Selection and Matrix Multiply

Check in and Reminders

- Assignment 4 is out and due next Wed
- I created a handout with examples of using the recursion tree method to solve recurrences: link on website/GLOW
- No class on Monday: reading period
- We will have regular office/TA hours during reading period
- Where we are in the course:
    - Wrapping up divide and conquer
    - Next topic: divide and conquer

Selection

Problem Statement

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

Idea: Break the problem into smaller subproblems by partitioning around a pivot element. Find the kth smallest element recursively by searching in the correct (left or right) subarray.

Our goal. $O(n)$ time algorithm

Since we are doing $O(n)$ work partitioning, we want a recurrence where the cost is dominated at the root (exponentially decaying series)

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

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

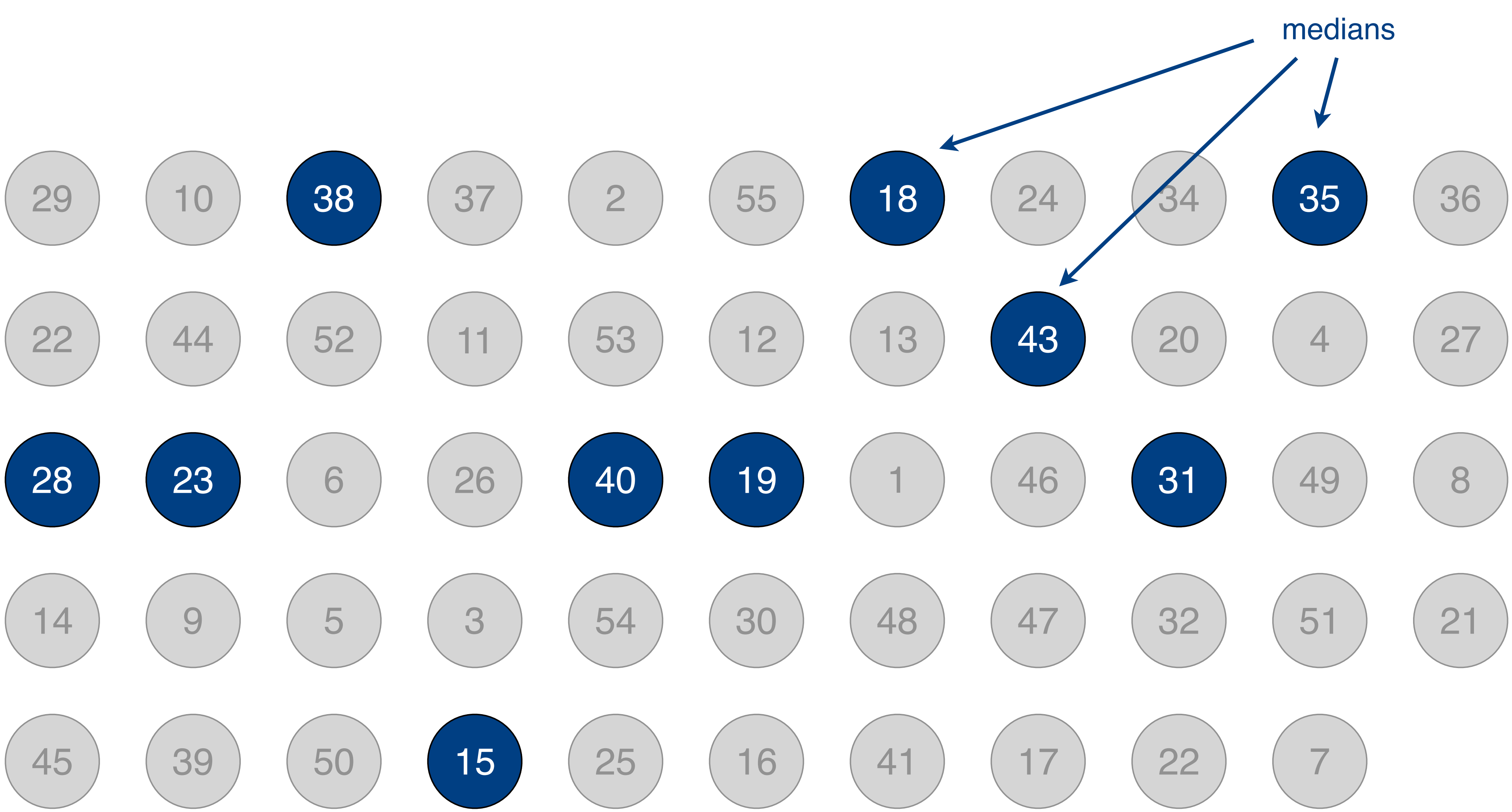
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group

