CSE 331: Algorithms & Complexity

"Divide & Conquer"

Prof. Charlie Anne Carlson (She/Her)

Lecture 22

Friday October 24th, 2025

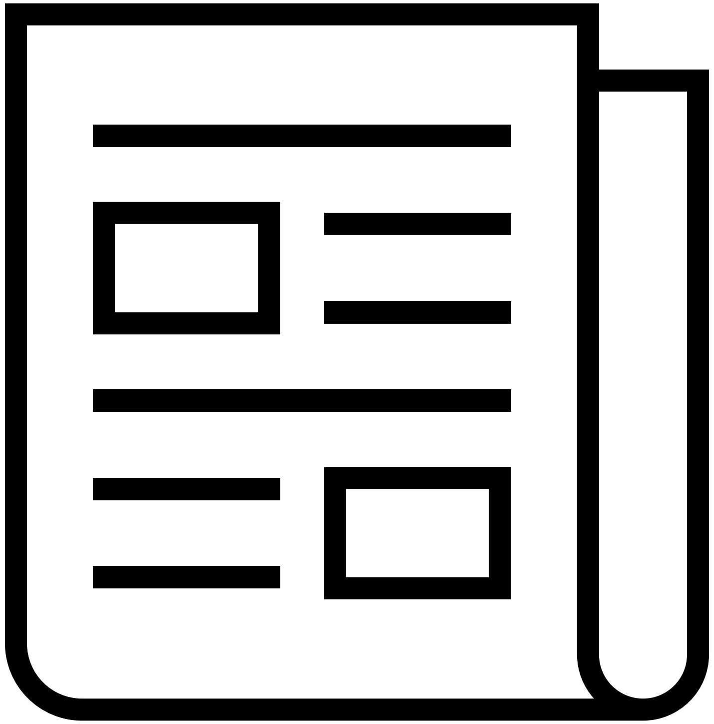
University at Buffalo

Schedule

1. Course Updates
2. Divide & Conquer
3. Merge Sort
4. Solving Recurrences

Course Updates

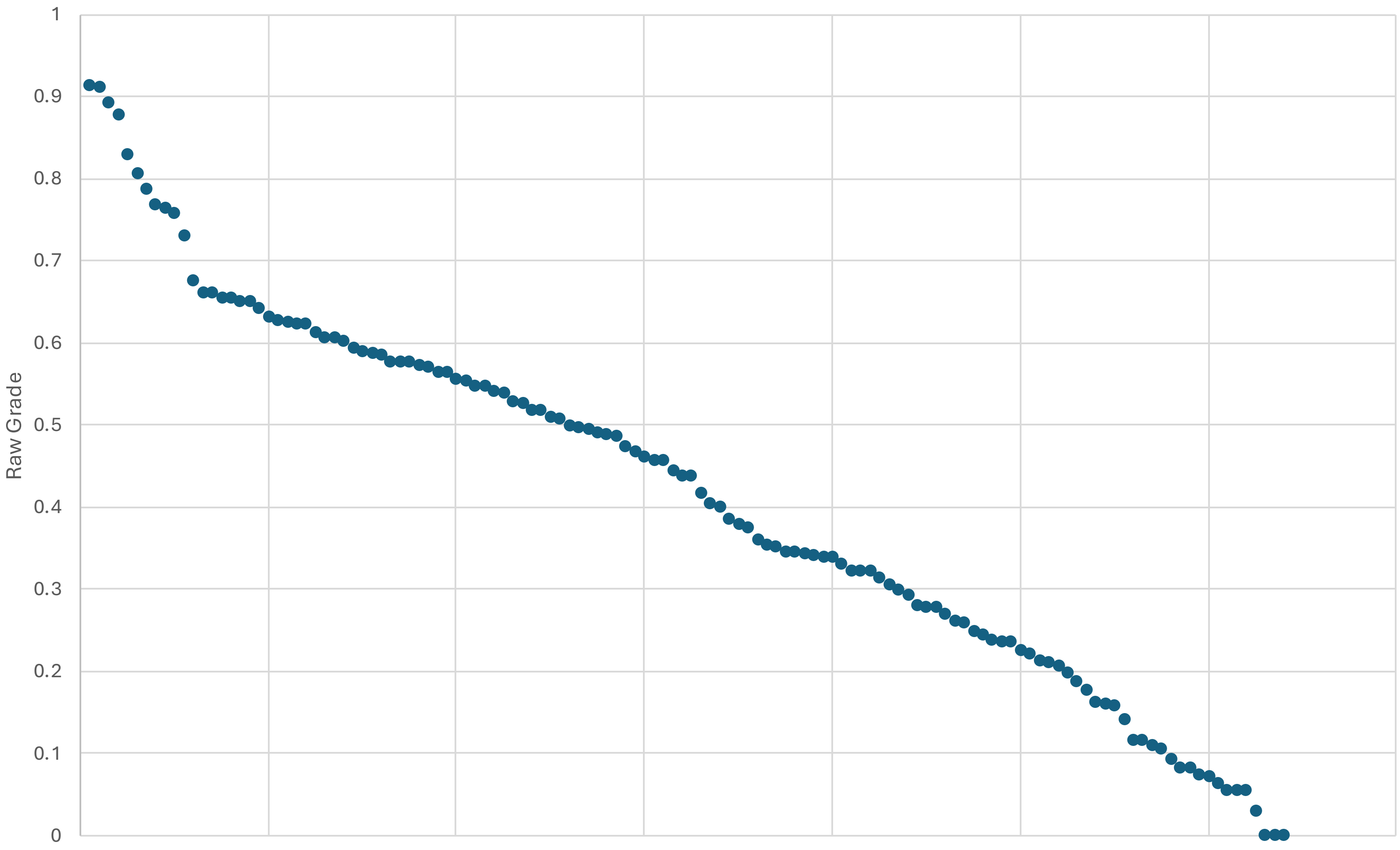
- Post Midterm Grades In Progress
- Feedback Survey Soon
- HW 4 Solutions Soon
- HW 5 Out
- Group Project
    - First Problems Oct 31st
- More Example Quizes



Midterm Evals Grades (One HW Drop)

Grade Includes:

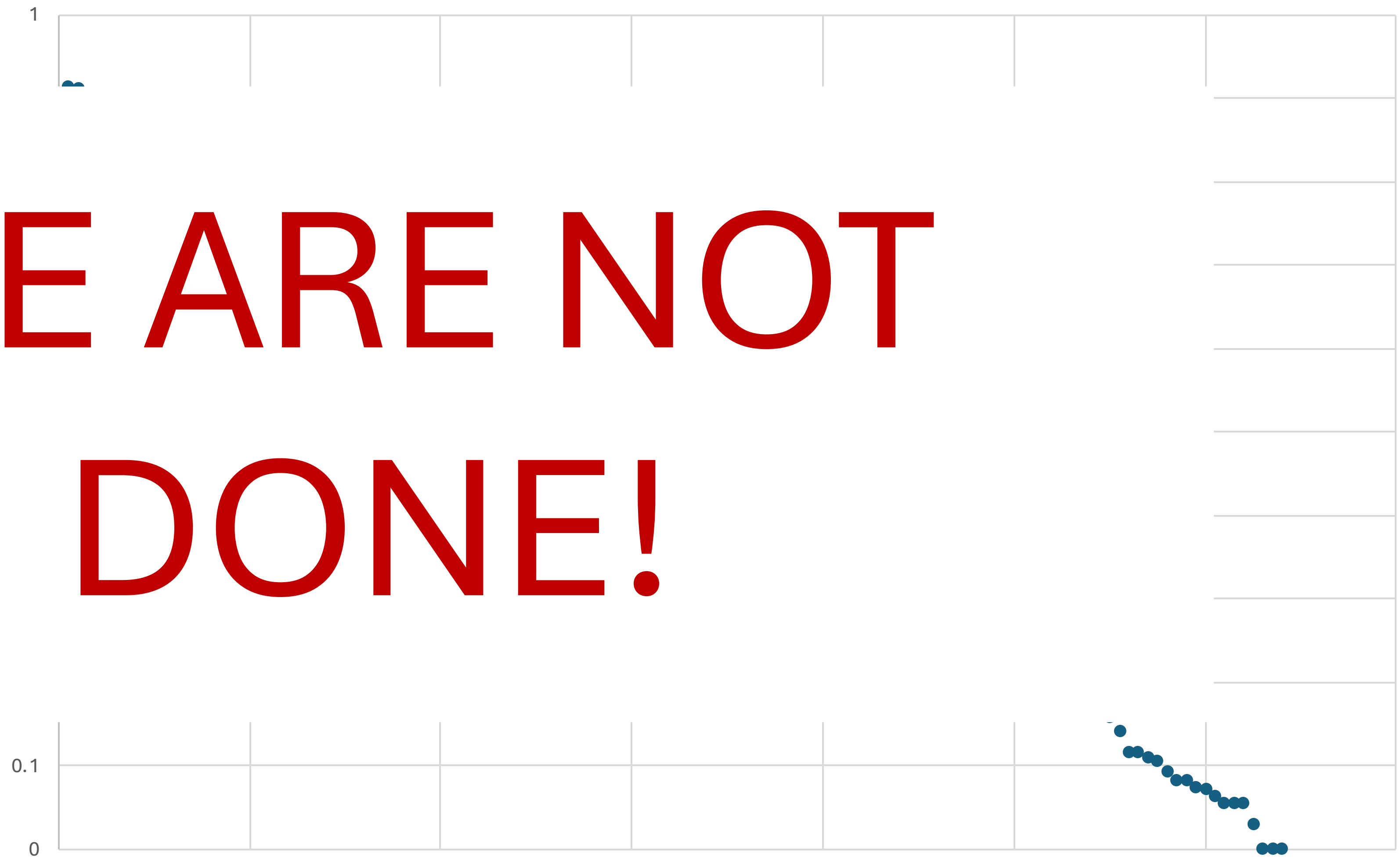
- Midterm (45%)
- Top HW (49%)
  - Up To HW 3
  - Per Part
- QUIZ (6%)
  - Quiz 1



Midterm Evals Grades (One HW Drop)

Grade Includes:

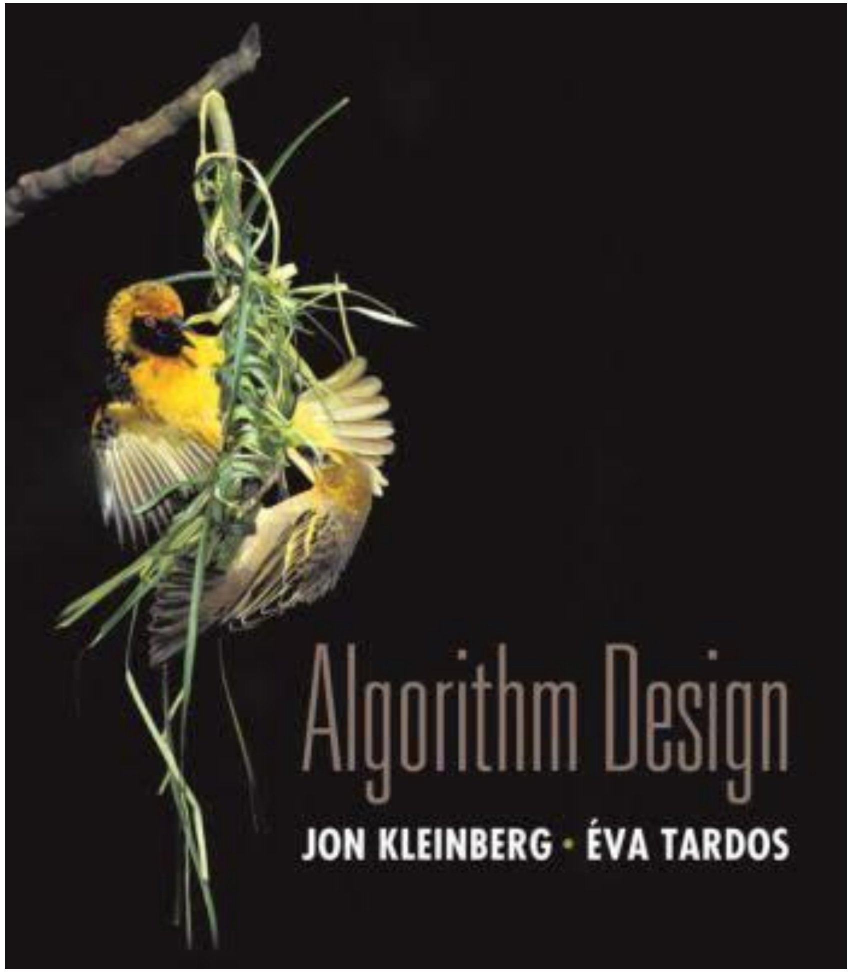
- M
- Tc
  -
  -
- Q
  -



WE ARE NOT DONE!

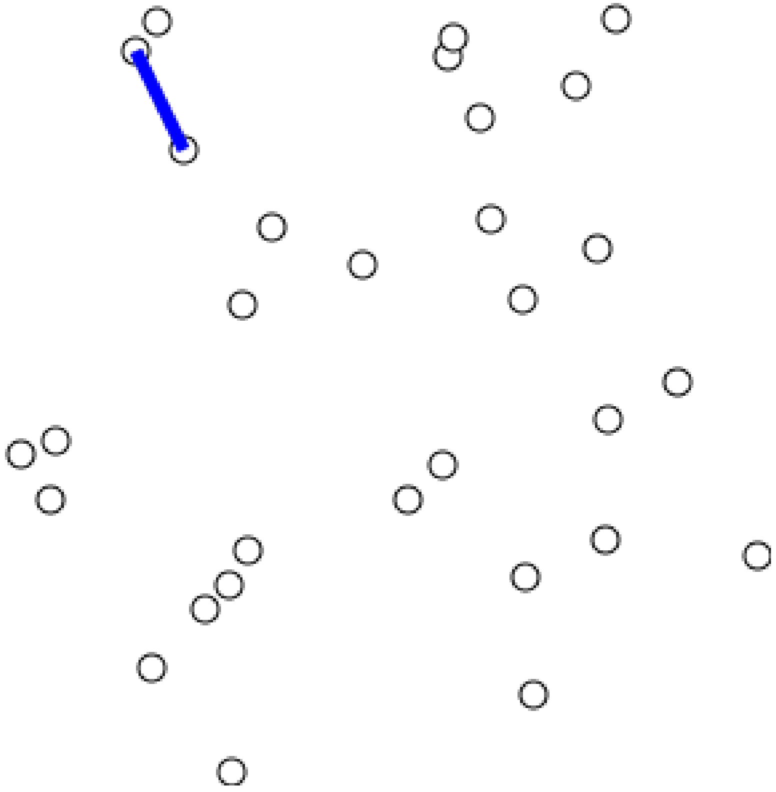
Reading

- You should have read:
  - Finished KT 5.1
  - Started KT 5.2
- Before Next Class:
  - Finish KT 5.2
  - Start KT 5.3



Prim's Algorithm Runtime

- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
  - Pick s in V arbitrarily
  - Let S = {s}
  - While S != V:
    - Find minimum weight edge e = (u,v) where u is in S but v is not.
    - Add v to S

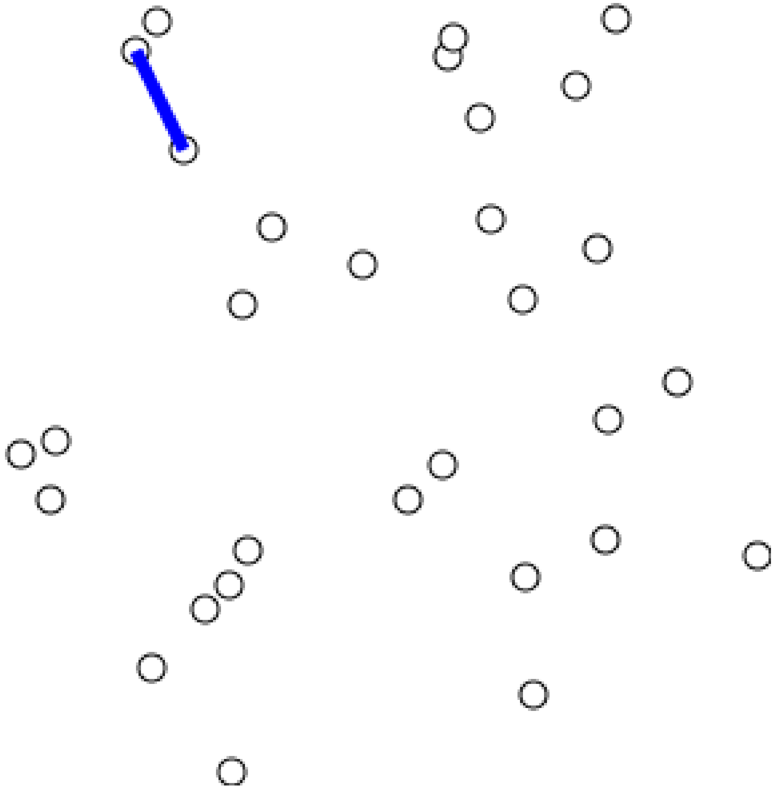


https://en.wikipedia.org/wiki/File:PrimAlgDemo.gif

Prim's Algorithm Runtime O(m log(n))

Claim: Using a priority queue, Prim's Algorithm can be implemented on a graph with n nodes and m edges to run in O(m) time, plus the time for n `PopMin` and m `DecreasePriority` operations.

