CSE 331: Algorithms & Complexity

"Solving recurrence relations"

Prof. Charlie Anne Carlson (She/Her)

Lecture 23

Monday October 27th, 2025

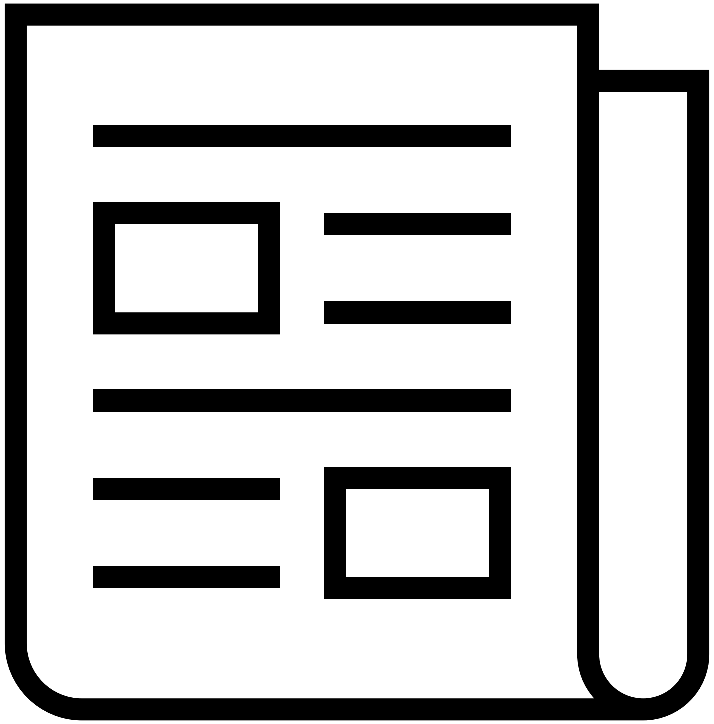
University at Buffalo

Schedule

1. Course Updates
2. Mergesort
3. Recurrences

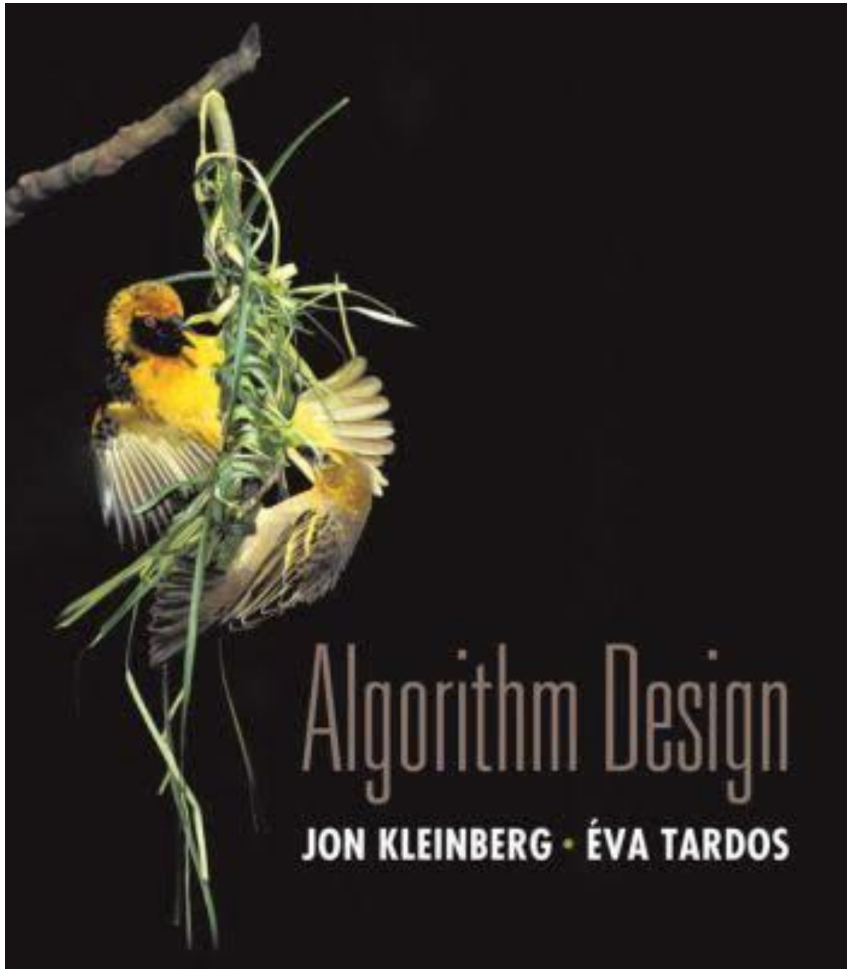
Course Updates

- Midterm Out
- Post Midterm Grades <-piazza
- HW 5 Due Tomorrow
- HW 6 Out Tomorrow
- Group Project
  - First Problems Oct 31st



Reading

- You should have read:
  - Finished KT 5.1
  - Finished KT 5.2
  - Started 5.3
- Before Next Class:
  - Finish KT 5.3
  - Start KT 5.5



Course Update

Check Piazza!

Course Updates

Updated 33 minutes ago by Charlie Anne Carlson

Hello All,

Grade Evals:

I'm still working on getting a copy of your grades into UBLearns. For now, you can calculate your grade using the raw grades given on autolab. Here is a little formula that I would suggest using to compute your "benchmark grade":

Benchmark Grade = MG*(25/55) + HWG*(27/55) + QG*(3/55)

where

- MG = (Midterm Grade)/100
- HWG = (P1A+P1B+P2A+P2B+P3)/100
  - P1A = Max Two Scores for Problem 1A of HW1, HW2, or HW3.
  - P1B = Max Two Scores for Problem 1B of HW1, HW2, or HW3.
  - P2A = Max Two Scores for Problem 2A of HW1, HW2, or HW3.
  - P2B = Max Two Scores for Problem 2B of HW1, HW2, or HW3.
  - P3 = Max Two Scores for Problem 3 of HW1, HW2, or HW3.
- Q = (Quiz 1 Grade)/10

