CSE202
Design and Analysis of Algorithms

Week 3 — Divide & Conquer 2: Rankings, Selection,…

I. Comparing Rankings

Compare Two Rankings

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: ?

Me	You
A	D
C	A
D	B
B	C

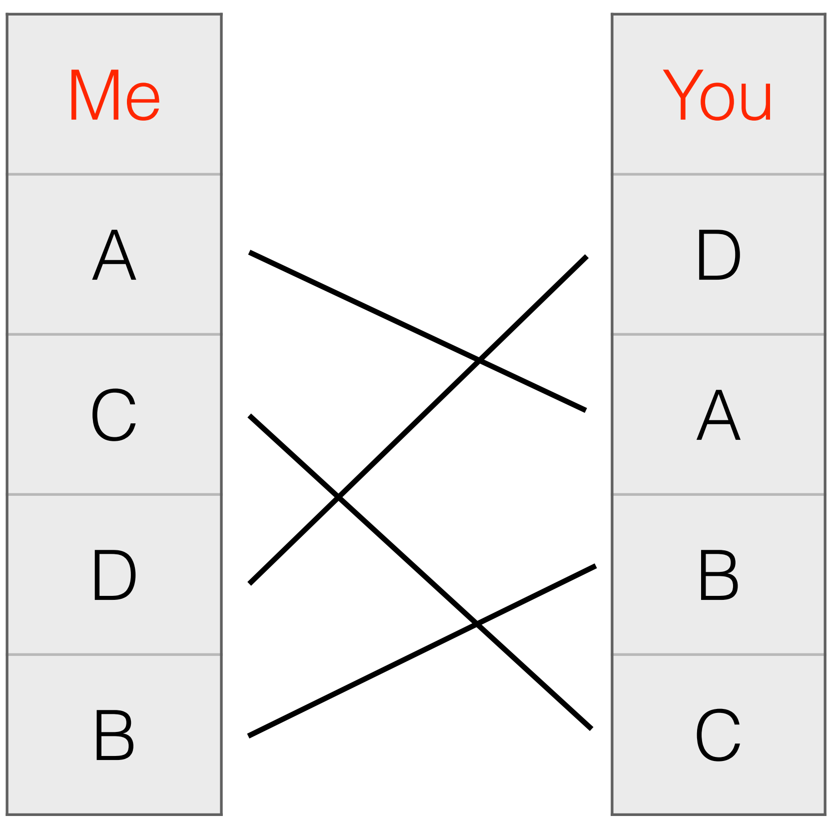
Compare Two Rankings

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

[Kendall-tau distance]



(A,D) (C,D) (C,B)

Number of crossings of pairs of line segments: 3

Compare Two Rankings

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

[Kendall-tau distance]

Me	
A	: 1
C	: 2
D	: 3
B	: 4

You	
D	: 3
A	: 1
B	: 4
C	: 2

Array A:

3	1	4	2
1	2	3	4

Number of inversions:

pairs $i < j$ such that $A[i] > A[j]$

Counting Inversions

Input. An array A

Output. Number of pairs $i < j$ such that $A[i] > A[j]$

Brute force algorithm: check all $O(n^2)$ pairs i and j

Divide and Conquer: $O(n \log n)$

Counting Inversions: Applications

Voting theory

Rank aggregation for meta-searching on the Web

Measuring the "sortedness" of an array

…

Rank aggregation methods for the Web

Cynthia Dwork[1], **Ravi Kumar**[2], **Moni Naor**[3], **D. Sivakumar**[2]

1: *Compaq Systems Research Center*, Palo Alto, CA, USA.
2: *IBM Almaden Research Center*, San Jose, CA, USA.
3: *Weizmann Institute of Science*, Rehovot, Israel.
(Visiting *IBM Almaden* and *Stanford University*)

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.

Counting Inversions: DAC

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

Divide into 2 sublists of equal size

1	5	4	8	10	2

6	9	12	11	3	7

Recursively count the inversions

5 yellow-yellow inv.

8 red-red inv.

Combine: add recursive counts and yellow-red inv

9 yellow-red inv.

Total: 5 + 8 + 9 = 22

Counting Inversions: DAC

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

Cost

Divide into 2 sublists of equal size

1	5	4	8	10	2

6	9	12	11	3	7

O(1)

Recursively count the inversions

5 yellow-yellow inv.

8 red-red inv.

2* C(n/2)

Combine: add recursive counts and yellow-red inv

9 yellow-red inv.

Total: 5 + 8 + 9 = 22

???

Counting Inversions: DAC

Variation of **merge-sort**

Combine: count yellow-red inversions

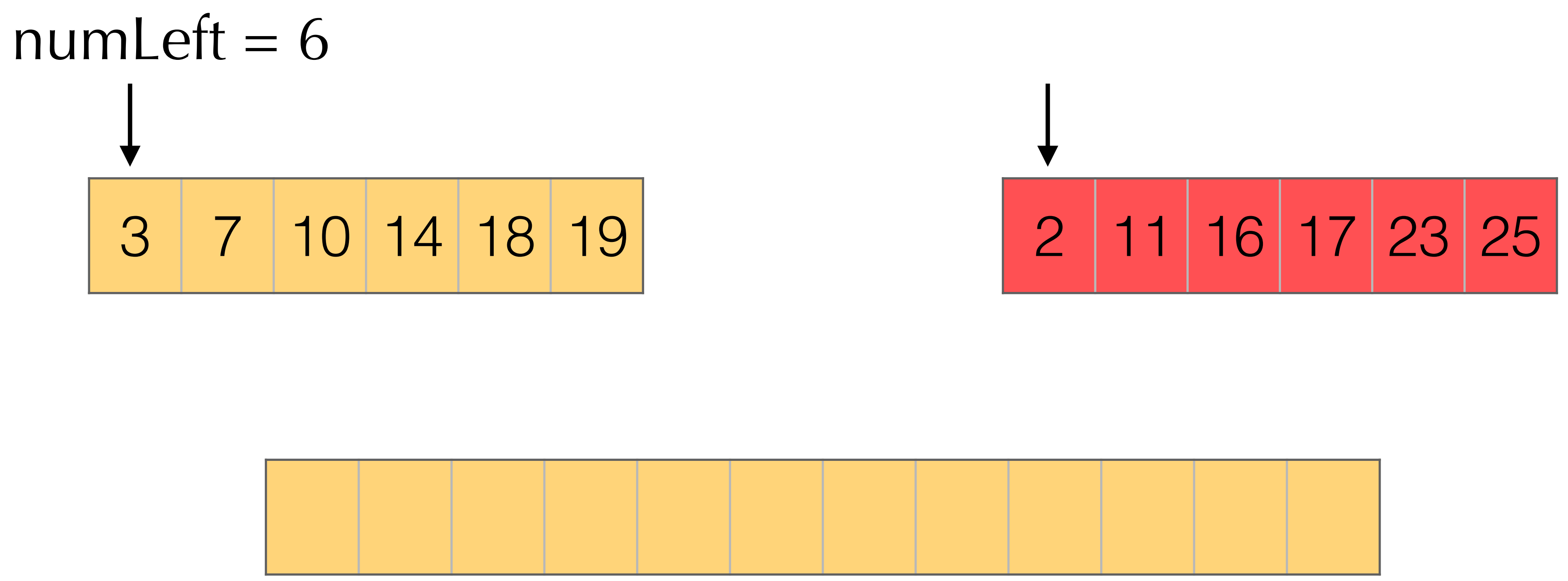
- assume each half is sorted
- count inversions where A[i] and A[j] are in different halves
- merge two sorted halves into sorted whole

Merge-and-Count: count inversions while merging the two sorted lists

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

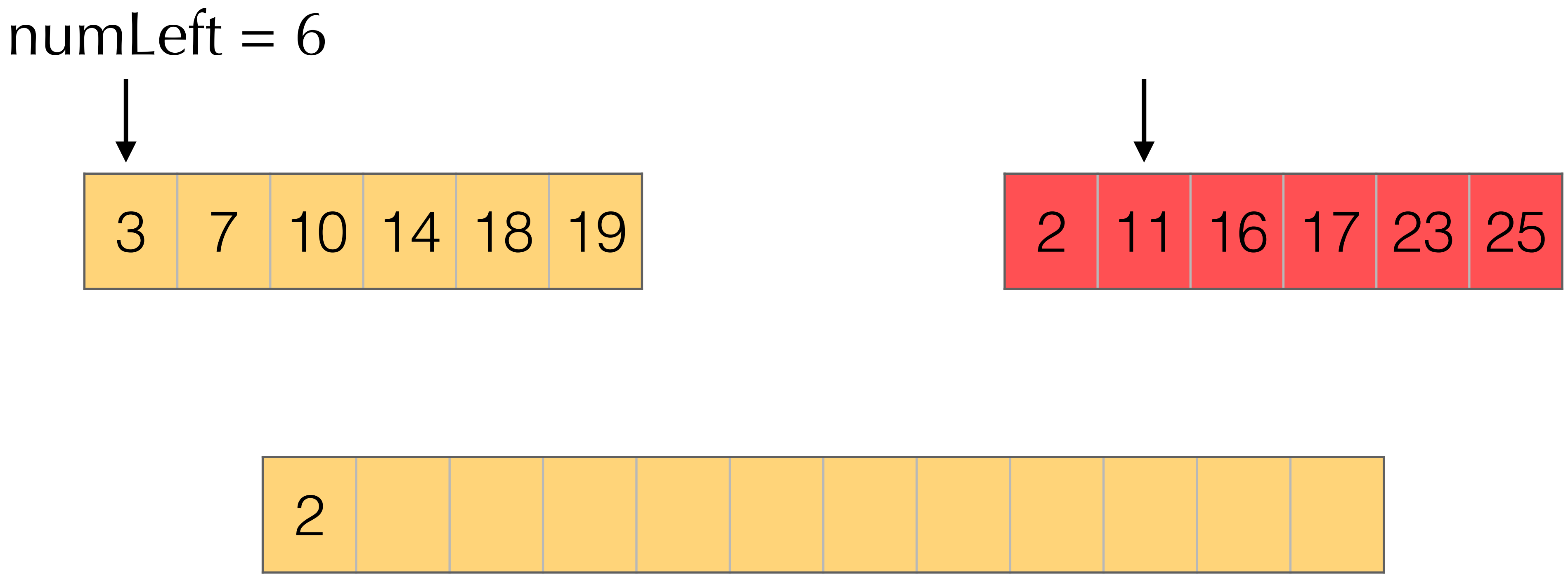


Total:

