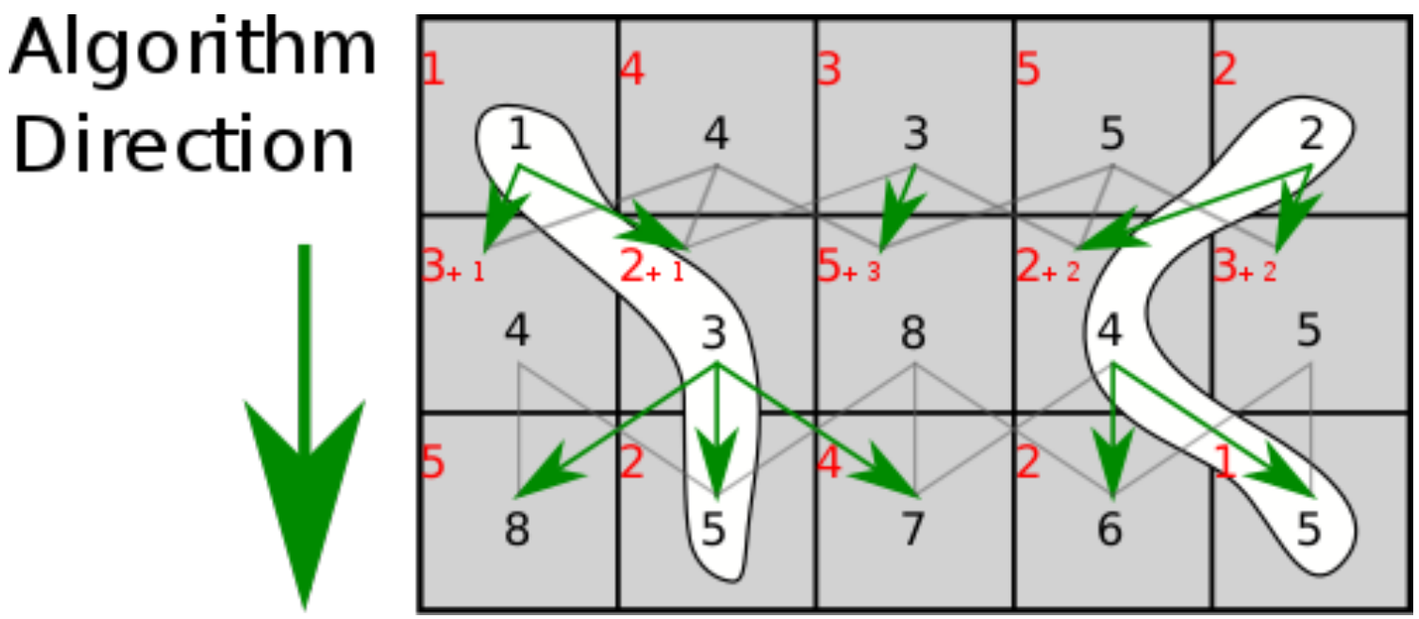
Selection and Intro to Dynamic Programming



Admin

- Assignment 4 due tomorrow (Saturday), Oct 17
- Assignment 5 due next Saturday, Oct 24
  - Shorter problem set, 3 problems
  - I'll get Assignment 5 and graded Assignment 3 back as soon as possible
- Midterm prep resources announced soon

October

November 2015

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12 Columbus Day	13	14	15	16	17 A4
18	19	20	21	22	23	24 A5
25	26	27	28 M	29	30	31 Halloween