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

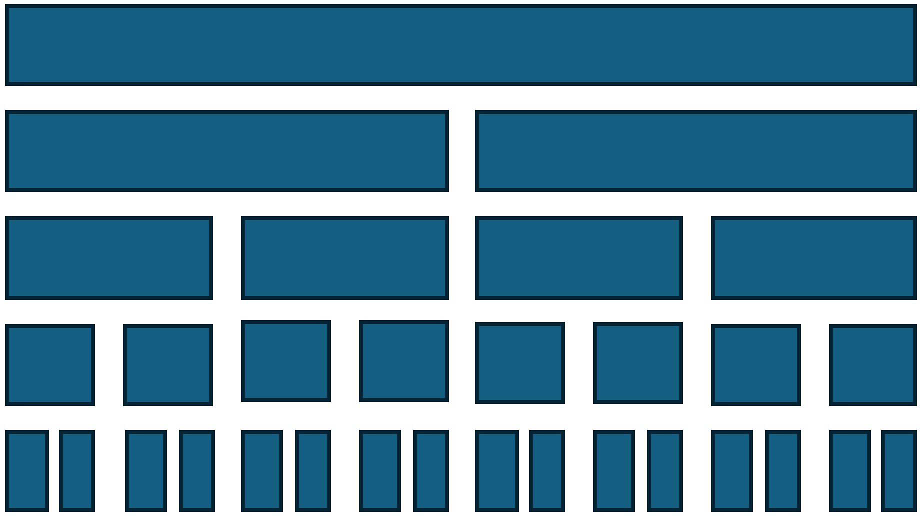
Sorting

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

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

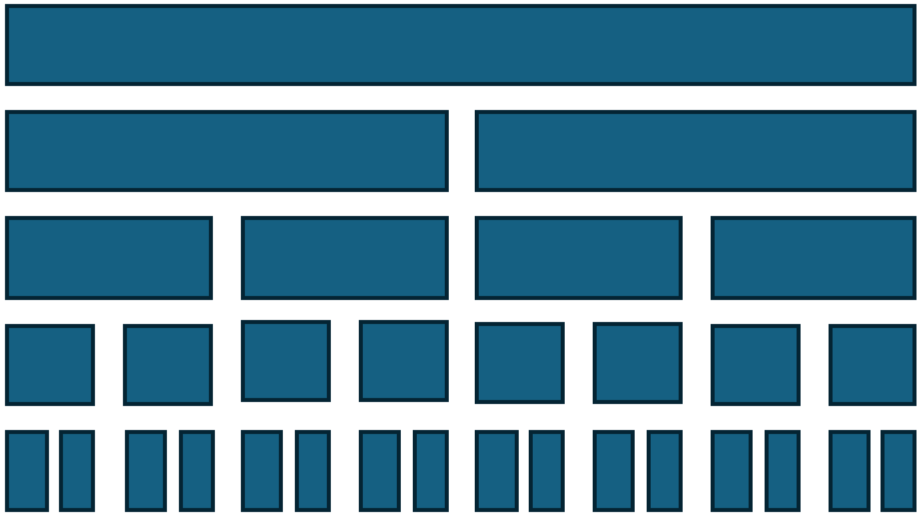
Mergesort

- **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.



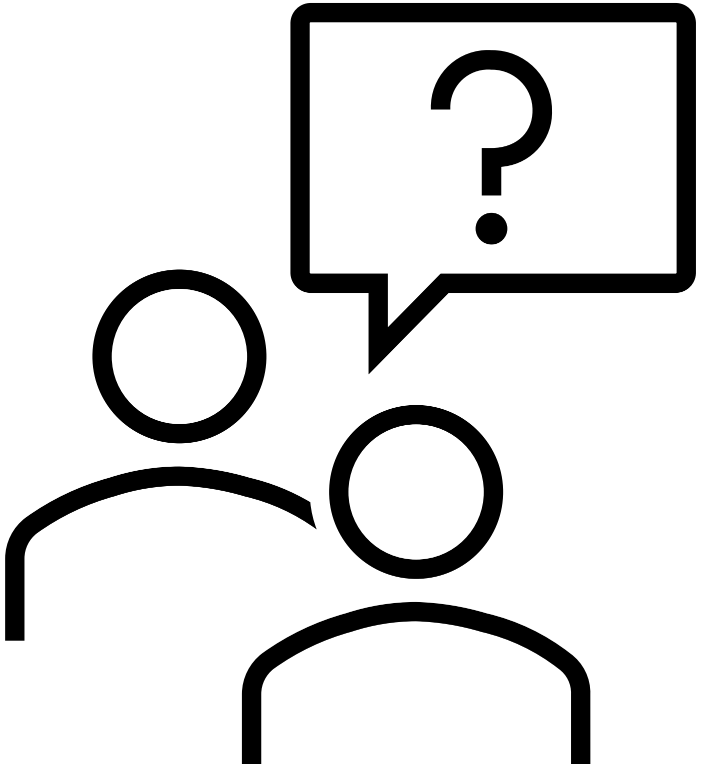
Mergesort

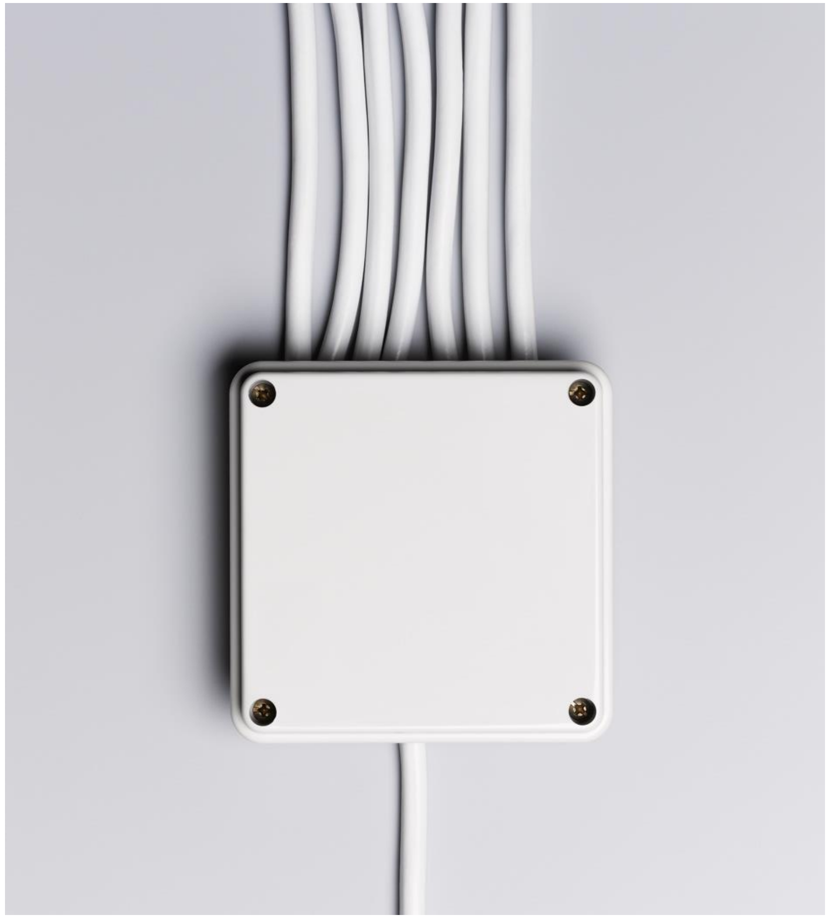
- **Base Case:** If array has length less than 2, brute force.
- **Divides**: Divides input into two pieces of equal size in linear time.
  - Assume even length for now.
- **Conquer**: Recursively calls mergesort on each piece.
- **Unite**: Merges the two sorted lists in linear time.



