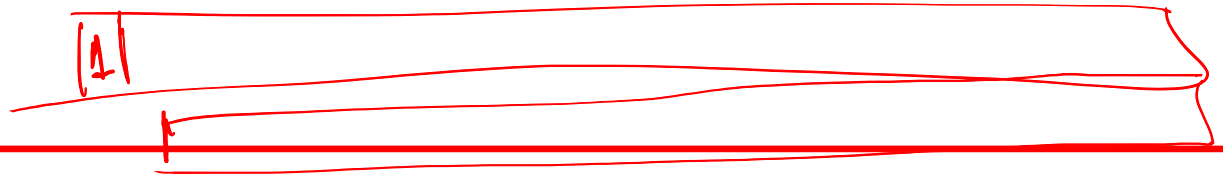
Discuss

Order the following pivots from worst pivot to the best pivot. 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

$1 \rightarrow 5 \rightarrow \begin{matrix}2\\4\end{matrix} \rightarrow 3$





Runtime, given the ideal pivot

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

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.

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.

The Master Theorem

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

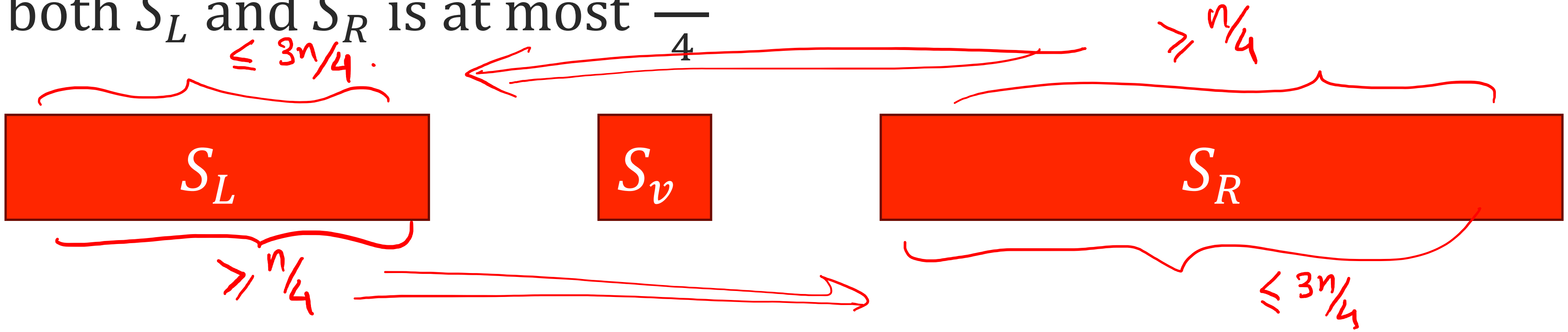
$$T(n) = \begin{cases} O\left(n^d\right) & \text{if } a < b^d \\ O\left(n^d \log(n)\right) & \text{if } a = b^d \\ O\left(n^{\log_b(a)}\right) & \text{if } a > b^d \end{cases}$$

“Good” pivots

I still don’t know how to find such a pivot, but ok for now ....

Any pivot between the $\frac{n}{4}$ *th* smallest and $\frac{3n}{4}$ th smallest element is good enough!

Length of both S_L and S_R is at most $\frac{3n}{4}$



What’s the runtime if pivot is between the $\frac{n}{4}$ th and $\frac{3n}{4}$ th smallest element?

