CS 170
Efficient Algorithms and Intractable Problems

Lecture 4:
Divide and Conquer III

Nika Haghtalab and John Wright

EECS, UC Berkeley

Announcements

How are discussions going?

Homework party

- Tomorrow (Friday), 10-2 @ Cory courtyard

Nika's office hour for next week:

- No after-class OH on Tuesday --- Lecture will be by Prof. John Wright.
- Instead Monday (Feb 3) 1:15-2pm @Cory courtyard

Recap of last lecture

Matrix Multiplication:

Strassen's algorithm

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

(Median) Selection

We saw that a good pivot selection gives us O(n)

This lecture

- Continue with Median selection

→Formalize the discussion of pivots and prove O(n) runtime

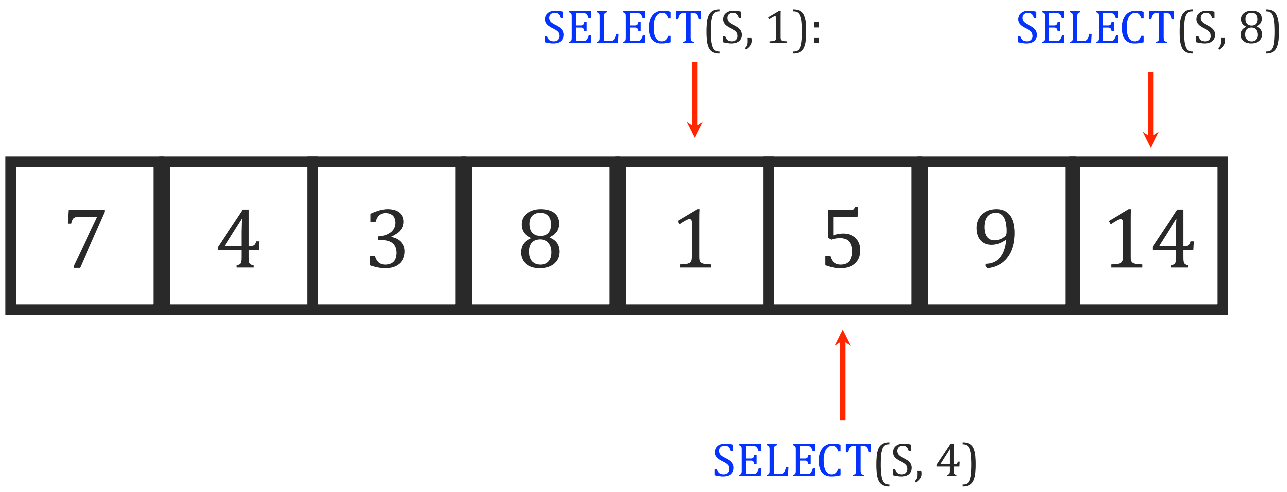
- One last example of Divide and Conquer: the Closest Pair problem

(Median) Selection

Recap: The k-select Problem

Input: Given an array S of n numbers and $k \in \{1, 2, \ldots, n\}$,

Output: Find the kth smallest element of S.



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

Recap: Big Question

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

Recap: Divide and Conquer for SELECT(S, k)

We want to divide the problem to subproblems. How?

- Given a **"pivot" v.** Split the array into three pieces

2	36	5	21	8	13	11	20	5	4	1

Given "pivot"

2	4	1

S_L: Elements less than the pivot

5	5

S_v: Elements equal to the pivot

36	21	8	13	11	20

S_R: Elements larger than the pivot

SELECT(S, k):

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

Recap: The Recurrence Relation

SELECT(S, k):

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

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

$$T(n) \leq T(\max\{len(S_L), len(S_R)\}) + O(n)$$

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

Let’s formalize what is a ”good” pivot

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 len(S_L) and 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)?

