

5. DIVIDE AND CONQUER I

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *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 (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

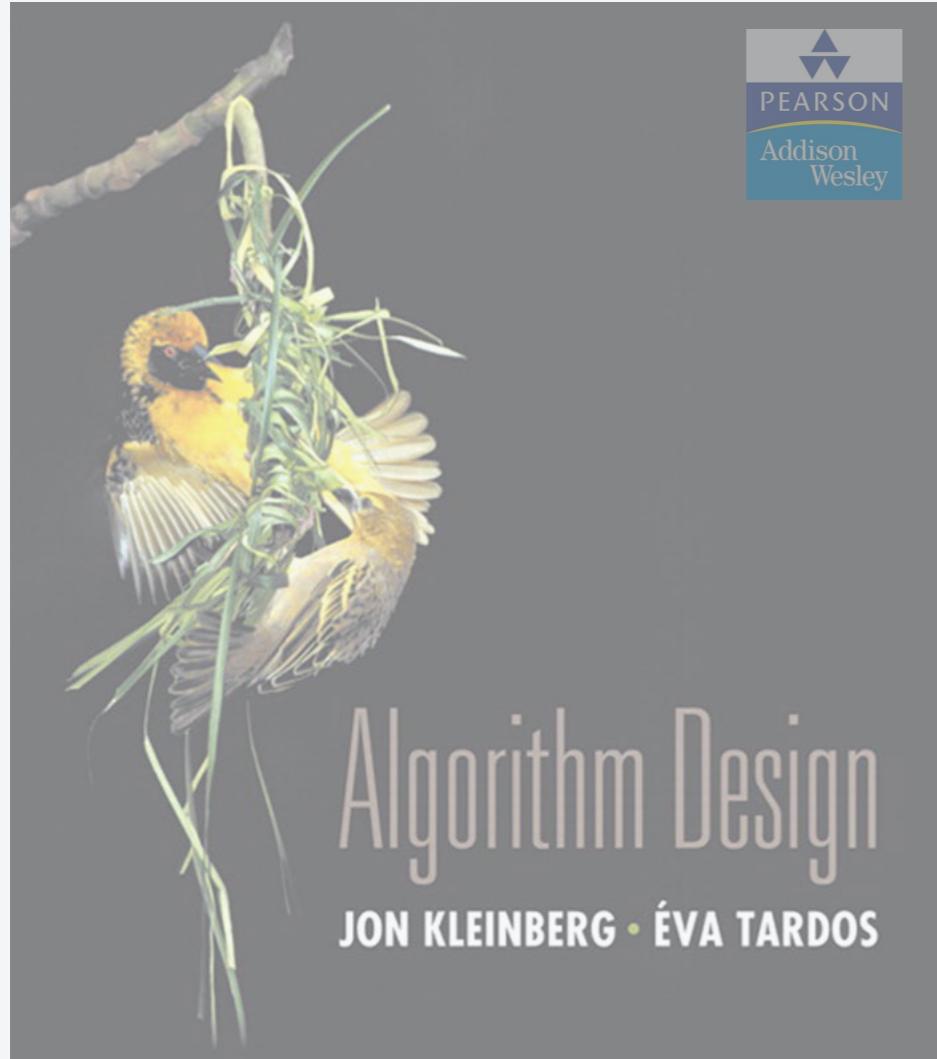
- Divide problem of size n into two subproblems of size $n/2$. $\leftarrow O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. $\leftarrow O(n)$ time

Consequence.

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



attributed to Julius Caesar



5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*

SECTIONS 5.1–5.2

Sorting problem

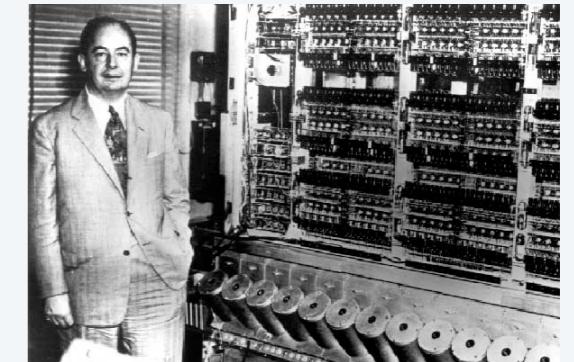
Problem. Given a list L of n elements from a totally ordered universe, rearrange them in ascending order.

The image shows a computer screen displaying a music player application. At the top, there is a horizontal bar showing several album covers, including "Born In The U.S.A." by Bruce Springsteen, "Meat Loaf And Friends" by Meat Loaf, and "Born To Run" by Bruce Springsteen. Below this is a list of songs in a table format:

| | 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 | Bonnie 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"/> Turn! Turn! Turn! (To Everything) | The Byrds | 2:57 | Forrest Gump: The Soundtrack (Disc 2) |

Mergesort

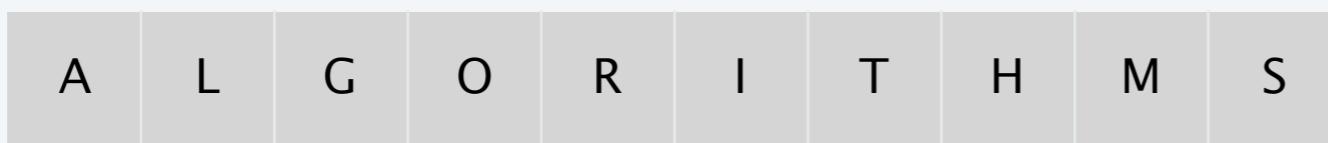
- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.



**First Draft
of a
Report on the
EDVAC**

John von Neumann

input



sort left half



sort right half



merge results



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 | 20 | 23 |
|---|----|-------|----|----|



merge to form sorted list C

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



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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

sorted list B

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

Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 2

sorted list C

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 3

sorted list C

| | | | | | | | | | |
|---|--|--|--|--|--|--|--|--|--|
| 2 | | | | | | | | | |
|---|--|--|--|--|--|--|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 7

sorted list C

| | | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|
| 2 | 3 | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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

sorted list B

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



compare minimum entry in each list: copy 10

sorted list C

| | | | | | | | | | |
|---|---|---|--|--|--|--|--|--|--|
| 2 | 3 | 7 | | | | | | | |
|---|---|---|--|--|--|--|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 11

sorted list C

| | | | | | | | | |
|---|---|---|----|--|--|--|--|--|
| 2 | 3 | 7 | 10 | | | | | |
|---|---|---|----|--|--|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 14

sorted list C

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



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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

sorted list B

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



compare minimum entry in each list: copy 16

sorted list C

| | | | | | | | | |
|---|---|---|----|----|----|--|--|--|
| 2 | 3 | 7 | 10 | 11 | 14 | | | |
|---|---|---|----|----|----|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 18

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|--|--|--|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | | | |
|---|---|---|----|----|----|----|--|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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

sorted list B

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



list A exhausted: copy 20

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|----|--|--|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 18 | | |
|---|---|---|----|----|----|----|----|--|--|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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

sorted list B

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



list A exhausted: copy 23

sorted list C

| | | | | | | | | |
|---|---|---|----|----|----|----|----|----|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 18 | 20 |
|---|---|---|----|----|----|----|----|----|



Merge demo

Given two sorted lists A and B , merge into sorted list C .

sorted list A

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

sorted list B

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



done

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 18 | 20 | 23 |
|---|---|---|----|----|----|----|----|----|----|



Mergesort implementation

Input. List L of n elements from a totally ordered universe.

Output. The n elements in ascending order.

MERGE-SORT(L)

IF (list L has one element)

RETURN L .

Divide the list into two halves A and B .

$A \leftarrow \text{MERGE-SORT}(A).$ $\longleftarrow T(n / 2)$

$B \leftarrow \text{MERGE-SORT}(B).$ $\longleftarrow T(n / 2)$

$L \leftarrow \text{MERGE}(A, B).$ $\longleftarrow \Theta(n)$

RETURN L .

A useful recurrence relation

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

Recurrence.

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


between $\lfloor n / 2 \rfloor$ and $n - 1$ compares

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

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

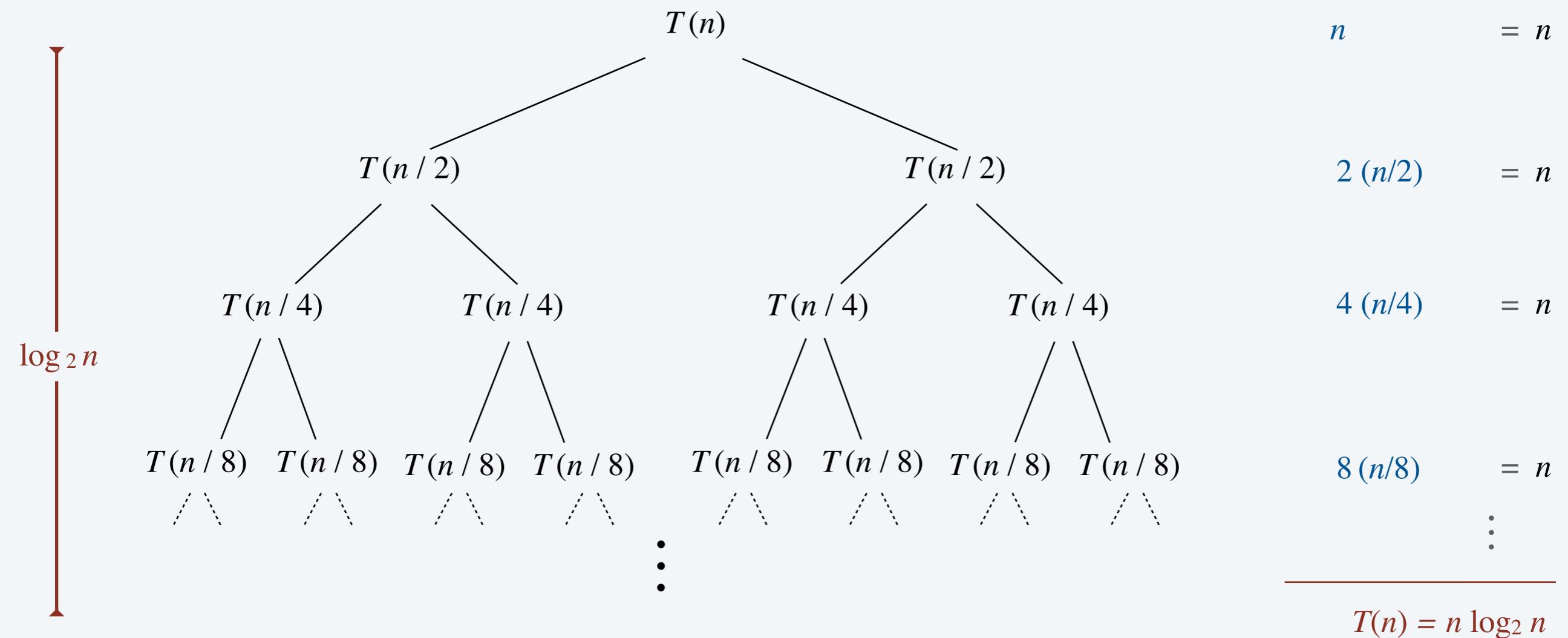
Initially we assume n is a power of 2 and replace \leq with $=$ in the recurrence.

Divide-and-conquer recurrence: recursion tree

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

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

assuming n
is a power of 2



Proof by induction

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

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

assuming n
is a power of 2

Pf. [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 $T(2n) = 2n \log_2 (2n)$.

$$T(2n) = 2T(n) + 2n$$

↑
recurrence

$$\begin{aligned} \text{inductive hypothesis } \longrightarrow &= 2n \log_2 n + 2n \\ &= 2n(\log_2(2n) - 1) + 2n \\ &= 2n \log_2(2n). \blacksquare \end{aligned}$$



Which is the exact solution of the following recurrence?

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

 no longer assuming n is a power of 2

- A. $T(n) = n \lfloor \log_2 n \rfloor$
- B. $T(n) = n \lceil \log_2 n \rceil$
- C. $T(n) = n \lfloor \log_2 n \rfloor + 2^{\lfloor \log_2 n \rfloor} - 1$
- D. $T(n) = n \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$
- E. Not even Knuth knows.

Analysis of mergesort recurrence

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

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

↑
no longer assuming n
is a power of 2

Pf. [by strong induction on n]

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

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ \text{inductive hypothesis } \rightarrow &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n \lceil \log_2 n_2 \rceil + n \\ &\leq n (\lceil \log_2 n \rceil - 1) + n \quad \leftarrow \text{log}_2 n_2 \leq \lceil \log_2 n \rceil - 1 \\ &= n \lceil \log_2 n \rceil. \blacksquare \end{aligned}$$

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

\uparrow
an integer

Digression: sorting lower bound

Challenge. How to prove a lower bound for **all** conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

A1. Yes. Java, Python, C++, ...