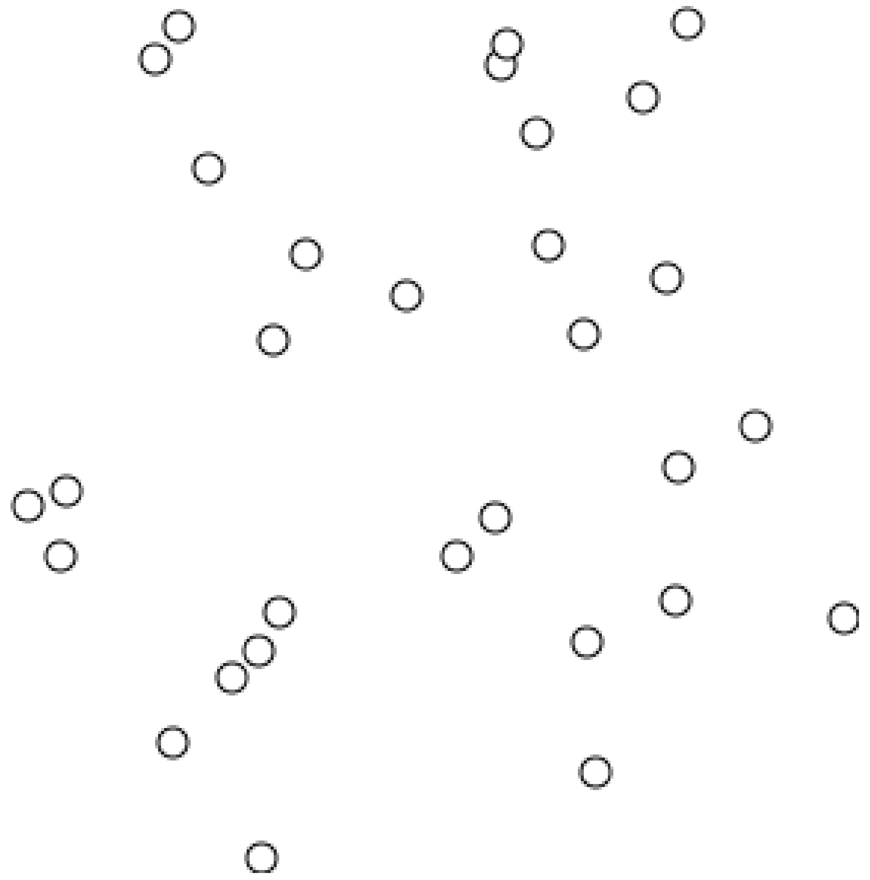
Corollary: Using a heap-based priority queue we get a running time of O(m log(n)).



https://en.wikipedia.org/wiki/File:PrimAlgDemo.gif

Kruskal’s Algorithm Runtime

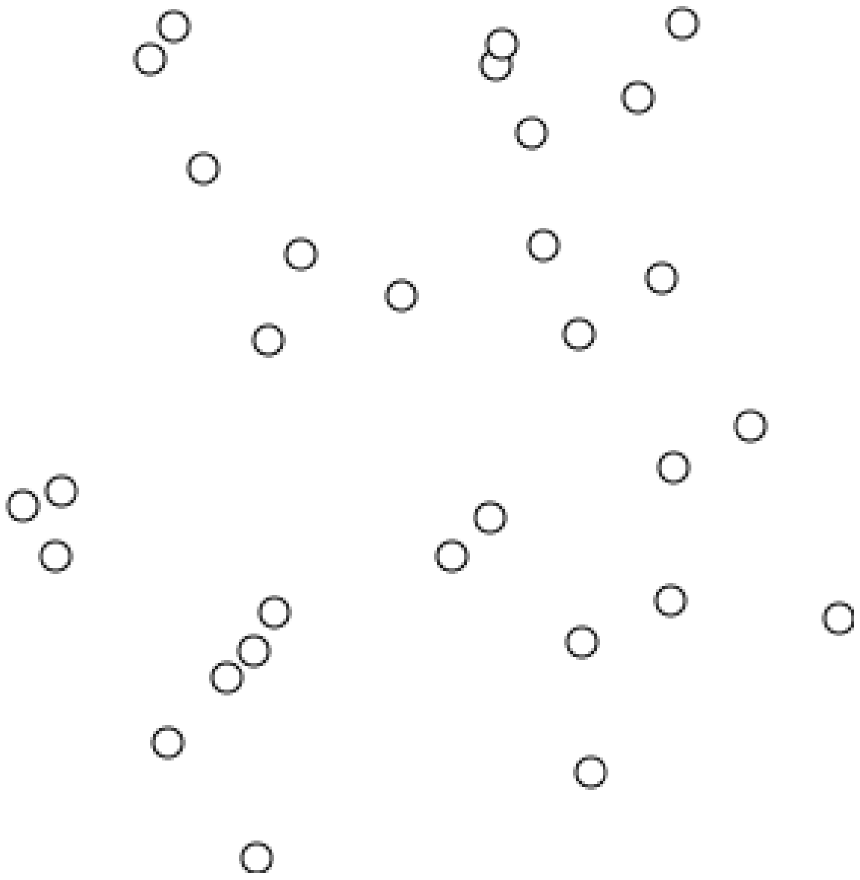
- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
  - Sort E using values in L
    - Break ties arbitrarily
  - Let T be an empty graph
  - For e in E:
    - If adding e to T doesn’t case a cycle, add it.



https://en.wikipedia.org/wiki/File:KruskalDemo.gif

Kruskal's Algorithm Runtime O(m log(n))

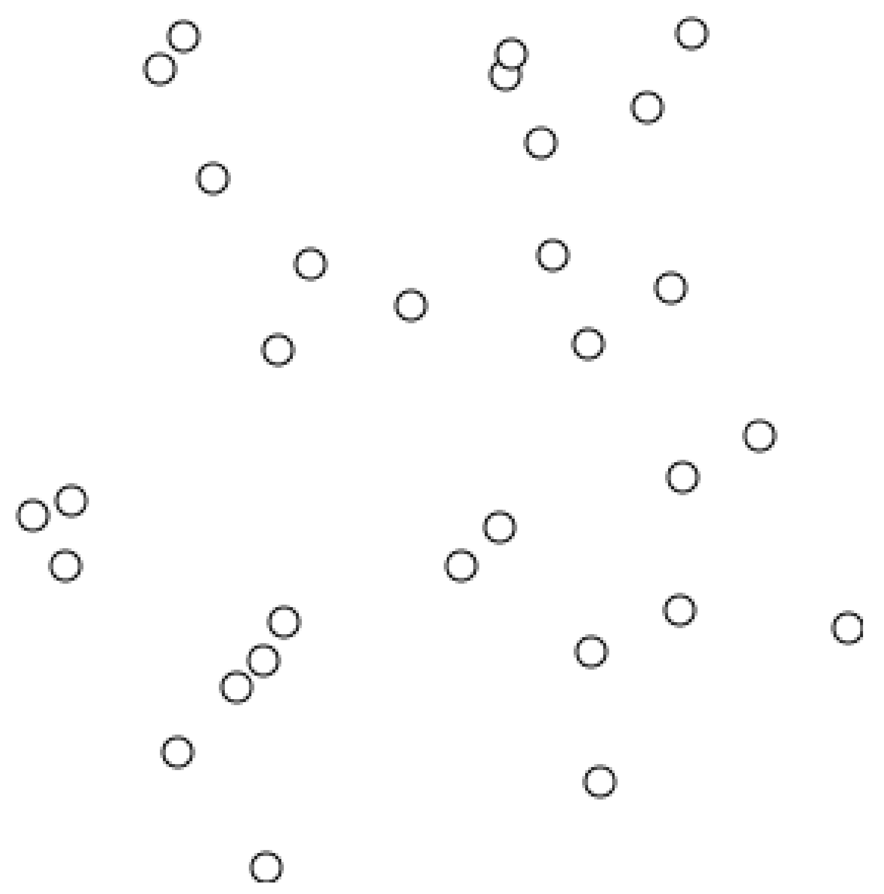
- **Input:** Undirected graph G = (V,E) and weights L
- **Output:** MST of G
  - Sort E using values in L
    - Break ties arbitrarily
  - Let T be an empty graph
  - For e in E:
    - If adding e to T doesn't case a cycle, add it.



https://en.wikipedia.org/wiki/File:KruskalDemo.gif

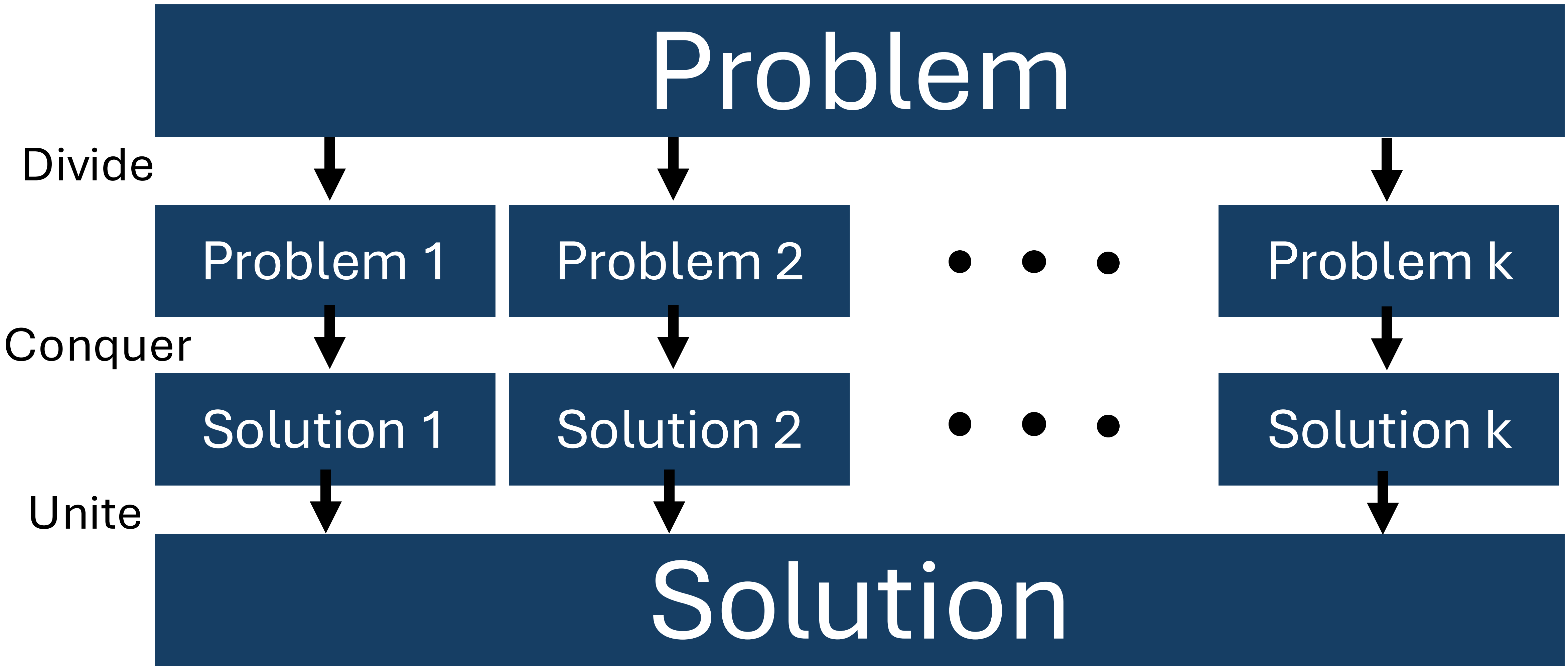
Kruskal’s Algorithm Runtime O(m log(n))

- To prove this running time, we need a Union-Find data structure.
  - Keeps track of which elements in a ground set belong to the same subsets.
  - `Find(u)`: Returns name of set that contains u.
  - `Union(A,B)` combine sets A and B into one set.
  - Read KT 4.6!