S_L S_v S_R

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

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

In the Tree Method

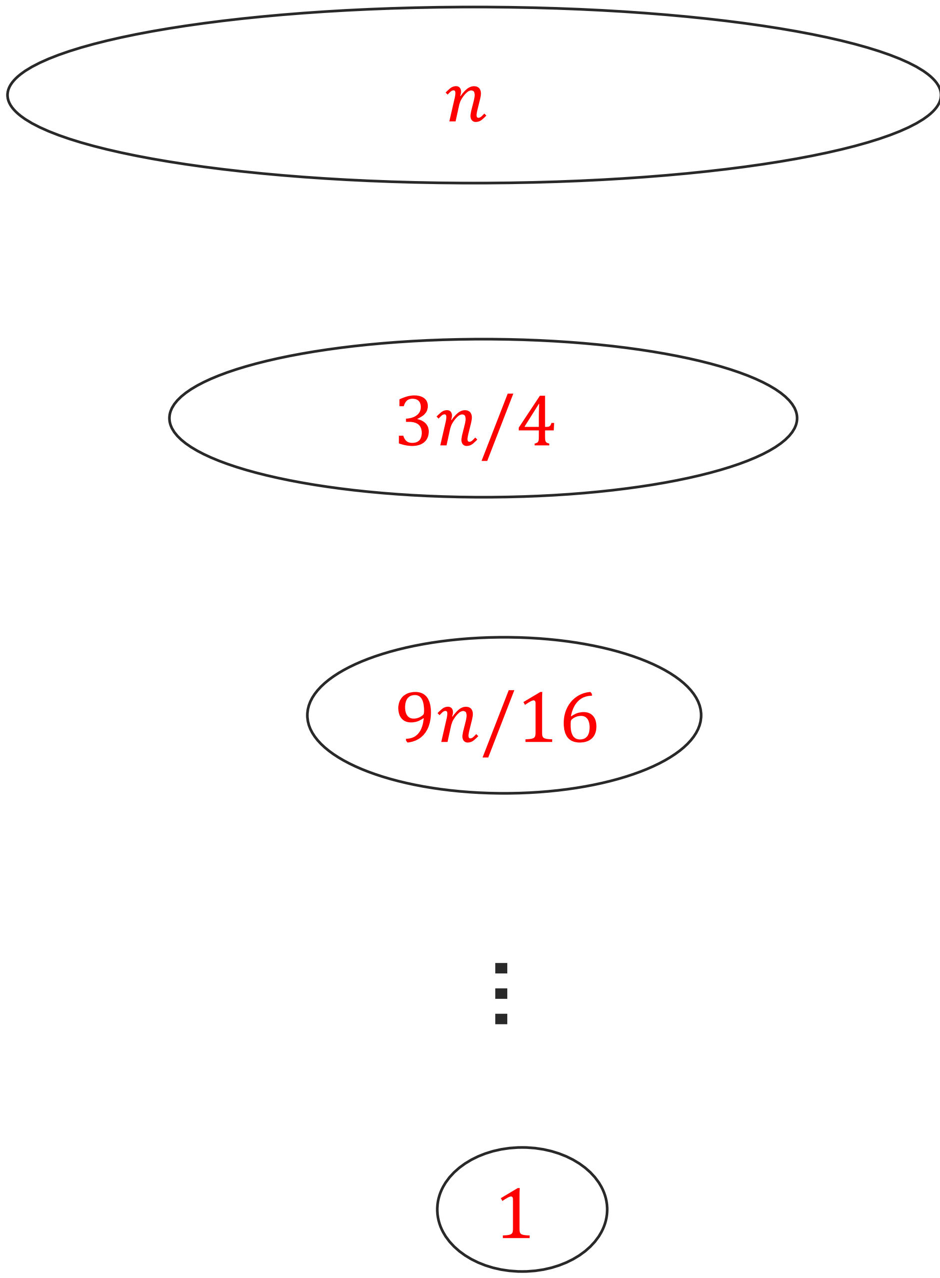
Let's repeat the runtime analysis with the tree method: If in every round we got a "good" pivot, then we multiply the size by $\leq 3/4$.

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?

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



Another Thought Exercise on Good Pivots

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

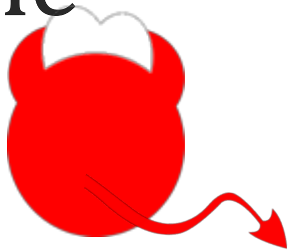
→ Every time we have a "good" pivot, the problem size shrinks to $\frac{3}{4}$ of the last one.

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

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

This algorithm is called QuickSelect and selects a uniformly random pivot on every turn.

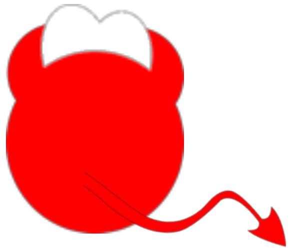
Randomized Algorithms and Expected Runtime