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

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

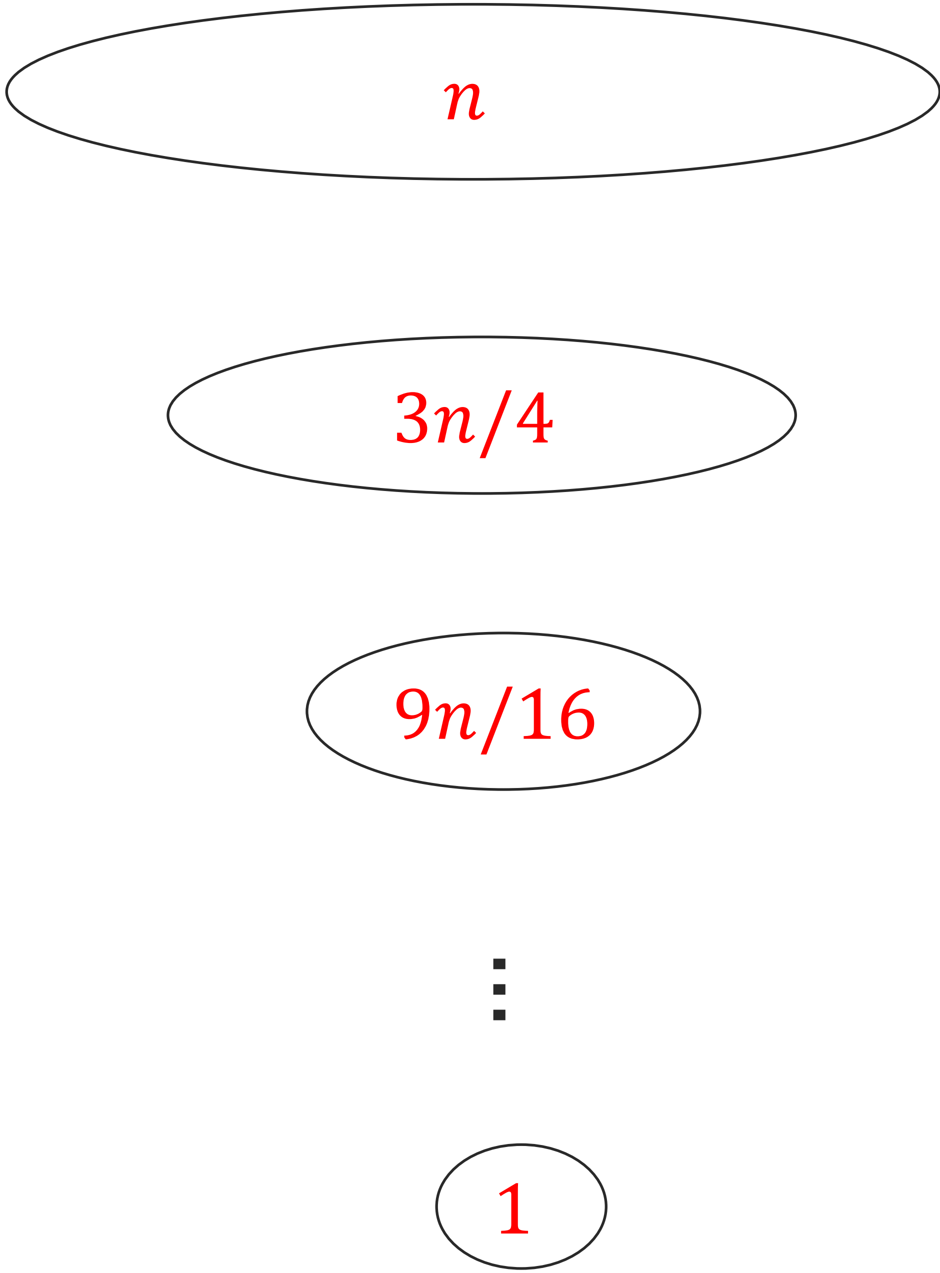
$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log(n)) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Trees Revisited

Imagine: At every round we got a “good” pivot.
So we multiply the size by $\leq 3/4$.

Single node at layer i of size $n\left(\frac{3}{4}\right)^{i}$. Total contribution at layer i is $\leq c \cdot n\left(\frac{3}{4}\right)^{i}$.

What is the total amount of work in all layers?