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

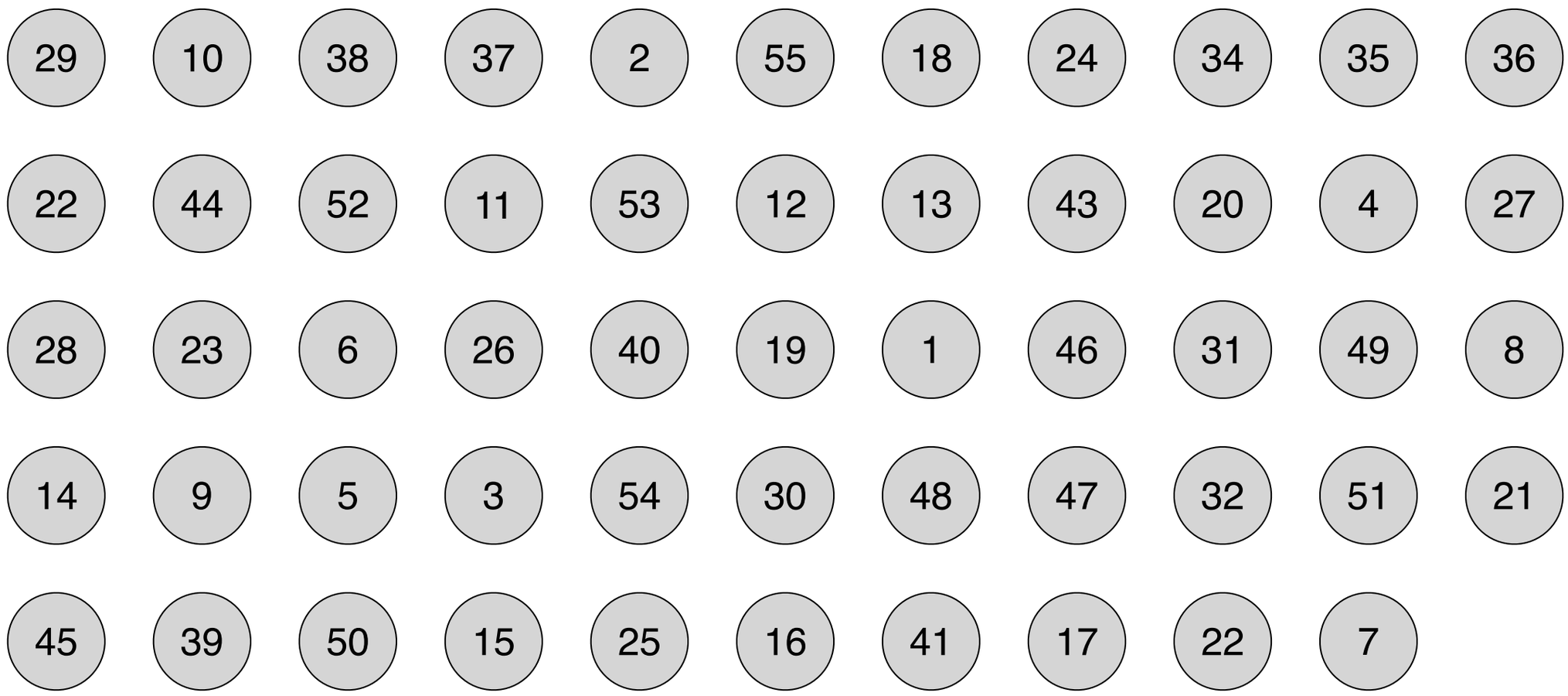
- Find a pivot that's close to the median—-has a rank between $3n/10$ and $7n/10$, in time $O(n)$

- But the array is unsorted? How do we do that?