Sorting

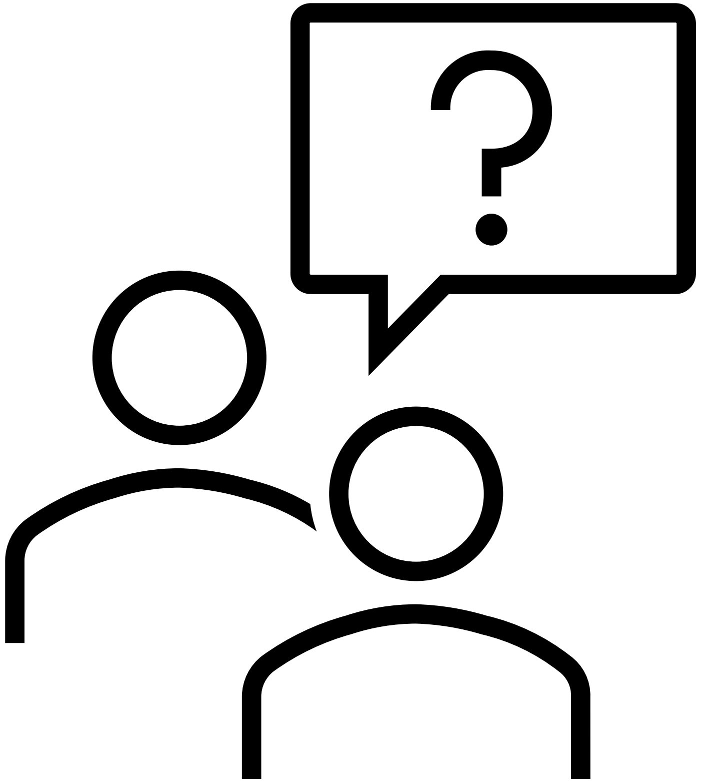
- **Problem**: Given two sorted lists A and B, find a sorted list of their union.





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

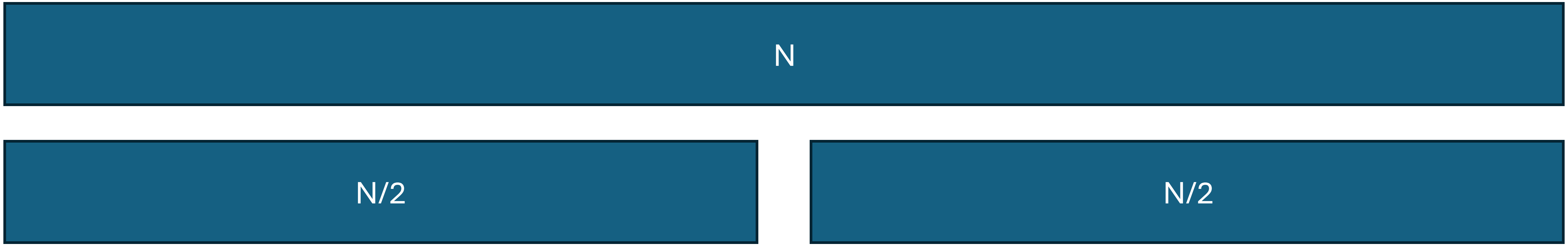


Mergesort Runtime?

- **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)**

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

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)**

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

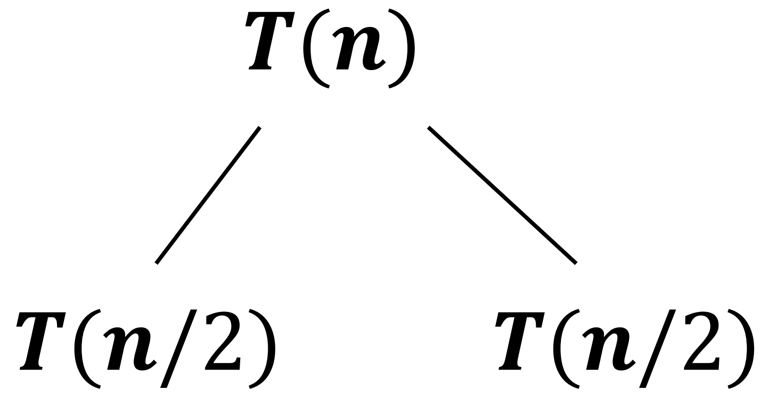
How do you solve a recurrence?

- **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!

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

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.



