



5. DIVIDE AND CONQUER I

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *closest pair of points*
- ▶ *randomized quicksort*
- ▶ *median and selection*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size n into **two** subproblems (of the same kind) of size $n/2$ in **linear time**.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in **linear time**.

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $\Theta(n \log n)$.



attributed to Julius Caesar



5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *closest pair of points*
- ▶ *randomized quicksort*
- ▶ *median and selection*

Sorting problem

Problem. Given a list of n elements from a totally-ordered universe, rearrange them in ascending order.



	Name	Artist	Time	Album
12	<input checked="" type="checkbox"/> Let It Be	The Beatles	4:03	Let It Be
13	<input checked="" type="checkbox"/> Take My Breath Away	BERLIN	4:13	Top Gun – Soundtrack
14	<input checked="" type="checkbox"/> Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	<input checked="" type="checkbox"/> Dancing With Myself	Billy Idol	4:43	Don't Stop
16	<input checked="" type="checkbox"/> Rebel Yell	Billy Idol	4:49	Rebel Yell
17	<input checked="" type="checkbox"/> Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	<input checked="" type="checkbox"/> Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
19	<input checked="" type="checkbox"/> The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
20	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	<input checked="" type="checkbox"/> Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	<input checked="" type="checkbox"/> Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	<input checked="" type="checkbox"/> Hurricane	Bob Dylan	8:32	Desire
25	<input checked="" type="checkbox"/> The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	<input checked="" type="checkbox"/> Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
28	<input checked="" type="checkbox"/> Runaway	Bon Jovi	3:53	Cross Road
29	<input checked="" type="checkbox"/> Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	<input checked="" type="checkbox"/> Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	<input checked="" type="checkbox"/> Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	<input checked="" type="checkbox"/> Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	<input checked="" type="checkbox"/> Holding Out For A Hero	Bonny Tyler	5:49	Meat Loaf And Friends
34	<input checked="" type="checkbox"/> Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	<input checked="" type="checkbox"/> Thunder Road	Bruce Springsteen	4:51	Born To Run
36	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
37	<input checked="" type="checkbox"/> Jungleland	Bruce Springsteen	9:34	Born To Run
38	<input checked="" type="checkbox"/> Tunnel Tunnel Tunnel (To Eurothis)	The Buds	3:57	Forest Gump The Soundtrack (Disc 2)

Sorting applications

Obvious applications.

- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.

- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

Non-obvious applications.

- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Minimum spanning trees (Kruskal's algorithm).
- Scheduling to minimize maximum lateness or average completion time.
- ...

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.

input

A	L	G	O	R	I	T	H	M	S
---	---	---	---	---	---	---	---	---	---

sort left half

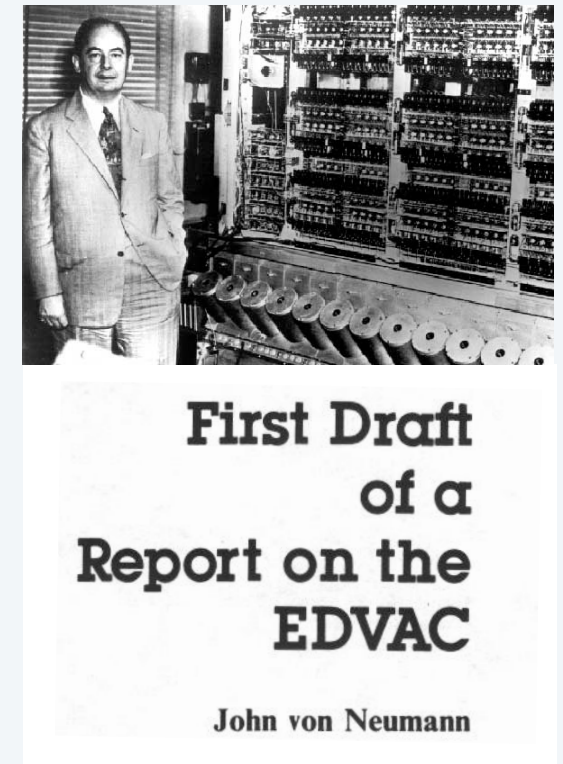
A	G	L	O	R	I	T	H	M	S
---	---	---	---	---	---	---	---	---	---

sort right half

A	G	L	O	R	H	I	M	S	T
---	---	---	---	---	---	---	---	---	---

merge results

A	G	H	I	L	M	O	R	S	T
---	---	---	---	---	---	---	---	---	---



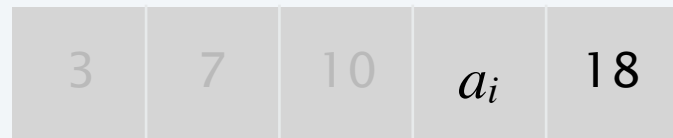
Merging

Goal. Combine two sorted lists A and B into a sorted whole C .

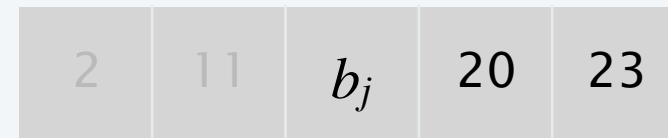


- Scan A and B from left to right.
- Compare a_i and b_j .
- If $a_i \leq b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).

sorted list A



sorted list B



merge to form sorted list C



A useful recurrence relation

Def. $T(n)$ = max number of compares to mergesort a list of size $\leq n$.

Note. $T(n)$ is monotone nondecreasing.

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

Solution. $T(n)$ is $O(n \log_2 n)$.

Assorted proofs. We describe several ways to solve this recurrence.

Initially we assume n is a power of 2 and replace \leq with $=$ in the recurrence.

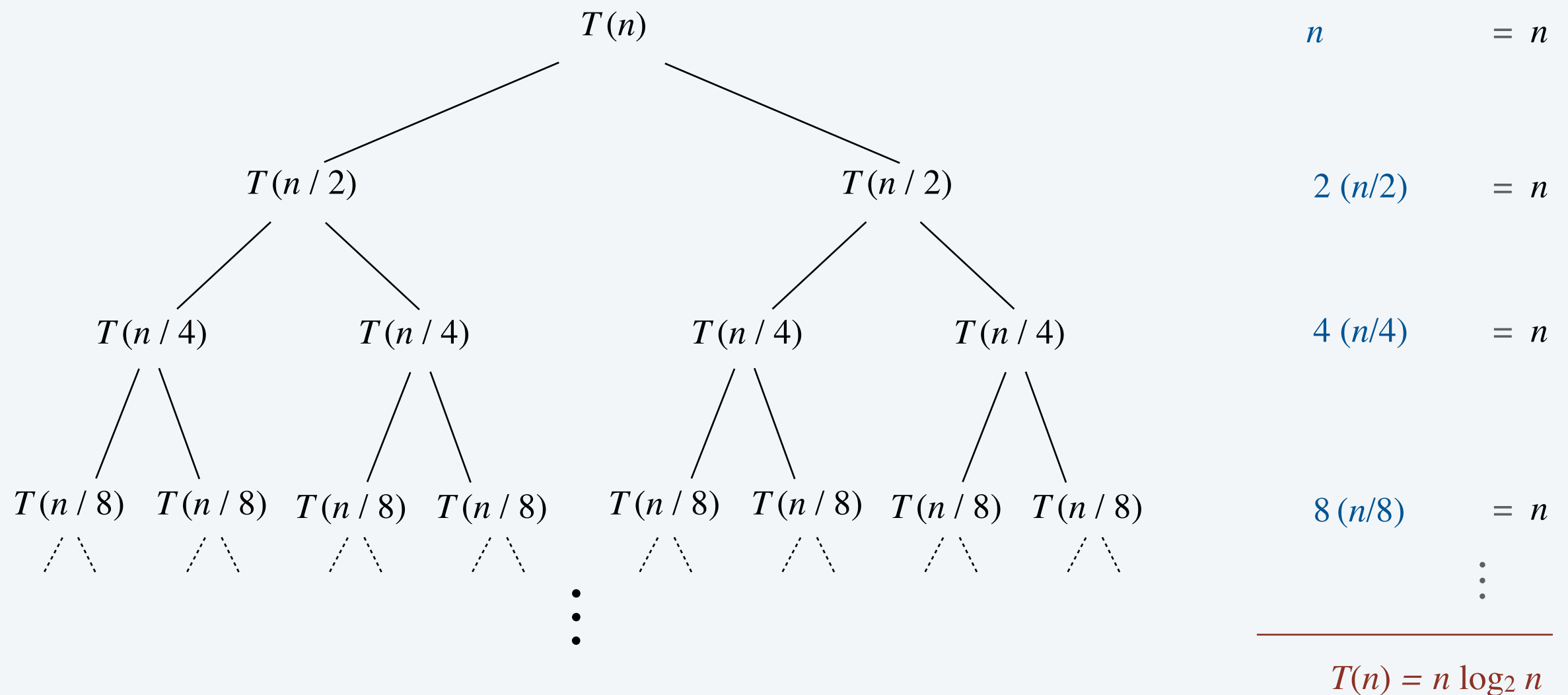
Divide-and-conquer recurrence: proof by recursion tree

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2 T(n / 2) + n & \text{otherwise} \end{cases}$$

assuming n
is a power of 2

Pf 1.



Proof by induction

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2 T(n/2) + n & \text{otherwise} \end{cases}$$

assuming n
is a power of 2

Pf 2. [by induction on n]

- Base case: when $n = 1$, $T(1) = 0 = n \log_2 n$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) &= 2 T(n) + 2n \\ &= 2 n \log_2 n + 2n \\ &= 2 n (\log_2 (2n) - 1) + 2n \\ &= 2 n \log_2 (2n). \quad \blacksquare \end{aligned}$$

Analysis of mergesort recurrence

Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \log_2 n \rceil$.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

Pf. [by strong induction on n]

- Base case: $n = 1$.
- Define $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$.
- Induction step: assume true for $1, 2, \dots, n-1$.

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n & n_2 &= \lceil n/2 \rceil \\ &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n & &\leq \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\ &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n & &= 2^{\lceil \log_2 n \rceil} / 2 \\ &= n \lceil \log_2 n_2 \rceil + n & \longleftarrow & \log_2 n_2 \leq \lceil \log_2 n \rceil - 1 \\ &\leq n (\lceil \log_2 n \rceil - 1) + n \\ &= n \lceil \log_2 n \rceil. \quad \blacksquare \end{aligned}$$



5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ ***counting inversions***
- ▶ *closest pair of points*
- ▶ *randomized quicksort*
- ▶ *median and selection*

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of **inversions** between two rankings.

- My rank: $1, 2, \dots, n$.
- Your rank: a_1, a_2, \dots, a_n .
- Songs i and j are inverted if $i < j$, but $a_i > a_j$.

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 inversions: 3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs.

Counting inversions: 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).

Rank Aggregation Methods for the Web

Cynthia Dwork*

Ravi Kumar†

Moni Naor‡

D. Sivakumar§

ABSTRACT

We consider the problem of combining ranking results from various sources. In the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. We develop a set of techniques for the rank aggregation problem and compare their performance to that of well-known methods. A primary goal of our work is to design rank aggregation techniques that can effectively combat “spam,” a serious problem in Web searches. Experiments show that our methods are simple, efficient, and effective.

Keywords: rank aggregation, ranking functions, meta-search, multi-word queries, spam

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B .
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with $a \in A$ and $b \in B$.
- Return sum of three counts.

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 $a \in A$ and $b \in B$

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: how to combine two subproblems?

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

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B .
- For each element $b \in B$,
 - binary search in A to find how elements in A are greater than b .

list A

7	10	18	3	14
---	----	----	---	----

list B

20	23	2	11	16
----	----	---	----	----

sort A

3	7	10	14	18
---	---	----	----	----

sort B

2	11	16	20	23
---	----	----	----	----

binary search to count inversions (a, b) with $a \in A$ and $b \in B$

3	7	10	14	18
---	---	----	----	----

2	11	16	20	23
---	----	----	----	----

5 2 1 0 0

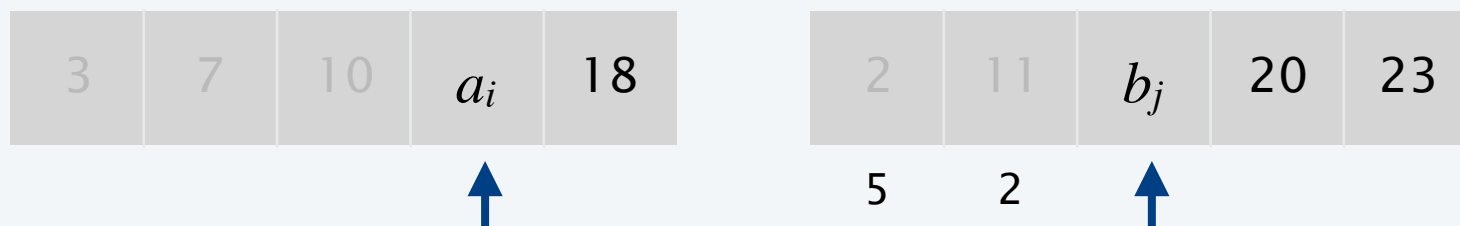
Counting inversions: how to combine two subproblems?

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

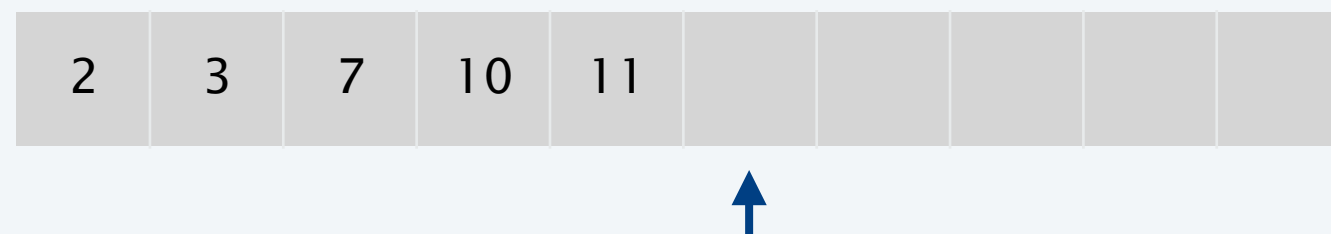
- Scan A and B from left to right.
- Compare a_i and b_j .
- If $a_i < b_j$, then a_i is not inverted with any element left in B .
- If $a_i > b_j$, then b_j is inverted with every element left in A .
- Append smaller element to sorted list C .



count inversions (a, b) with $a \in A$ and $b \in B$



merge to form sorted list C



Counting inversions: divide-and-conquer algorithm implementation

Input. List L .