- Want to *always* be successful

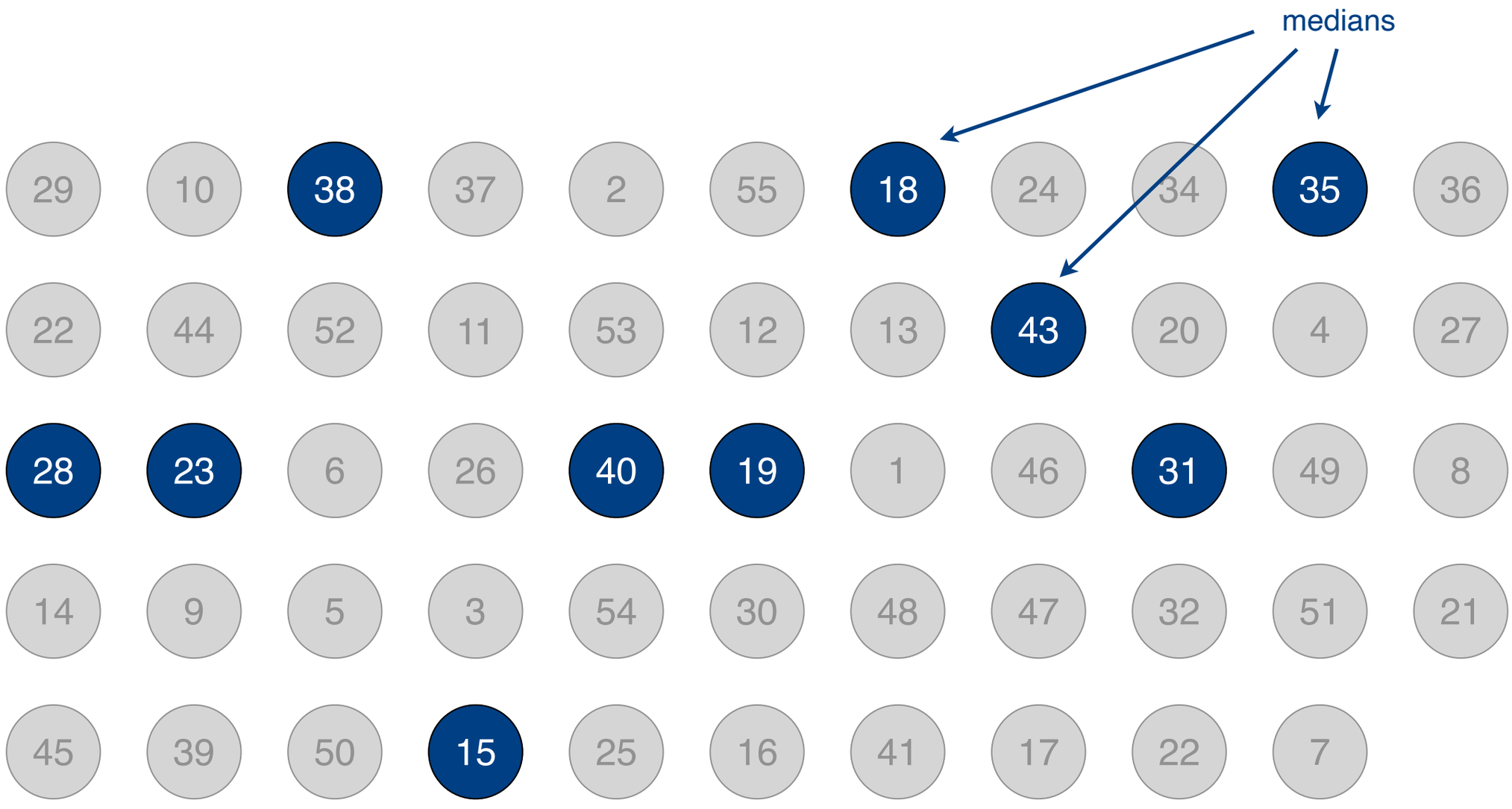
Finding an Approximate Median

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



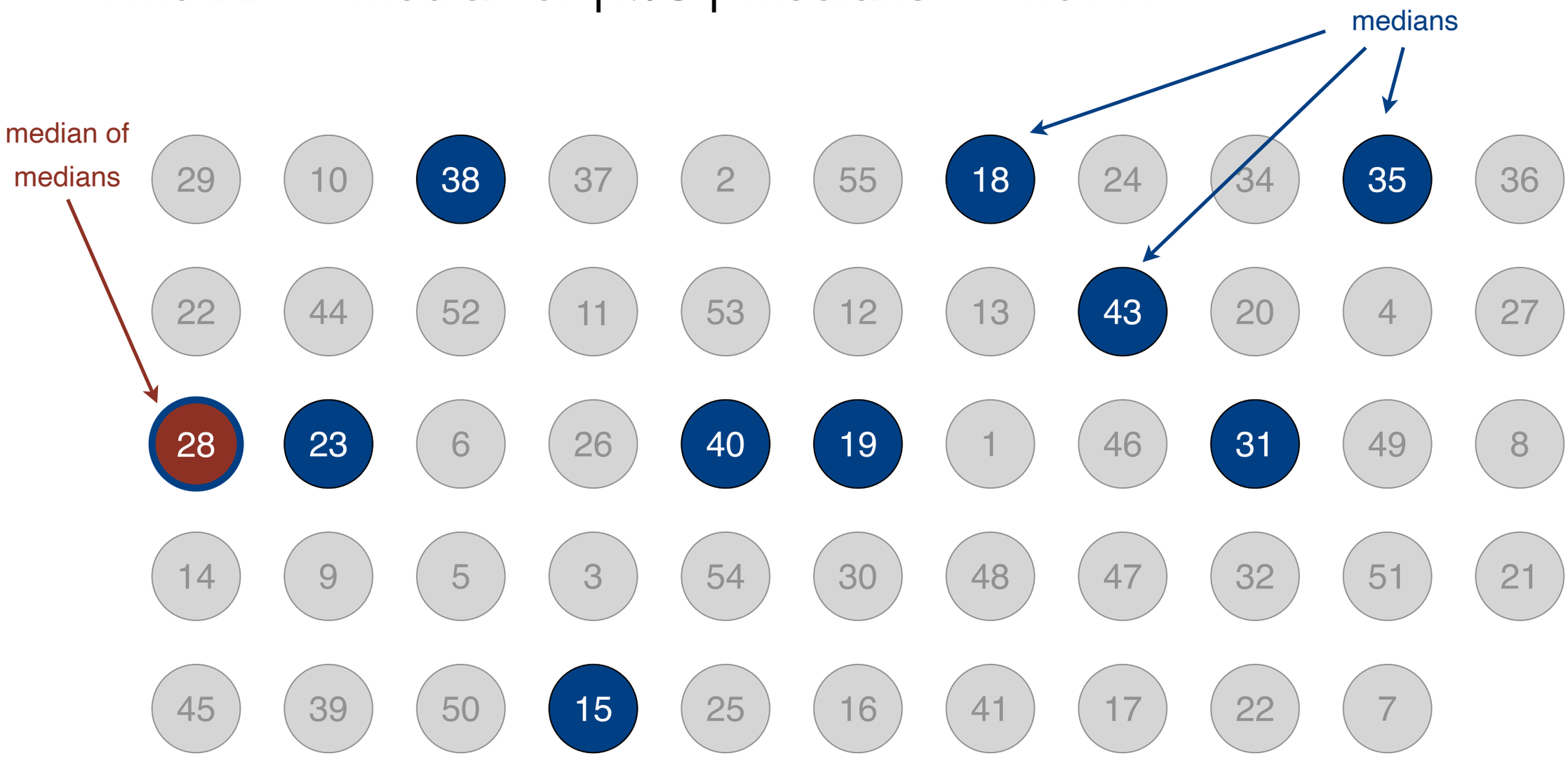
Finding an Approximate Median

- 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

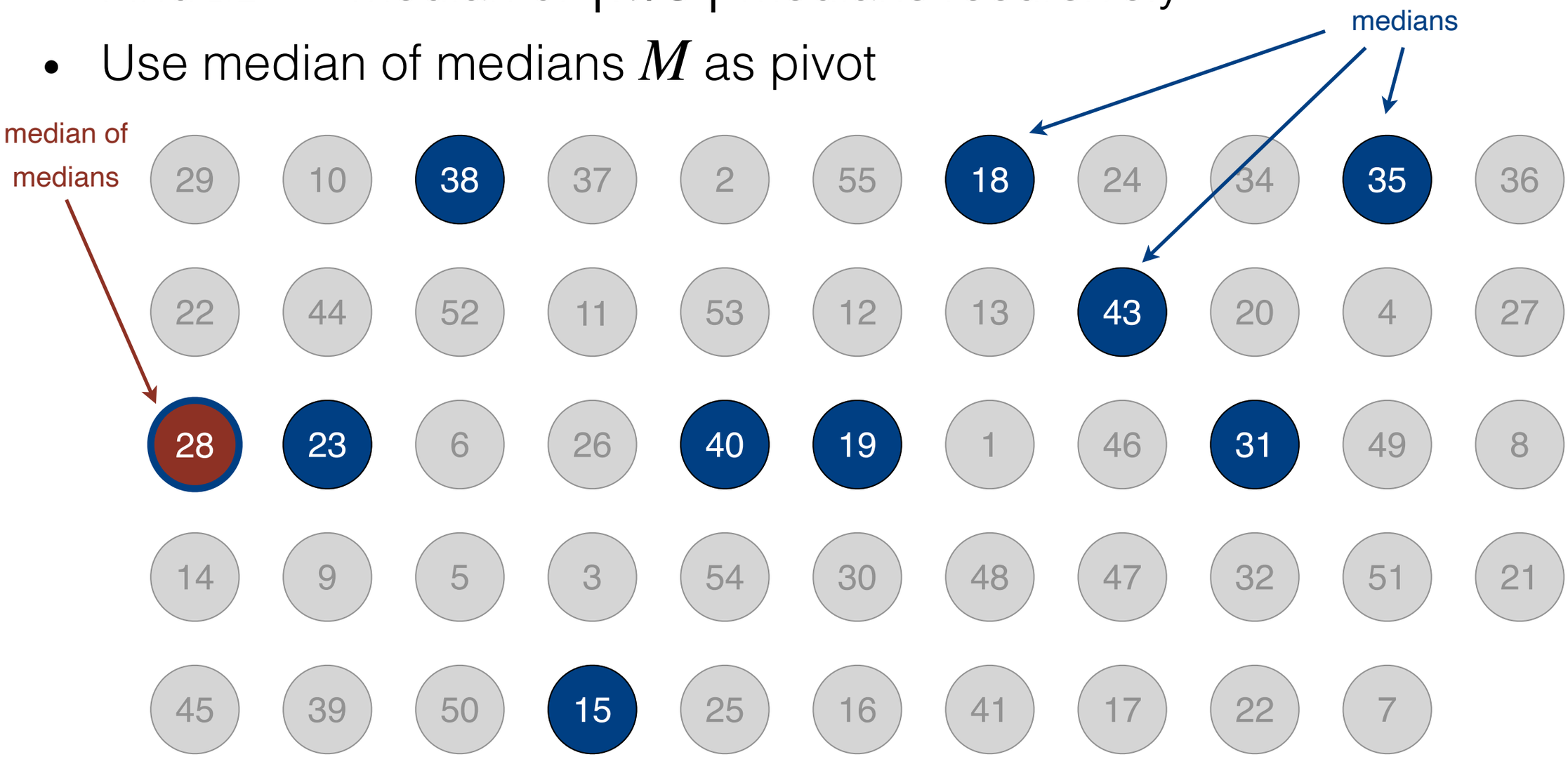
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group
- Find $M \leftarrow$ median of $\lceil n/5 \rceil$ medians —- how???



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
- Find $M \leftarrow$ median of $\lceil n/5 \rceil$ medians recursively
- Use median of medians M as pivot