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

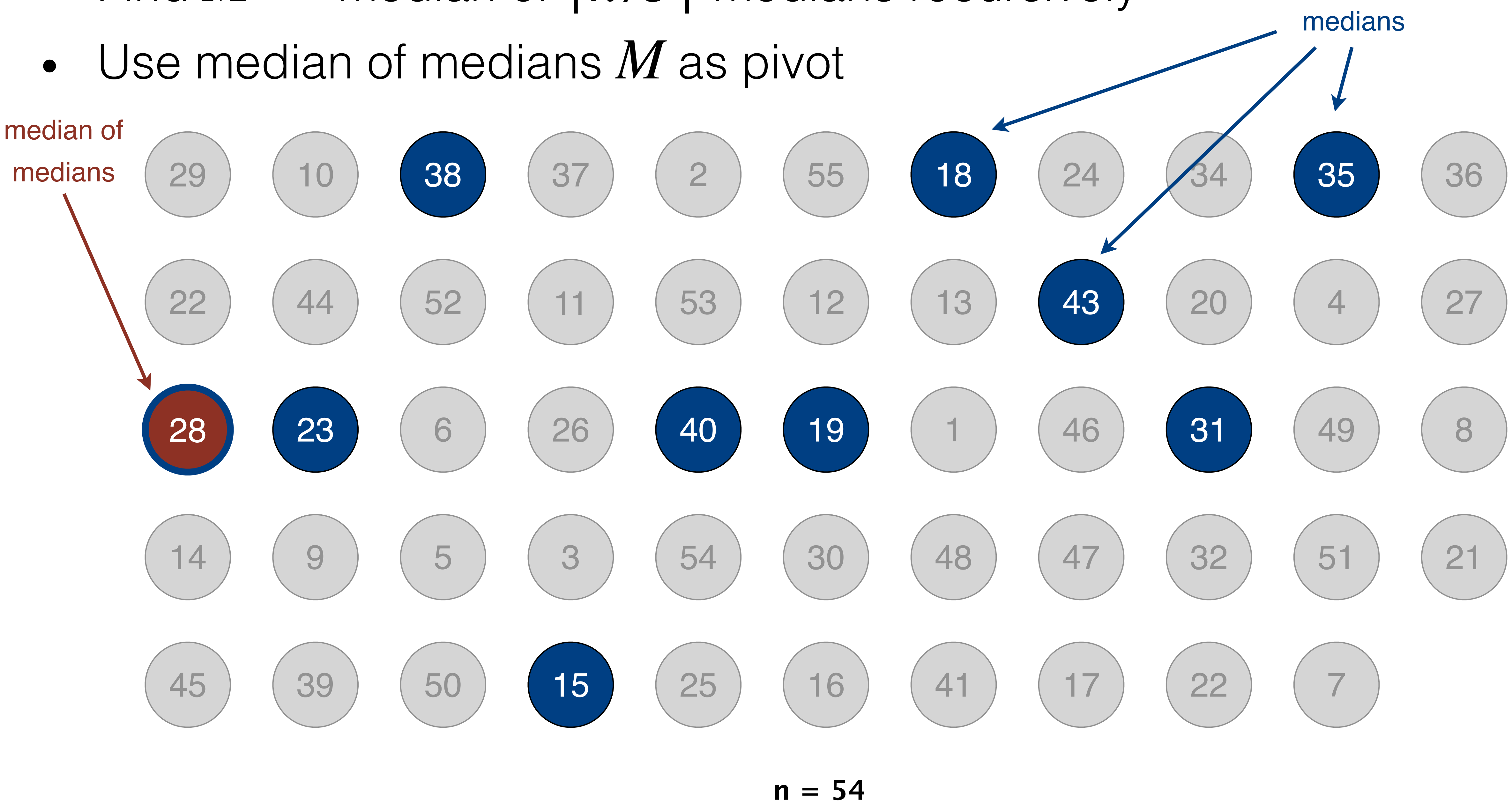
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group