Output. Number of inversions in L **and** sorted list of elements L' .

***SORT-AND-COUNT** (L)*

IF list L has one element

RETURN $(0, L)$.

DIVIDE the list into two halves A and B .

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

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

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

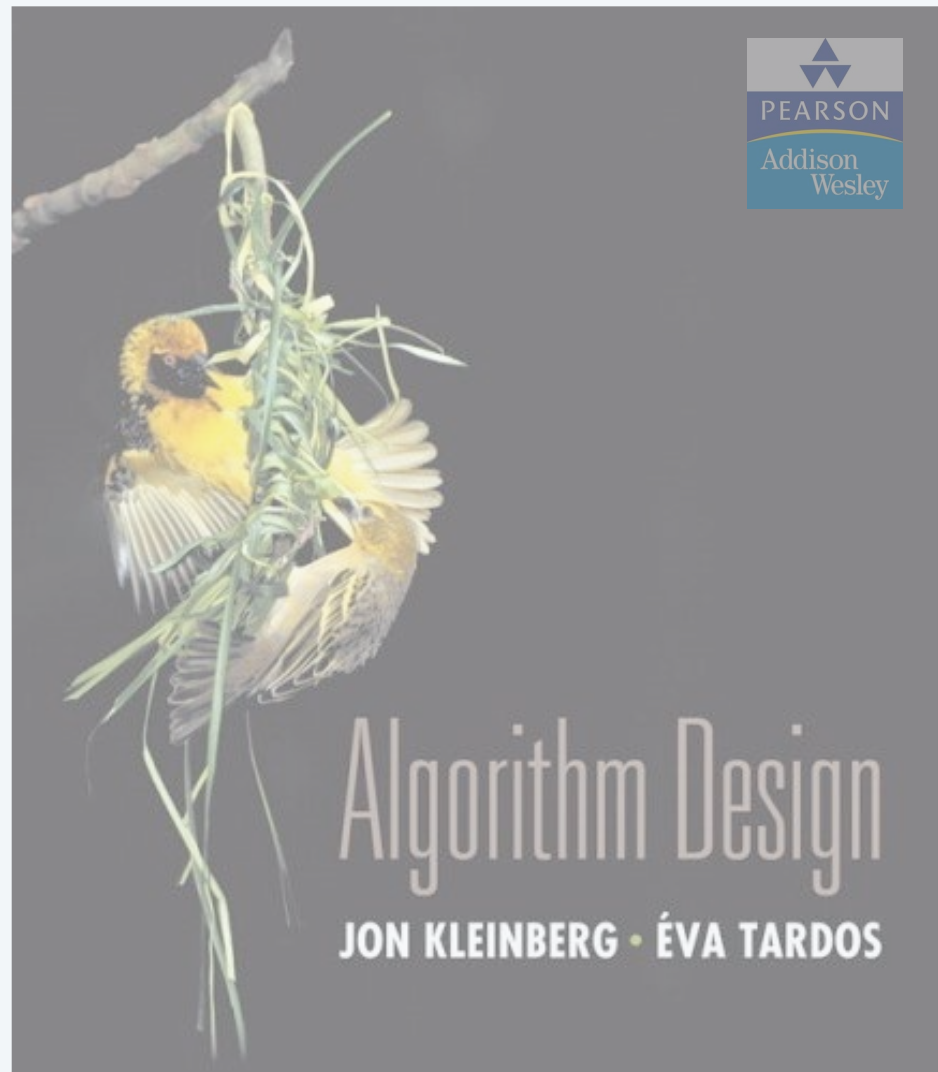
RETURN $(r_A + r_B + r_{AB}, L')$.

Counting inversions: divide-and-conquer algorithm analysis

Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Pf. The worst-case running time $T(n)$ satisfies the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{otherwise} \end{cases}$$



5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ ***closest pair of points***
- ▶ *randomized quicksort*
- ▶ *median and selection*

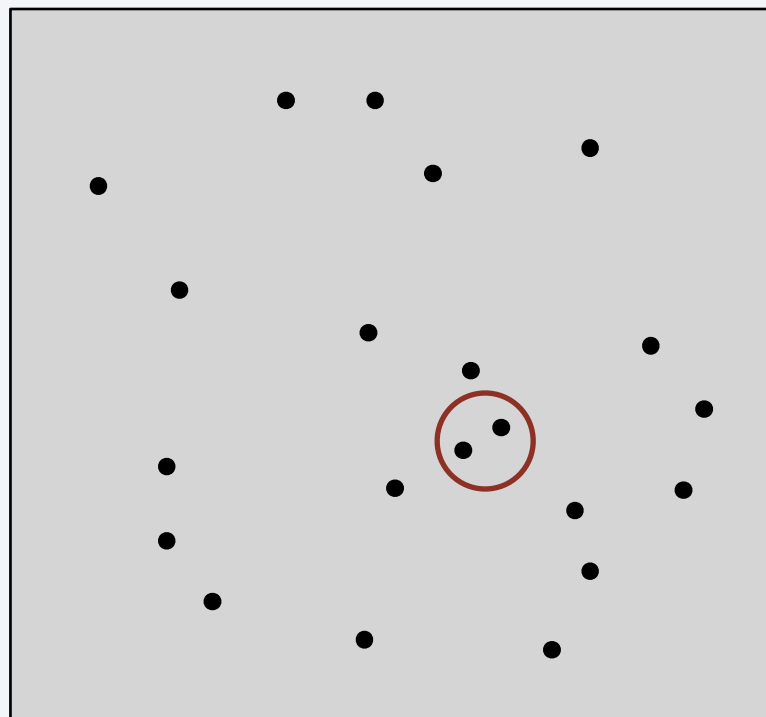
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems



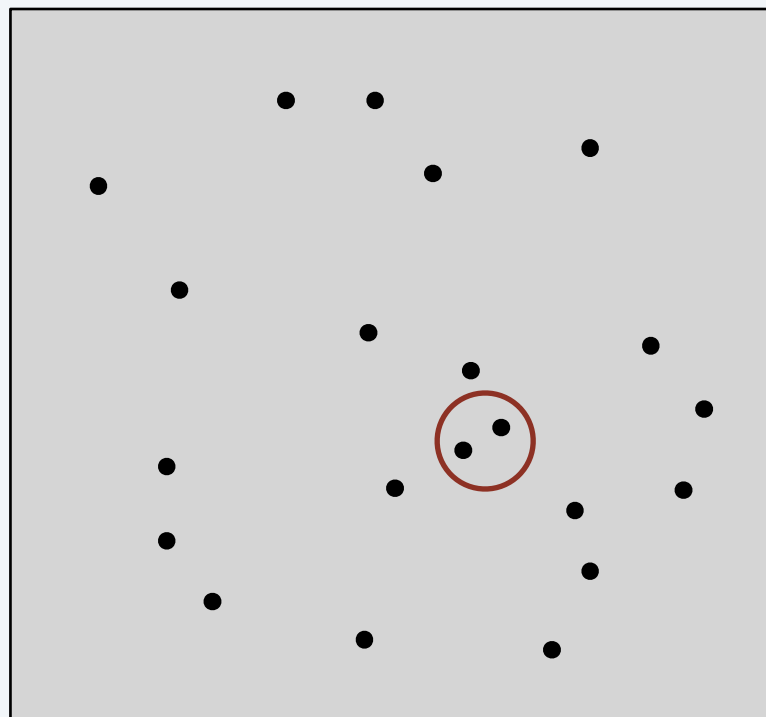
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

1d version. Easy $O(n \log n)$ algorithm if points are on a line.

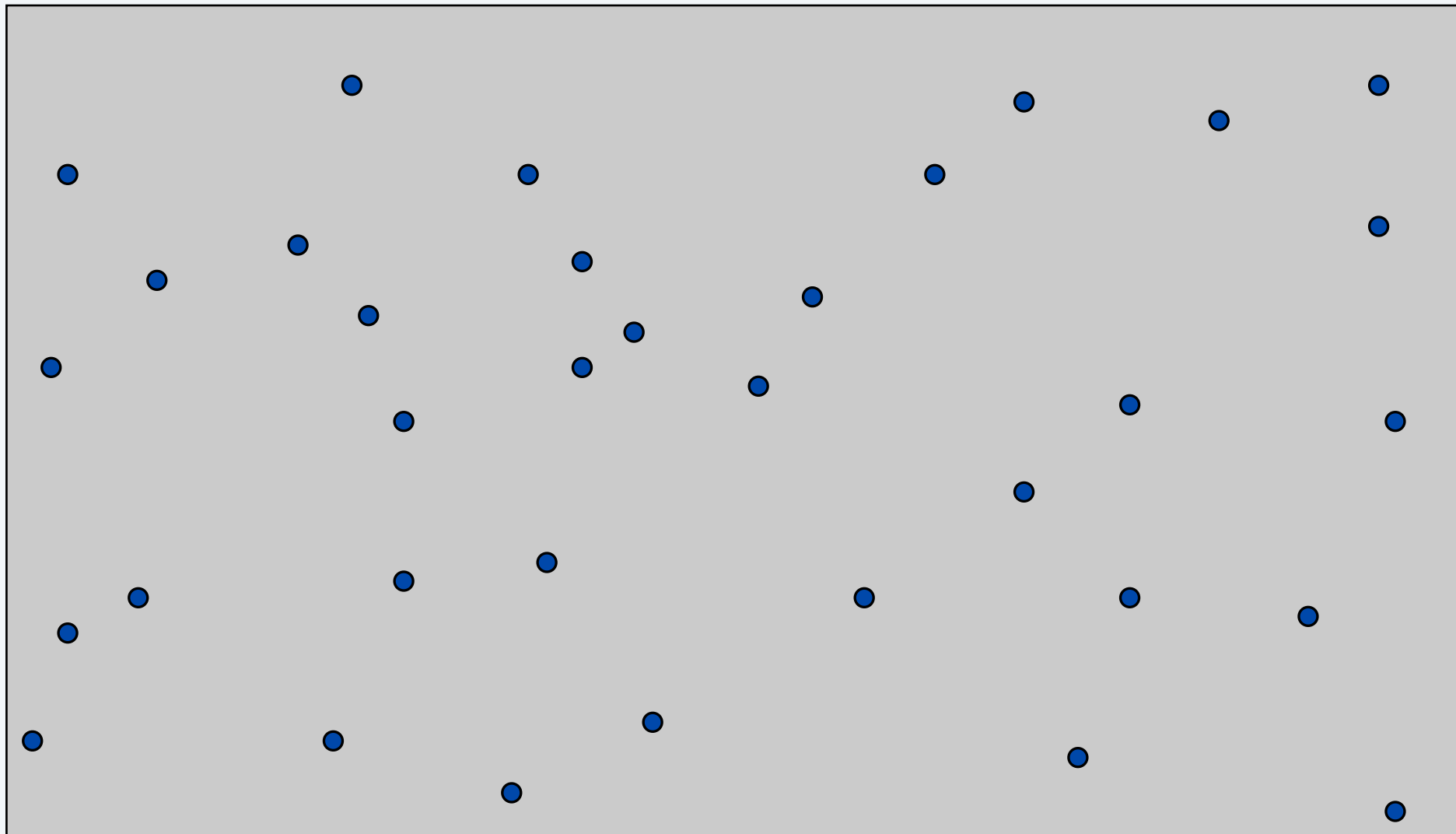
Nondegeneracy assumption. No two points have the same x -coordinate.



Closest pair of points: first attempt

Sorting solution.

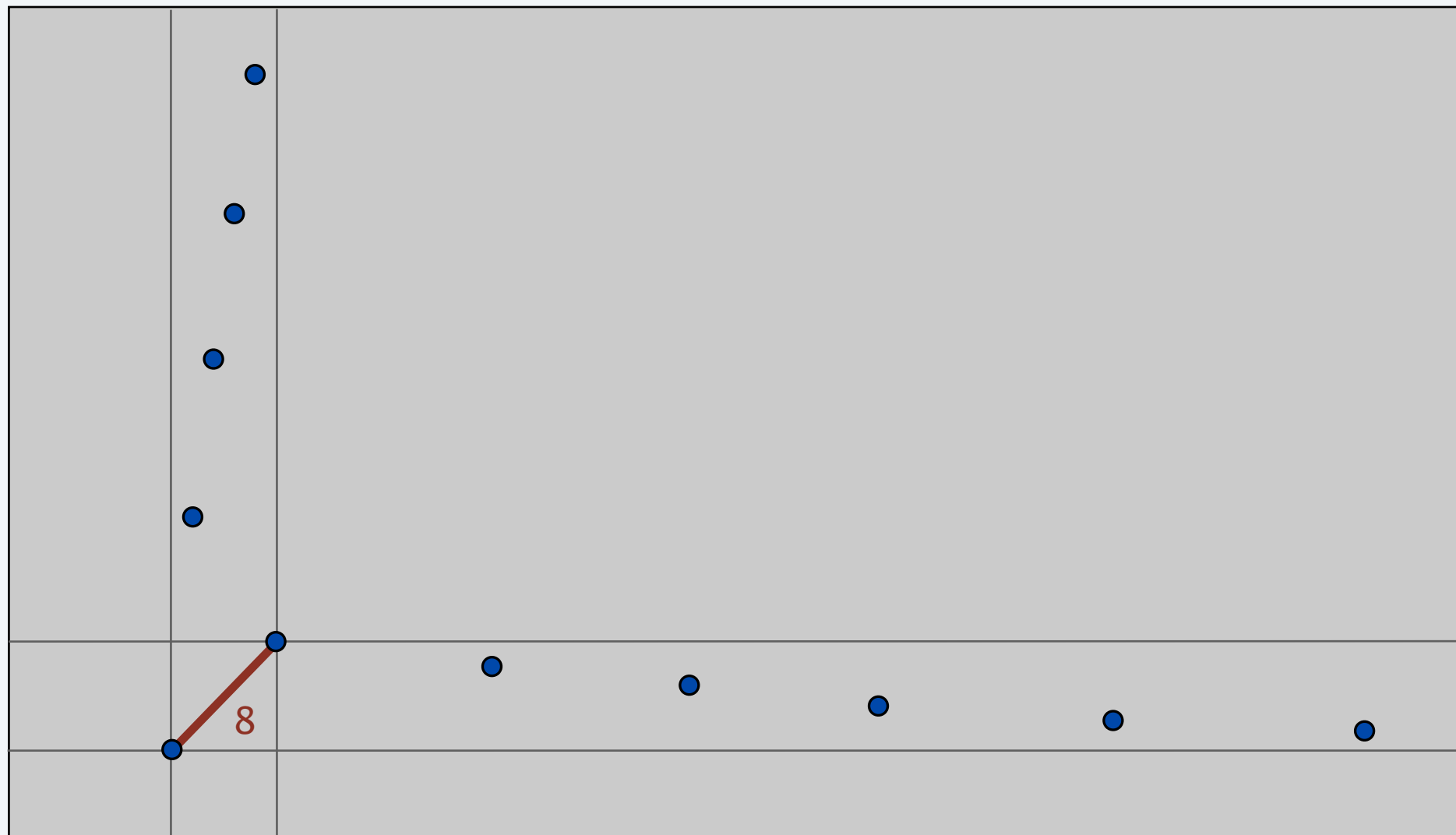
- Sort by x -coordinate and consider nearby points.
- Sort by y -coordinate and consider nearby points.



