5. DIVIDE AND CONQUER I

▸ *mergesort*

▸ *counting inversions*

▸ *closest pair of points*

▸ *randomized quicksort*

▸ *median and selection*

Last updated on Oct 2, 2013 9:51 AM

Divide-and-conquer paradigm

Divide-and-conquer.
- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.
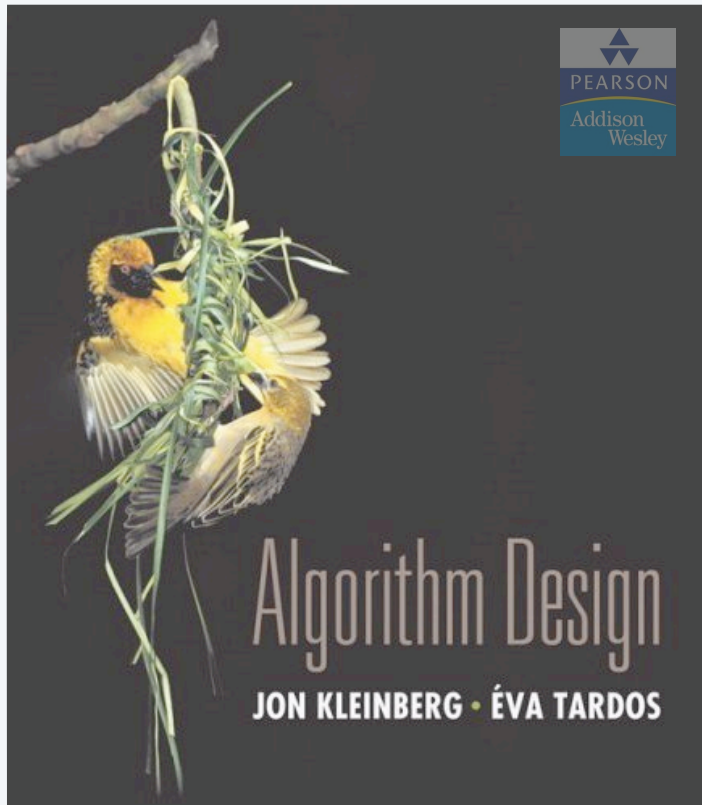
Most common usage.
- Divide problem of size n into two subproblems of size $n/2$ in linear time.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in linear time.

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

attributed to Julius Caesar

5. DIVIDE AND CONQUER

‣ *mergesort*

SECTION 5.1

Sorting problem

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

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.

First Draft of a Report on the EDVAC

John von Neumann

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

3	7	10	a_i	18

sorted list B

2	11	b_j	17	23

5   2

merge to form sorted list C

2	3	7	10	11					
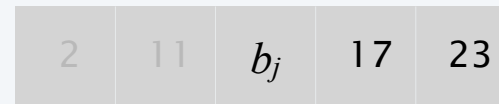
A useful recurrence relation

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

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.

$T(n)$ n $= n$

$T(n/2)$ $T(n/2)$ $2\,(n/2)$ $= n$

$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$ $4\,(n/4)$ $= n$

$\log_2 n$

$T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $8\,(n/8)$ $= n$

$T(n) = n \lg n$

9
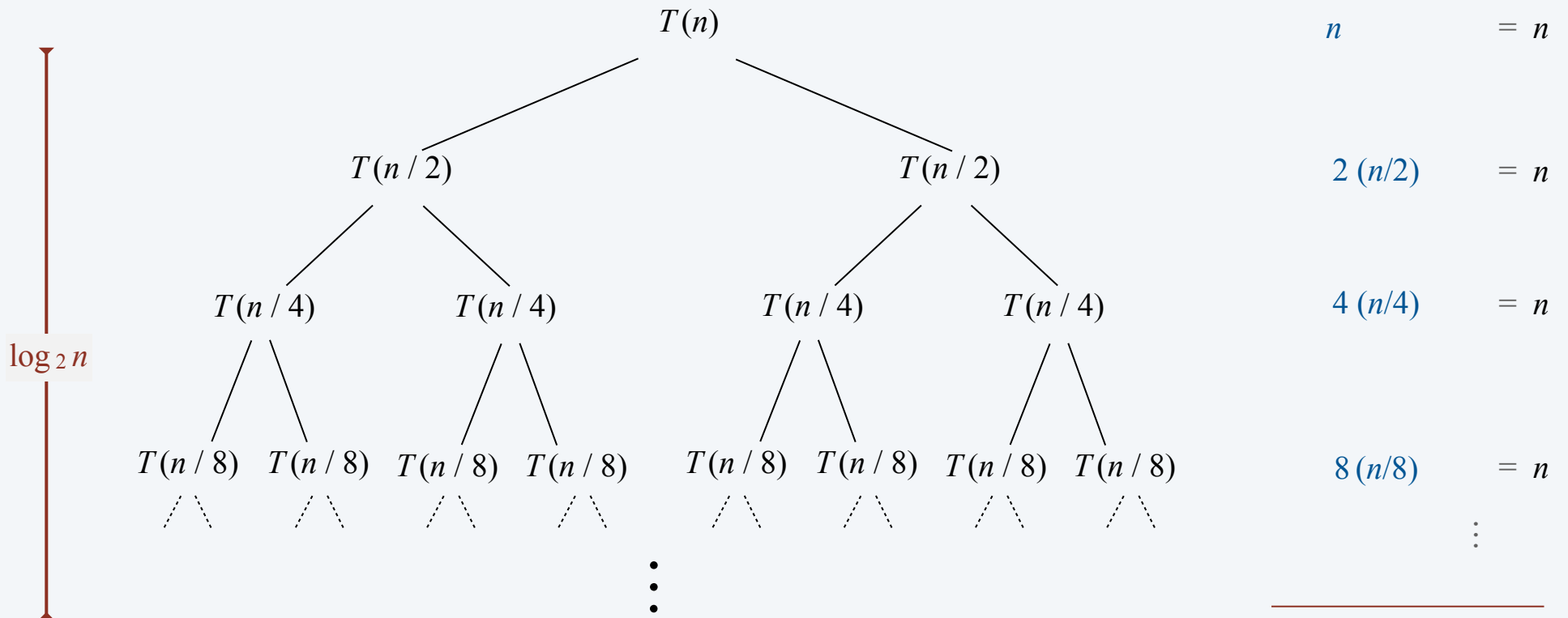
Proof by induction

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 $n = 1$, $T(1) = 0$.
- 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
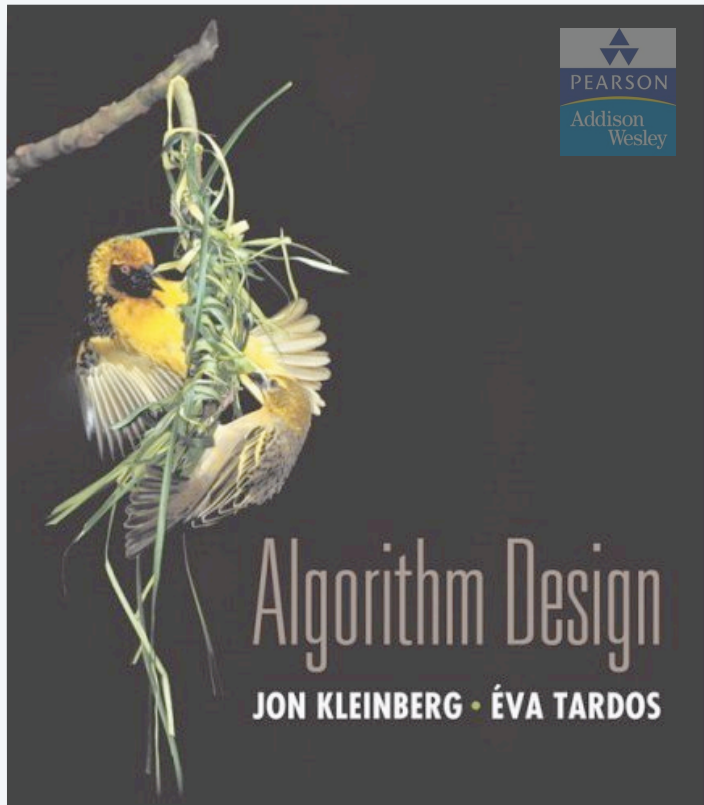
Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \le n \lceil \log_2 n \rceil$.

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

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 $1, 2, \dots, n-1$.

$$
\begin{aligned}
T(n) \quad &\le \quad T(n_1) + T(n_2) + n \\
&\le \quad n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
&\le \quad n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
&= \quad n \lceil \log_2 n_2 \rceil + n \qquad \longleftarrow \\
&\le \quad n (\lceil \log_2 n \rceil - 1) + n \\
&= \quad n \lceil \log_2 n \rceil. \qquad \blacksquare
\end{aligned}
$$

$$
\begin{aligned}
n_2 &= \lceil n/2 \rceil \\
&\le \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\
&= 2^{\lceil \log_2 n \rceil} / 2
\end{aligned}
$$

$$\log_2 n_2 \le \lceil \log_2 n \rceil - 1$$

5. DIVIDE AND CONQUER

SECTION 5.3

Algorithm Design

JON KLEINBERG · ÉVA TARDOS

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: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \ldots, a_n$.
- 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 ∈ 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: 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

17	23	2	11	16

sort A

3	7	10	14	18

sort B

2	11	16	17	23

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

3	7	10	14	18

2	11	16	17	23

| 5 | 2 | 1 | 1 | 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 ∈ A and b ∈ B

| 3 | 7 | 10 | a_i | 18 |

| 2 | 11 | b_j | 17 | 23 |

5   2

merge to form sorted list C

| 2 | 3 | 7 | 10 | 11 | | | | | |

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.

