CSC373

Algorithm Design, Analysis & Complexity

Karan Singh

Introduction

- Instructors
  - Karan Singh
    - dgp.toronto.edu/~karan, karan@dgp, BA 5258
    - SEC 5101 and 5201
  - Nisarg Shah
    - cs.toronto.edu/~nisarg, nisarg@cs, SF 2301C
    - SEC 5301
- TAs: Too many to list

Introduction

- Lectures
  - 5101: Tue 1–3 in BA1170, Thu 2–3 in BA1170
  - 5201: Tue 3–4 in BA1170, Thu 3–5 in SS 2117
- Tutorials
  - Every Mon 5-6pm
  - Divided by birth month
  - 5101: Jan-Jun: SS 1070, Jul-Dec: SS 1073
  - 5201: Jan-Jun: SS 1074, Jul-Dec: UC 244
- Office Hours Tue noon-1, Thu 1-2 in BA5258

No tutorial on Sep 9

Check the course webpage
for further announcements

Course Information

- Course Page
  www.cs.toronto.edu/~nisarg/teaching/373f19/
  - All the information below is in the course information sheet, available on the course page

- Discussion Board
  piazza.com/utoronto.ca/fall2019/csc373

- Grading – MarkUs system
  - Link will be distributed after about two weeks
  - LaTeX preferred, scans are OK!
  - An arbitrary subset of questions may be graded…

Course Organization

- Tutorials
  - A problem sheet will be posted ahead of the tutorial
  - Easier problems that are warm-up to assignments/exams
  - You’re expected to try them before coming to the tutorial
  - TAs will solve the problems on the board
  - *No written/typed solutions will be posted*

Course Organization

- Assignments
  - 4 assignments
  - In *groups of up to three* students
  - Final marks will be taken from best 3 out of 4
  - Questions will be more difficult
    - May need to mull them over for several days; do *not* expect to start and finish the assignment on the same day!
    - May include bonus questions
  - Submit *a single PDF* on MarkUs
    - May need to compress the PDF

Course Organization

- Exams
  - Two term tests, one final exam
  - Details will be posted on the course webpage
  - In each exam, you’ll be allowed to bring one 8.5” x 11” sheet of *handwritten* notes on *one side*

Grading Policy

• 3 homeworks	*	10%	=	30%
• 2 term tests	*	20%	=	40%
• Final exam	*	30%	=	30%

- NOTE: If you earn less than 40% on the final exam, your final course grade will be reduced below 50

Textbook

- Primary reference: lecture slides

- Primary textbook (required)
  - [CLRS] Cormen, Leiserson, Rivest, Stein: *Introduction to Algorithms*.

- Supplementary textbooks (optional)
  - [DPV] Dasgupta, Papadimitriou, Vazirani: *Algorithms*.
  - [KT] Kleinberg; Tardos: *Algorithm Design*.

Other Policies

- Collaboration
  - Free to discuss with classmates or read online material
  - Must write solutions in your own words
    - Easier if you do not take any pictures/notes from discussions

- Citation
  - For each question, must cite the peer (write the name) or the online sources (provide links), if you obtained a significant insight directly pertinent to the question
  - Failing to do this is plagiarism!

Other Policies

- “No Garbage” Policy
  - Borrowed from: Prof. Allan Borodin (citation!)

1. Partial marks for viable approaches
2. Zero marks if the answer makes no sense
3. 20% marks if you admit to not knowing how to approach the question (“I do not know how to approach this question”)

- 20% > 0% !!

Other Policies

- Late Days
  - 4 total late days across all 4 assignments
  - Managed by MarkUs
  - At most 2 late days can be applied to a single assignment
  - Already covers legitimate reasons such as illness, university activities, etc.
    - Petitions will only be granted for circumstances which cannot be covered by this

Enough with the boring stuff.

What will we study?

Why will we study it?

Muhammad ibn Musa al-Khwarizmi
c. 780 – c. 850

What is this course about?

- Algorithms
  - Ubiquitous in the real world
    - From your smartphone to self-driving cars
    - From graph problems to graphics problems
  - Important to be able to design and analyze algorithms
  - For some problems, good algorithms are hard to find
    - For some of these problems, we can formally establish complexity results
    - We'll often find that one problem is easy, but its minor variants are suddenly hard

What is this course about?

- Algorithms
  - Algorithmic prefixes… distributed, parallel, streaming, sublinear time, spectral, genetic…
  - There are also other concerns with algorithms
    - Fairness, ethics, …

…mostly beyond the scope of this course.

What is this course about?

- Algorithm design paradigms in this course
  - Divide and Conquer
  - Greedy
  - Dynamic programming
  - Network flow
  - Linear programming
  - Approximation algorithms
  - Randomized algorithms

What is this course about?

- How do we know which paradigm is right for a given problem?
  - A very interesting question!
  - Subject of much ongoing research…
    - Sometimes, you just know it when you see it…

- How do we analyze an algorithm?
  - Proof of correctness
  - Proof of running time
    - We'll try to prove the algorithm is *efficient* in the *worst case*
    - In practice, average case matters just as much (or even more)

What is this course about?

- What does it mean for an algorithm to be efficient in the worst case?
  - Polynomial time
  - It should use at most poly(n) steps on *any* n-bit input
    - $n, n^2, n^{100}, 100n^6 + 237n^2 + 432, \dots$
  - How much is too much?

What is this course about?

Better Balance by Being Biased:
A 0.8776-Approximation for Max Bisection

Per Austrin*, Siavosh Benabbas*, and Konstantinos Georgiou†

has a lot of flexibility, indicating that further improvements may be possible. We remark that, while polynomial, the running time of the algorithm is somewhat abysmal; loose estimates places it somewhere around $O\left(n^{10^{100}}\right)$; the running time of the algorithm of [RT12] is similar.

What is this course about?

Picture-Hanging Puzzles*

Erik D. Demaine[†] Martin L. Demaine[†] Yair N. Minsky[‡] Joseph S. B. Mitchell[§]
Ronald L. Rivest[†] Mihai Pătraşcu[¶]

Theorem 7 *For any $n \geq k \geq 1$, there is a picture hanging on n nails, of length $n^{c'}$ for a constant c', that falls upon the removal of any k of the nails.*

$n^{6{,}100 \log_2 c}$. Using the $c \leq 1{,}078$ upper bound, we obtain an upper bound of $c' \leq 6{,}575{,}800$. Using

So, while this construction is polynomial, it is a rather large polynomial. For small values of n, we can use known small sorting networks to obtain somewhat reasonable constructions.

What is this course about?

- What if we can’t find an efficient algorithm for a problem?
  - Try to prove that the problem is hard
  - Formally establish complexity results
  - NP-completeness, NP-hardness, …

- We’ll often find that one problem may be easy, but its simple variants may suddenly become hard…
  MST vs. Steiner Tree or bounded degree MST,
  shortest vs. longest simple path,
  2-colorability vs. 3-colorability.

I’m not convinced.

Will I really ever need to know how to design abstract algorithms?

At the very least…

This will help you prepare for your technical job interview!

Microsoft: Four people with one flashlight, need to cross a rickety bridge at night. Two people max. can cross the bridge at one time, and anyone crossing must walk with the flashlight. A takes 1 minute to cross the bridge, B takes 2, C takes 5, and D takes 10 minutes. A pair must walk together. Find the fastest way for them to cross.