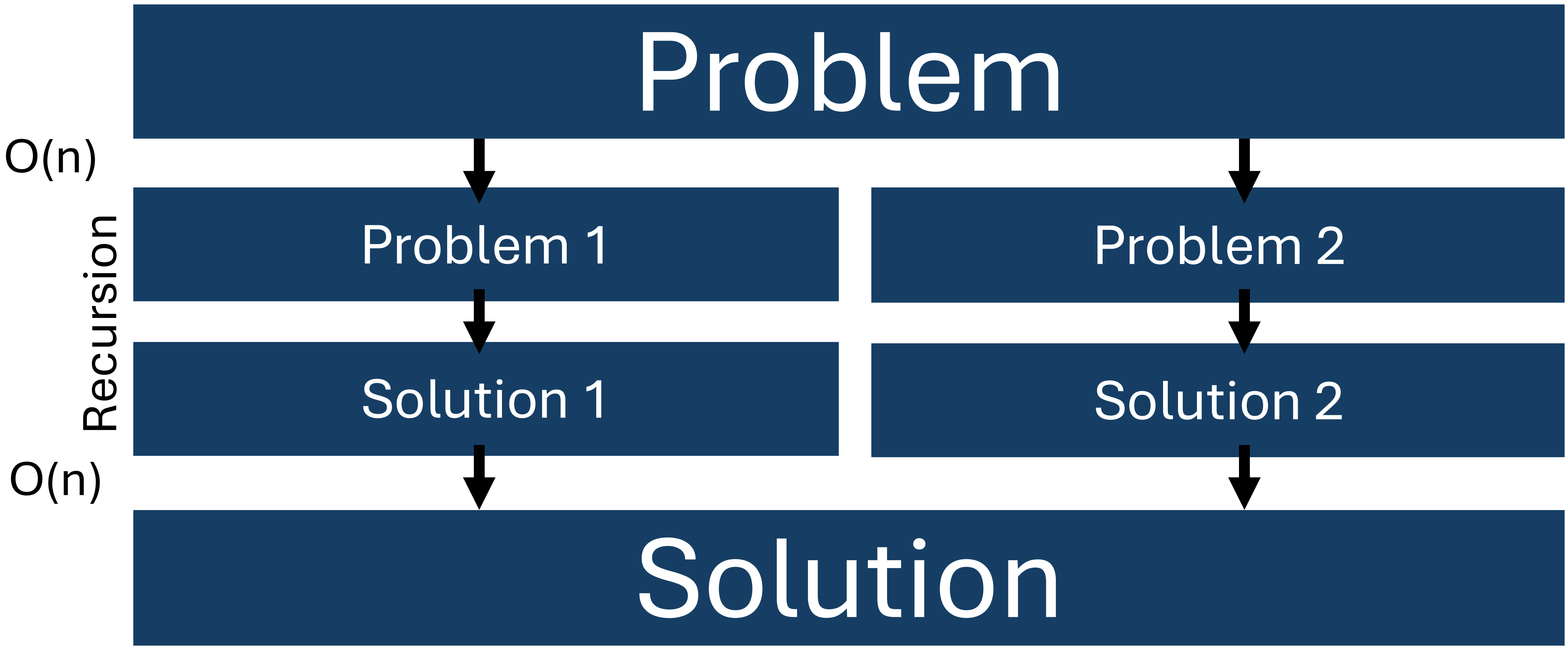


https://en.wikipedia.org/wiki/File:KruskalDemo.gif

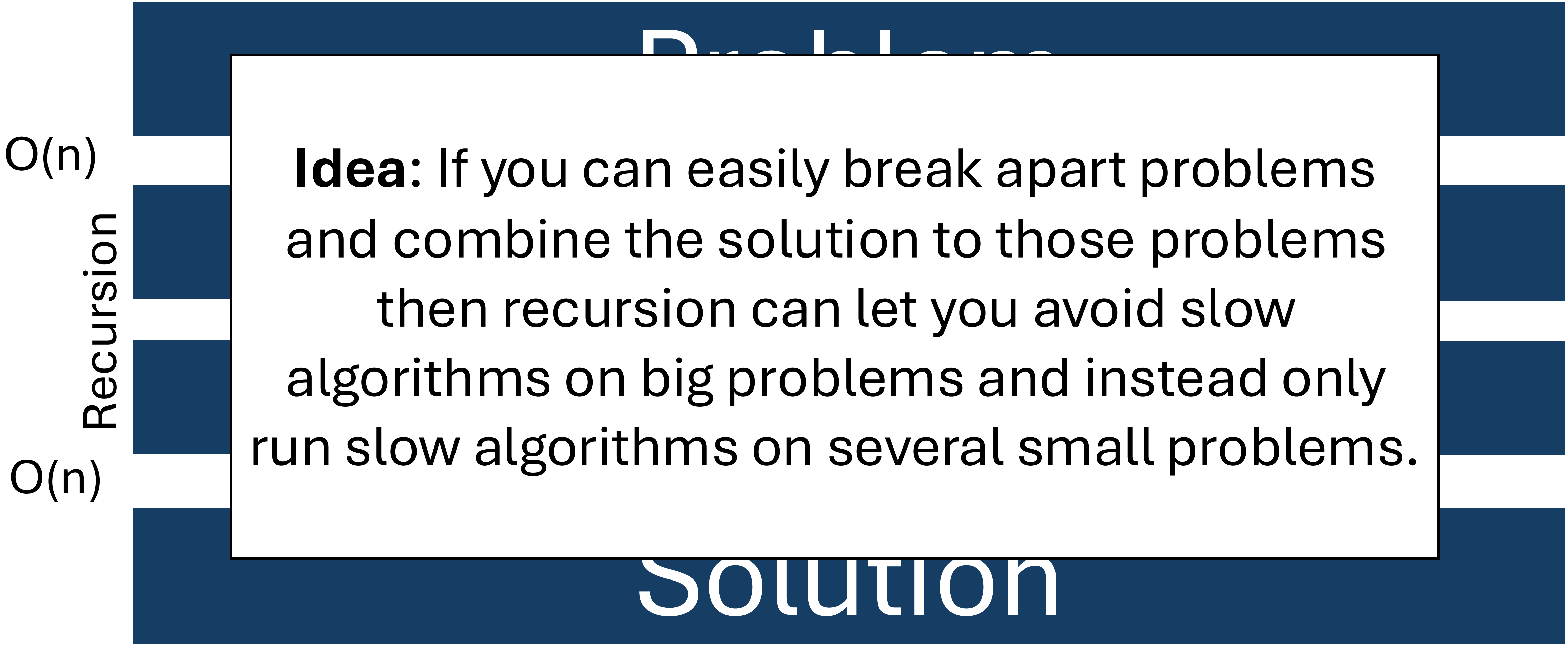
Divide & Conquer (KT 5.1 and KT 5.2)



Why Divide & Conquer?

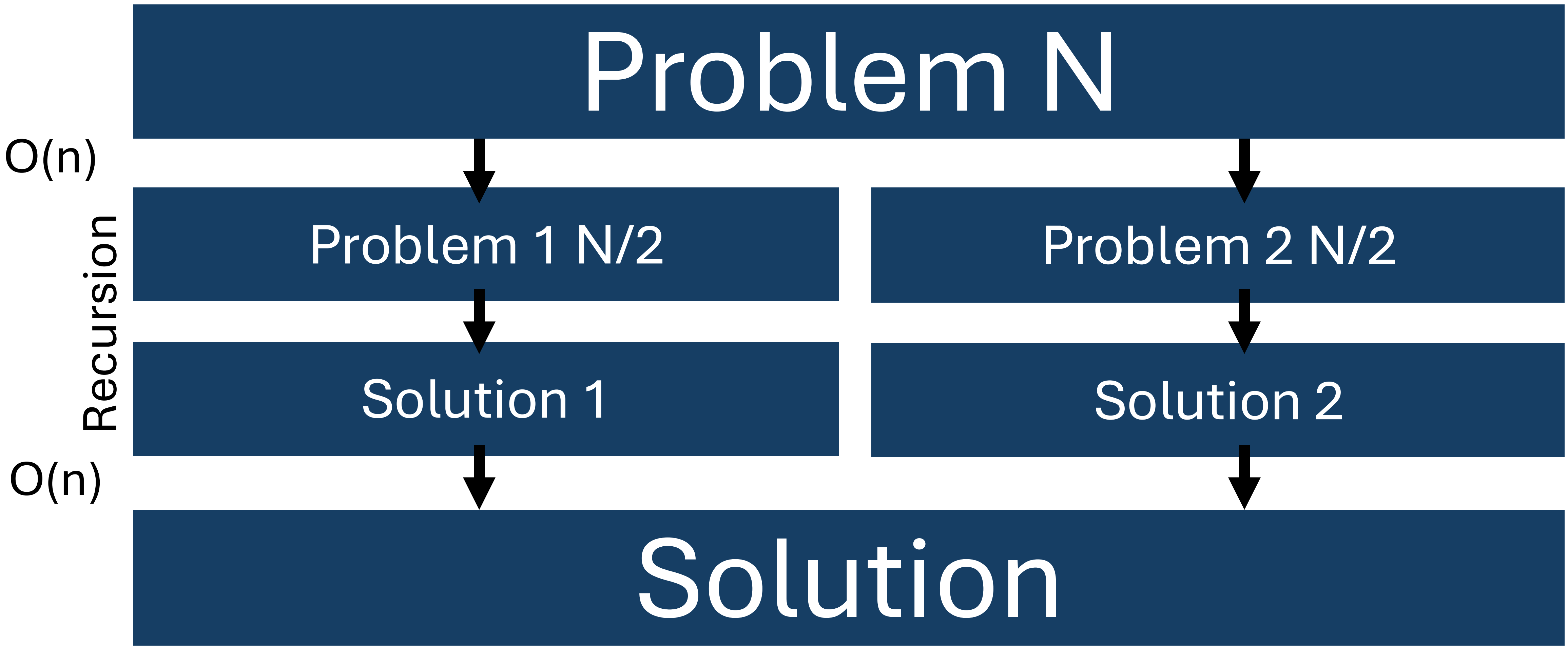


Why Divide & Conquer?

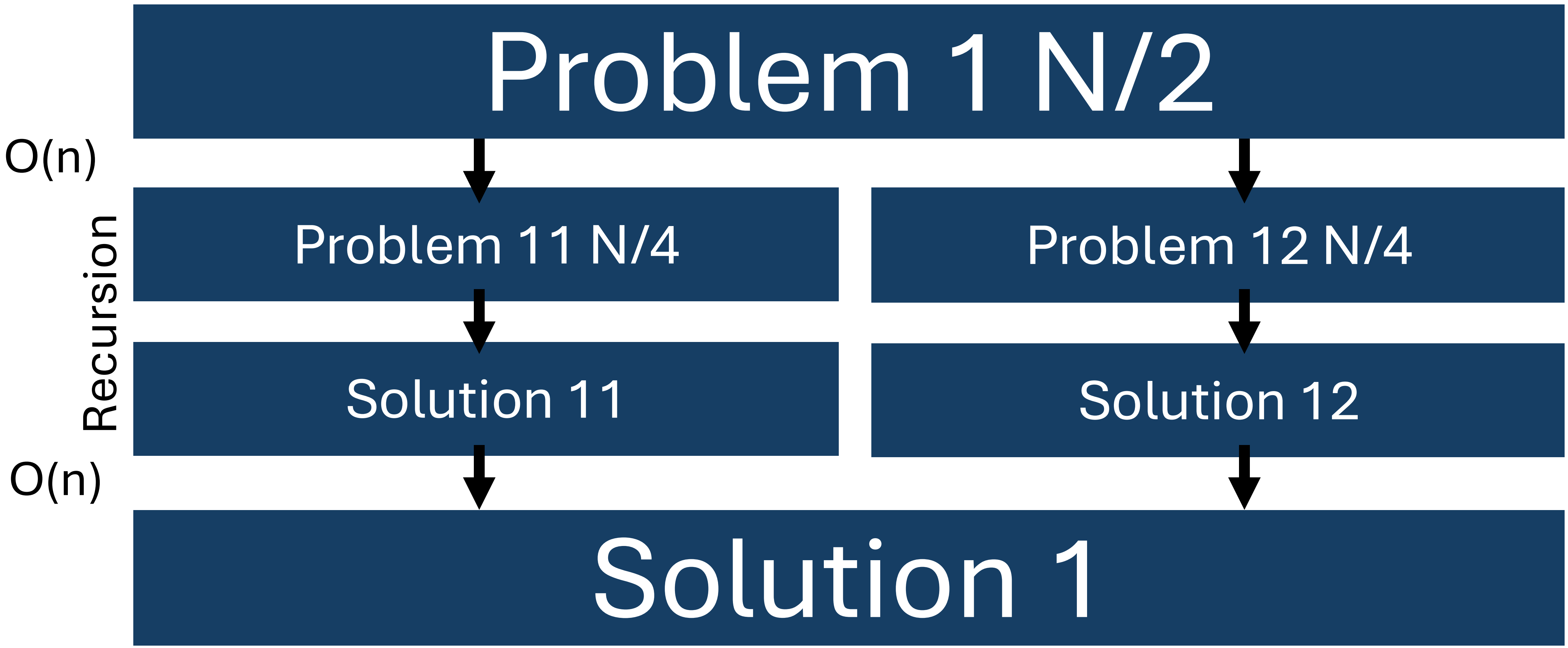


Idea: If you can easily break apart problems and combine the solution to those problems then recursion can let you avoid slow algorithms on big problems and instead only run slow algorithms on several small problems.

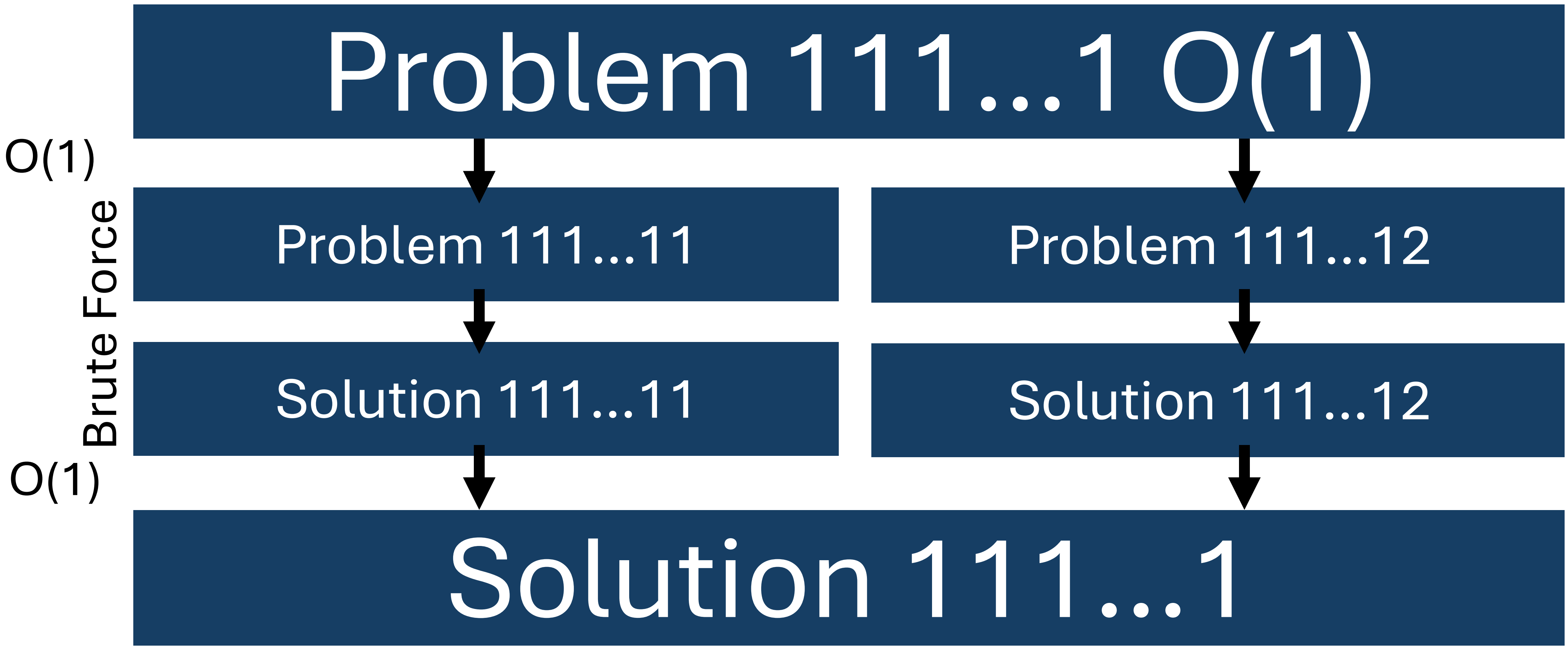
Why Divide & Conquer?