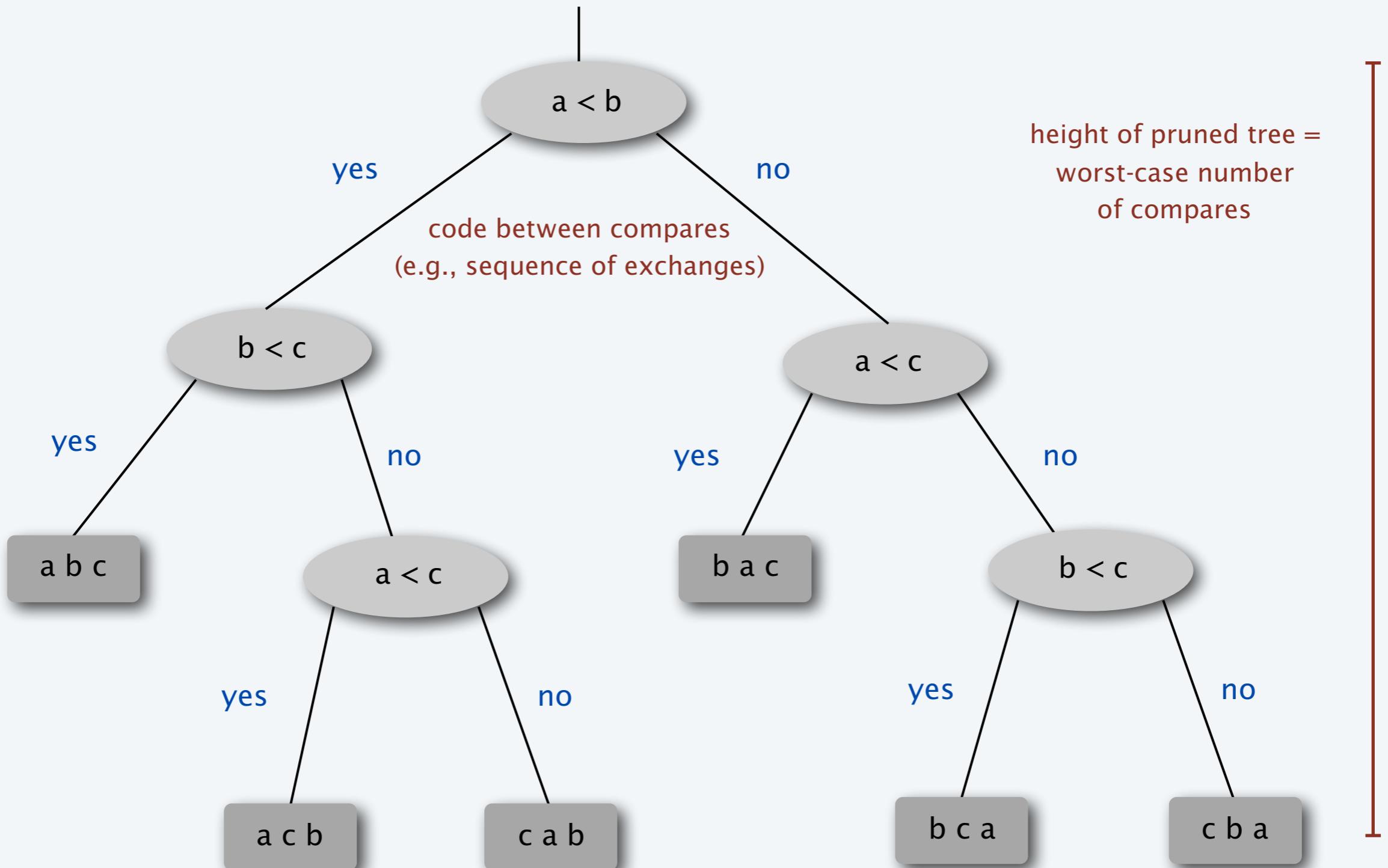
A2. Yes. Mergesort, insertion sort, quicksort, heapsort, ...

A3. No. Bucket sort, radix sorts, ...

sort(*, key=None, reverse=False)

This method sorts the list in place, using only `<` comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

Comparison tree (for 3 distinct keys a, b, and c)



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n .
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow n!$ reachable leaves.



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

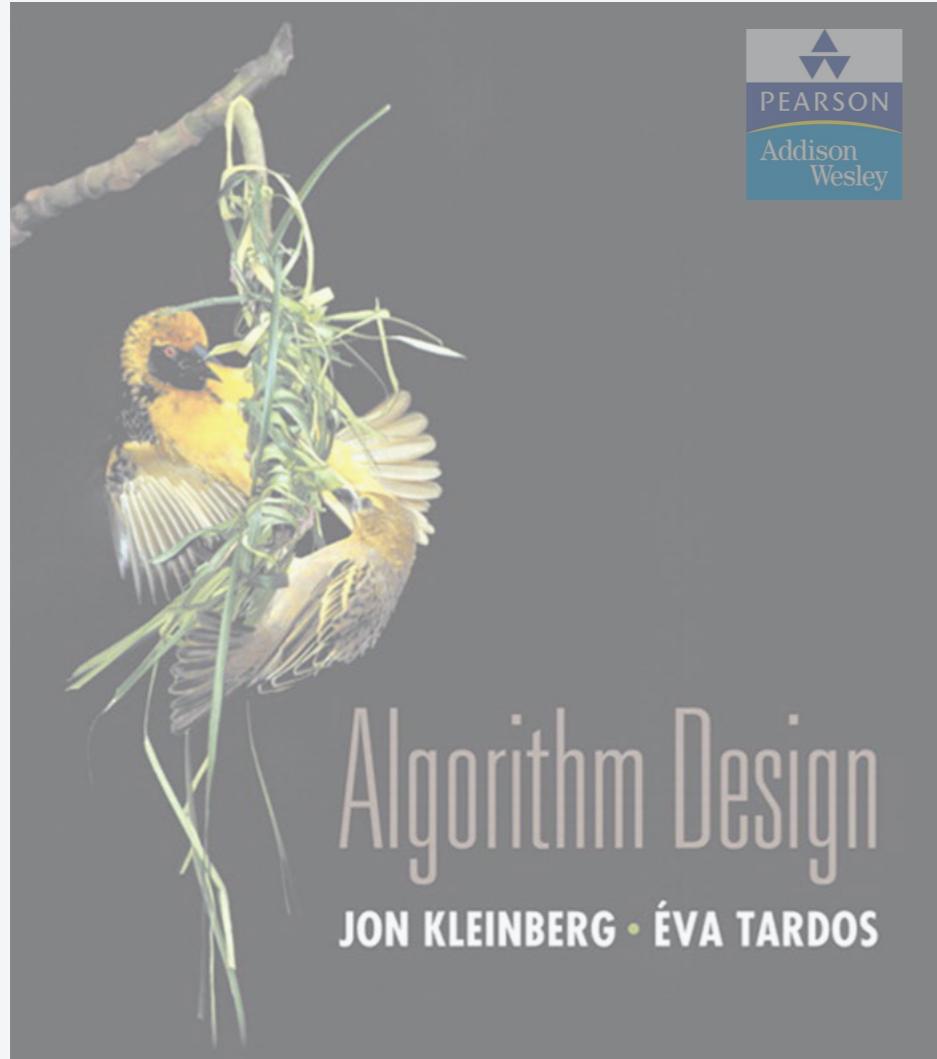
- Assume array consists of n distinct values a_1 through a_n .
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow n!$ reachable leaves.

$$\begin{aligned} 2^h &\geq \# \text{ leaves} \geq n! \\ \Rightarrow h &\geq \log_2(n!) \\ &\geq n \log_2 n - n / \ln 2 \quad \blacksquare \end{aligned}$$

↑
Stirling's formula



Note. Lower bound can be extended to include randomized algorithms.



SECTION 5.3

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *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: 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

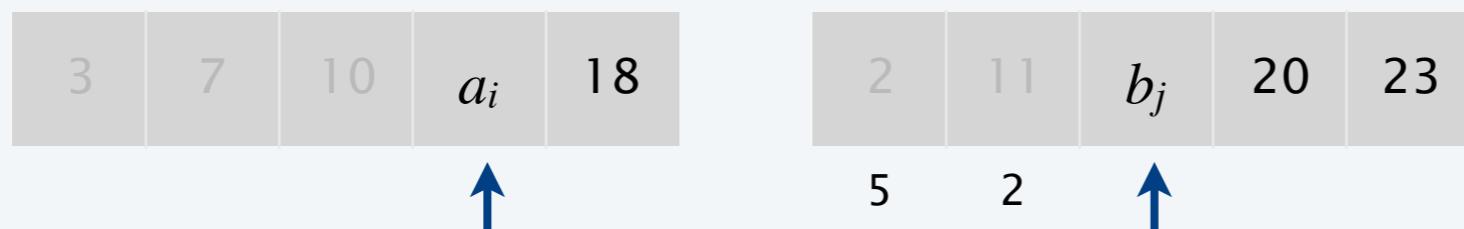
Counting inversions: how to combine two subproblems?

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



Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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

sorted list B

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

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

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



compare minimum entry in each list: copy 2 and add x to inversion count

sorted list C

| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|
| | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|



$x = 5$ ← number of elements remaining in A
inversions = 0

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

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

5



compare minimum entry in each list: copy 3 and decrement x

sorted list C

| | | | | | | | | | |
|---|--|--|--|--|--|--|--|--|--|
| 2 | | | | | | | | | |
|---|--|--|--|--|--|--|--|--|--|



$x = 5$

inversions = 5

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

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

5



compare minimum entry in each list: copy 7 and decrement x

sorted list C

| | | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|
| 2 | 3 | | | | | | | | |
|---|---|--|--|--|--|--|--|--|--|



$x = 4$

inversions = 5

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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

sorted list B

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



compare minimum entry in each list: copy 10 and decrement x

sorted list C

| | | | | | | | | | |
|---|---|---|--|--|--|--|--|--|--|
| 2 | 3 | 7 | | | | | | | |
|---|---|---|--|--|--|--|--|--|--|



$$x = 3$$

$$\text{inversions} = 5$$

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

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

5



compare minimum entry in each list: copy 11 and add x to increment count

sorted list C

| | | | | | | | | | |
|---|---|---|----|--|--|--|--|--|--|
| 2 | 3 | 7 | 10 | | | | | | |
|---|---|---|----|--|--|--|--|--|--|



$x = 2$

inversions = 5

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

| | | | | |
|---|----|----|----|----|
| 2 | 11 | 16 | 20 | 23 |
| 5 | 2 | | | |



compare minimum entry in each list: copy 14 and decrement x

sorted list C

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



$$x = 2$$

$$\text{inversions} = 7$$

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

| | | | | |
|---|----|----|----|----|
| 2 | 11 | 16 | 20 | 23 |
| 5 | 2 | | | |



compare minimum entry in each list: copy 16 and add x to increment count

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|--|--|--|--|
| 2 | 3 | 7 | 10 | 11 | 14 | | | | |
|---|---|---|----|----|----|--|--|--|--|



$$x = 1$$

$$\text{inversions} = 7$$

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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



sorted list B

| | | | | |
|---|----|----|----|----|
| 2 | 11 | 16 | 20 | 23 |
| 5 | 2 | 1 | 20 | ↑ |

compare minimum entry in each list: copy 18 and decrement x

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|--|--|--|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | | | |
|---|---|---|----|----|----|----|--|--|--|



$$x = 1$$

$$\text{inversions} = 8$$

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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

sorted list B

| | | | | |
|---|----|----|----|----|
| 2 | 11 | 16 | 20 | 23 |
| 5 | 2 | 1 | 20 | |



5

2

1



list A exhausted: copy 20

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|----|--|--|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 18 | | |
|---|---|---|----|----|----|----|----|--|--|



$x = 0$

inversions = 8

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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

sorted list B

| | | | | |
|---|----|----|----|----|
| 2 | 11 | 16 | 20 | 23 |
| 5 | 2 | 1 | 0 | |



list A exhausted: copy 23

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|--|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 18 | 20 | |
|---|---|---|----|----|----|----|----|----|--|



$x = 0$

inversions = 8

Merge-and-count demo

Given two sorted lists A and B ,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C .

sorted list A

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

sorted list B

| | | | | |
|---|----|----|----|----|
| 2 | 11 | 16 | 20 | 23 |
| 5 | 2 | 1 | 0 | 0 |



done: return 8 inversions

sorted list C

| | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|
| 2 | 3 | 7 | 10 | 11 | 14 | 16 | 18 | 20 | 23 |
|---|---|---|----|----|----|----|----|----|----|



$x = 0$

inversions = 8

Counting inversions: divide-and-conquer algorithm implementation

Input. List L .

Output. Number of inversions in L and L in sorted order.

SORT-AND-COUNT(L)

IF (list L has one element)

RETURN $(0, L)$.

Divide the list into two halves A and B .

$(r_A, A) \leftarrow \text{SORT-AND-COUNT}(A).$ $\longleftarrow T(n / 2)$

$(r_B, B) \leftarrow \text{SORT-AND-COUNT}(B).$ $\longleftarrow T(n / 2)$

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

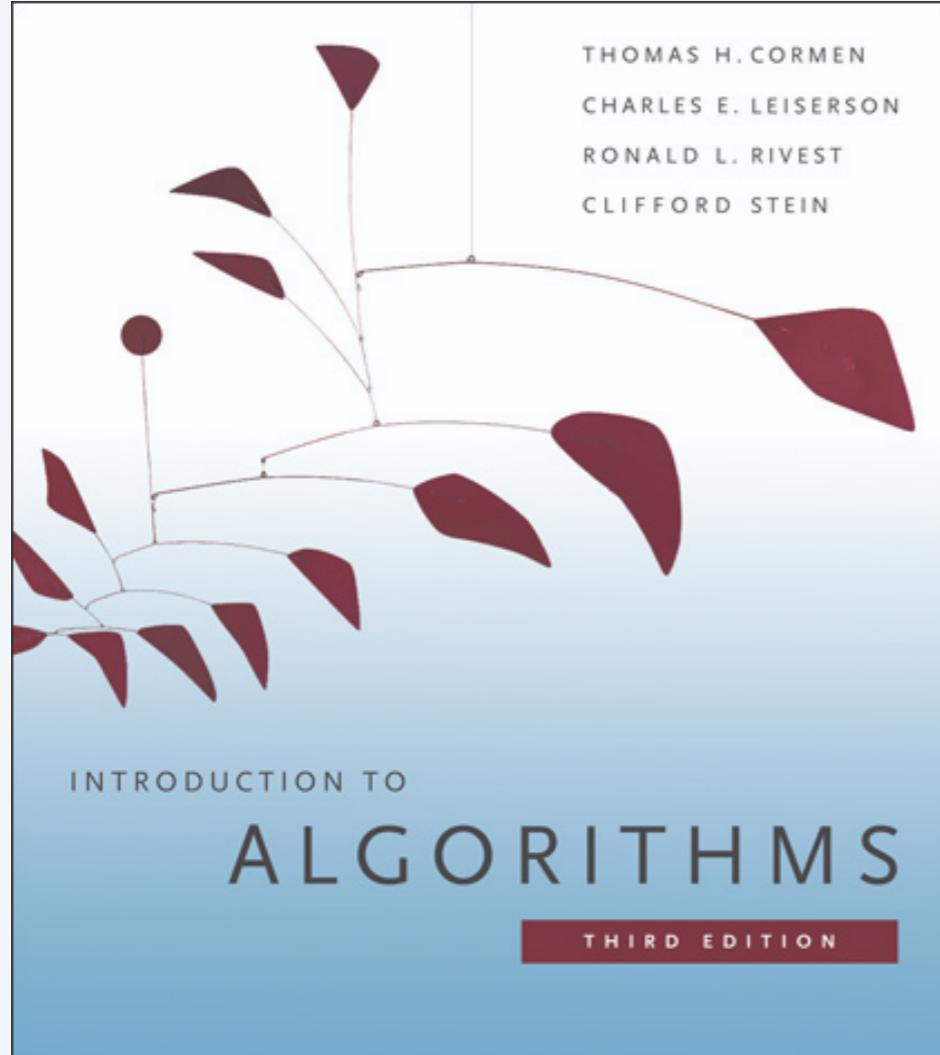
RETURN $(r_A + r_B + r_{AB}, L)$.