Divide & Conquer? Greedy?

Disclaimer

- The course is theoretical in nature
  - You'll be working with abstract notations, proving correctness of algorithms, analyzing the running time of algorithms, designing new algorithms, and proving complexity results.
- Question
  - How many of you are somewhat scared going into the course?
  - How many of you feel comfortable with proofs, and want challenging problems to solve?
  - How many prefer concrete examples to abstract symbols?

We'll have something for everyone to enjoy this course

Related/Follow-up Courses

- Direct follow-up
  - CSC473: Advanced Algorithms
  - CSC438: Computability and Logic
  - CSC463: Computational Complexity and Computability

- Algorithms in other contexts
  - CSC304: Algorithmic Game Theory and Mechanism Design (Nisarg Shah)
  - CSC384: Introduction to Artificial Intelligence
  - CSC436: Numerical Algorithms
  - CSC418: Computer Graphics

Divide & Conquer

History?

- How many of you saw some divide & conquer algorithms in, say, CSC236/CSC240 and/or CSC263/CSC265?

- Maybe you saw a subset of these algorithms?
  - Mergesort - $O(n \log n)$
  - Karatsuba algorithm for fast multiplication - $O\left(n^{\log_2 3}\right)$ rather than $O(n^2)$
  - Largest subsequence sum in $O(n)$
  - …

Divide & Conquer

- General framework
  - Break (a large chunk of) a problem into smaller subproblems of the same type
  - Solve each subproblem recursively
  - At the end, quickly combine solutions from the subproblems and/or solve any remaining part of the original problem

- Hard to formally define when a given algorithm is divide-and-conquer…
- Let's see some examples!

Raytracing: Where is the light coming from?

Divide&Conquer: Shoot multiple rays (sub-problems) recursively reflecting/refracting off objects in the scene and combine the results to determine color of pixels.

Master Theorem

- Here's the master theorem, as it appears in CLRS
  - Useful for analyzing divide-and-conquer running time
  - If you haven't already seen it, please spend some time understanding it

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

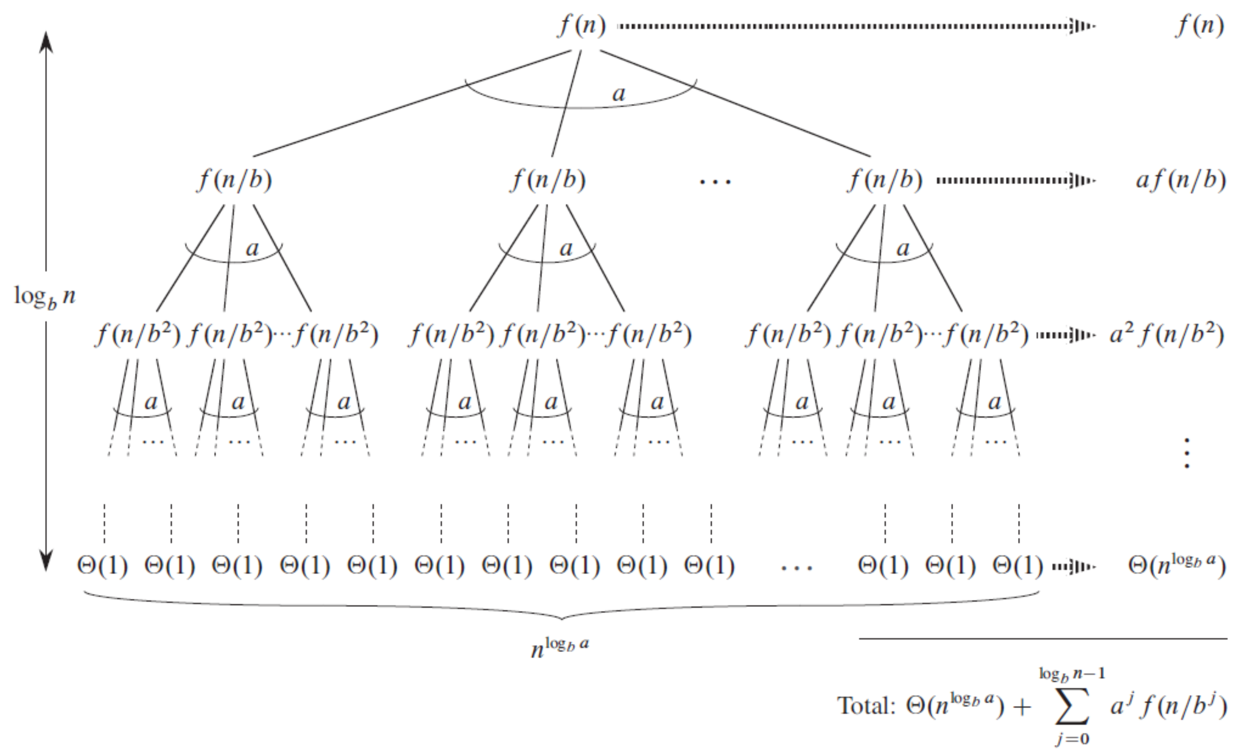
where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Master Theorem

Intuition:

Compare the function *f(n)* with the function $n^{\log_b a}$. The larger of the two functions determines the recurrence solution.



Counting Inversions

- Problem
  - Given an array a of length n, count the number of pairs (i, j) such that $i < j$ but $a[i] > a[j]$
- Applications
  - Voting theory
  - Collaborative filtering
  - Measuring the "sortedness" of an array
  - Sensitivity analysis of Google's ranking function
  - Rank aggregation for meta-searching on the Web
  - Nonparametric statistics (e.g., Kendall's tau distance)

Counting Inversions

- Problem
  - Count (i, j) such that $i < j$ but $a[i] > a[j]$
- Brute force
  - Check all $\Theta(n^2)$ pairs
- Divide & conquer
  - Divide: break array into two equal halves x and y
  - Conquer: count inversions in each half recursively
  - Combine:
    - Solve (remaining): count inversions with one entry in x and one in y
    - Merge: add all three counts

Counting Inversions

From Kevin Wayne's slides

SORT-AND-COUNT (L)

IF list L has one element

  RETURN $(0, L)$.

DIVIDE the list into two halves A and B.

$(r_A, A) \leftarrow$ SORT-AND-COUNT(A).

$(r_B, B) \leftarrow$ SORT-AND-COUNT(B).

$(r_{AB}, L') \leftarrow$ MERGE-AND-COUNT(A, B).

RETURN $(r_A + r_B + r_{AB},\ L')$.