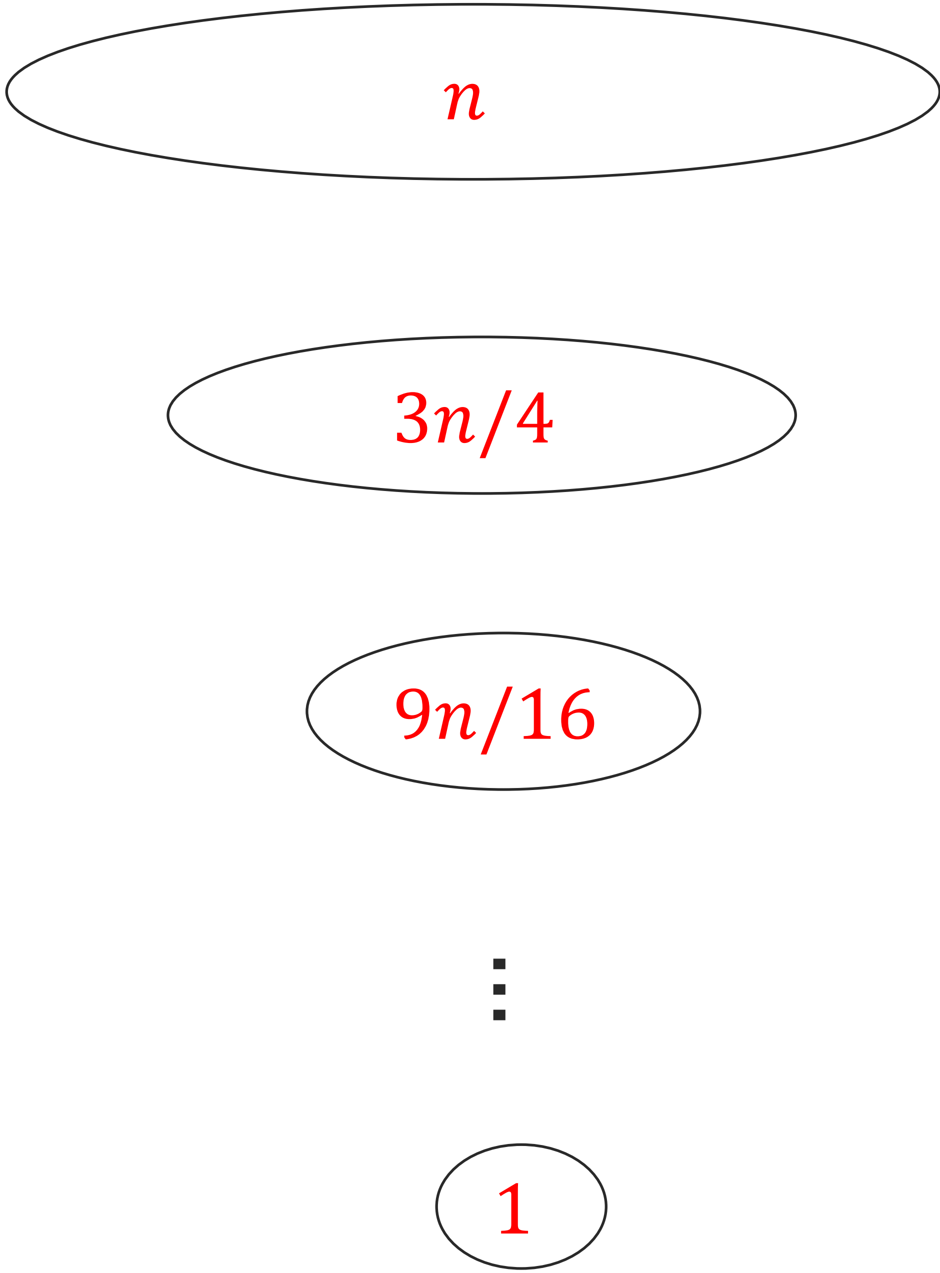
Trees Revisited

Imagine: At every round we got a “good” pivot. So we multiply the size by $\leq 3/4$.

Single node at layer i of size $n\left(\frac{3}{4}\right)^i$. Total contribution at layer i is $\leq c \cdot n\left(\frac{3}{4}\right)^i$.

What is the total amount of work in all layers?

$$T(n) \leq \sum_{i=0}^{\log_{4/3}(n)} c\, n\left(\frac{3}{4}\right)^i \in O(n)$$



How do we pick a "good" pivot?

Two ideas:

1. Pick it uniformly at random from array S.

- We get a "good" pivot in the $n/4$-$3n/4$ range with probability $1/2$.
- Show that the algorithm runs in $O(n)$ in expectation.

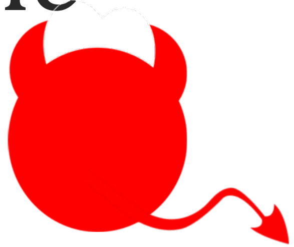
We will do this one.

2. Find a good enough pivot deterministically.

- It always runs in $O(n)$.
- Much harder analysis and in practice it is slower that the random pivot.

We will post readings for this.