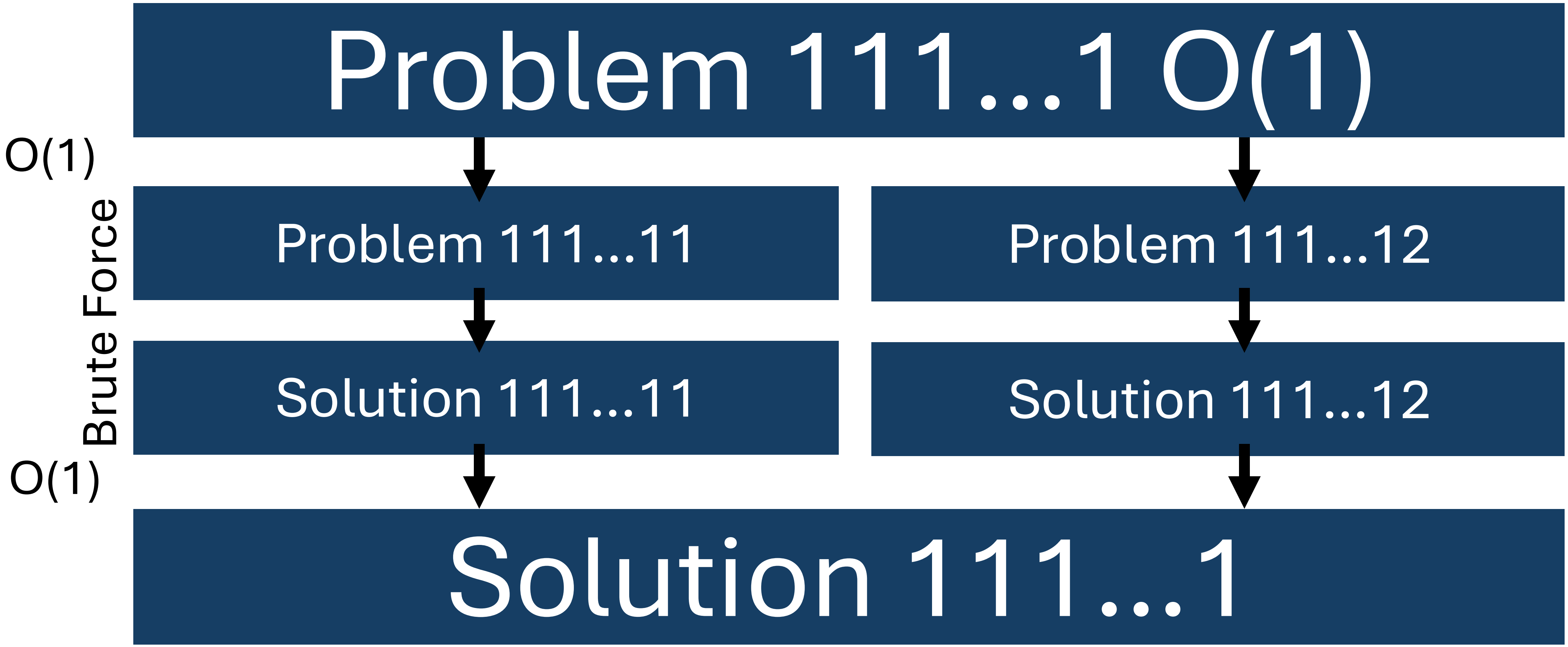
Why Divide & Conquer?



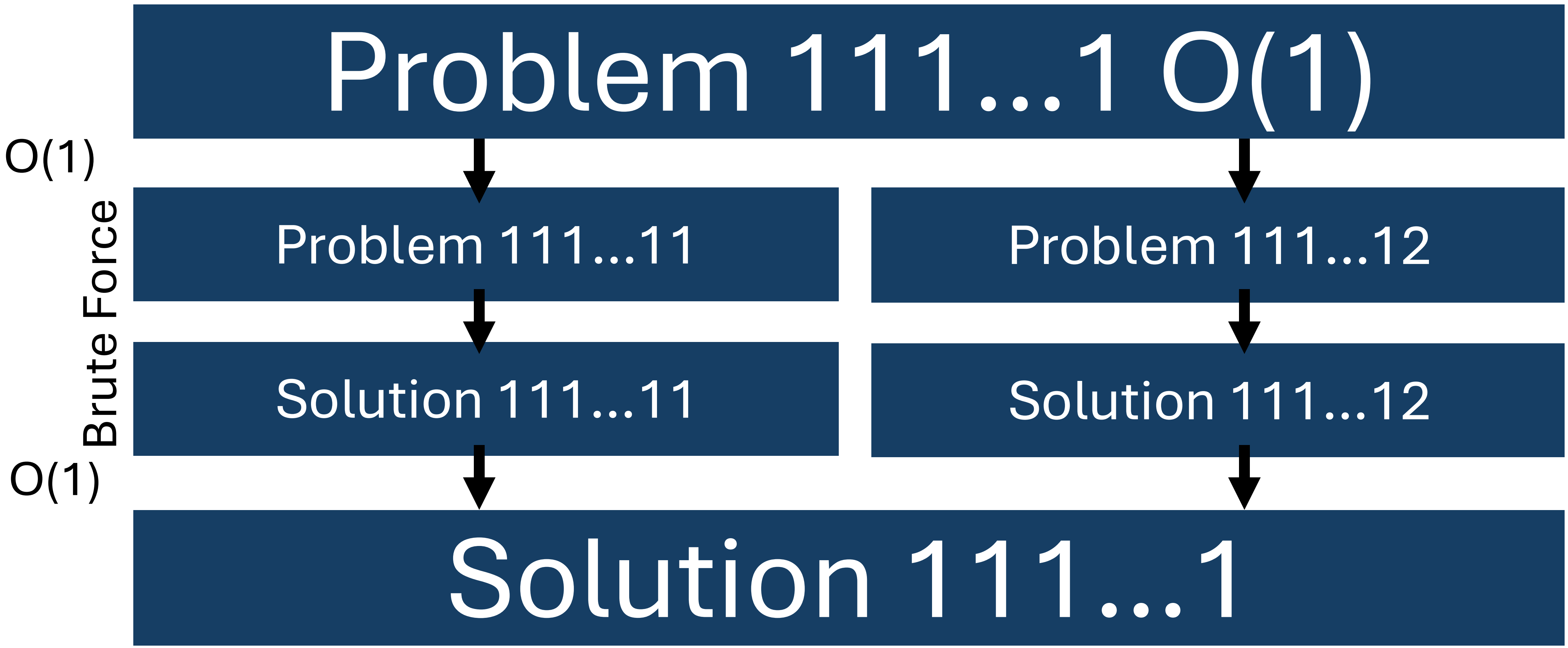
Why Divide & Conquer?



Q: How many times can you split in half?

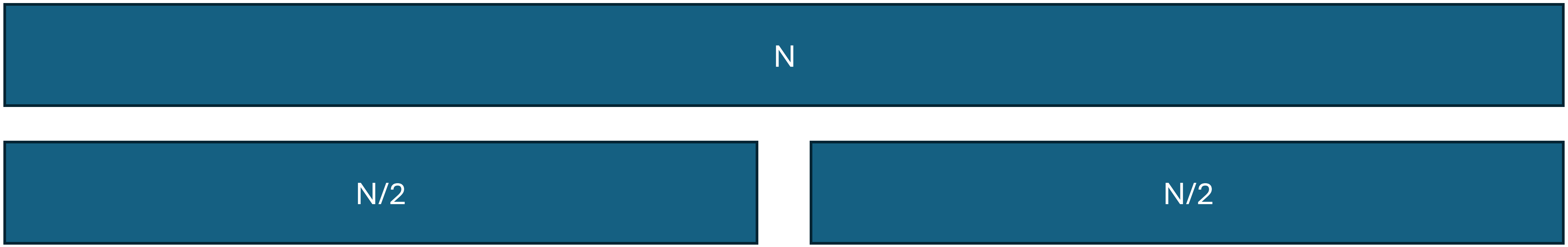


A: After log times, you will get a constant!

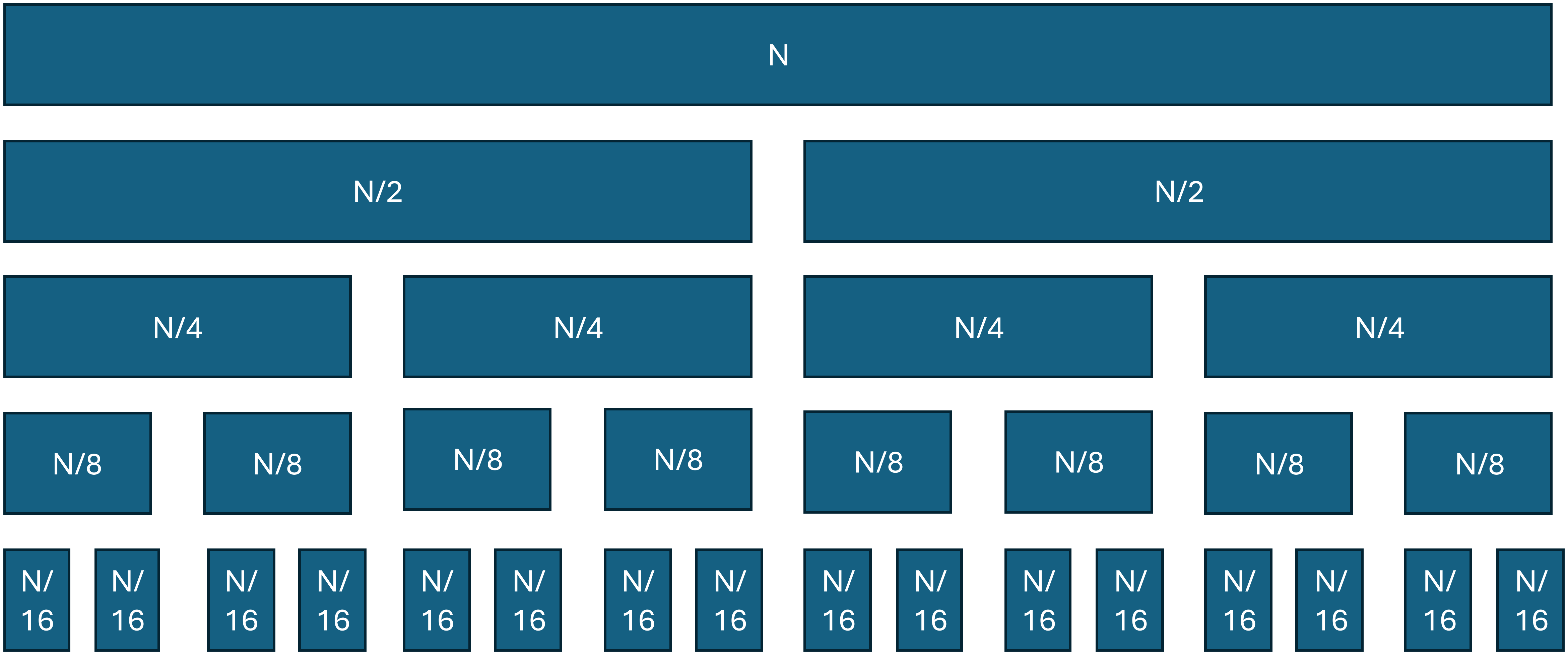


Another Look

- Consider a thermotical problem PROBLEM.
- Suppose we have an algorithm for PROBLEM that takes N^2 time to run on an input of size N.
- Suppose that we can in N time break PROBLEM into two small problems of size N/2 such that if we find their solutions we get the solution the original problem in N time (we can combine the solutions).