Counting Inversions

input

1	5	4	8	10	2	6	9	3	7

count inversions in left half A

1	5	4	8	10

5-4

count inversions in right half B

2	6	9	3	7

6-3 9-3 9-7

count inversions (a, b) with a ∈ A and b ∈ B

1	5	4	8	10

2	6	9	3	7

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

output 1 + 3 + 13 = 17

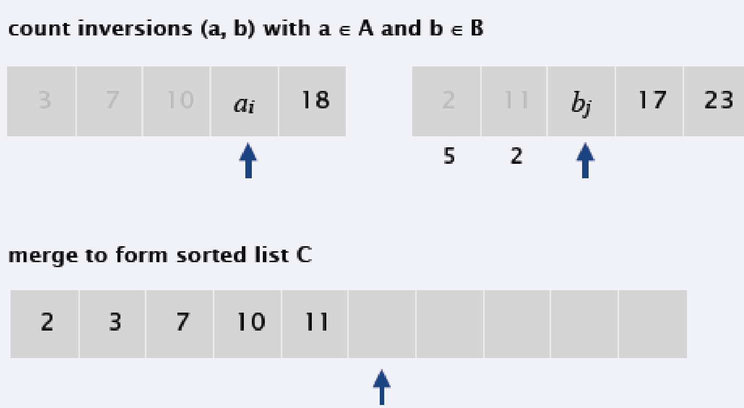
Counting Inversions

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



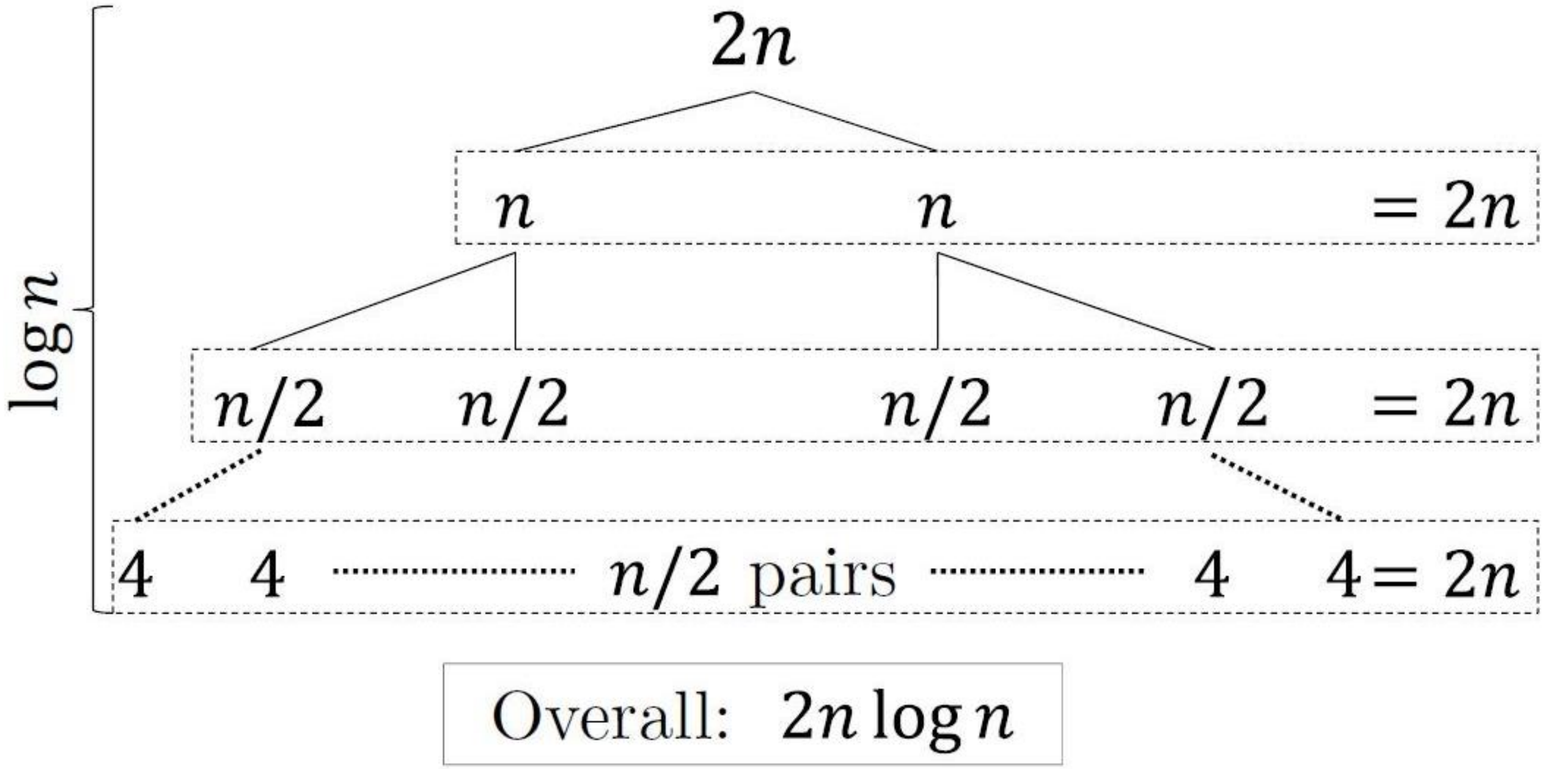
Counting Inversions

- How do we formally prove correctness?
  - Induction on n is usually very helpful
  - Allows you to assume correctness of subproblems

- Running time analysis
  - Suppose $T(n)$ is the running time for inputs of size n
  - Our algorithm satisfies $T(n) = 2\,T(n/2) + O(n)$
  - Master theorem says this is $T(n) = O(n \log n)$

Without Master Theorem

Let's say $T(n) = 2\,T(^n/_2) + 2n$



Closest Pair in $\mathbb{R}^2$

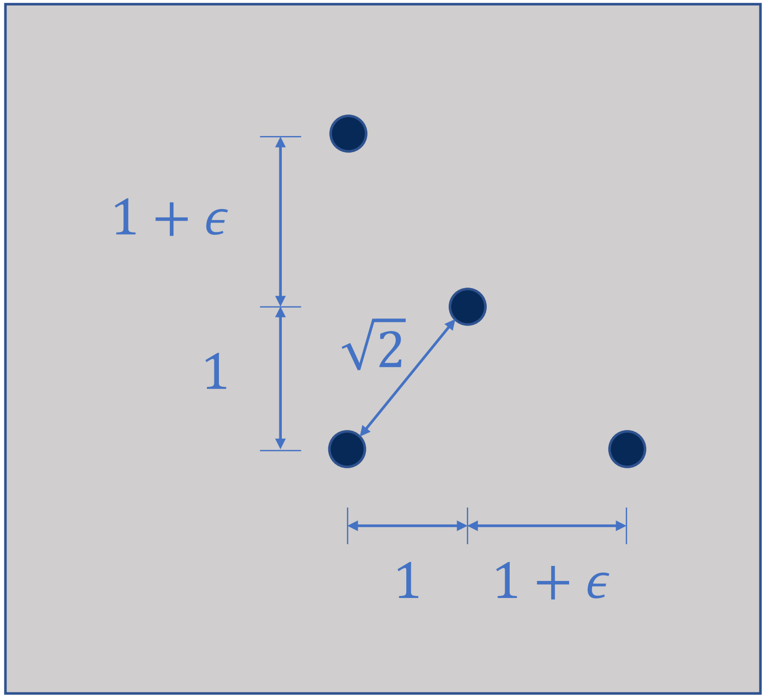
- Problem:
  - Given n points of the form (x_i, y_i) in the plane, find the closest pair of points.
- Applications:
  - Basic primitive in graphics and computer vision
  - Geographic information systems, molecular modeling, air traffic control
  - Special case of nearest neighbor
- Brute force: $\Theta(n^2)$

Intuition from 1D?