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

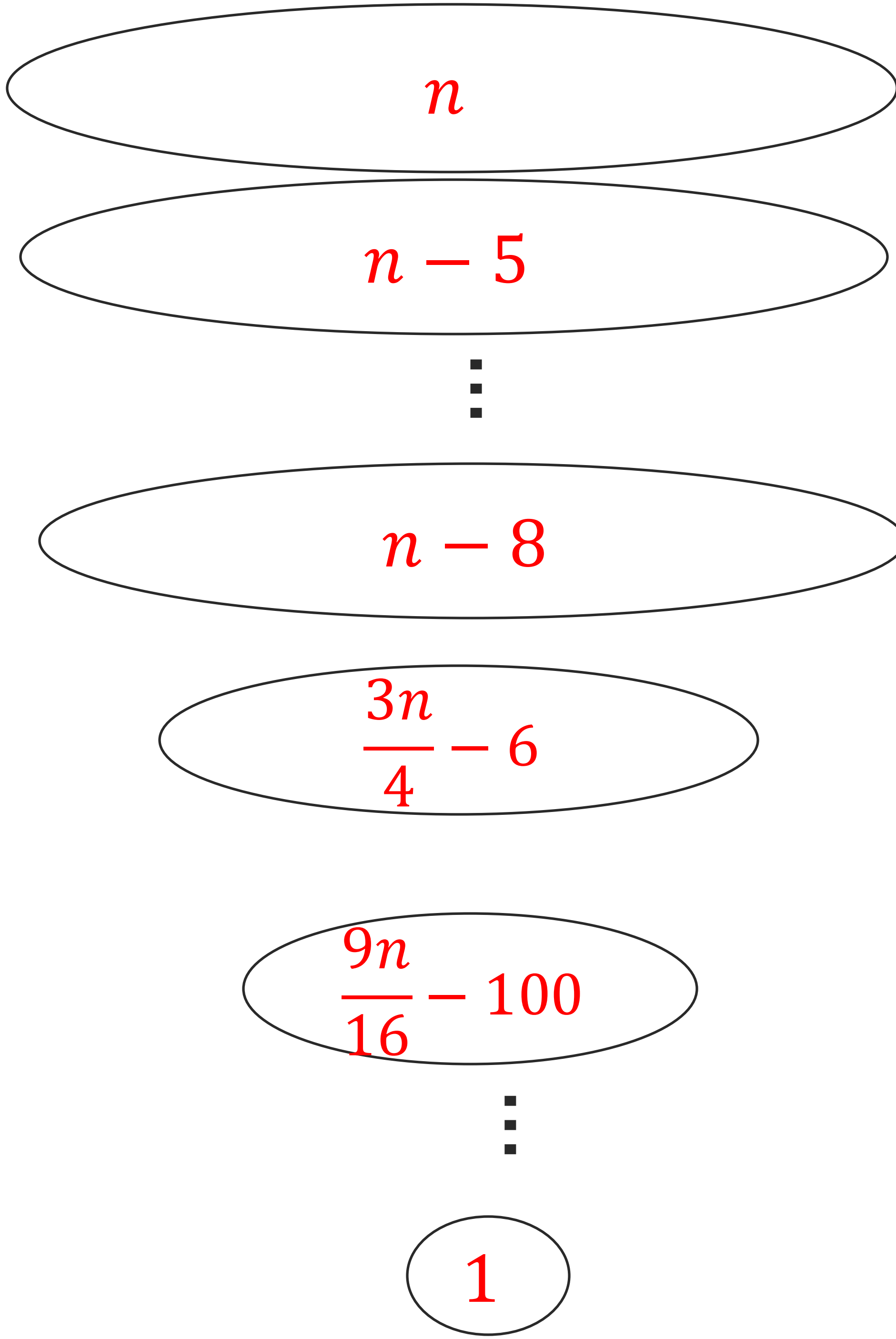
Trees Revisited

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

Whenever we get a “good” pivot, we multiply the problem size by $\leq 3/4$.

In some steps, we don’t get a good pivot.

Divide the tree method to phases, indicating when the problem size shrinks by $3/4$.



Trees Revisited

Partition layers to "phases":

- Phase i include layers $[s_i, s_{i+1})$.
- s_{i+1}: layer when the problem size first becomes $\leq \frac{3}{4}$ problem size of layer s_i.

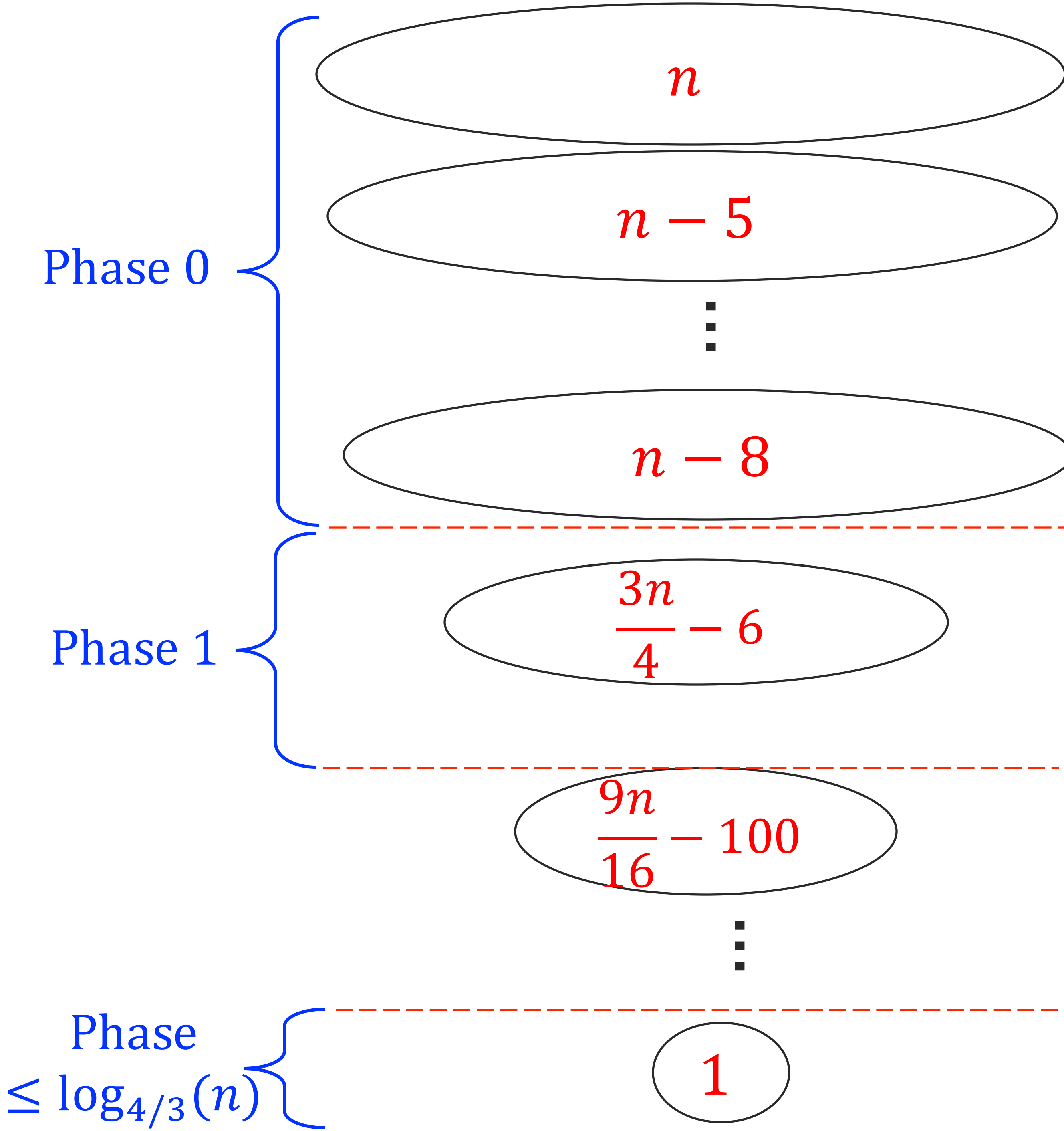
Lemma

In phase i, problem size $\leq \left(\frac{3}{4}\right)^i n$

Lemma

Total runtime:

