Fun Problems & Review

Announcements/Logistics

- We’ll meet here tomorrow at the normal time
- Bring something to write (and erase) with
    - You’ll be given a blank copy of the exact same***** exam
    - I’ll be around to answer questions
- Academic Accommodations?
    - It may be too late to make new arrangements, but we have several existing options that we can work with if you haven’t yet reached out

Today’s Goals

- **Wrap up** divide and conquer paradigm
- **Feel good** about tomorrow’s exam (or at least the material on it)
- **Practice** solving interesting/fun problems on topics we’ve covered so far
- **Connect** material to the larger course context

Selection

Selection: Problem Statement

Given an array $A[1,\ldots,n]$ of size n, find the kth smallest element for any $1 \le k \le n$

- Special cases: min $k = 1$, max $k = n$:
  - Linear time, $O(n)$
- What about **median** $k = \lfloor n+1 \rfloor / 2$?
  - Sorting: $O(n \log n)$
  - Binary heap: $O(n \log k)$

Question. Can we do it in $O(n)$?

- **Surprisingly yes.**
- Selection is easier than sorting.

Selection: Problem Statement

Example. Take this array of size 10:

$$A = 12|2|4|5|3|1|10|7|9|8$$

Suppose we want to find 4th smallest element

- First, take any pivot p from $A[1,\ldots n]$
- If p is the 4th smallest element, return it
- Else, we partition A around p and recurse

Selection Algorithm: Idea

Select (A, k):

If $|A| = 1$: return $A[1]$

Else:

- Choose a pivot $p \leftarrow A[1,\ldots,n]$; let r be the rank of p
- $r, A_{<p}, A_{>p} \leftarrow$ Partition((A, p))
- If $k == r$, return p
- Else:
  - If $k < r$: Select $(A_{<p}, k)$
  - Else: Select $(A_{>p}, k - r)$

Selection: Problem Statement

Example. Take this array of size 10:

$$A = 12\,|\,2\,|\,4\,|\,5\,|\,3\,|\,1\,|\,10\,|\,7\,|\,9\,|\,8$$

Suppose we want to find 4th smallest element

- Choose pivot 8
- What is its rank?
  - Rank 7
- So let's find all of the smaller elements of A:
  - $A' = 2\,|\,4\,|\,5\,|\,3\,|\,1\,|\,7$
- Want to find the element of rank 4 in this new array

Selection: Problem Statement

Example. Take this array of size 10:

$$A = 12 | 2 | 4 | 5 | 3 | 1 | 10 | 7 | 9 | 8$$

Suppose we want to find 4th smallest element

- Choose as pivot 3
- What is its rank?
  - Rank 3
- So let's find all of the **larger** elements of A:
  - $A' = 12 | 4 | 5 | 10 | 7 | 9 | 8$
- Want to find the element of rank $4 - 3 = 1$ in this new array

When is this method good?

- If we guess the pivot right! (but we can't always do that)
- If we partition the array pretty evenly (the pivot is close to the middle)
  - Let's say our pivot is not in the first or last $3/10$ths of the array
  - What is our recurrence?
  - $T(n) \leq T(7n/10) + O(n)$
  - $T(n) = O(n)$

Our high-level goal

- Find a pivot that's close to the median—has a rank between $3n/10$ and $7n/10$, in time $O(n)$
- But the array is unsorted? How do we do that?
- Want to *always* be successful