- In 1D, the problem would be easily $O(n \log n)$
  - Sort and check!
- Sorting attempt in 2D
  - Find closest points by x coordinate
  - Find closest points by y coordinate
- Non-degeneracy assumption
  - No two points have the same x or y coordinate

Intuition from 1D?

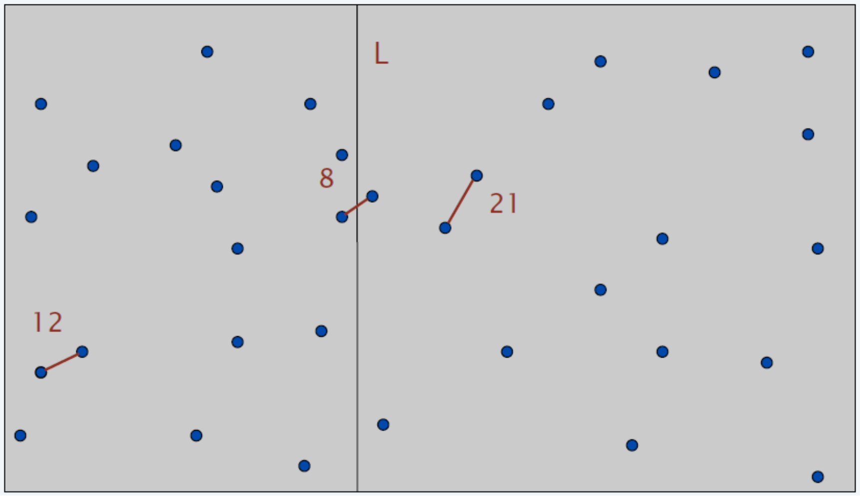
- Sorting attempt in 2D
  - Find closest points by x or y coordinate
  - Doesn't work!



Closest Pair in $\mathbb{R}^2$

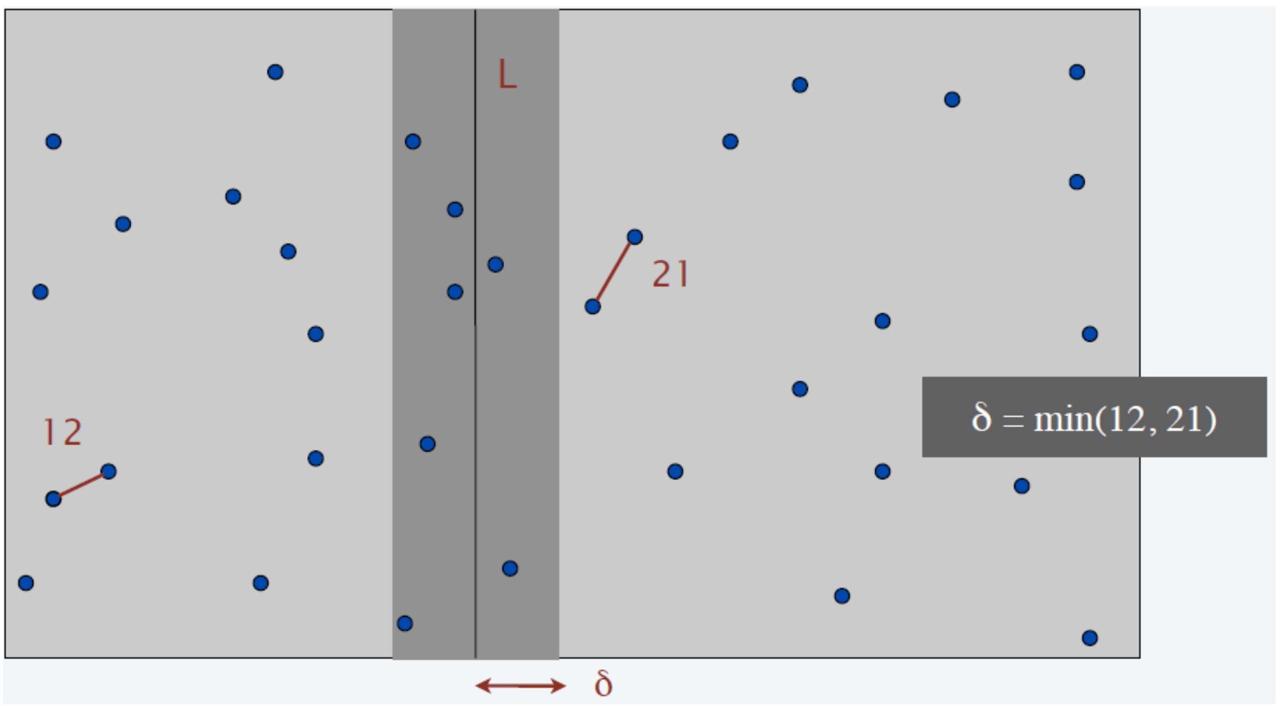
- Let's try divide-and-conquer!
  - Divide: points in equal halves by drawing a vertical line L
  - Conquer: solve each half recursively
  - Combine: find closest pair with one point on each side of L
  - Return the best of 3 solutions

Seems like $\Omega(n^2)$ ☹



Closest Pair in $\mathbb{R}^2$

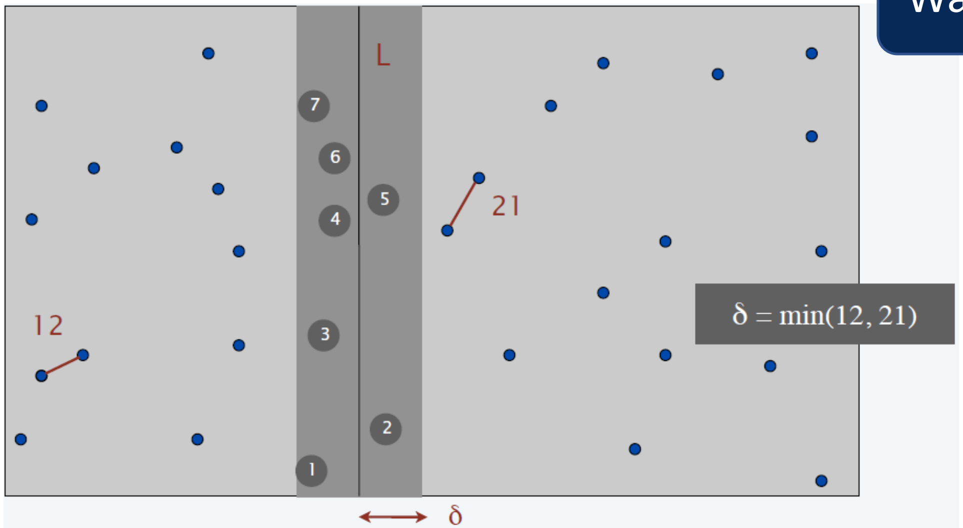
- Combine
  - We can restrict our attention to points within δ of L on each side, where δ = best of the solutions in two halves



