Recursion Tree Method and Selection

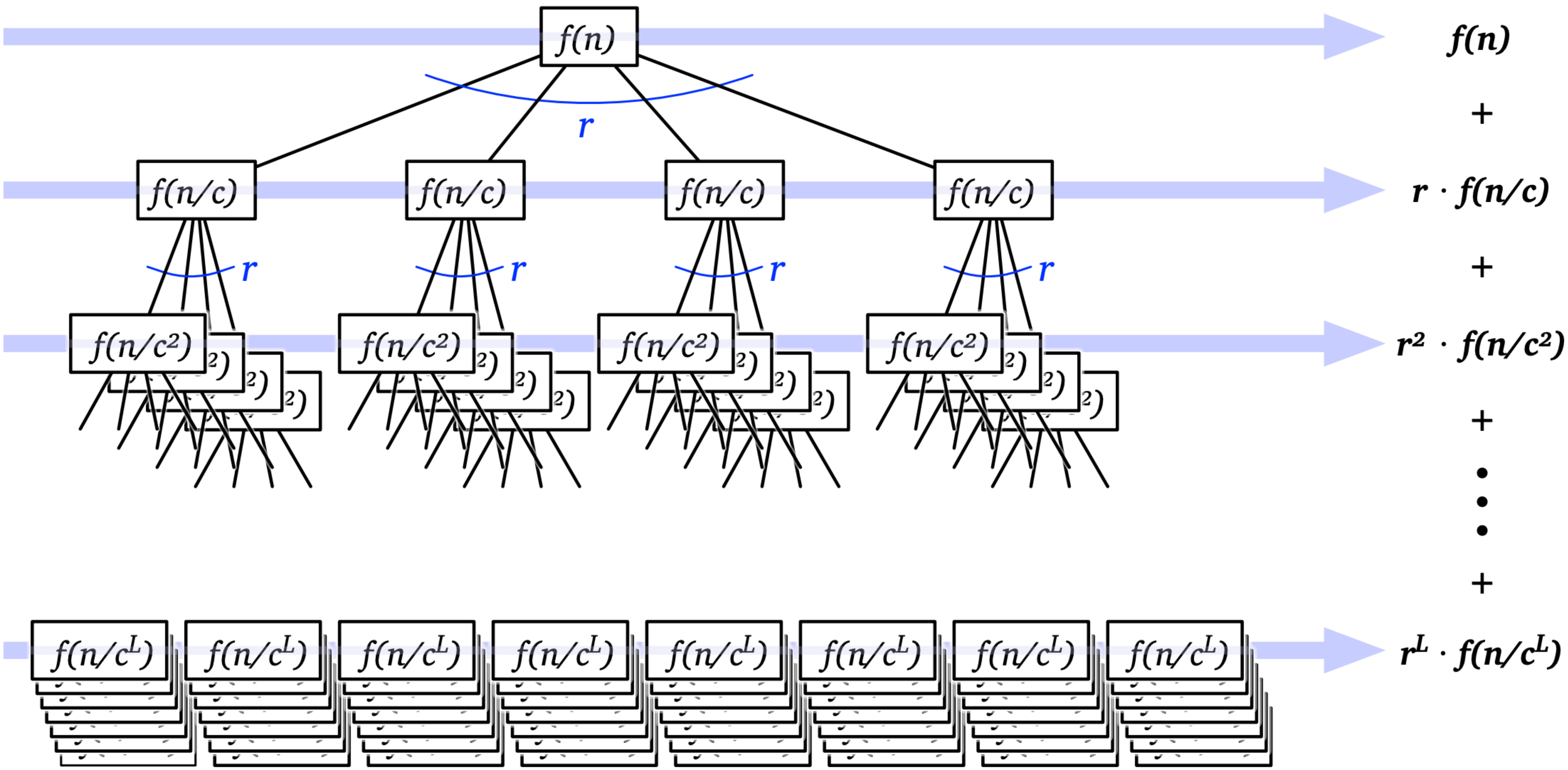
Check in and Reminders

- Assignment 3 was due last night
- If you are taking a late day, I have **office hours today** from **12.30-2 pm @ CS common room**
- I also have **office hours Thursday 1-2 pm @ my office**
- **Friday office hours on GLOW**
- Come see me if you have any questions, or just to chat!
  - I always want to know how my students are doing
- Assignment 4 is out (Divide and Conquer)
  - Use **strong induction** to prove correctness of divide and conquer algorithms
  - Jeff Erickson's book has really good coverage of recursion trees and analyzing recurrences

General Recursion Trees

- Consider a divide and conquer algorithm that
  - spends $O(f(n))$ time on non-recursive work and makes r recursive calls, each on a problem of size n/c
- Up to constant factors (which we hide in $O()$), the running time of the algorithm is given by what **recurrence**?
  - $T(n) = rT(n/c) + f(n)$
- Because we care about asymptotic bounds, we can assume base case is a small constant, say $T(n) = 1$

General Recursion Trees



A recursion tree for the recurrence $T(n) = rT(n/c) + f(n)$

- For each i, the ith level of tree has exactly r^i nodes
- Each node at level i, has cost $f(n/c^i)$

General Recursion Trees

- Running time $T(n)$ of a recursive algorithm is the sum of all the values (sum of work at all nodes at each level) in the recursion tree
- For each i, the ith level of tree has exactly r^i nodes
- Each node at level i, has cost $f(n/c^i)$
- Thus, $T(n) = \sum_{i=0}^{L} r^i \cdot f(n/c^i)$
- Here $L = \log_c n$ is the depth of the tree
- Number of leaves in the tree: $r^L = n^{\log_c r}$