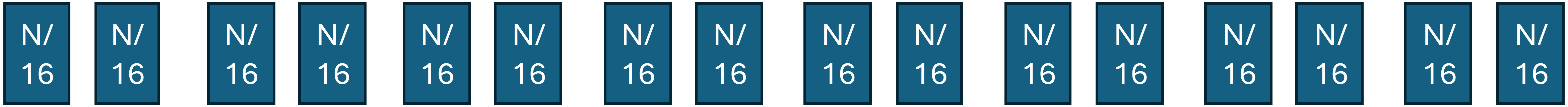


Another Look

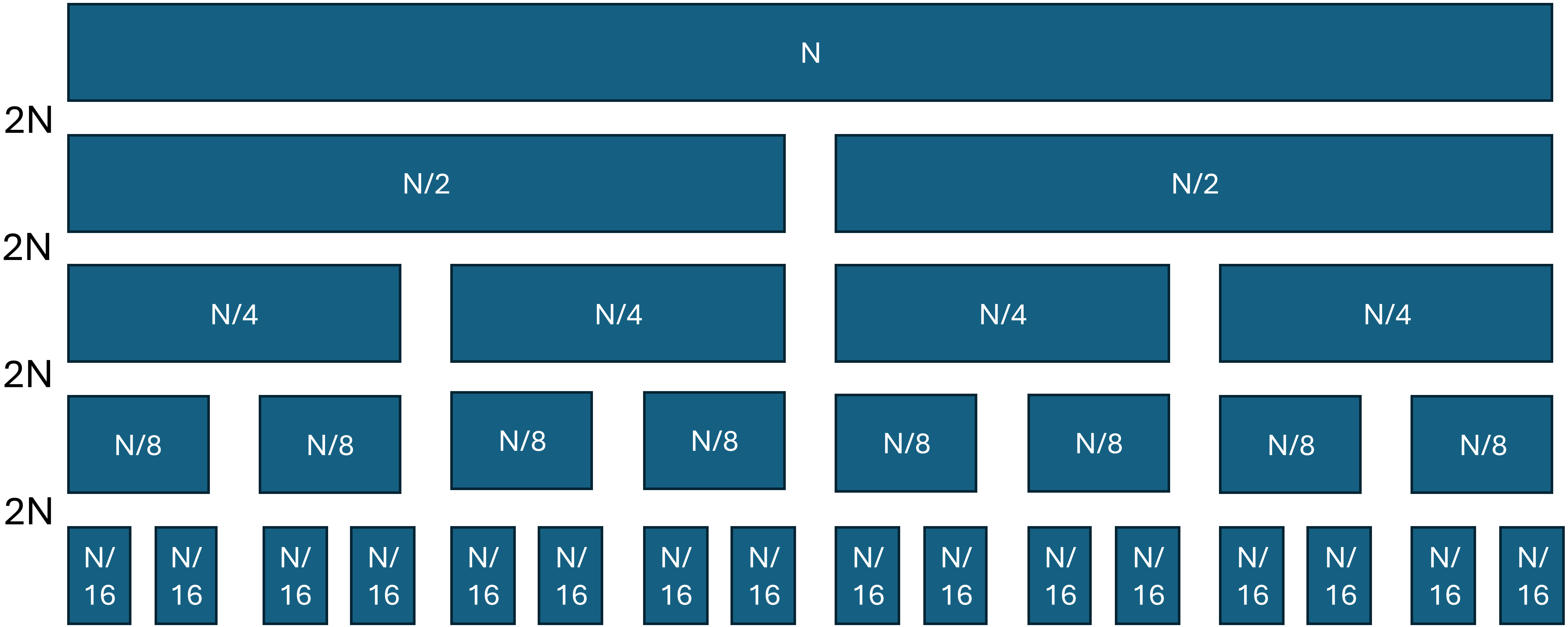


Another Look

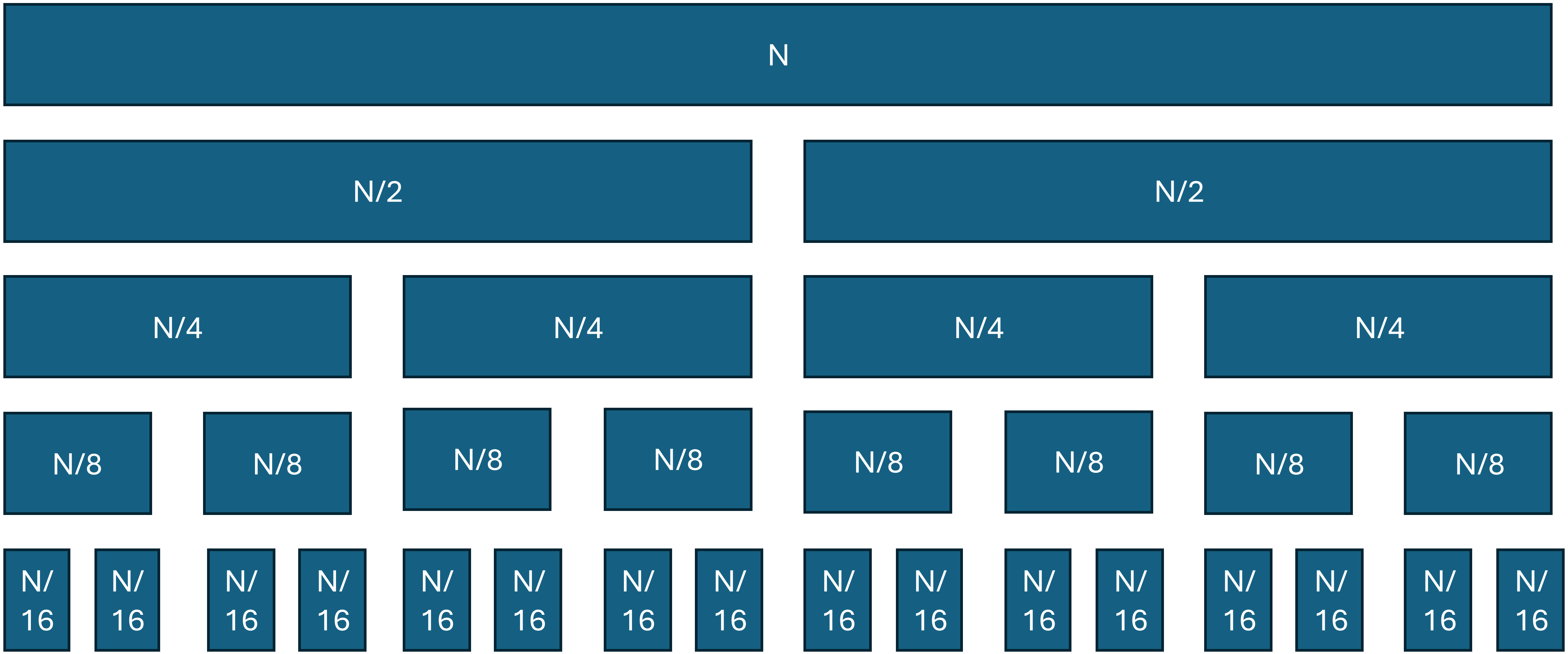
- Observe that it would take N^2 time to use the algorithm for PROBLEM on the initial input.
- Instead consider running the same algorithm on the subproblems of size N/16.
  - How much work is done?



It takes 8N to do splitting and combining



It takes N^2/16 to do algorithm on small parts.



16 Problems each of size N/16 and each taking time (N/16)^2 time.

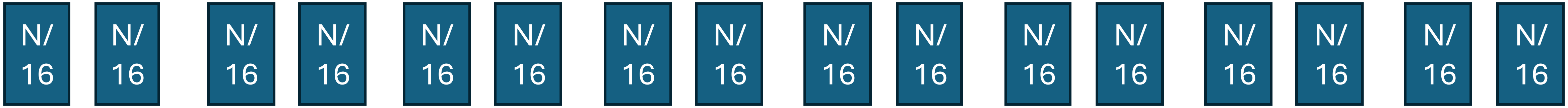
Another Look

- Observe that it would take N^2 time to use the algorithm for PROBLEM on the initial input.
- Instead consider running the same algorithm on the subproblems of size N/16.
  - Q: How much work is done?
    - A: It takes a total of N^2/16 + 10N time.
- Q: When is N^2 > N^2/16 + 10N?

N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16 N/16

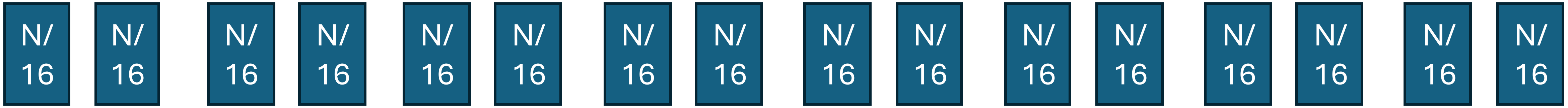
Another Look

- Observe that it would take N^2 time to use the algorithm for PROBLEM on the initial input.
- Instead consider running the same algorithm on the subproblems of size $N/16$.
  - Q: How much work is done?
    - A: It takes a total of $N^2/16 + 10N$ time.
- Q: When is $N^2 > N^2/16 + 10N$?
  - A: When $N > 32/3$



Another Look

- Observe that it would take N^2 time to use the algorithm for PROBLEM on the initial input.
- Instead consider running the same algorithm on the subproblems of size $N/16$.
    - Q: How much work is done?
        - A: It takes a total of $N^2/16 + 10N$ time.
- Q: When is $N^2 > N^2/16 + 10N$?
    - A: When $N > 32/3$ **<- So there is speedup as N grows!**



Sorting

- **Problem**: Given a list of n numbers L, rearrange them in ascending order.

- E.g.
  - **Input**: [3,2,5,5,1,6,7,8]
  - **Output**: [1,2,3,5,5,6,7,8]

Sorting

- **Problem**: Given a list of n numbers L, rearrange them in ascending order.

- Sorting Algorithms:
    - Bubble Sort
    - Insertion Sort
    - Mergesort
    - Radix Sort
    - Quicksort
    - Introsort

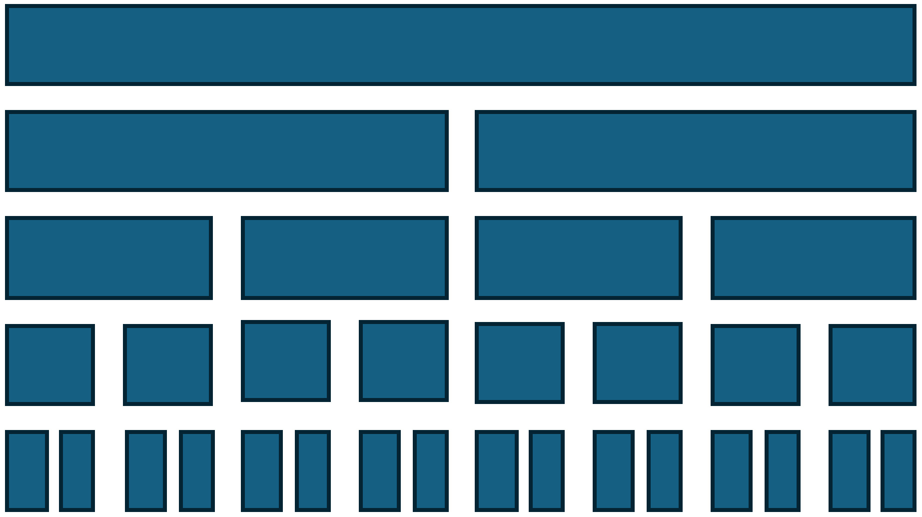
Sorting

- **Problem**: Given a list of n numbers L, rearrange them in ascending order.

- Sorting Algorithms:
  - Bubble Sort
  - Insertion Sort
  - **Mergesort**
  - Radix Sort
  - Quicksort
  - Introsort

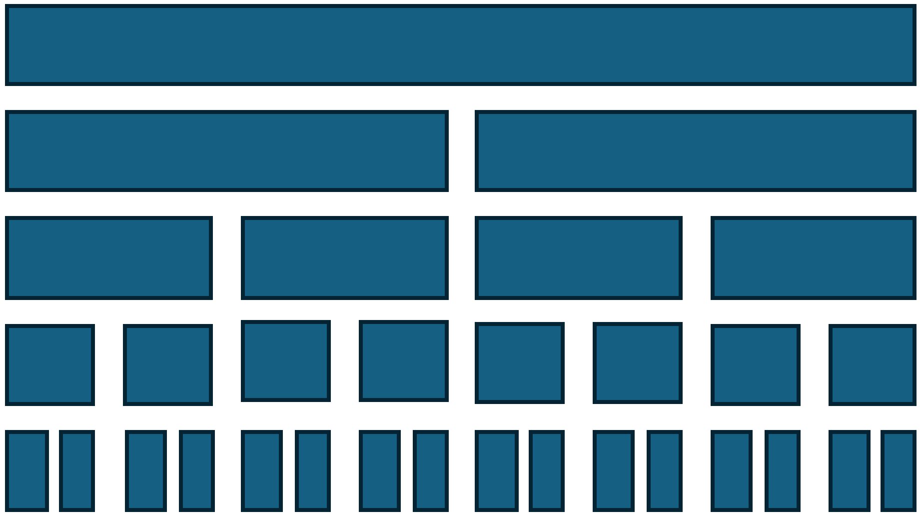
Mergesort

- **Divides**: Divides input into two pieces of equal size in linear time.
  - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece.
- **Unite**: Merges the two sorted lists in linear time.



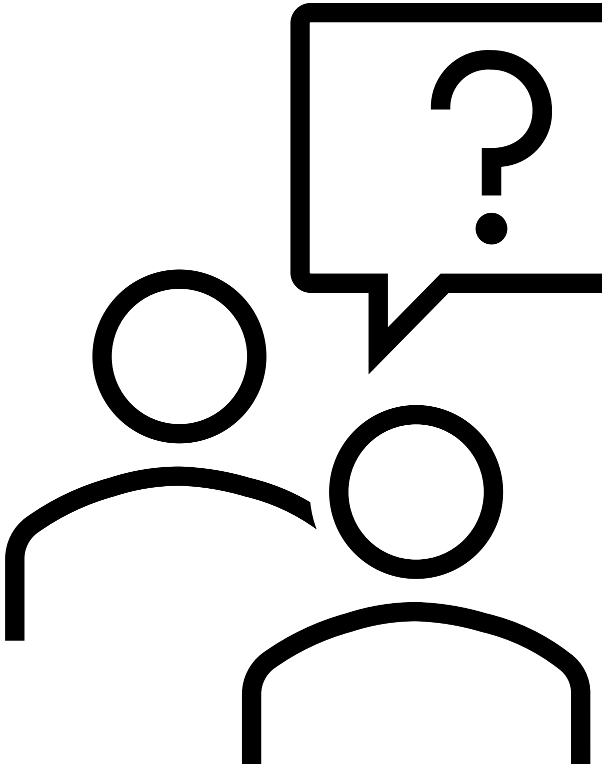
Mergesort

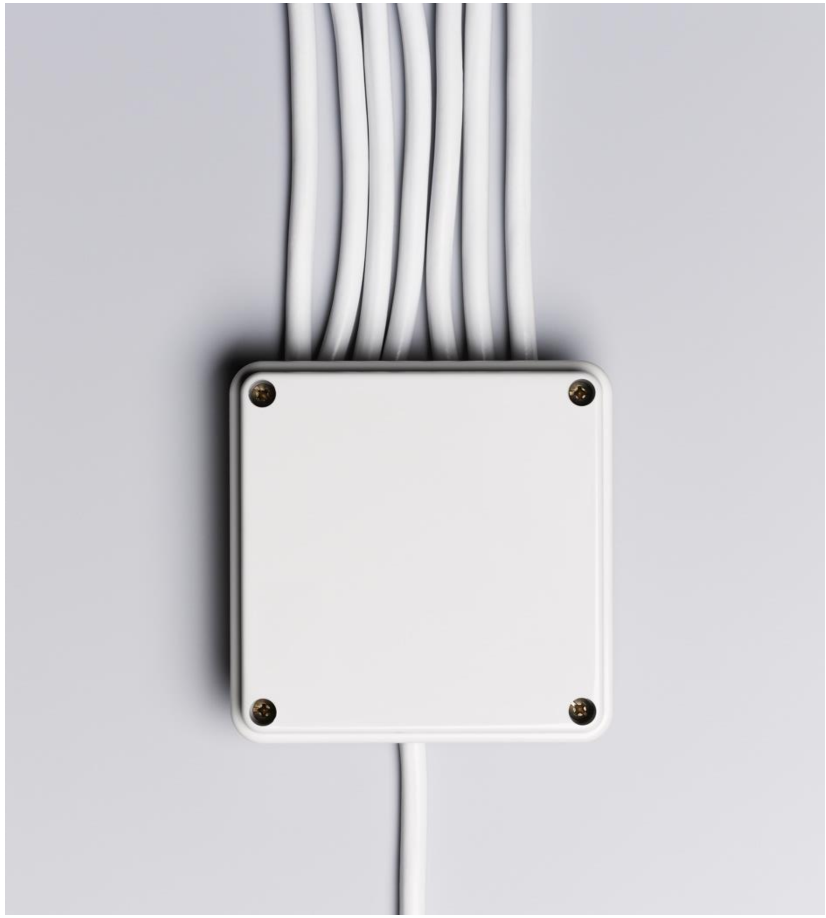
- **Base Case:** If array has length less than 2, brute force.
- **Divides**: Divides input into two pieces of equal size in linear time.
  - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece.
- **Unite**: Merges the two sorted lists in linear time.