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

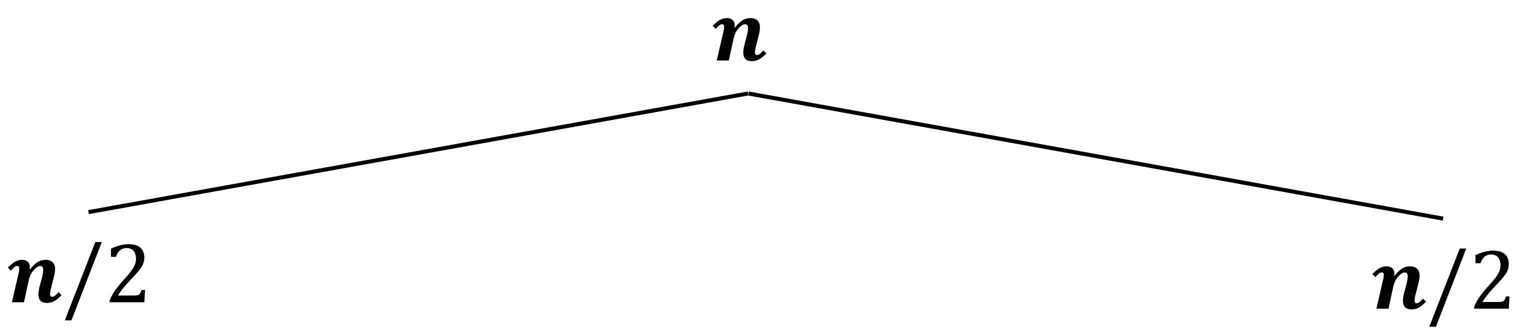
Unrolling a few levels

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

$\boldsymbol{n}$

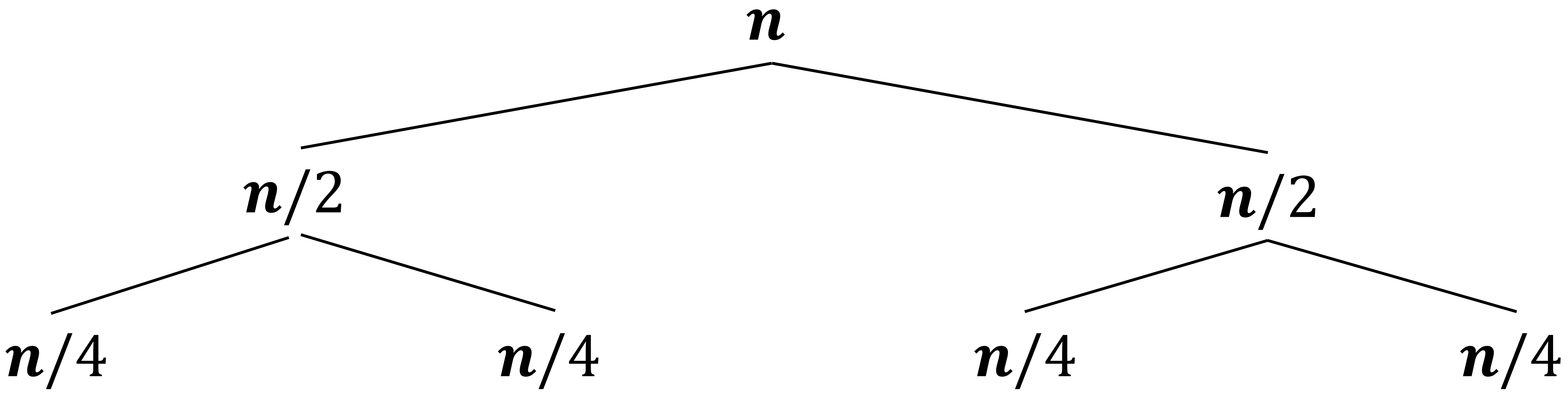
Unrolling a few levels

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



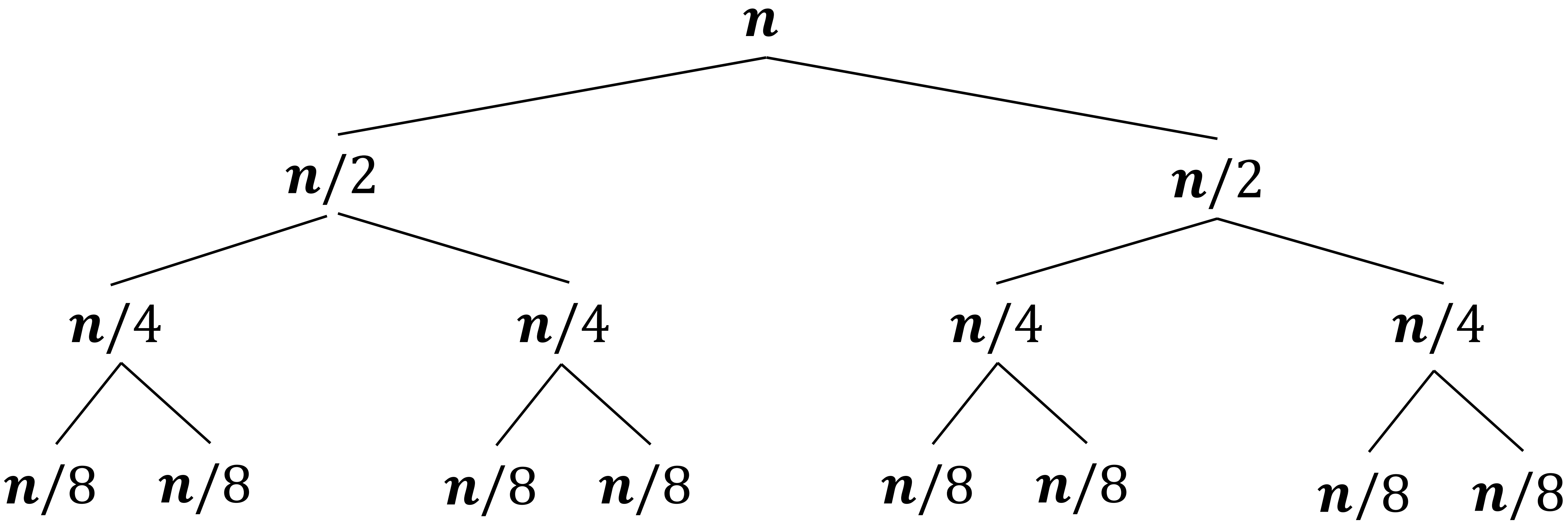
Unrolling a few levels