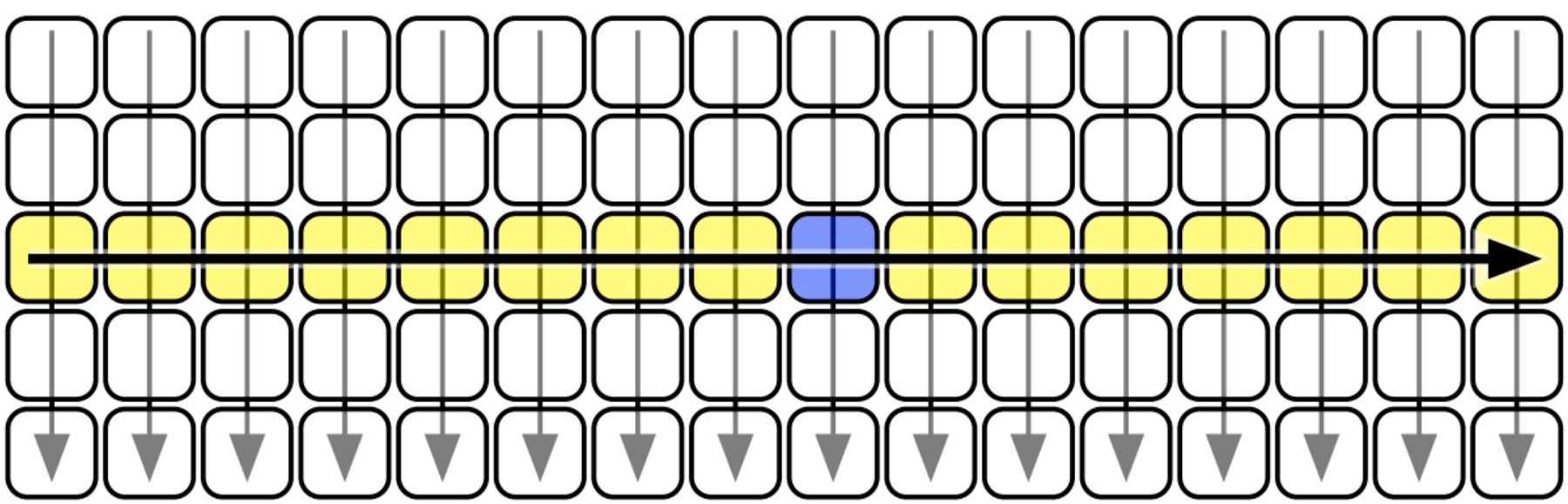
n = 54

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

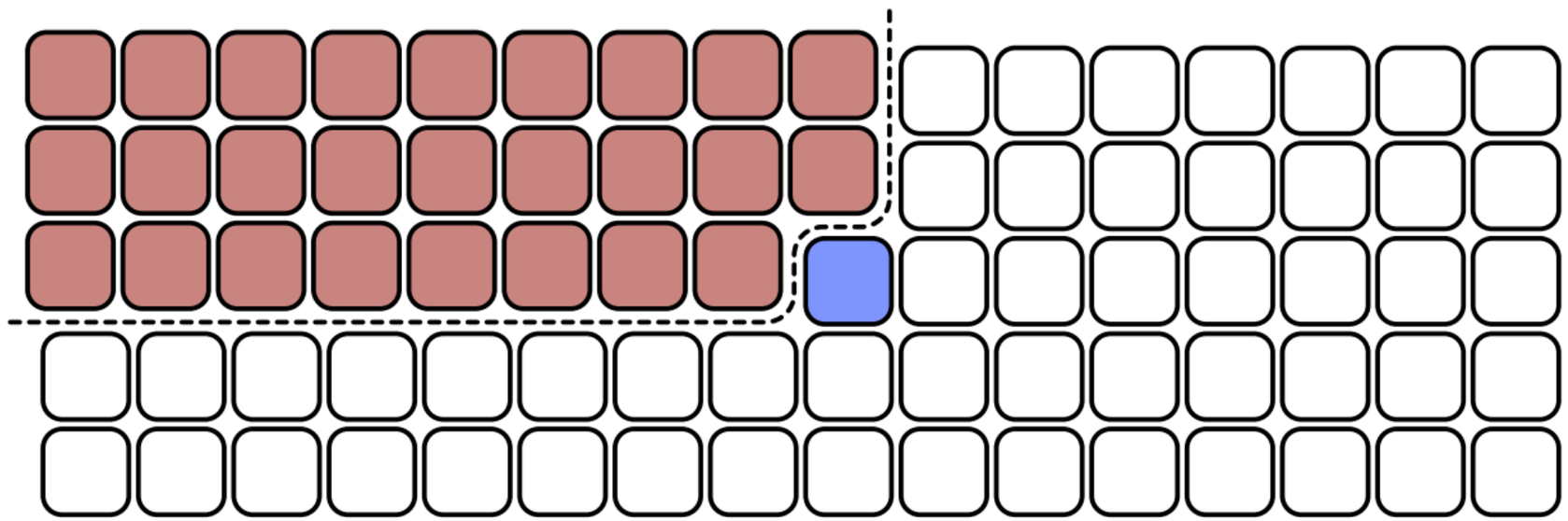
Visualizing MoM

- In the $5 \times n/5$ 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 sort anything!
- MoM is the element closest to center of grid



Visualizing MoM

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



How Good is the MoM?

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
  - Each median is greater than 2 elements in its group
  - Thus $M \geq 3g/2 \geq 3n/10$ elements
- Symmetrically, $M \leq 3n/10$ elements. ∎

How to Use the MoM?

- There are $3n/10$ elements smaller than the MoM
- By the same argument: $3n/10$ elements larger than the MoM
- So we can throw out $3n/10$ elements, adjust the value of k we are looking for, and recurse!
- Don't forget: we *also* recursed to find the MoM!

Recall: Selection

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

Linear time Selection

Select (A, k):

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

- Group elements into subarrays of size 5; find median in each
- Choose a pivot p as the median of these medians
- $r, A_{<p}, A_{>p} \leftarrow$ Partition$((A, p)$
- If $k == r$, return p
- Else:
  - If $k < r$: Select $(A_{<p}, k)$
  - Else: Select $(A_{>p}, k - r)$