Finding an Approximate Median

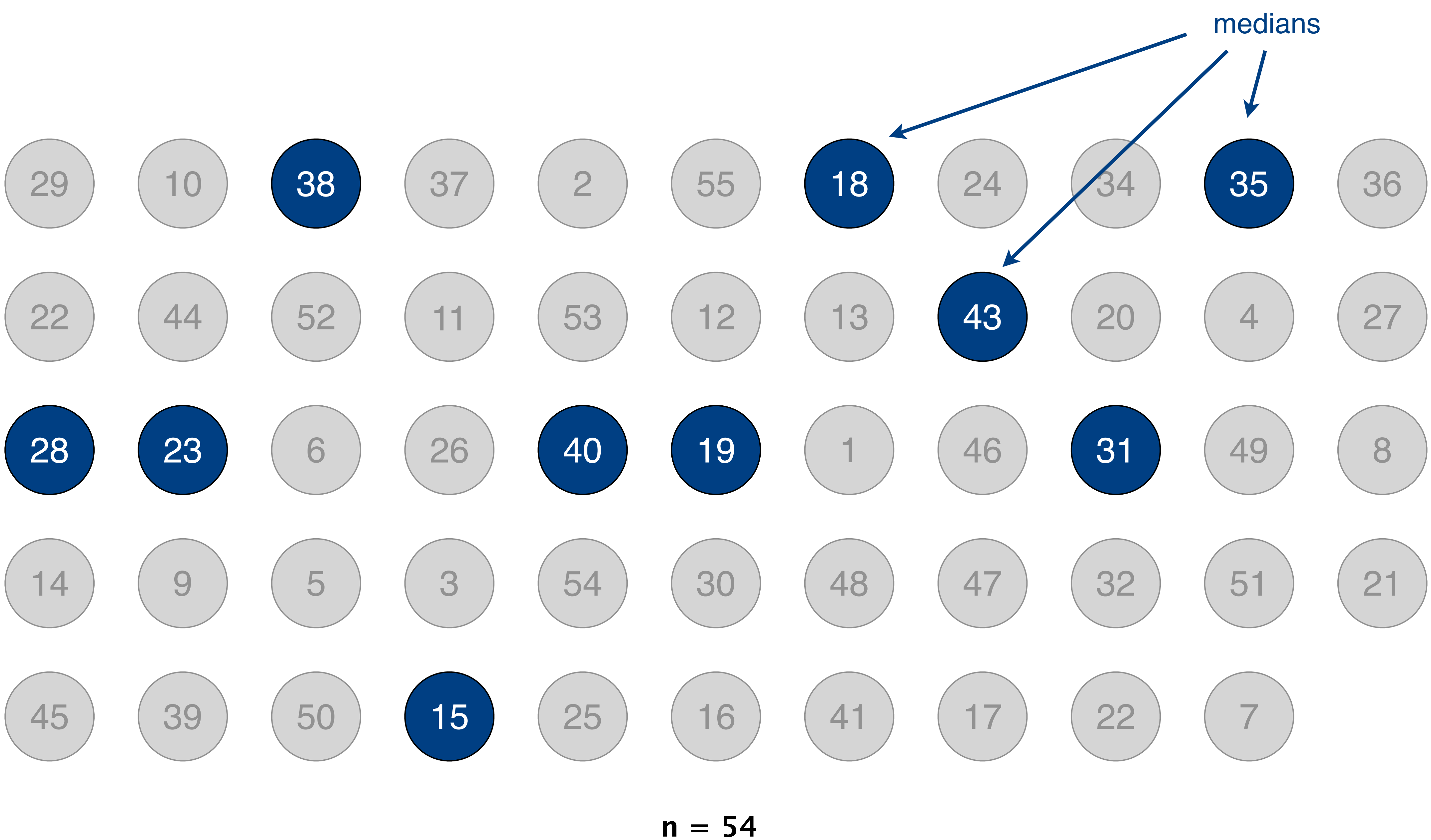
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group