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

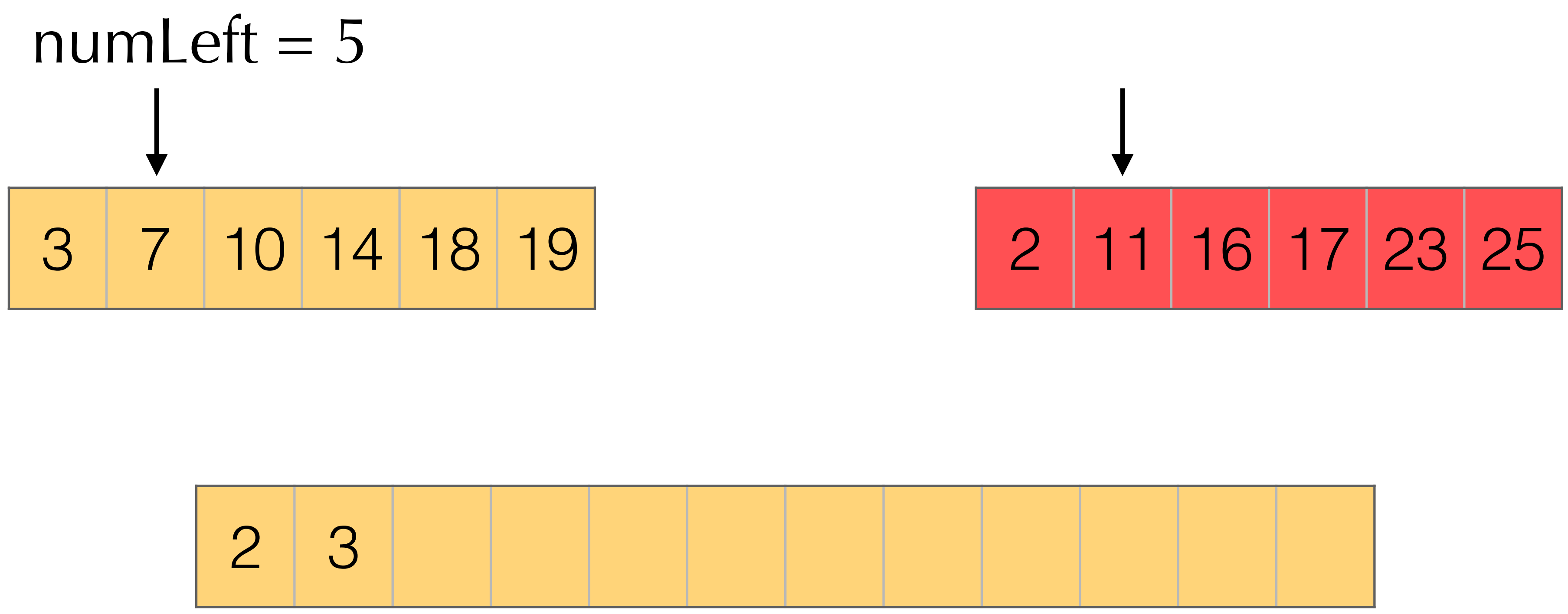


Total: 6

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

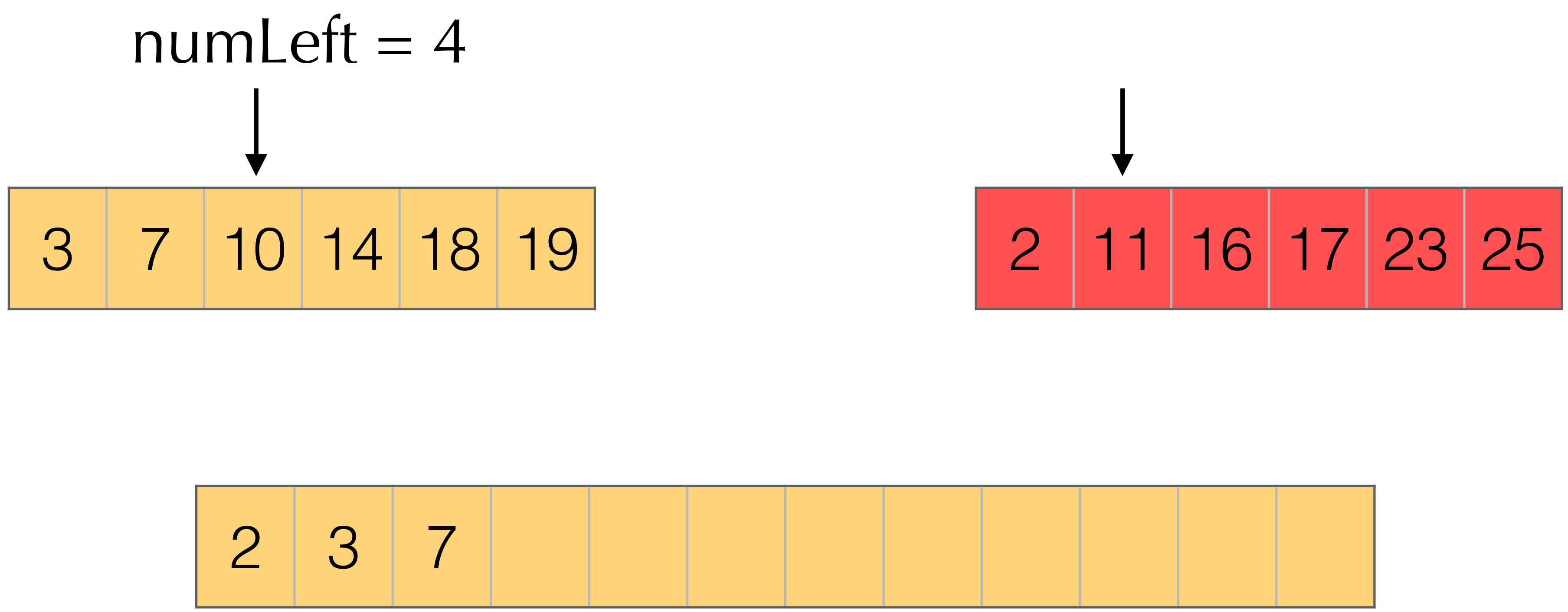


Total: 6

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

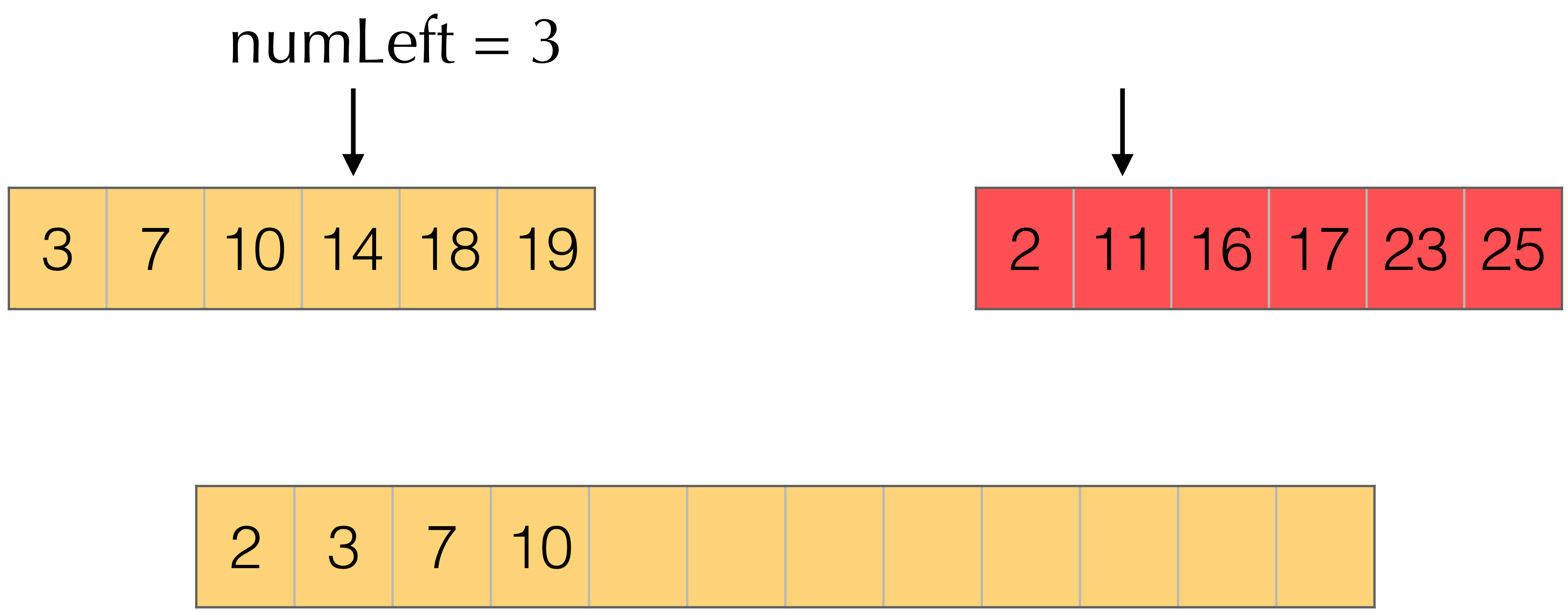


Total: 6

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

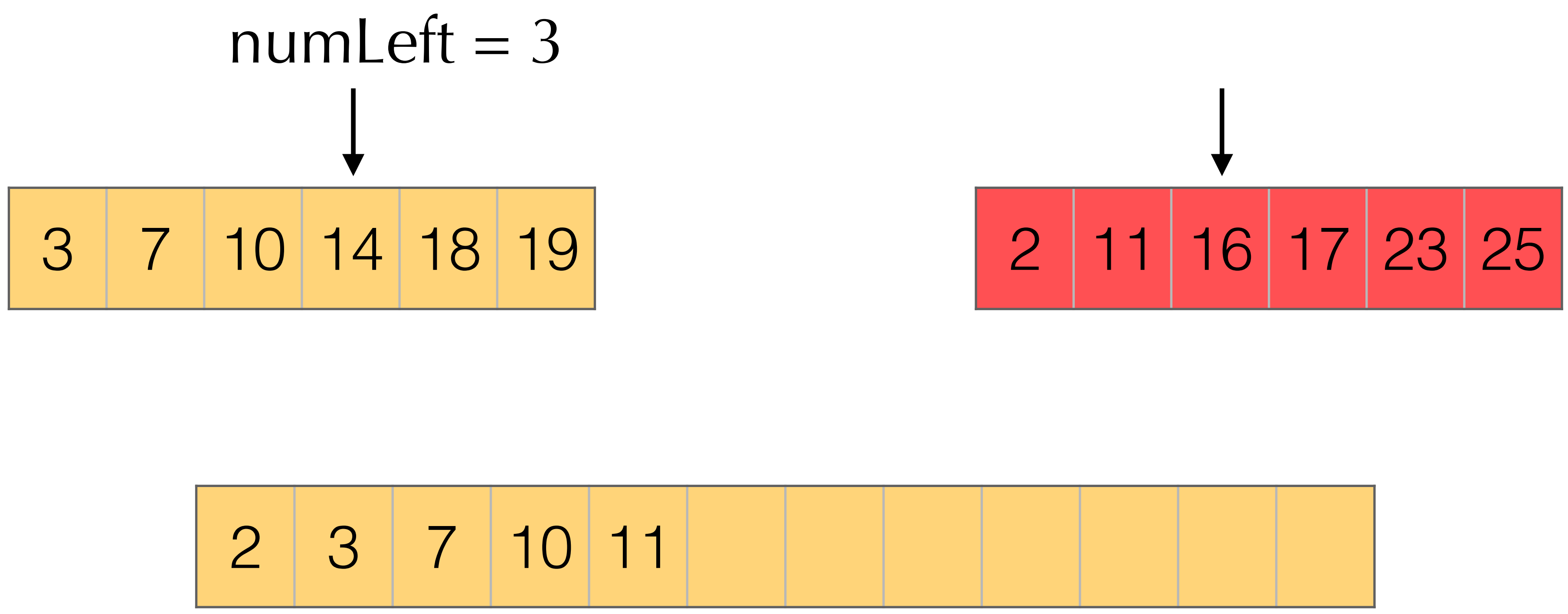


Total: 6

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

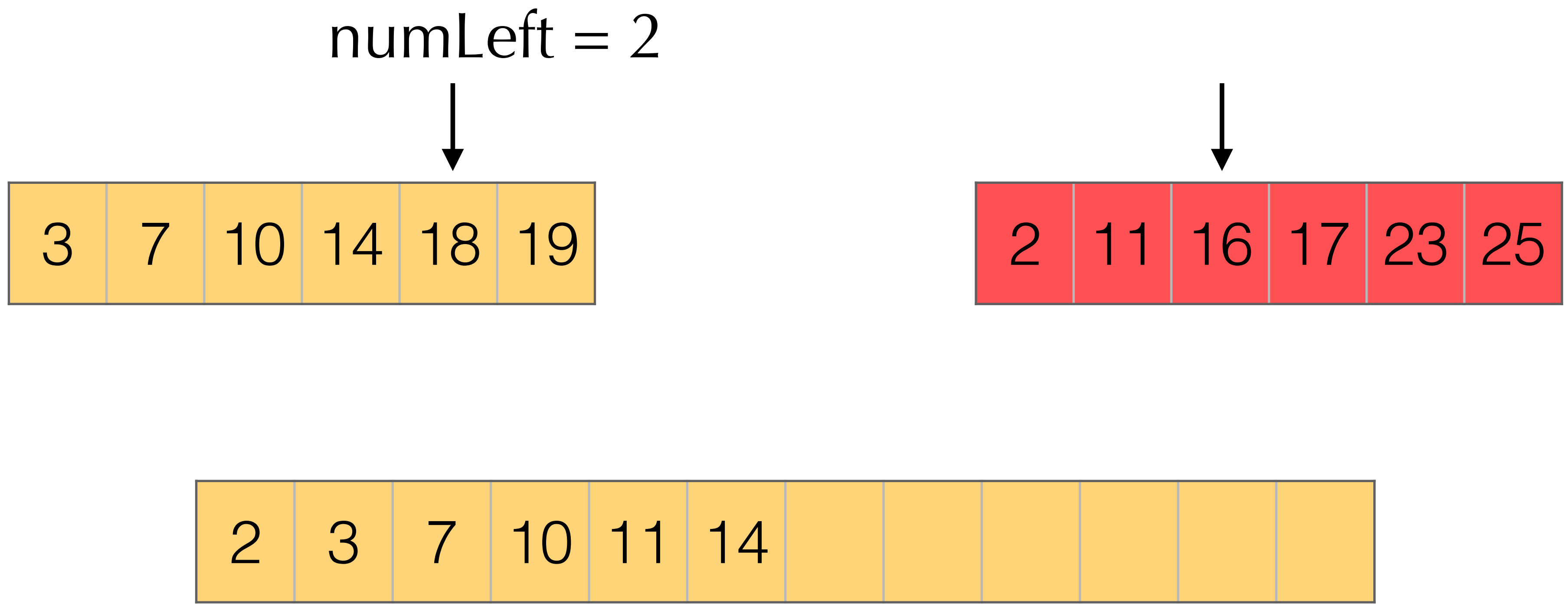


Total: 6 + 3

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole



Total: 6 + 3

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

numLeft = 2

3	7	10	14	18	19

2	11	16	17	23	25

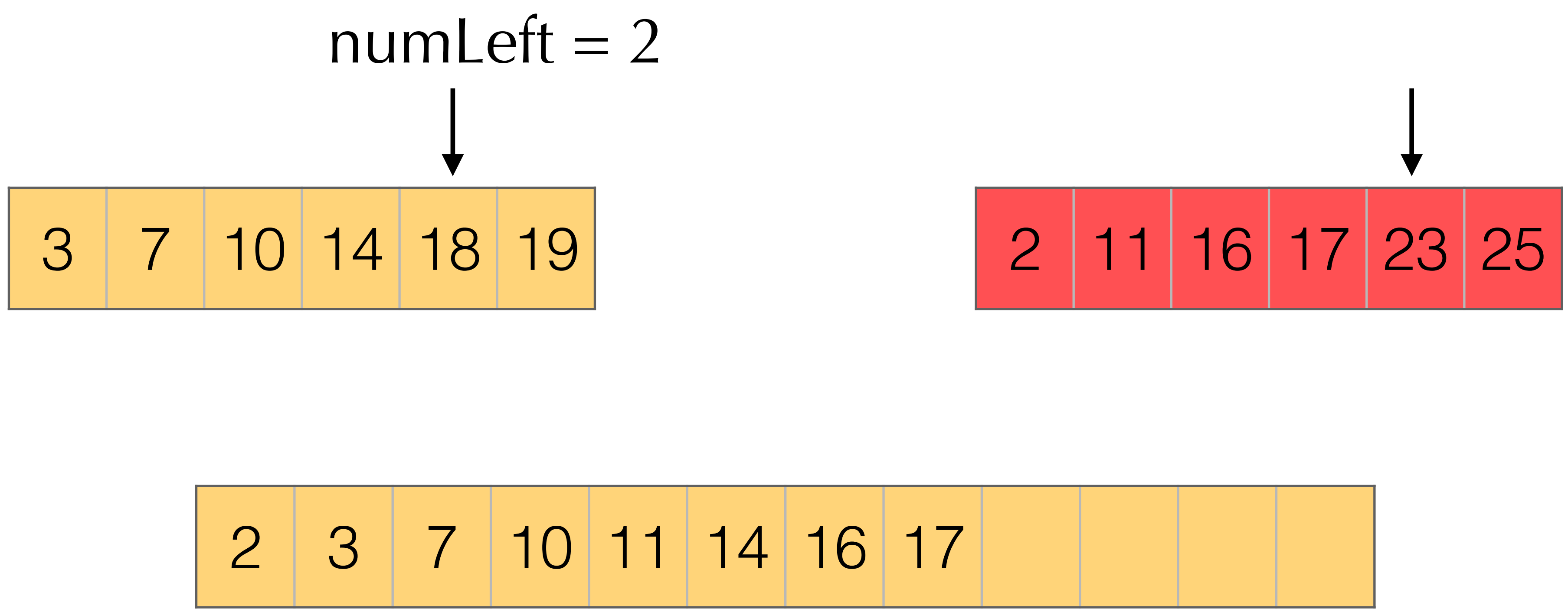
2	3	7	10	11	14	16					

Total: 6 + 3 + 2

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

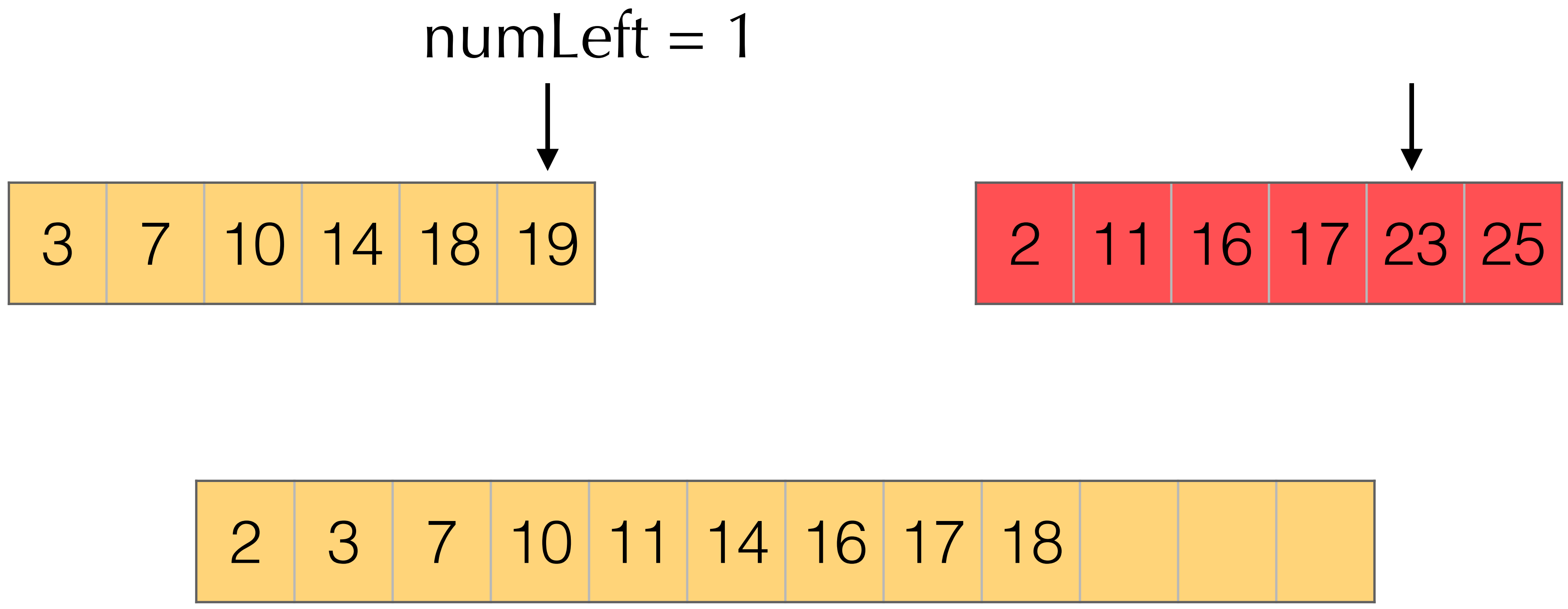


Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

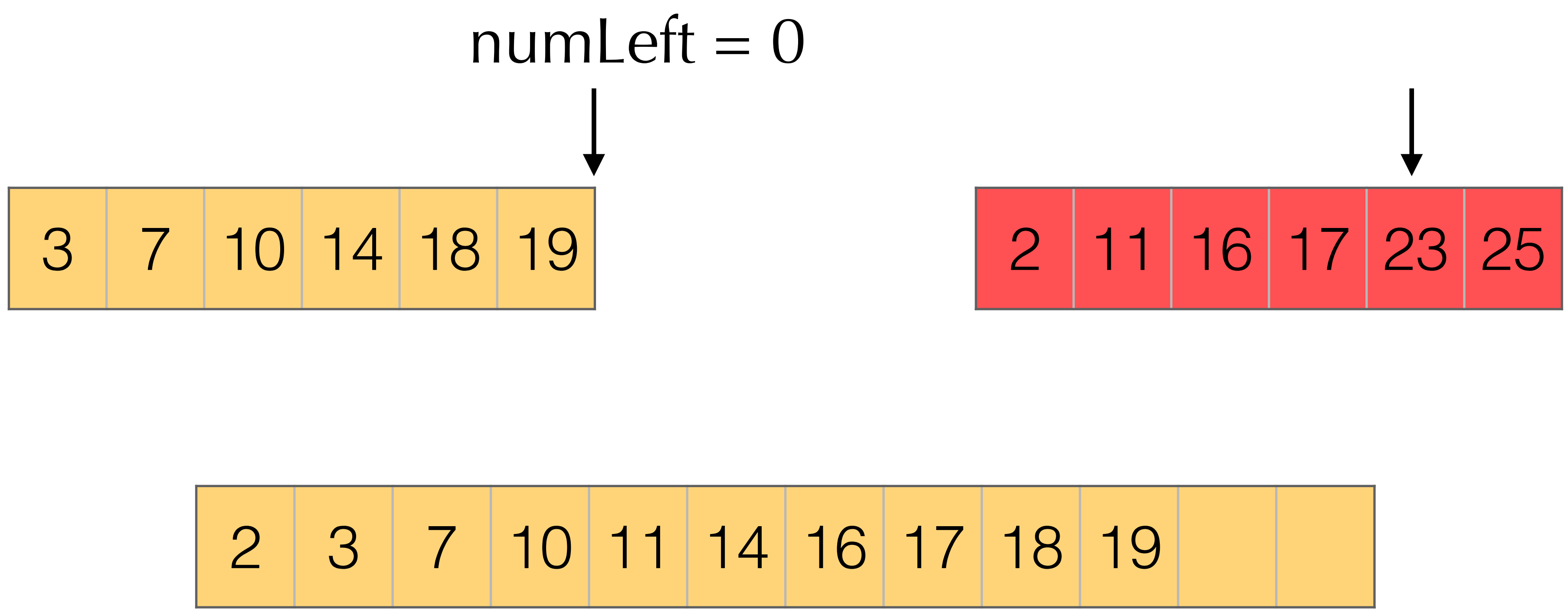


Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