$(r_A, A) \leftarrow$ 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: 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*

SECTION 5.4

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
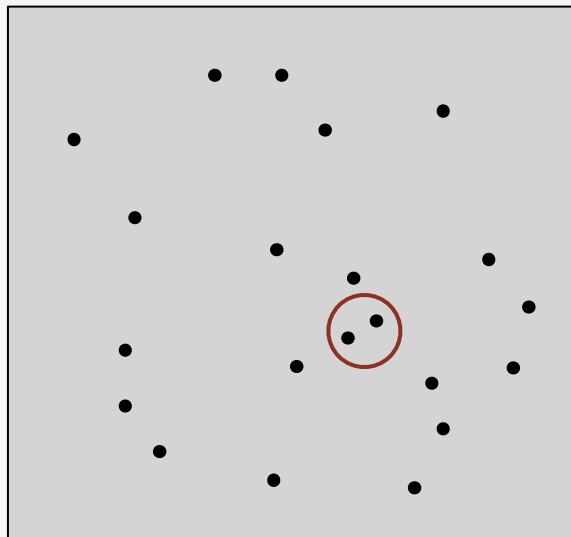
Closest pair of points

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.
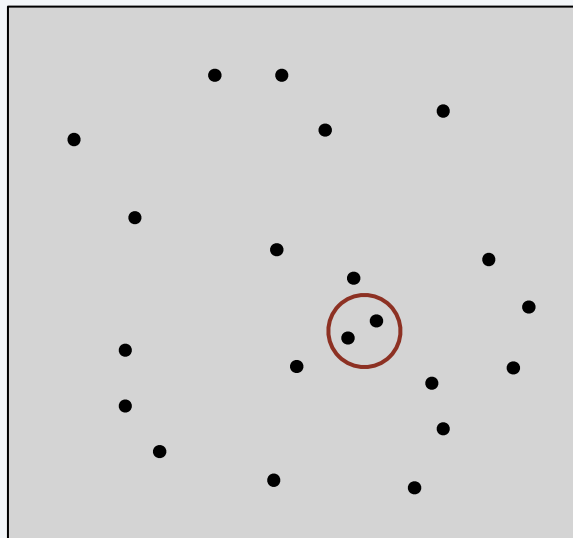
Closest pair of points: first attempt

Sorting solution.

- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.
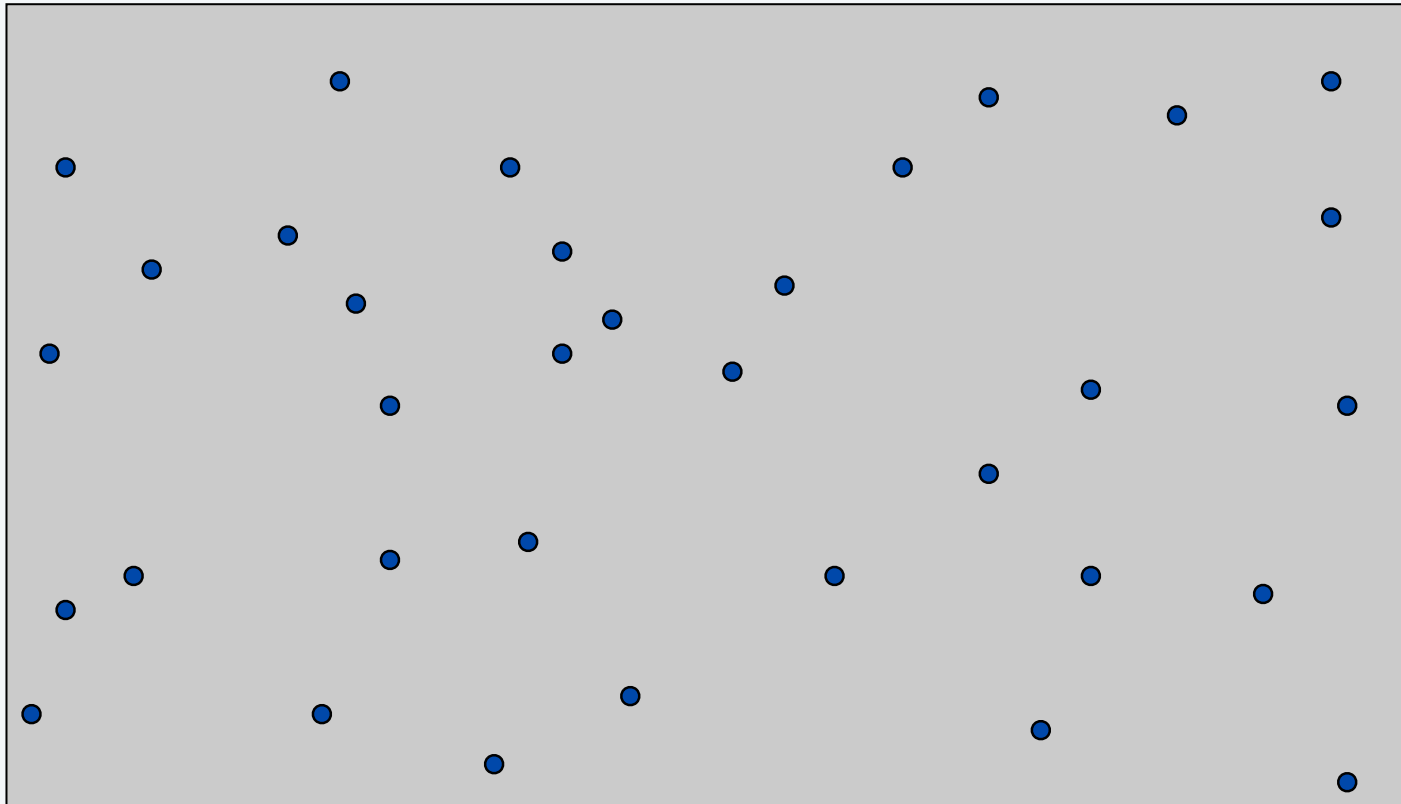
Closest pair of points: first attempt

Sorting solution.

- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.
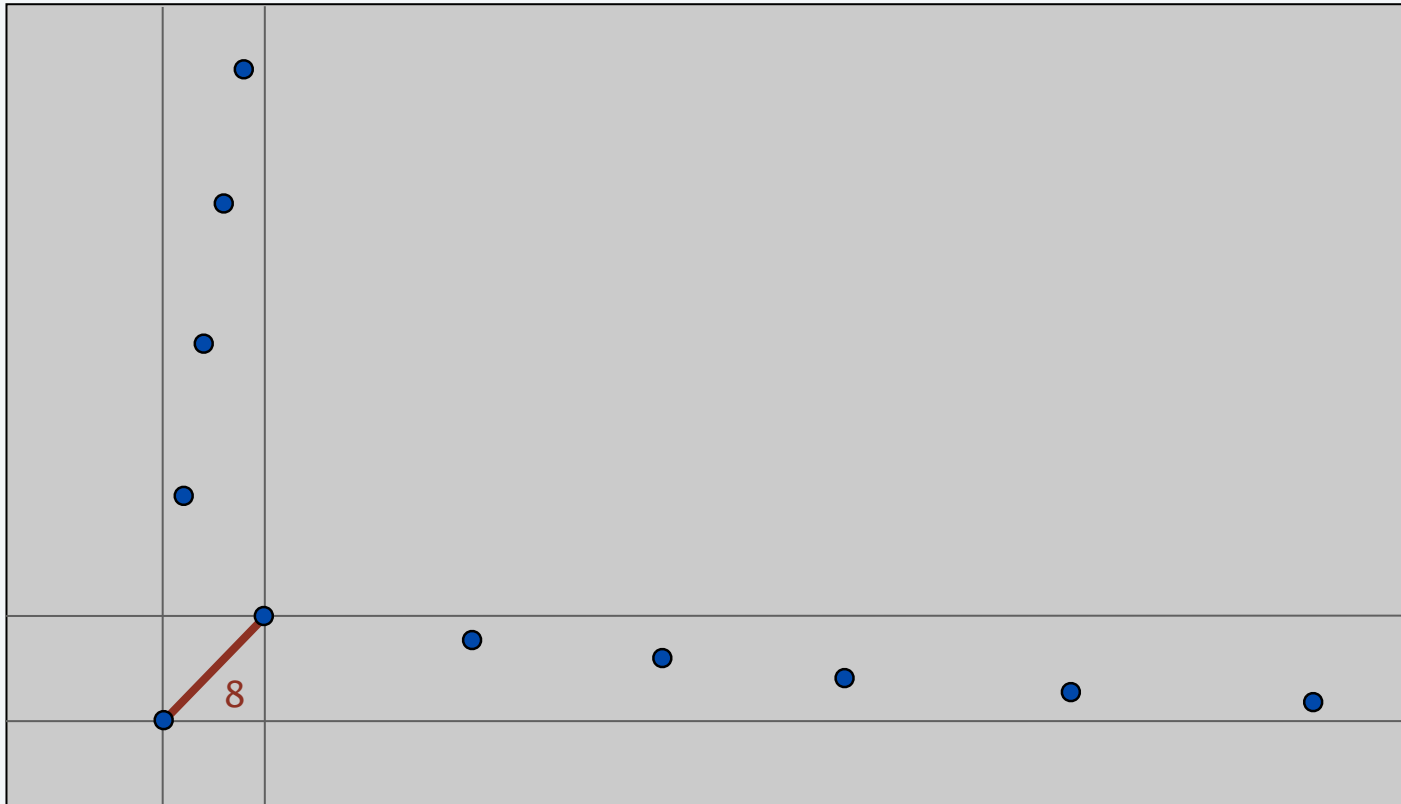
Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.
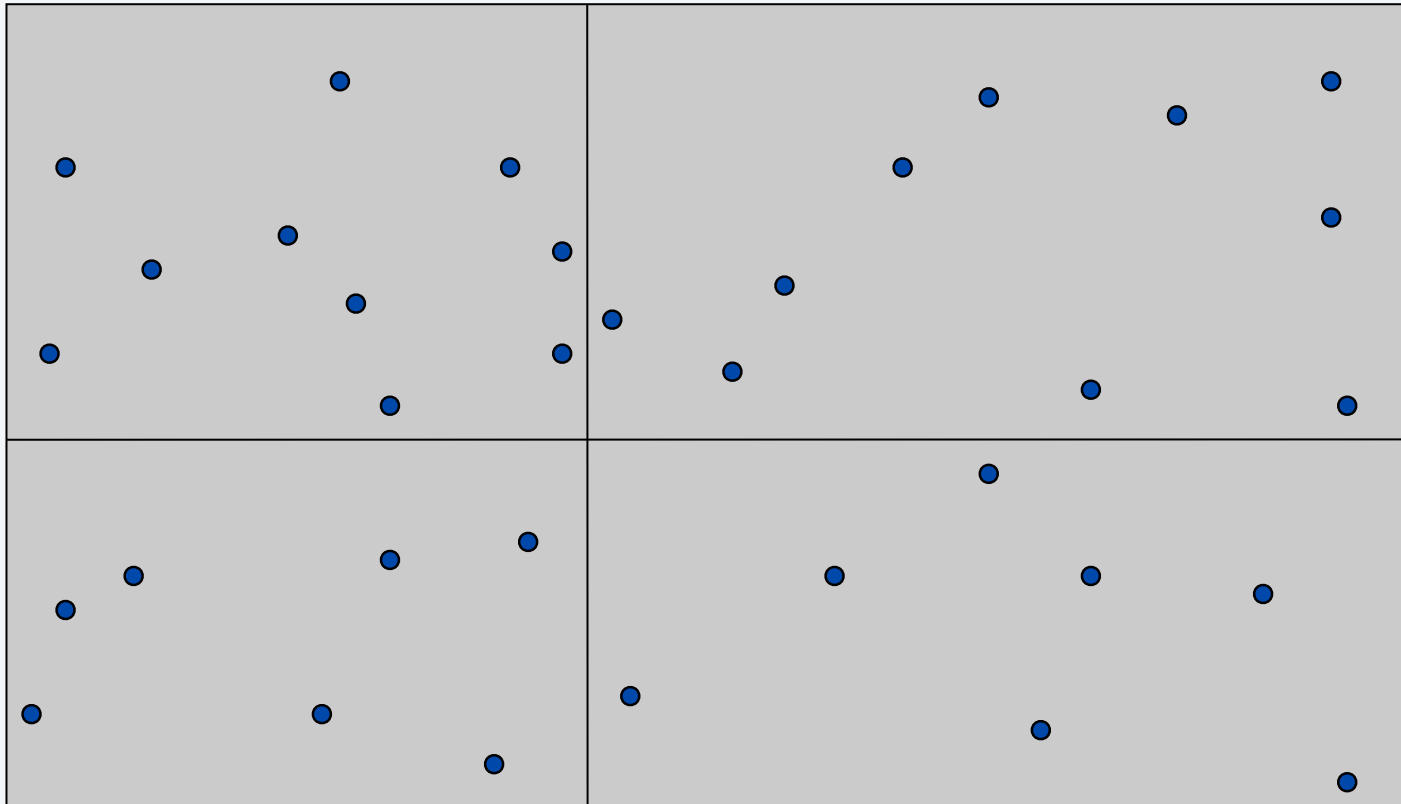
Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