Closest Pair in $\mathbb{R}^2$

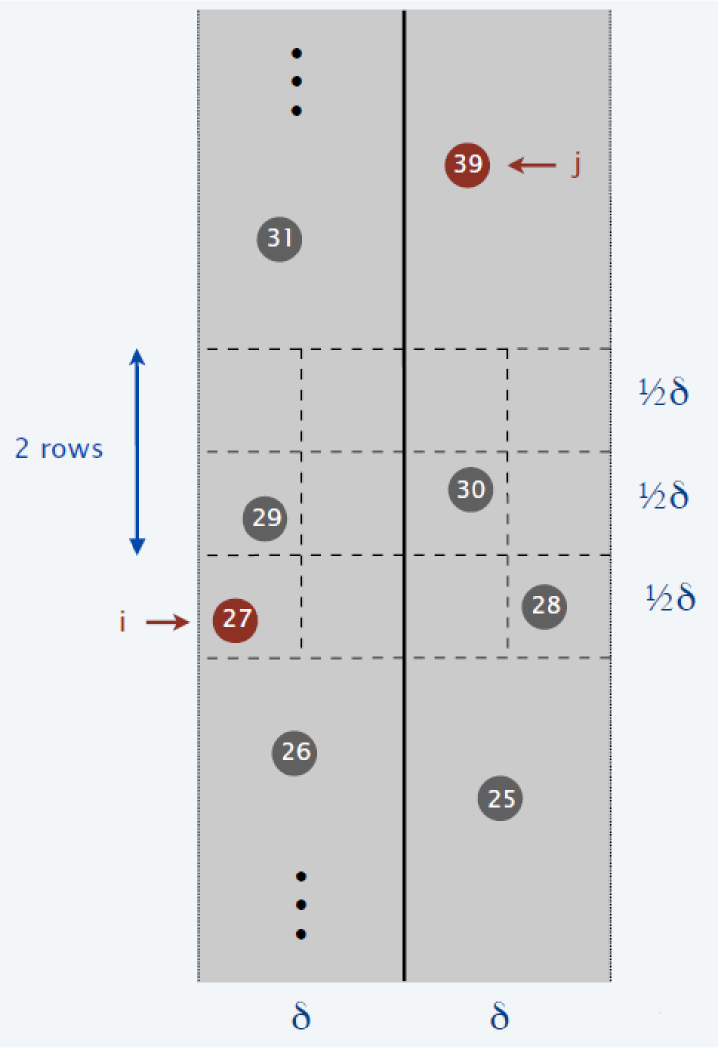
- Combine (let δ = best of solutions in two halves)
  - Only need to look at points within δ of L on each side,
  - Sort points on the strip by y coordinate
  - Only need to check each point with next 11 points in sorted list!

Wait, what? Why 11?



Why 11?

- Claim:
  - If two points are at least 12 positions apart in the sorted list, their distance is at least δ
- Proof:
  - No two points lie in the same $\delta/2 \times \delta/2$ box
  - Two points that are more than two rows apart are at distance at least δ



Recap: Karatsuba's Algorithm

- Fast way to multiply two n digit integers x and y
- Brute force: $O(n^2)$ operations

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

- Karatsuba's observation:
  - Divide each integer into two parts
    - $x = x_1 * 10^{n/2} + x_2, y = y_1 * 10^{n/2} + y_2$
    - $xy = (x_1 y_1) * 10^n + (x_1 y_2 + x_2 y_1) * 10^{n/2} + (x_2 y_2)$
  - Four ${}^{n}/_{2}$-digit multiplications can be replaced by three
    - $x_1 y_2 + x_2 y_1 = (x_1 + x_2)(y_1 + y_2) - x_1 y_1 - x_2 y_2$
  - Running time
    - $T(n) = 3\,T({}^{n}/_{2}) + O(n) \Rightarrow T(n) = O\left(n^{\log_2 3}\right)$

Strassen's Algorithm

- Generalizes Karatsuba's insight to design a fast algorithm for multiplying two $n \times n$ matrices
  - ➢ Call n the "size" of the problem

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

  - ➢ Naively, this requires 8 multiplications of size $n/2$
    - $A_{11} * B_{11}, A_{12} * B_{21}, A_{11} * B_{12}, A_{12} * B_{22}, \ldots$
  - ➢ Strassen's insight: replace 8 multiplications by 7
    - Running time: $T(n) = 7\,T(n/2) + O(n^2) \Rightarrow T(n) = O\left(n^{\log_2 7}\right)$

Strassen's Algorithm

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

STRASSEN(n, A, B)

IF $(n = 1)$ RETURN $A \times B$.

Partition A and B into 2-by-2 block matrices.

$P_1 \leftarrow$ STRASSEN$(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow$ STRASSEN$(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow$ STRASSEN$(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow$ STRASSEN$(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow$ STRASSEN$(n / 2, (A_{11} + A_{22}) \times (B_{11} + B_{22}))$.

$P_6 \leftarrow$ STRASSEN$(n / 2, (A_{12} - A_{22}) \times (B_{21} + B_{22}))$.

$P_7 \leftarrow$ STRASSEN$(n / 2, (A_{11} - A_{21}) \times (B_{11} + B_{12}))$.

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

RETURN C.

assume n is a power of 2

keep track of indices of submatrices (don't copy matrix entries)

Median & Selection

Selection: Given n comparable elements, find kth smallest.
minimum: $k = 1$; maximum: $k = n$; median: $k = \lfloor (n + 1) / 2 \rfloor$.

- $O(n)$ compares for min or max.
  Can you do better than n-1?
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap.

Applications: order statistics, "top k"; bottleneck paths, …

- Q. Can we do it with $O(n)$ compares?
- A. Yes! Selection is easier than sorting.

Quick (Randomized) Select

Partially sort array relative to a pivot element, and look for the *kth* smallest in subarray to the left or right of pivot.

Look for *kth* smallest in array A[p..r]

QUICK-SELECT (A; p; r; k)

if p == r **return** A[p] // single element array, k must be 1.

q = QUICK-PARTITION(A; p; r) // A[p..q-1] <= A[q] <= A[q+1..r]