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

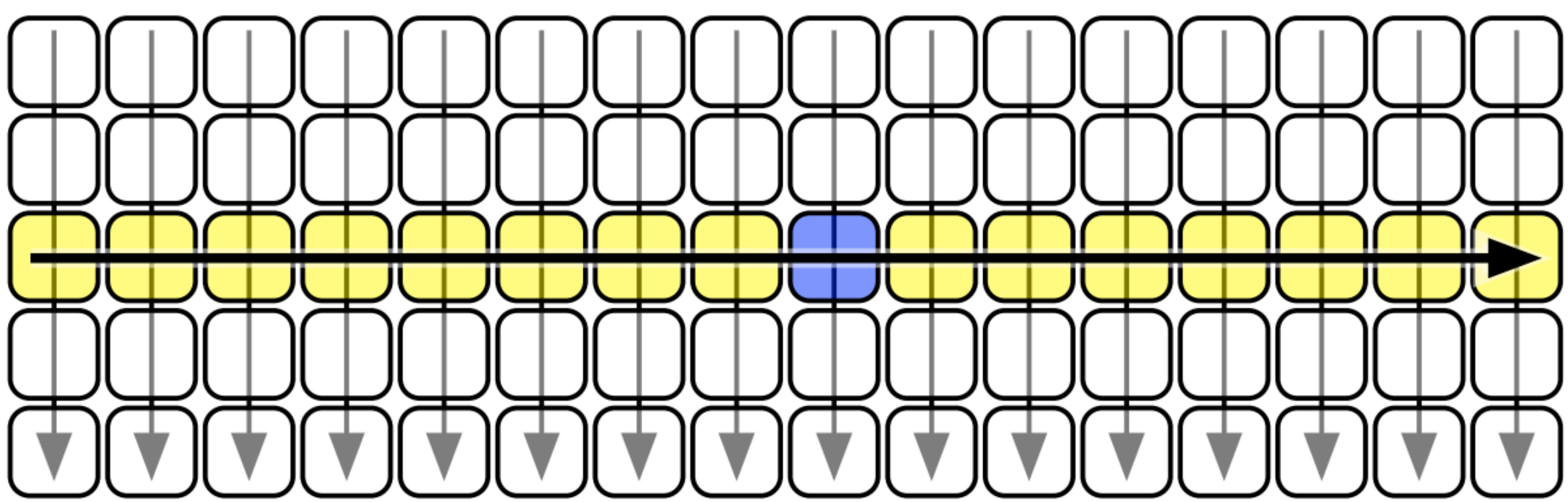


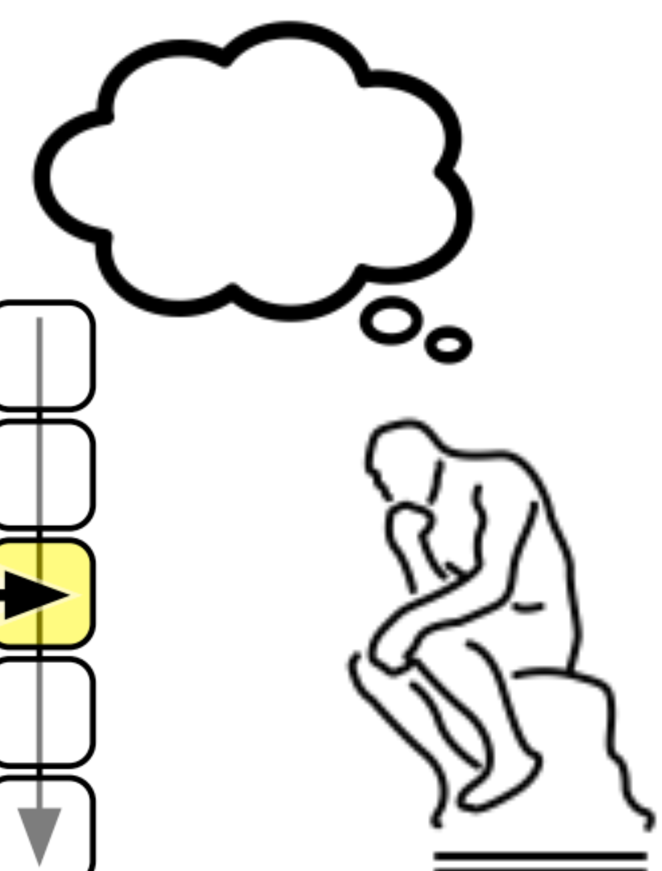
What did we gain?