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



Length of a Phase

The length of $Phase_i$:

- Random variable
- Eqv. how many tries it gets to shrink the problem size to ¾
- Length of phase ≤ # random pivots until we get a "good" pivot in phase i.

Lemma

Total runtime:

$$T(n) \leq \sum_{i=0}^{\log_{4/3}(n)} \underbrace{len(Phase_i)}_{\text{Random variable}} \, c\, n \left(\frac{3}{4}\right)^i$$

Discuss

Every time we choose a pivot at random, with probability 50%, it is a ``good''. So, $\Pr[len(Phase_i) = 1] = 0.5$.

What is $\Pr[len(Phase_i) = 2]$?

What is $\Pr[len(Phase_i) = 3]$?

What is $\Pr[len(Phase_i) = m]$?

For ease, assume no element is repeated.
Remove this assumption at home!

Expected Phase Length

We want to compute the expected phase length $len(Phase_i)$

$$\mathbb{E}[len(Phase_i)] = \sum_{m=1}^{\infty} m \Pr[len(Phase_i) = m]$$

Discuss

Compute $\mathbb{E}[len(Phase_i)]$?

Computing the Expected Runtime

Total expected runtime

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

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

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

Recall

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

Recall

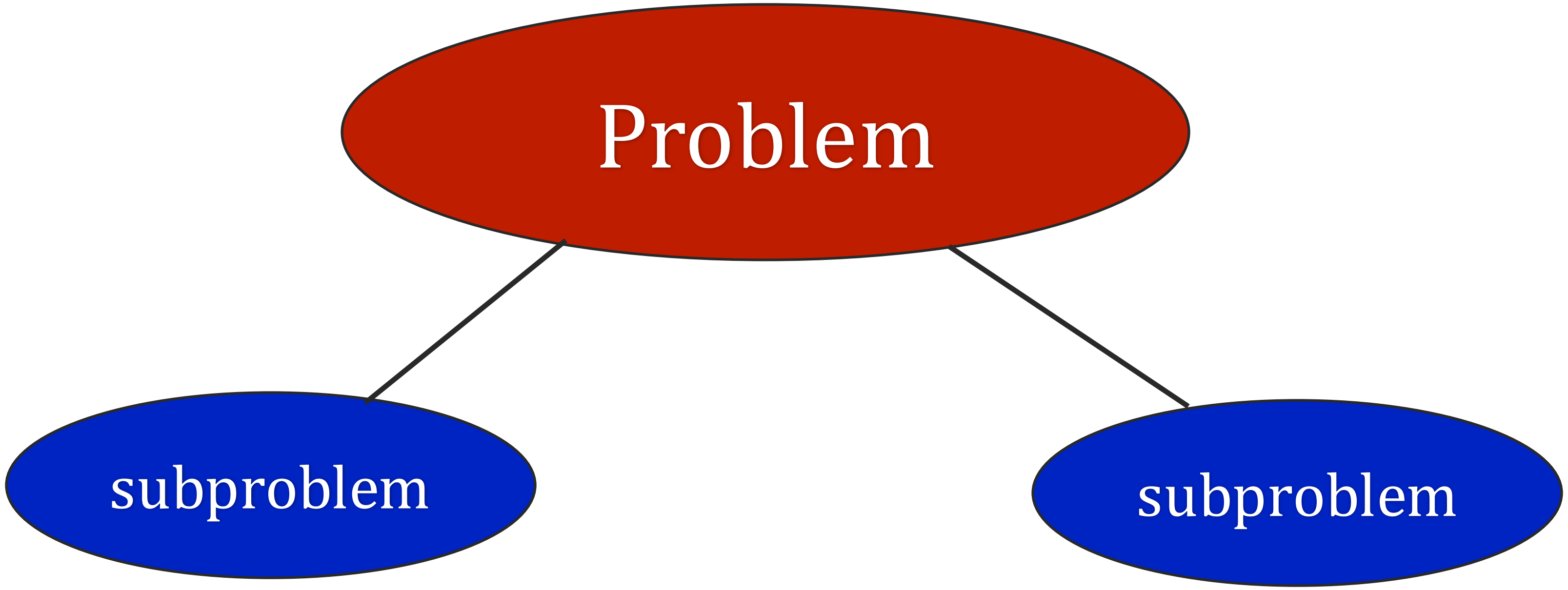
$$\mathbb{E}\left[len(Phase_i)\right] \leq 2$$

More on SELECT(S, k)

We gave a randomized algorithm, with *expected* $O(n)$ runtime.

There is also a cool deterministic algorithm, whose runtime is always $O(n)$.
→ It requires a more involved pivot selection mechanism
→ We won't talk about it in class.