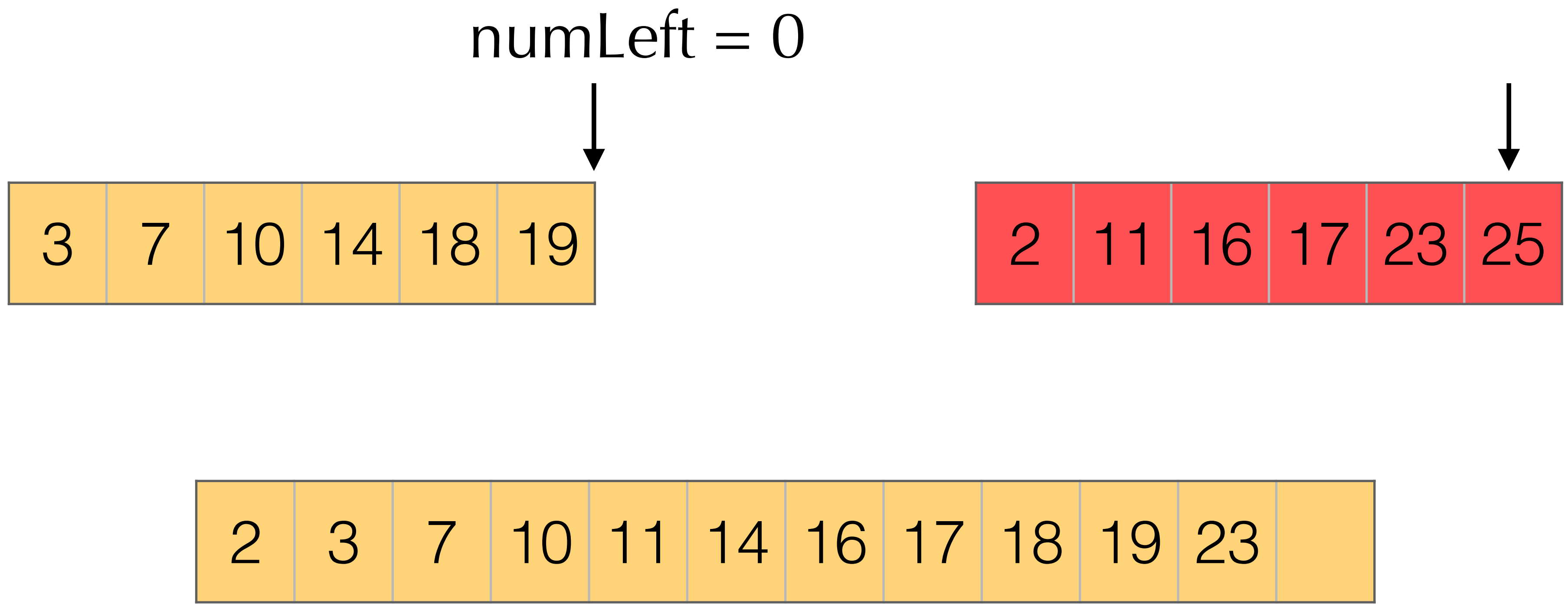


Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole

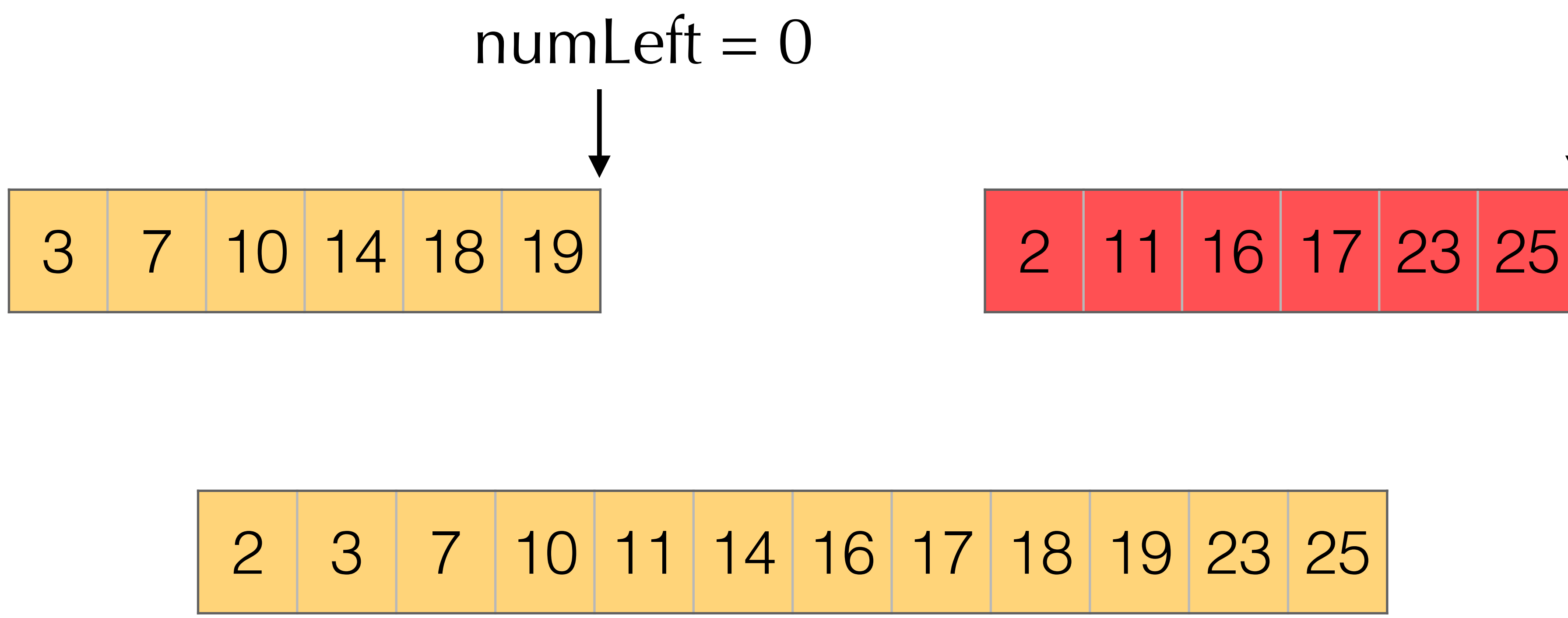


Total: 6 + 3 + 2 + 2

Merge and Count

Merge and count step:

- given two sorted halves, count the number of inversions where A[i] and A[j] are in different halves
- combine two sorted halves into sorted whole



Total: 6 + 3 + 2 + 2 = 13

Merge and Count

```
Sort-and-Count(A):
  if A has one element
    return (0,A)

  Divide A into two halves A1, A2
  (r1, A1) ← Sort-and-Count(A1)
  (r2,A2)  ← Sort-and-Count(A2)

  (rC, A) ← Merge-and-Count(A1,A2)
  return (r1+r2+rC, A)
```

```
Merge-and-Count(A1,A2):
  Initialize an empty array B
  Inv ← 0
  If A1 or A2 is empty …
  Compare first elems of A1, A2
  If the smallest is in A1:
      move it at the end of B
  Else
      move it at the end of B
      Inv += |A1|

  return (Inv, B)
```

Complexity of the Sort-and-Count

Notation:

$$\lceil x/2 \rceil_1 = \lceil x/2 \rceil$$

$$\lceil x/2 \rceil_{k+1} = \left\lceil \lceil x/2 \rceil_k /2 \right\rceil$$

N : power of 2 s.t.

$$n \le N < 2n$$