Obstacle. Impossible to ensure $n/4$ points in each piece.
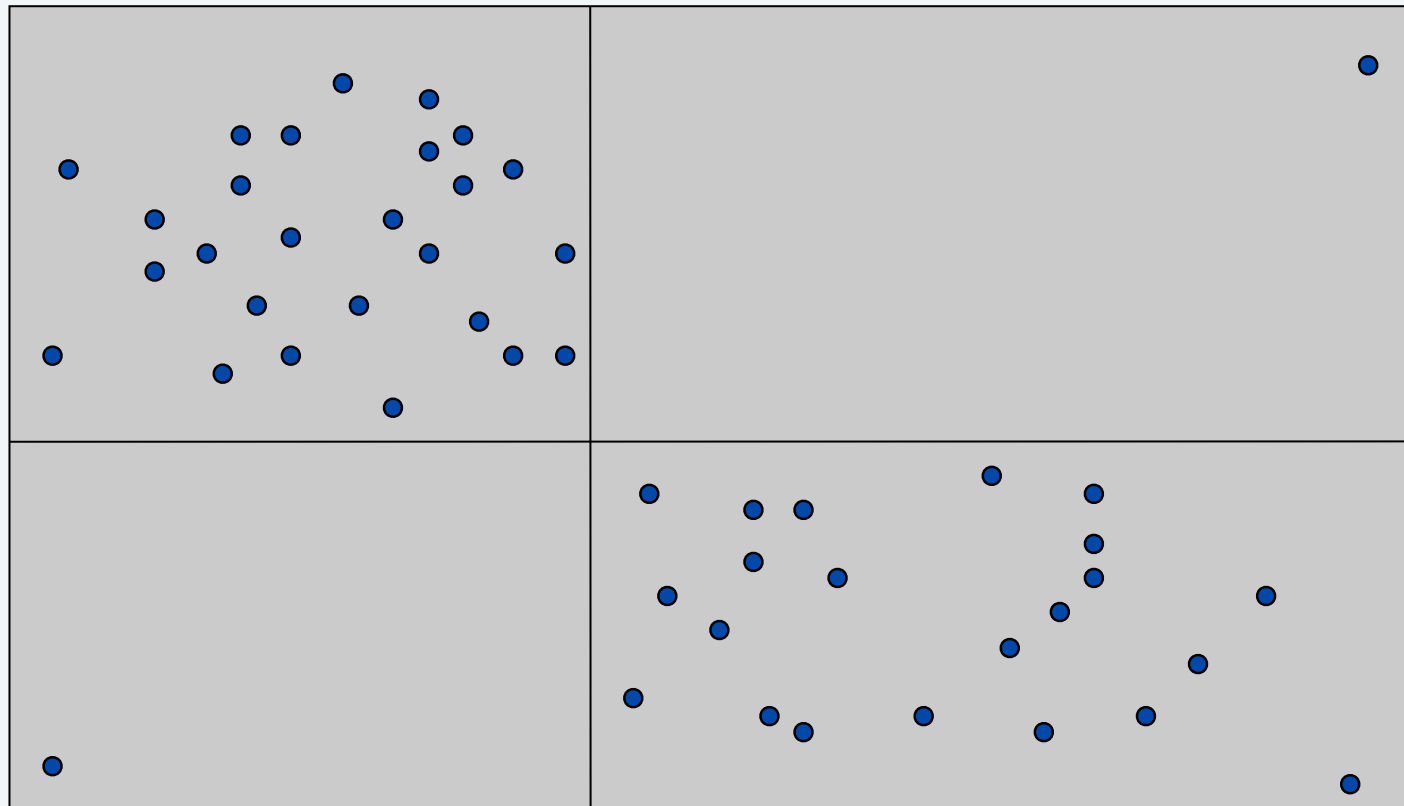
Closest pair of points: divide-and-conquer algorithm

- 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 $\Theta(N^2)$
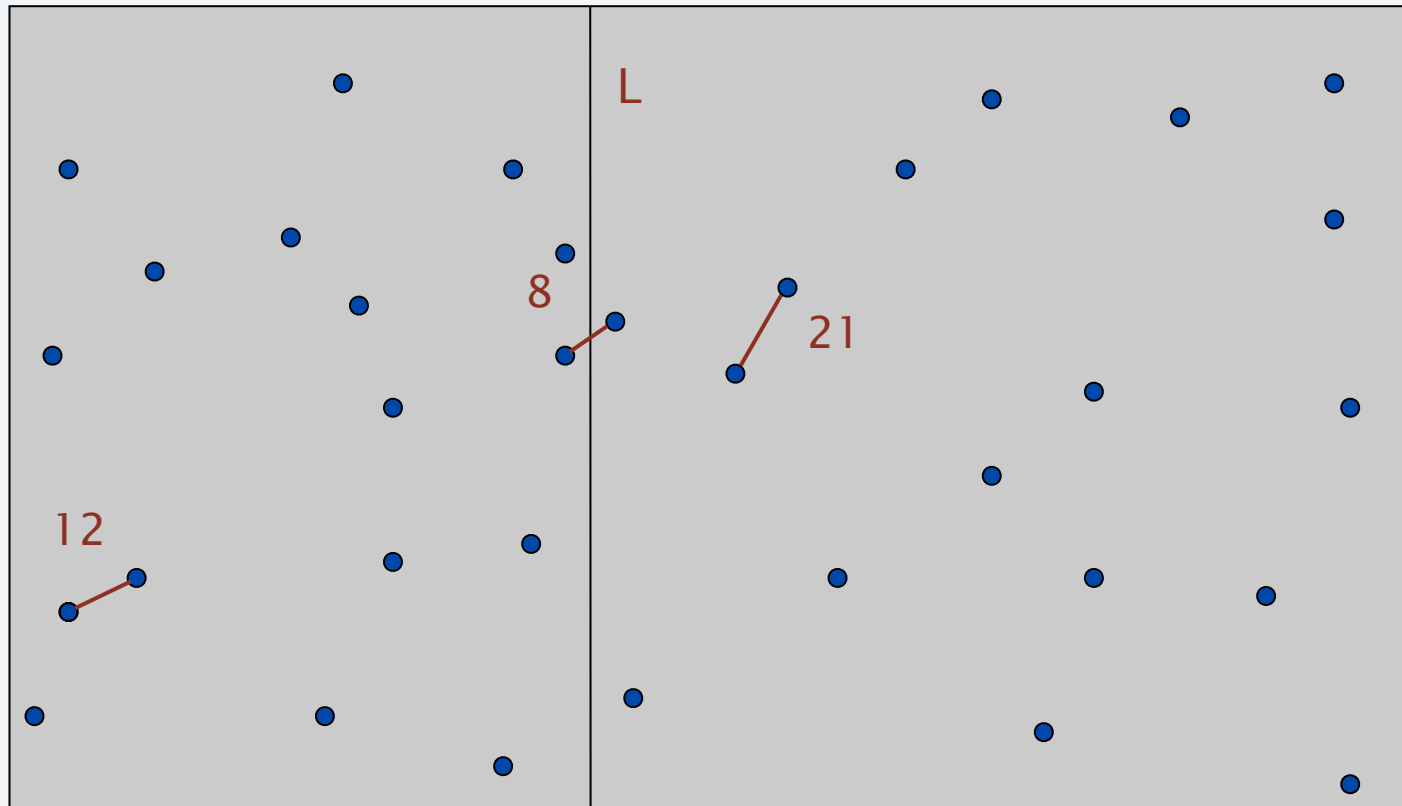
How to find closest pair with one point in each side?