- 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

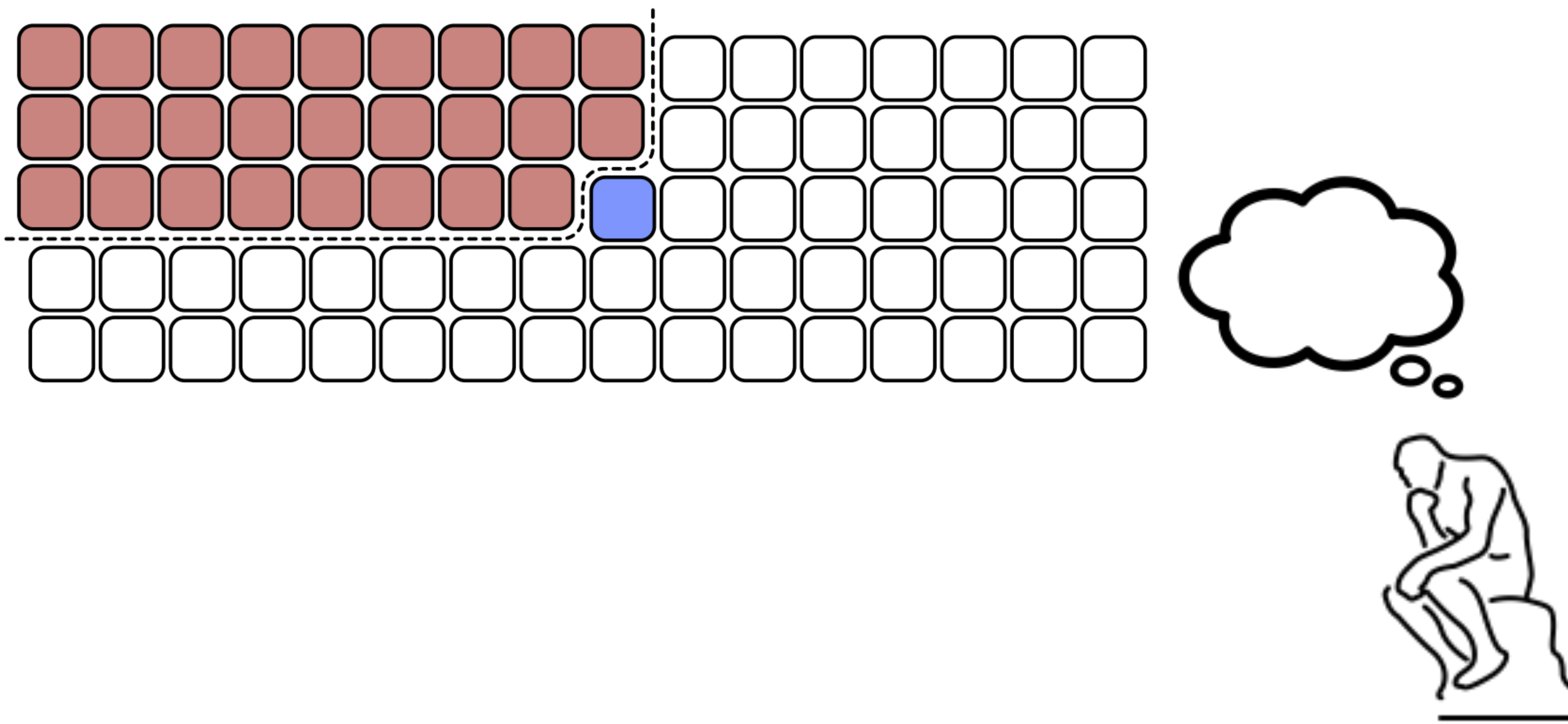
- In the 5 x n/5 grid, each column represents five consecutive elements
- **Imagine** each column is sorted top down
- **Imagine** the columns as a whole are sorted left-right
    - We don’t actually do this!
- MoM is the element closest to center of grid





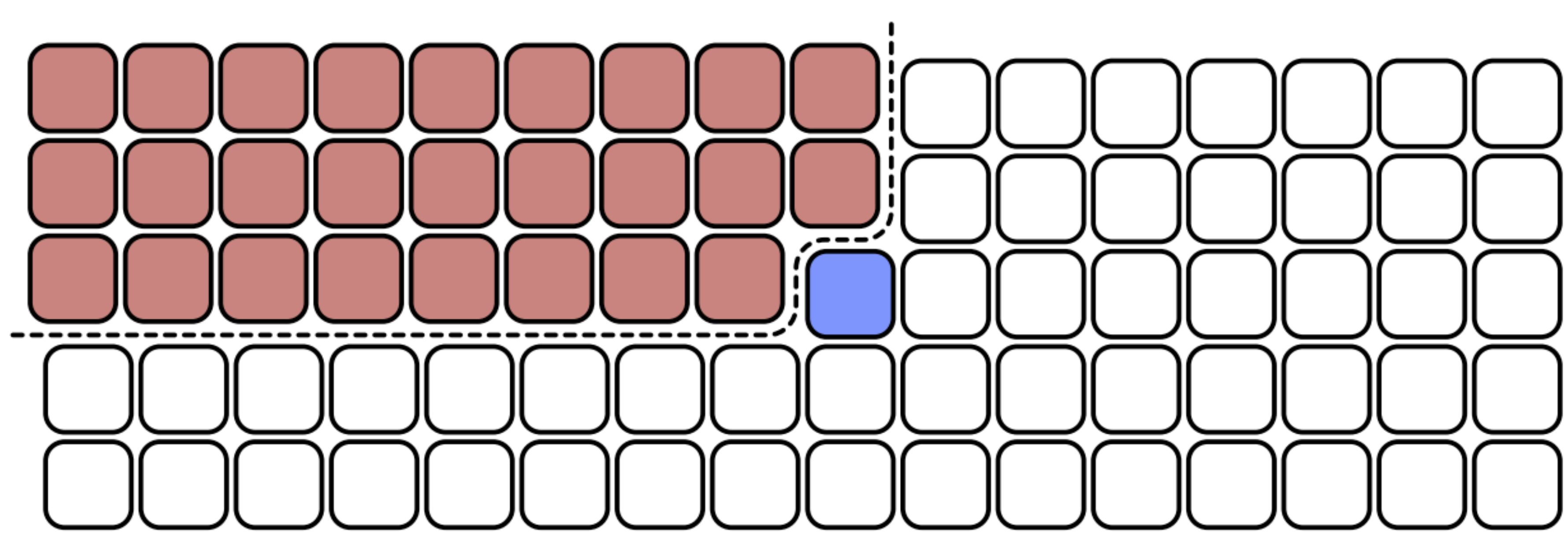
Visualizing MoM

- 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 smaller 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
  - Each median is greater than 2 elements in its group
  - Thus $M \geq 3g/2 = 3n/10$ 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!