$$C(n) \le 2C(\lceil n/2 \rceil) + \lambda n$$

iterate once

$$\le \lambda n + 2\lambda \lceil n/2 \rceil + 4C(\lceil n/2 \rceil_2)$$

iterate k-1 times, use N

$$\le \lambda N (1 + 1 + \cdots + 1) + 2^k C(\lceil n/2 \rceil_k)$$

trivial summation

$$\le \lambda N(k-1) + 2^k C(\lceil n/2 \rceil_k)$$

use $k = \lceil \log_2 n \rceil$

$$= \lambda n(\lceil log_2 n \rceil - 1) + 2^{\lceil log_2 n \rceil}$$

$$= O(nlogn)$$

II. Selection: Linear Time with DAC

Complexity of DAC algorithms

$O(\log n)$: binary powering

$O(n \log n)$: merge sort, counting inversions

$O(n^{\log_2 3} \approx n^{1.58})$: Karatsuba multiplication (integers, polynomials)

$O(n^{\log_2 7} \approx n^{2.80})$: Strassen's matrix multiplication

Next, we talk about a linear time algorithm

Statement of the Problem

Select: $(A := \{a_1, \ldots, a_n\}, k) \mapsto x \in A \text{ s.t. } |\{a \in A \mid a \leq x\}| = k$

Statement of the Problem

Select: $(A := \{a_1, \ldots, a_n\}, k) \mapsto x \in A$ s.t. $|\{a \in A \mid a \leq x\}| = k$

Min/Max: Select with $k = 1$ and $k = n$, resp.