Closest pair of points: first attempt

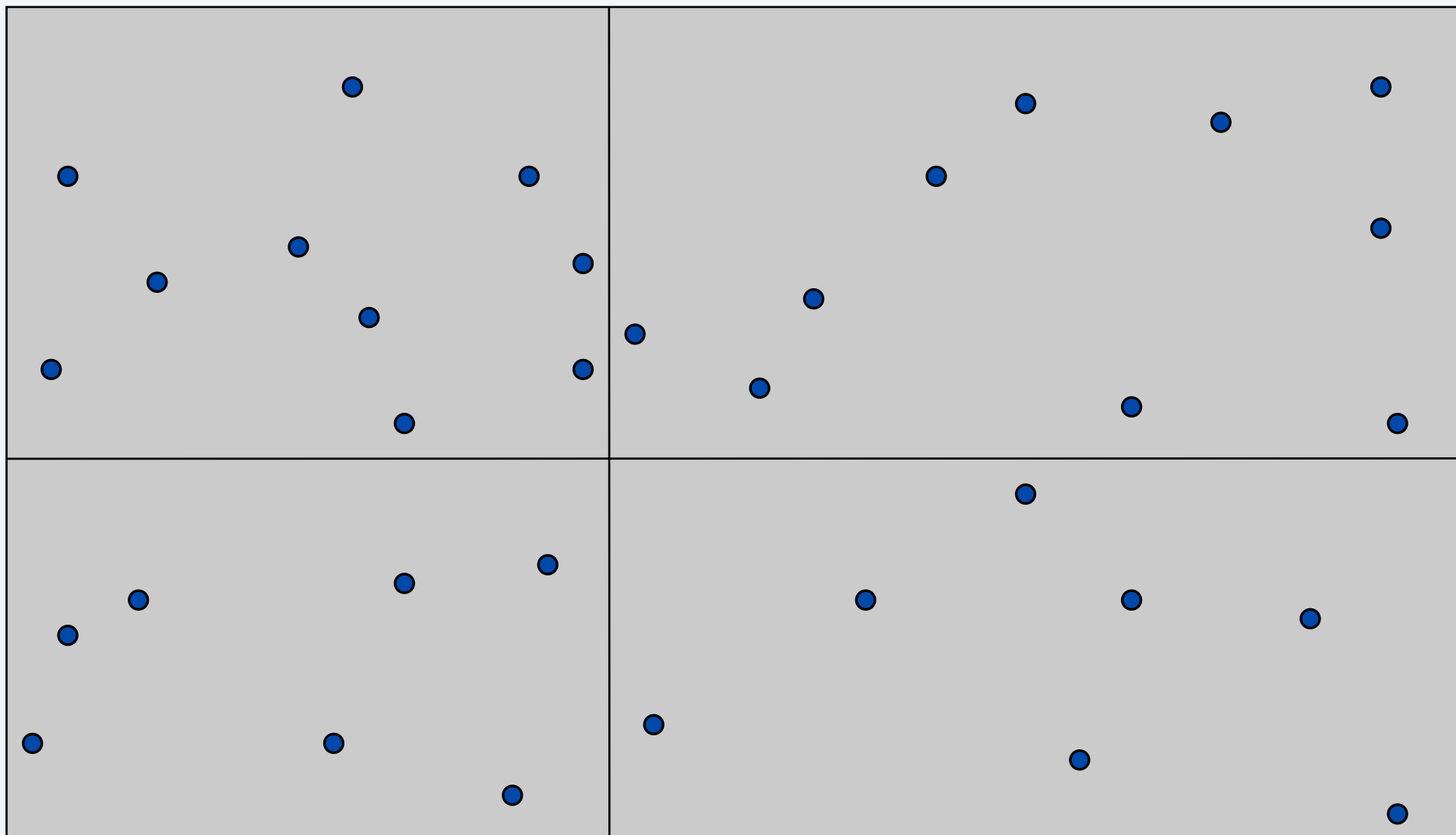
Sorting solution.

- Sort by x -coordinate and consider nearby points.
- Sort by y -coordinate and consider nearby points.



Closest pair of points: second attempt

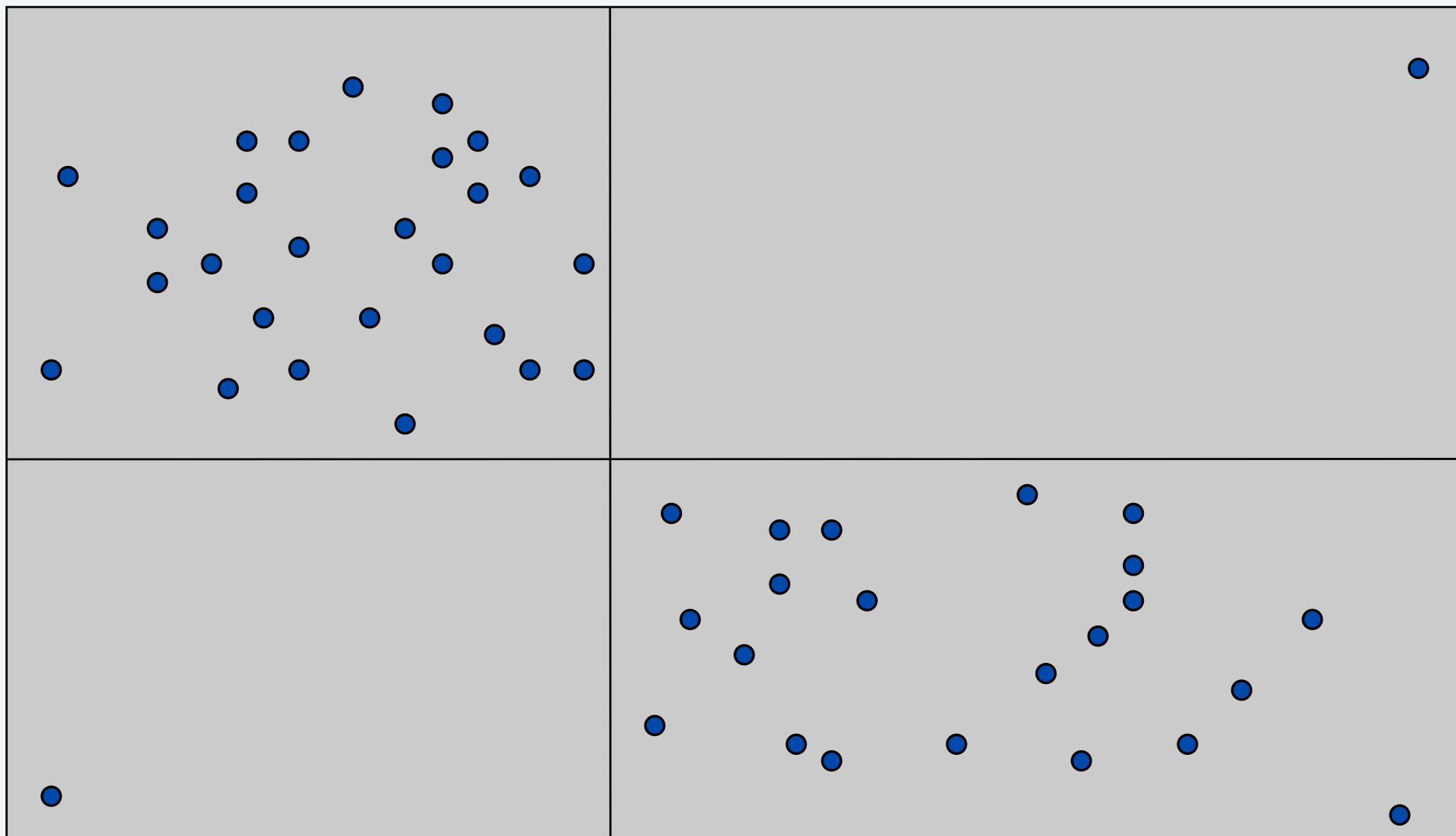
Divide. Subdivide region into 4 quadrants.



Closest pair of points: second attempt

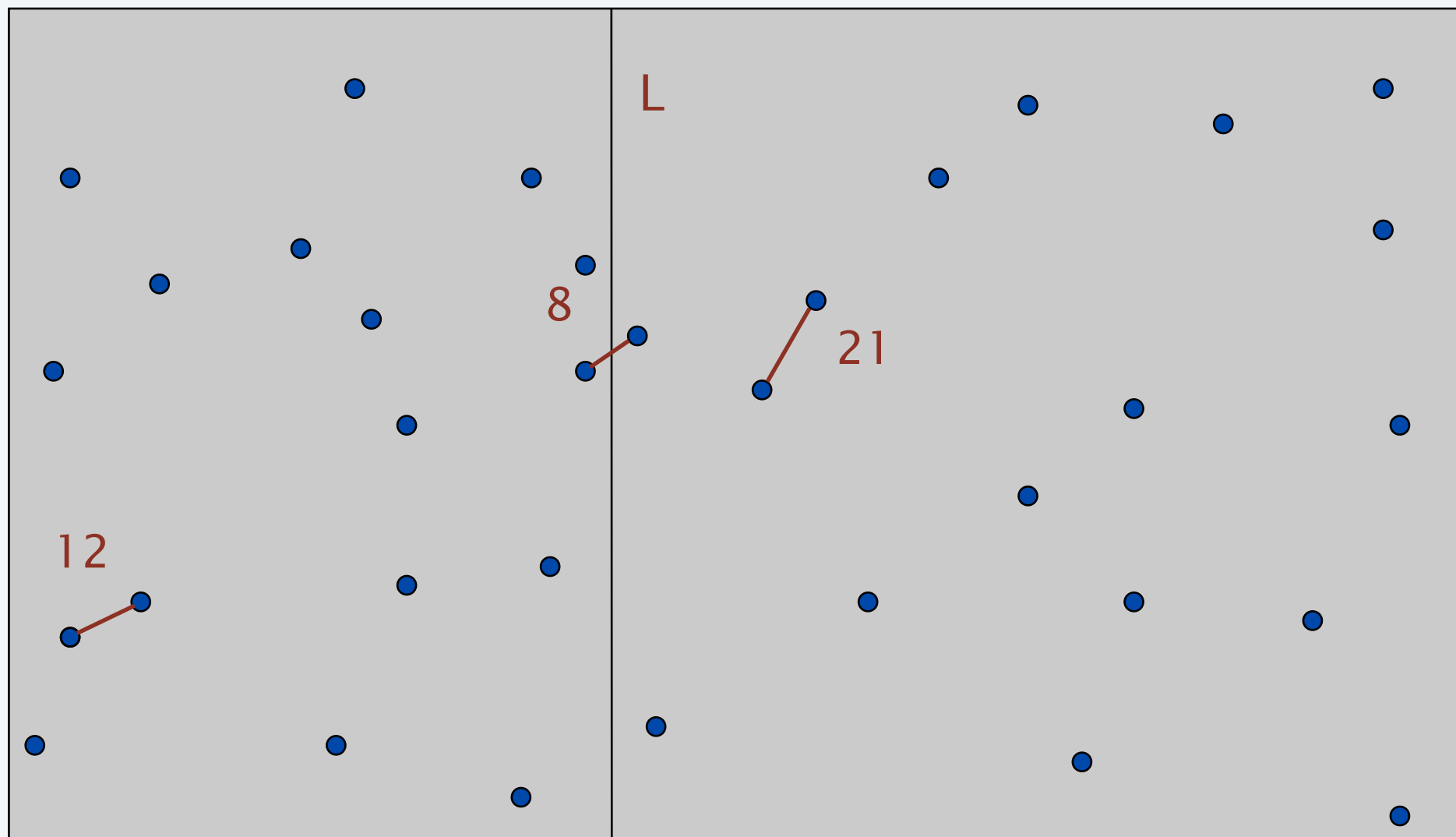
Divide. Subdivide region into 4 quadrants.

Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest pair of points: divide-and-conquer algorithm

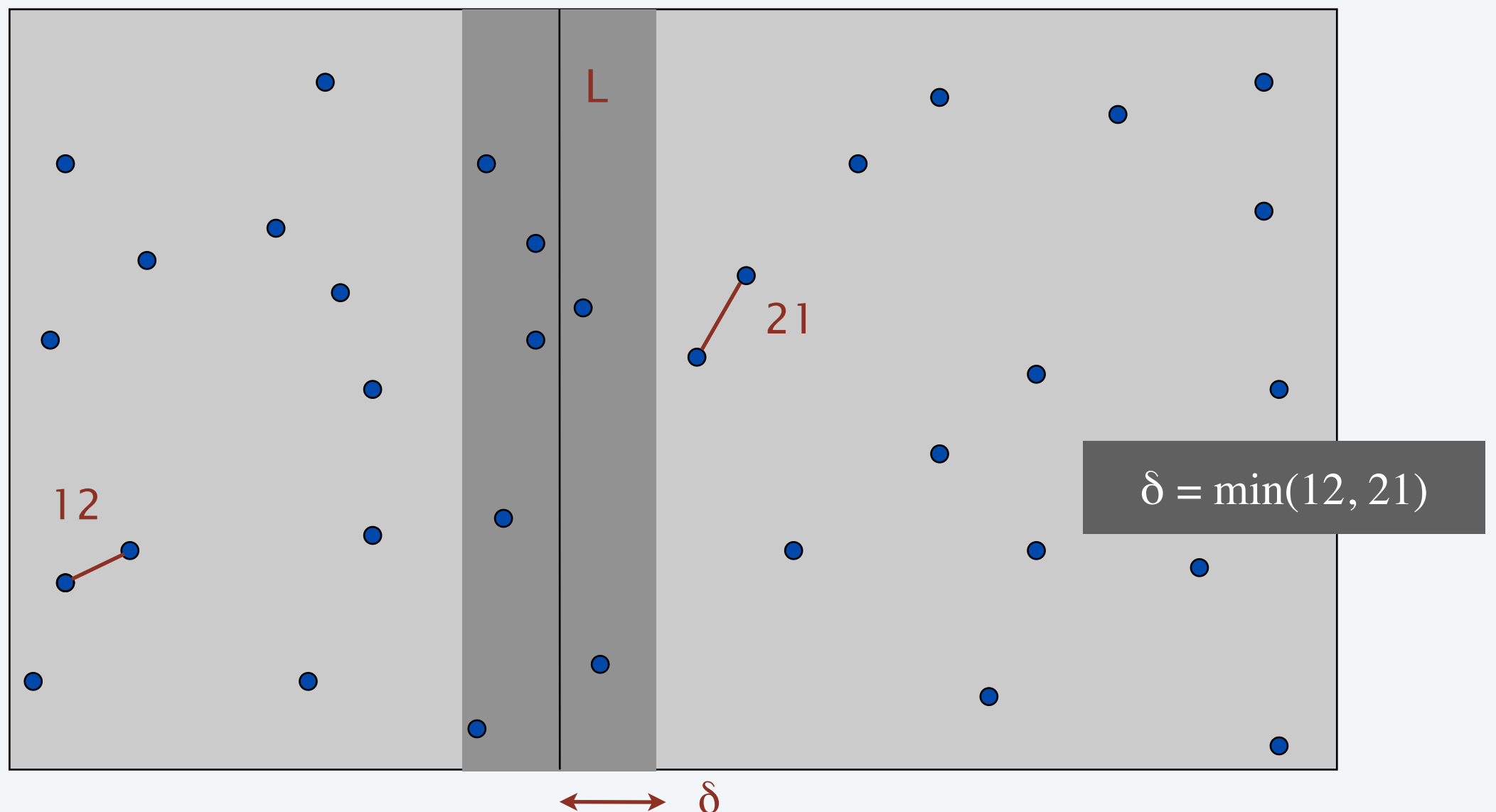
- Divide: draw vertical line L so that $n/2$ points on each side.
 - Conquer: find closest pair in each side recursively.
 - **Combine:** find closest pair with one point in each side.
 - Return best of 3 solutions.
- seems like $\Theta(N^2)$



How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: only need to consider points within δ of line L .

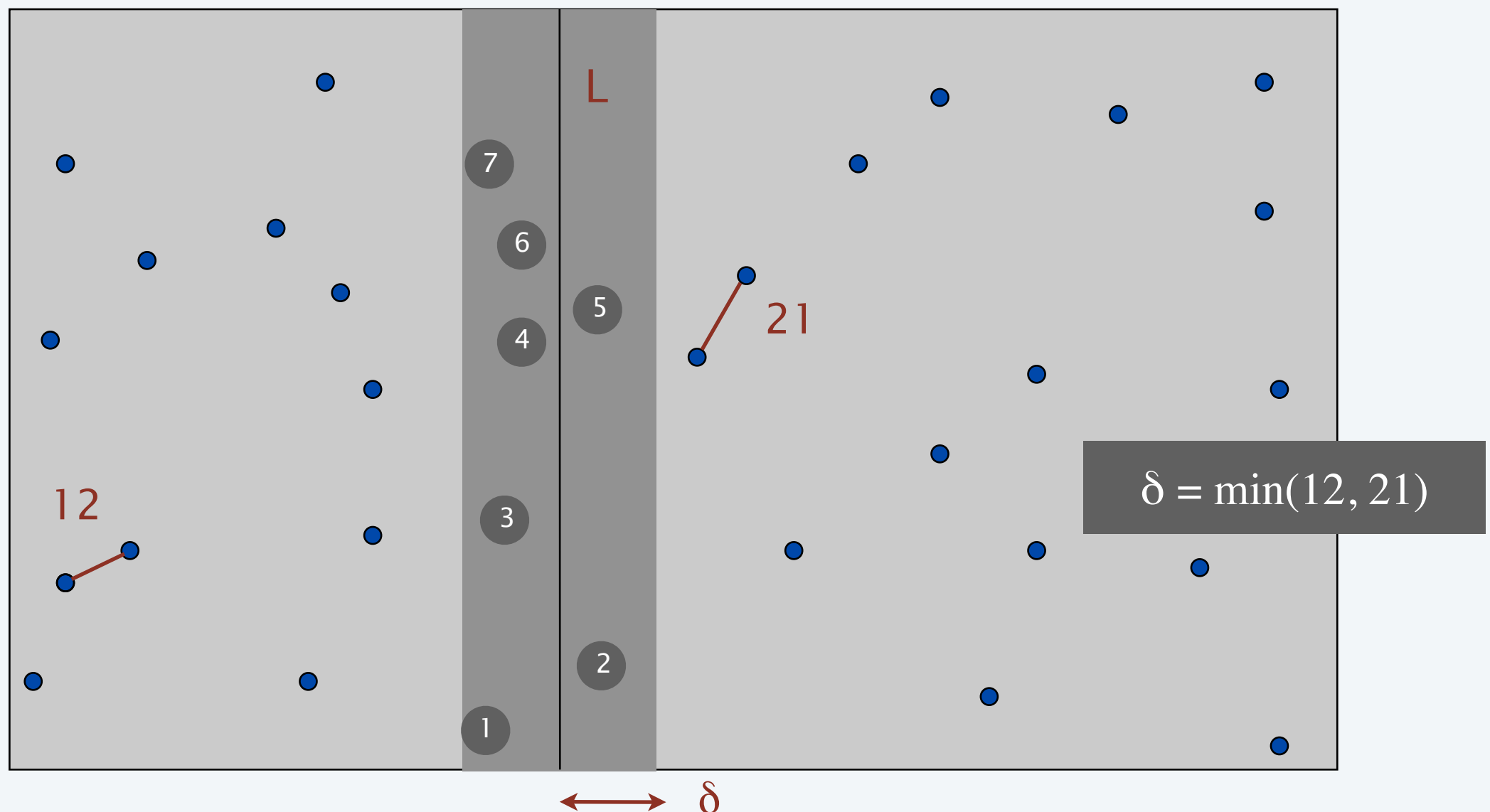


How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y -coordinate.
- Only check distances of those within 11 positions in sorted list!

why 11?



How to find closest pair with one point in each side?

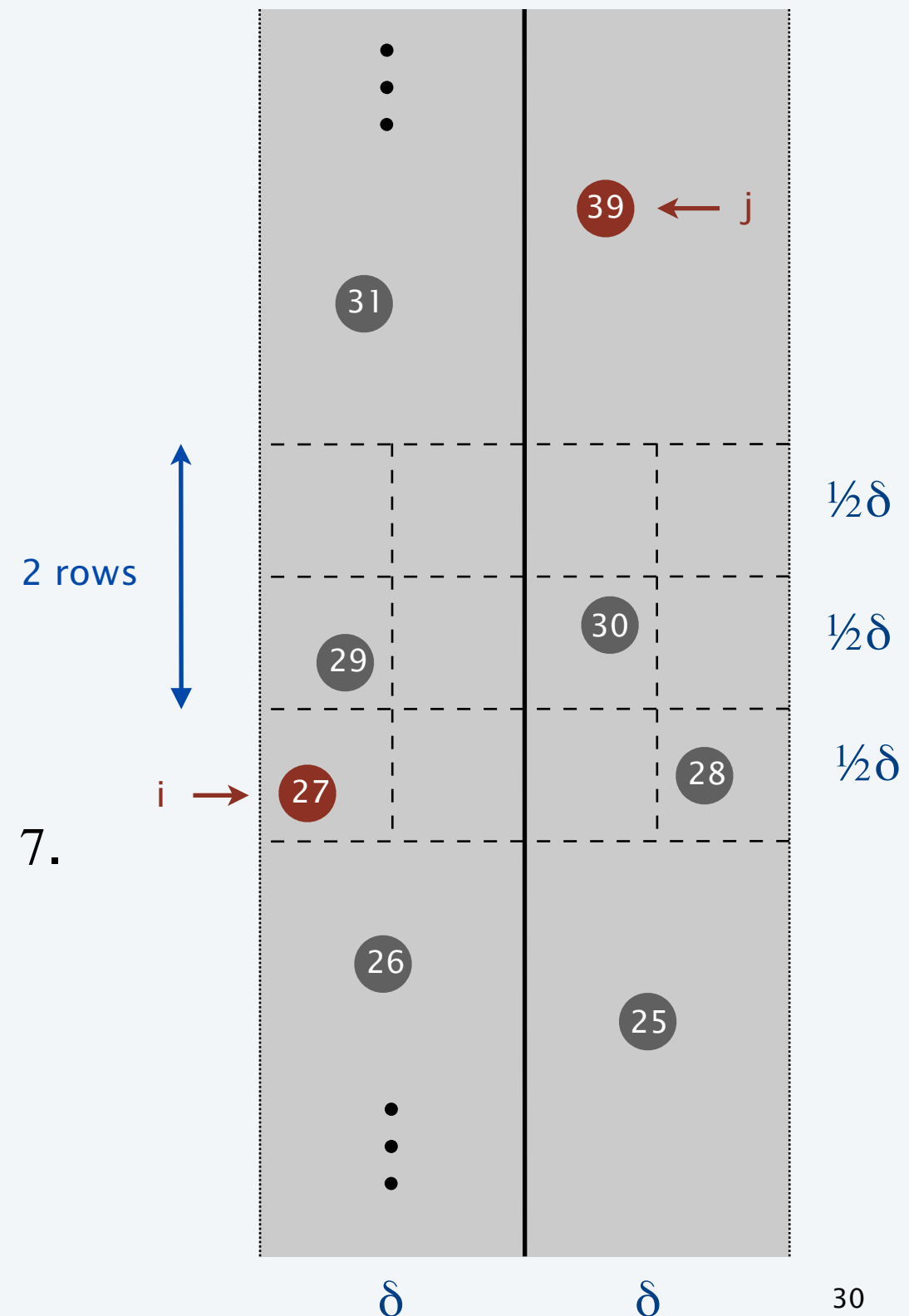
Def. Let s_i be the point in the 2δ -strip, with the i^{th} smallest y -coordinate.

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ .

Pf.

- No two points lie in same $\frac{1}{2}\delta$ -by- $\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ■

Fact. Claim remains true if we replace 12 with 7.



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR (p_1, p_2, \dots, p_n)

Compute separation line L such that half the points are on each side of the line.

$\delta_1 \leftarrow$ **CLOSEST-PAIR** (points in left half).

$\delta_2 \leftarrow$ **CLOSEST-PAIR** (points in right half).

$\delta \leftarrow \min \{ \delta_1, \delta_2 \}$.

Delete all points further than δ from line L .

Sort remaining points by y -coordinate.

Scan points in y -order and compare distance between each point and next 11 neighbors. If any of these distances is less than δ , update δ .

RETURN δ .

← $O(n \log n)$

← $2 T(n / 2)$

← $O(n)$


← $O(n \log n)$

← $O(n)$

Closest pair of points: analysis

Theorem. The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log^2 n)$ time.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n) & \text{otherwise} \end{cases}$$


$$(x_1 - x_2)^2 + (y_1 - y_2)^2$$

Lower bound. In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.

Improved closest pair algorithm

Q. How to improve to $O(n \log n)$?


A. Yes. Don't sort points in strip from scratch each time.

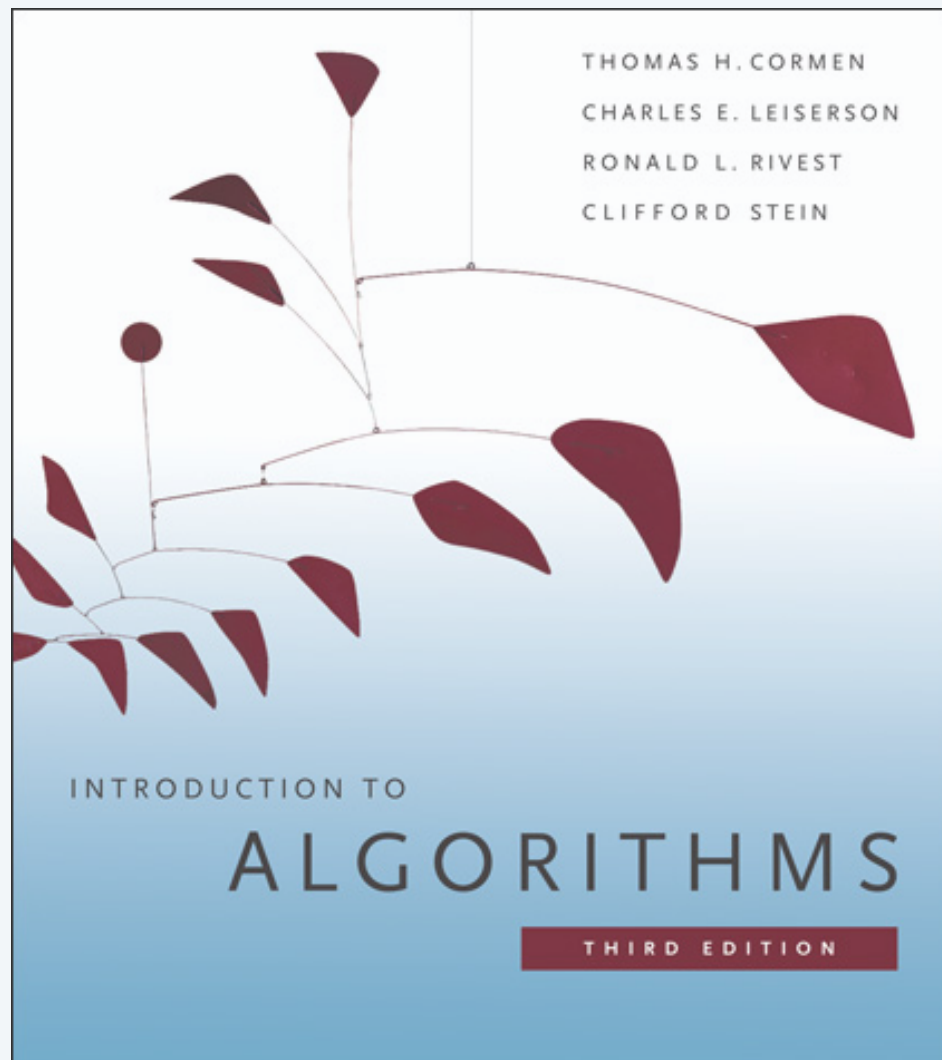
- Each recursive returns two lists: all points sorted by x -coordinate, and all points sorted by y -coordinate.
- Sort by **merging** two pre-sorted lists.