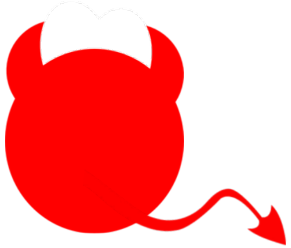
Randomized Algorithms and Expected Runtime



We typically think about runtime of an Alg on the **worst possible** problem instance.

Randomized Algorithms:

1. Write down the algorithm description.
2. Adversary sees the description and picks a bad instance.



3. Run the algorithm and throw the dice.

The adversary (choice of bad problem instance) doesn’t depend on the randomness.

The running time is a **random variable.**

- It makes sense to talk about **expected running time**.

Expected Running Time and Divide and Conquer

We are interested in **expected runtime.**

$$\mathbb{E}[T(n)]$$

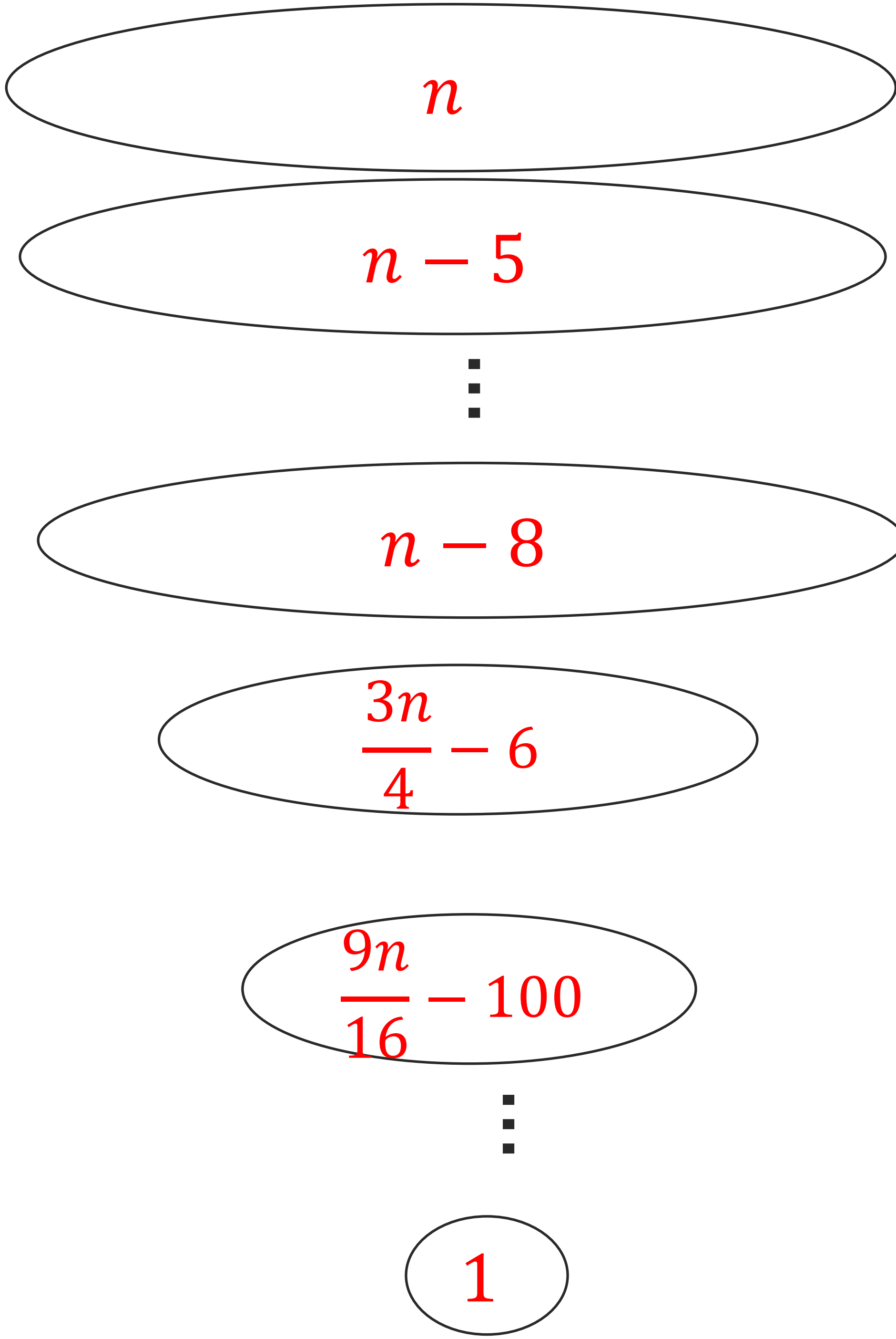
averages over runtimes $T(i)$ based on the probability of getting a subproblem of size i.