The Big Picture of Divide and Conquer



First divide the problem to subproblems
→Subproblems are solved recursively ← <u>Non-trivial Divide Step:</u> QuickSelect, Strassen's, Karatsuba

Then combine the subproblems solutions to provide the answer to the big problem.

<u>Next: An example of non-trivial combine step</u>

ClosestPair

Input: Given n points on the plane $S = \{p_i = (x_i, y_i) \mid i = 1, \dots, n\}$

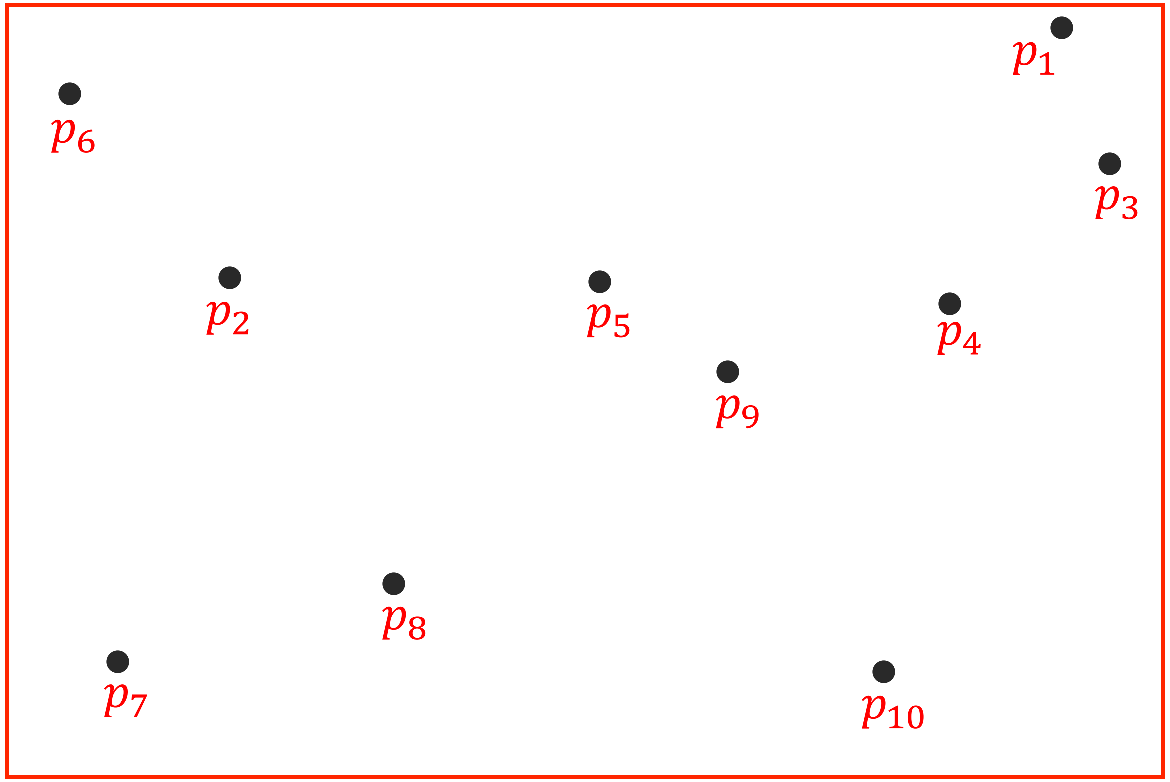
Output: Closest pair of points (p_i, p_j).

Naïve Algorithm:

- Compute all (p_i, p_j) distances, and output the pair with closest distance.
- $O(n^2)$ pairs, so $O(n^2)$ runtime.

Can we do better?

Can we get $O(n \ln(n))$?



Divide

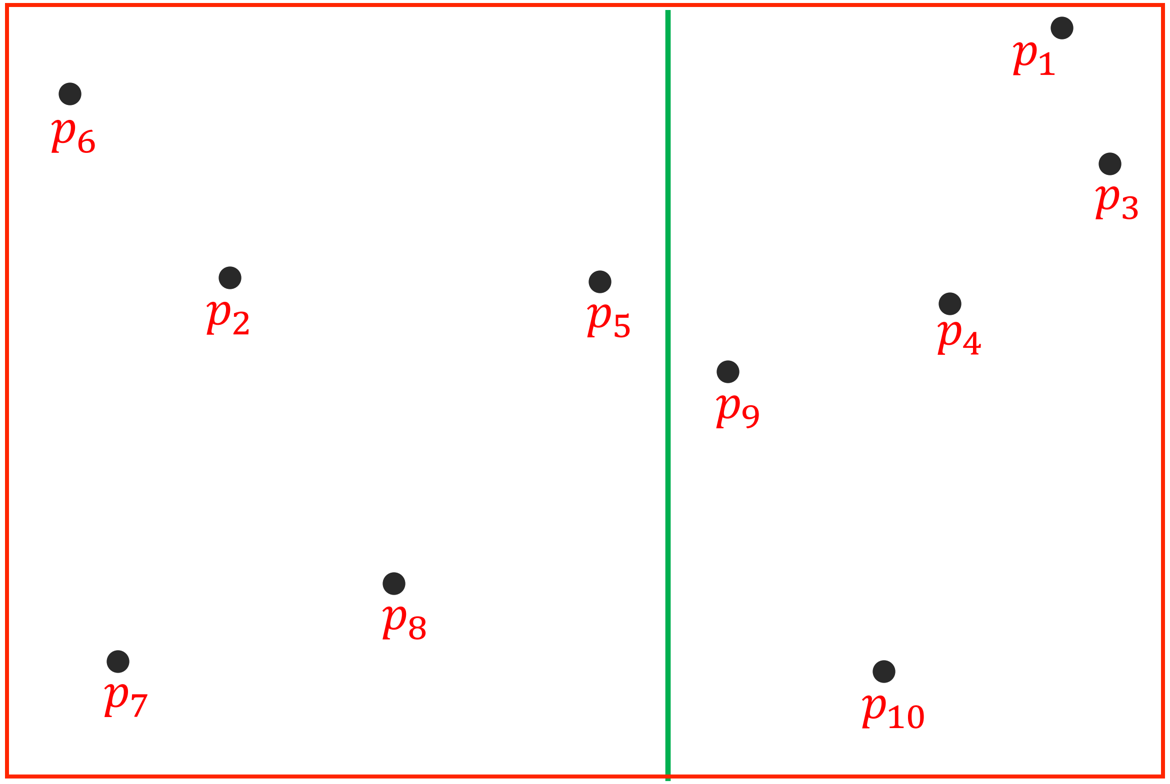
Since we are shooting for $O(n \ln(n))$, we might as well make two sorted lists according to the x-axis and y-axis at the beginning.