Straightforward algorithm in $O(n)$ comparisons

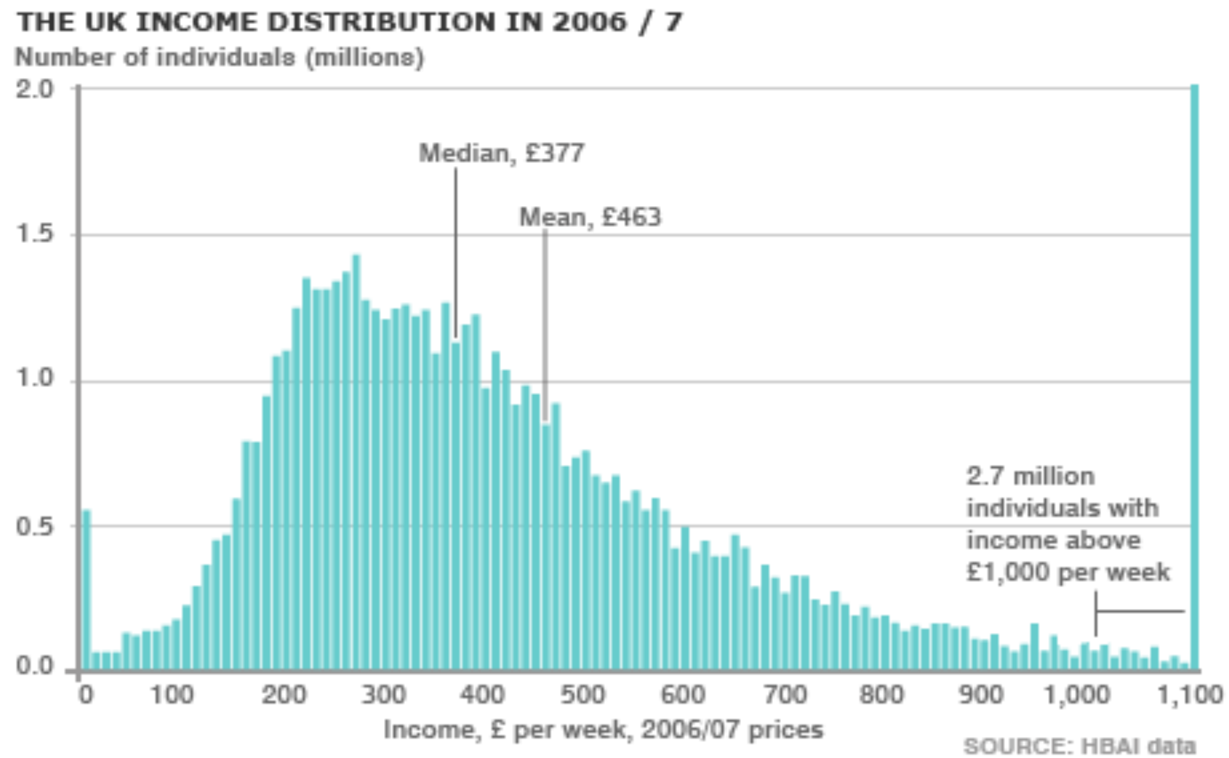
Statement of the Problem

Select: $(A := \{a_1, \ldots, a_n\}, k) \mapsto x \in A$ s.t. $|\{a \in A \mid a \leq x\}| = k$

Median: Select with $k = \lfloor n/2 \rfloor$

Sorting gives an algorithm in $O(n \log n)$ comparisons

Median vs Mean:



Statement of the Problem

Select: $(A := \{a_1, \ldots, a_n\}, k) \mapsto x \in A \text{ s.t. } |\{a \in A \mid a \leq x\}| = k$

Sorting gives an algorithm in $O(n \log n)$ comparisons

Applications: Order statistics; find the "top k", …

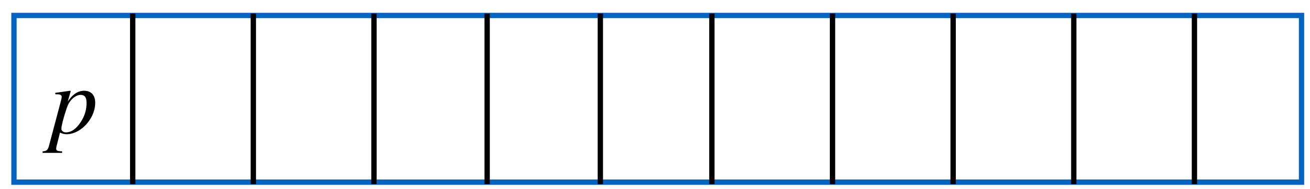
Q. Can we do it with O(n) compares ?

A. Yes! Selection is easier than sorting

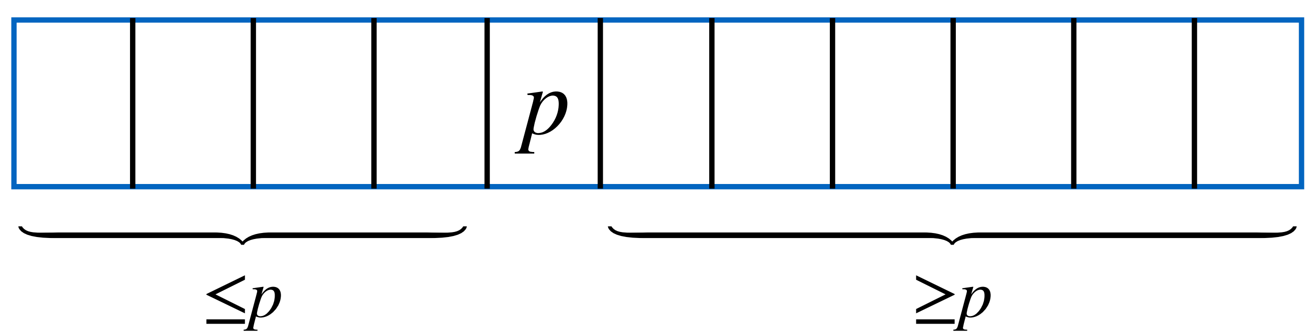
Recall: QuickSort Partitioning (CSE103)

Input: an array of n comparable elements

a pivot p among them



Output: array partitioned around p; new index of p.



Complexity: $O(n)$ comparisons

Select using Partitioning

Algorithm $\text{Select}(A, k)$:

If $|A| = 1$, return $A[0]$

Choose a good pivot p

$q := \text{Partition}(a, p)$

If $q = k$ return q

If $q > k$ return $\text{Select}(A[: q], k)$

If $q < k$ return $\text{Select}(A[q :], k - q)$

Worst-case: $C(n) \leq C(n) + O(n) \longrightarrow O(n^2)$

Select using Partitioning

Algorithm $\text{Select}(A, k)$:

If $|A| = 1$, return $A[0]$

Choose a good pivot p

$q := \text{Partition}(a, p)$

If $q = k$ return q

If $q > k$ return $\text{Select}(A[: q], k)$

If $q < k$ return $\text{Select}(A[q :], k - q)$

Low complexity:

depends on

$\Big| q - |A|/2 \Big|$

being small

Worst-case: $C(n) \le C(n) + O(n) \longrightarrow O(n^2)$

Better analysis:
randomized algs.
expected time=O(n)

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 ≤ 7/10 n elements.

Q. How to find approximate median in linear time ?

A. Recursively compute median of sample of ≤ 2/10 n elements

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

two sub-problems of different sizes!

$$\Longrightarrow \quad C(n) = \Theta(n)$$

to prove

Choosing the pivot element

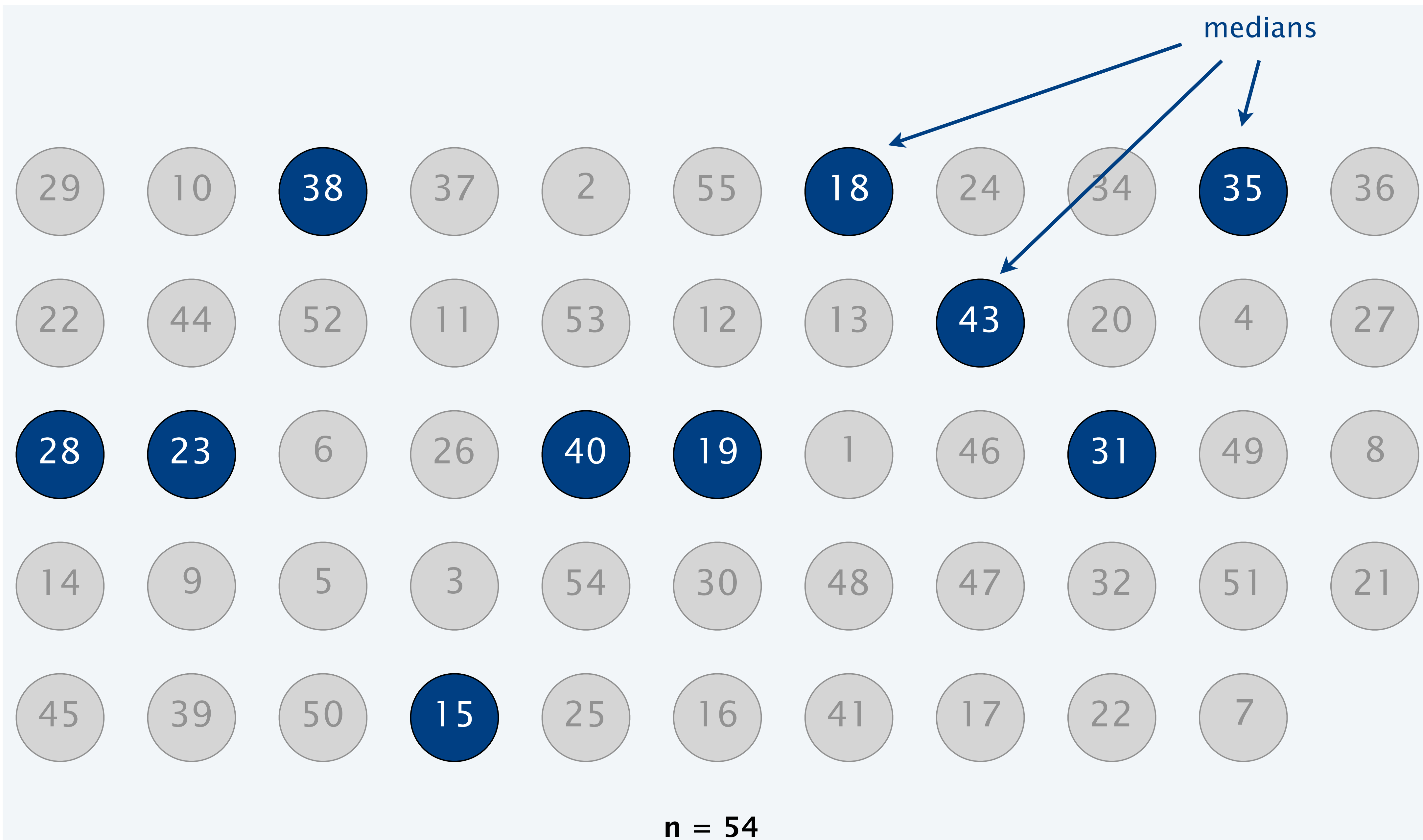
- Divide n elements into n/5 groups of 5 elements each (plus extra).
- Find median of each group (except extra).