- Cost at leaves: $O(n^{\log_c r} f(1))$

General Recursion Trees

- Running time $T(n)$ of a recursive algorithm is the sum of all the values (sum of work at all nodes at each level) in the recursion tree
- For each i, the ith level of tree has exactly r^i nodes
- Each node at level i, has cost $f(n/c^i)$
- Thus, $T(n) = \sum_{i=0}^{L} r^i \cdot f(n/c^i)$
- Here $L = \log_c n$ is the depth of the tree
- Number of leaves in the tree: $r^L = n^{\log_c r}$ (why?)
- Cost at leaves: $O(n^{\log_c r} f(1))$

$$r^L = r^{\log_c n} = (2^{\log_2 r})^{\log_c n} = (2^{\log_c n})^{\log_2 r} = (2^{\log_2 n})^{\frac{\log_2 r}{\log_2 c}} = n^{\log_c r}$$

Easy Cases to Evaluate

$$T(n) = \sum_{i=0}^{L} r^i \cdot f(n/c^i)$$

- **Decreasing series.** If the series decays exponentially (every term is a constant factor smaller than previous): cost at root dominates:
$$T(n) = O(f(n))$$

- **Equal.** If all terms in the series are equal:
$$T(n) = O(f(n) \cdot L) = O(f(n) \log n)$$

- **Increasing series.** If the series grows exponentially (every terms is constant factor larger than previous): the cost at leaves dominates:
$$T(n) = O(n^{\log_c r})$$

In Class Exercises

- Take a few minutes to draw recursions trees for each of the following recurrences
- Then break into small groups (~size 3) and discuss which of the three cases each of them fall into

- $T(n) = 2(Tn/2) + n^2$
- $T(n) = 3T(n/2) + n$

[Akra–Bazzi '98]: Master Theorem

(Master Theorem.) Let $a \geq 1,\ b > 1$ are constants and $f(n) \geq 0$. Let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = r\ (n/c) + f(n)$, where we interpret n/c as $\lfloor n/c \rfloor$ or $\lceil n/c \rceil$.

Then $T(n)$ can be bounded asymptotically as follows.