Sorting

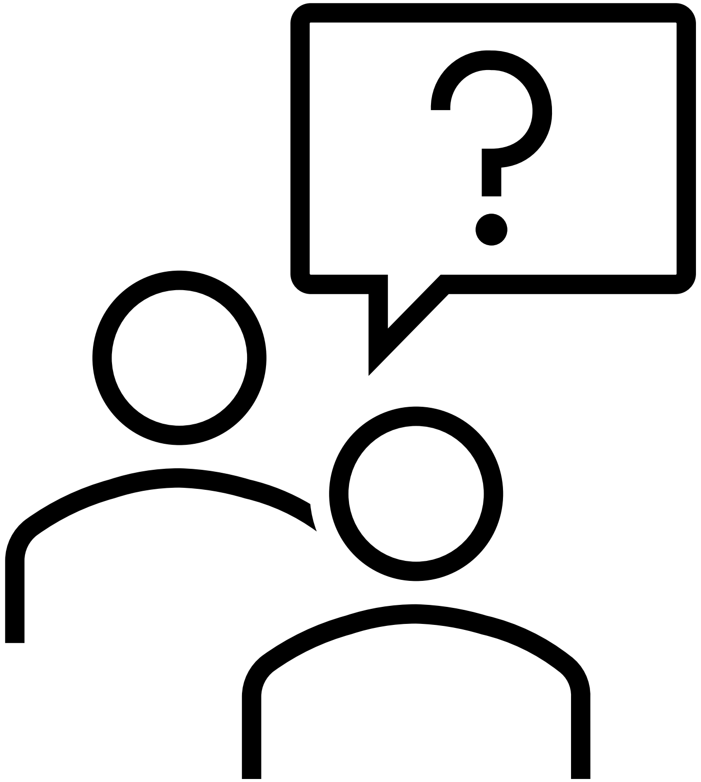
- **Problem**: Given two sorted lists A and B, find a sorted list of their union.





Merging

- **Input**: Two sorted lists A and B of length n/2
- **Output**: Sorted list of A and B
- Initialize list C to be empty
- Let i = 0 and j = 0
- While (i < n/2 or j < n/2):
    - If j == n/2 or A[i] <= B[j]:
        - C.append(A[i])
        - i += 1
    - Else:
        - C.append(B[j])
        - j += 1

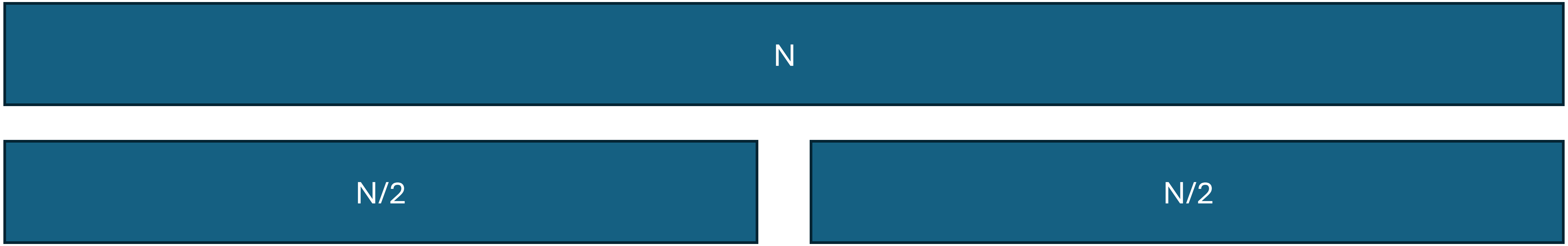


Mergesort Runtime?

- **Base Case:** If array has length less than 2, brute force.
- **Divides**: Divides input into two pieces of equal size in linear time.
    - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece.
- **Unite**: Merges the two sorted lists in linear time.

Let T(n) be runtime of Mergesort.

- **Base Case:** If array has length less than 2, brute force. **O(1)**
- **Divides**: Divides input into two pieces of equal size in linear time. **O(n)**
    - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece. **T(n/2)**
- **Unite**: Merges the two sorted lists in linear time. **O(n)**



Let T(n) be runtime of Mergesort.

- **Base Case:** If array has length less than 2, brute force. **O(1)**
- **Divides**: Divides input into two pieces of equal size in linear time. **O(n)**
  - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece. **T(n/2)**
- **Unite**: Merges the two sorted lists in linear time. **O(n)**

$$\boldsymbol{T(n)} \leq ?$$

Let T(n) be runtime of Mergesort.

- **Base Case:** If array has length less than 2, brute force. **O(1)**
- **Divides**: Divides input into two pieces of equal size in linear time. **O(n)**
  - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece. **T(n/2)**
- **Unite**: Merges the two sorted lists in linear time. **O(n)**

$$\boldsymbol{T(n)} \le \begin{cases} O(1) & n \le 2 \\ 2T(n/2) + O(n) & o.w. \end{cases}$$

Let T(n) be runtime of Mergesort.

- **Base Case:** If array has length less than 2, brute force. **O(1)**
- **Divides**: Divides input into two pieces of equal size in linear time. **O(n)**
- **Conquer**: Recursively calls mergesort on each piece. **T(n/2)**
- **Unite**: Merges the two sorted lists in linear time. **O(n)**

$$\boldsymbol{T(n)} \leq \begin{cases} O(1) & n \leq 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil + c'n & o.w. \end{cases}$$

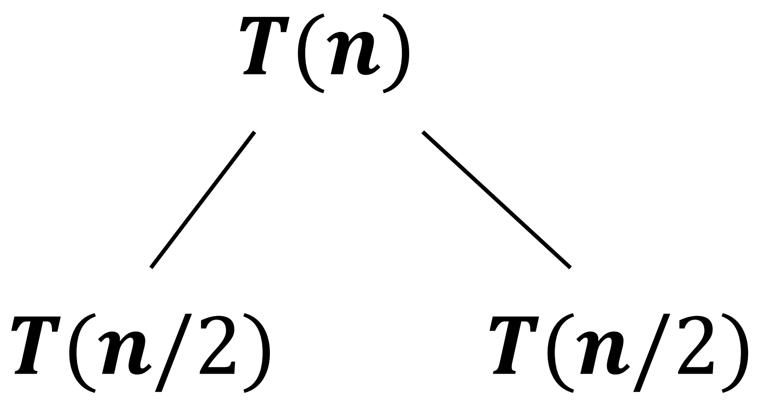
How do you solve a recurrence?

- **Unrolling:** We analyze the first few ”levels” of the recursion, find a pattern and then prove that the pattern is correct.
- **Guess and Check:** We guess what the answer and the substitute it in to check that it works. That is, we prove it works.
- REVIEW KT 5.1 and KT 5.2 IF YOU HAVEN’T ALREADY!

$$\boldsymbol{T(n)} \leq \begin{cases} O(1) & n \leq 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil + c'n & o.w. \end{cases}$$

Unrolling

- **Unrolling:**
  - Sketch out a few levels of the ”recursion tree”
  - Identify how many problems on each level.
  - Identify how much work done at each level.
  - Identify how small each problem is at each level.
  - Identify how many levels before base case.



$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

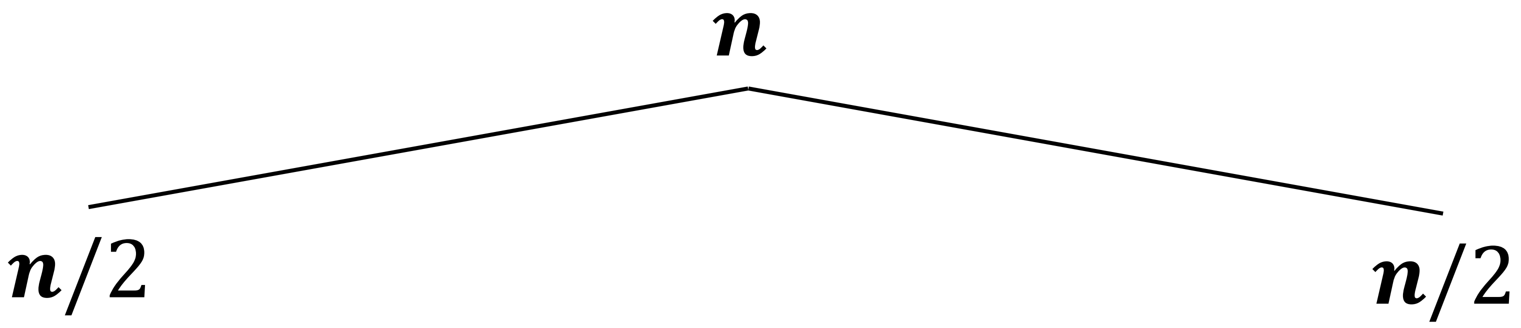
Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c' & o.w. \end{cases}$$

$\boldsymbol{n}$

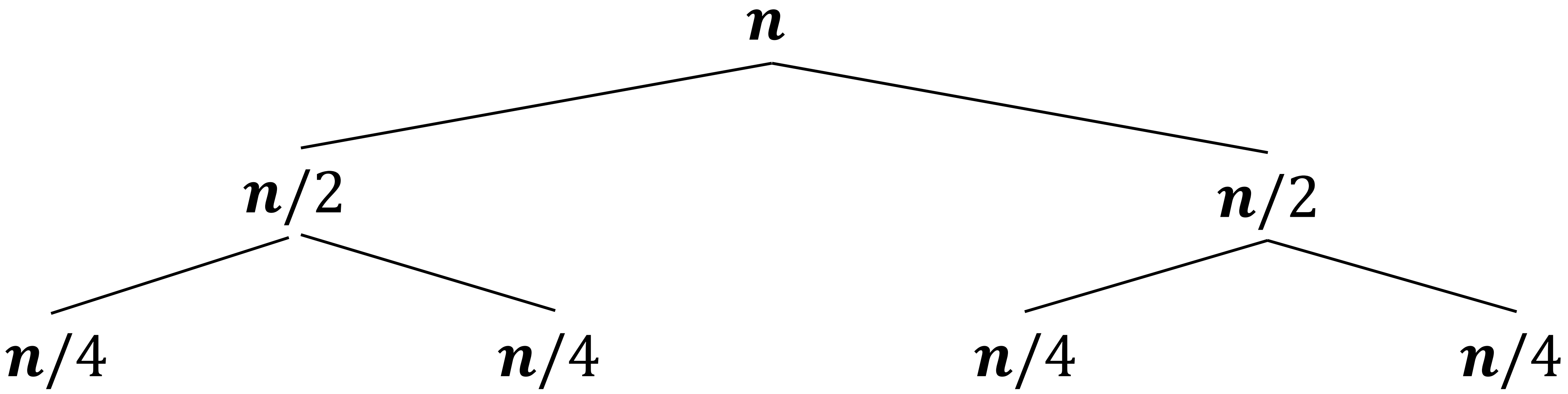
Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



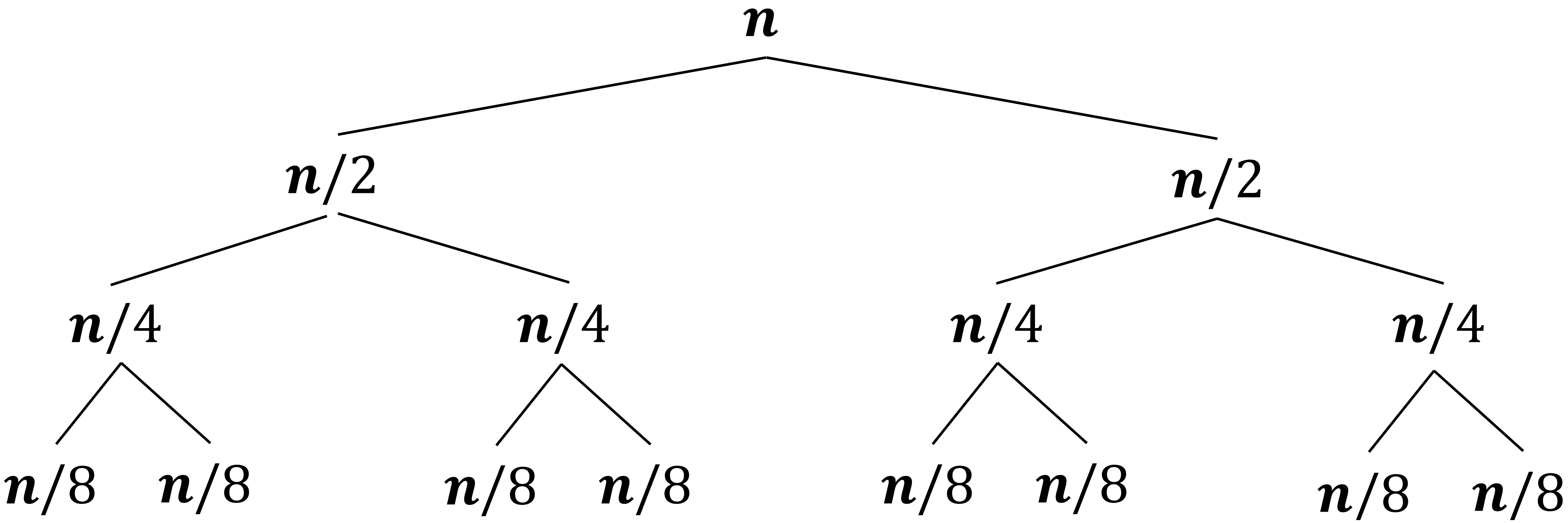
Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



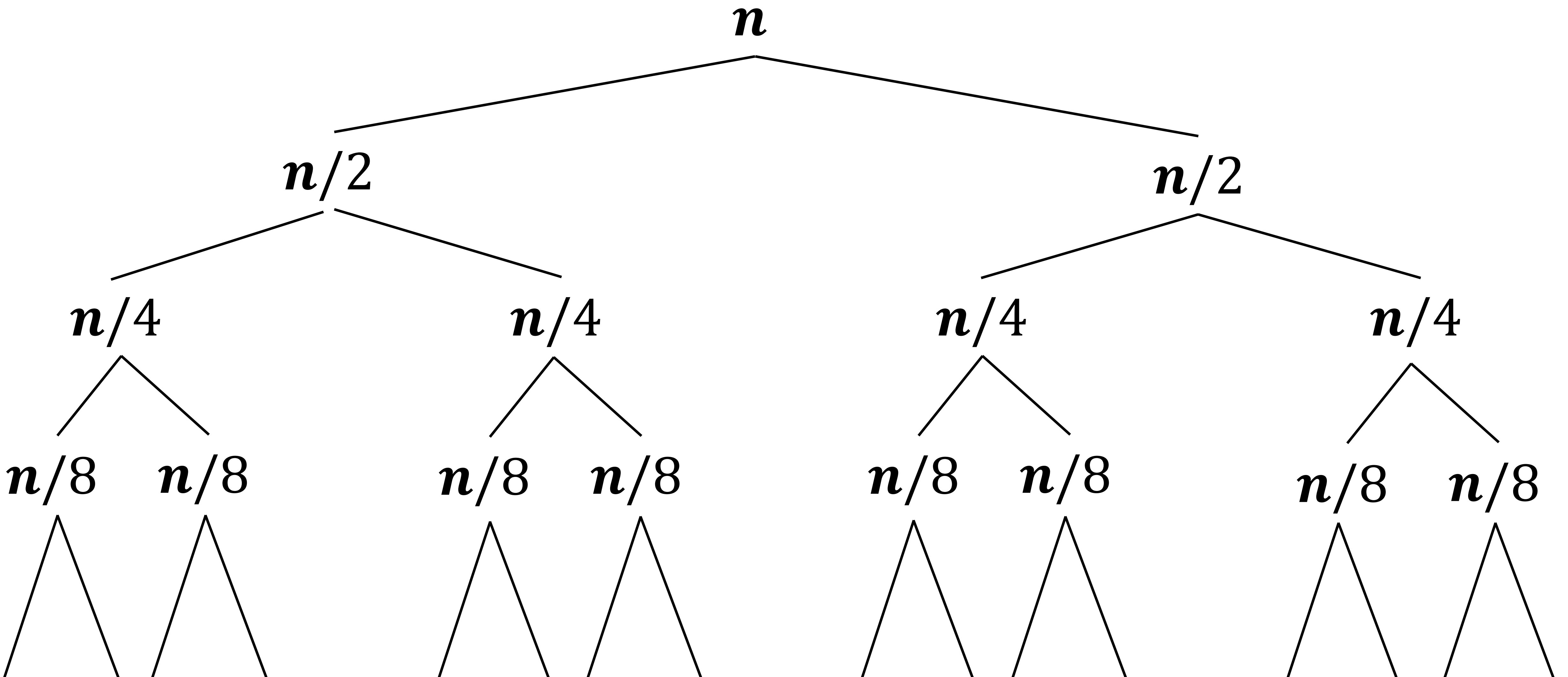
Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