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

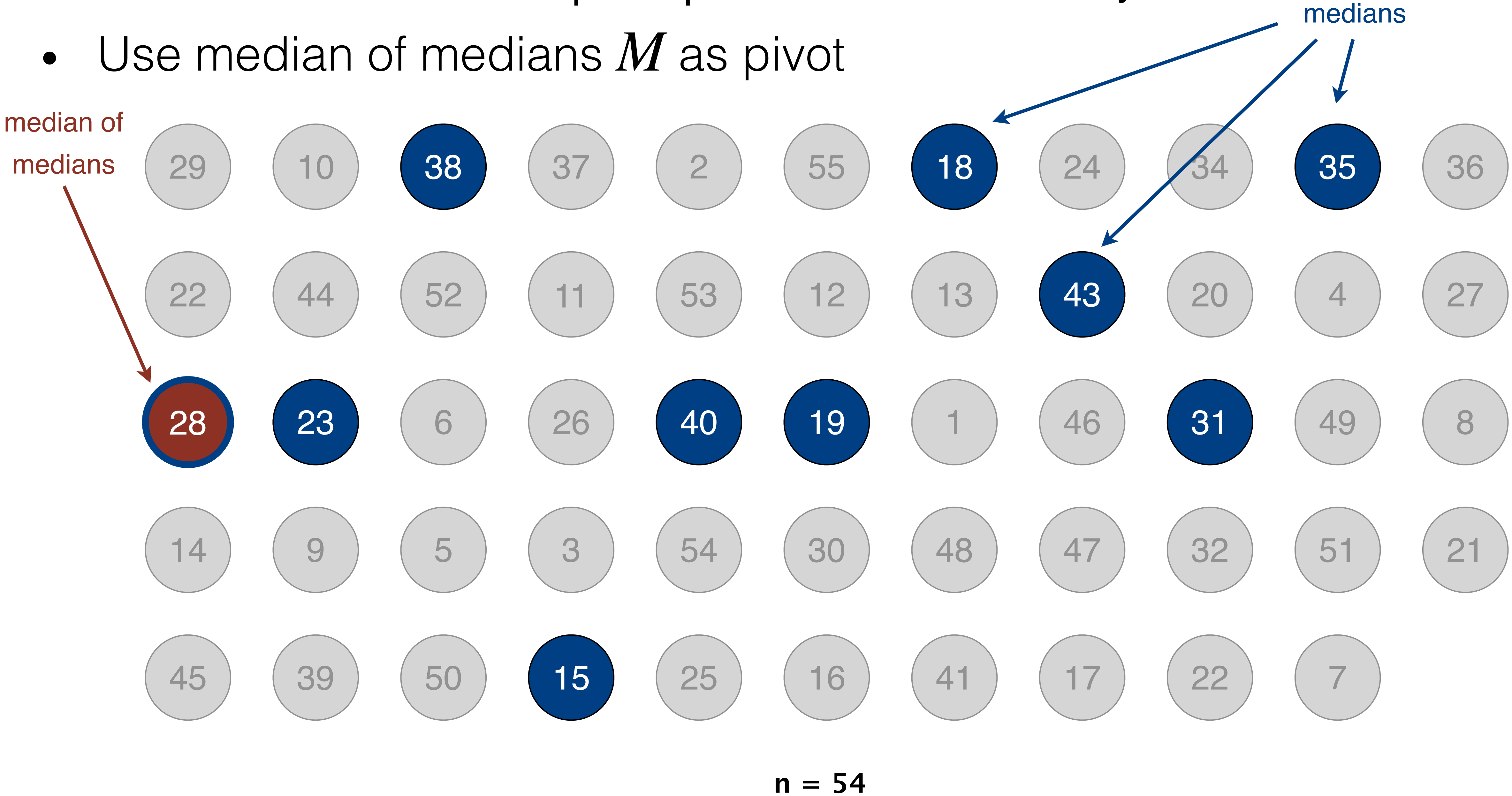
Finding an Approximate Median

- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group



Finding an Approximate Median

- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group
- Find $M \leftarrow$ median of $\lceil n/5 \rceil$ medians recursively
- Use median of medians M as pivot