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Pf.
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{otherwise} \end{cases}$$

Note. See SECTION 13.7 for a randomized $O(n)$ time algorithm.

 not subject to lower bound
since it uses the floor function



CHAPTER 7

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *closest pair of points*
- ▶ ***randomized quicksort***
- ▶ *median and selection*

Randomized quicksort

3-way partition array so that:

- Pivot element p is in place.
- Smaller elements in left subarray L .
- Equal elements in middle subarray M .
- Larger elements in right subarray R .

the array A

7	6	12	3	11	8	9	1	4	10	2
---	---	----	---	----	---	---	---	---	----	---

p

the partitioned array A

3	1	4	2	6	7	12	11	8	9	10
---	---	---	---	---	---	----	----	---	---	----

|----- L -----| M |----- R -----|

Recur in both left and right subarrays.

RANDOMIZED-QUICKSORT (A)

IF list A has zero or one element

RETURN.

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, a_i).$

RANDOMIZED-QUICKSORT (L).

RANDOMIZED-QUICKSORT (R).

3-way partitioning
can be done in-place
(using $n-1$ compares)

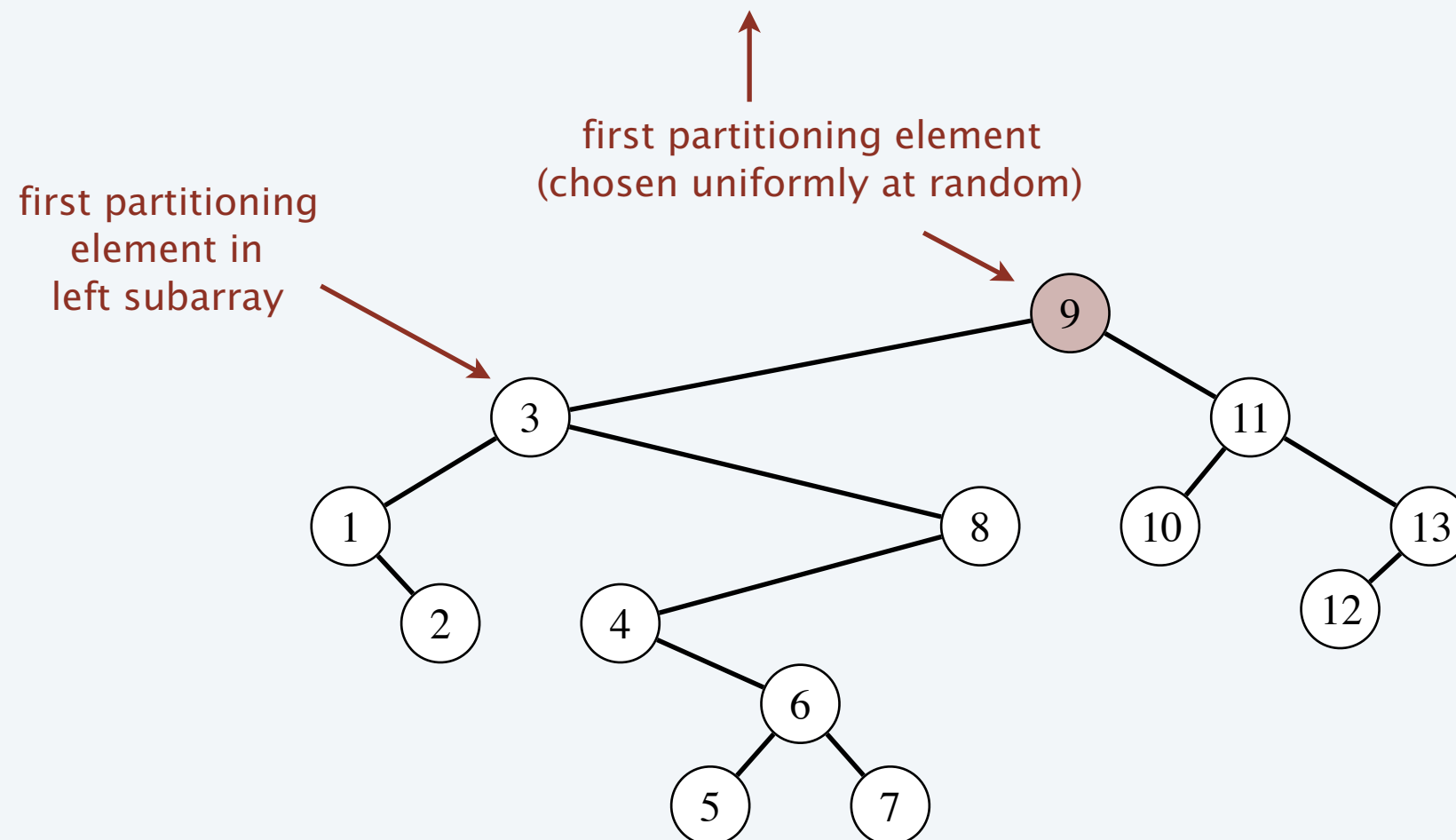
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

the original array of elements A

7	6	12	3	11	8	9	1	4	10	2	13	5
---	---	----	---	----	---	---	---	---	----	---	----	---

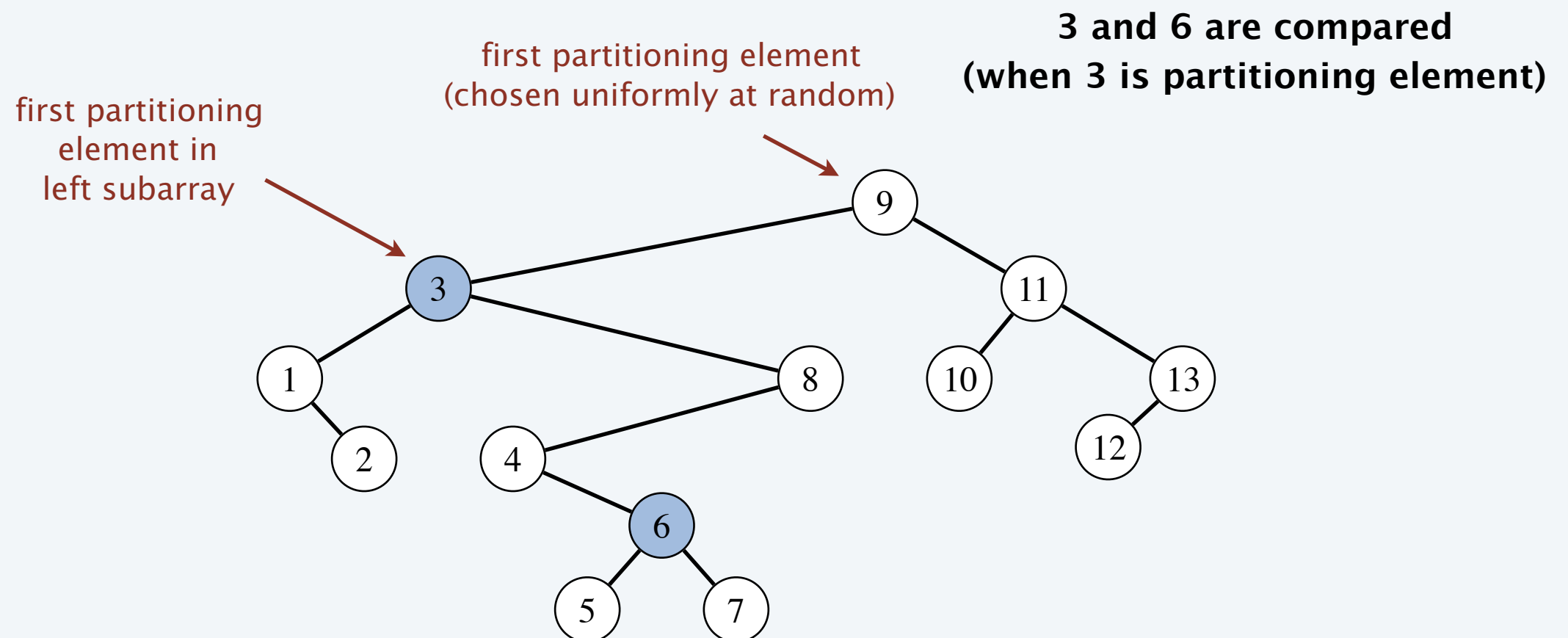


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.

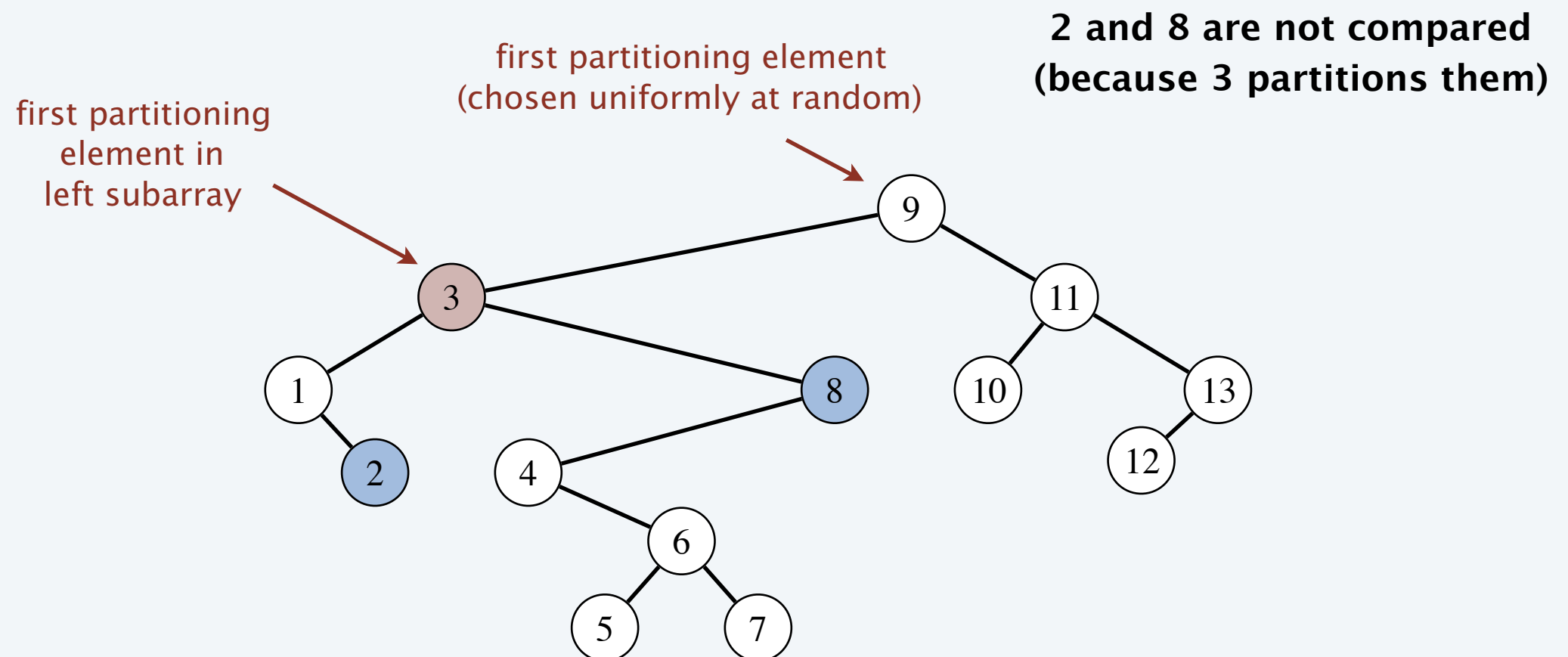


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.