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:

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



SECTION 7.1-7.3

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *randomized quicksort*
- ▶ *median and selection*

3-WAY PARTITIONING



Goal. Given an array A and pivot element p , partition array so that:

- Smaller elements in left subarray L .
- Equal elements in middle subarray M .
- Larger elements in right subarray R .

Challenge. $O(n)$ time and $O(1)$ space.



the array A

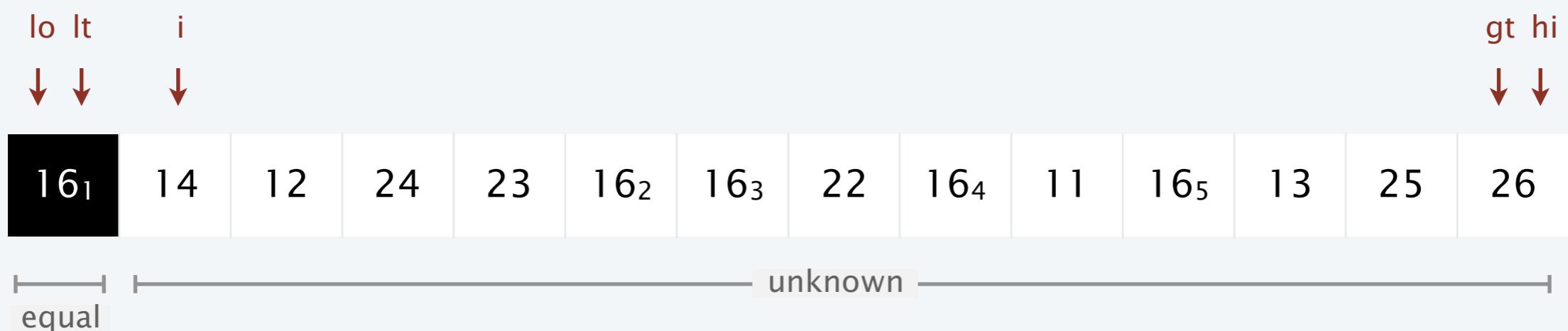


the partitioned array A



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



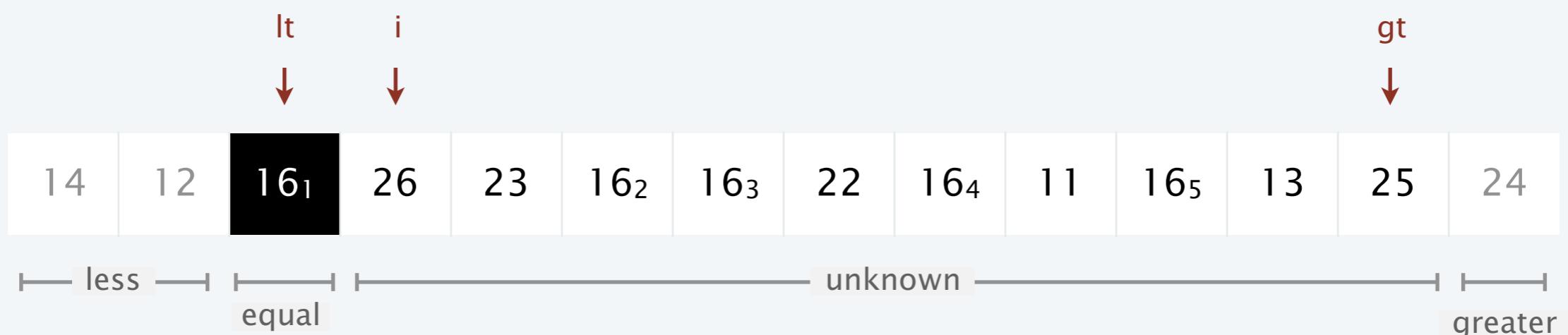
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



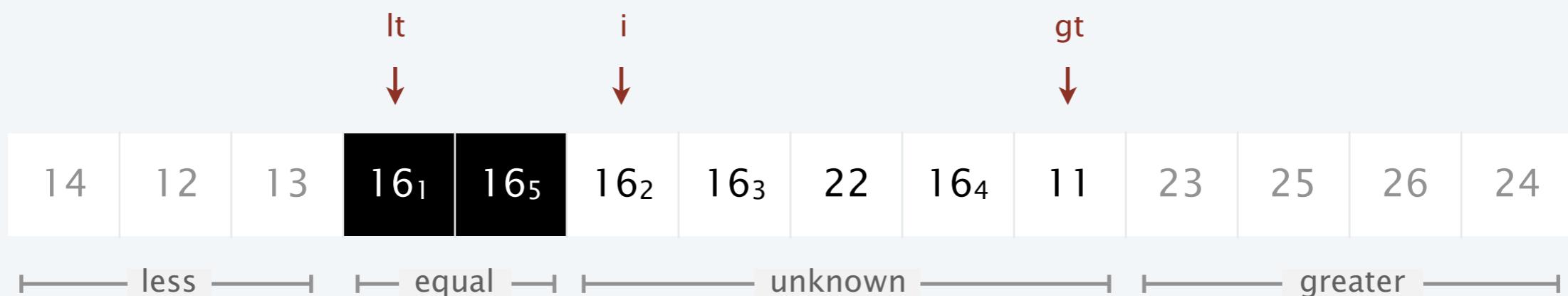
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



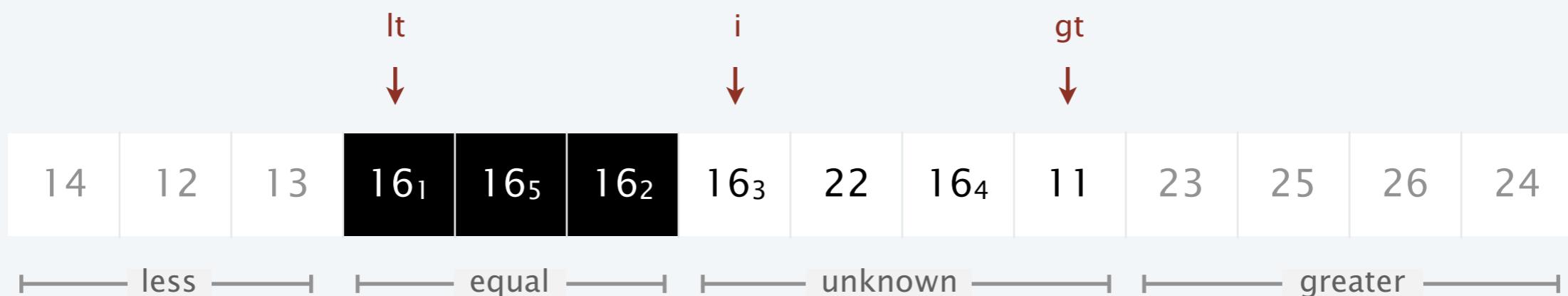
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



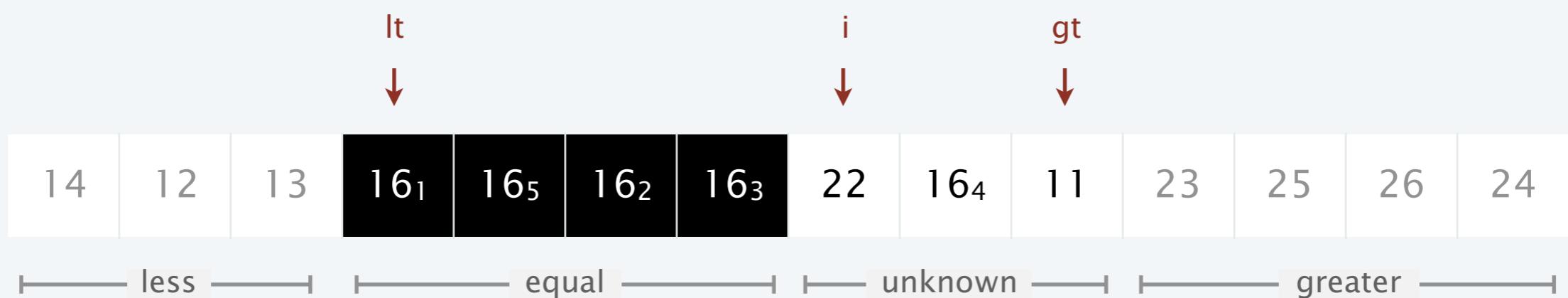
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



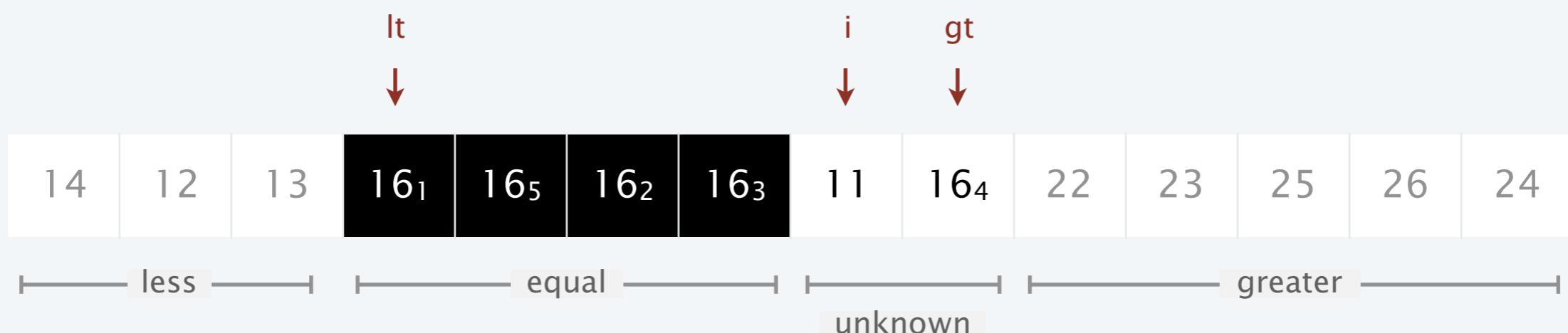
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



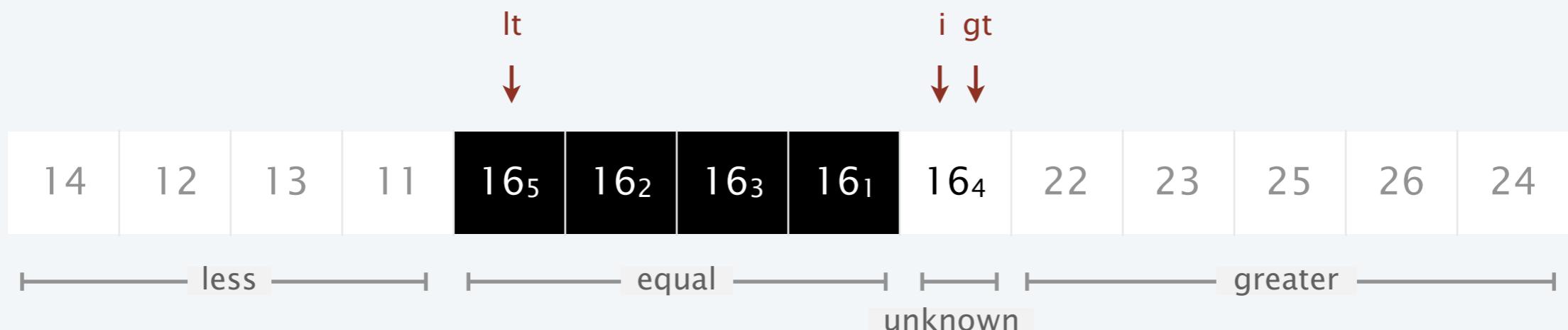
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



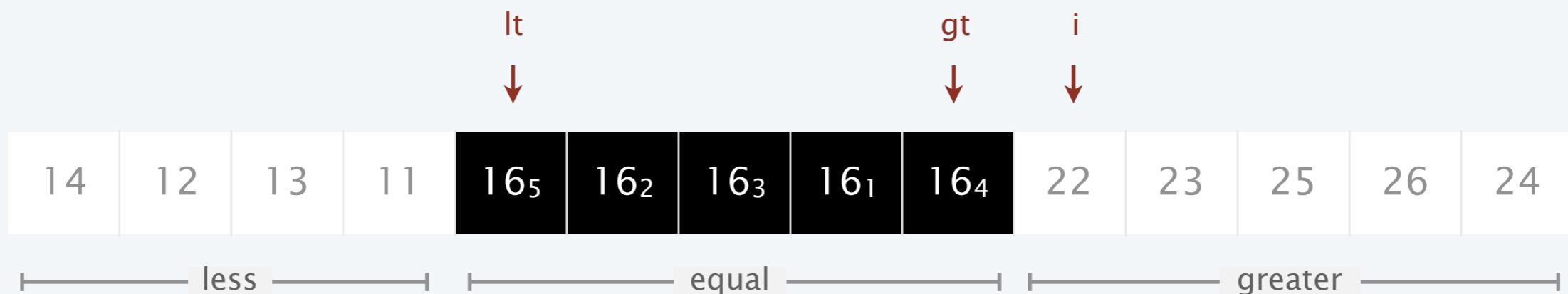
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



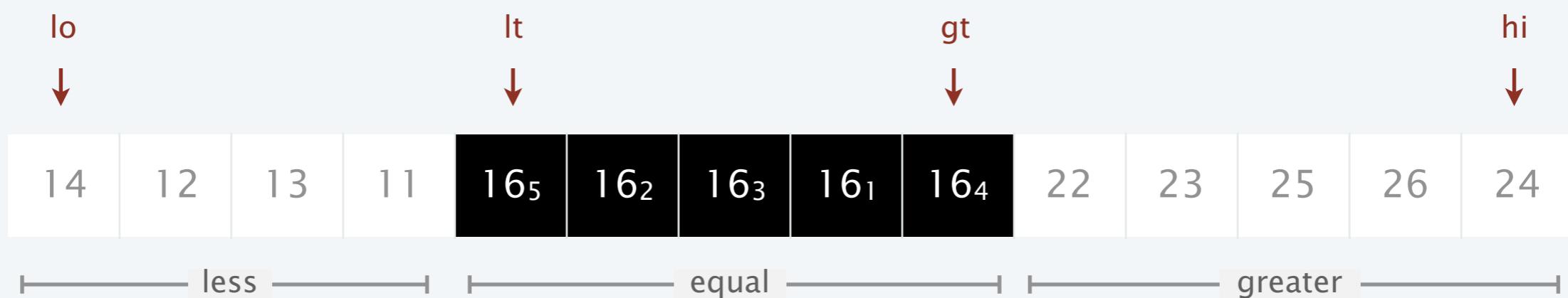
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo .
- Scan i from left to right.
 - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
 - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
 - $(A[i] = p)$: increment i



Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recursively sort both L and R .



Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recursively sort both L and R .

RANDOMIZED-QUICKSORT(A)

IF (array A has zero or one element)

RETURN.

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

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, p)$. $\longleftarrow \Theta(n)$

$\text{RANDOMIZED-QUICKSORT}(L)$. $\longleftarrow T(i)$

$\text{RANDOMIZED-QUICKSORT}(R)$. $\longleftarrow T(n - i - 1)$

[new analysis required
 i is a random variable—depends on p]

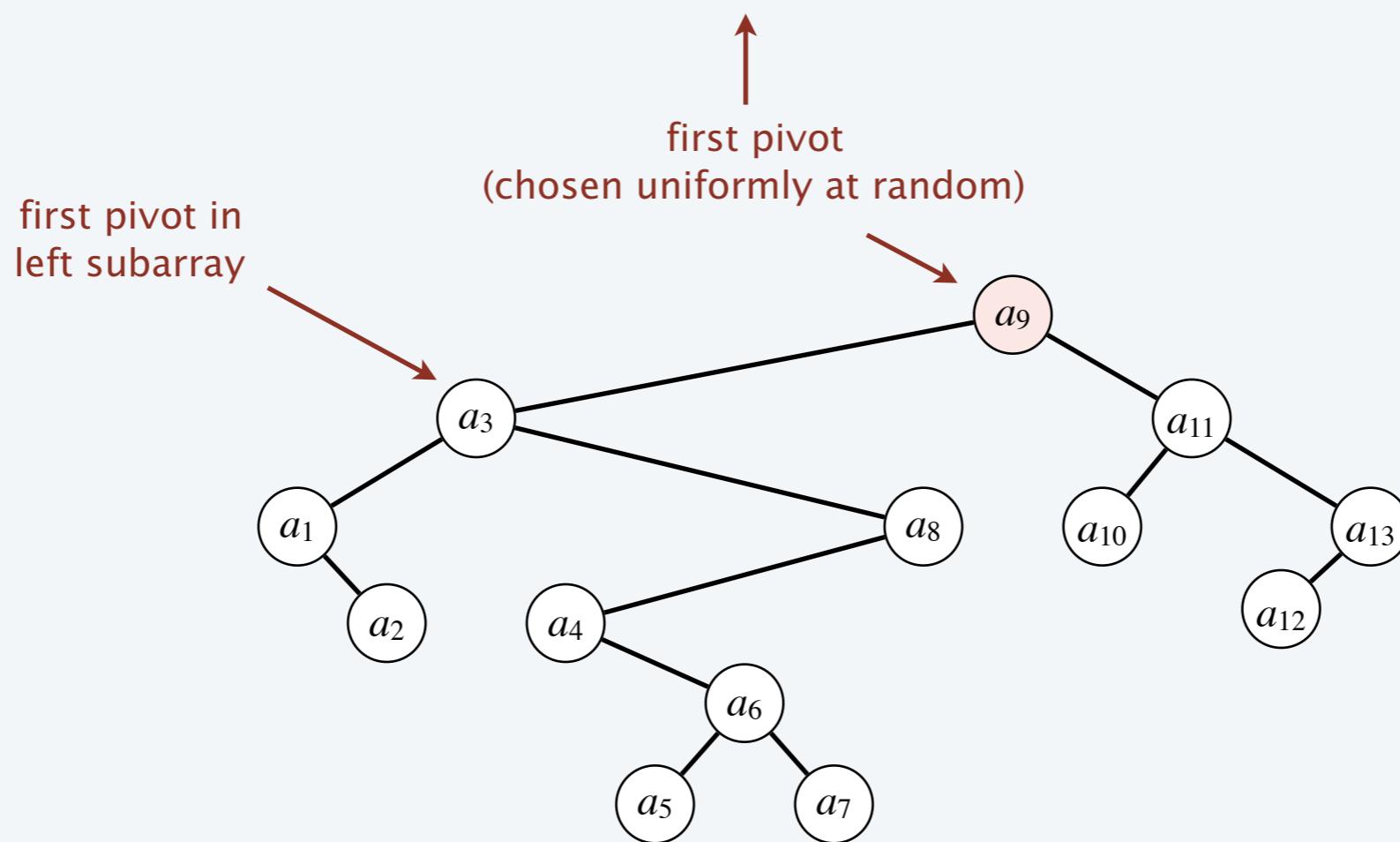
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

the original array of elements A

| | | | | | | | | | | | | |
|-------|-------|----------|-------|----------|-------|-------|-------|-------|----------|-------|----------|-------|
| a_7 | a_6 | a_{12} | a_3 | a_{11} | a_8 | a_9 | a_1 | a_4 | a_{10} | a_2 | a_{13} | a_5 |
|-------|-------|----------|-------|----------|-------|-------|-------|-------|----------|-------|----------|-------|

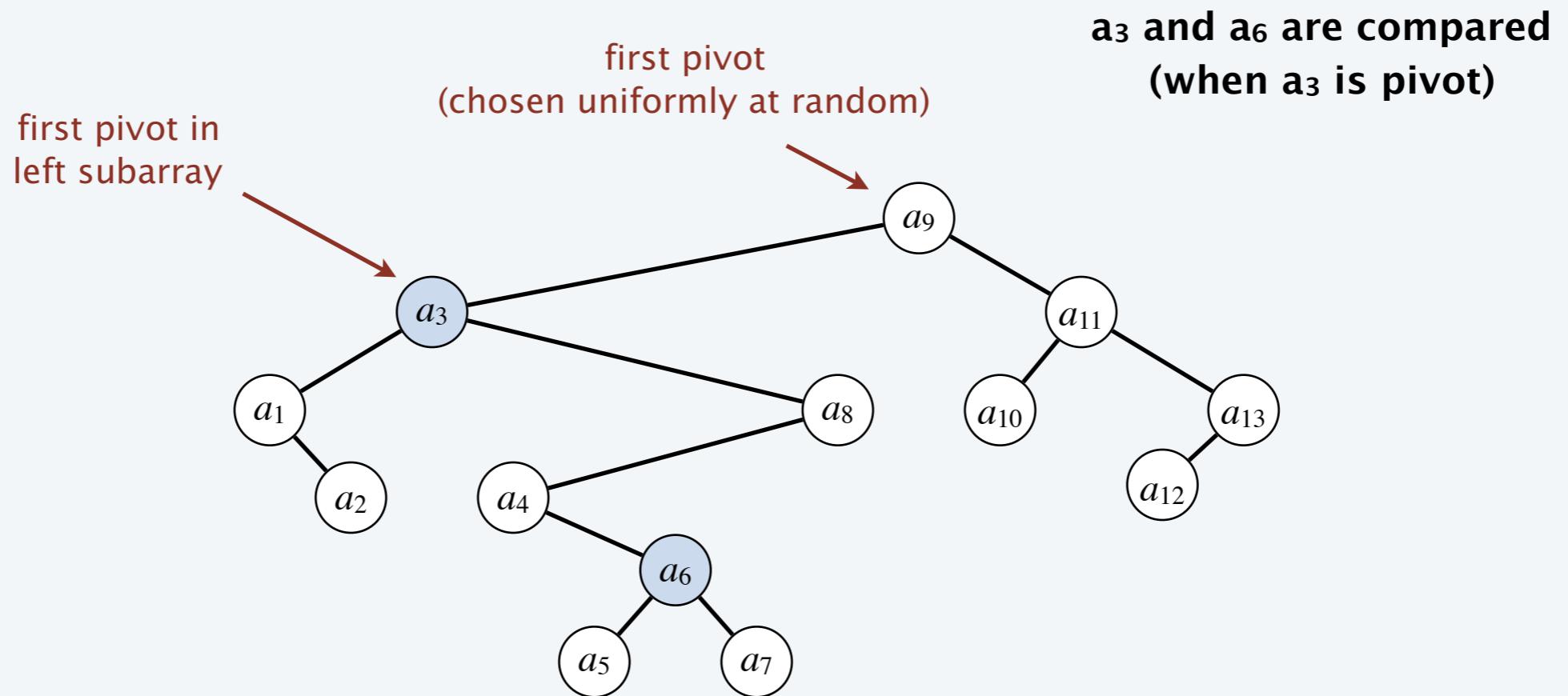


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.

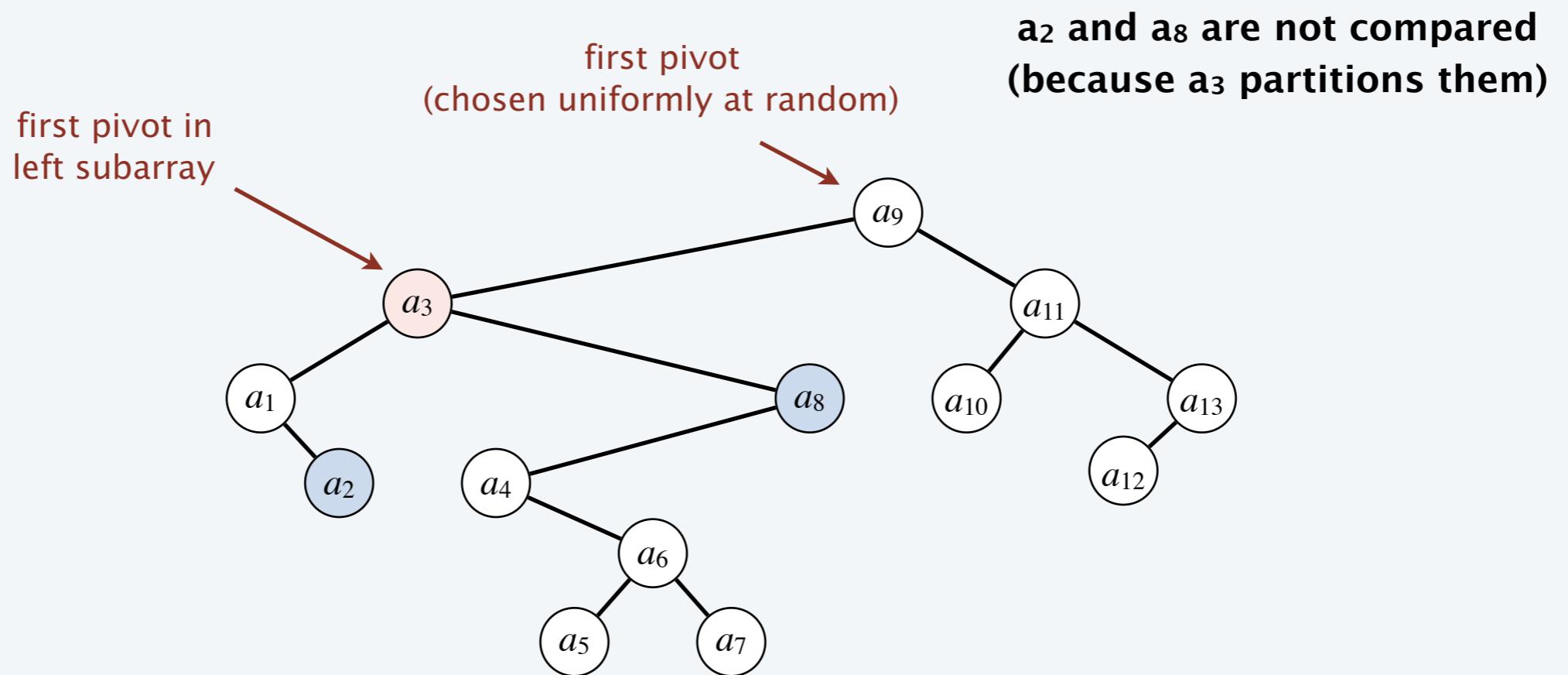


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

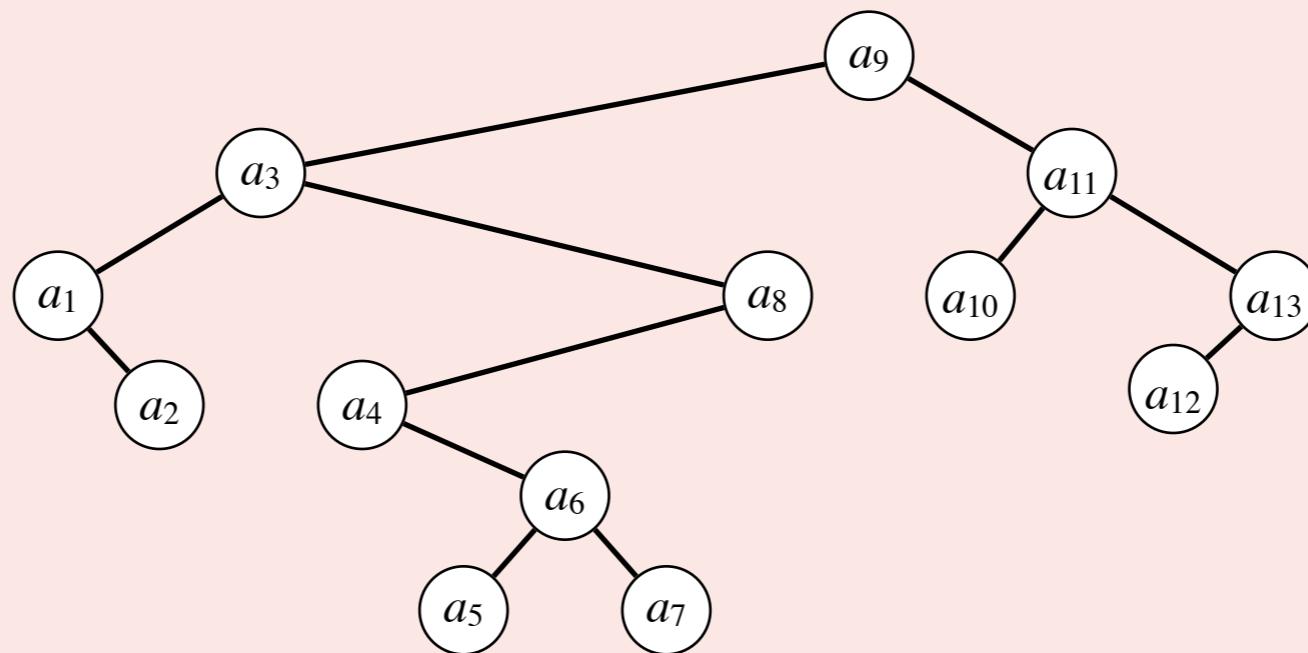
- a_i and a_j are compared once iff one is an ancestor of the other.