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

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

L

21

12

$\delta = \min(12, 21)$

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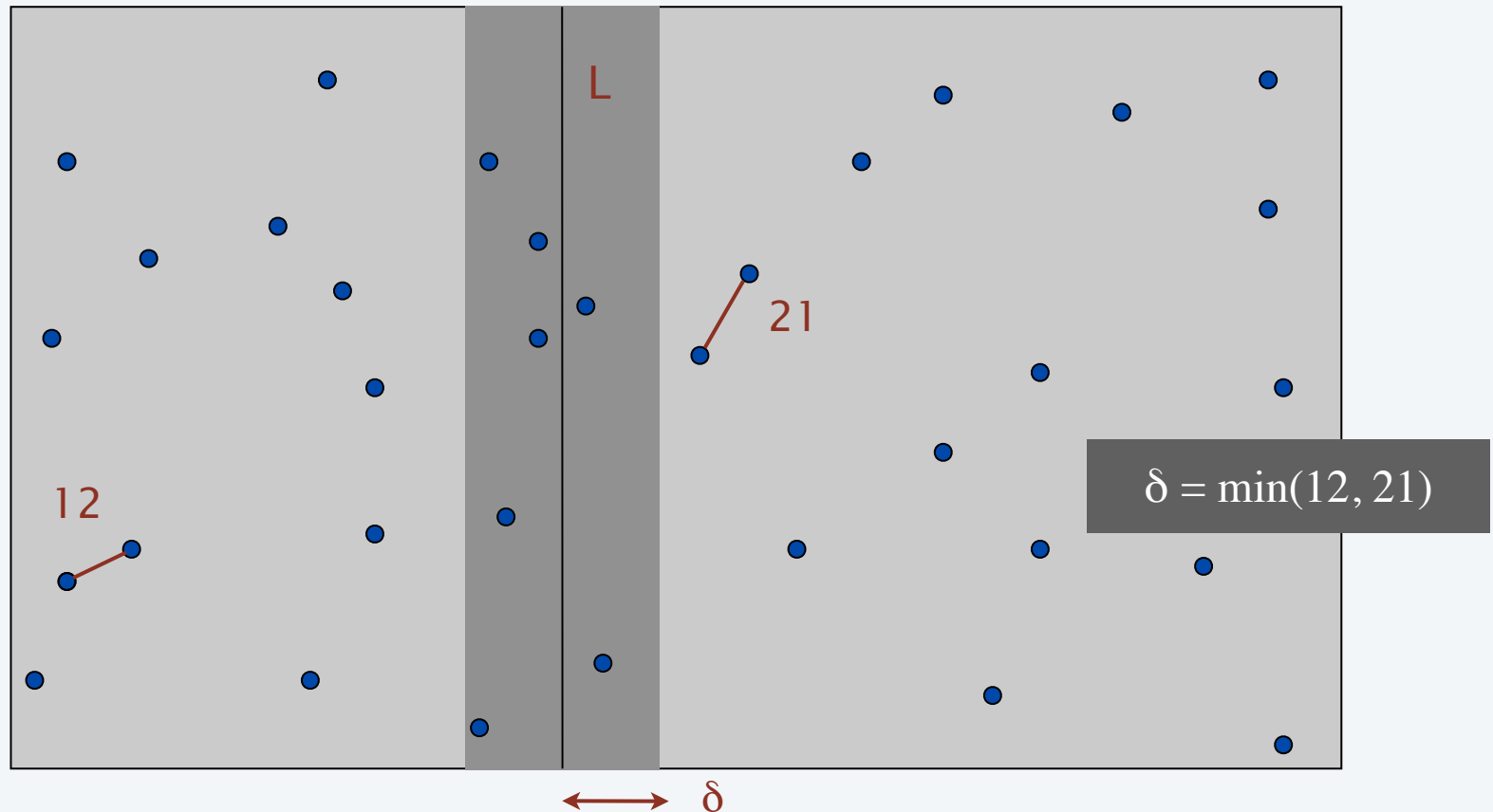
How to find closest pair with one point in each side?

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?

L

7

6

5

4

21

3

12

$\delta = \min(12, 21)$

2

1

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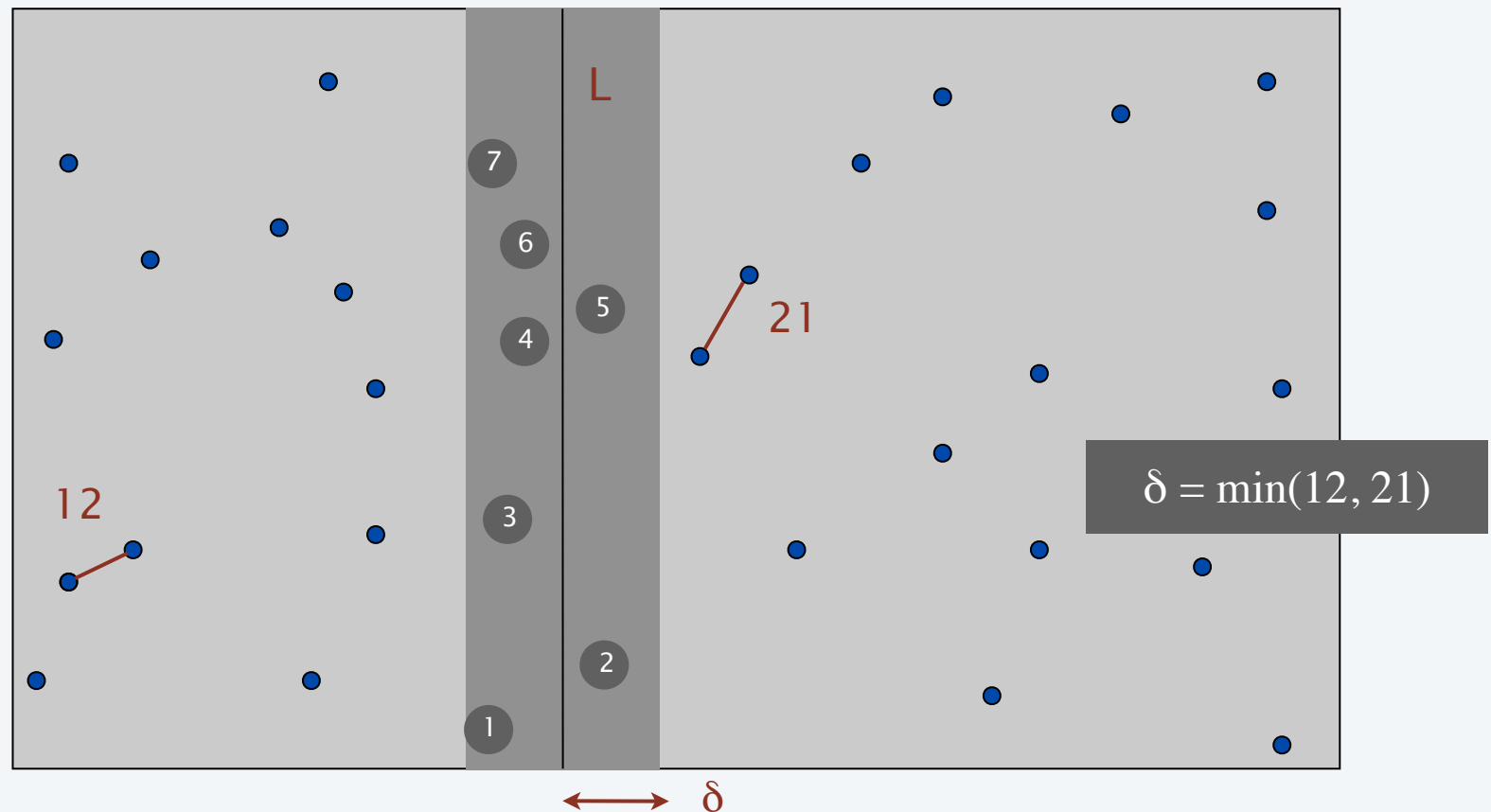
How to find closest pair with one point in each side?

Def. Let s_i be the point in the 2δ-strip, with the ith 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.
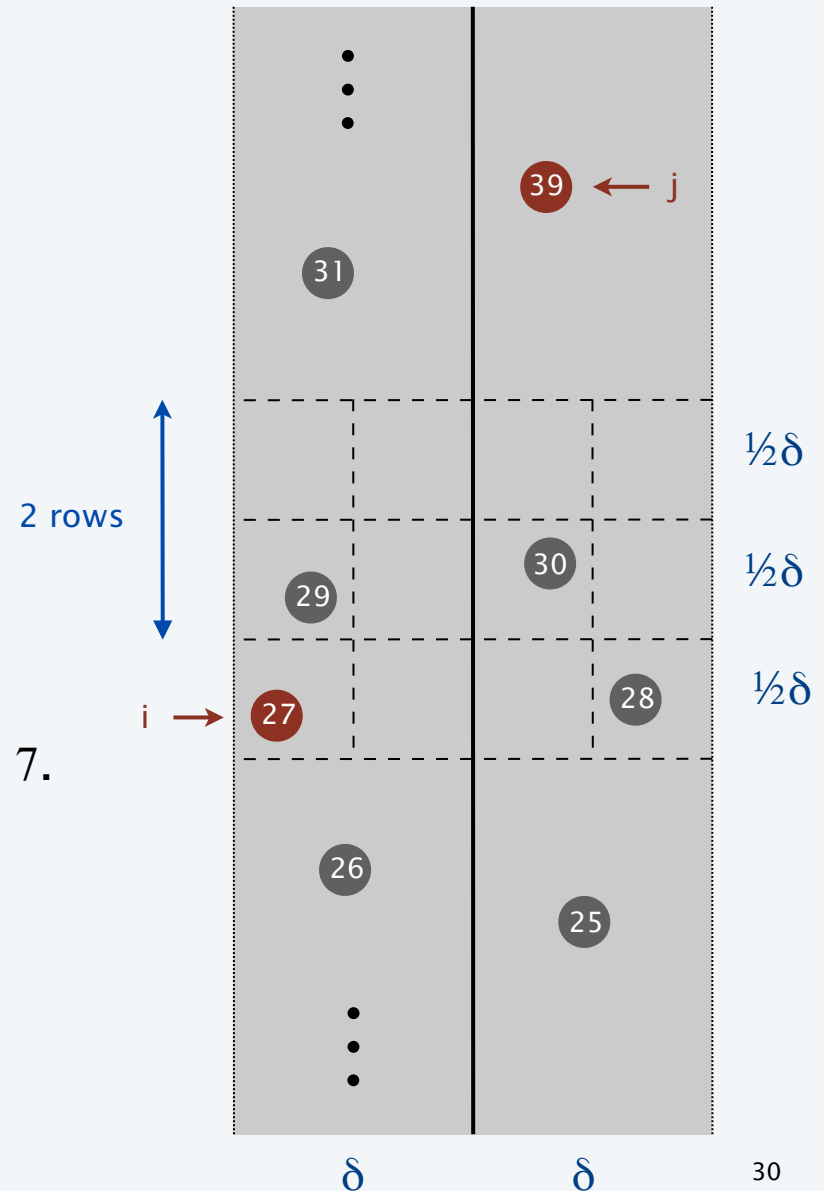
Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR $(p_1, p_2, \ldots, p_n)$