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



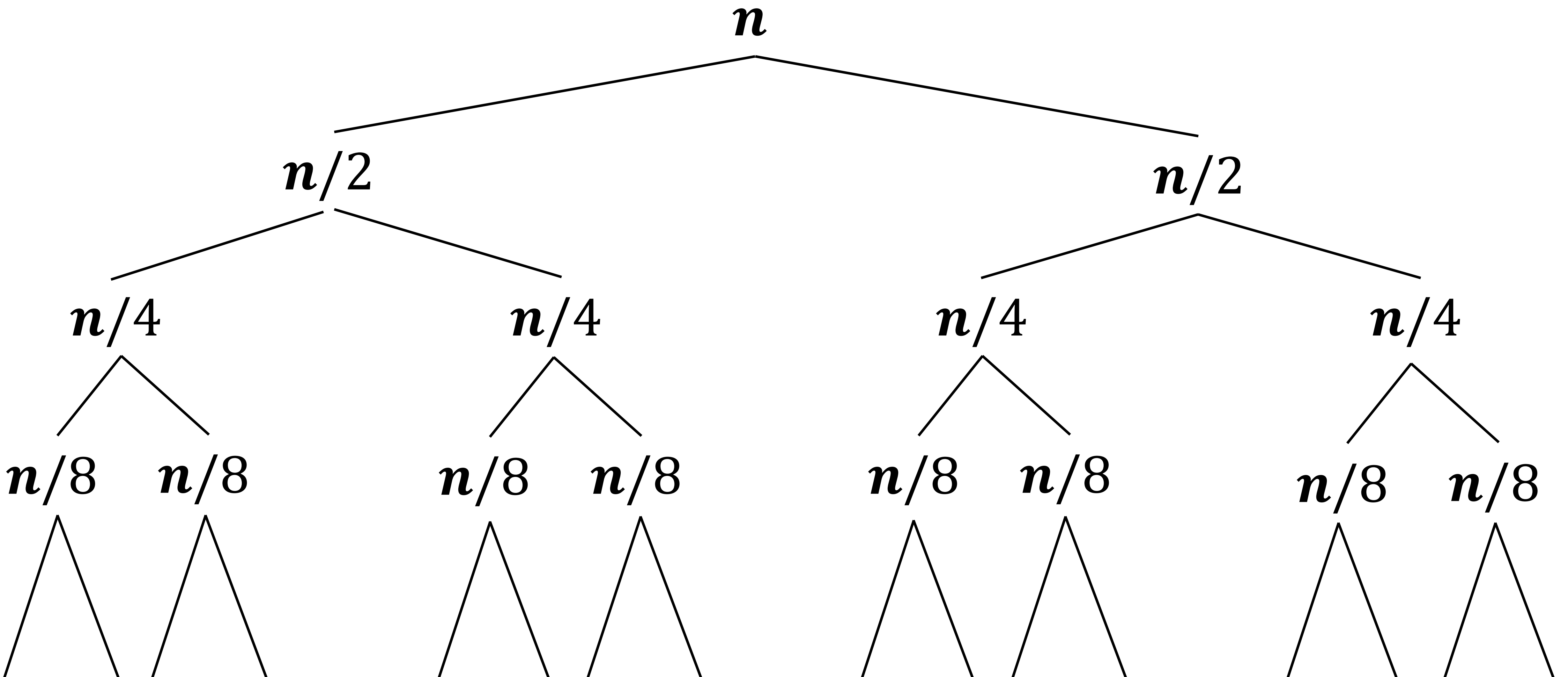
Unrolling a few levels

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



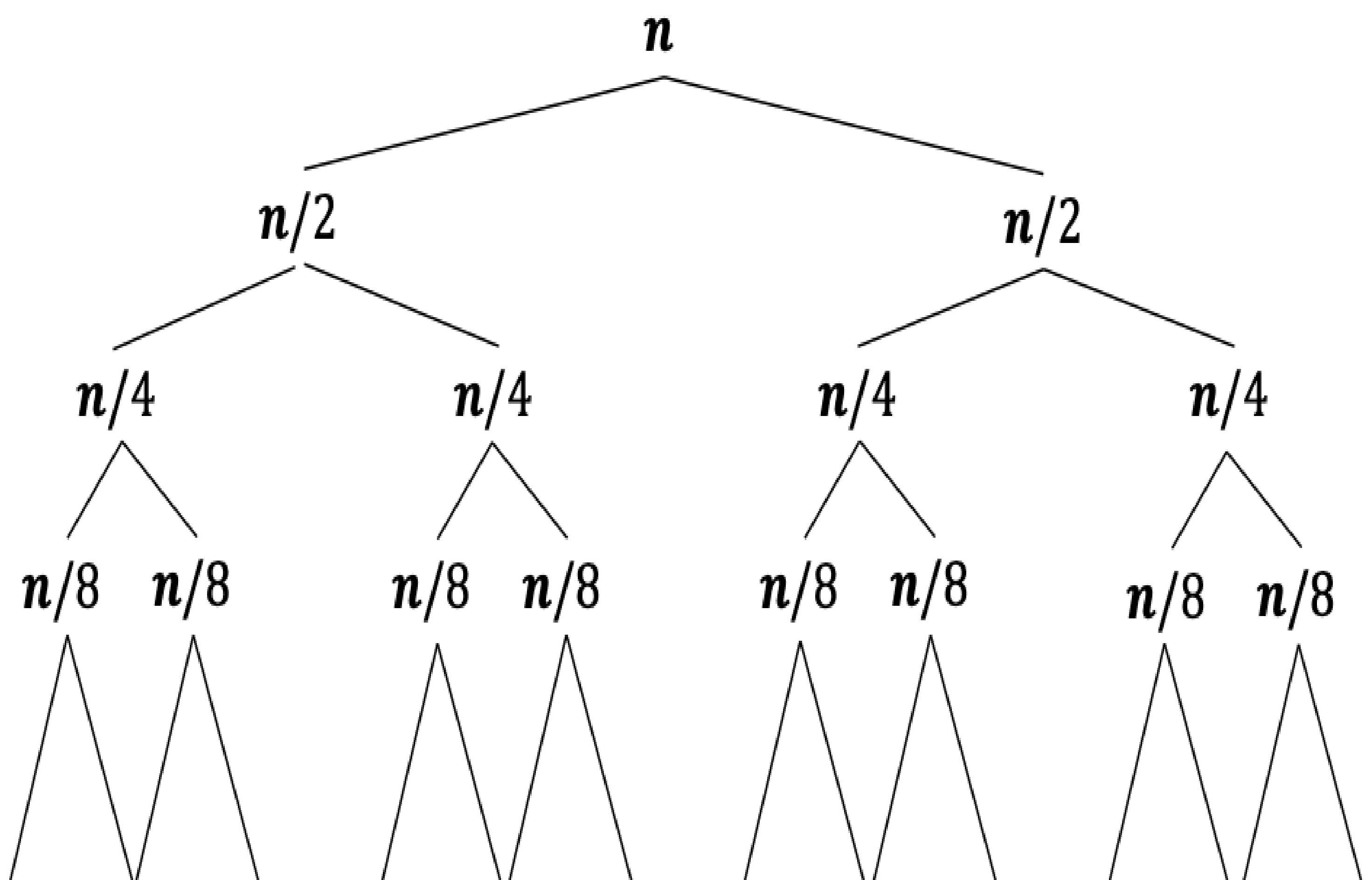
Unrolling a few levels

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



Unrolling a few levels

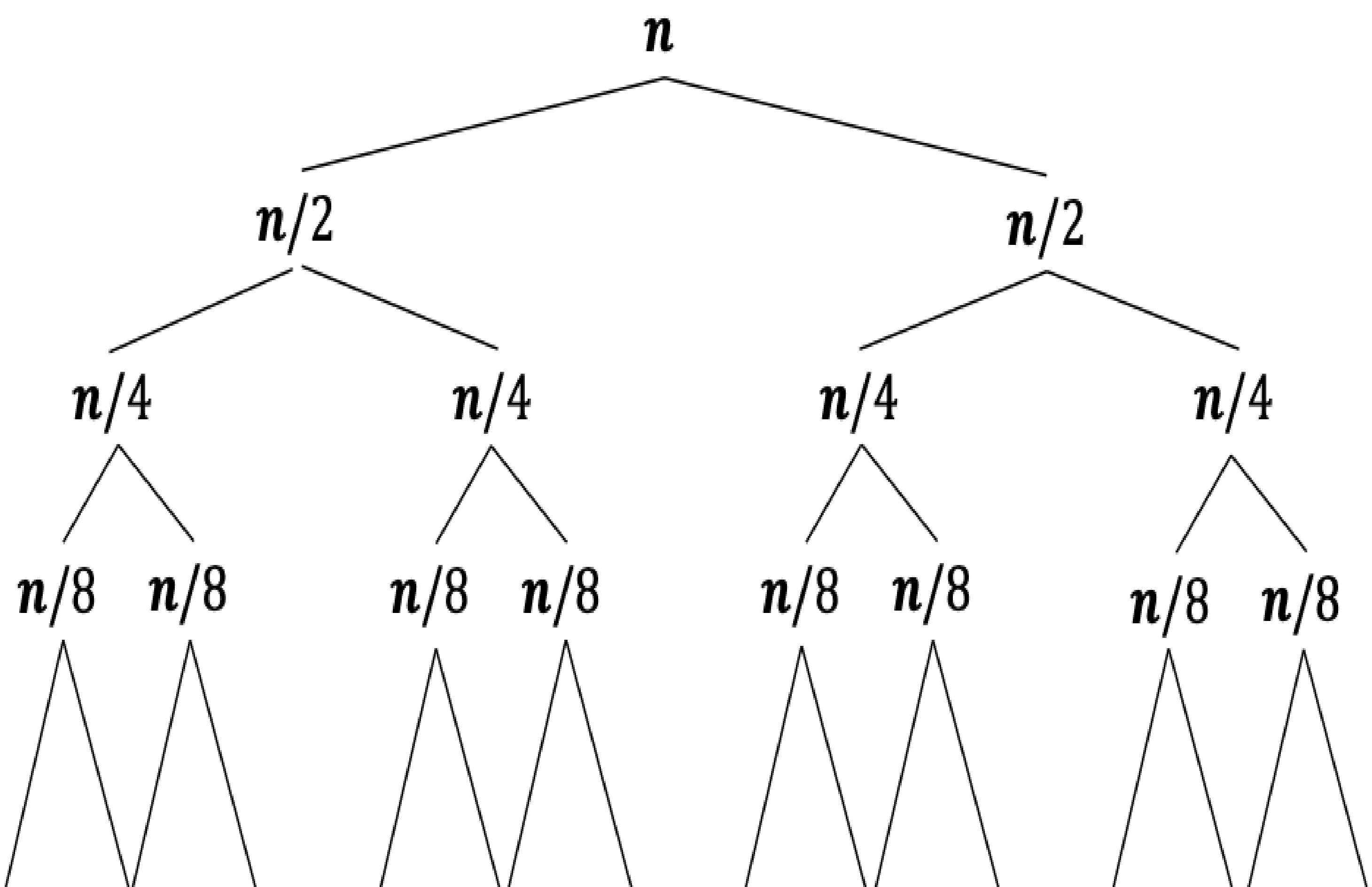
$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$