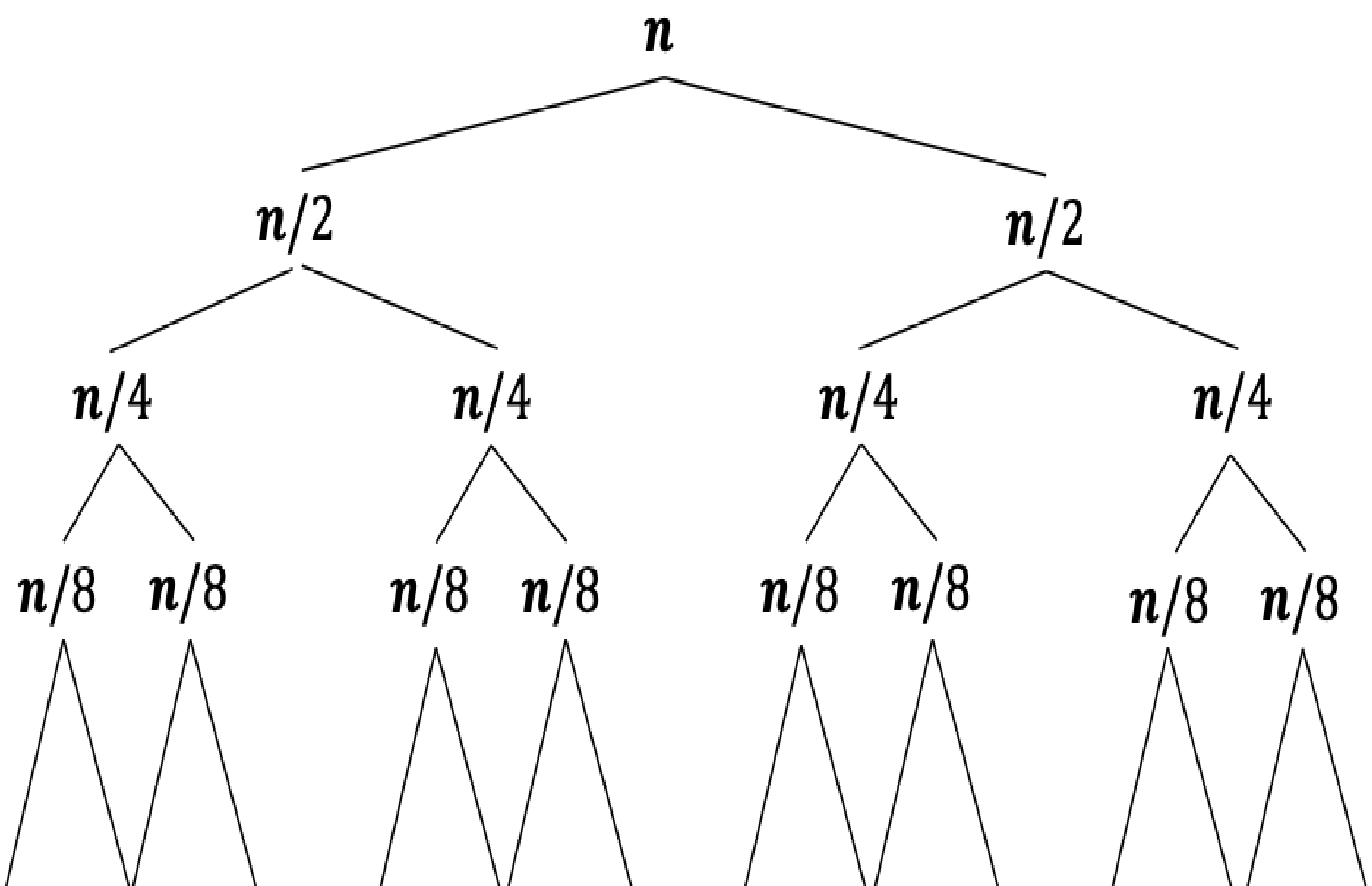
Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



Unrolling a few levels

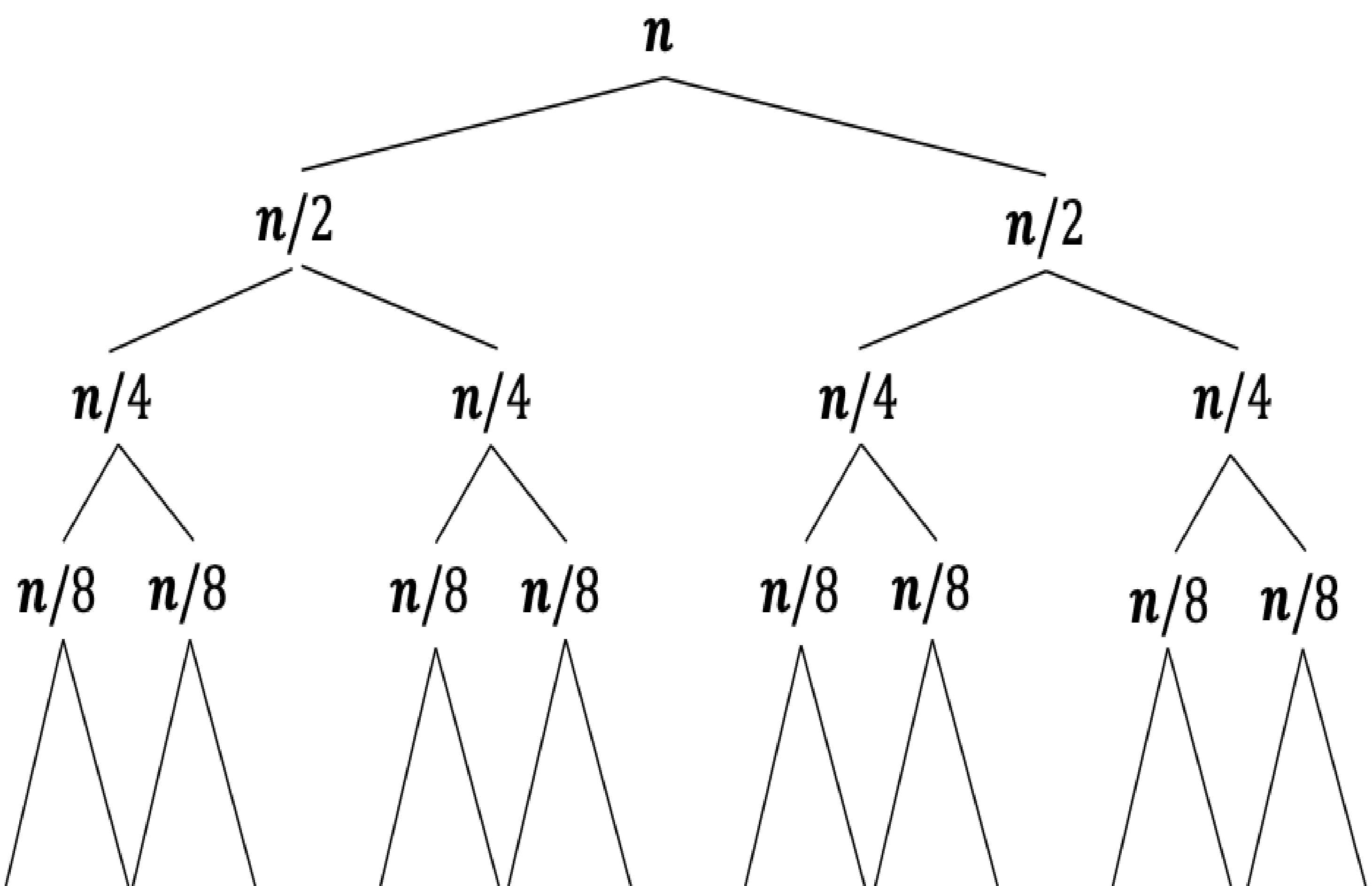
$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



Level	Problem Size	Total Work
0	?	?
1	?	?
2	?	?
3	?	?

Unrolling a few levels

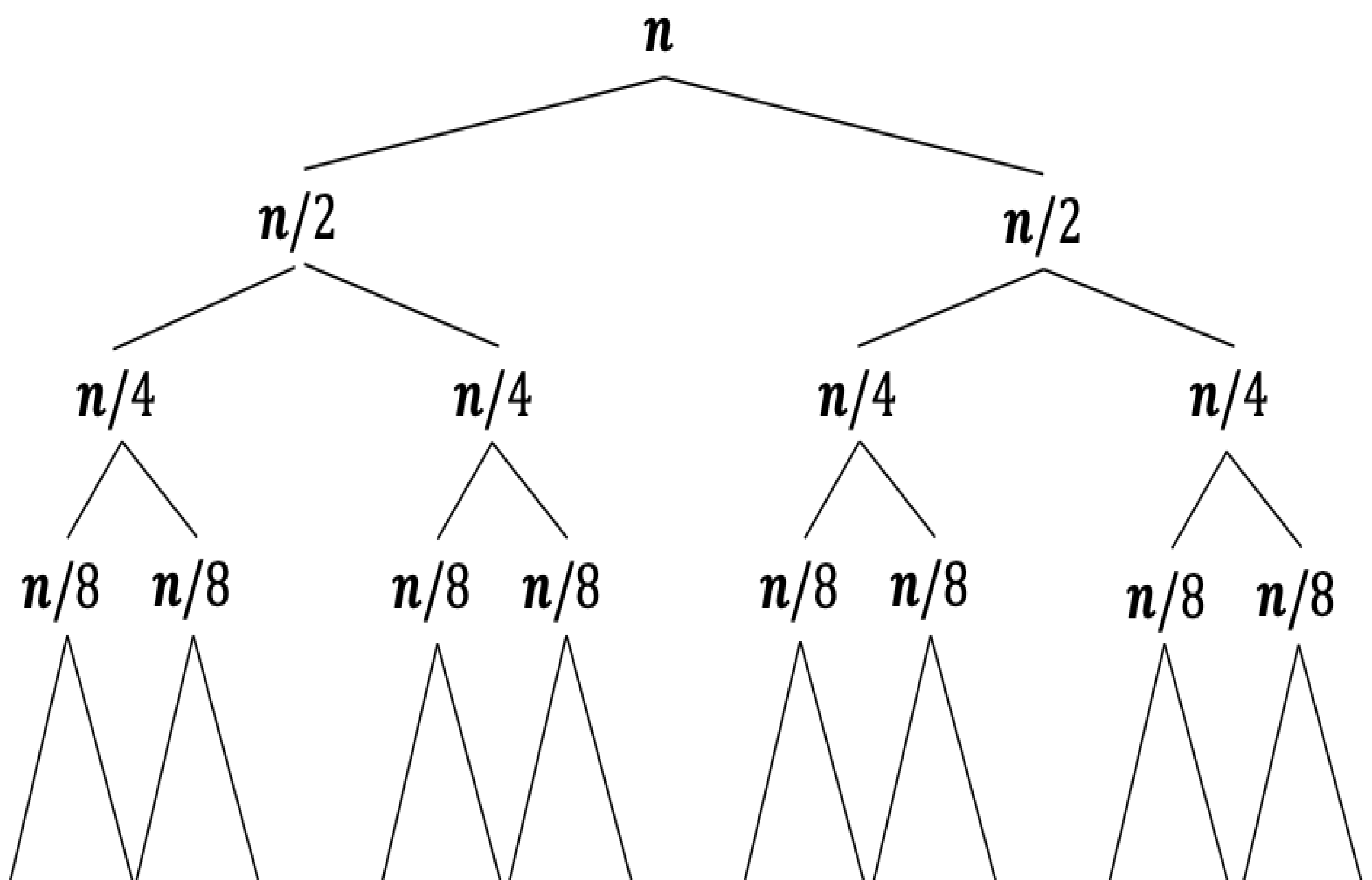
$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



Level	Problem Size	Total Work
0	n	?
1	n/2	?
2	n/4	?
3	n/8	?

Unrolling a few levels

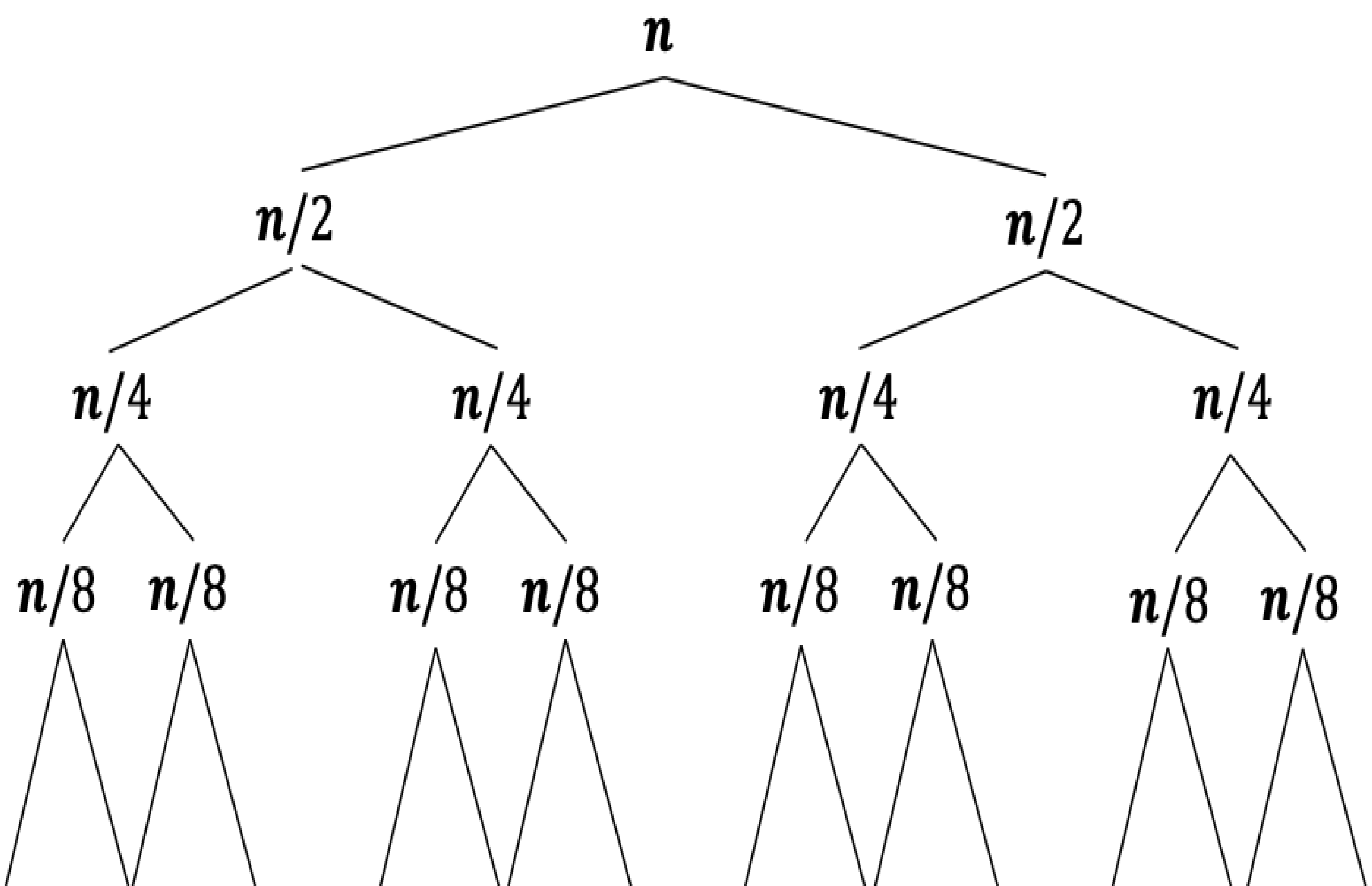
$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