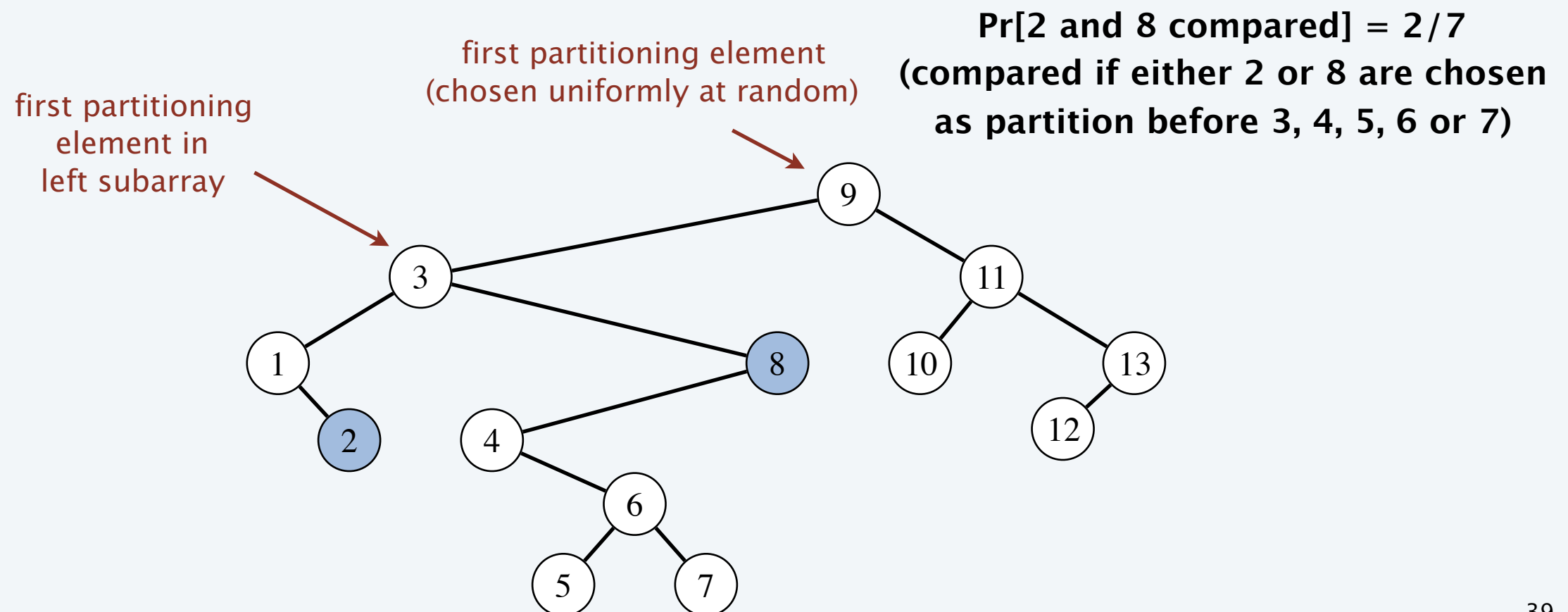


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
- $\Pr [a_i \text{ and } a_j \text{ are compared}] = 2 / |j - i + 1|$.




Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

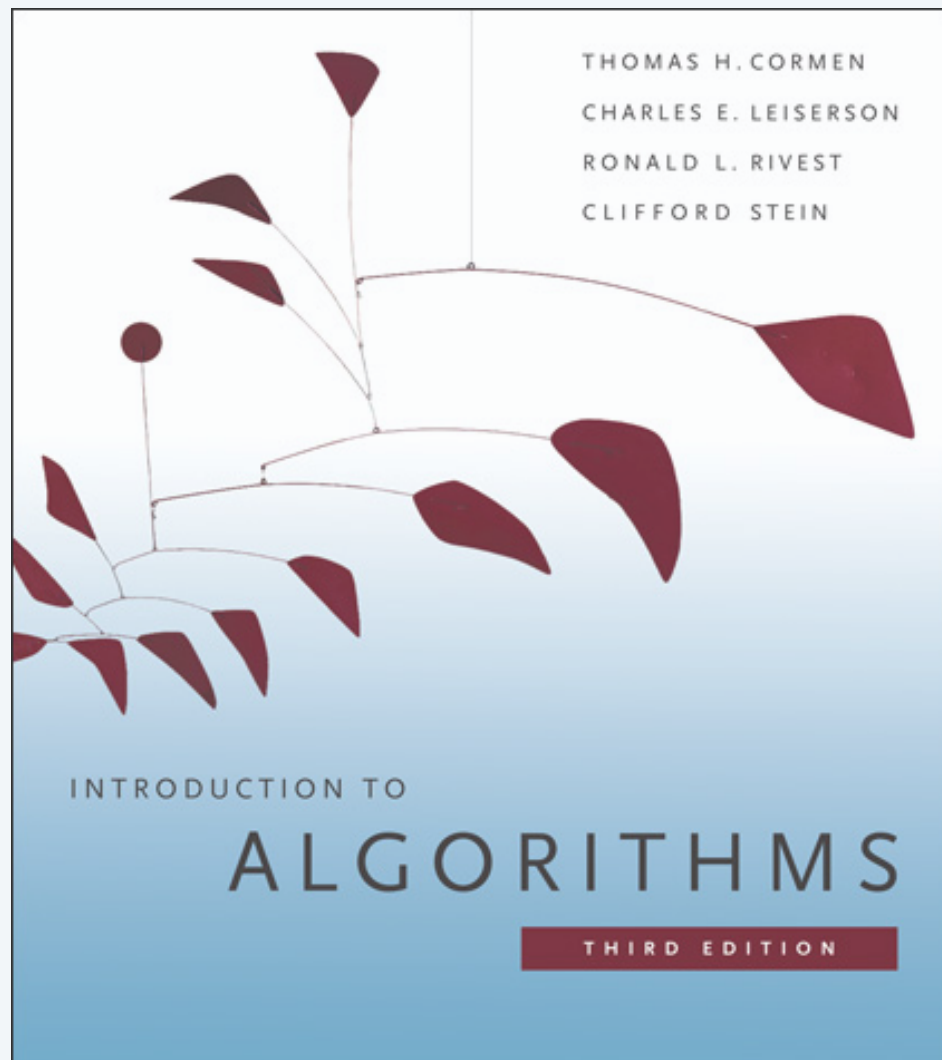
- An element is compared with only its ancestors and descendants.
- **Pr** [a_i and a_j are compared] = $2 / |j - i + 1|$.

- Expected number of compares = $\sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i-1} = 2 \sum_{i=1}^n \sum_{j=2}^{n-i+1} \frac{1}{j}$


all pairs i and j

$$\leq 2n \sum_{j=1}^n \frac{1}{j}$$
$$\sim 2n \int_{x=1}^n \frac{1}{x} dx$$
$$= 2n \ln n$$

Remark. Number of compares only decreases if equal elements.



CHAPTER 9

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *closest pair of points*
- ▶ *randomized quicksort*
- ▶ *median and selection*

Median and selection problems

Selection. Given n elements from a totally ordered universe, find k^{th} smallest.

- Minimum: $k = 1$; maximum: $k = n$.
- Median: $k = \lfloor (n + 1) / 2 \rfloor$.
- $O(n)$ compares for min or max.
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap.

Applications. Order statistics; find the “top k ”; bottleneck paths, ...

Q. Can we do it with $O(n)$ compares?

A. Yes! Selection is easier than sorting.

Quickselect

3-way partition array so that:


- Pivot element p is in place.
- Smaller elements in left subarray L .
- Equal elements in middle subarray M .
- Larger elements in right subarray R .



Recur in **one** subarray—the one containing the k^{th} smallest element.

QUICK-SELECT (A, k)

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, p).$  3-way partitioning
can be done in-place
(using $n-1$ compares)

IF $k \leq |L|$ RETURN QUICK-SELECT (L, k).

ELSE IF $k > |L| + |M|$ RETURN QUICK-SELECT ($R, k - |L| - |M|$)

ELSE RETURN p .

Quickselect analysis

Intuition. Split candy bar uniformly \Rightarrow expected size of larger piece is $\frac{3}{4}$.

$$T(n) \leq T(\frac{3}{4}n) + n \Rightarrow T(n) \leq 4n$$

Def. $T(n, k)$ = expected # compares to select k^{th} smallest in an array of size $\leq n$.

Def. $T(n) = \max_k T(n, k)$.

Proposition. $T(n) \leq 4n$.

Pf. [by strong induction on n]

- Assume true for $1, 2, \dots, n-1$.
- $T(n)$ satisfies the following recurrence:

can assume we always recur on largest subarray
since $T(n)$ is monotonic and
we are trying to get an upper bound

$$\begin{aligned} T(n) &\leq n + 2/n [T(n/2) + \dots + T(n-3) + T(n-2) + T(n-1)] \\ &\leq n + 2/n [4n/2 + \dots + 4(n-3) + 4(n-2) + 4(n-1)] \\ &= n + 4(3/4n) \\ &= 4n. \quad \blacksquare \end{aligned}$$


tiny cheat: sum should start at $T(\lfloor n/2 \rfloor)$

Selection in worst case linear time

Goal. Find pivot element p that divides list of n elements into two pieces so that each piece is **guaranteed** to have $\leq 7/10 n$ elements.

Q. How to find approximate median in linear time?

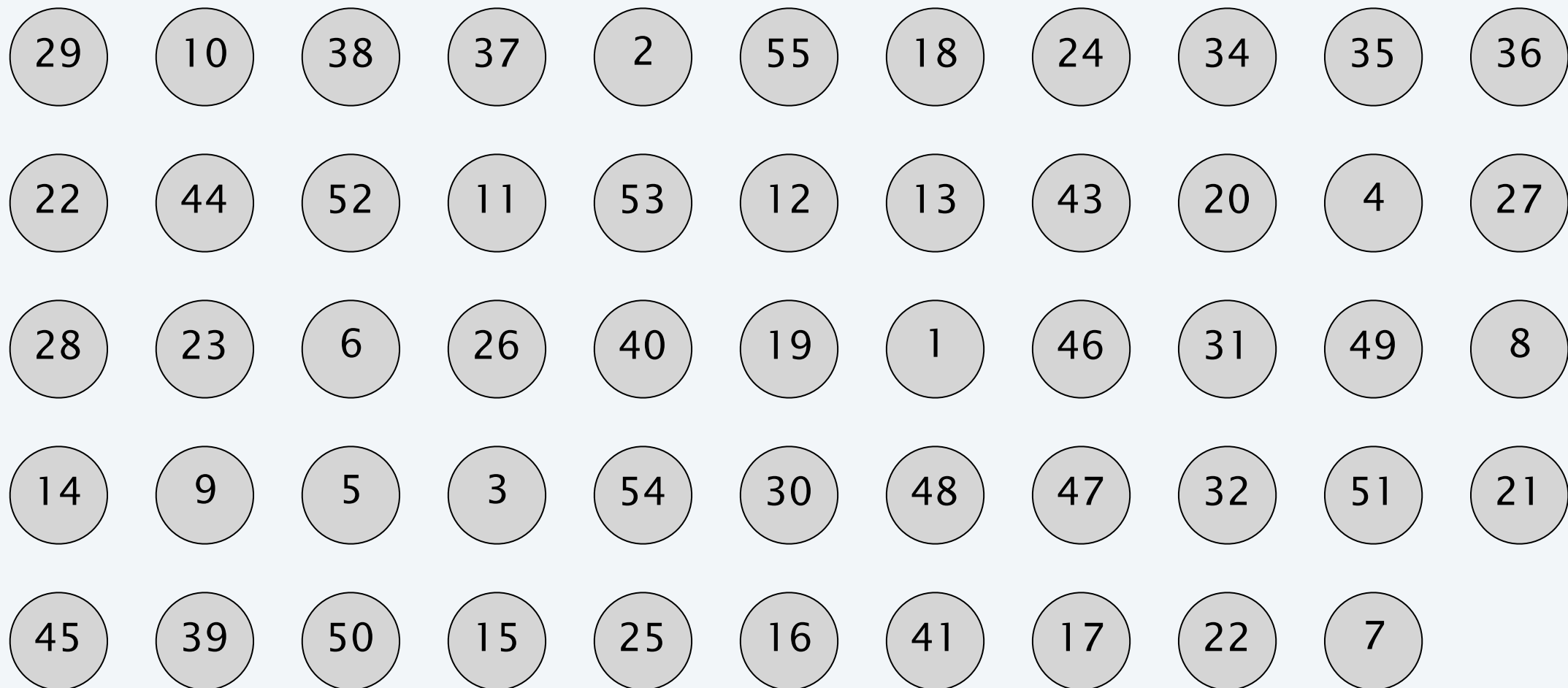
A. Recursively compute median of sample of $\leq 2/10 n$ elements.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(7/10 n) + T(2/10 n) + \Theta(n) & \text{otherwise} \end{cases}$$


two subproblems
of different sizes!

Choosing the pivot element

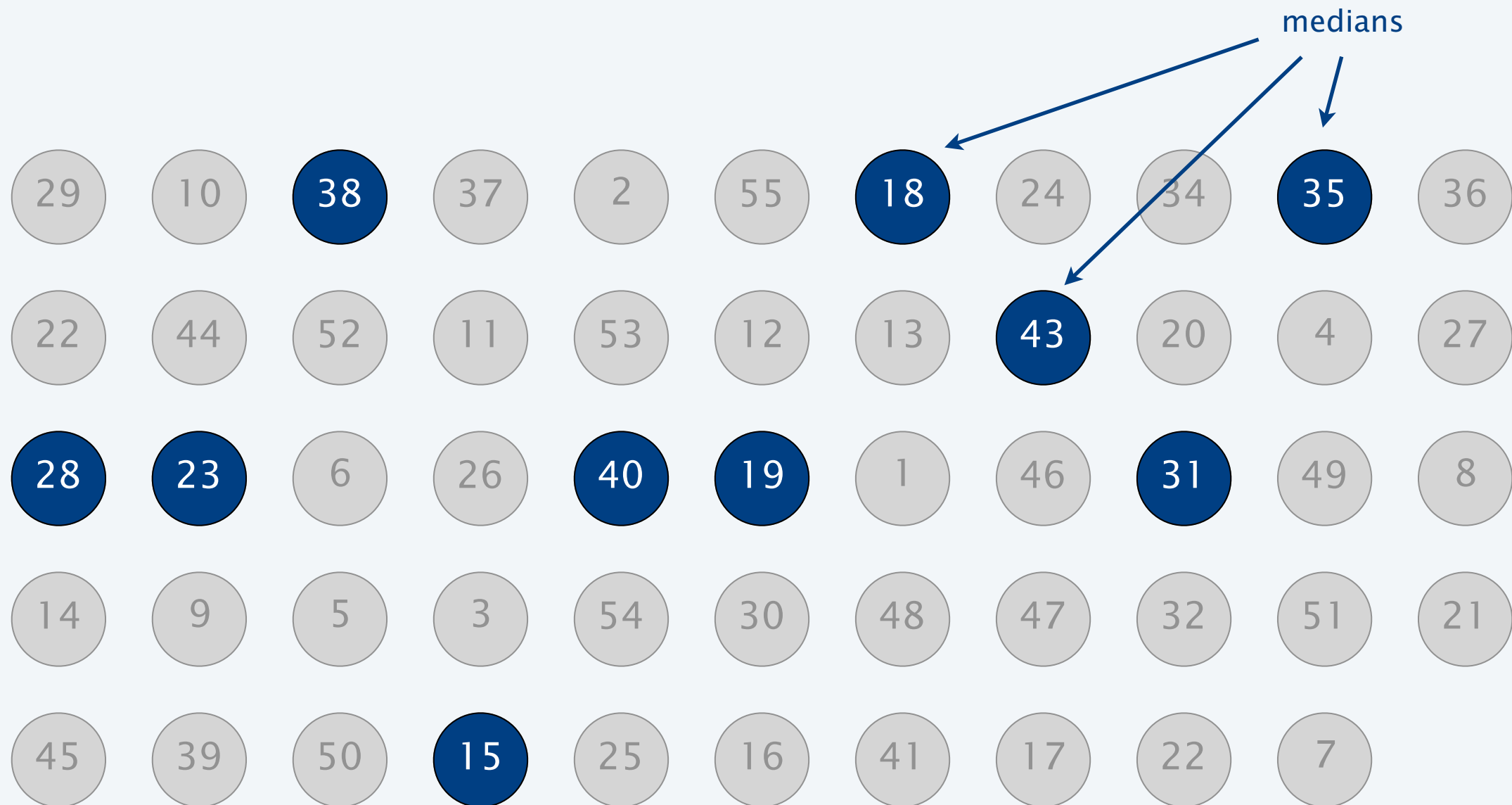
- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).