$\mathbb{E}[T(n)]$ is small when large size i has very low probability of happening

Trees Revisited

If every time we got a “good” pivot, we multiply the size by $\leq 3/4$.

In reality, in some rounds we are using bad pivots and in some rounds we are using good pivots.



n

$n-5$

$\vdots$

$n-8$

$\frac{3n}{4}-6$

$\frac{9n}{16}-100$

$\vdots$

1

Trees Revisited

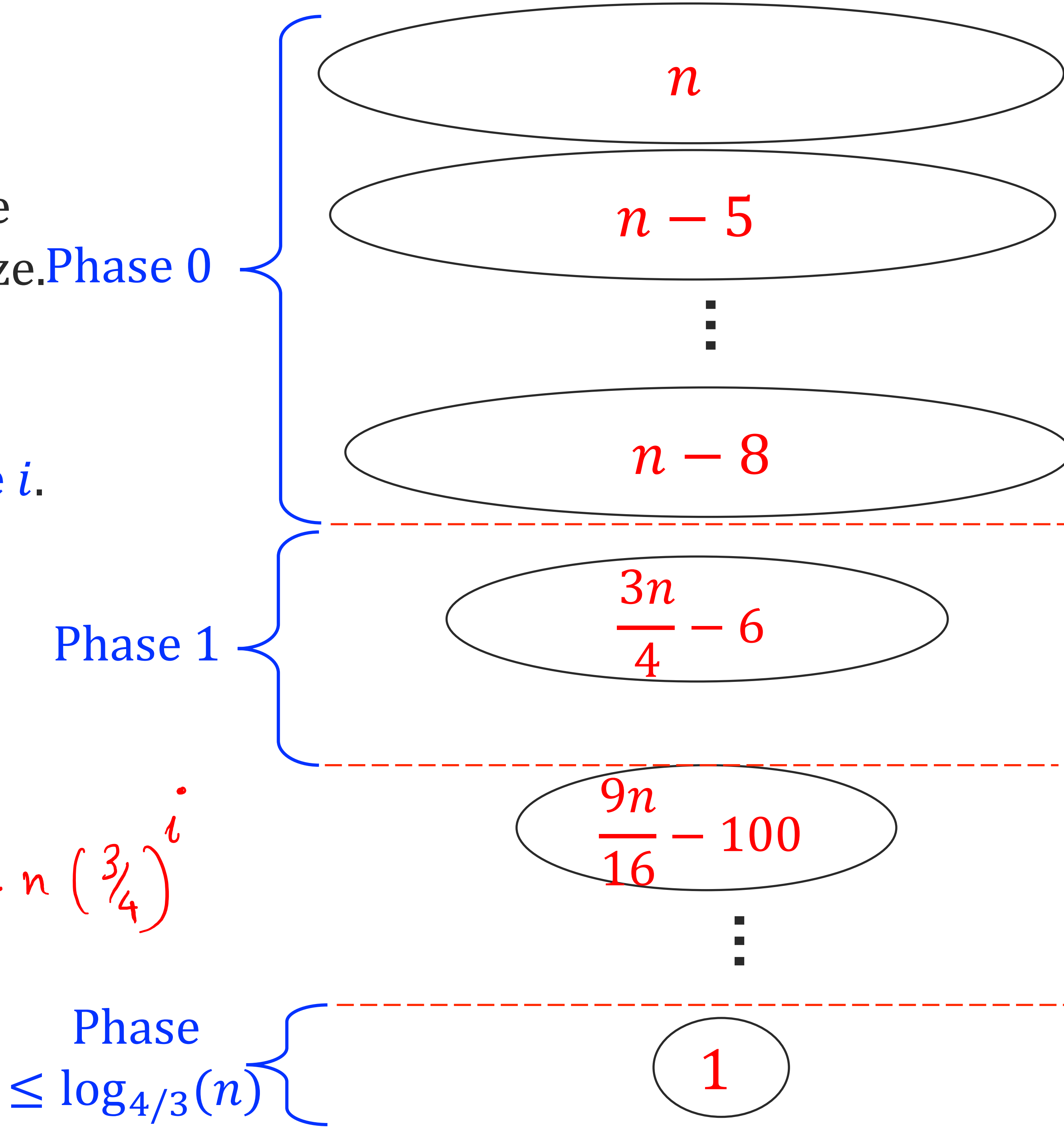
Partition layers to “phases”, when the size drops to ¾ or less of the original array size.