- If $f(n) = n^{\log_c r - \epsilon}$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_c r})$
- If $f(n) = \Theta(n^{\log_c r})$, then $T(n) = \Theta(n^{\log_c r} \log n)$
- If $f(n) = \Omega(n^{\log_c r + \epsilon})$, for some constant $\epsilon > 0$, and if $rf(n/b) \leq c_0 f(n)$ for some constant $c_0 < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Theorem Broken Down

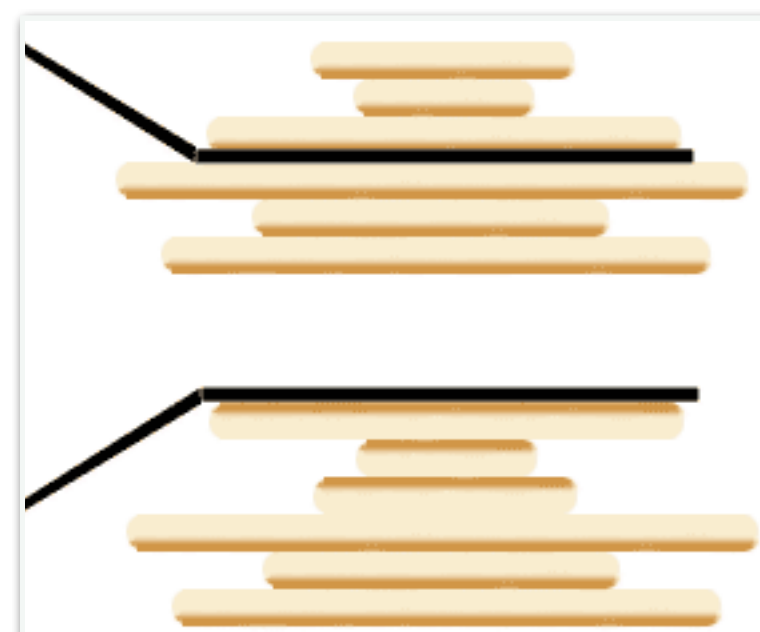
Intuitively Master theorem is comparing work at root $f(n)$versus work at leaves $\Theta(n^{\log_c r})$ of the recursion tree:

Three cases as before:

- Work at leaves dominates, then $T(n) = \Theta(n^{\log_c r})$
- Work is same at root and leaves $f(n)$ (and thus throughout the tree), then $T(n) = \Theta(f(n) \log n)$ (work times number of levels)
- Work at root dominates, then $T(n) = \Theta(f(n))$

Fun: Pancake Sorting

- You are given a stack of n pancakes of different sizes
- **Goal.** sort the pancakes so that smaller pancakes are on top of larger pancakes
- Only operation you can perform is a **flip**
    - Insert a spatula under the top k pancakes and flip them all over
- Describe an algorithm to sort an arbitrary stack of n pancakes using $O(n)$ flips.
- Best known lower and upper bounds on # of flips: $1.07n$ and $1.64n$
- Famous software developer wrote a paper on this as an undergrad…



Selection: Problem Statement

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

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

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

- **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

- If we can find a pivot p from $A[1,\ldots n]$
    - Such that 3/10 of the array is less than p and 6/10 fo the array is greater than p
- Then we have found the 4th smallest element!
- We can return p!
- 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 $(A_{>p}, k - r)$

How to Choose a Good Pivot?

- Recurrence for pivot of rank r
  - $T(n) = \max\{T(r), T(n-r)\} + O(n)$
- We don't know r, so assuming the worst:
  - $T(n) = \max_{1 \le r \le n} \max\{T(r), T(n-r)\} + O(n)$
  - Simplify: use ℓ = length of recursive subproblem
  - $T(n) = \max_{1 \le \ell \le n-1} T(\ell) + O(n)$
  - For what ℓ do we get a linear solution?