Given an array of $n \geq 8$ distinct elements $a_1 < a_2 < \dots < a_n$, what is the probability that a_7 and a_8 are compared during randomized quicksort?

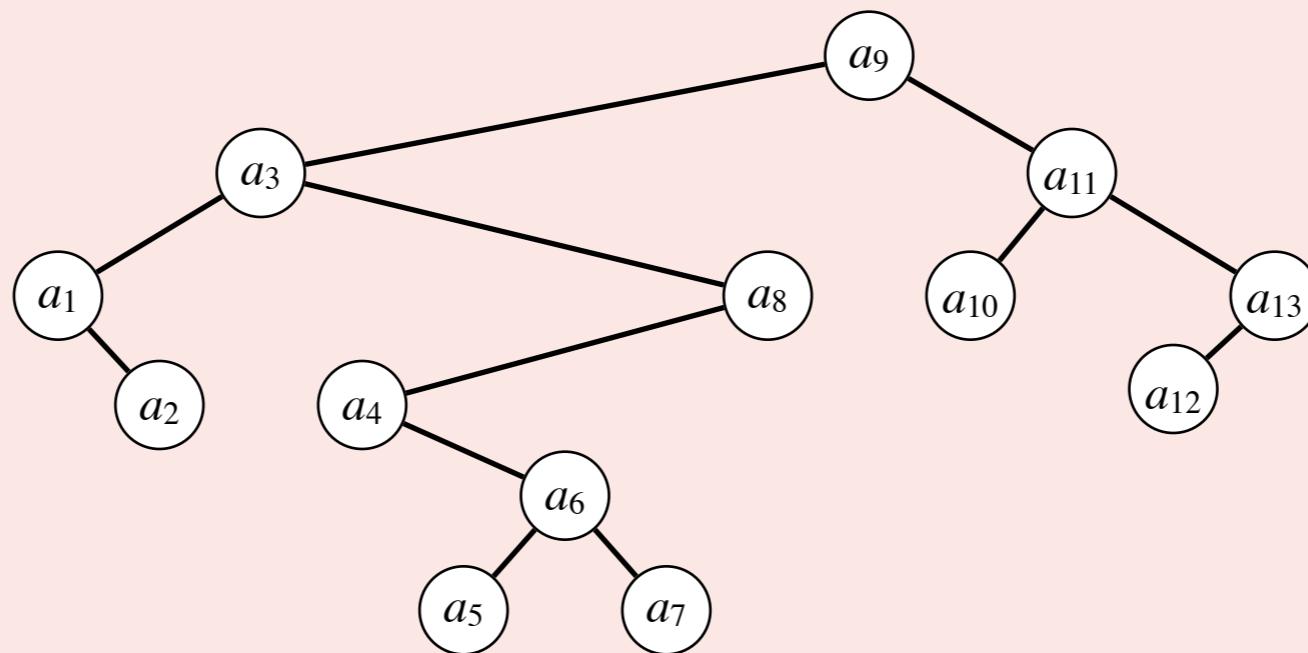
- A. 0
- B. $1 / n$
- C. $2 / n$
- D. 1





Given an array of $n \geq 2$ distinct elements $a_1 < a_2 < \dots < a_n$, what is the probability that a_1 and a_n are compared during randomized quicksort?

- A. 0
- B. $1 / n$
- C. $2 / n$
- D. 1



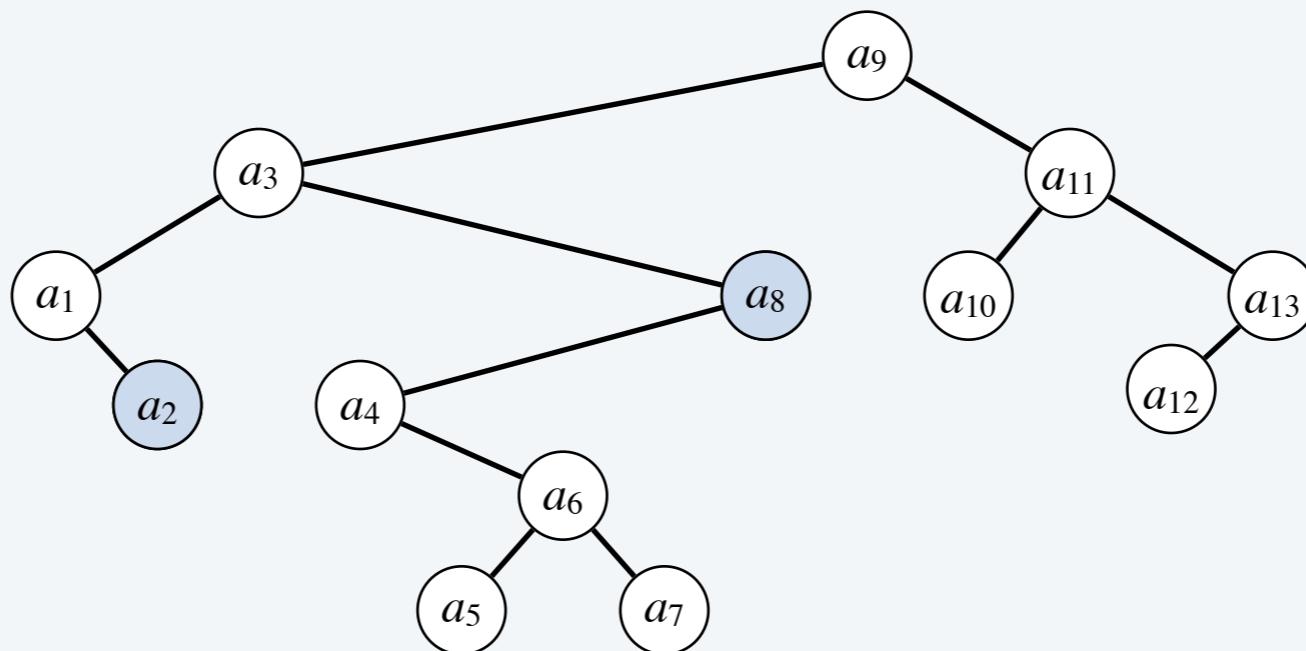
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- $\Pr [a_i \text{ and } a_j \text{ are compared}] = 2 / (j - i + 1)$, where $i < j$.

$\Pr[a_2 \text{ and } a_8 \text{ compared}] = 2/7$
compared iff either a_2 or a_8 is chosen
as pivot before any of $\{a_3, a_4, a_5, a_6, a_7\}$



Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- $\Pr [a_i \text{ and } a_j \text{ are compared}] = 2 / (j - i + 1)$, where $i < j$.
- Expected number of compares $= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j - i + 1} = 2 \sum_{i=1}^n \sum_{j=2}^{n-i+1} \frac{1}{j}$

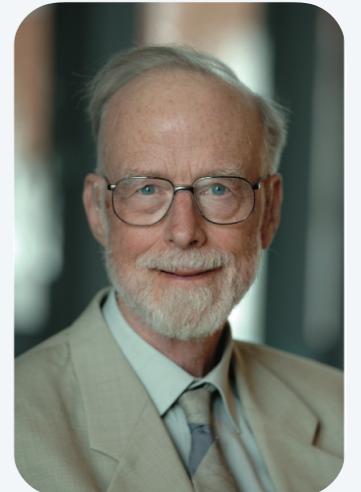
all pairs i and j
 $\leq 2n \sum_{j=1}^n \frac{1}{j}$
 $\leq 2n (\ln n + 1)$ ■

harmonic sum

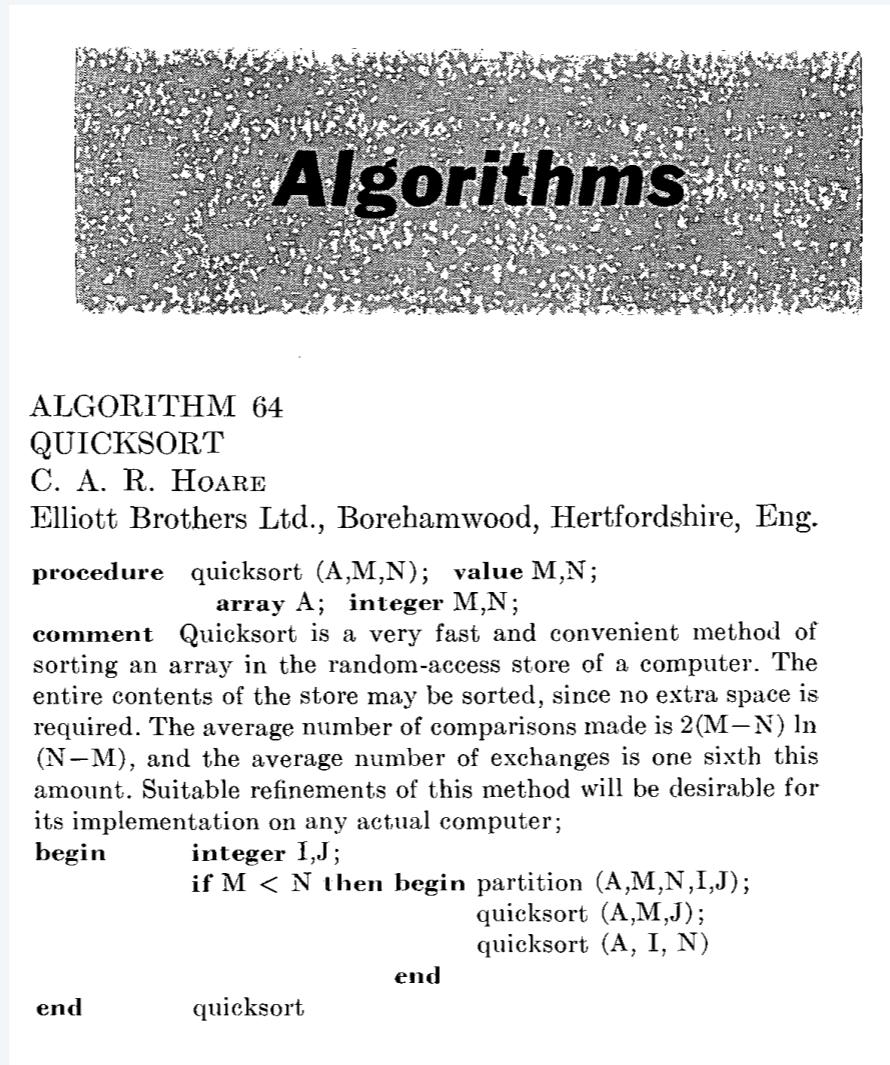
Remark. Number of compares only decreases if equal elements.

Tony Hoare

- Invented quicksort to translate Russian into English.
[but couldn't explain his algorithm or implement it!]
- Learned Algol 60 (and recursion).
- Implemented quicksort.