- In phase i, problem size $\leq \left(\frac{3}{4}\right)^i n$.
- X_i : random variable for length of phase i. Equiv, # tries until we choose a good pivot.

$\leq X_i \cdot c \left(3/4\right)^i n$

What is the contribution of phase i?

Total runtime: $\mathbb{E}[T(n)] \leq \sum_{i=0}^{\log_{4/3}(n)} \mathbb{E}[X_i]\, c \cdot n \left(3/4\right)^i$



Trees Revisited

Partition layers to “phases”, when the size drops to ¾ or less of the original array size.

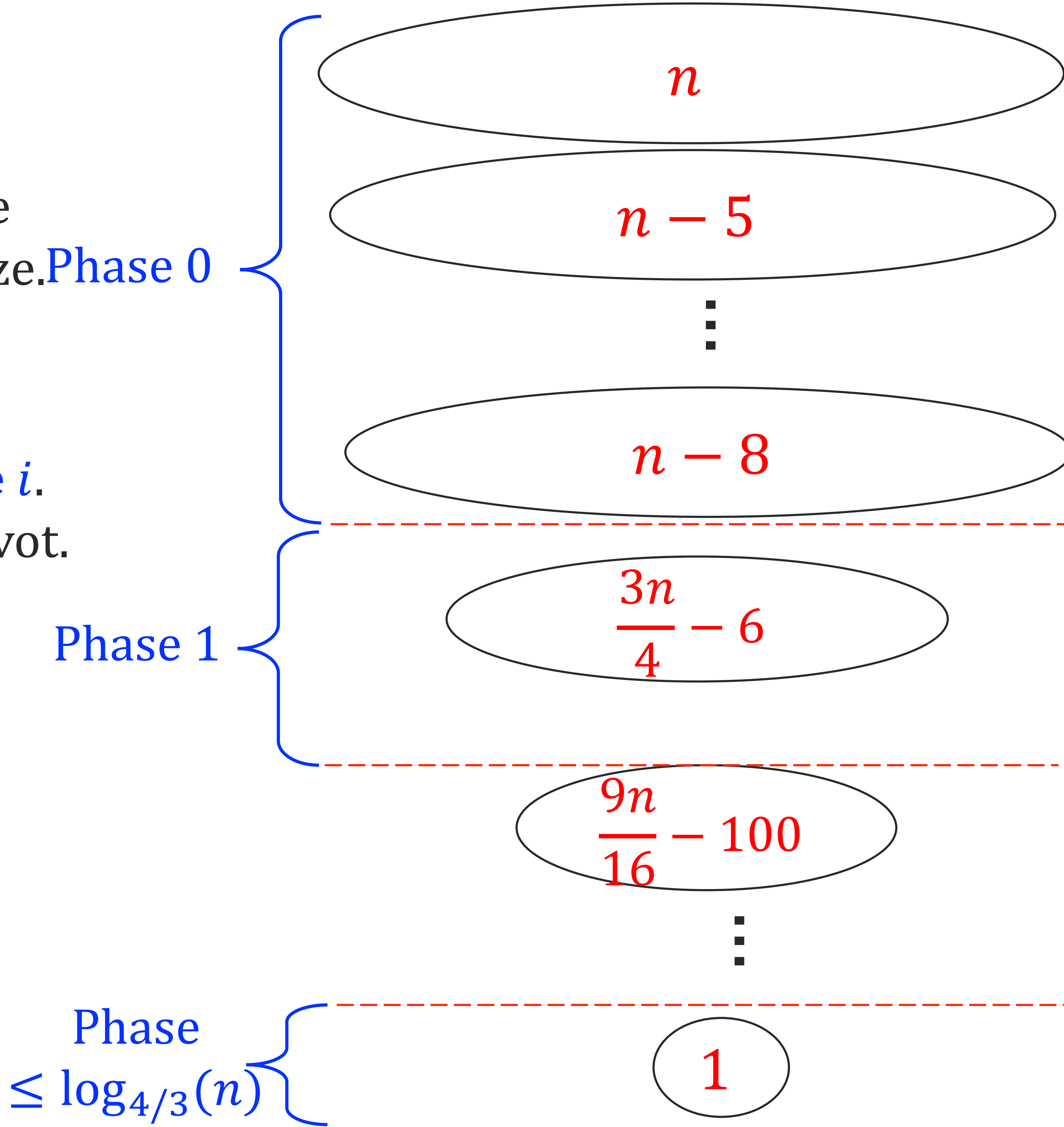
- In phase i, problem size $\leq \left(\frac{3}{4}\right)^i n$.
- X_i : random variable for length of phase i.
  Equiv, # tries until we choose a good pivot.