j =q-p+1 // k is size of p..q

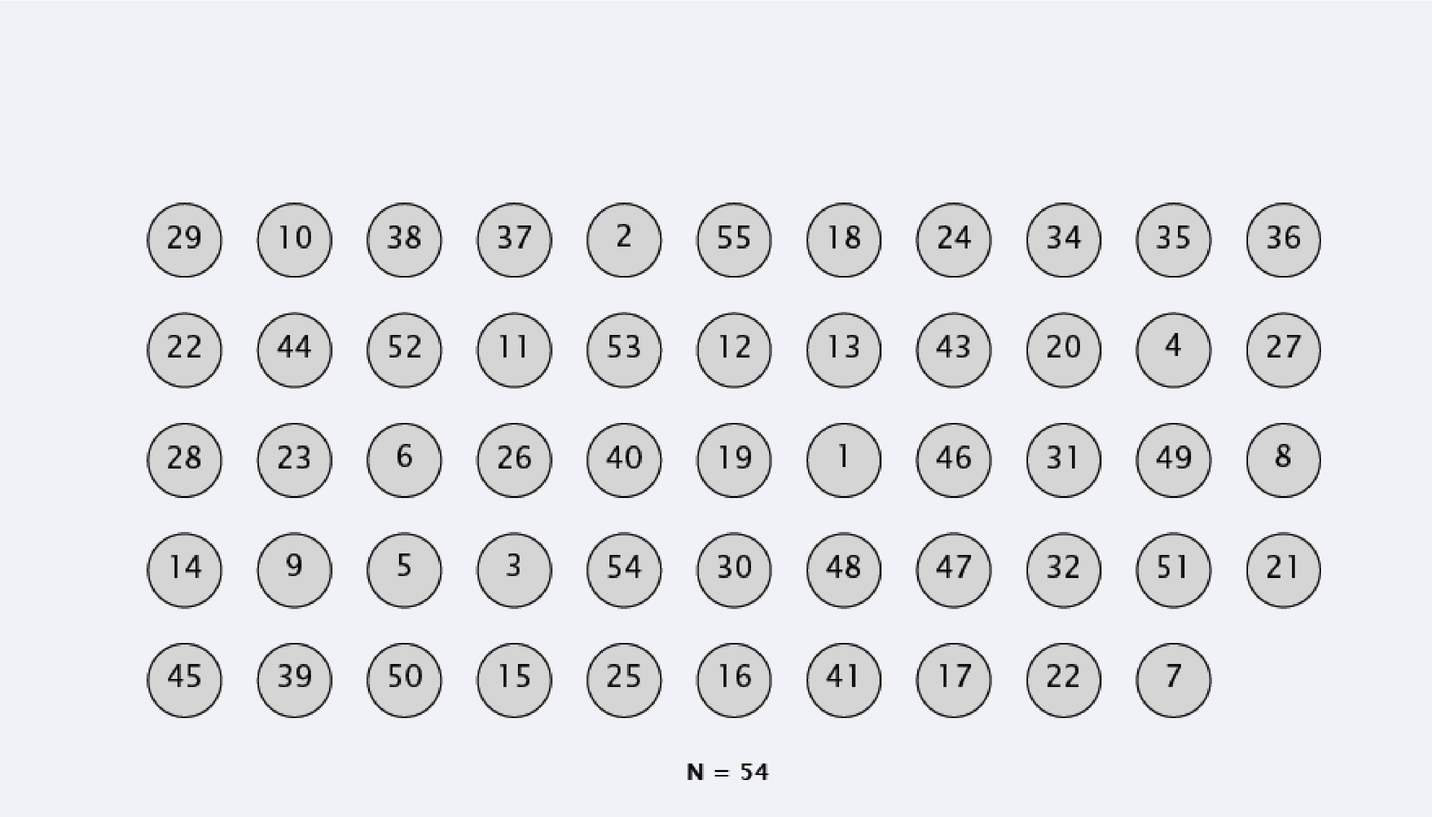
if k == j **return** A[q] // the pivot is kth smallest

elseif k < j **return** QUICK-SELECT(A;p;q-1; k) // search in p..q-1

else return QUICK-SELECT(A;q+1;r;k –j) // search in q+1..r

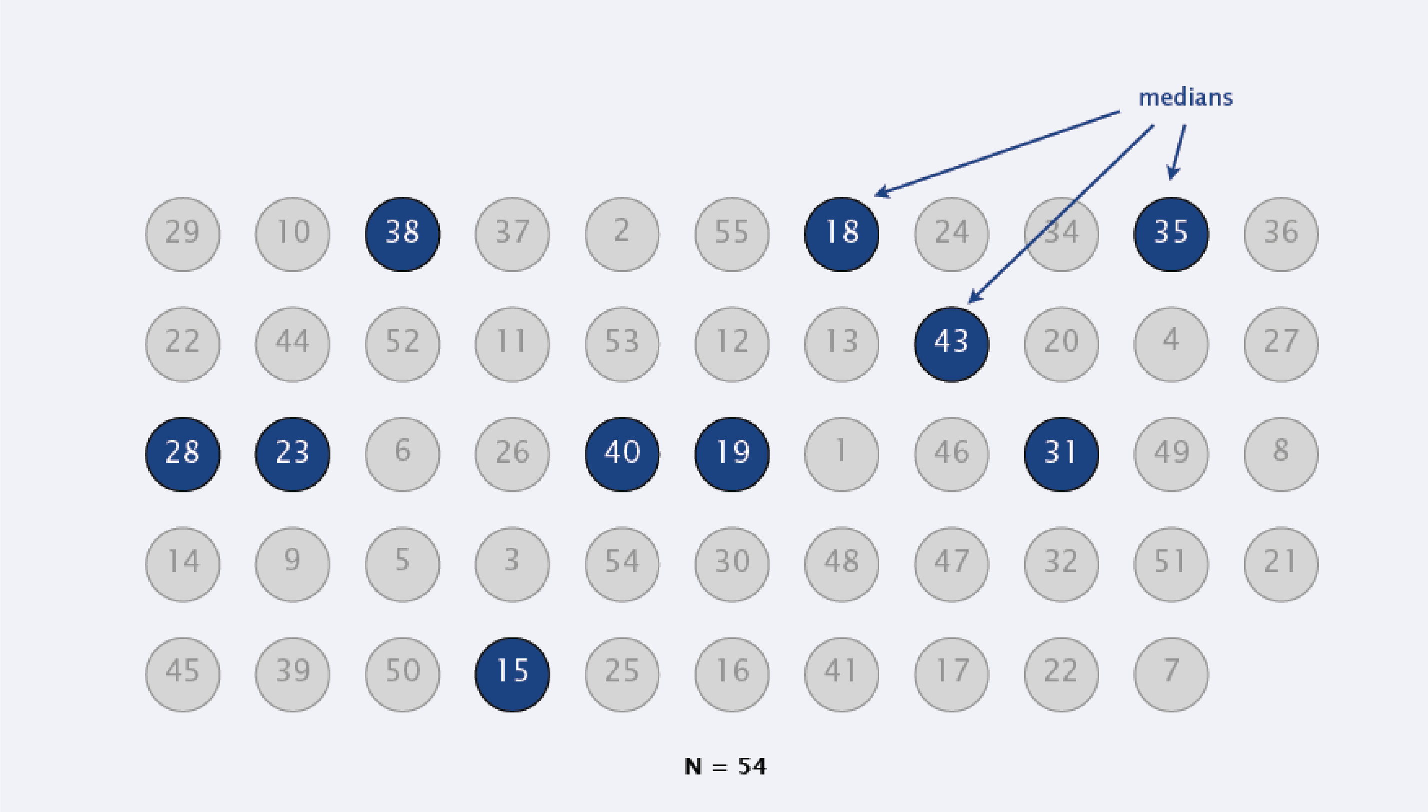
Finding a good pivot

- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).