How should we divide?

- Divide on x-axis, along the *median*!
- Split the list (keep it sorted)

We have two subproblems

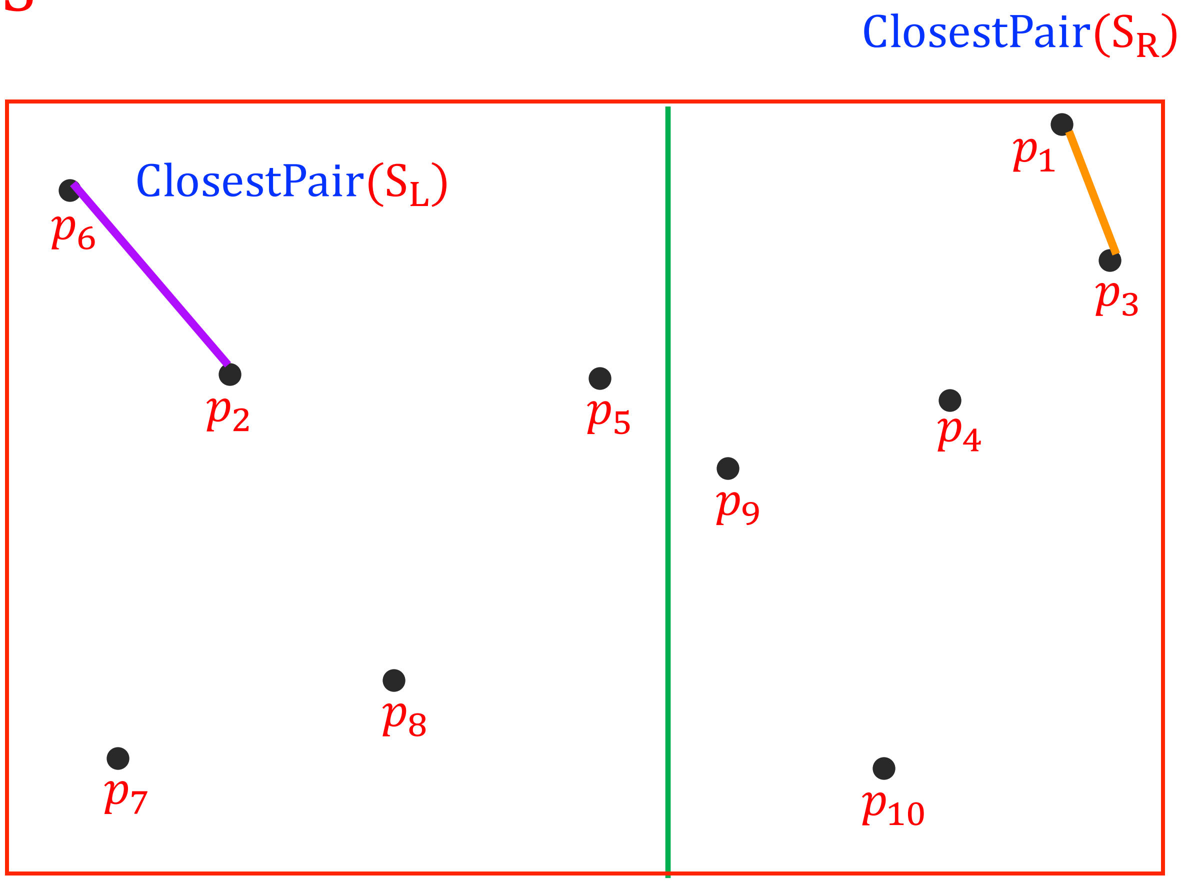
- $S_L = \{p_6, p_7, p_2, p_8, p_5\}$
- $S_R = \{p_9, p_{10}, p_4, p_1, p_3\}$



Where is the closest pair?

We should solve the two subproblems recursively:

- ClosestPair$(S_L) = (p_6, p_2)$ and ClosestPair$(S_R) = (p_1, p_3)$
- One of these could be the closest pair in S
- Or, the closest pair crosses the median!