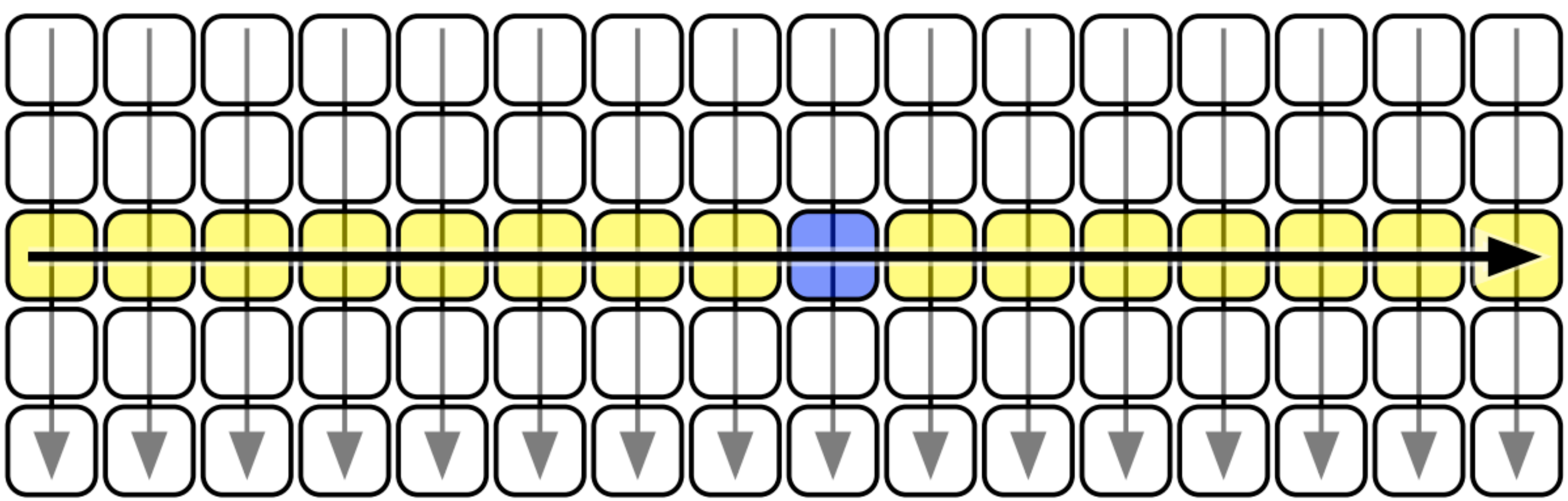


What did we gain?

- How can I show that the median of medians is “close to the center” of the array?
- What elements can I say, for sure, are $\leq$ the median of medians?
    - The smaller half of the medians
    - $n/10$ elements
- Any other elements?
    - Another 2 elements in each median’s list

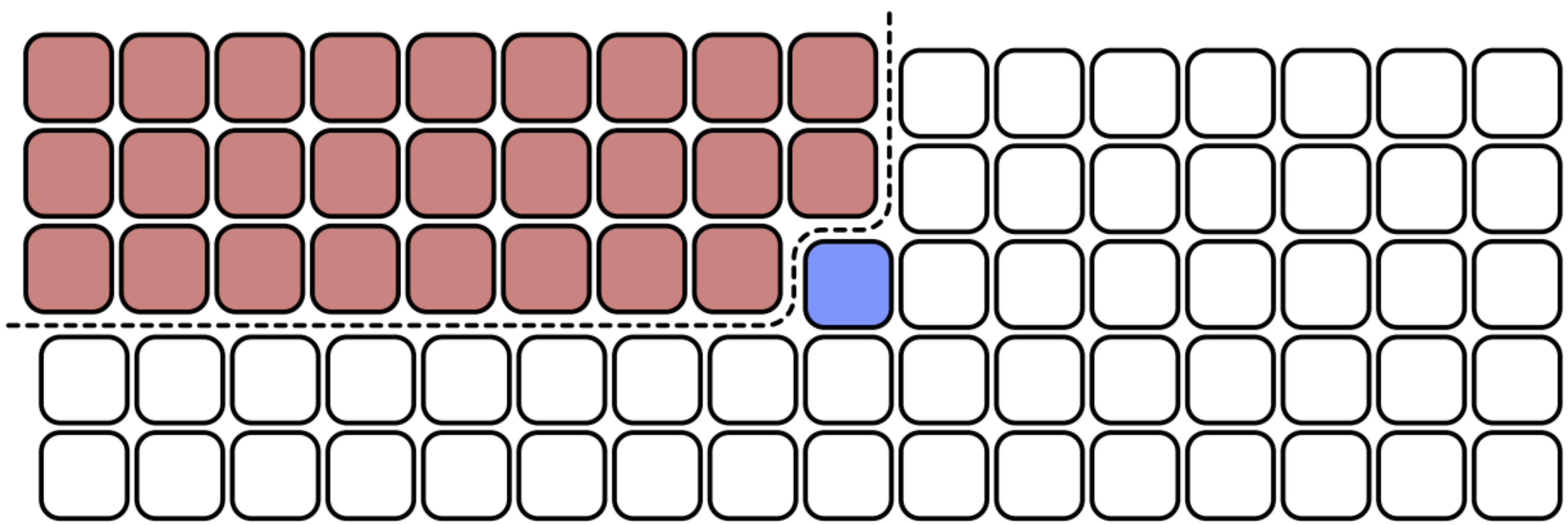
Visualizing MoM

- In the 5 x n/5 grid, each column represents five consecutive elements
- **Imagine** each column is sorted top down
- **Imagine** the columns as a whole are sorted left-right
    - We don't actually do this!
- MoM is the element closest to center of grid