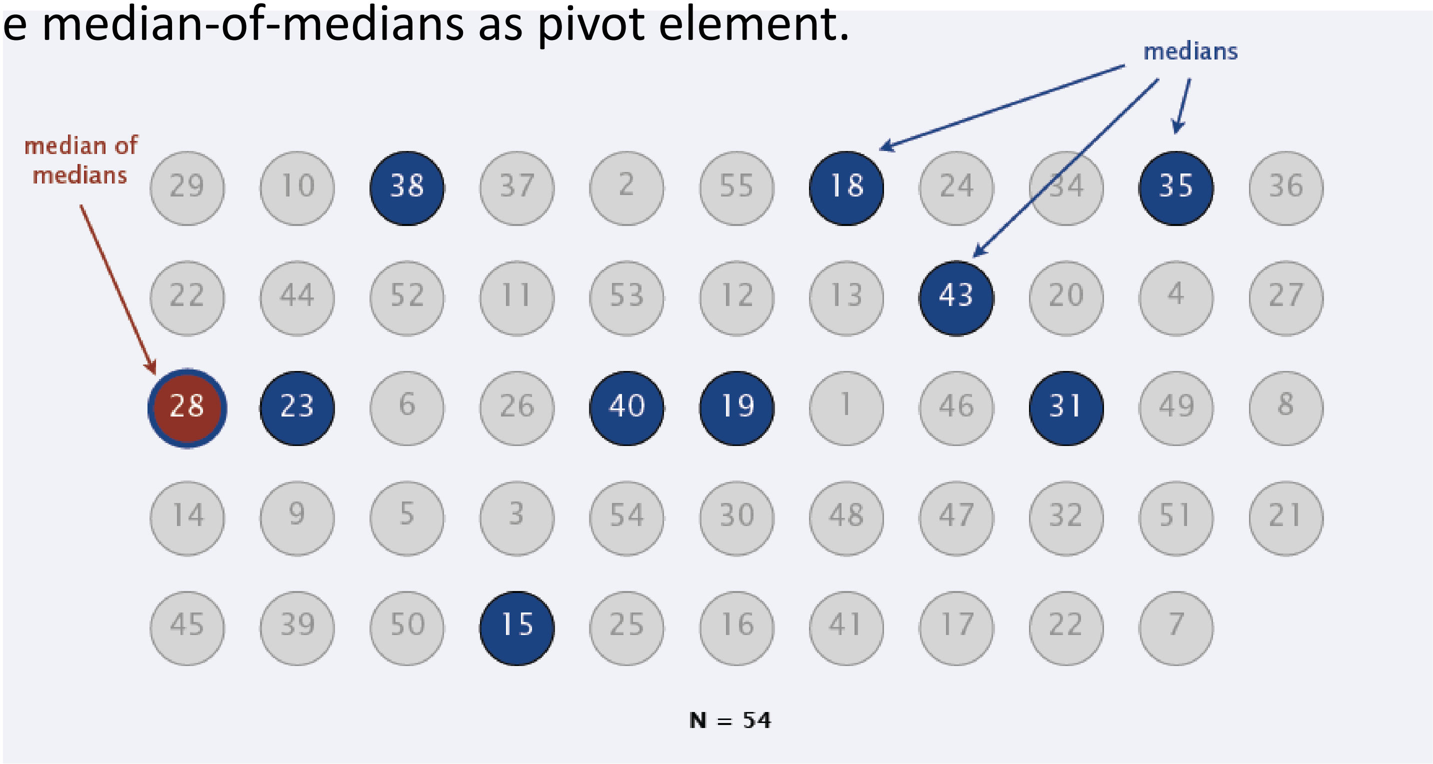
Finding a good pivot

- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).



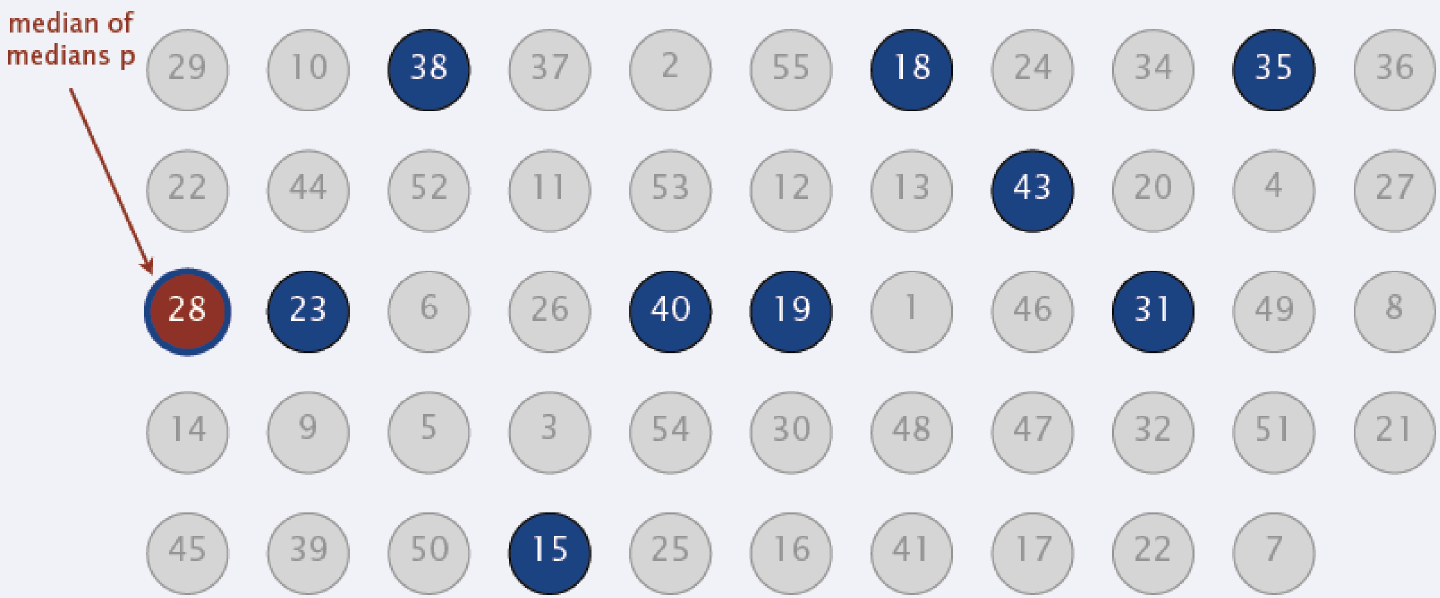
Finding a good pivot

- Divide n elements into $\lfloor n/5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of medians recursively.
- Use median-of-medians as pivot element.



Analysis of median-of-medians selection algorithm

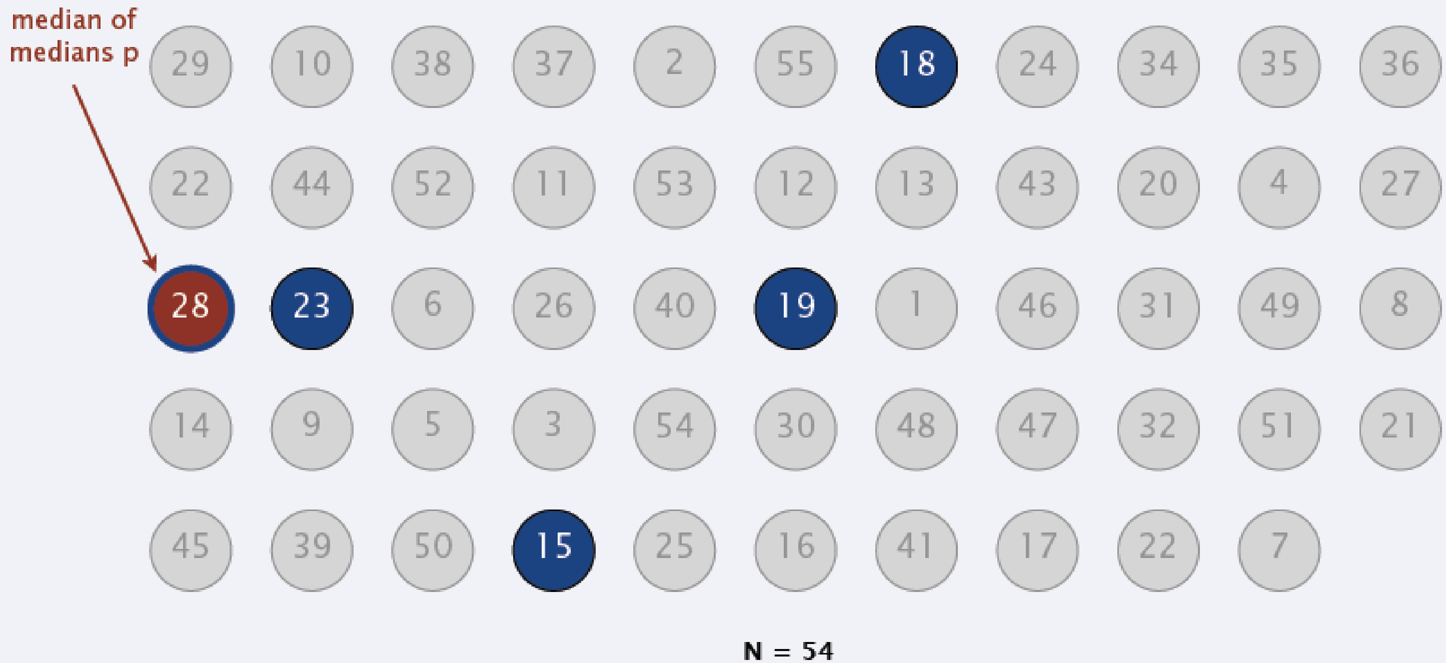
- At least half of 5-element medians $\leq$ p.