Compute separation line L such that half the points are on each side of the line.　　　　$O(n \log n)$

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

$\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half).　　　$2\,T(n\,/\,2)$

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

Delete all points further than δ from line L.　　　$O(n)$

Sort remaining points by y-coordinate.　　　$O(n \log n)$

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.　　　$O(n)$

RETURN δ.

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
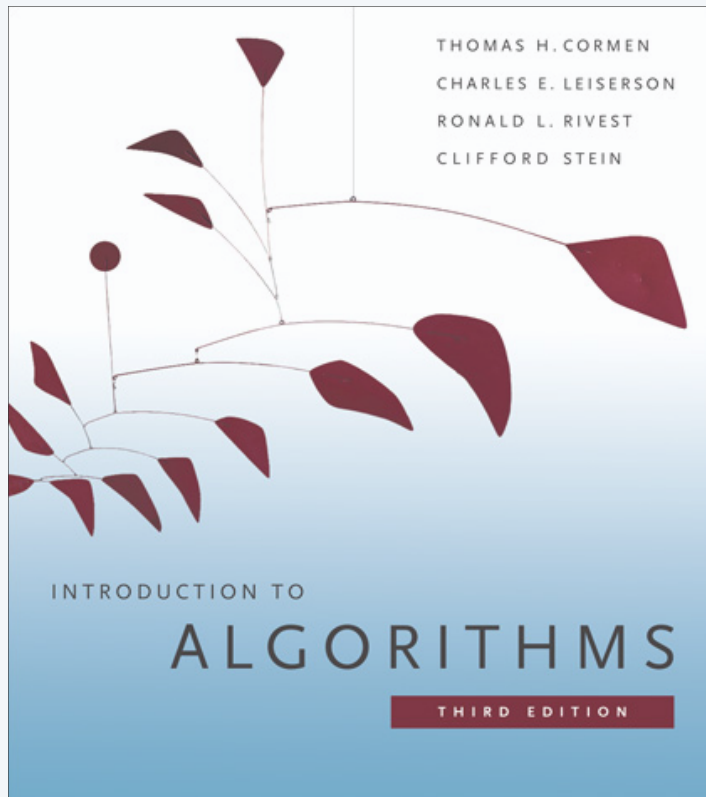
5. DIVIDE AND CONQUER

INTRODUCTION TO
ALGORITHMS

THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN

THIRD EDITION

CHAPTER 7

▸ 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.

Recur in both left and right subarrays.

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 ──────┤

RANDOMIZED-QUICKSORT (A)

IF list A has zero or one element

   RETURN.

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

$(L, M, R) \leftarrow$ PARTITION-3-WAY (A, a_i). ◄──── 3-way partitioning can be done in-place (using n−1 compares)

RANDOMIZED-QUICKSORT(L).

RANDOMIZED-QUICKSORT(R).

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

first partitioning element
(chosen uniformly at random)

first partitioning
element in
left subarray

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.

first partitioning element in left subarray

first partitioning element (chosen uniformly at random)

3 and 6 are compared (when 3 is partitioning element)

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.

first partitioning element
(chosen uniformly at random)

2 and 8 are not compared
(because 3 partitions them)

first partitioning
element in
left subarray

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] $= 2 \ / \ |j - i + 1|$.

first partitioning element in left subarray

first partitioning element (chosen uniformly at random)

Pr[2 and 8 compared] = 2/7
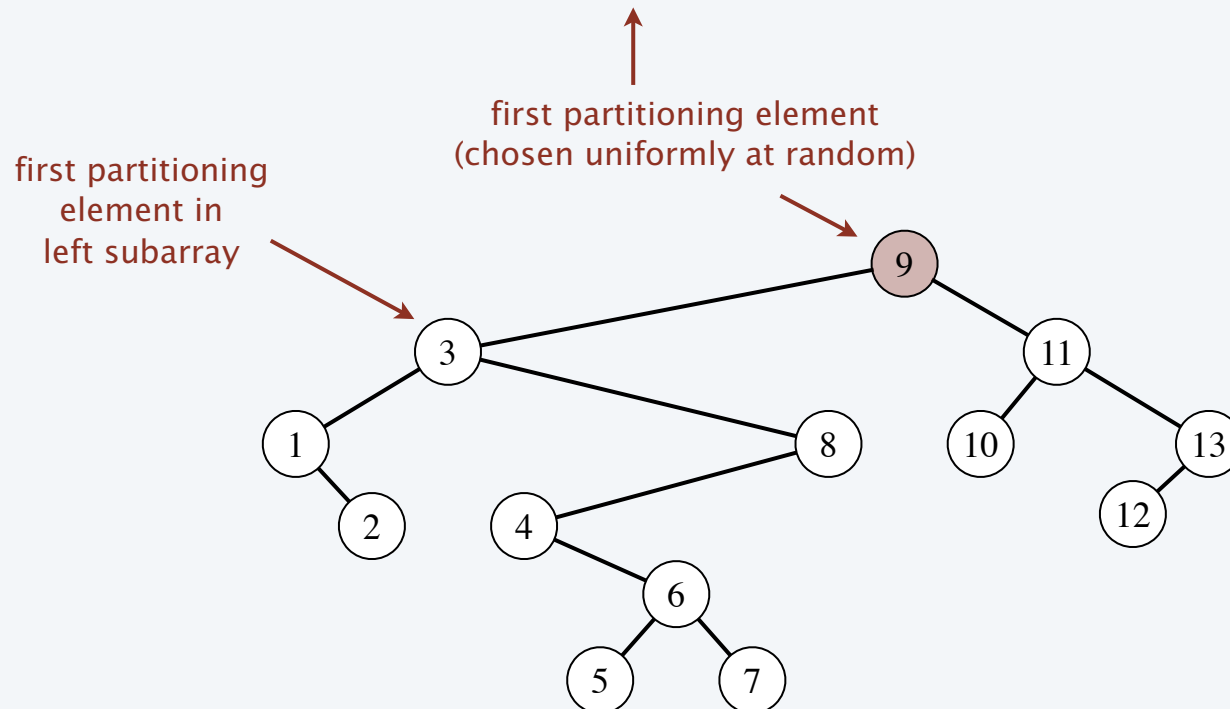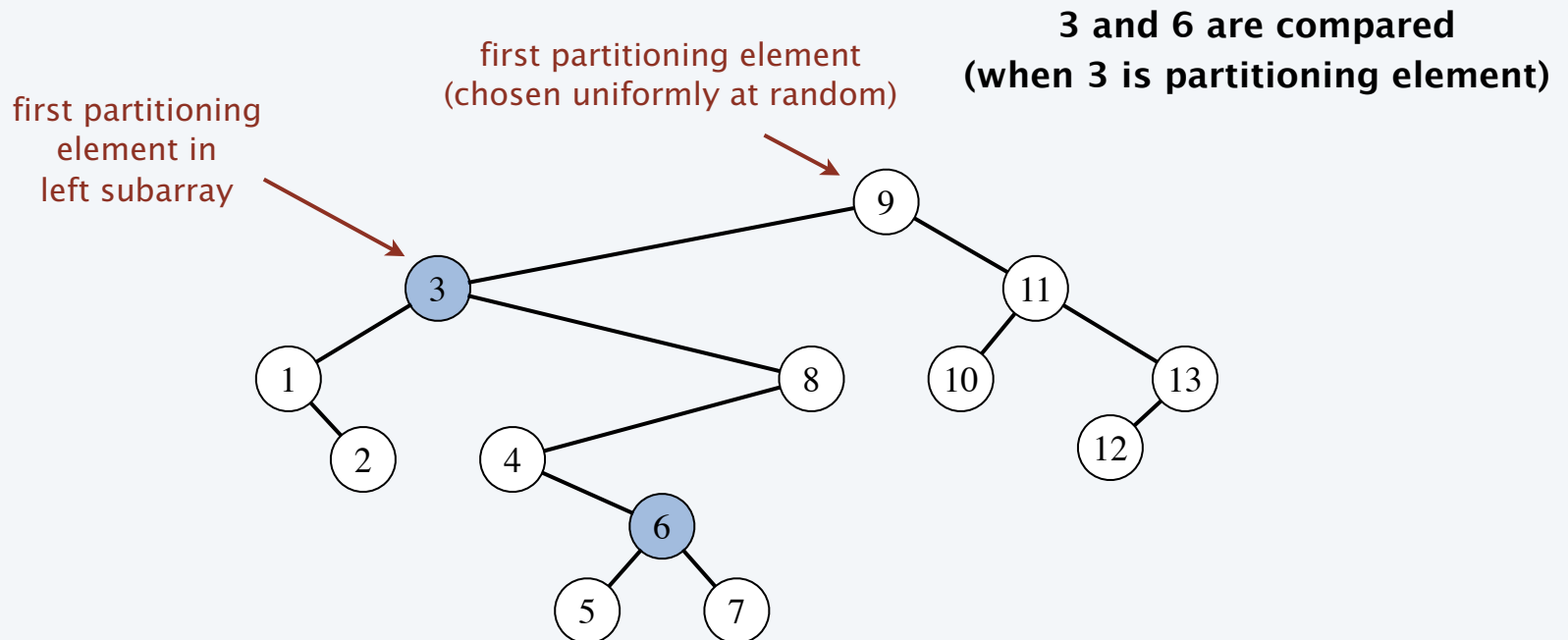(compared if either 2 or 8 are chosen as partition before 3, 4, 5, 6 or 7)