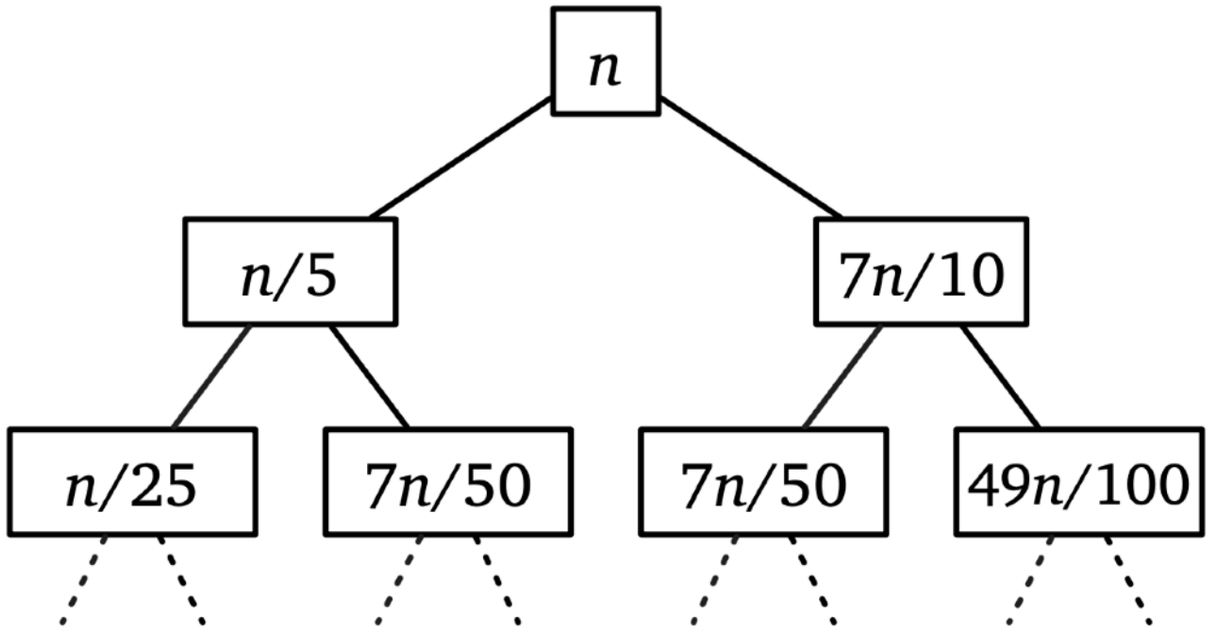
$T(n/5) + O(n)$

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

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

Selection Recurrence

- Okay, so we have a good pivot
- 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 will analyze this when we do randomized algorithms

Recall Challenge Recurrence

- Recall the challenge recurrence

$$T(n) = \sqrt{n}T(\sqrt{n}) + O(n)$$

- How much work at each level? $O(n)$
- Analyzing how quickly the problem size goes down
- $n \to n^{1/2} \to n^{1/4} \to \ldots \to n^{1/2^L}$
- What is L for this to be a small constant?
- $L = \log\log n$ (number of levels)
- $T(n) = \Theta(n \log\log n)$,

Floors and Ceilings

- Why doesn't floors and ceilings matter?
- Suppose $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$
- First, for upper bound, we can safely overestimate
  - $T(n) \leq 2T(\lceil n/2 \rceil) + n \leq 2T(n/2 + 1) + n$
- Second, we can define a function $S(n) = T(n + \alpha)$, so that $S(n)$ satisfies $S(n) \leq S(n/2) + O(n)$

$$
\begin{aligned}
S(n) = T(n + \alpha) &\leq 2T(n/2 + \alpha/2 + 1) + n + \alpha \\
&= 2T(n/2 + \alpha - \alpha/2 + 1) + n + \alpha \\
&= 2S(n/2 - \alpha/2 + 1) + n + \alpha \\
&\leq 2S(n/2) + n + 2, \text{ for } \alpha = 2
\end{aligned}
$$

Floors & Ceilings Don't Matter

- Why doesn't floors and ceilings matter?
- Suppose $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$
- First, for upper bound, we can safely overestimate
    - $T(n) \leq 2T(\lceil n/2 \rceil) + n \leq 2T(n/2 + 1) + n$
- Second, we can define a function $S(n) = T(n + \alpha)$, so that $S(n)$ satisfies $S(n) \leq S(n/2) + O(n)$
    - Setting $\alpha = 2$ works
- Finally, we know $S(n) = O(n \log n) = T(n + 2)$
- $T(n) = O((n-2)\log(n-2)) = O(n \log n)$

Can Assume Powers of 2

- Why doesn't taking powers of 2 matter?
- Running time $T(n)$ is monotonically increasing
- Suppose n is not a power of 2, let $n' = 2^{\ell}$ be such that $n \leq n' \leq 2n$; then
- We can upper bound our asymptotic using n' and lower bound using $n'/2$
- In particular, let $T(n) \leq T(n')$
- And $T(n) \geq T(n'/2)$
- That is, $T(n) = \Theta(T(n'))$

Dynamic Programming

“Those who cannot remember the past are condemned to repeat it.”