Tony Hoare
1980 Turing Award



ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure quicksort (A,M,N); value M,N;
    array A; integer M,N;
comment Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is  $2(M-N) \ln$ 
 $(N-M)$ , and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
            if M < N then begin partition (A,M,N,I,J);
                           quicksort (A,M,J);
                           quicksort (A, I, N)
                         end
end      quicksort
```

Communications of the ACM (July 1961)

NUTS AND BOLTS



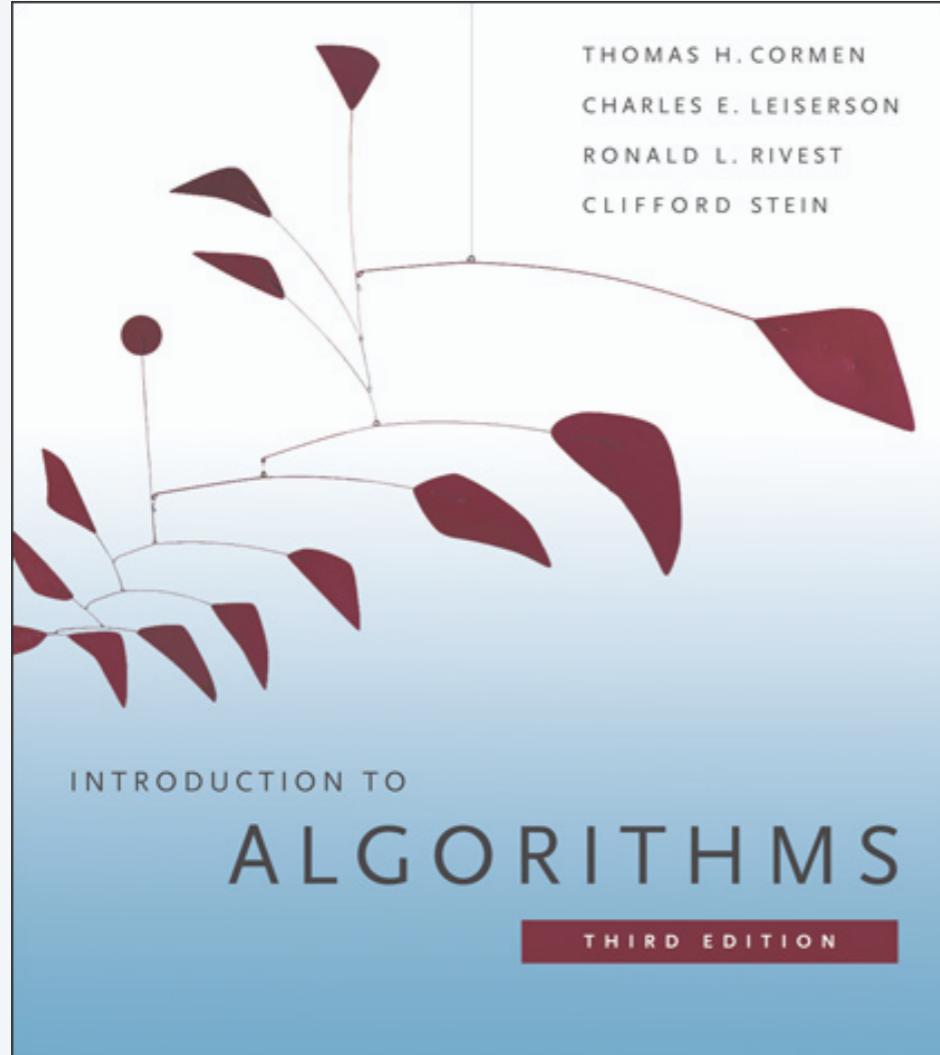
Problem. A disorganized carpenter has a mixed pile of n nuts and n bolts.

- The goal is to find the corresponding pairs of nuts and bolts.
- Each nut fits exactly one bolt and each bolt fits exactly one nut.
- By fitting a nut and a bolt together, the carpenter can see which one is bigger (but cannot directly compare either two nuts or two bolts).



Brute-force solution. Compare each bolt to each nut— $\Theta(n^2)$ compares.

Challenge. Design an algorithm that makes $O(n \log n)$ compares.



SECTION 9.3

5. DIVIDE AND CONQUER

- ▶ *mergesort*
- ▶ *counting inversions*
- ▶ *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. ← max heap with k smallest

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.

Randomized quickselect

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and 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.

$(L, M, R) \leftarrow \text{PARTITION-3-WAY}(A, p)$. $\longleftarrow \Theta(n)$

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

ELSE IF $(k > |L| + |M|)$ RETURN QUICK-SELECT($R, k - |L| - |M|$) $\longleftarrow T(n - i - 1)$

ELSE IF $(k = |L|)$ RETURN p .

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

select the $k = 8^{\text{th}}$ smallest

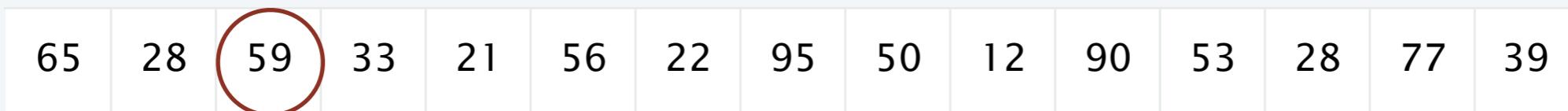
| | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 65 | 28 | 59 | 33 | 21 | 56 | 22 | 95 | 50 | 12 | 90 | 53 | 28 | 77 | 39 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

$k = 8^{\text{th}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

choose a pivot element at random and partition



$k = 8^{\text{th}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

partitioned array



$k = 8^{\text{th}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

recursively select 8th smallest element in left subarray

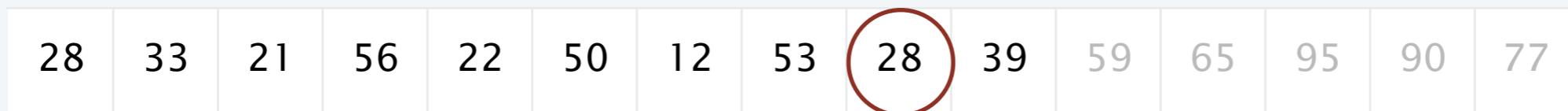
| | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 28 | 33 | 21 | 56 | 22 | 50 | 12 | 53 | 28 | 39 | 59 | 65 | 95 | 90 | 77 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

$k = 8^{\text{th}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

choose a pivot element at random and partition