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 $\displaystyle = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{2}{j - i + 1} \ = \ 2 \sum_{i=1}^{N} \sum_{j=2}^{N-i+1} \frac{1}{j}$

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.
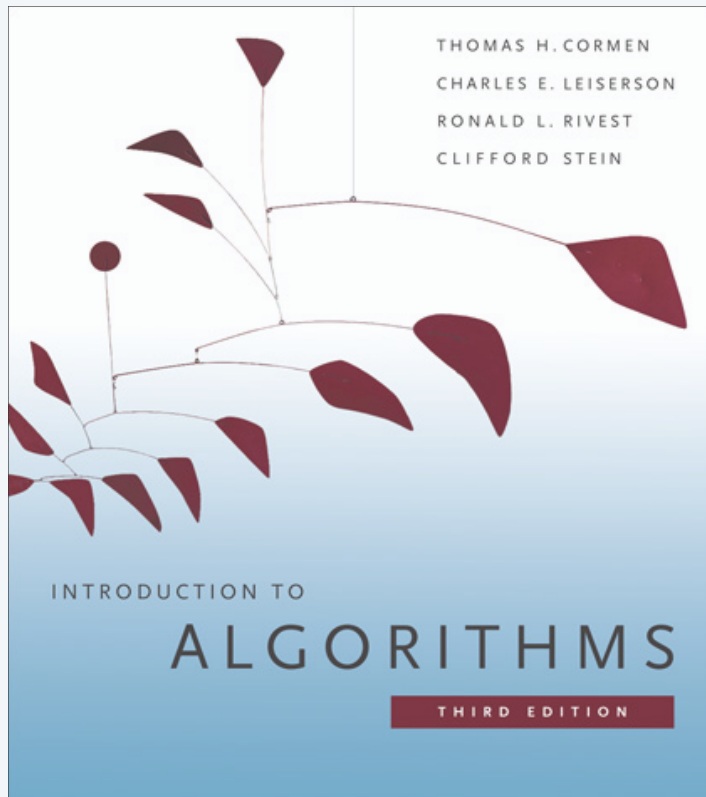
5. DIVIDE AND CONQUER

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

CHAPTER 9

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 kth smallest element.

QUICK-SELECT (A, k)

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

$(L, M, R) \leftarrow$ PARTITION-3-WAY (A, p). $\longleftarrow$ 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 ¾.

$$T(n) \leq T(\tfrac{3}{4}\,n) + n \quad \Rightarrow \quad T(n) \leq 4\,n$$

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

Pf. [by strong induction on n]

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

- Assume true for $1, 2, \ldots, n-1$.

- $T(n)$ satisfies the following recurrence:

$$T(n) \leq n + 2/n\, [\; T(n/2) + \ldots + T(n-3) + T(n-2) + T(n-1) \;]$$

$$\leq n + 2/n\, [\; 4\,n/2 + \ldots + 4(n-3) + 4(n-2) + 4(n-1) \;]$$

$$= n + 4\,(3/4\,n)$$

$$= 4\,n. \quad \blacksquare$$

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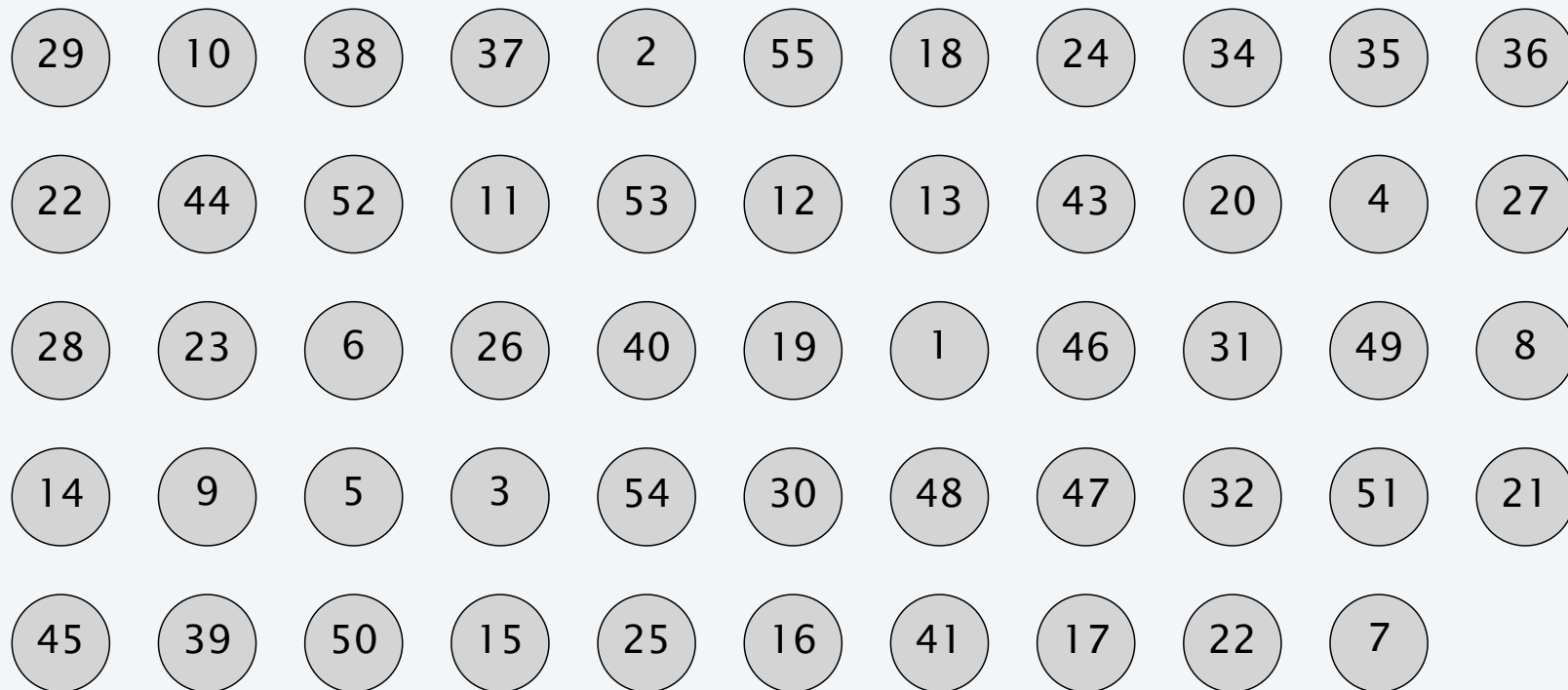
- Divide n elements into $\lfloor n/5 \rfloor$ groups of 5 elements each (plus extra).

29	10	38	37	2	55	18	24	34	35	36
22	44	52	11	53	12	13	43	20	4	27
28	23	6	26	40	19	1	46	31	49	8
14	9	5	3	54	30	48	47	32	51	21
45	39	50	15	25	16	41	17	22	7	

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).

medians

29	10	**38**	37	2	55	**18**	24	34	**35**	36
22	44	52	11	53	12	13	**43**	20	4	27
28	**23**	6	26	**40**	**19**	1	46	**31**	49	8
14	9	5	3	54	30	48	47	32	51	21
45	39	50	**15**	25	16	41	17	22	7	