How to Choose a Good Pivot?

$$T(n) = \max_{1 \leq n-1} T(\ell) + O(n)$$

- If we reduce subproblem size by constant factor each time, we get a linear solution
- That is, $\ell \leq \alpha n$ for some constant $\alpha < 1$
- $T(n) \leq T(\alpha n) + O(n)$ for some constant $\alpha < 1$
- Expands to a decreasing geometric series
- Largest term at root dominates: $T(n) = O(n)$

Take away.

- We want a pivot that partitions such that where larger subproblem is constant factor smaller than n
- If we can find an **“approximate median”** in linear time, we can find the median in linear time as well!

Finding an Approximate Median

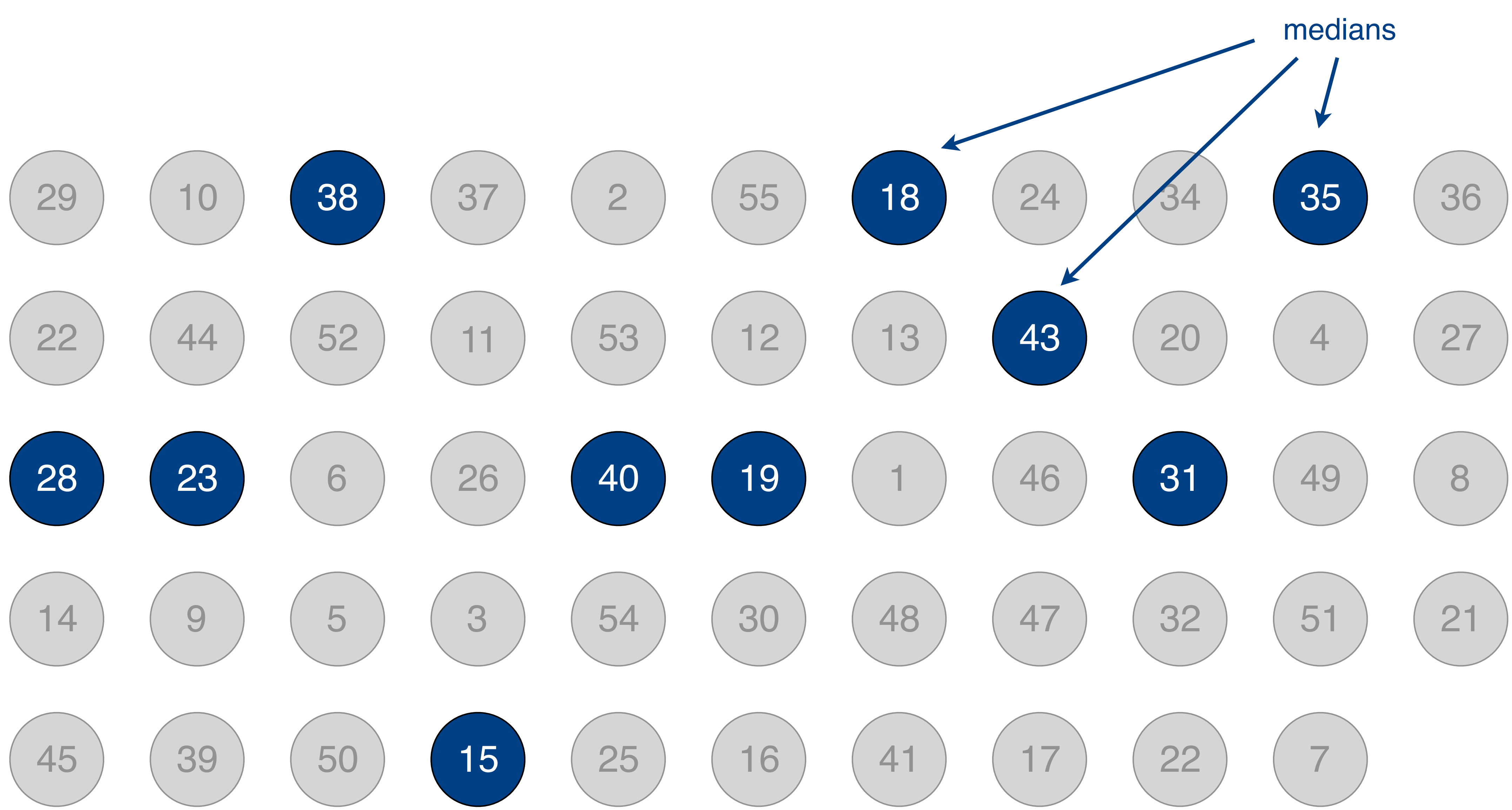
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group