$k = 8^{\text{th}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

partitioned array



$k = 8^{\text{th}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

recursively select the 3rd smallest element in right subarray

| | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 21 | 22 | 12 | 28 | 28 | 33 | 56 | 50 | 53 | 39 | 59 | 65 | 95 | 90 | 77 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

$k = 3^{\text{rd}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

choose a pivot element at random and partition

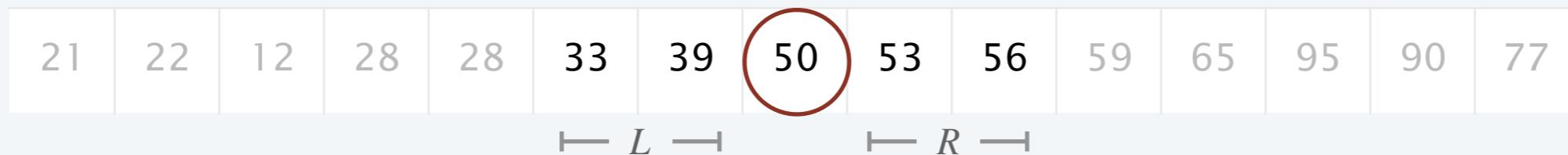


$k = 3^{\text{rd}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

partitioned array



$k = 3^{\text{rd}}$ smallest

Quickselect demo

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L , M , and R .
- Recur in **one** subarray—the one containing the k^{th} smallest element.

stop: desired element is in middle subarray



Randomized quickselect analysis

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

$$T(n) \leq T(3n/4) + n \Rightarrow T(n) \leq 4n$$

not rigorous: can't assume
 $E[T(i)] \leq T(E[i])$



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

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 of
larger of two subarrays since $T(n)$
is monotone non-decreasing



$$T(n) \leq n + 1/n [2T(n/2) + \dots + 2T(n-3) + 2T(n-2) + 2T(n-1)]$$

$$\leq n + 1/n [8(n/2) + \dots + 8(n-3) + 8(n-2) + 8(n-1)]$$

$$\leq n + 1/n (3n^2)$$

$$= 4n. \blacksquare$$

inductive hypothesis

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.

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

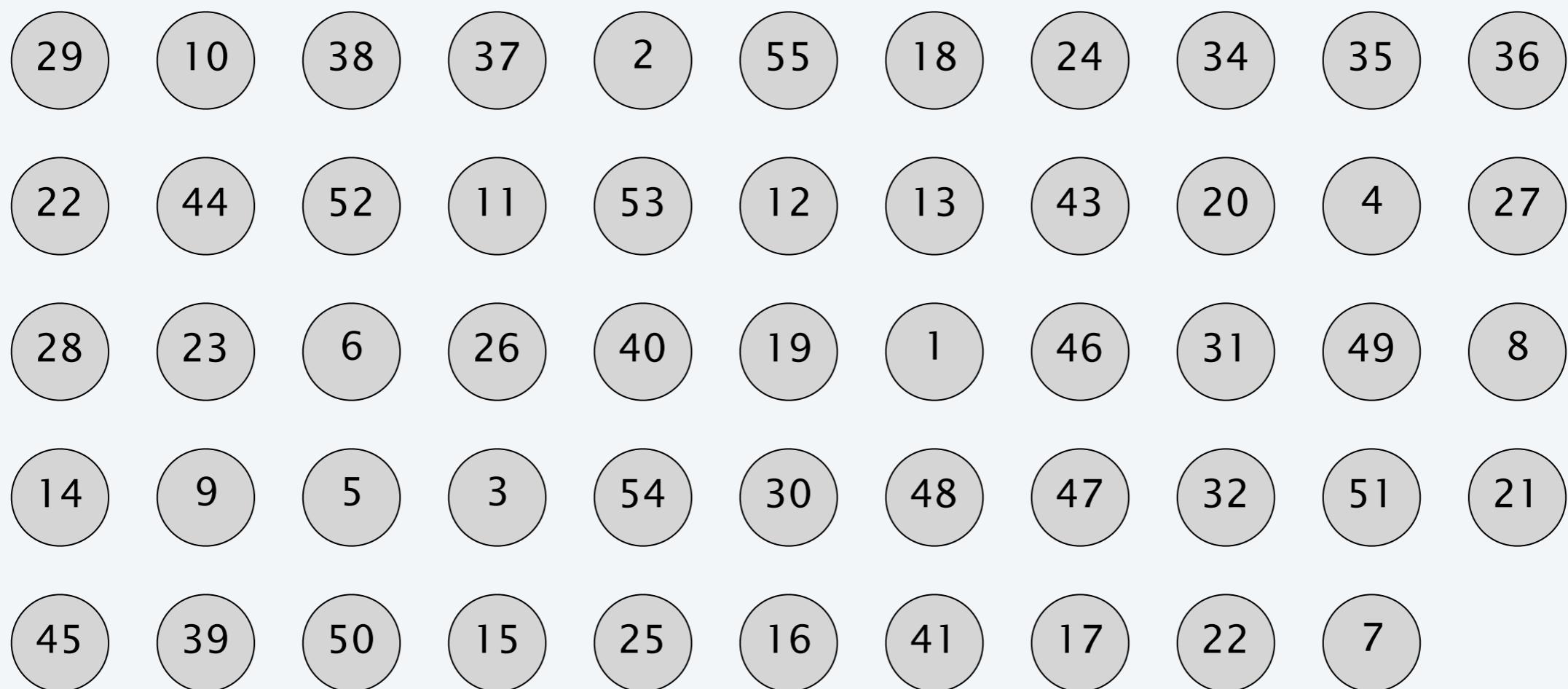
two subproblems
of different sizes!

$$\Rightarrow T(n) = \Theta(n)$$

we'll need to show this

Choosing the pivot element

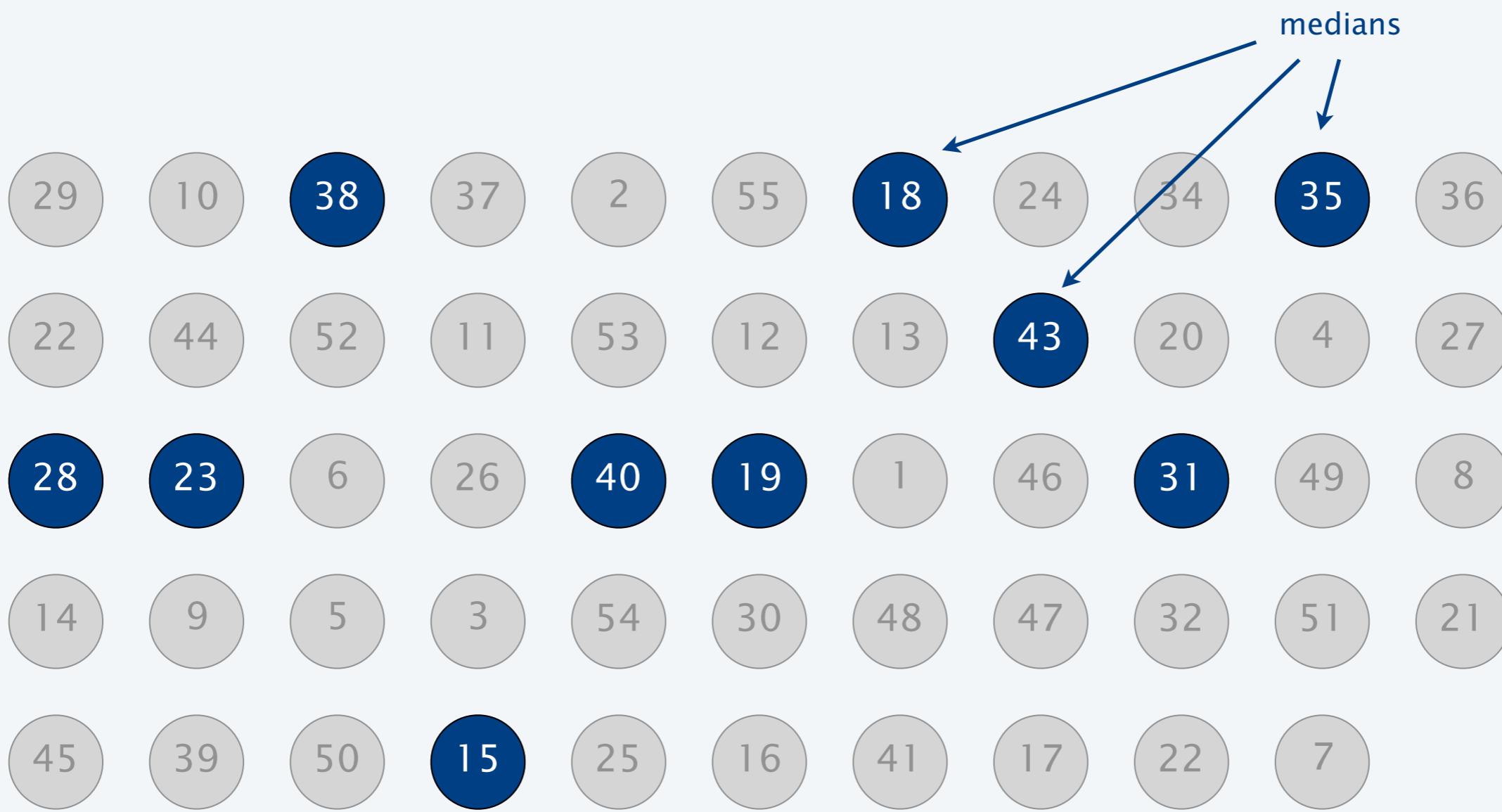
- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).



$n = 54$

Choosing the pivot element

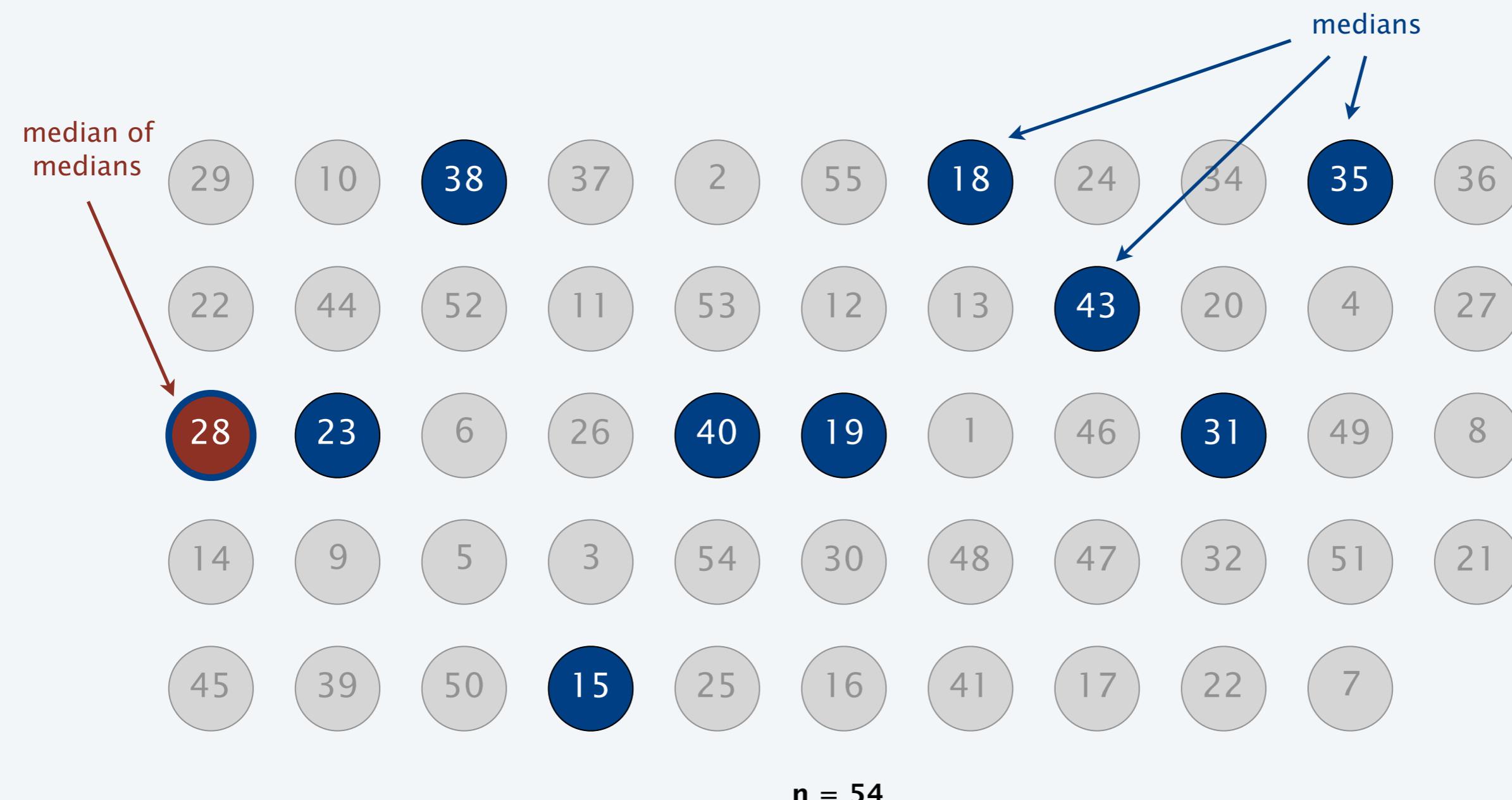
- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).



$n = 54$

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.



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 (ignore leftovers).

$B \leftarrow$ 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).

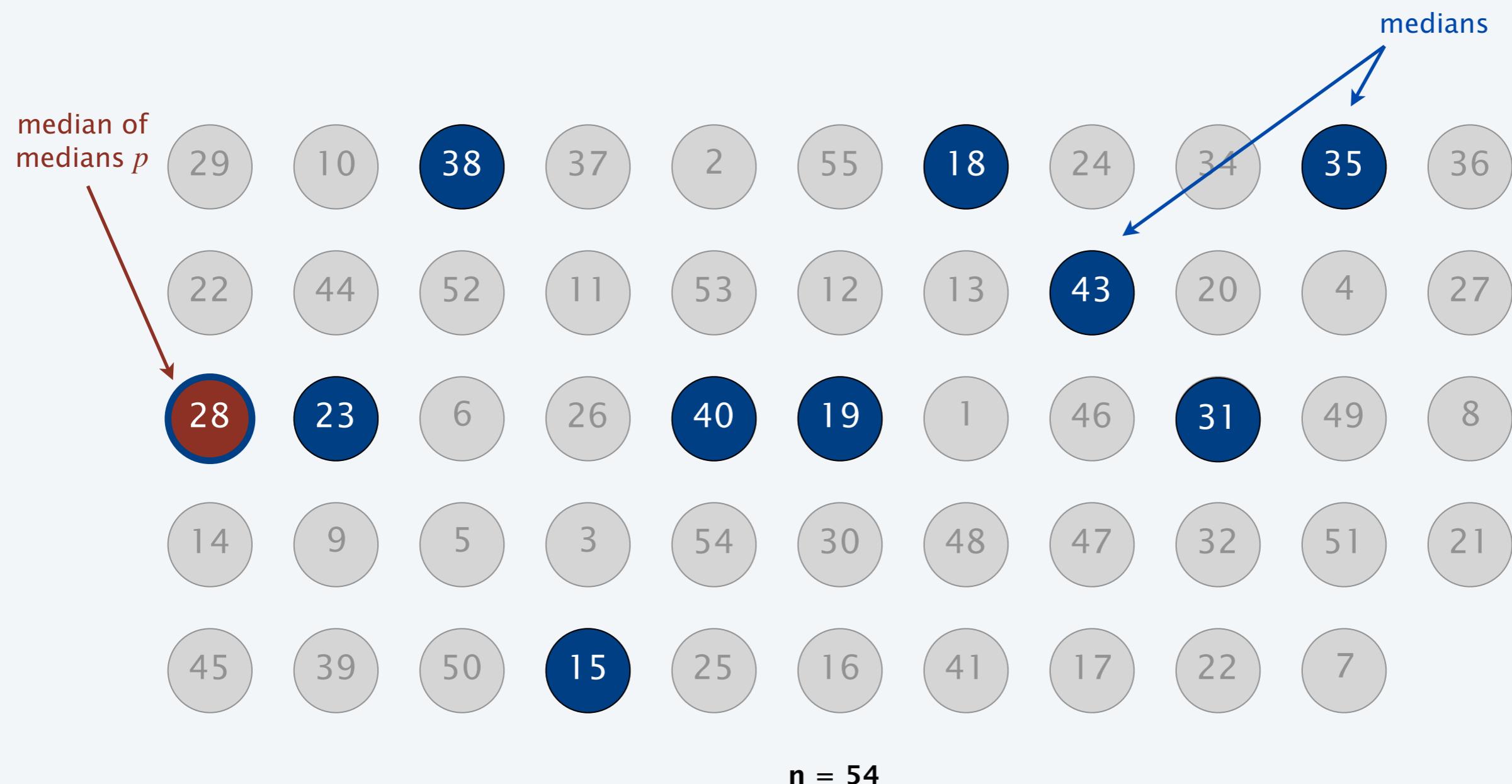
IF $(k \leq |L|)$ **RETURN** **MOM-SELECT**(L, k).

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

ELSE **RETURN** p .

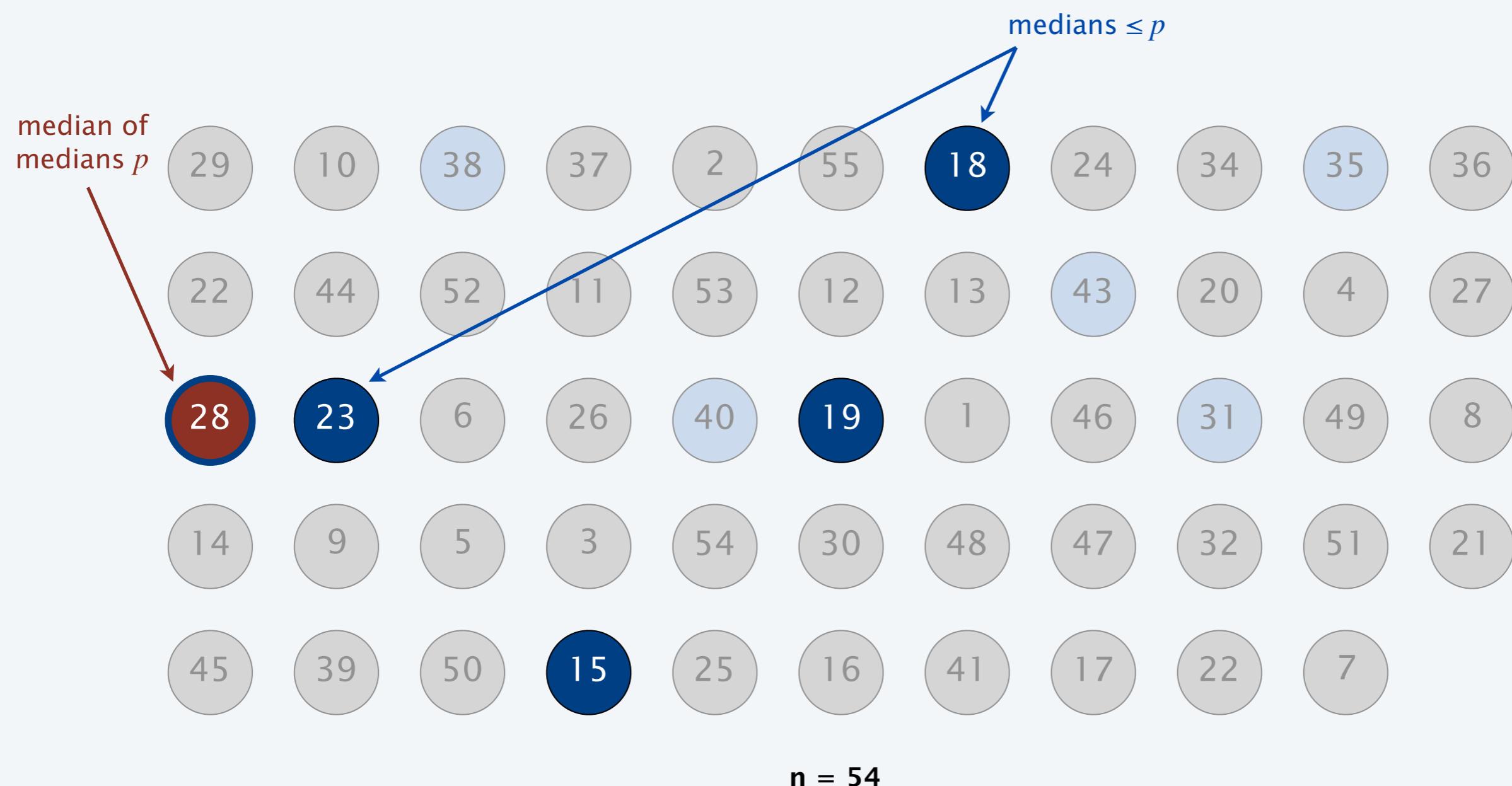
Analysis of median-of-medians selection algorithm

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



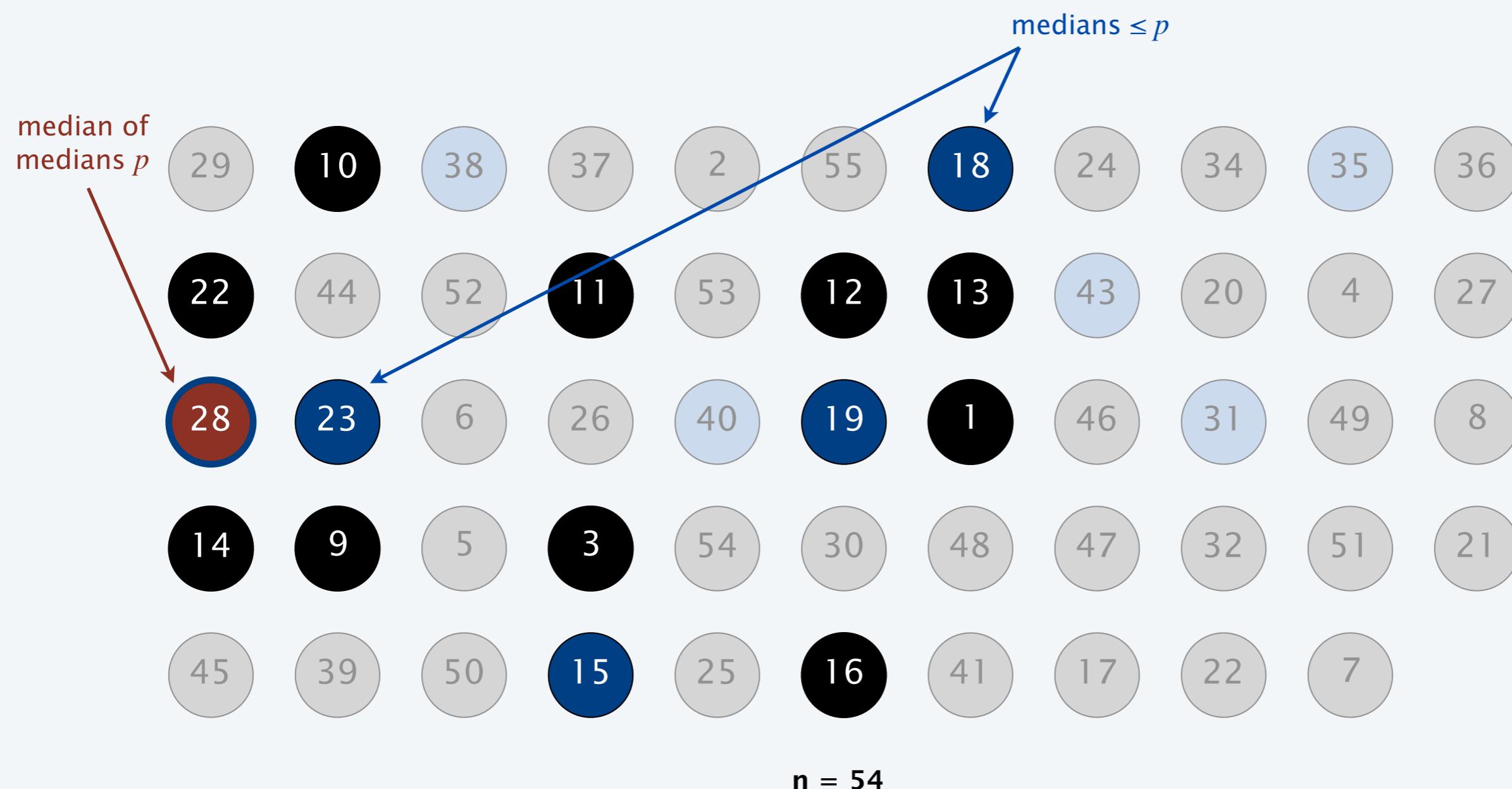
Analysis of median-of-medians selection algorithm

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



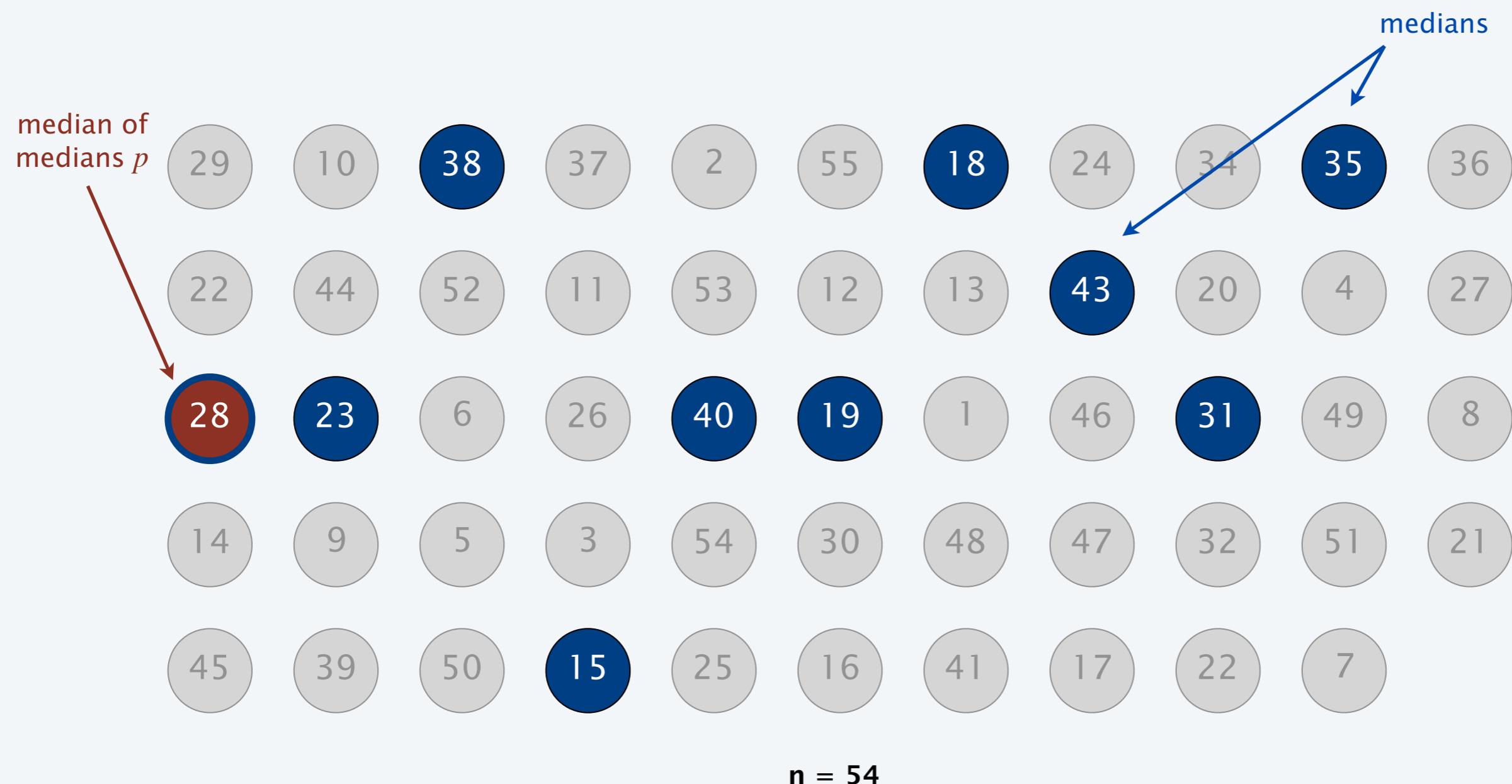
Analysis of median-of-medians selection algorithm