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

Unrolling a few levels

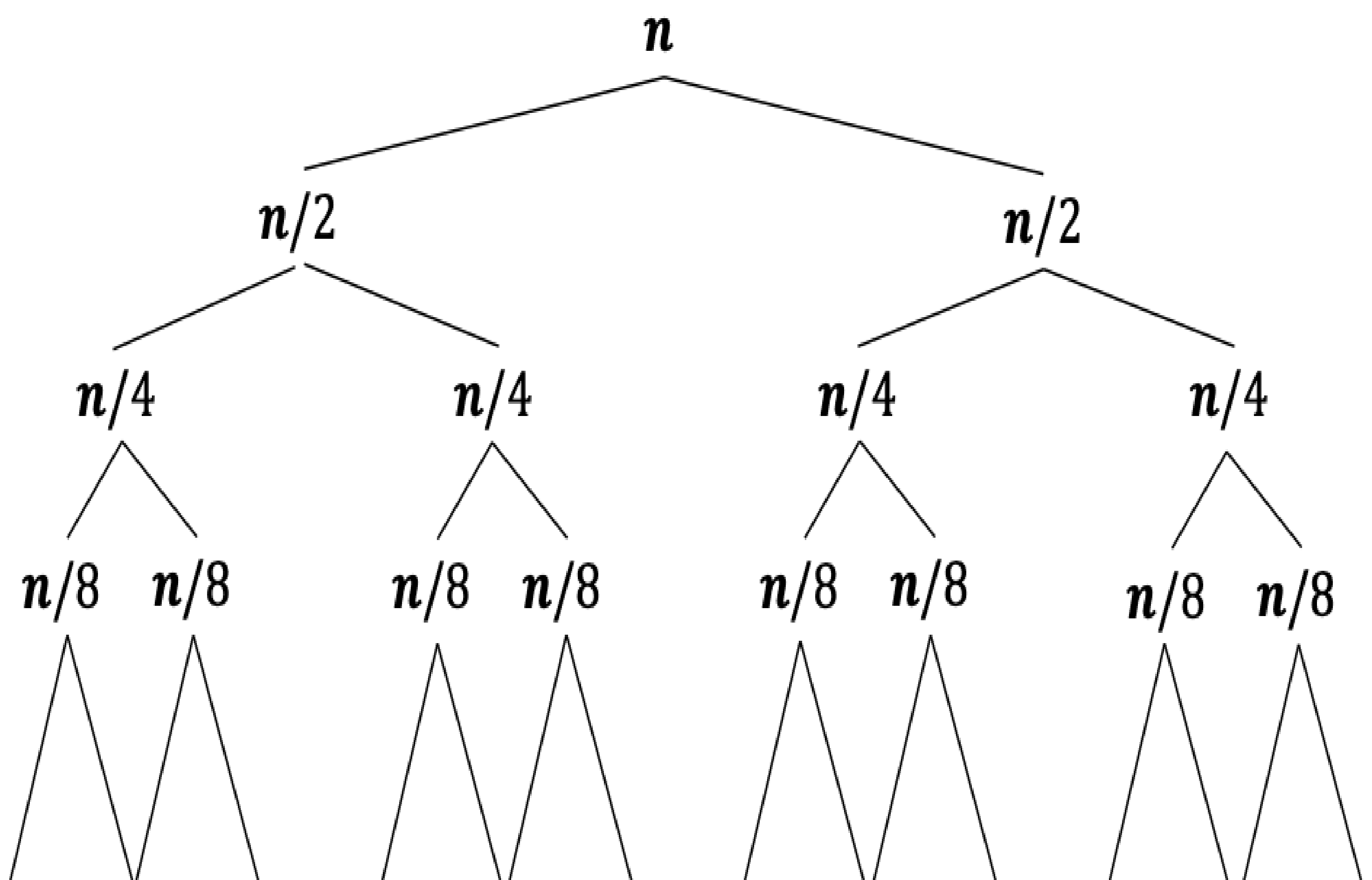
$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & o.w. \end{cases}$$