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

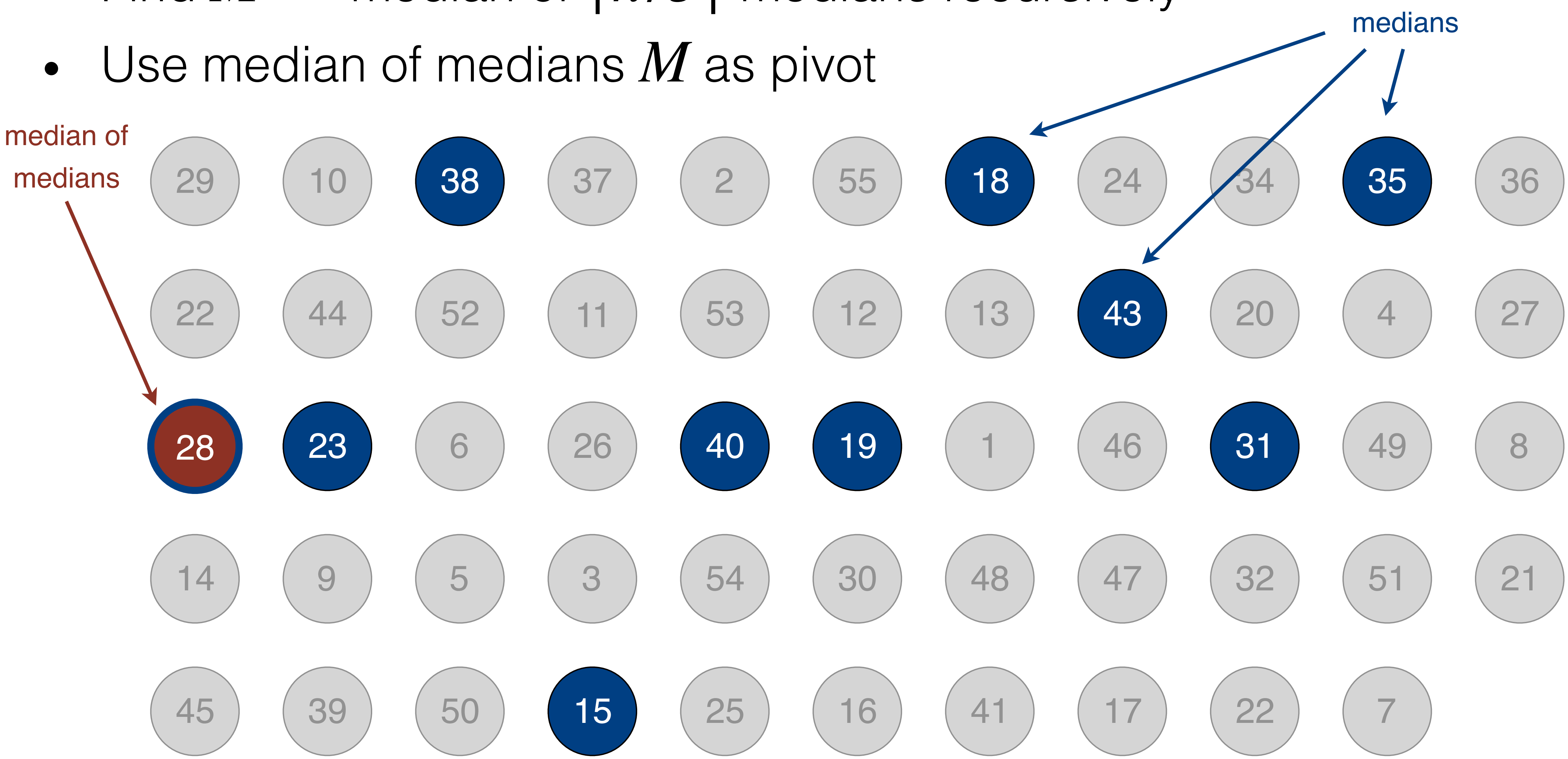
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group