Level	Problem Size	Total Work
0	n	c'n
1	n/2	2*(c'n/2)
2	n/4	4*(c'n/4)
3	n/8	8*(c'n/8)

Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$



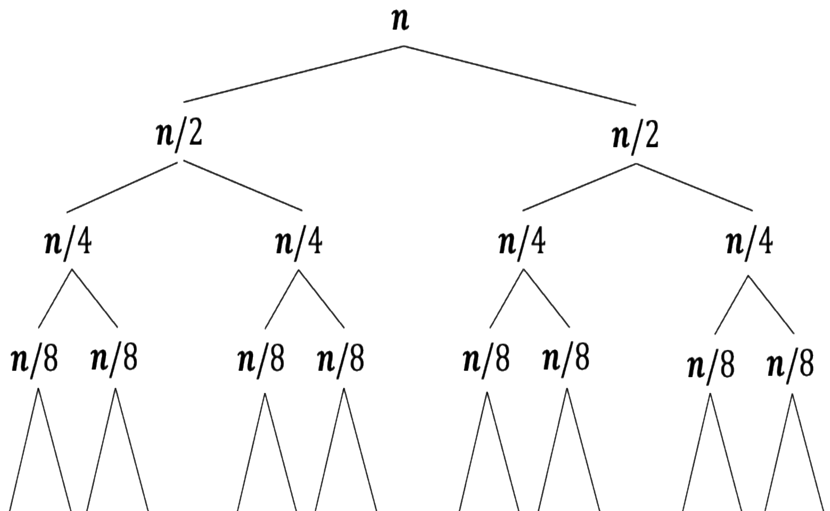
Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n

Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$

Q: What will be the problem size and total work done at level i?

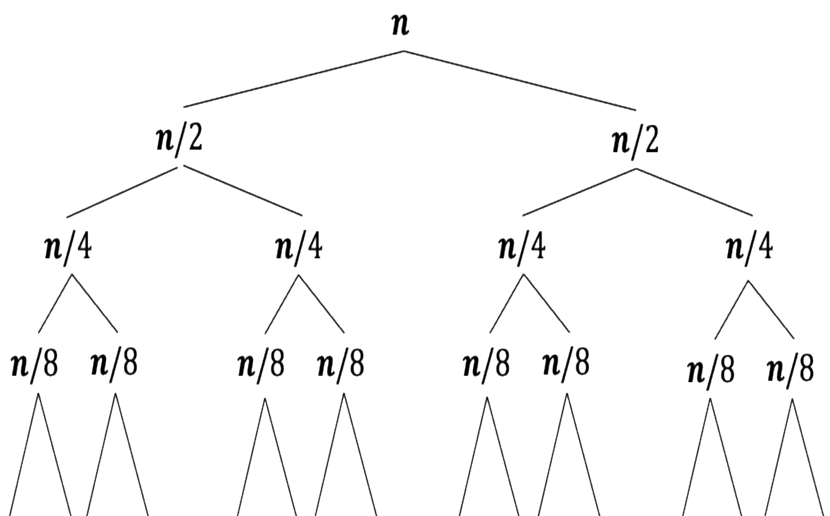
Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n



Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

A: At level i, the problem size (if not the base case) will be n/2^i. The total work done will be c'n.



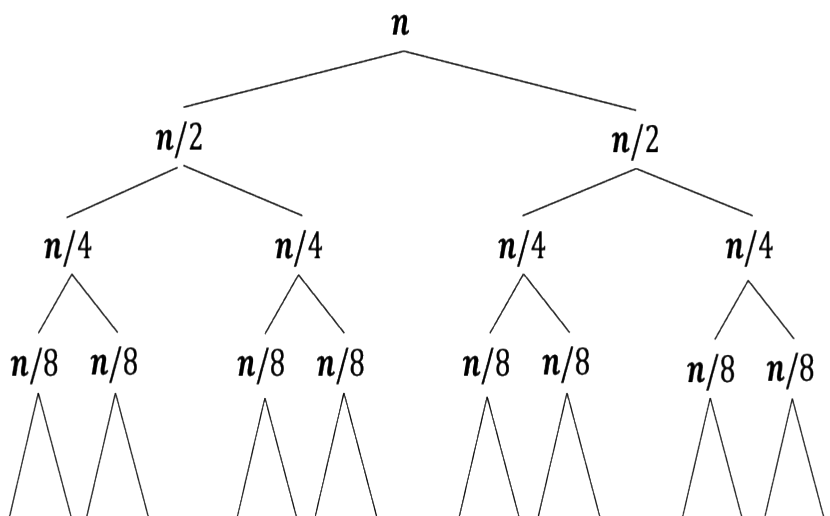
Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n

Unrolling a few levels

$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

Q: How many levels until base case?

Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n

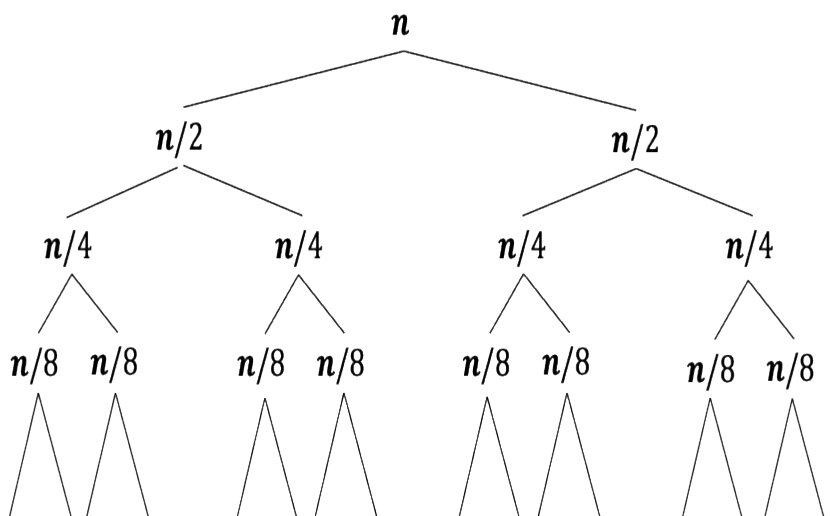


Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

A: After O(log(n)) levels, the problem size will be at most c and we can do the base case.

Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n

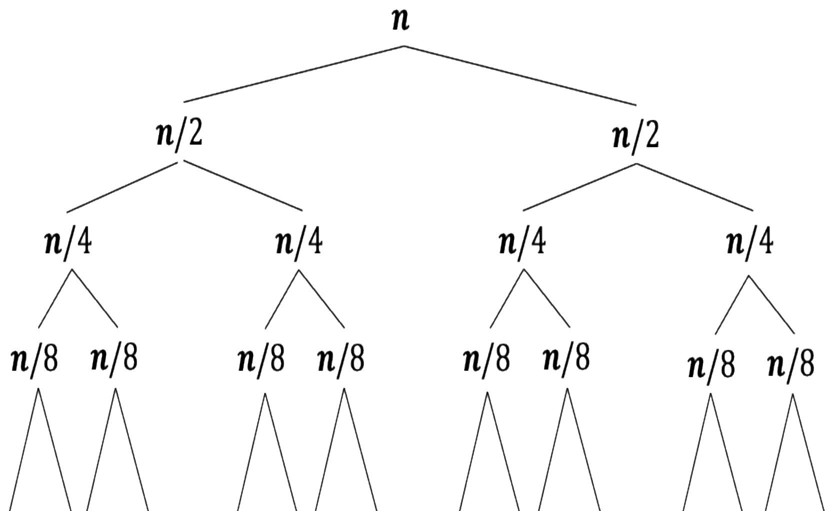


Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

Q: How much total work done over all levels?

Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n

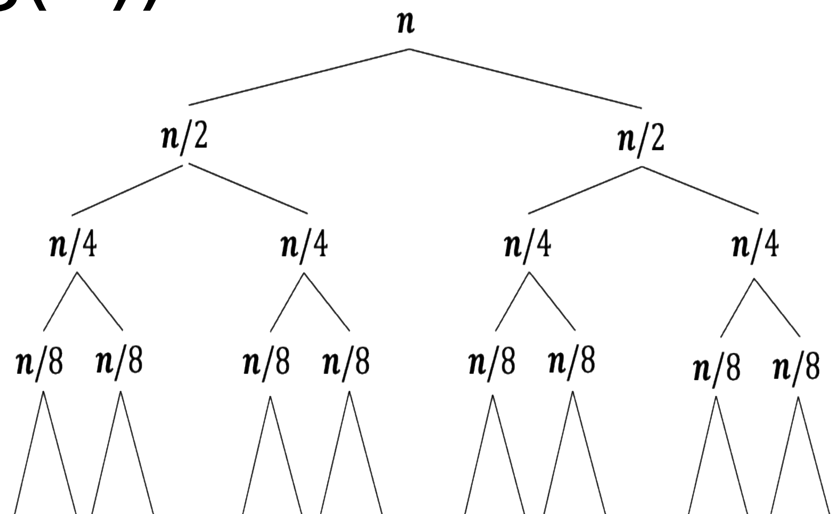


Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

A: The work done at each level is c’n and the total number of levels is O(log(n)).
Hence, O(n log(n)).

Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n

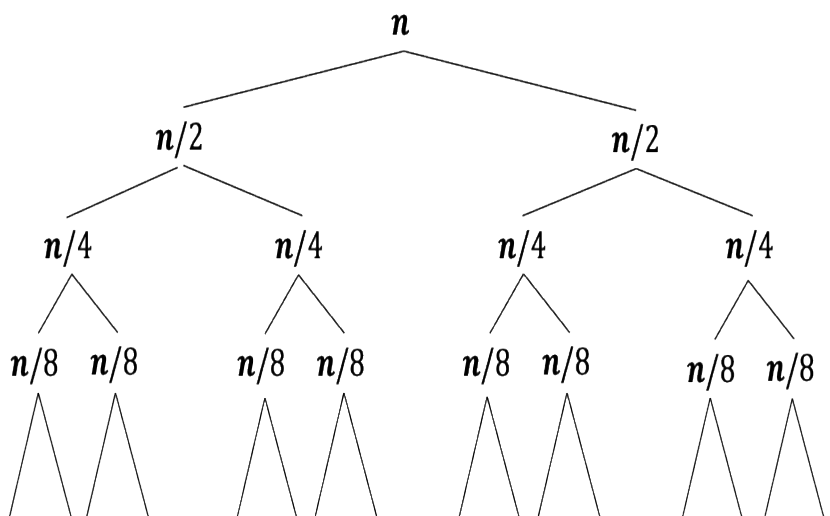


Unrolling a few levels

$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$

Work Done is the sum of work done at each level and we "showed" that the work done at each level is c'n and there are at most O(log(n)) levels.

Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n



Guess & Check

$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$

- You might have guessed T(n) = O(nlog(n)) because you've been told that before or because you recognize the recurrence.
- How could you directly prove T(n) = c''nlog(n) if you suspected it was true?

Guess & Check

$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$

- You might have guessed T(n) = O(nlog(n)) because you've been told that before or because you recognize the recurrence.
- How could you directly prove T(n) = $c' n log_2(n)$ if you suspected it was true?
  - You could use induction to show that this solves the recurrence.

Guess & Check

$$\boldsymbol{T(n)} \leq \begin{cases} \mathrm{c} & \mathrm{n} \leq 2 \\ 2\mathrm{T(n/2)} + \mathrm{c'n} & \mathrm{o.w.} \end{cases}$$

- **Base Case:**
  - We can sort a list of size at most 2 in constant time, $T(1) \leq T(2) \leq c$ for some constant c as desired.

Guess & Check

$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$

- **Base Case:**
  - We can sort a list of size at most 2 in constant time, $T(1) \leq T(2) \leq c$ for some constant c as desired.
- **IH:**
  - Suppose we know that $T(m) \leq c' m log_2(\mathrm{m})$ for some all $\mathrm{m} < \mathrm{n}$.

Guess & Check

$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$

- **Base Case:**
  - We can sort a list of size at most 2 in constant time, $T(1) \leq T(2) \leq c$ for some constant c as desired.
- **IH:**
  - Suppose we know that $T(m) \leq c' m log_2(m)$ for some all $m < n$.
- **Inductive Case:**
  - Since this is not the base case and n/2 < m, $T(n) \leq 2 * c''(n/2) log_2(n/2) + c'(n/2)$

Guess & Check

$$\boldsymbol{T(n)} \le \begin{cases} \mathrm{c} & \mathrm{n} \le 2 \\ 2\mathrm{T(n/2)} + \mathrm{c'n} & \mathrm{o.w.} \end{cases}$$

- **Inductive Case:**
  - Since this is not the base case and n/2 < m,

$$T(n) \le 2 * c'(n/2)log_2(n/2) + c'\mathrm{n}$$
$$\le c'\mathrm{n}log_2(n/2) + c'n$$
$$\le c'\mathrm{n}\,(log_2(n) - 1) + c'n$$
$$\le c'\mathrm{n}\,log_2(n) - \mathrm{c'n} + c'n$$
$$\le c'\mathrm{n}\,log_2(n)$$

- This concludes the proof.

Partial Guess

$$T(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$

- You might not know the coefficients or bases:

$$T(n) \leq 2 * k(n/2) log_b(n/2) + c'n$$

$$\leq k\, n\, log_b(n/2) + c'n$$

Seems like b = 2

$$\leq kn\,(log_2(n) - 1) + c'n$$

$$\leq kn\, log_2(n) - kn + c'n$$

Seems like k = c'

$$\leq c'n\, log_2(n)$$

Q: What happens?

$$T_1(n) \leq \begin{cases} c & n \leq 2 \\ qT(n/2) + c'n & \text{o.w.} \end{cases}$$

$$T_2(n) \leq \begin{cases} c & n \leq 2 \\ 2T(n/2) + c'n\text{^}2 & \text{o.w.} \end{cases}$$

$$T_3(n) \leq \begin{cases} c & n \leq 2000 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$