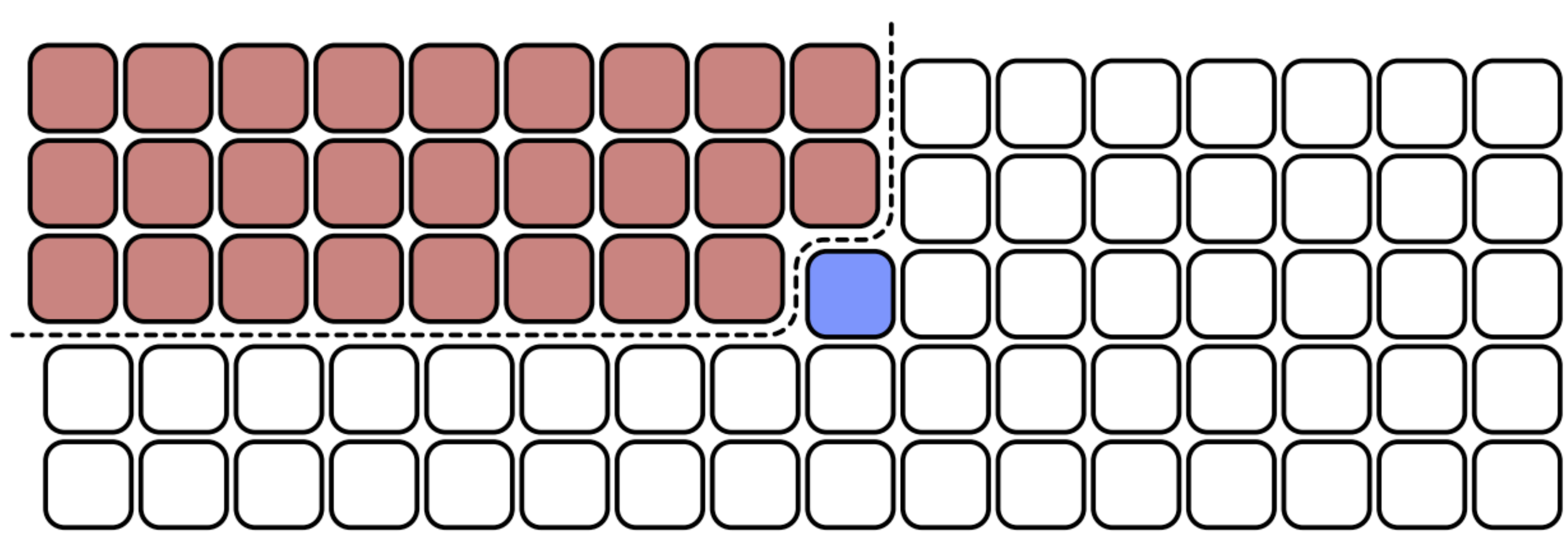
Visualizing MoM

- Red cells (at least $3n/10$) are smaller than M



Visualizing MoM

- Red cells (at least $3n/10$) in size are smaller than M
- If we are looking for an element larger than M, we can throw these out, before recursing
- Symmetrically, we can throw out $3n/10$ elements larger than M if looking for a smaller element
- Thus, the recursive problem size is at most $7n/10$



How Good is Median of Medians

Claim. Median of medians M is a good pivot, that is, at least $3/10$th of the elements are $\geq M$ and at least $3/10$th of the elements are $\leq M$.

Proof.