What is the contribution of phase i?

$$\leq X_i \cdot c \left(\frac{3}{4}\right)^i n$$

Total runtime:

$$\mathbb{E}[T(n)] \leq \sum_{i=0}^{\log_{4/3}(n)} \mathbb{E}[X_i] c\, n \left(\frac{3}{4}\right)^i$$



Expected Phase Length

We want to compute the expected phase length X_i.

$$\mathbb{E}[X_i] = \sum_{s=1}^{\infty} s \Pr[X_i = s]$$

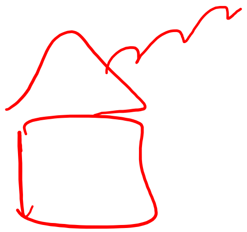
Recall, X_i is the number of times we choose a pivot in phase i.

Same as the number of pivots chosen until one falls in the middle 50% of the elements.

Discuss

What is $\Pr[X_i = s]$? $\left(\frac{1}{2}\right)^{s-1} \cdot \left(\frac{1}{2}\right) = \left(\frac{1}{2}\right)^{s}$

And what is $\mathbb{E}[X_i]$? $\sum_{s=1}^{\infty} \frac{s}{2^s} \leq 2$



Expected Phase Length

We want to compute the expected phase length X_i.

$$\mathbb{E}[X_i] = \sum_{s=1}^{\infty} s \Pr[X_i = s]$$

Recall, X_i is the number of times we choose a pivot in phase i. Same as the number of pivots chosen until one falls in the middle 50% of the elements.

Discuss

What is $\Pr[X_i = s]$? $= \left(\frac{1}{2}\right)^{s-1} \times \frac{1}{2} = \left(\frac{1}{2}\right)^{s}$

Explanation: $X_i = s$ means that the first $s-1$ pivots were bad (happens with prob $½^{s-1}$) and the last pivot was good (happens with prob ½).

And what is $\mathbb{E}[X_i]$? $\sum_{s=1}^{\infty} \frac{s}{2^s} \leq 2$

Computing the Expected Runtime

There are at most $\log_{4/3}(n)$ phases and each contributes $\leq X_i \cdot c \left(\frac{3}{4}\right)^i n$

- Total expected runtime

$$\mathbb{E}[T(n)] \leq \mathbb{E}\left[\sum_{i=0}^{\log_{\frac{4}{3}}(n)} X_i \cdot c \cdot n \left(\frac{3}{4}\right)^i\right]$$

$$= \sum_{i=0}^{\log_{4/3}(n)} \mathbb{E}[X_i] \cdot c \cdot n \left(\frac{3}{4}\right)^i$$

$$= \sum_{i=0}^{\log_{4/3}(n)} 2 \cdot c \cdot n \left(\frac{3}{4}\right)^i \in O(n)$$

Yes! There is randomized algorithm that solves $\text{SELECT}(S,k)$ in expected runtime of $O(n)$!

This algorithm is called QuickSelect.

Wrap up

Matrix Multiplication:

Strassen's algorithm

Similar to Karatsuba, we reduce the number of subproblems from 8 to 7.

k-Select

There is a randomized alg, with expected $O(n)$ runtime.

There is also a really cool deterministic algorithm, whose runtime is always $O(n)$.

Master theorem in action:

Matrix multiplication and selection

Next time

- Multiplying polynomials!
- Fast Fourier Transform