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

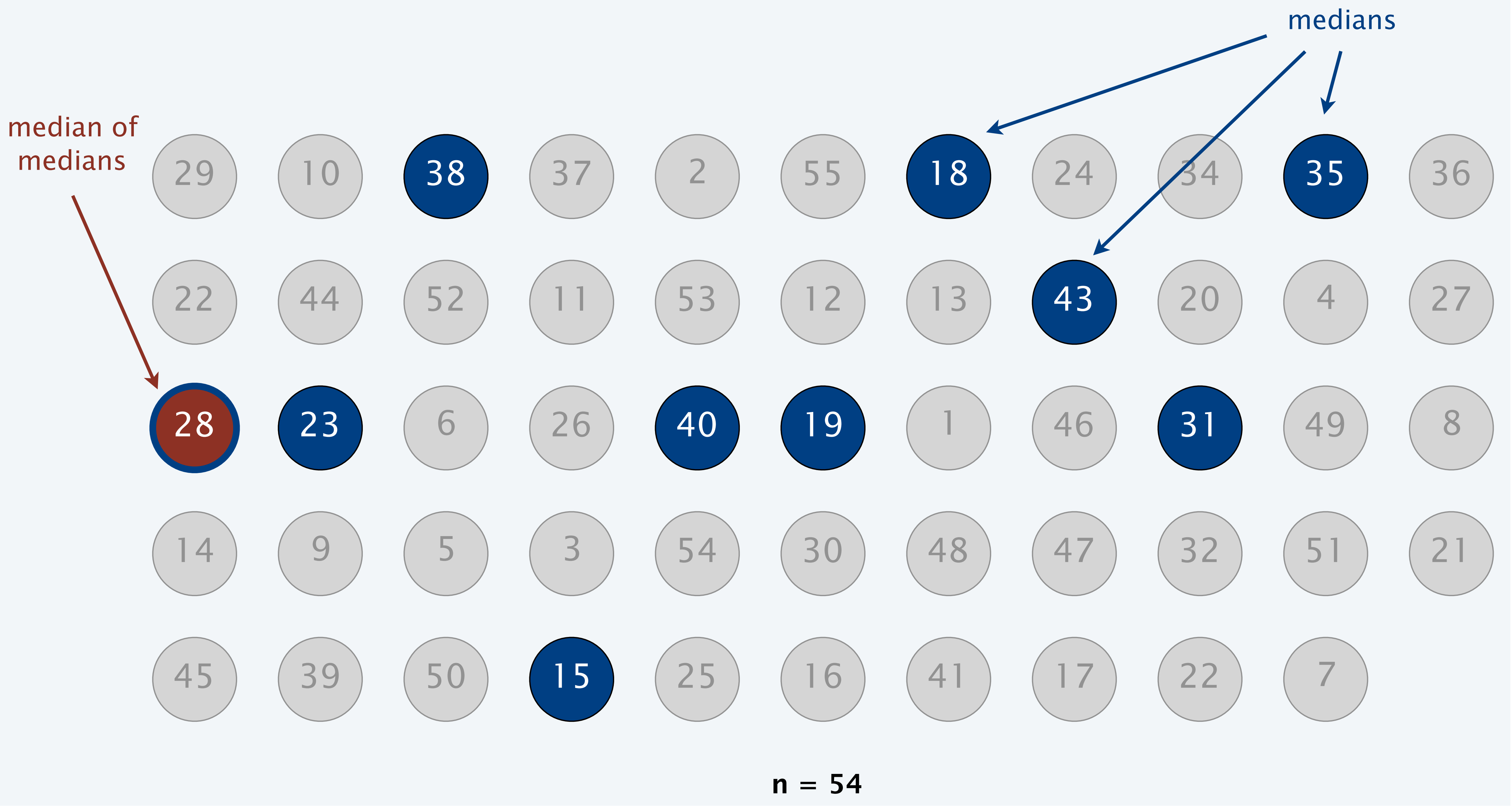
Choosing the pivot element

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



Choosing the pivot element

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



Median-of-medians selection algorithm

MOM-Select(A,k):

n ← |A|

if (n < 50):

return k-th smallest element of A via mergesort

Group A into n/5 groups of 5 elements each (ignore leftovers)

B ← median of each group of 5

p ← MOM-Select(B, n/10) ← median of medians

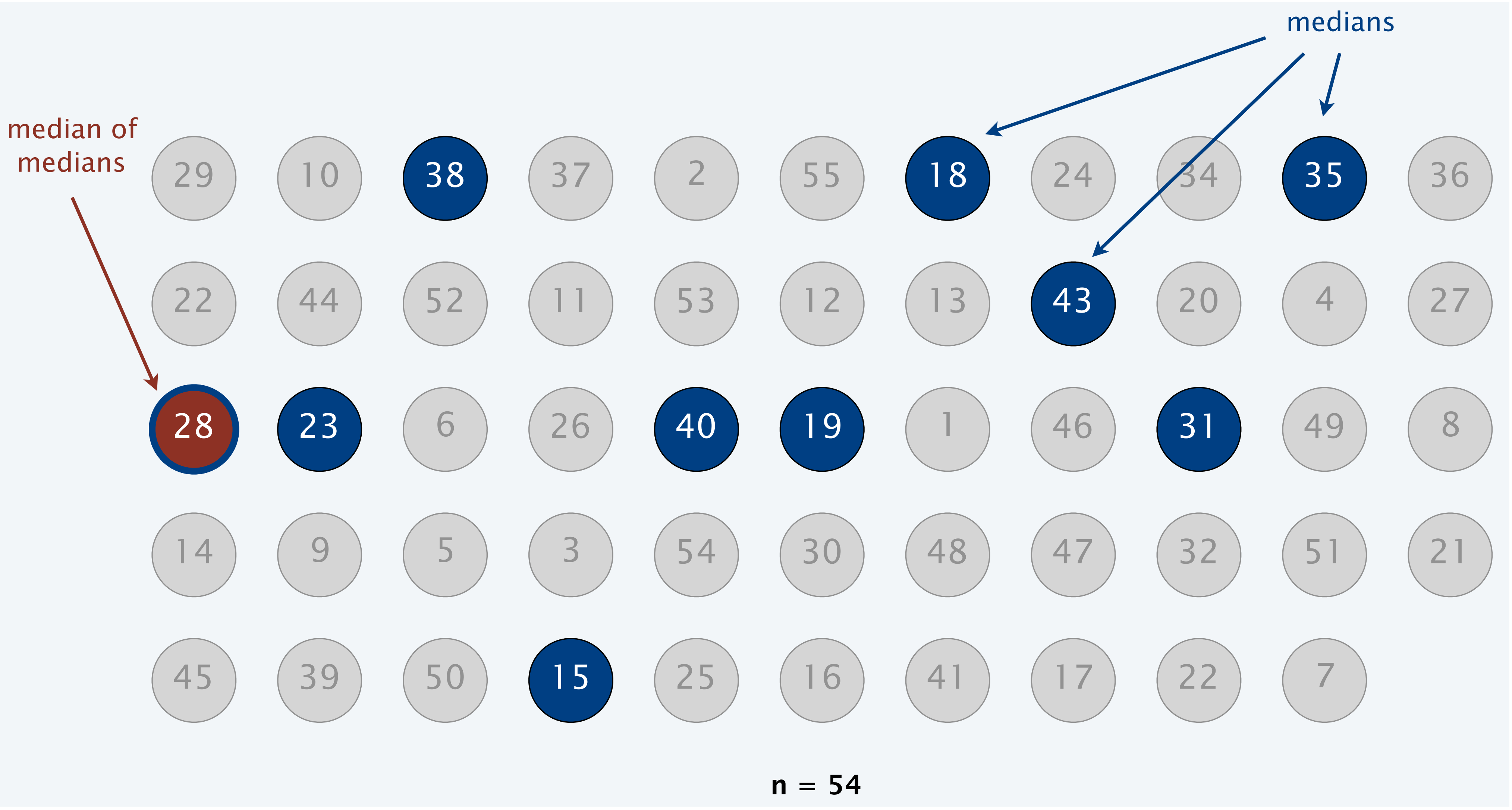
Median-of-medians selection algorithm

```
MOM-Select(A,k):
    n ← |A|
    if ( n < 50 ):
        return k-th smallest element of A via mergesort
    Group A into n/5 groups of 5 elements each (ignore leftovers)
    B ← median of each group of 5
    p ← MOM-Select(B, n/10)        ← median of medians
    (L, R) ← Partition(A,p)
    if ( k < |L| ) return MOM-Select(L,k)
    else if ( k > |L| ) return MOM-Select(R, k - |L|)
    else return p
```