Checking for pairs across median naively:
→ Try every pair that crosses the median.
→ $\frac{n}{2} \times \frac{n}{2} \in \Theta(n^2)$

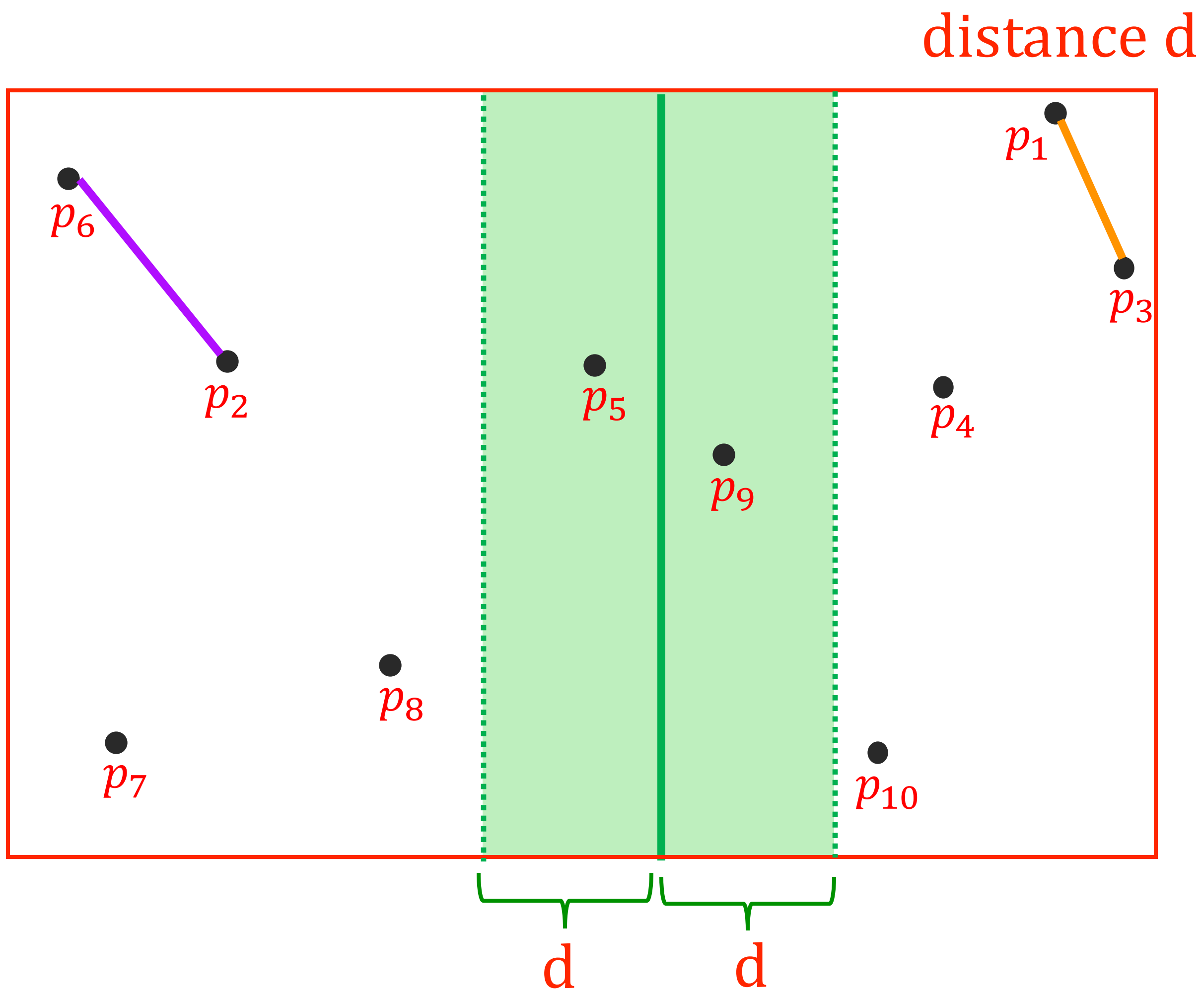
Idea 1

Let d be the min of two distances ClosestPair(S_L) and ClosestPair(S_R)