n = 54

Finding an Approximate Median

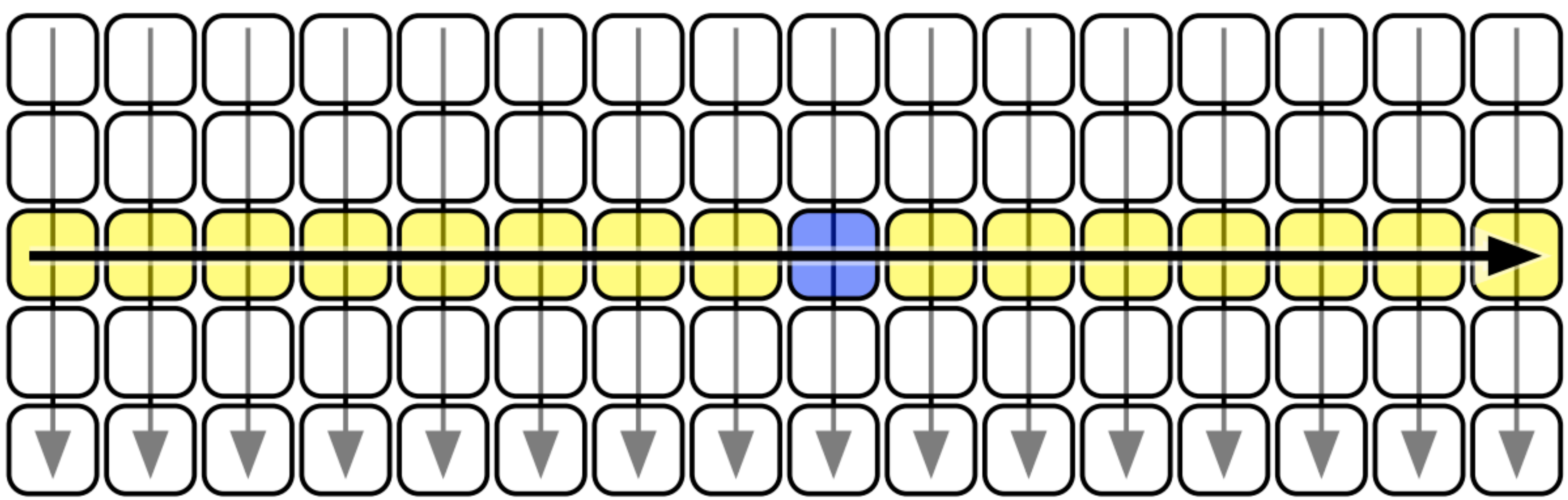
- Divide the array of size n into $\lceil n/5 \rceil$ groups of 5 elements (ignore leftovers)
- Find median of each group
- Find $M \leftarrow$ median of $\lceil n/5 \rceil$ medians recursively
- Use median of medians M as pivot