N = 54

Choosing the pivot element

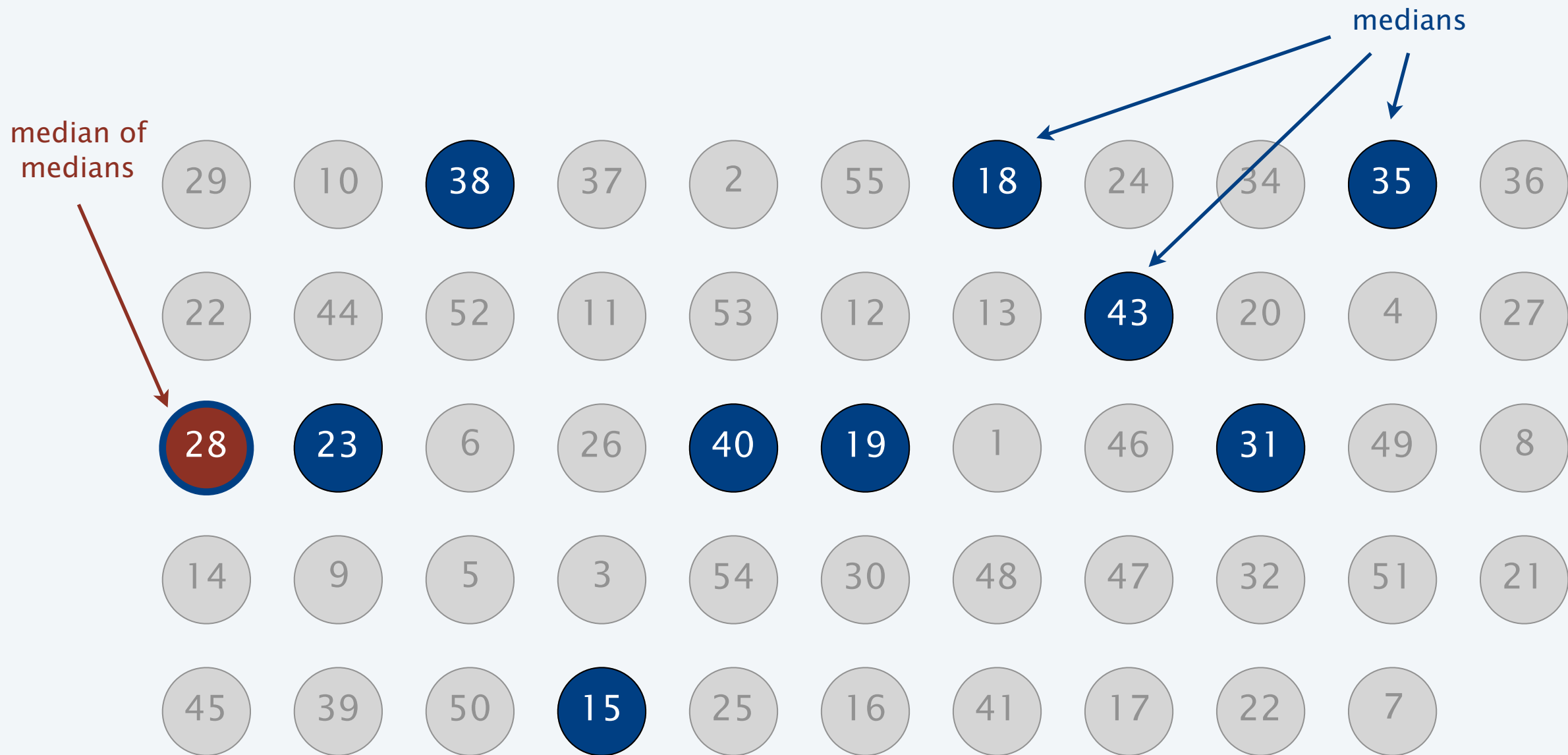
- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).



N = 54

Choosing the pivot element

- 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 $\lfloor n / 5 \rfloor$ medians recursively.
- Use median-of-medians as pivot element.



N = 54

Median-of-medians selection algorithm

MOM-SELECT (A, k)

$n \leftarrow |A|.$

IF $n < 50$ **RETURN** k^{th} smallest of element of A via mergesort.

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

$B \leftarrow$ median of each group of 5.

$p \leftarrow$ **MOM-SELECT**($B, \lfloor n / 10 \rfloor$) \longleftarrow median of medians

$(L, M, R) \leftarrow$ PARTITION-3-WAY (A, p).

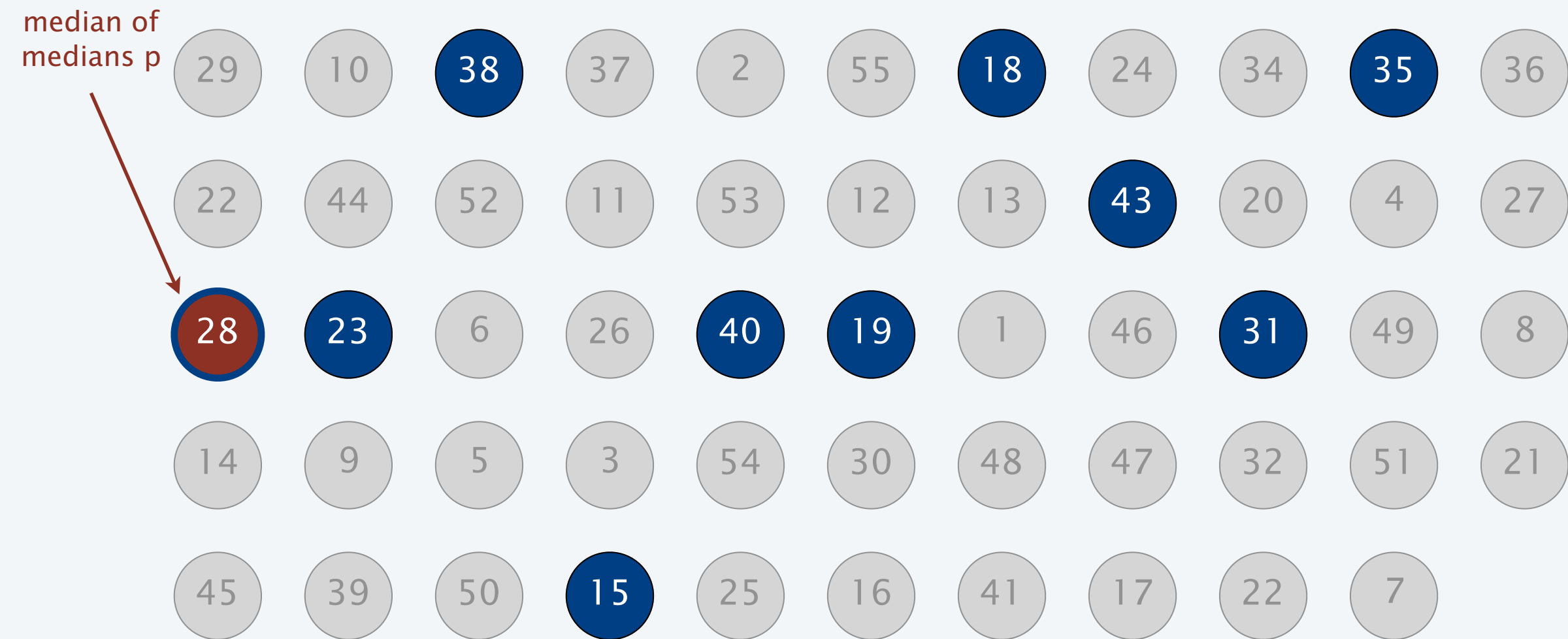
IF $k \leq |L|$ **RETURN** **MOM-SELECT** (L, k).

ELSE IF $k > |L| + |M|$ **RETURN** **MOM-SELECT** ($R, k - |L| - |M|$)

ELSE **RETURN** p .

Analysis of median-of-medians selection algorithm

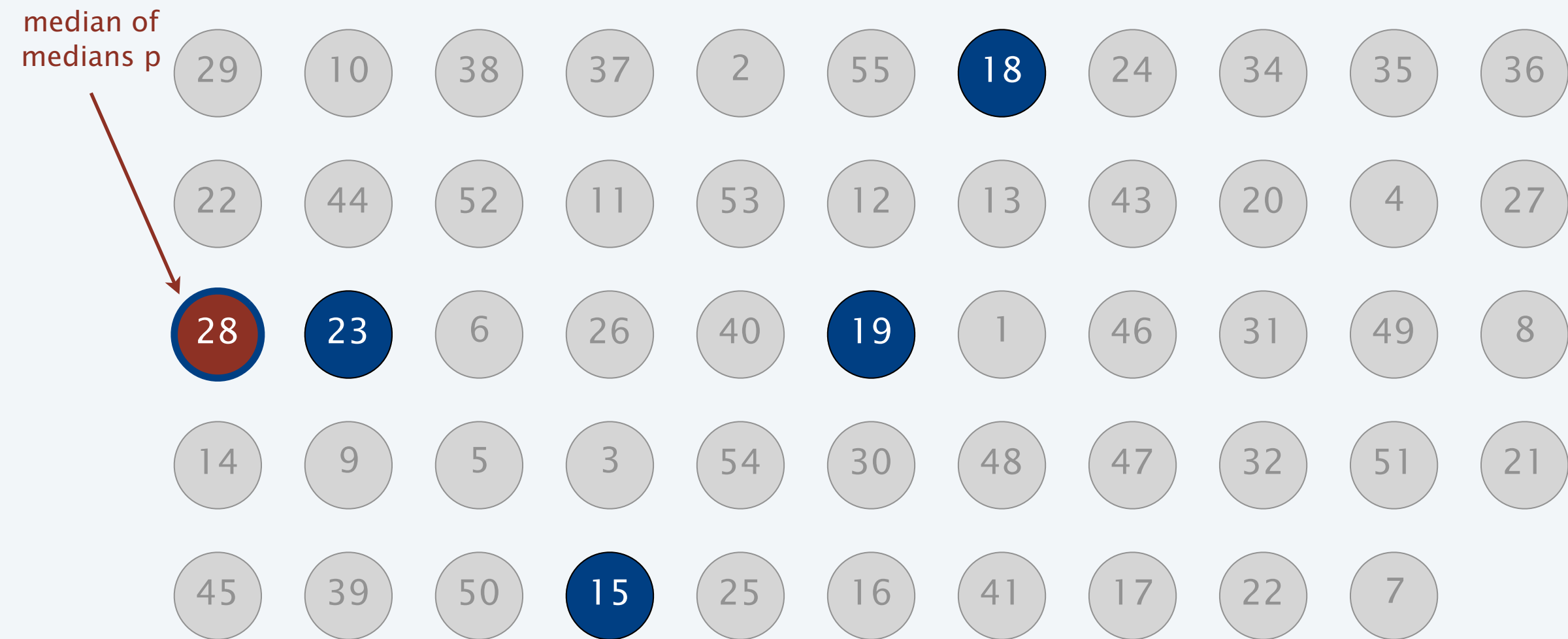
- At least half of 5-element medians $\leq p$.



$N = 54$

Analysis of median-of-medians selection algorithm

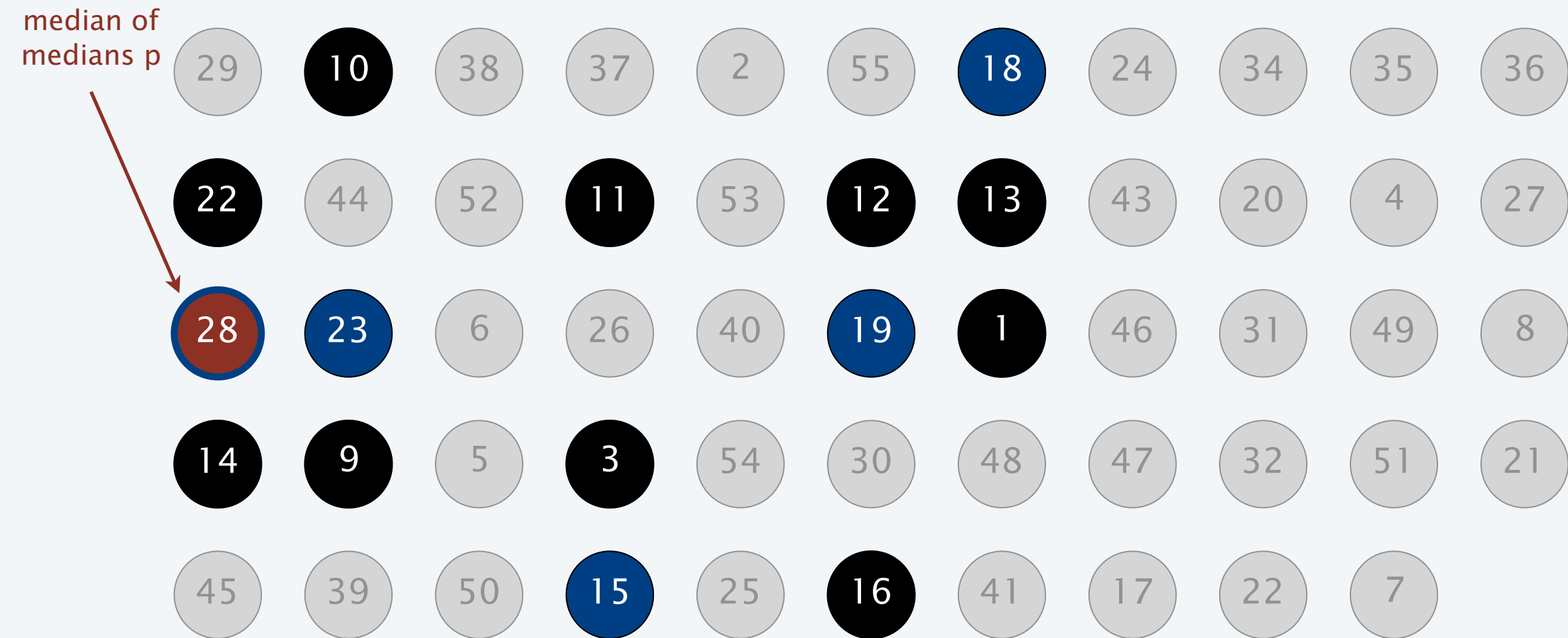
- At least half of 5-element medians $\leq p$.
- At least $\lfloor \lfloor n / 5 \rfloor / 2 \rfloor = \lfloor n / 10 \rfloor$ medians $\leq p$.