Then restrict attention to the strip of width d on each side of the median

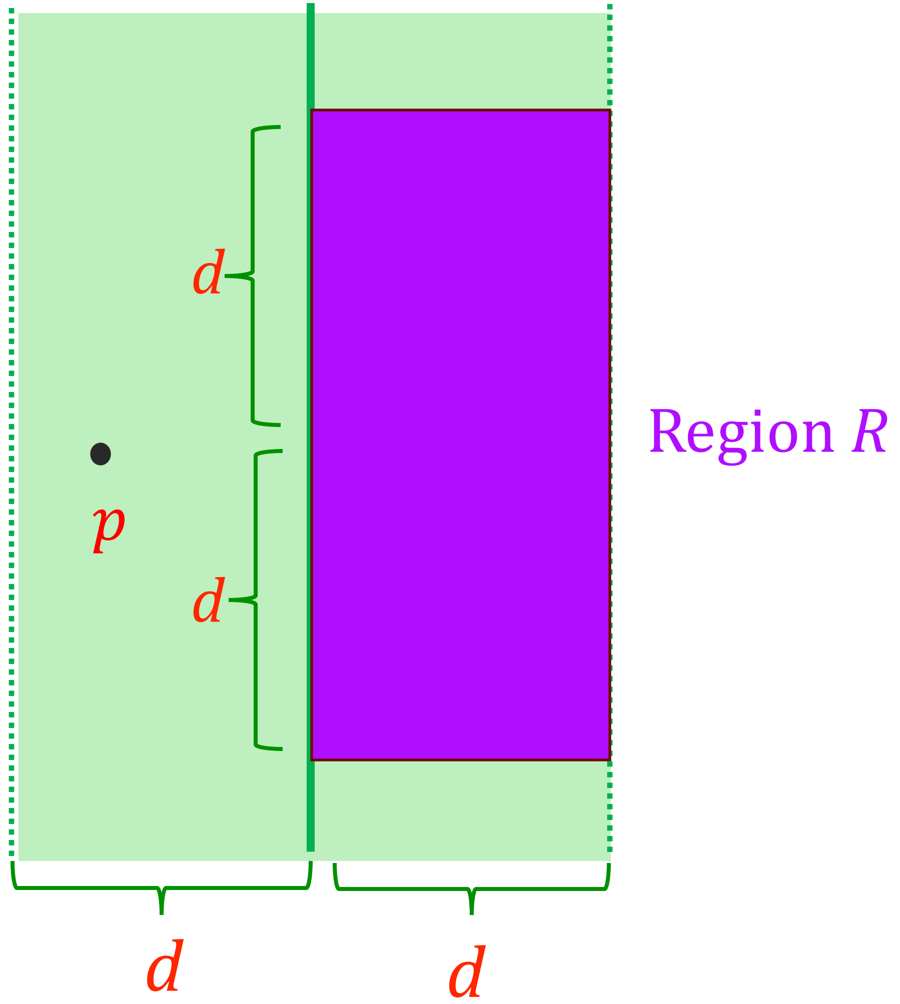
This is not good enough

- In the worst-case, all points can be in this strip → Still take $\Theta(n^2)$.



Better Idea

Any point p in this strip, needs to be measured only against points in region R.



Claim

There are $O(1)$ points in R!

Divide and Conquer Algorithm

Let's believe in this claim that every p in strip needs to be measured against only $O(1)$ other points. What is the algorithm and runtime?

Write pseudo-code (esp. steps 4-5)
Start with sorted arrays, both x and y

ClosestPair($\{p_i = (x_i, y_i) \mid i = 1, \ldots, n\}$)

1. $(p_L, p'_L) \leftarrow$ ClosestPair$(p_1, \ldots, p_{n/2})$
2. $(p_R, p'_R) \leftarrow$ ClosestPair$(p_{\frac{n}{2}+1}, \ldots, p_n)$

$\Big\}$ $2T\left(\frac{n}{2}\right)$

3. $d \leftarrow \min\{dist(p_L, p'_L), dist(p_R, p'_R)\}$
4. Compute points in the strip
5. For any point in the strip, compare the point to the $O(1)$ relevant points described in previous slide. Update minimum distance pair if needed.

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

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \in O(n\log(n))$$

Via Master Theorem

Proof that O(1) points are in Region R

Claim

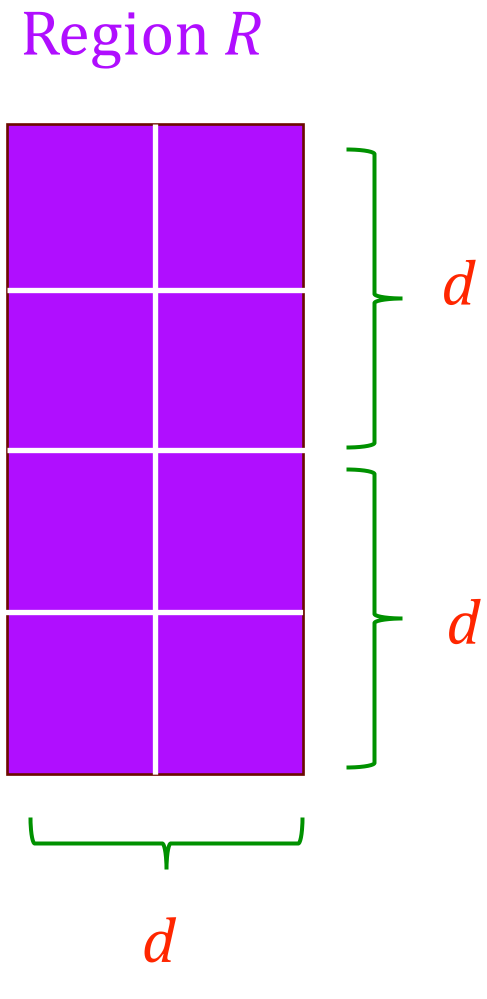
There are ≤ 8 points in R!

Proof: Divide region R to 8 squares, if size $\frac{d}{2} \times \frac{d}{2}$.

Assume there are more than 8 points in R.

- Pigeon hole: One square has more than 1 point
- The diameter of these squares is less than d, so these points have distance $< d$ to each other
- They are both on the same side of the median too.

→ Contradicts that d was the defined as the smallest distance between a pair of points on either side of the median!



More on ClosestPair(S, k)

We gave a deterministic algorithm, with runtime $O(n \log(n))$.

There is also a cool randomized algorithm, whose expected runtime is $O(n)$.
→ It requires a more involved divide and conquer analysis
→ We won't talk about it in class.

Wrap up

In the first two weeks of class, we saw many examples of Divide and Conquer

Integer and Matrix Multiplication
(Median) Selection
Closest Pair

Solving recurrences
Master theorem

Divide and Conquer
→ Is a powerful method
→ Both the divide and combine steps can be versatile

Next week:

Graph Problems and Algorithms

Prof. John Wright