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$((A, p)$

- If $k == r$, return p

- Else:

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

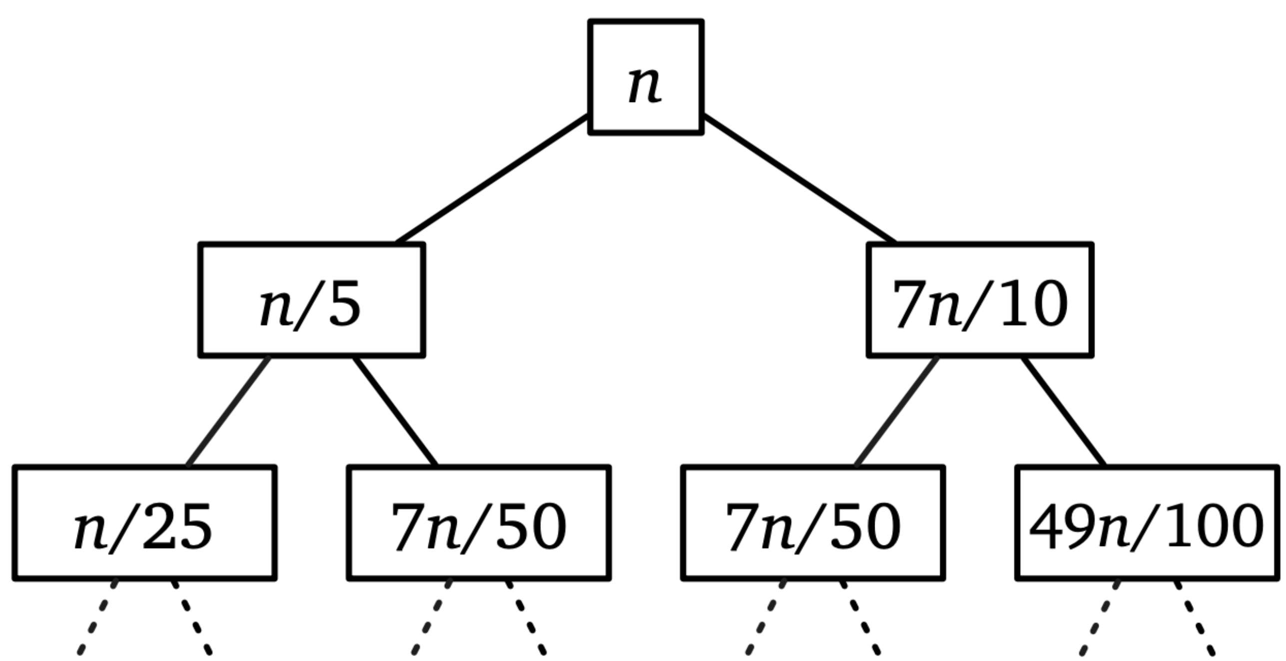
  - Else: Select $(A_{>p}, k - r)$

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

Matrix Multiplication

Matrix Multiplication

Problem. Given two n-by-n matrices A and B, compute matrix $C = A \cdot B$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Standard multiplication computes each c_{ij} as:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Complexity. $\Theta(n^3)$ operations (scalar multiplications)

Block Matrix Multiplication

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

C_{11} $\quad$ A_{11} $\quad$ A_{12} $\quad$ B_{11}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

B_{21}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Block Matrix Multiplication

To multiply two n-by-n matrices A and B:

- **Divide**: partition A and B into $\frac{n}{2}$ by $\frac{n}{2}$ matrices
- **Conquer**: multiply 8 pairs of $\frac{n}{2}$ by $\frac{n}{2}$ matrices recursively
- **Combine**: Add products using 4 matrix additions

n-by-n matrices

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

½n-by-½n matrices

8 matrix multiplications (of ½n-by-½n matrices)

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions (of ½n-by-½n matrices)

Block Matrix Multiplication

Running time recurrence.