N = 54

Analysis of median-of-medians selection algorithm

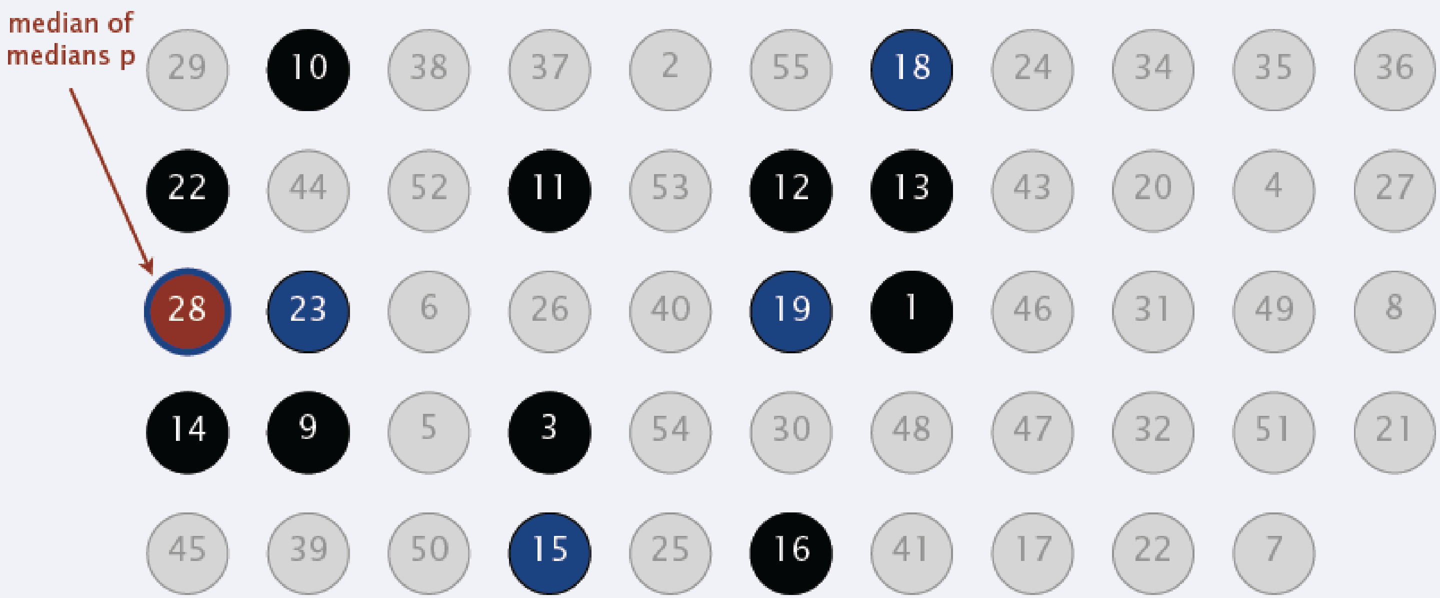
- At least half of 5-element medians $\leq p$.
- At least $\lfloor \lfloor n/5 \rfloor / 2 \rfloor = \lfloor n/10 \rfloor$ medians $\leq p$.



44

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p$.
- At least $\lfloor \lfloor n/5 \rfloor / 2 \rfloor = \lfloor n/10 \rfloor$ medians $\leq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$.



N = 54

45

Median-of-medians recurrence

- Select called recursively with $\lfloor n / 5 \rfloor$ elements to compute MOM p.
- At least $3 \lfloor n / 10 \rfloor$ elements $\leq p$.
- At least $3 \lfloor n / 10 \rfloor$ elements $\geq p$.
- Select called recursively with at most $n - 3 \lfloor n / 10 \rfloor$ elements.

Def. $C(n)$ = max # compares on an array of n elements.

$$C(n) \leq \underbrace{C(\lfloor n/5 \rfloor)}_{\text{median of medians}} + \underbrace{C(n - 3\lfloor n/10 \rfloor)}_{\text{recursive select}} + \underbrace{\tfrac{11}{5}\, n}_{\substack{\text{computing median of 5 (6 compares per group)} \\ \text{partitioning (n compares)}}}$$

- O(n), 44n works!

Algorithm Design

- Best algorithm for a problem?
  - Typically hard to determine
  - We still don't know best algorithms for multiplying two n-digit integers or two $n \times n$ matrices
    - Integer multiplication
      - Breakthrough in March 2019: first $O(n \log n)$ time algorithm
      - It is conjectured that this is asymptotically optimal
    - Matrix multiplication
      - 1969 (Strassen): $O(n^{2.807})$
      - 1990: $O(n^{2.376})$
      - 2013: $O(n^{2.3729})$
      - 2014: $O(n^{2.3728639})$

Algorithm Design

- Best algorithm for a problem?
  - Usually, we design an algorithm and then analyze its running time

  - Sometimes we can do the reverse:
    - E.g., if you know you want an $O(n^2 \log n)$ algorithm
    - Master theorem suggests that you can get it by
$$T(n) = 4\,T\left({}^{n}/_{2}\right) + O(n^2)$$
    - So maybe you want to break your problem into 4 problems of size $n/2$ each, and then do $O(n^2)$ computation to combine

Algorithm Design

- Access to input
  - For much of this analysis, we are assuming random access to elements of input
  - So we’re ignoring underlying data structures (e.g. doubly linked list, binary tree, etc.)

- Machine operations
  - We’re only counting comparison or arithmetic operations
  - So we’re ignoring issues like how real numbers will be represented in closest pair problem
  - When we get to P vs NP, representation will matter

Algorithm Design

- Size of the problem
  - Can be any reasonable parameter of the problem
  - E.g., for matrix multiplication, we used n as the size
    But an input consists of two matrices with n^2 entries
  - It doesn't matter whether we call n or n^2 the size of the problem
  - The actual running time of the algorithm won't change