— Jorge Agustín Nicolás Ruiz de Santayana y Borrás,

Slow Recursion: Fibonnacci

- So far we have seen recursion examples that are smart and lead to efficient solutions
- This is not always the case
- For example,
  - Recursive Fibonacci

Definition. Recall Fibonacci numbers are defined by the following recurrence

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Slow Recursion: Fibonnacci

- This naive recurrence is horribly slow
- Let $T(n)$ denote the # of recursive calls
    - $T(n) = T(n-1) + T(n-2) + 1$
    - Can we lower bound this?

```
RecFibo(n):
    if n = 0
        return 0
    else if n = 1
        return 1
    else
        return RecFibo(n − 1) + RecFibo(n − 2)
```

Slow Recursion: Fibonnacci

- Correct answer:
- $T(n) \geq F_n$ for all $n \geq 1$
- $F_n \geq \phi^{n-2}$ where $\phi = \left(\frac{1+\sqrt{5}}{2}\right) \approx 1.6^{n-2}$ (exponential!)

```
RecFibo(n):
    if n = 0
        return 0
    else if n = 1
        return 1
    else
        return RecFibo(n − 1) + RecFibo(n − 2)
```

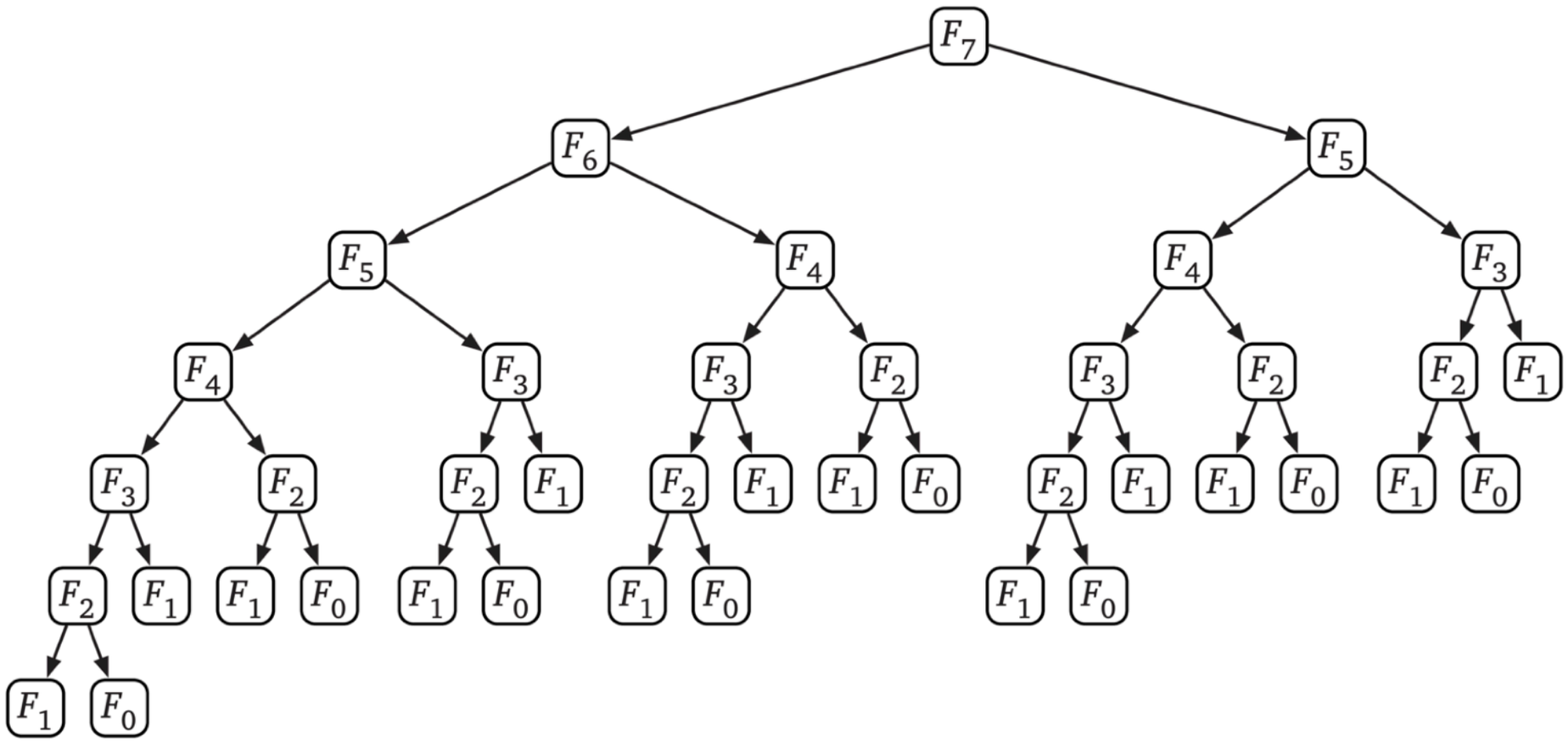
Slow Recursion: Fibonnacci

- Let's prove it's exponential; can we lower bound the running time using techniques we already have?
- $T(n) = T(n-1) + T(n-2) + \Theta(1)$
- $T(n) \geq 2T(n-2) + \Omega(1)$