n = 54

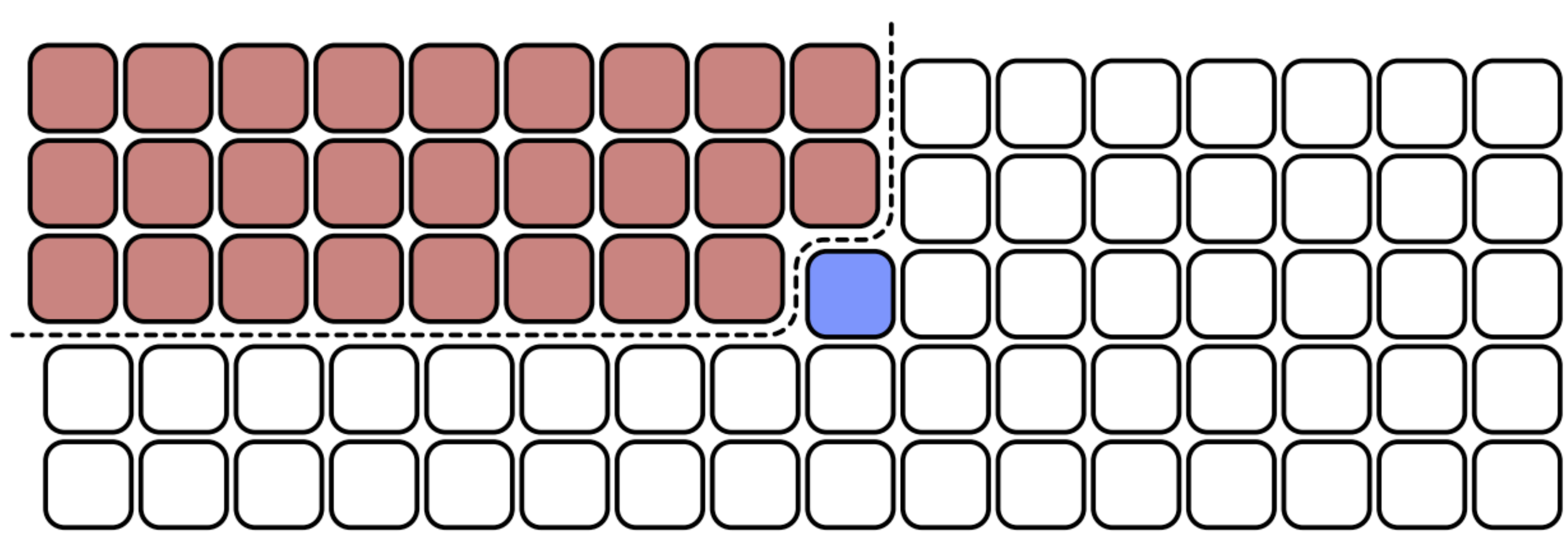
Visualizing MoM

- 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 $3n/10$) 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. ∎

Analysis: Running Time

- **Question.** How to compute median of median recursively?
- 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', $\lceil n/5 \rceil$)

Not recursive; $O(n)$

Not recursive; $O(n)$

Analysis: Running Time

- **Recurrence just for MoM:**
  - $T(n) = T(n/5) + O(n)$
- 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$(A', \lceil n/5 \rceil)$