Quick-select schema

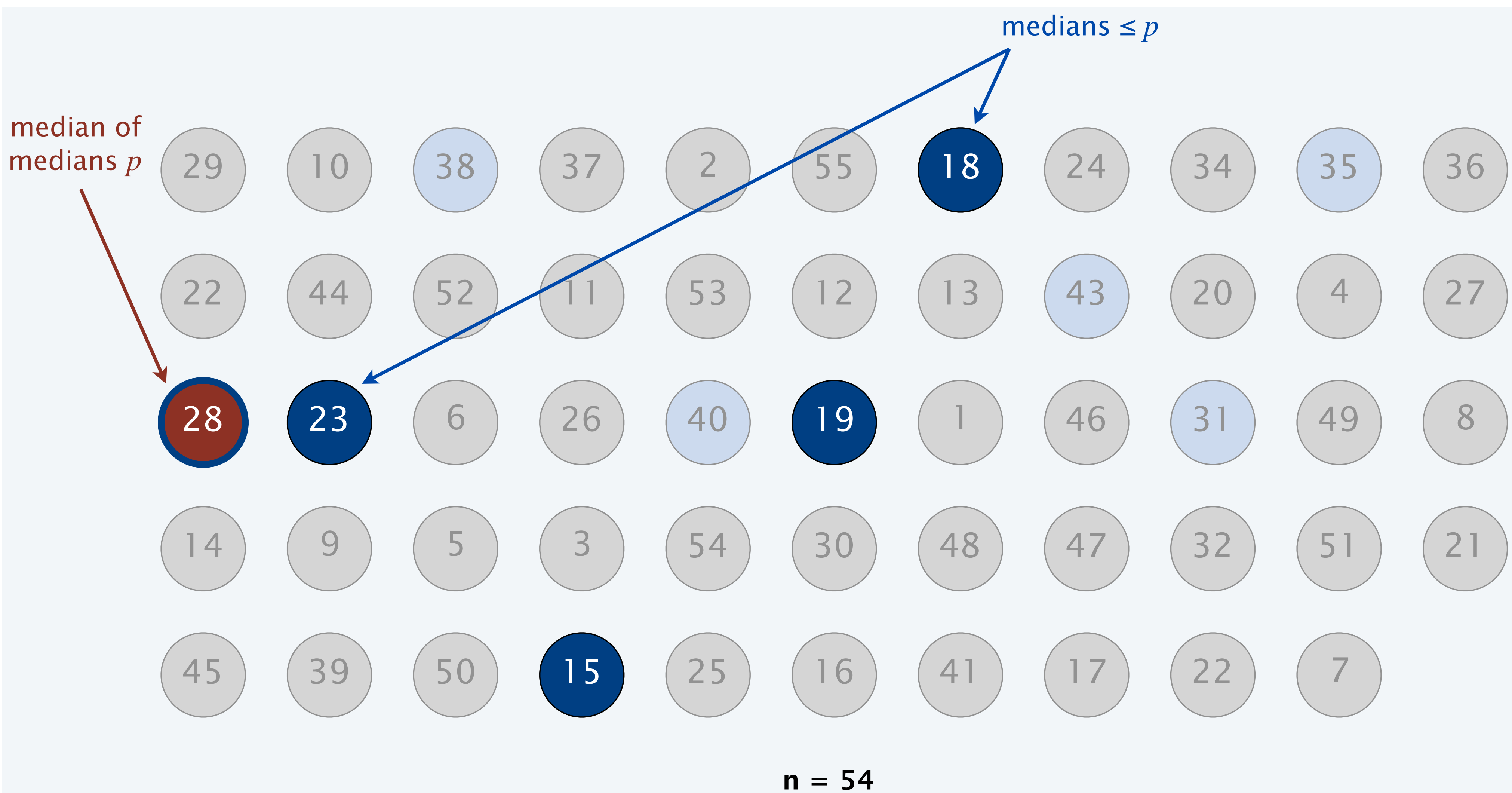
Complexity analysis

- At least half of 5-element medians ≤ p



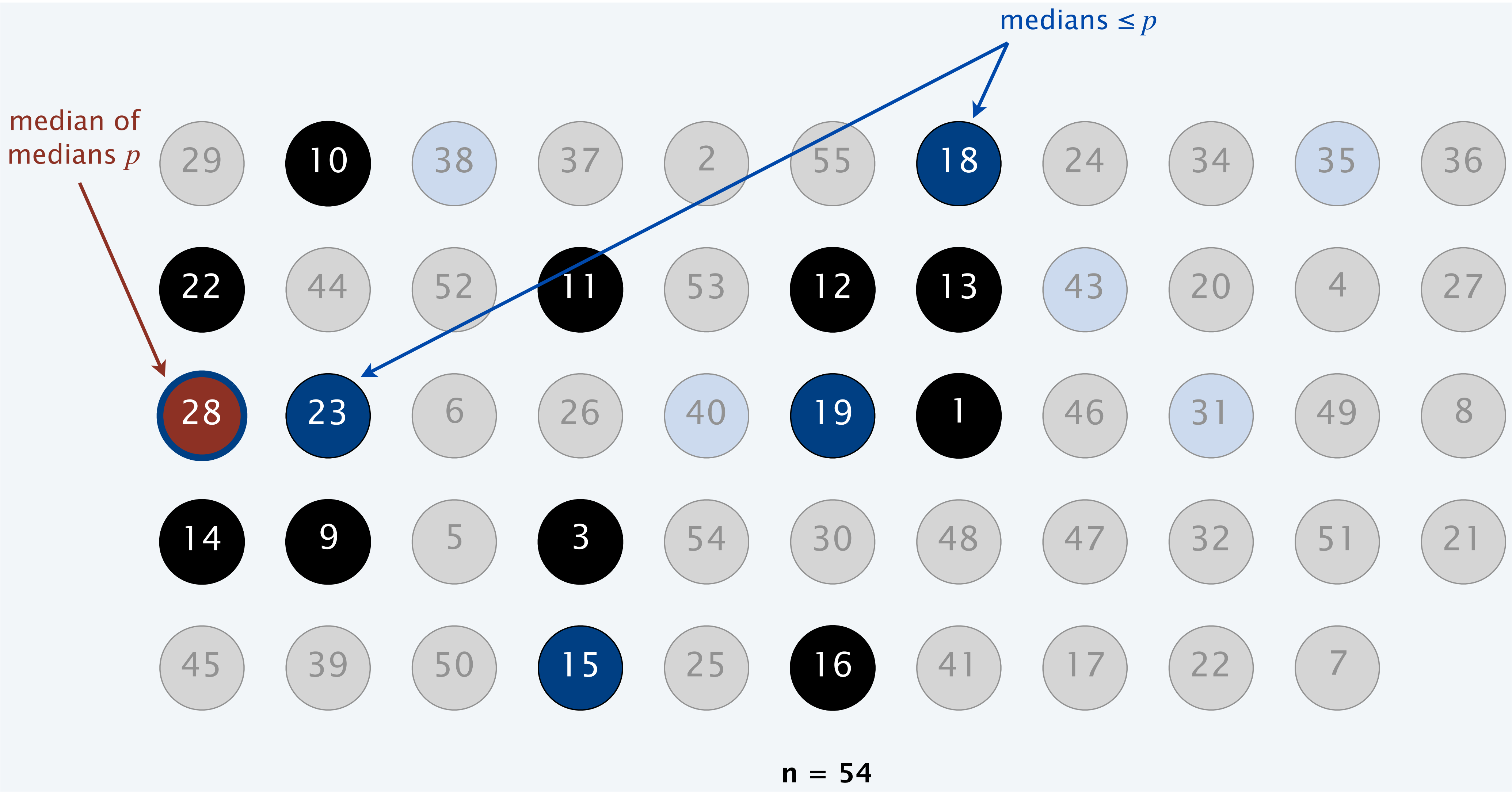
Complexity analysis

- At least half of 5-element medians ≤ p
- At least (n/5)/2 = n/10 medians ≤ p



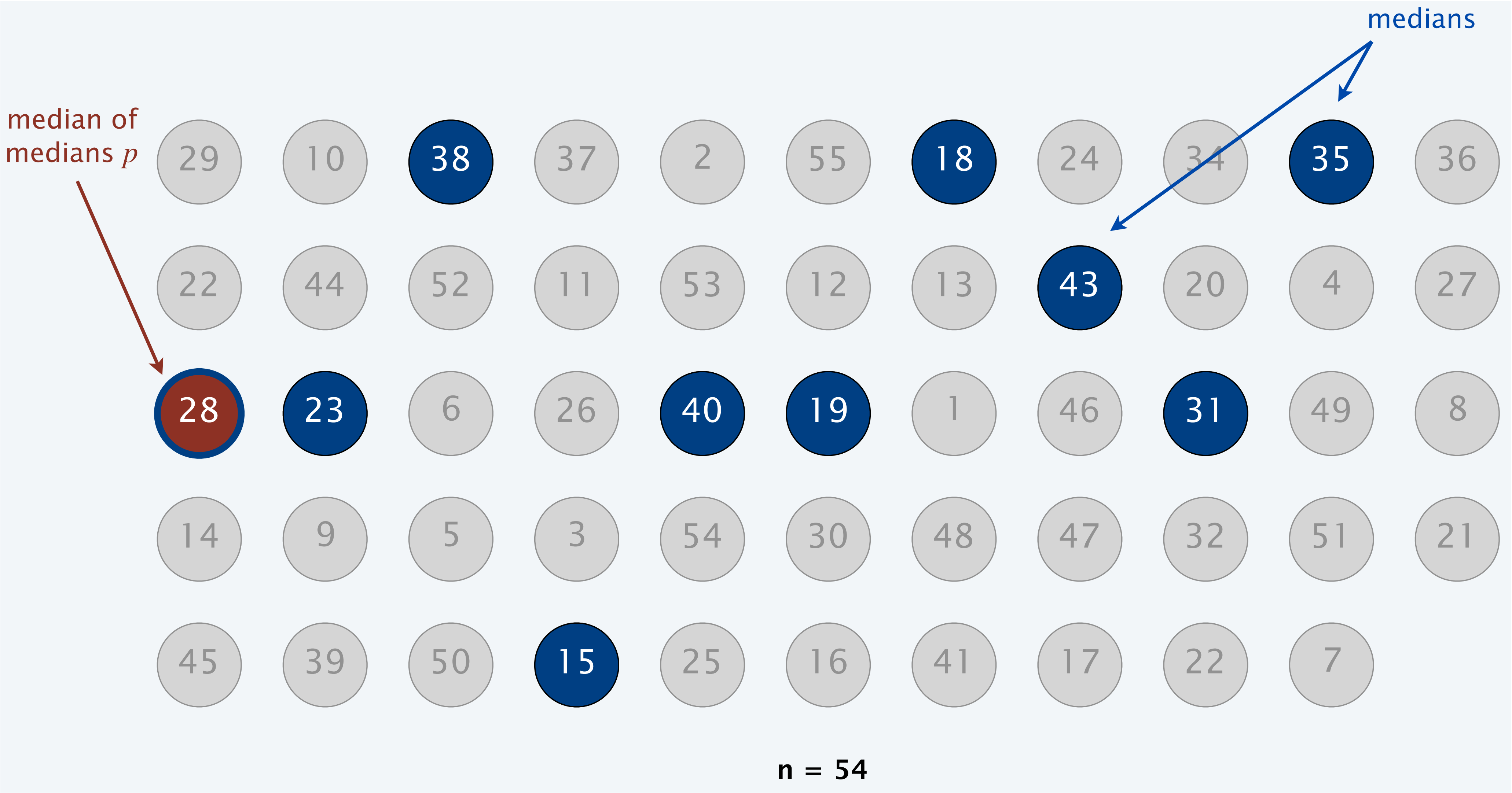
Complexity analysis

- At least half of 5-element medians ≤ p
- At least (n/5)/2 = n/10 medians ≤ p
- At least 3(n/10) elements ≤ p