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



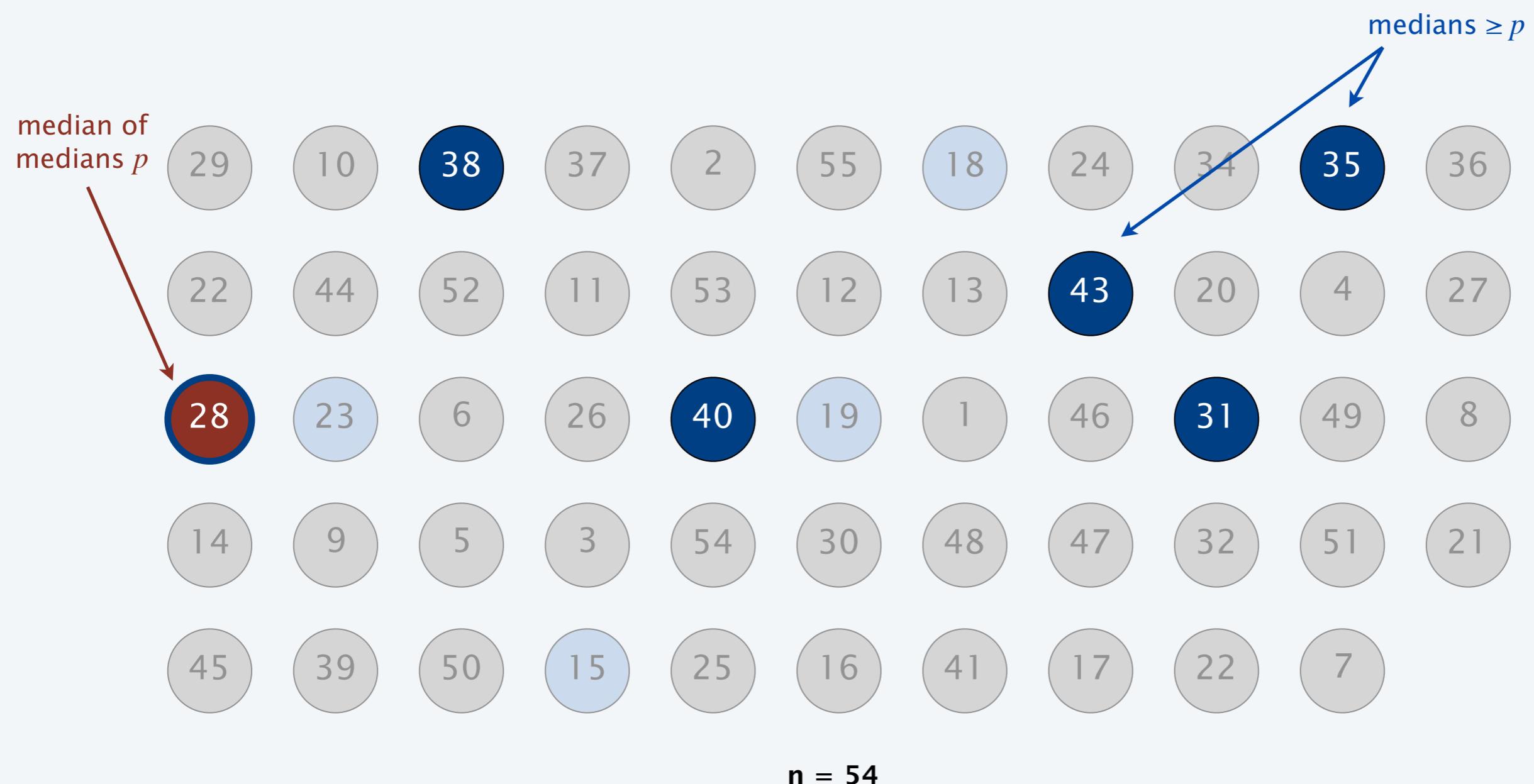
Analysis of median-of-medians selection algorithm

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



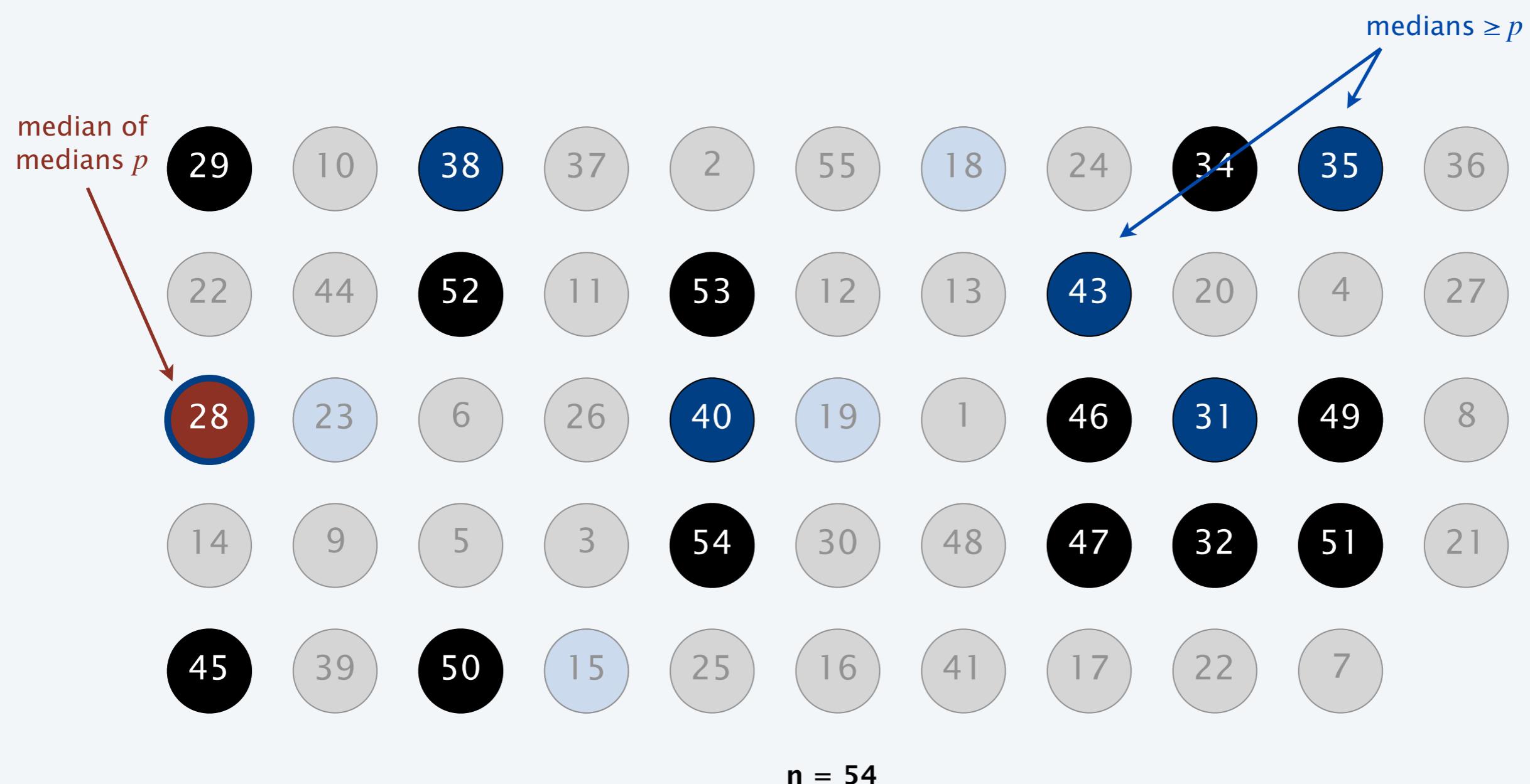
Analysis of median-of-medians selection algorithm

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



Analysis of median-of-medians selection algorithm

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



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. $C(n) = \max \# \text{ compares on any array of } n \text{ elements.}$

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

median of
medians recursive
select computing median of 5
(≤ 6 compares per group)
partitioning
($\leq n$ compares)

Intuition.

- $C(n)$ is going to be at least linear in $n \Rightarrow C(n)$ is super-additive.
- Ignoring floors, this implies that
$$\begin{aligned} C(n) &\leq C(n/5 + n - 3n/10) + 11/5 n \\ &= C(9n/10) + 11/5 n \\ &\Rightarrow C(n) \leq 22n. \end{aligned}$$

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. $C(n) = \max \# \text{ compares on any array of } n \text{ elements.}$

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

median of
medians recursive
select computing median of 5
(≤ 6 compares per group)
partitioning
($\leq n$ compares)

Now, let's solve given recurrence.

- Assume n is both a power of 5 and a power of 10 ?
- Prove that $C(n)$ is monotone non-decreasing.



Consider the following recurrence

$$C(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n & \text{if } n > 1 \end{cases}$$

Is $C(n)$ monotone non-decreasing?

- A. Yes, obviously.
- B. Yes, but proof is tedious.
- C. Yes, but proof is hard.
- D. No.

Median-of-medians selection algorithm recurrence

Analysis of selection algorithm recurrence.

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

$$T(n) \leq \begin{cases} 6n & \text{if } n < 50 \\ \max\{ T(n-1), T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n \} & \text{if } n \geq 50 \end{cases}$$

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

Pf. [by strong induction]

- 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 either $T(n) \leq T(n-1) \leq 44n$ or

$$T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5n$$

inductive hypothesis $\longrightarrow \leq 44(\lfloor n/5 \rfloor) + 44(n - 3\lfloor n/10 \rfloor) + 11/5n$

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



Suppose that we divide n elements into $\lfloor n / r \rfloor$ groups of r elements each, and use the median-of-medians of these $\lfloor n / r \rfloor$ groups as the pivot.
For which r is the worst-case running time of select $O(n)$?

- A. $r = 3$
- B. $r = 7$
- C. Both A and B.
- D. Neither A nor B.