Analysis: Overall

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

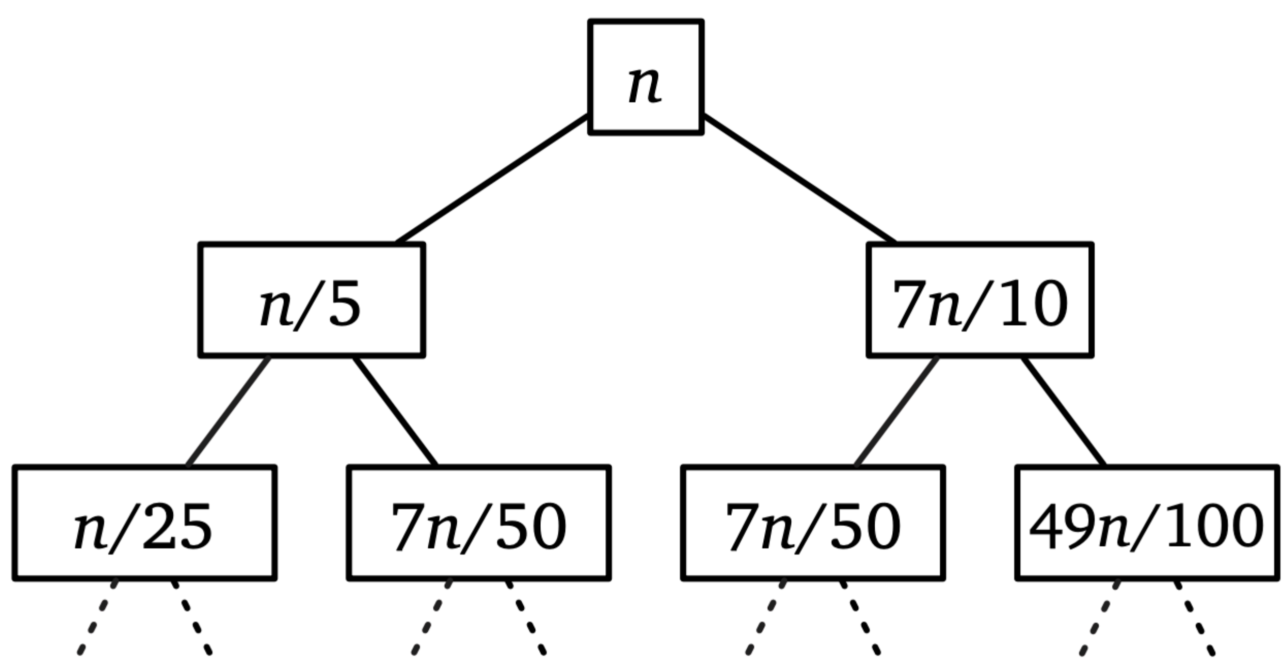
$T(n/5) + O(n)$

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

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?
    - $T(n) \leq cn$
  - Check $T(n) \leq cn + 1 \nleq cn$ for any $c > 0$
  - Is the guess wrong? Not asymptotically, can fix it up by adding lower-order terms
  - New guess $T(n) \leq cn - d$ (why minus?)
    - $T(n) \leq cn - 2d + 1 \quad \leq cn - d$ for any $d \geq 1$
  - c must be chosen large enough to satisfy boundary conditions

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
  - $T(n) \le 2T(\lceil n/2 \rceil) + n \le 2T(n/2 + 1) + n$
- Second, we can define a function $S(n) = T(n + \alpha)$, so that $S(n)$ satisfies $S(n) \le S(n/2) + O(n)$
  - 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 $n \le n' \le 2n$; then
- We can upper bound our asymptotic using n' and lower bound using $n'/2$
- In particular, let $T(n) \le T(n')$
- And $T(n) \ge T(n'/2)$
- That is, $T(n) = \Theta(T(n'))$

Recall Challenge Recurrence

- Recall the challenge recurrence

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

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

Extra: Verify by Induction

- Suppose I want to prove that the recurrence $T(n) = 2T(n/2) + 4n, T(1) = 8$ evaluates to $T(n) = O(n \log n)$
- I need to show that for all sufficiently large n, I can find a constant c, such that $T(n) \leq c \cdot n \log n$
- Base case?
  - $T(1) = 8 \nleq c \log 1 = 0$ (doesn't work yet, let us fix it up later)
- Assume holds for all $< n$
- $$\begin{aligned} T(n) &\leq 2(c(n/2)\log(n/2)) + n \\ &= cn \log(n/2) + n \\ &= cn \log n - cn \log 2 + n \leq cn \log n \text{ if } c \geq 1 \end{aligned}$$

Extra: Verify by Induction

- What about the base case?
- As long as $n \geq 4$, our recurrence does not depend on $T(1)$;
- We can just use $T(2)$ as the base case our induction!
  $T(2) = 2T(1) + 8 = 24 \leq c \log 2$ for $c > 24$
- Thus our induction holds for all $n \geq 2$ and $c > 24$
- This is how we usually verify our recurrences and prove they are correct: by induction.

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