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

Unrolling a few levels

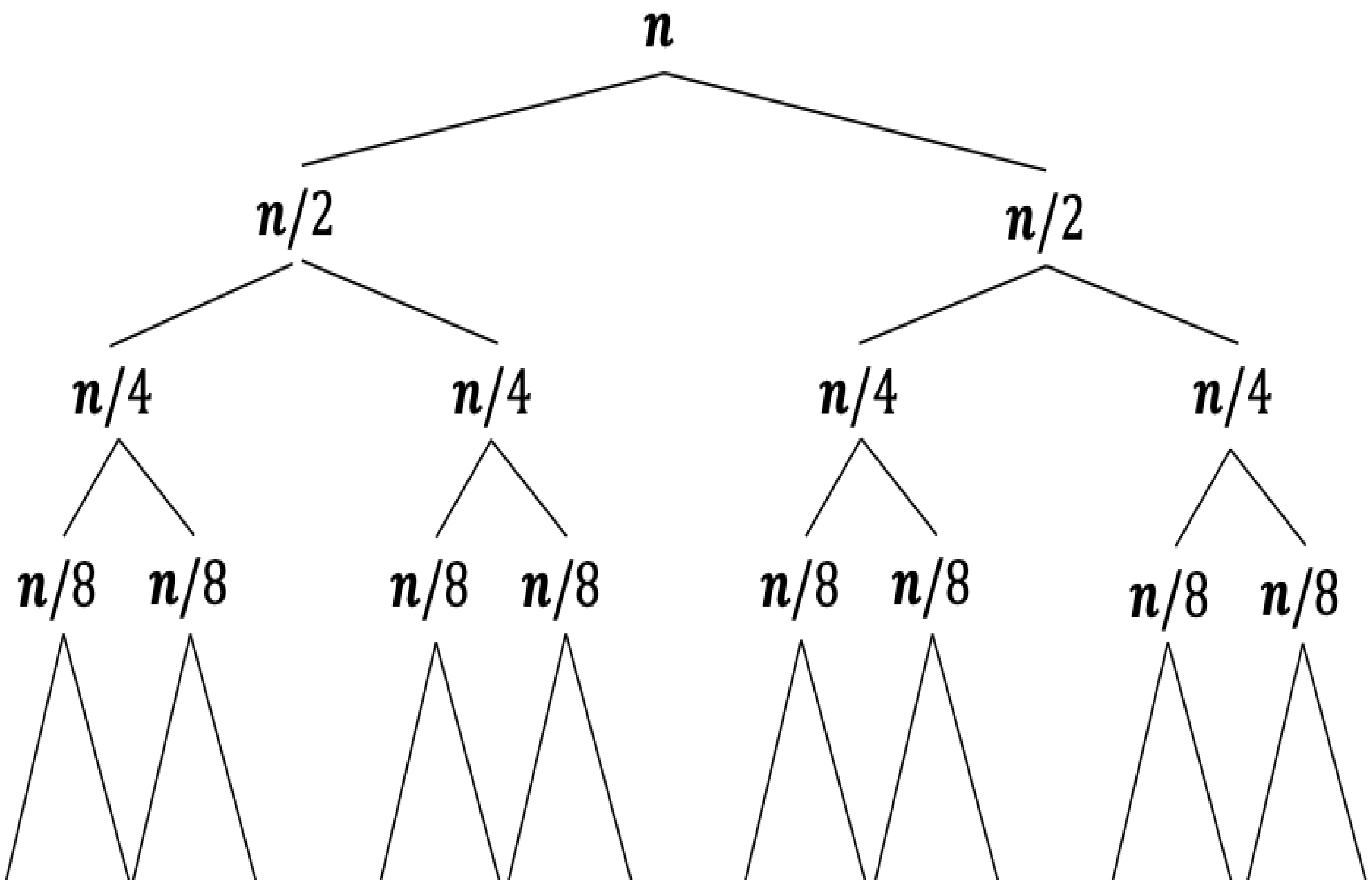
$$T(n) \le \begin{cases} c & n \le 2 \\ 2T(n/2) + c'n & \text{o.w.} \end{cases}$$



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

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

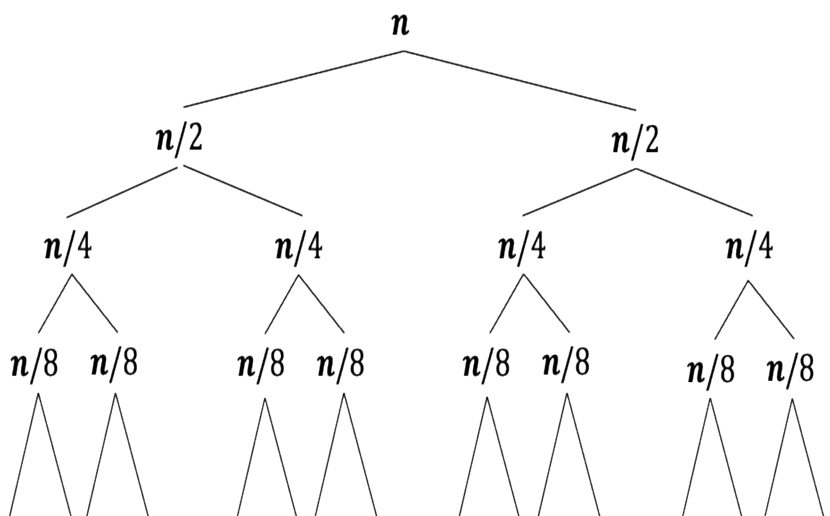


Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n

Unrolling a few levels

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

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

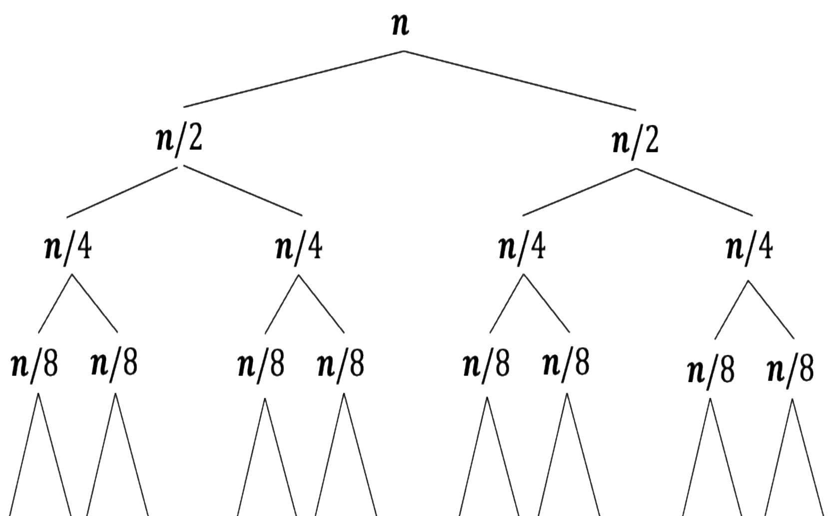


Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n

Unrolling a few levels

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

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.



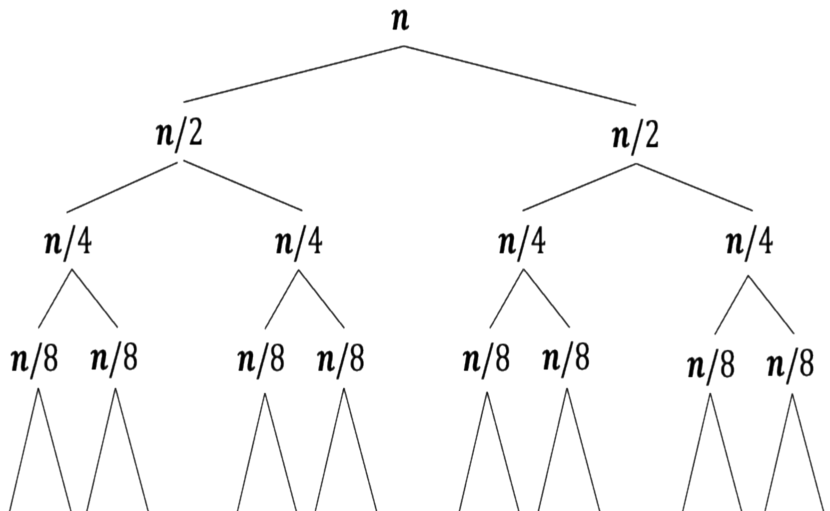
Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n

Unrolling a few levels

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

Q: How many levels until base case?