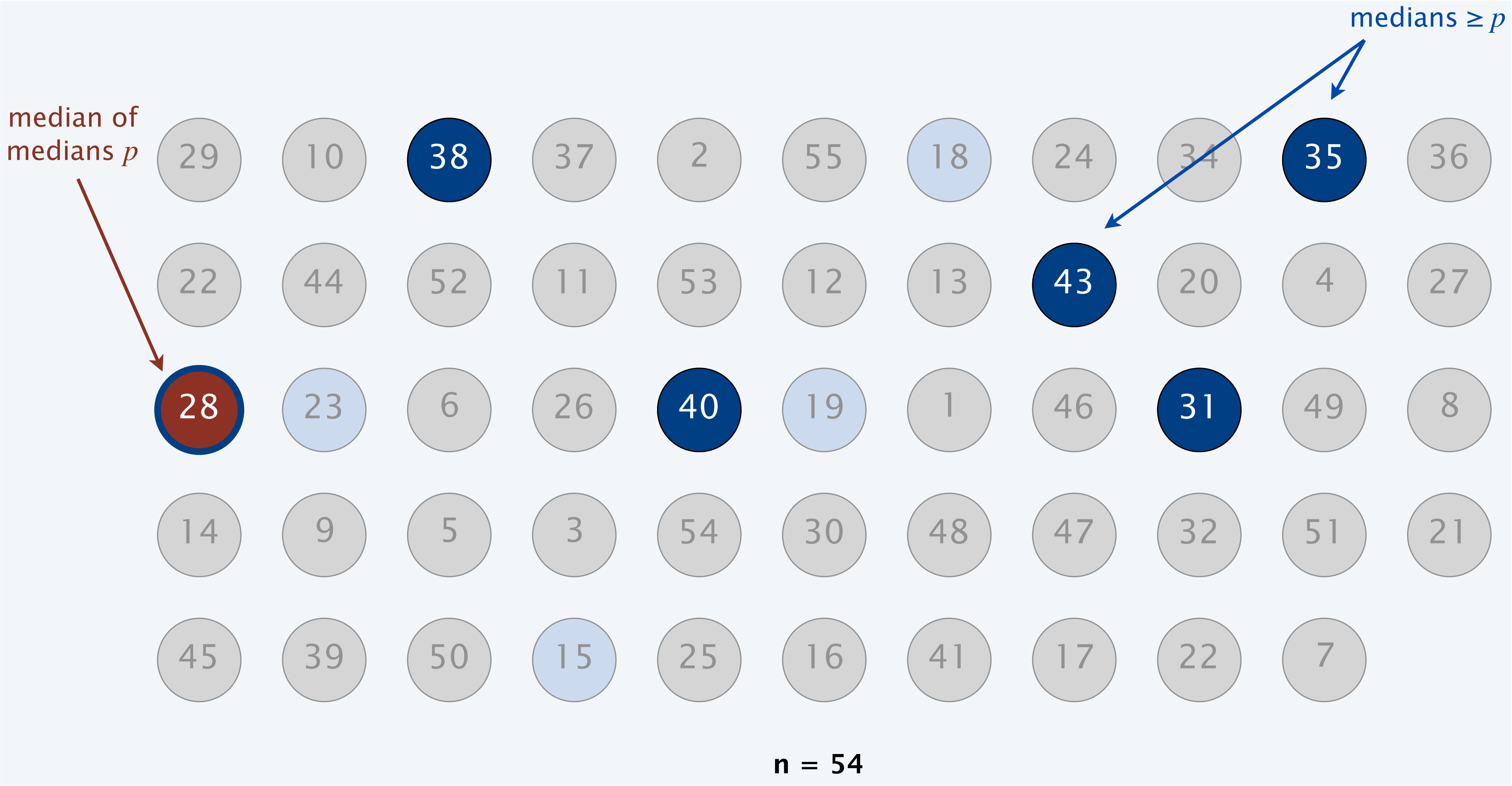
Complexity analysis

- At least half of 5-element medians ≥ p



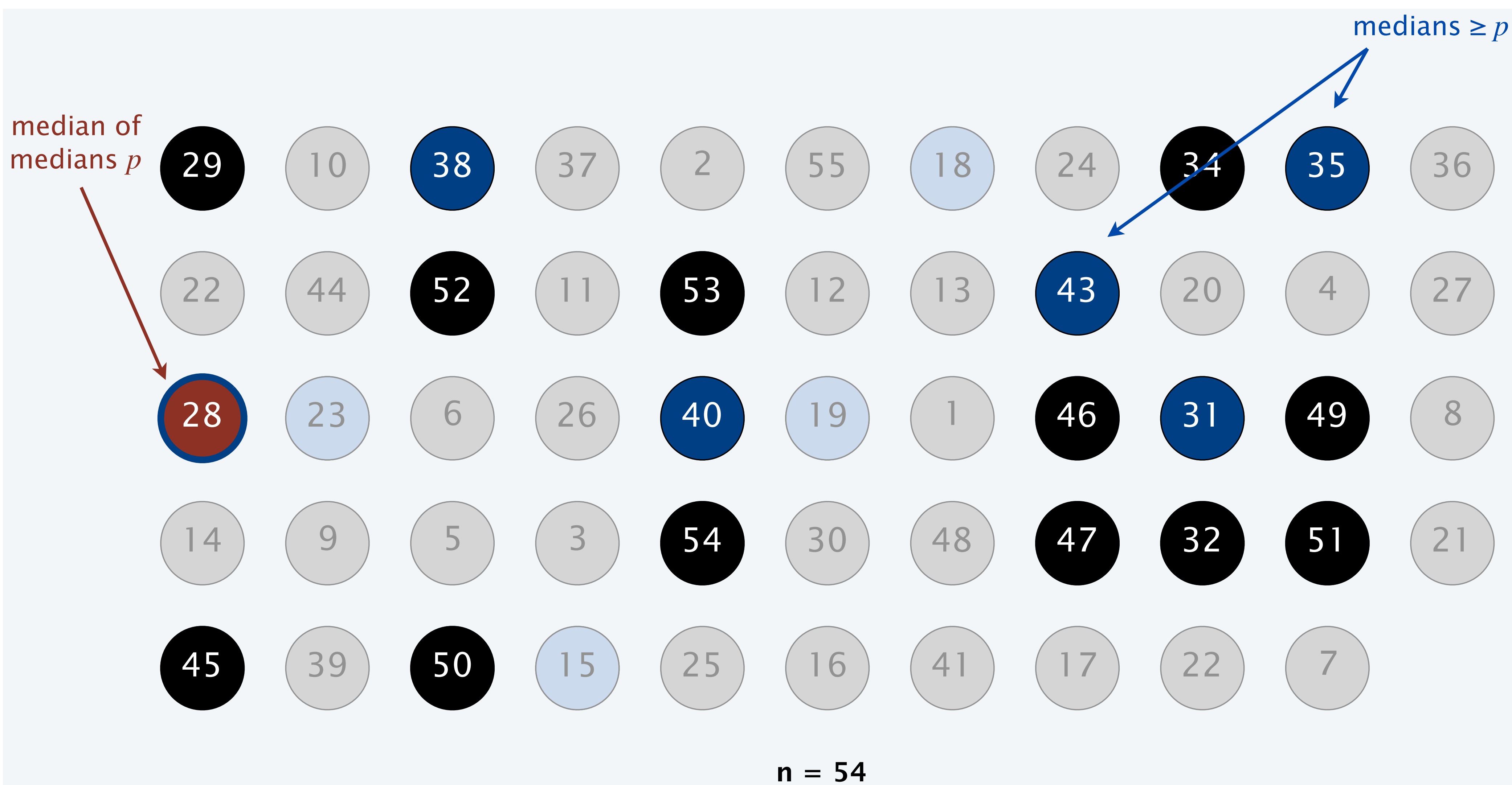
Complexity analysis

- At least half of 5-element medians $\geq$ p
- At least $(n/5)/2 = n/10$ medians $\geq$ p



Complexity analysis

- At least half of 5-element medians ≥ p
- At least (n/5)/2 = n/10 medians ≥ p
- At least 3(n/10) elements ≥ p



Recurrence

Median-of-medians selection algorithm recurrence.

- select called recursively with n/5 elements to compute MOM p
- at least 3/10 n elements ≤ p
- at least 3/10 n elements ≥ p
- select called recursively with at most n - (3/10 n) elements

$$C(n) \le \underbrace{C(\lfloor n/5 \rfloor)}_{\text{median of medians}} + \underbrace{C(n - 3\lfloor n/10 \rfloor)}_{\text{recursive select}} + \frac{11}{5}n$$

median of medians

recursive select

computing median of 5 (≤ 6 compares per group)

partitioning (≤ n compares)

Recurrence

$$C(n) \le C(1/5n) + C(7/10n) + \lambda n$$

C(x) + C(y) ≤ C(x+y) (super-additive)

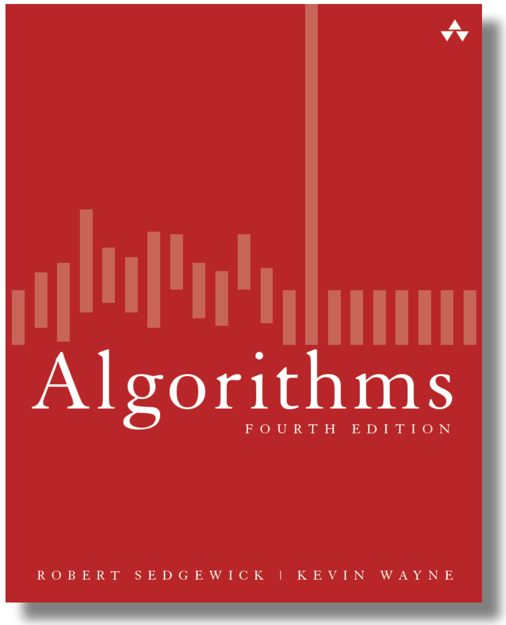
$$\begin{tikzcd}[column sep=tiny]
 & & & C(n) \arrow[dll] \arrow[drr] & & & & \\
 & C(1/5n) \arrow[dl] \arrow[dr] & & & & C(7/10n) \arrow[dl] \arrow[dr] & & \le C(\ (1/5 + 7/10)\ n) \\
C((1/5)^2 n) & & C(7/10 \cdot 1/5n) & & C(1/5 \cdot 7/10n) & & C((7/10)^2 n) & \le C(\ (1/5 + 7/10)^2\ n) \\
 & & & \cdots & & & & \le C(\ (1/5 + 7/10)^k\ n)
\end{tikzcd}$$

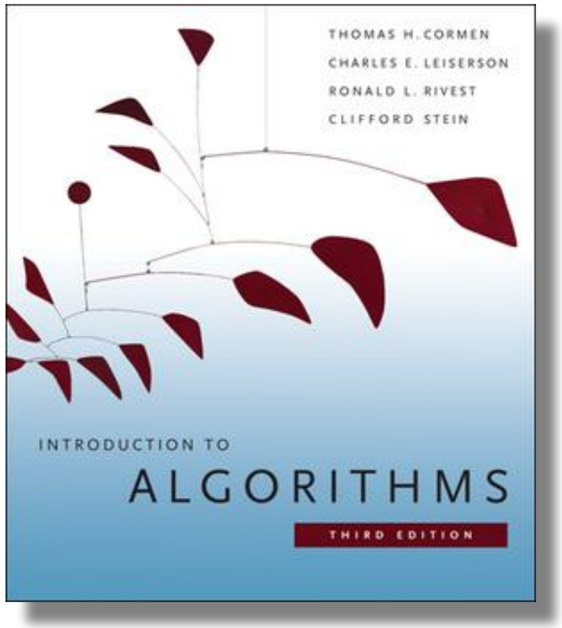
$$\begin{aligned}
C(n) &\le C(9/10n) + \lambda n \\
&\le C((9/10)^2 n) + \lambda n(1 + 9/10) \\
&\le C((9/10)^3 n) + \lambda n(1 + 9/10 + (9/10)^2) \\
&\quad \cdots \\
&\le C(n_0) + \lambda n(1 + 9/10 + (9/10)^2 + \ldots) = C(n_0) + \lambda n \cdot 10 = O(n)
\end{aligned}$$

References for this lecture

The slides are designed to be self-contained.

They were prepared using the following books that I recommend if you want to learn more:





Next

Next tutorial: Binary Search and friends

Next week: The end of divide-and-conquer
"Master theorem"; other examples

Feedback

Moodle

Questions: constantin.enea@polytechnique.edu