$N = 54$

Analysis of median-of-medians selection algorithm

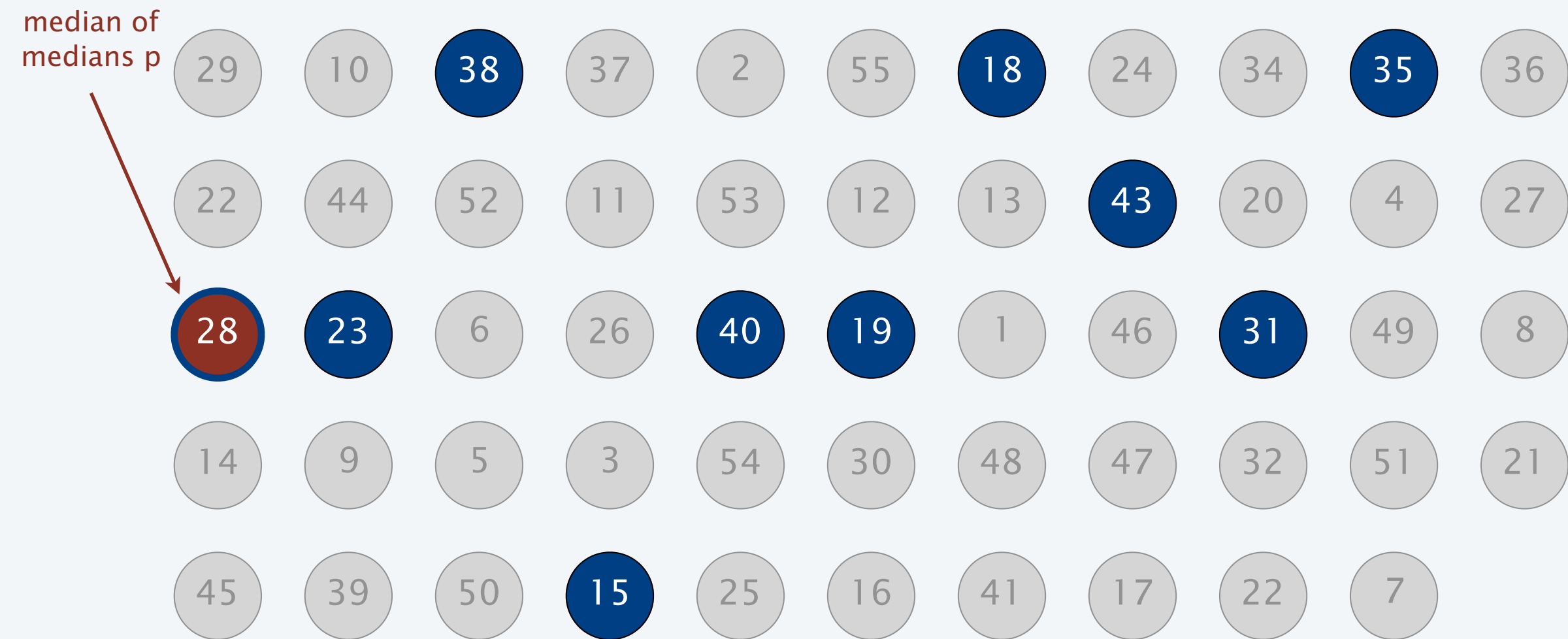
- At least half of 5-element medians $\leq p$.
- At least $\lfloor \lfloor n / 5 \rfloor / 2 \rfloor = \lfloor n / 10 \rfloor$ medians $\leq p$.
- At least $3 \lfloor n / 10 \rfloor$ elements $\leq p$.



$N = 54$

Analysis of median-of-medians selection algorithm

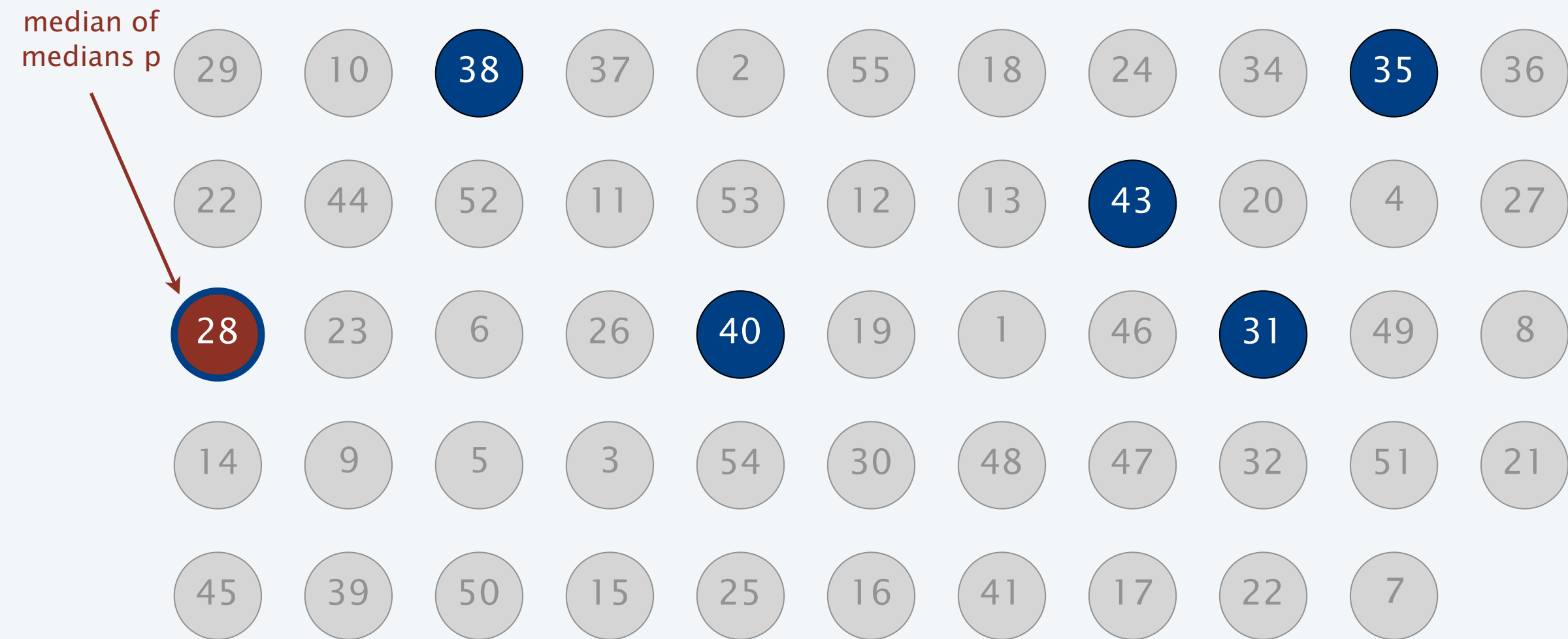
- At least half of 5-element medians $\geq p$.



$N = 54$

Analysis of median-of-medians selection algorithm

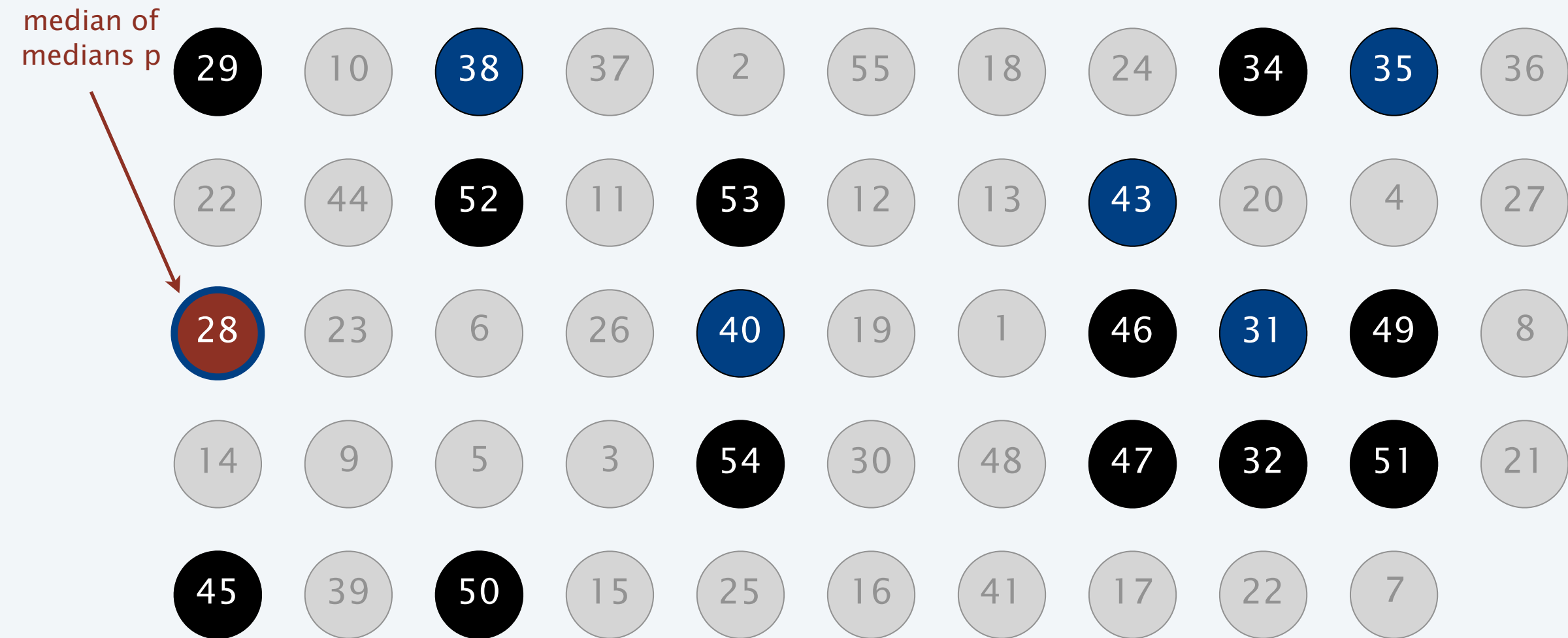
- At least half of 5-element medians $\geq p$.
- Symmetrically, at least $\lfloor n / 10 \rfloor$ medians $\geq p$.



$N = 54$

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.
- Symmetrically, at least $\lfloor n / 10 \rfloor$ medians $\geq p$.
- At least $3 \lfloor n / 10 \rfloor$ elements $\geq p$.



$N = 54$

Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm 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.

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

Now, solve recurrence.

- Assume n is both a power of 5 and a power of 10?
- Assume $C(n)$ is monotone nondecreasing?

Median-of-medians selection algorithm recurrence

Analysis of selection algorithm recurrence.

- $T(n)$ = max # compares on an array of $\leq n$ elements.
- $T(n)$ is monotone, but $C(n)$ is not!

$$T(n) \leq \begin{cases} 6n & \text{if } n < 50 \\ T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n & \text{otherwise} \end{cases}$$

Claim. $T(n) \leq 44n$.

- Base case: $T(n) \leq 6n$ for $n < 50$ (mergesort).
- Inductive hypothesis: assume true for $1, 2, \dots, n-1$.
- Induction step: for $n \geq 50$, we have:

$$\begin{aligned} T(n) &\leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n \\ &\leq 44(\lfloor n/5 \rfloor) + 44(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n \\ &\leq 44(n/5) + 44n - 44(n/4) + \frac{11}{5}n \quad \longleftarrow \text{for } n \geq 50, 3\lfloor n/10 \rfloor \geq n/4 \\ &= 44n. \quad \blacksquare \end{aligned}$$

Linear-time selection postmortem

Proposition. [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is $O(n)$.

Time Bounds for Selection

by .

Manuel Blum, Robert W. Floyd, Vaughan Pratt,
Ronald L. Rivest, and Robert E. Tarjan

Abstract

The number of comparisons required to select the i -th smallest of n numbers is shown to be at most a linear function of n by analysis of a new selection algorithm -- PICK. Specifically, no more than $5.4305 n$ comparisons are ever required. This bound is improved for

Theory.

- Optimized version of BFPRT: $\leq 5.4305 n$ compares.
- Best known upper bound [Dor–Zwick 1995]: $\leq 2.95 n$ compares.
- Best known lower bound [Dor–Zwick 1999]: $\geq (2 + \varepsilon) n$ compares.

Linear-time selection postmortem

Proposition. [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is $O(n)$.

Time Bounds for Selection

by .

Manuel Blum, Robert W. Floyd, Vaughan Pratt,
Ronald L. Rivest, and Robert E. Tarjan

Abstract

The number of comparisons required to select the i -th smallest of n numbers is shown to be at most a linear function of n by analysis of a new selection algorithm -- PICK. Specifically, no more than $5.4305n$ comparisons are ever required. This bound is improved for

Practice. Constant and overhead (currently) too large to be useful.

Open. Practical selection algorithm whose worst-case running time is $O(n)$.