- Let $g = \lceil n/5 \rceil$ be the size of each group.
- M is the median of g medians
  - So $M \geq g/2$ of the group medians
  - Additionally, each median is greater than 2 elements in its group
  - Thus $M \geq g/2 + 2g/2 = 3g/2 = 3n/10$ elements
- Symmetrically, $M \leq 3n/10$ elements. ∎

Median of Medians Subroutine

MoM(A, n):

If $n == 1$: return $A[1]$

Else:

Divide A into $\lceil n/5 \rceil$ groups

Compute median of each group

$A' \leftarrow$ group medians

Mom(A', $\lceil n/5 \rceil$)

$T(n/5) + O(n)$

Linear time Selection

Select (A, k):

If $|A| = 1$: return $A[1]$;

Else:

Call median of medians to find a good pivot

$p \leftarrow$ MoM(A, n); $n = |A|$

$T(n/5) + O(n)$

$r, A_{<p}, A_{>p} \leftarrow$ Partition((A, p))

Larger subproblem has size $\leq 7n/10$

If $k == r$, return p

Else:

If $k < r$: Select $(A_{<p}, k)$

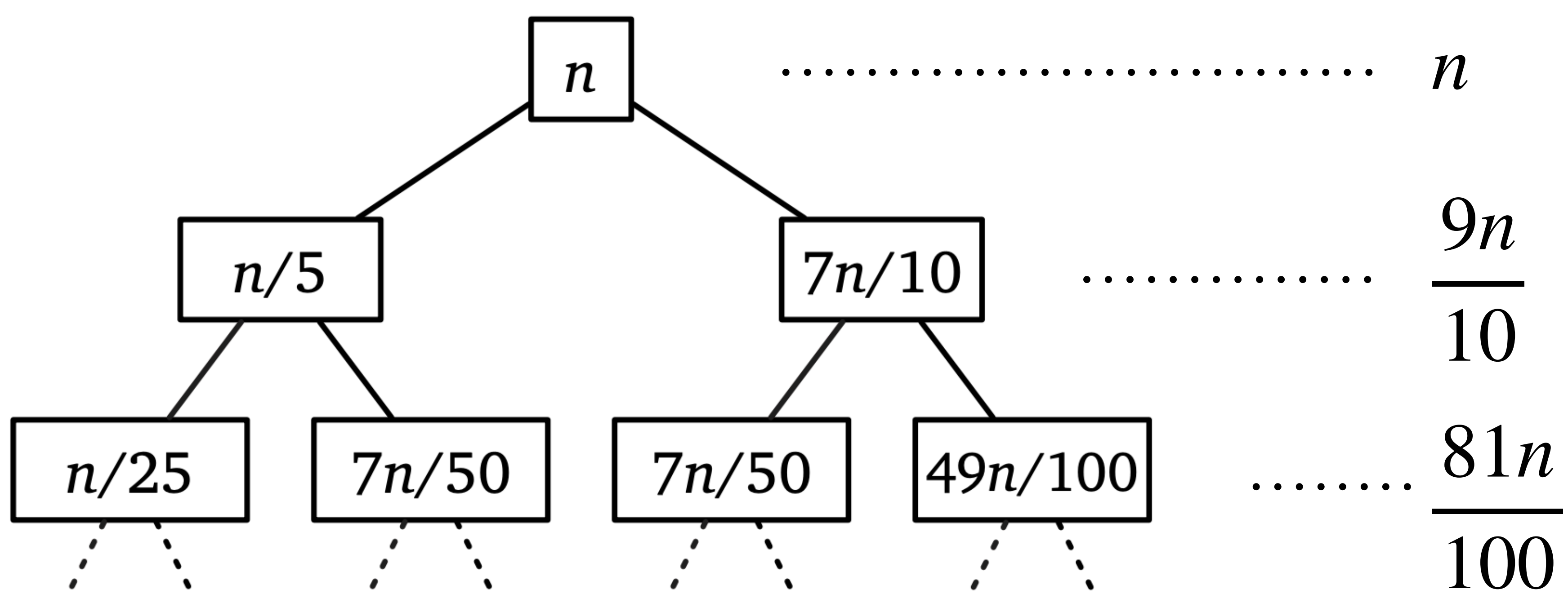
Else: Select $(A_{>p}, k - r)$

Overall: $T(n) = T(n/5) + T(7n/10) + O(n)$

Selection Recurrence

Okay, so we have a good pivot, but…

- We are still doing two recursive calls
  - $T(n) \leq T(n/5) + T(7n/10) + O(n)$
- Key: total work at each level still goes down!
- Decaying series gives us : $T(n) = O(n)$



Why the Magic Number 5?