- $T(n) = 8T(n/2) + \Theta(n^2)$
- How do we solve it with the recursion-tree method?
  - $T(n) = O(n^3)$

- Nice idea but it didn't improve the run time, oh well!
- Divide and conquer version is still more **cache-efficient**

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

½n-by-½n matrices

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

4 matrix additions (of ½n-by-½n matrices)

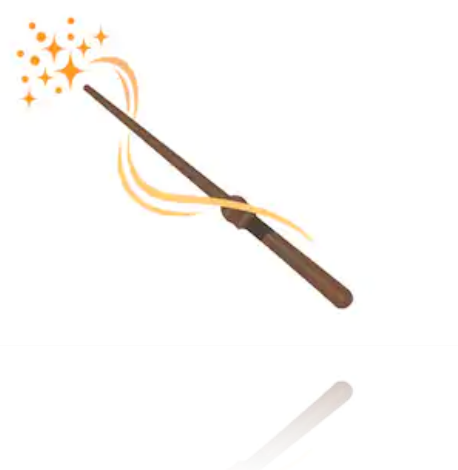
Block MM: Strassen's Trick

Key idea. Can multiply two 2-by-2 matrices via 7 scalar multiplications (plus 11 additions and 7 subtractions).

scalars

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$



$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 scalar multiplications

Pf. $$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \end{aligned}$$

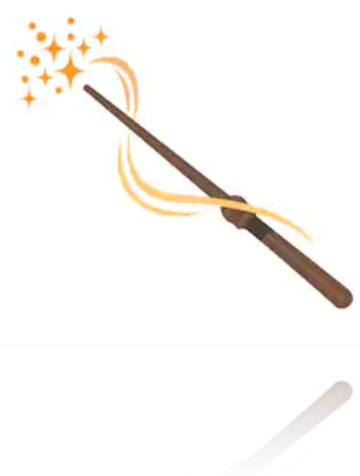
Block MM: Strassen's Trick

Key idea. Can multiply two n-by-n matrices via 7 $n/2$-by-$n/2$ matrix multiplications (plus 11 additions and 7 subtractions).

½n-by-½n matrices

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$



$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

7 scalar multiplications

Pf.
$$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \end{aligned}$$

Strassen's MM Algorithm

assume n is a power of 2

$\text{STRASSEN}(n, A, B)$

IF $(n = 1)$ RETURN $A \times B$.

Partition A and B into ½n-by-½n blocks.

$P_1 \leftarrow \text{STRASSEN}(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow \text{STRASSEN}(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow \text{STRASSEN}(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow \text{STRASSEN}(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow \text{STRASSEN}(n / 2, (A_{11} + A_{22}), (B_{11} + B_{22}))$.

$P_6 \leftarrow \text{STRASSEN}(n / 2, (A_{12} - A_{22}), (B_{21} + B_{22}))$.

$P_7 \leftarrow \text{STRASSEN}(n / 2, (A_{11} - A_{21}), (B_{11} + B_{12}))$.

$7\,T(n / 2) + \Theta(n^2)$

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

$\Theta(n^2)$

RETURN C.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Strassen's MM Algorithm Analysis

- We get the following recurrence
    - $T(n) = 7T(n/2) + \Theta(n^2)$
- What does the running time recurrence solve to?
    - We have a increasing geometric series
    - Thus, the cost is dominated by the leaves
    - $T(n) = \Theta(r^L) = \Theta(7^{\log_2 n}) = \Theta(n^{\log_2 r}) \approx \Theta(n^{2.81})$
    - We have a much faster algorithm!

History of Matrix Multiplication

year	algorithm	arithmetic operations
1858	**“grade school”**	$O(n^3)$
1969	**Strassen**	$O(n^{2.808})$
1978	**Pan**	$O(n^{2.796})$
1979	**Bini**	$O(n^{2.780})$
1981	**Schönhage**	$O(n^{2.522})$
1982	**Romani**	$O(n^{2.517})$
1982	**Coppersmith–Winograd**	$O(n^{2.496})$
1986	**Strassen**	$O(n^{2.479})$
1989	**Coppersmith–Winograd**	$O(n^{2.3755})$
2010	**Strother**	$O(n^{2.3737})$
2011	**Williams**	$O(n^{2.372873})$
2014	**Le Gall**	$O(n^{2.372864})$
		$O(n^{2+\varepsilon})$

galactic algorithms

“Galactic algorithm: runs faster than any other algorithm for problems that are sufficiently large, but "sufficiently large" is so big that the algorithm is never used in practice.”