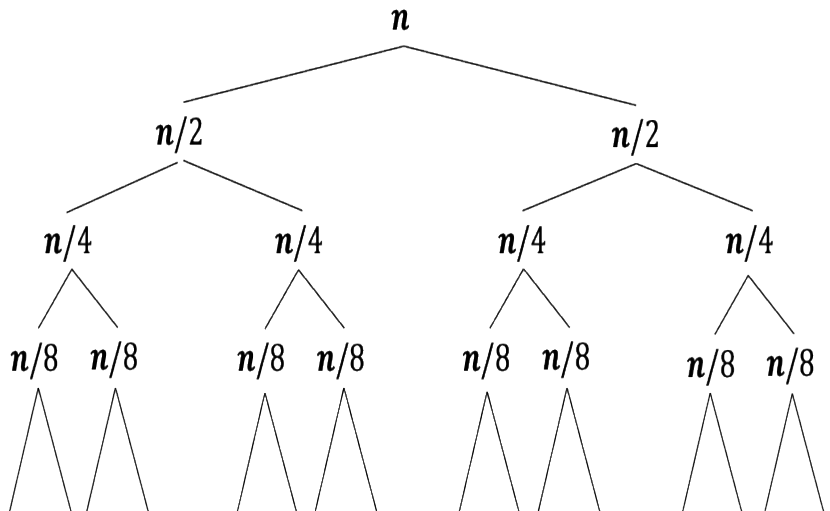
Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n



Unrolling a few levels

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

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



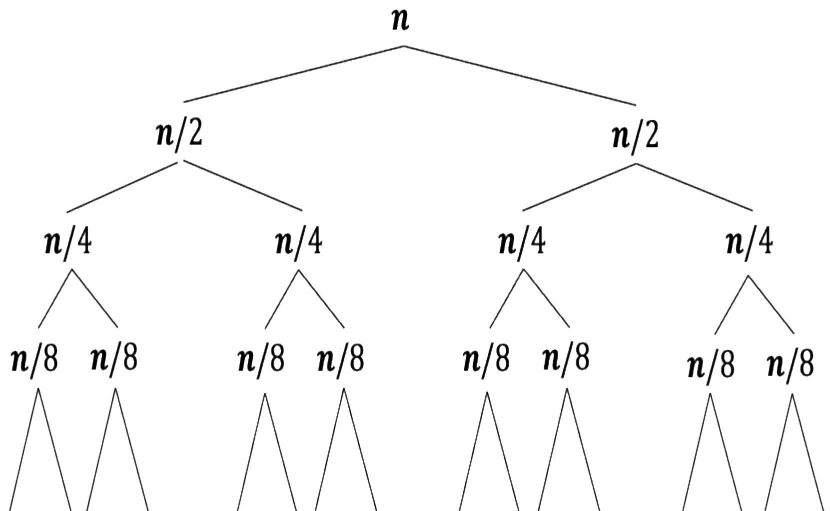
Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n

Unrolling a few levels

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

Q: How much total work done over all levels?

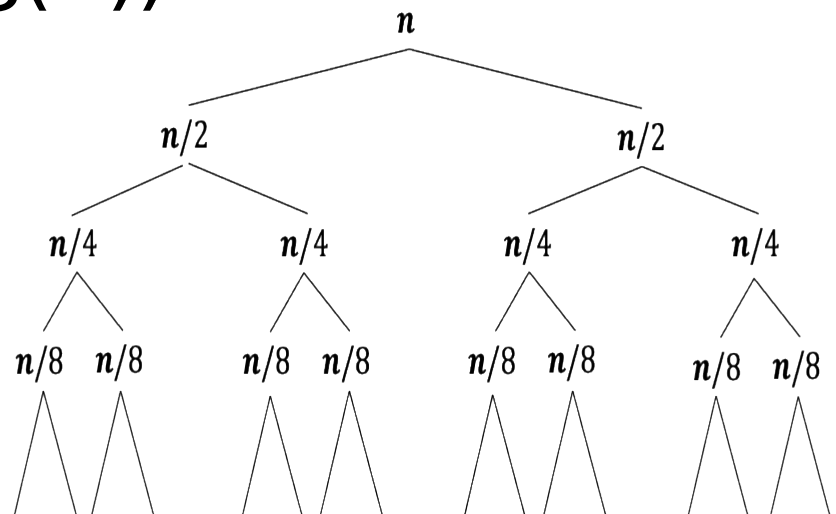
Level	Problem Size	Total Work
0	n	c'n
1	n/2	c'n
2	n/4	c'n
3	n/8	c'n



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)).

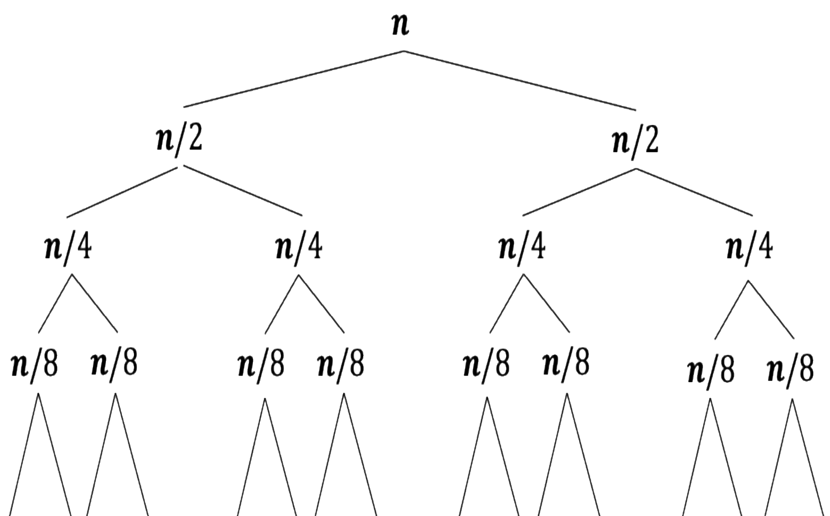


Level	Problem Size	Total Work
0	n	c’n
1	n/2	c’n
2	n/4	c’n
3	n/8	c’n

Unrolling a few levels

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

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

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

- 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

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

- 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) = $\mathrm{c'n}log_2(\mathrm{n})$ if you suspected it was true?
  - You could use induction to show that this solves the recurrence.

Guess & Check

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

- **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 $T(m) \leq c'mlog_2(m)$ for some all $m < n$.

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 $T(m) \leq c'mlog_2(m)$ 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

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

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

$$\begin{aligned} T(n) &\leq 2 * c'(n/2)log_2(n/2) + c'n \\ &\leq c'n log_2(n/2) + c'n \\ &\leq c'n\,(log_2(n) - 1) + c'n \\ &\leq c'n\,log_2(n) - c'n + c'n \\ &\leq c'n\,log_2(n) \end{aligned}$$

- This concludes the proof.

Partial Guess

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

- You might not know the coefficients or bases:

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

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

Seems like b = 2

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

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

Seems like k = c'

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

Q: What happens?

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

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

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