- Level i has cost 2^i.
- There are $n/2$ levels
- $T(n) = \Omega(2^{n/2})$

Memo(r)ization

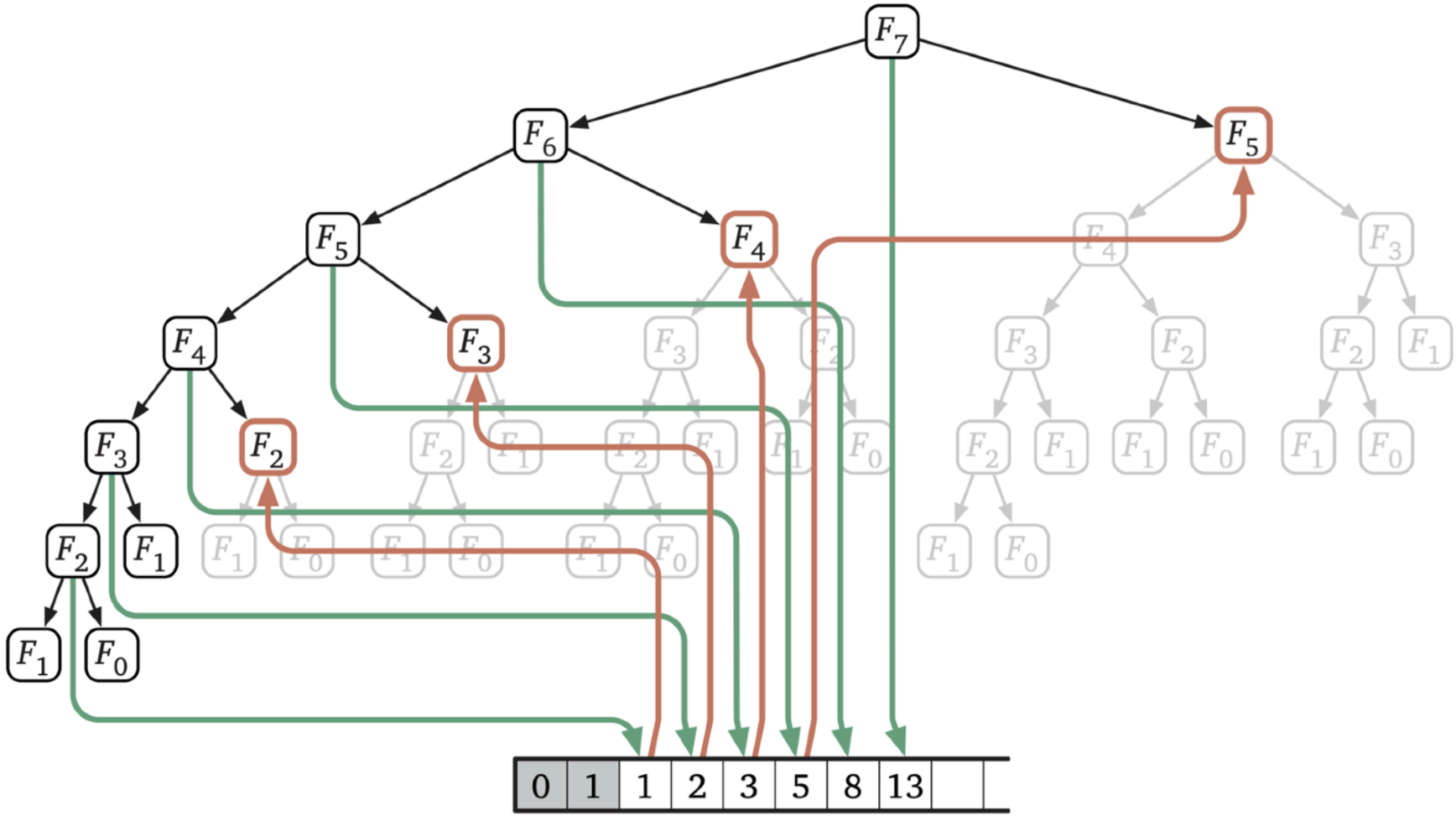
- Recursive Fibonacci algorithm is slow because it computes the same functions over and over
- Can speed it up considerably by writing down the results of our recursive calls, and looking them up when we need them later



Dynamic Programming: Smart Recursion

- Dynamic programming is all about smart recursion by using memoization
- Here it cuts down on all useless recursive calls

$$T[n] = T[n-1] + T[n-2] + 1$$



Memoization

- Memoization: technique to store expensive function calls so that they can be looked up later
- (Avoids calling the expensive function multiple times)

- A core concept of dynamic programming, but also used elsewhere

Memoizing Fibonacci

- Write each entry down in an array when you compute it
- How do we compute the nth Fibonacci number?
  - Fill in the first two Fibonacci numbers.
  - Use those to fill in the third, then fourth, etc.
  - Takes $O(1)$ to fill in a table entry
  - $O(n)$ overall

$$A = 1 \mid 1 \mid 2 \mid 3 \mid 5 \mid 8 \mid 13 \mid 21$$

Dynamic Programming

- Formalized by Richard Bellman in the 1950s

> We had a very interesting gentleman in Washington named Wilson. He was secretary of Defense, and he actually had a pathological fear and hatred of the word "*research*". I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term "*research*" in his presence. You can imagine how he felt, then, about the term "*mathematical*". . . . I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose?

- Chose the name **"dynamic programming"** to hide the mathematical nature of the work from military bosses

Recipe for a Dynamic Program

- **Formulate the right subproblem.** The subproblem must have an optimal substructure
- **Formulate the recurrence.** Identify how the result of the smaller subproblems can lead to that of a larger subproblem
- **State the base case(s)**. The subproblem thats so small we know the answer to it!
- **State the final answer.** (In terms of the subproblem)
- **Choose a memoization data structure.** Where are you going to store already computed results? (Usually a table)
- **Identify evaluation order.** Identify the dependencies: which subproblems depend on which ones? Using these dependencies, identify an evaluation order
- **Analyze space and running time.** As always!

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)