Tons of Applications

- Lots of problem reduce to matrix multiplication complexity

linear algebra problem	expression	arithmetic complexity
matrix multiplication	$A \times B$	$MM(n)$
matrix squaring	A^2	$\Theta(MM(n))$
matrix inversion	A^{-1}	$\Theta(MM(n))$
determinant	$\|A\|$	$\Theta(MM(n))$
rank	$rank(A)$	$\Theta(MM(n))$
system of linear equations	$Ax = b$	$\Theta(MM(n))$
LU decomposition	$A = L\,U$	$\Theta(MM(n))$
least squares	$\min \|\|Ax - b\|\|_2$	$\Theta(MM(n))$

numerical linear algebra problems with the same arithmetic complexity MM(n) as matrix multiplication

Boring Slides Alert:

Including for Completeness

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, $T(n) = \Theta(T(n'))$

Guess & Verify Recurrences

- **Method 3.** Requires some practice and creativity
- Verification by induction may run into issues
  - Example, $T(n) = 2T(n/2) + 1$
  - Guess?
    - $T(n) \le cn$
  - Check $T(n) \le cn + 1 \not\le cn$ for any $c > 0$
  - Is the guess wrong? Not asymptotically, can fix it up by adding lower-order terms
  - New guess $T(n) \le cn - d$ (why minus?)
    - $T(n) \le cn - 2d + 1 \quad \le cn - d$ for any $d \ge 1$
  - c must be chosen large enough to satisfy boundary conditions

Challenge Problem

- Suppose we run quick sort where the pivot is always recursively of rank $\sqrt{n}$
- Then the recurrence for quick sort becomes
  - $T(n) = T(n - \sqrt{n}) + T(\sqrt{n} + n$
- Analyze the running time of this algorithm

End of Divide & Conquer

Dynamic Programming

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

— Jorge Agustín Nicolás Ruiz de Santayana y Borrás,

Stupid 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

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

Stupid Recursion: Fibonnacci

- This naive recurrence is horribly slow
- Let $T(n)$ denote the # of recursive calls
  - $T(n) = T(n-1) + T(n-2) + 1$