N = 54
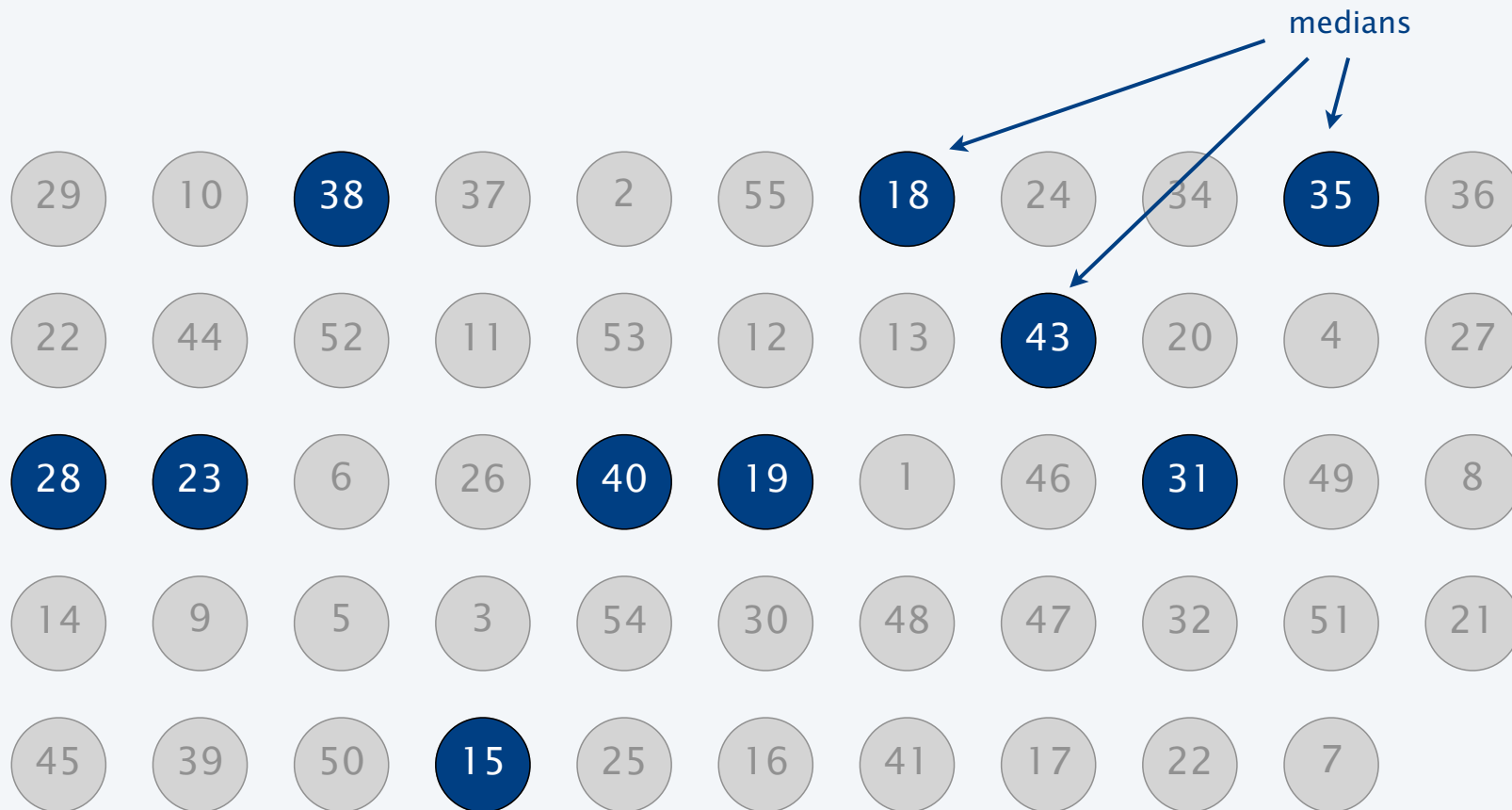
Choosing the pivot element

- 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.

medians

median of medians

| 29 | 10 | 38 | 37 | 2 | 55 | 18 | 24 | 34 | 35 | 36 |

| 22 | 44 | 52 | 11 | 53 | 12 | 13 | 43 | 20 | 4 | 27 |

| 28 | 23 | 6 | 26 | 40 | 19 | 1 | 46 | 31 | 49 | 8 |

| 14 | 9 | 5 | 3 | 54 | 30 | 48 | 47 | 32 | 51 | 21 |

| 45 | 39 | 50 | 15 | 25 | 16 | 41 | 17 | 22 | 7 |

N = 54
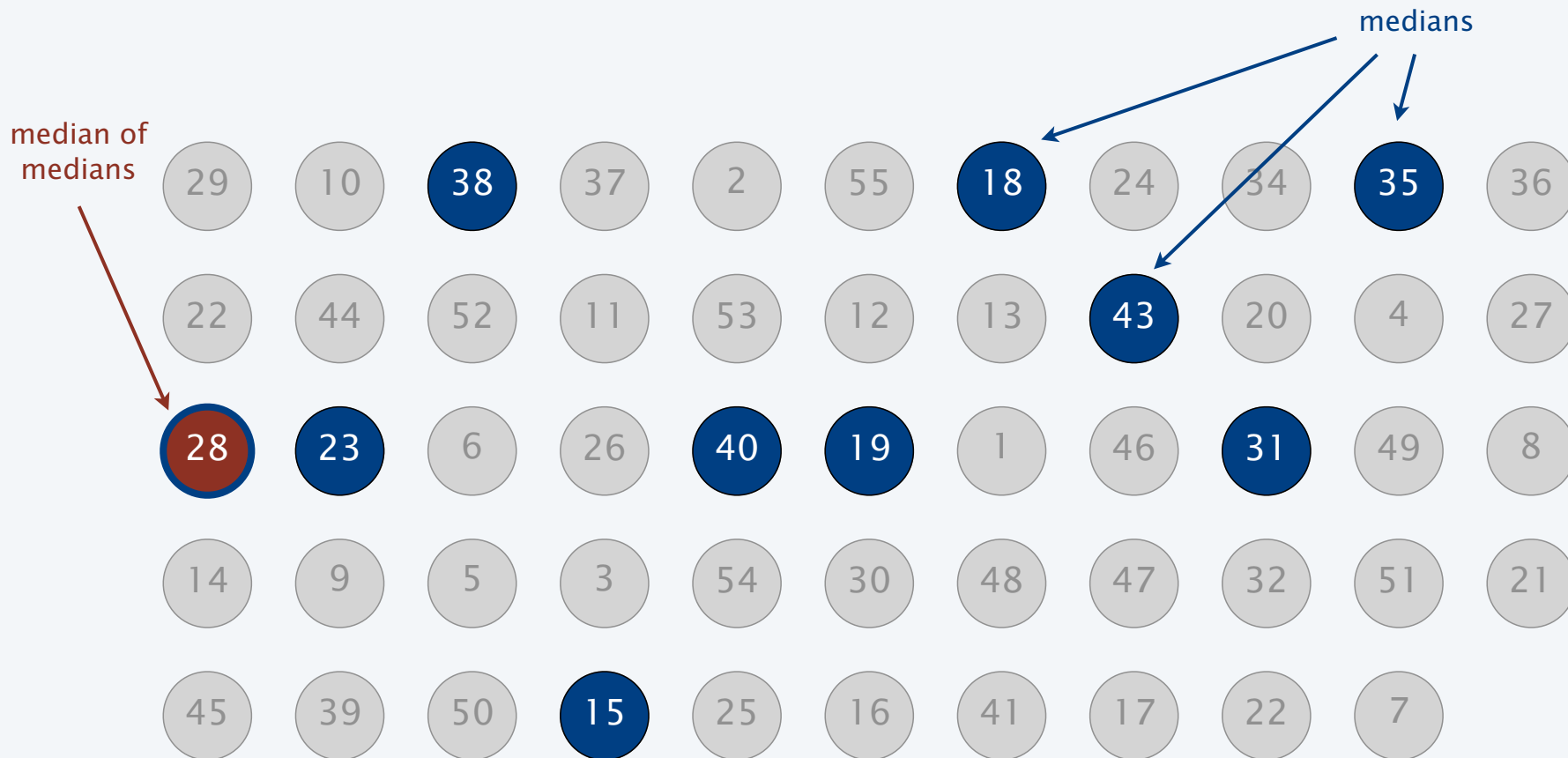
Median-of-medians selection algorithm

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$(B, \lfloor n / 10 \rfloor)$ ⟵——— median of medians

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

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

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

ELSE $\quad\quad\quad\quad\quad\quad$ RETURN p.

Analysis of median-of-medians selection algorithm

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

median of
medians 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 $\leq p$.

median of
medians p

29 10 38 37 2 55 **18** 24 34 35 36

22 44 52 11 53 12 13 43 20 4 27

28 **23** 6 26 40 **19** 1 46 31 49 8

14 9 5 3 54 30 48 47 32 51 21

45 39 50 **15** 25 16 41 17 22 7

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 $\leq p$.
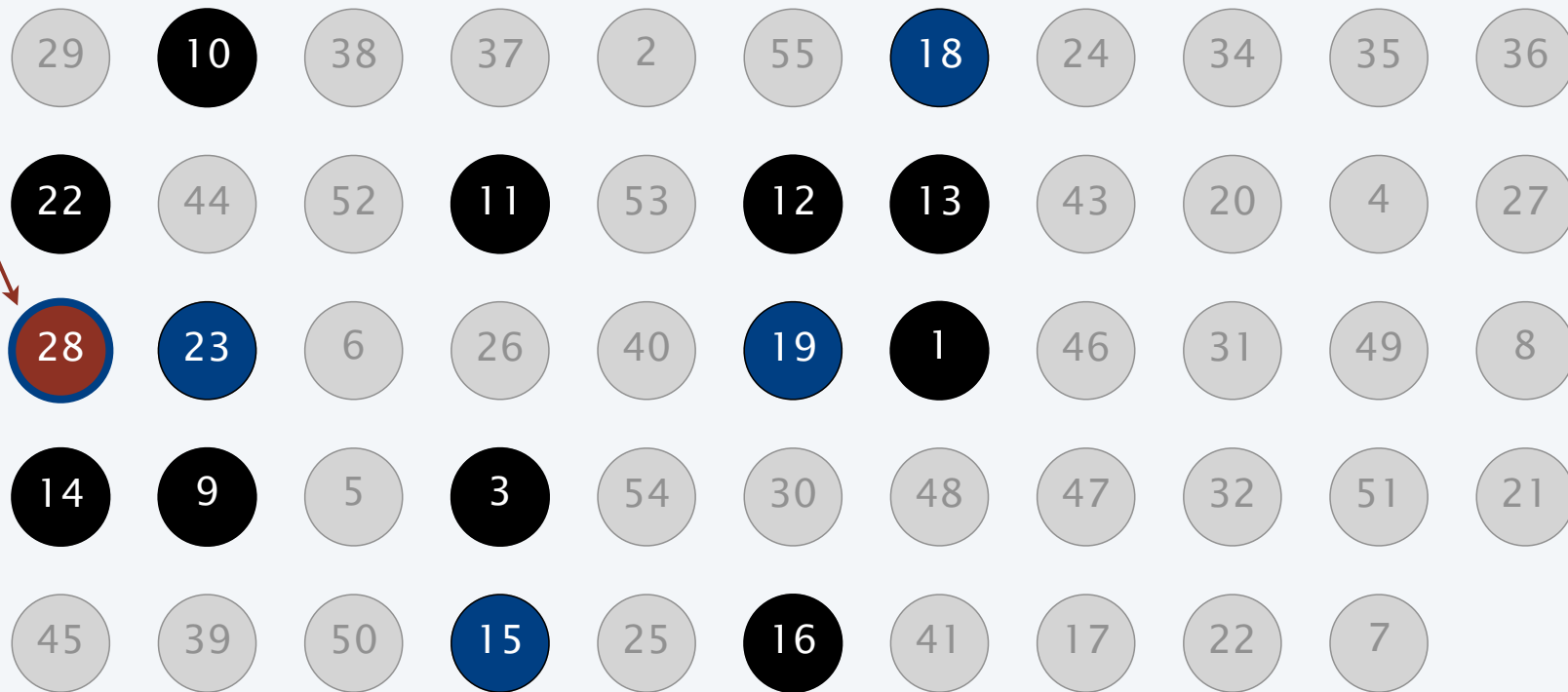- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$.