- What was so special about 5 in our algorithm?
- It is the smallest odd number that works!
  - (Even numbers are problematic for medians)
- Let us analyze the recurrence with groups of size 3
  - $T(n) \leq T(n/3) + T(2n/3) + O(n)$
  - Work is equal at each level of the tree!
  - $T(n) = \Theta(n \log n)$

Theory vs Practice

- $O(n)$-time selection by [Blum–Floyd–Pratt–Rivest–Tarjan 1973]
  - Does $\leq 5.4305n$ compares
- Upper bound:
  - [Dor–Zwick 1995] $\leq 2.95n$ compares
- Lower bound:
  - [Dor–Zwick 1999] $\geq (2 + 2^{-80})n$ compares.
- Constants are still too large for practice
- Random pivot works well in most cases!
  - We may analyze this when we do randomized algorithms

Fun Problems!

Choose your own adventure

- Greedy coin changing
- Divide and conquer majority element

Majority Element

In an n-element array A, a **majority element** is an element $e \in A$, such that e appears more than $n/2$ times.

- Can there be more than one majority element?
- Does every array have a majority element?

Problem: Given an n-element array A, find the majority e if one exists, or report (correctly) that there is no majority element

Majority Element

Problem: Given an n-element array A, find the majority e if one exists, or report (correctly) that there is no majority element

Strategy: **Divide and conquer**!

- Not the only (or even the fastest) way, but it is the most fun :)
- Subproblems?
  - Size?
  - How to combine?
    - Suppose we have the solutions (majority elements from) our subproblems. What can we say about a majority element in A with respect to those solutions?

Making Change

Pretend you paid for something using **cash**. What is the algorithm to return change in US currency using the **minimum** number of coins?

- It is greedy!
- To make change for $\$r$, start with biggest denomination less than $\$r$, subtract and repeat

Making Change

The greedy change algorithm is **optimal** for US coins!

But it is **not optimal** in general. Ideas why not? **(Counterexample?)**

- Imagine 25c, 20c, 10c, 5c, 1c coins
- How do you make change for 40c?
  - Greedy?
    - 25c, 10c, 5c
  - Optimal?
    - 20c, 20c

An Optimal Greedy Example

- How to prove that a greedy solution is optimal?
  - Exchange argument!
- Normally, “schedule” each coin, but can use slightly different notation. Why?
  - Once we “give out a type of coin”, greedy moves to the next denomination
  - Let $S = \{s_1, s_2, s_3, s_4\}$ be the number of Quarters, Dimes, Nickels, and Pennies returned by algorithm S for a dollar value $\$r$
  - We can write greedy’s coin count of each type as follows:
    - $g_1 = \lfloor r/25 \rfloor$, and let $r_q = r \mod 25$ // g_1: quarters
    - $g_2 = \lfloor r_q/10 \rfloor$, and let $r_d = r_q \mod 10$ // g_2: dimes
    - $g_3 = \lfloor r_d/5 \rfloor$, and let $r_n = r_d \mod 5$ // g_3: nickels
    - $g_4 = r_n$ // g_4: give any remaining change out as pennies

Exchange Argument Sketch

- Let $G = \{g_1, g_2, g_3, g_4\}$ be the number of Quarters, Dimes, Nickels, and Pennies returned by greedy for a dollar value $\$r$
- Let $O = \{o_1, o_2, o_3, o_4\}$ be the number of Quarters, Dimes, Nickels, and Pennies returned by optimal for a dollar value $\$r$

We'll do induction on denomination $i \in \{0,1,2,3,4\}$. We'll show how to exchange any o_i with g_i by exchanging coins in O's schedule at some denomination $j > i$ for coins in denomination i and also reducing the total number of coins used. (Base case?)

- By definition, $o_1 \leq g_1$, since greedy gives the maximum number of quarters it can
    - If $g_1 = o_1$, we are done with this denomination
    - If $g_1 > o_1$, we can convert O to $O' = \{g_1, o_2', o_3', o_4'\}$ by exchanging other coins in $o_i \in O \,|\, j > i$ for quarters such that $g_1 = o_1$, and also reducing the total number of coins in O'.
        - If $o_2 \geq 3$, we can replace 3 dimes for a quarter and a nickel
        - If $o_2 < 3$, we can show how to replace some combination of dimes, nickels, and pennies with a quarter …

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)