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

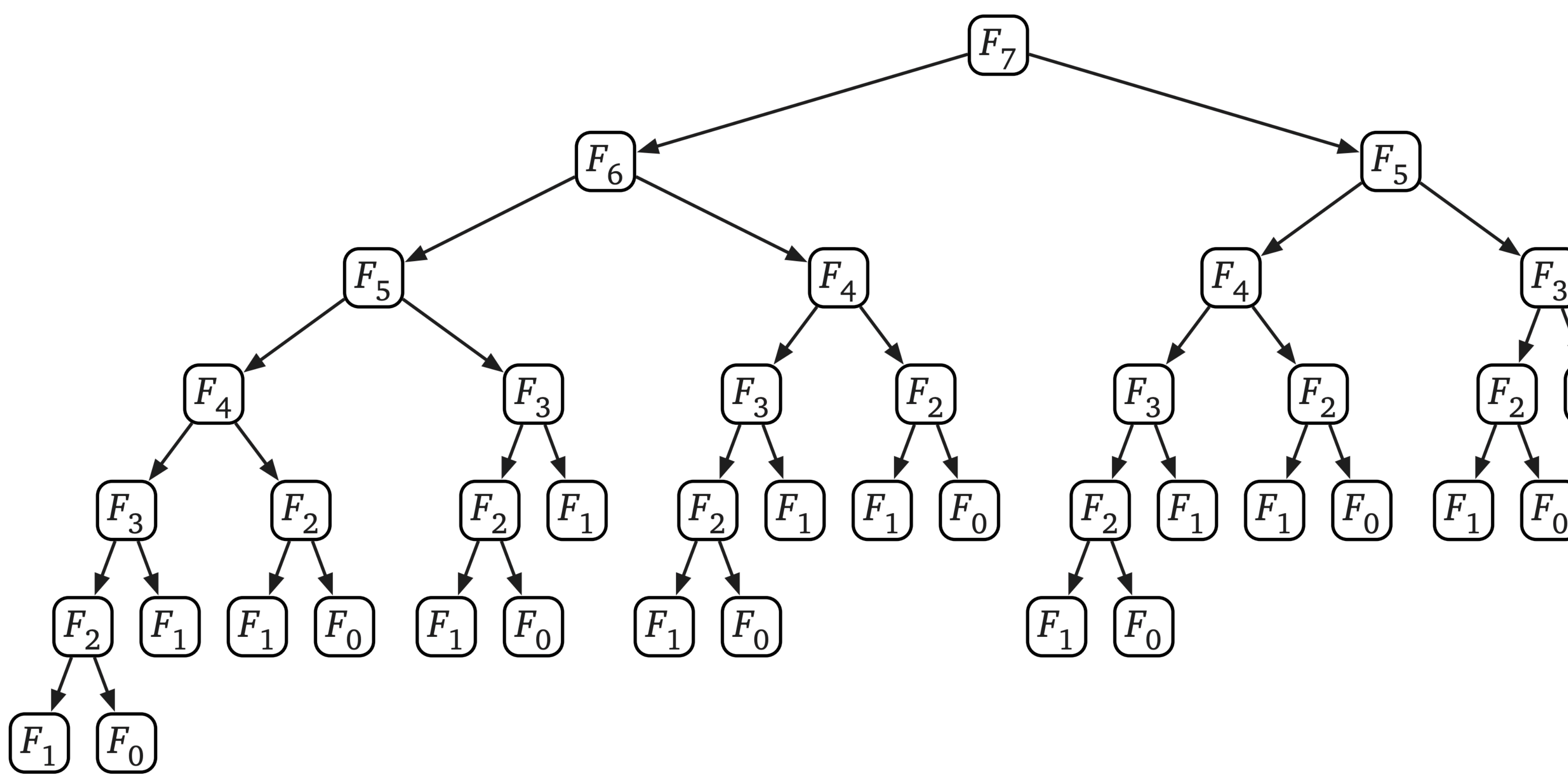
Stupid Recursion: Fibonnacci

- $T(n) \geq F_n$ 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)
```

Memo(r)ization

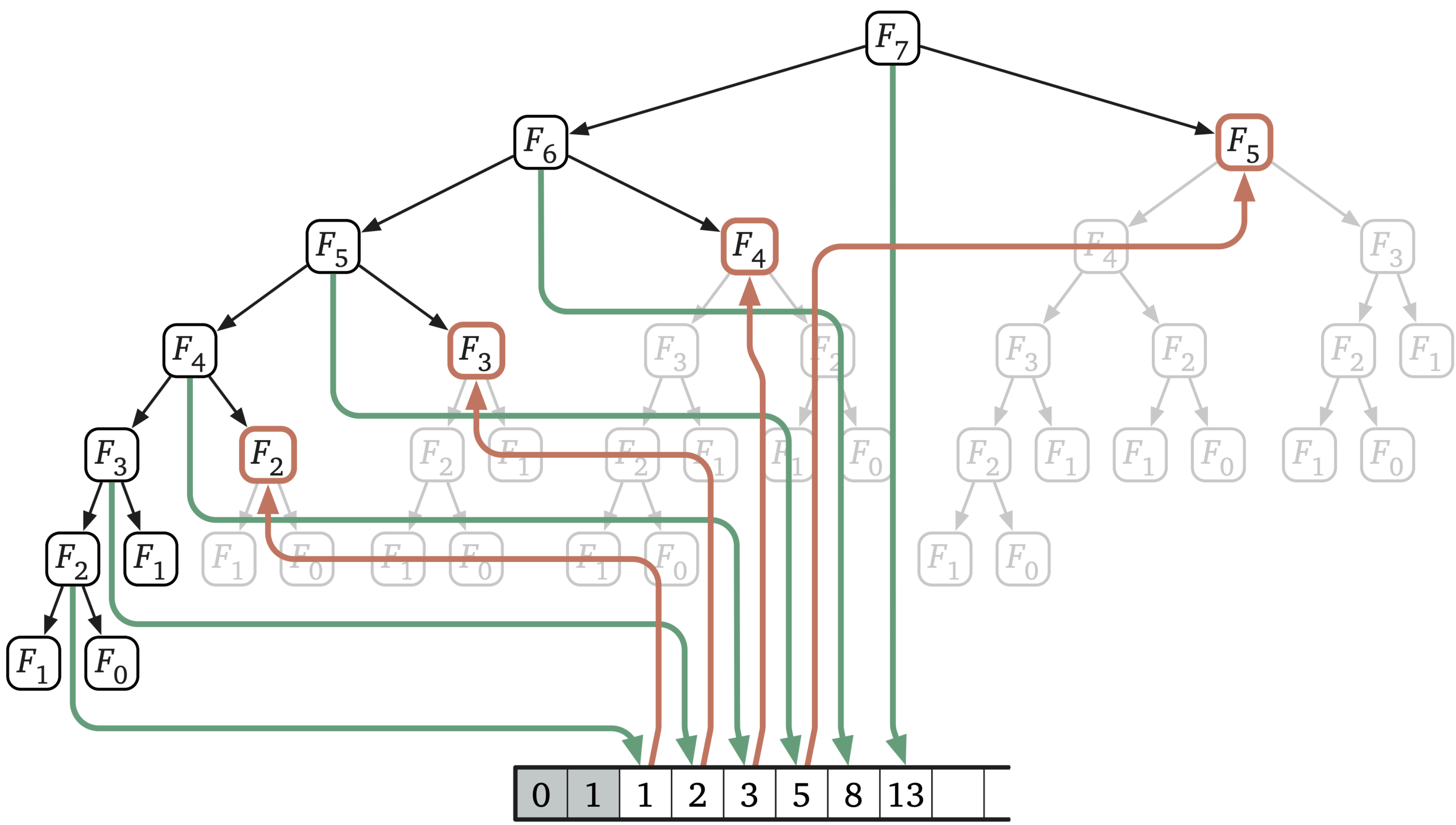
- Recursive Fibonacci algorithm is slow because it computes the same functions over and over
- Can speed it up considerably by writing down the results of our recursive calls, and looking them up when we need them later



Dynamic Programming: Smart Recursion

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

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



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

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)
  - CLRS Algorithms book