median of medians p

N = 54

Analysis of median-of-medians selection algorithm

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

median of medians 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 $\geq p$.

median of
medians p

29 10 38 37 2 55 18 24 34 35 36

22 44 52 11 53 12 13 43 20 4 27

28 23 6 26 40 19 1 46 31 49 8

14 9 5 3 54 30 48 47 32 51 21

45 39 50 15 25 16 41 17 22 7
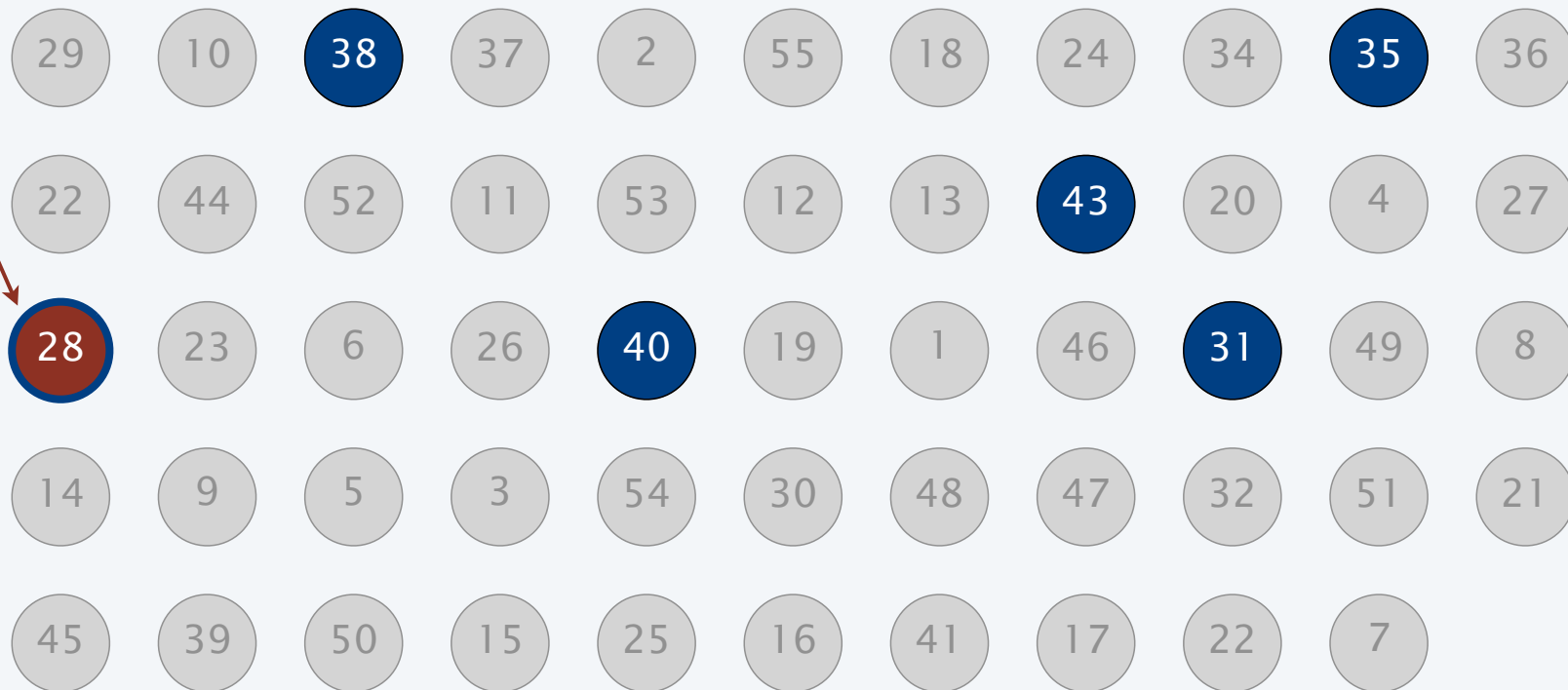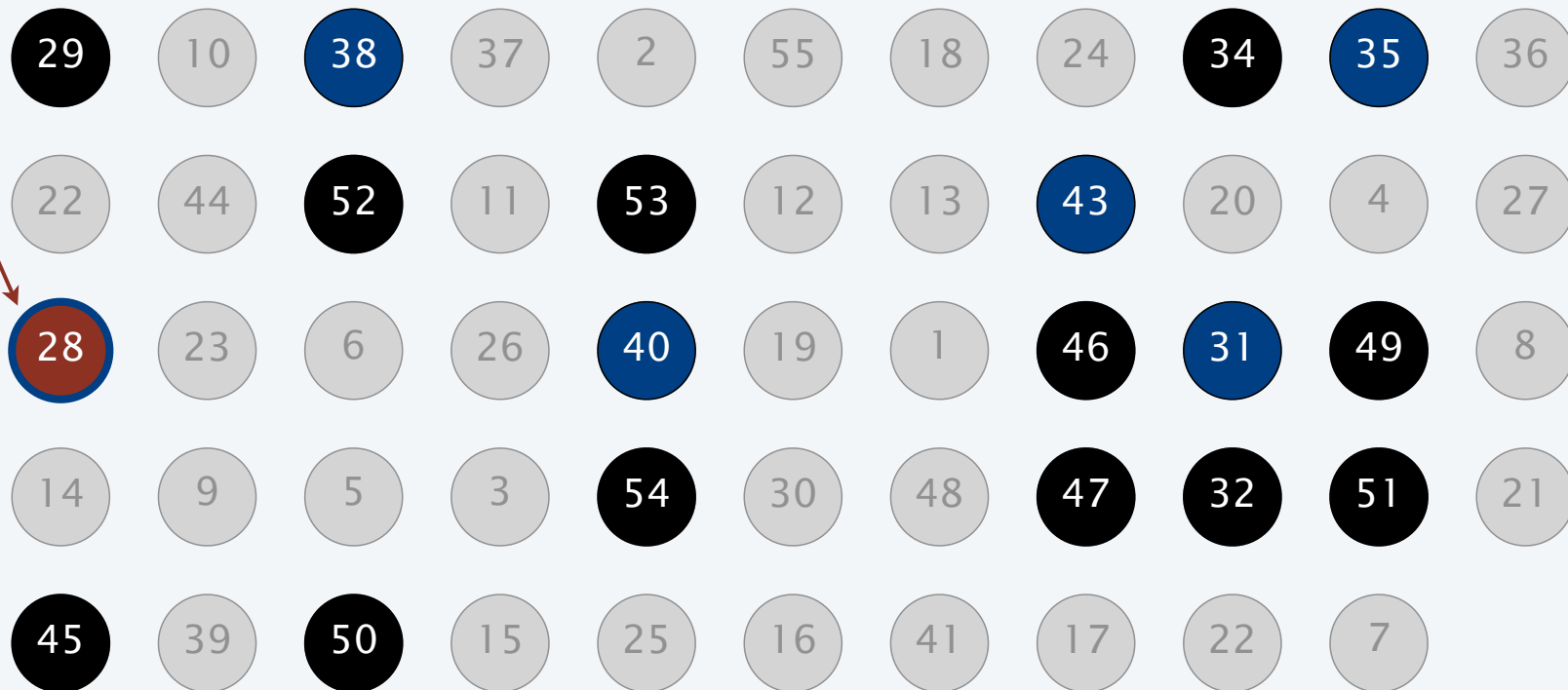
N = 54

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 $\geq p$.

median of medians 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.

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3 \lfloor n/10 \rfloor) + \tfrac{11}{5}\, n$$

median of medians   recursive select   computing median of 5 (6 compares per group)

partitioning (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!

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

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

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

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

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

    5.4305 n comparisons are ever required. This bound is improved for
```
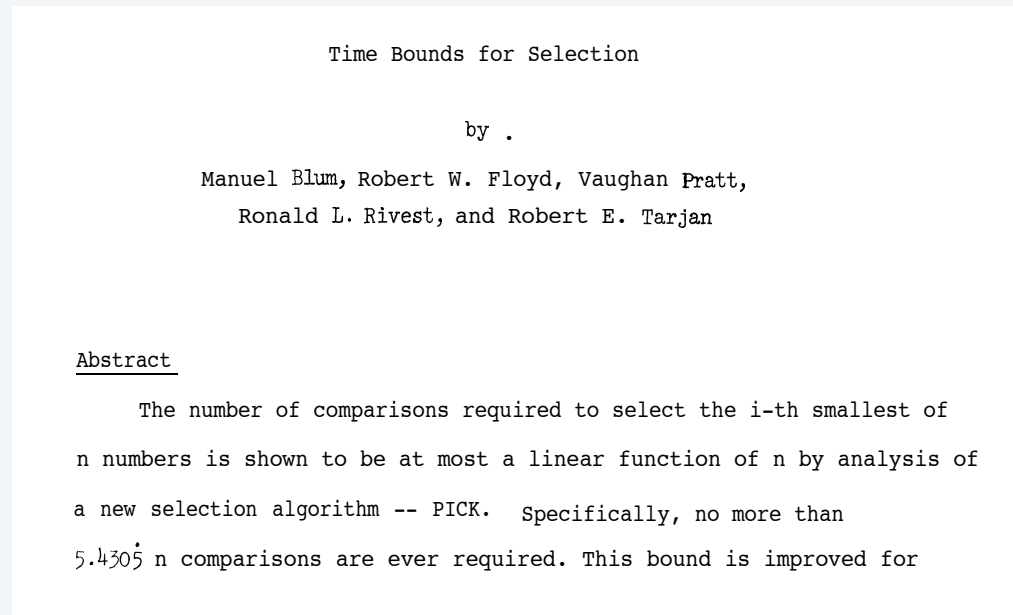
Practice. Constant and overhead (currently) too large to be useful.

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