Discussion 7/14/2019

Recurrence Relation

- Recurrence relations are used to determine the running time of recursive programs – recurrence relations themselves are recursive.
- F(n) = F(n -1) + F(n-2) , F(1) =1 , F(0) = 1 Fibbanocci Sequence

Divide and Conquer

- Divide and Conquer is an algorithm strategy
  - Divide the input data into multiple pieces
  - Recurse into each piece
  - Join the answers together

Divide and Conquer

```
findMax(array):
    part1 = lower half of array
    part2 = upper half of array
    max1 = findMax(part1)
    max2 = findMax(part2)
    return max(max1,max2)
```

Figuring the Costs, Per Layer

- The total split cost **per layer** is:

$$numSplits * costPerSplit$$

- For simple binary algorithms like this one, at layer `L`:

$$numSplits = 2^L$$

$$costPerSplit = c_0 \frac{n}{2^L}$$

where c_0 is the constant on the split cost.

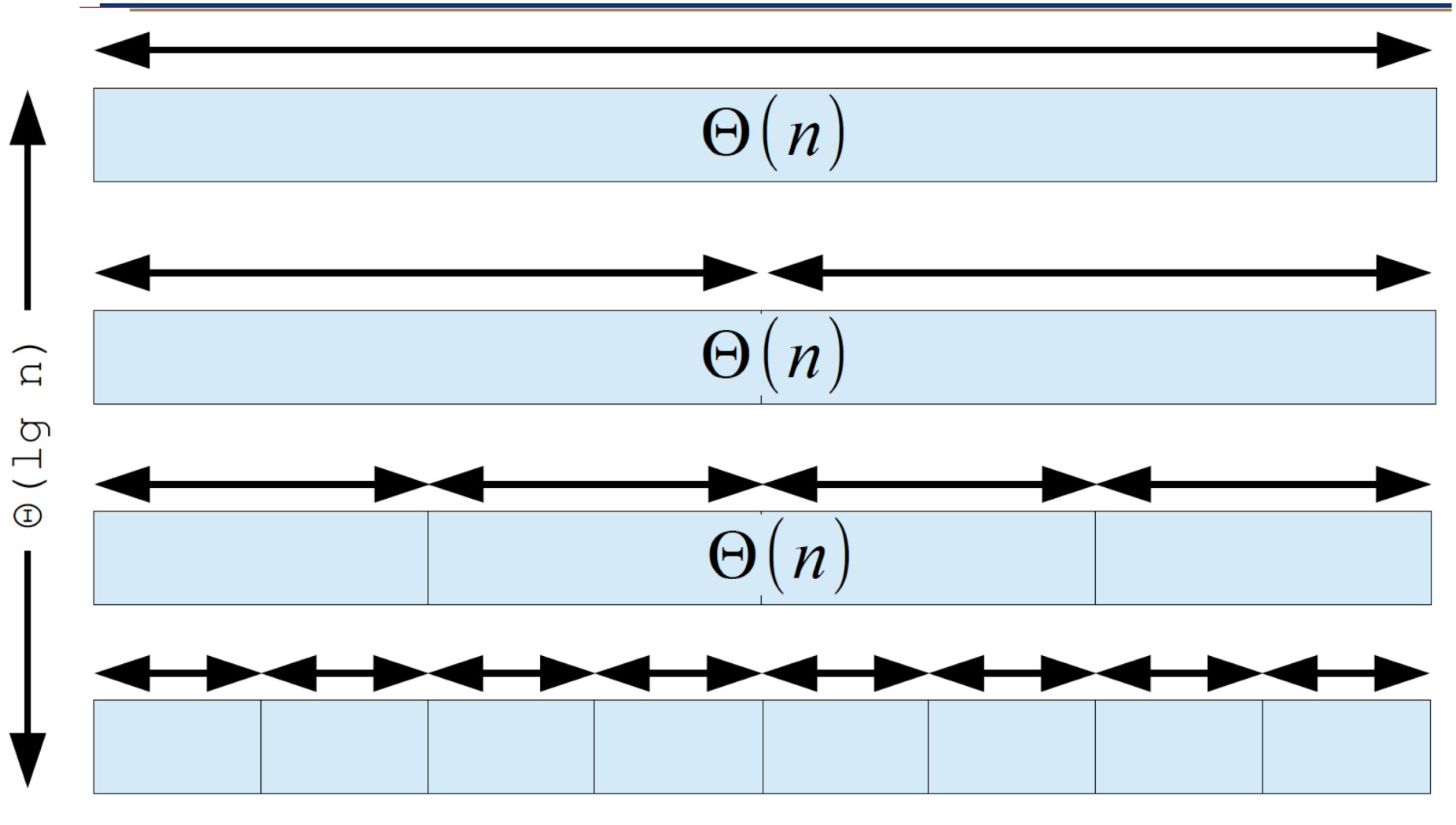
Figuring the Costs, Per Layer

- Thus, the total cost **per layer** is:

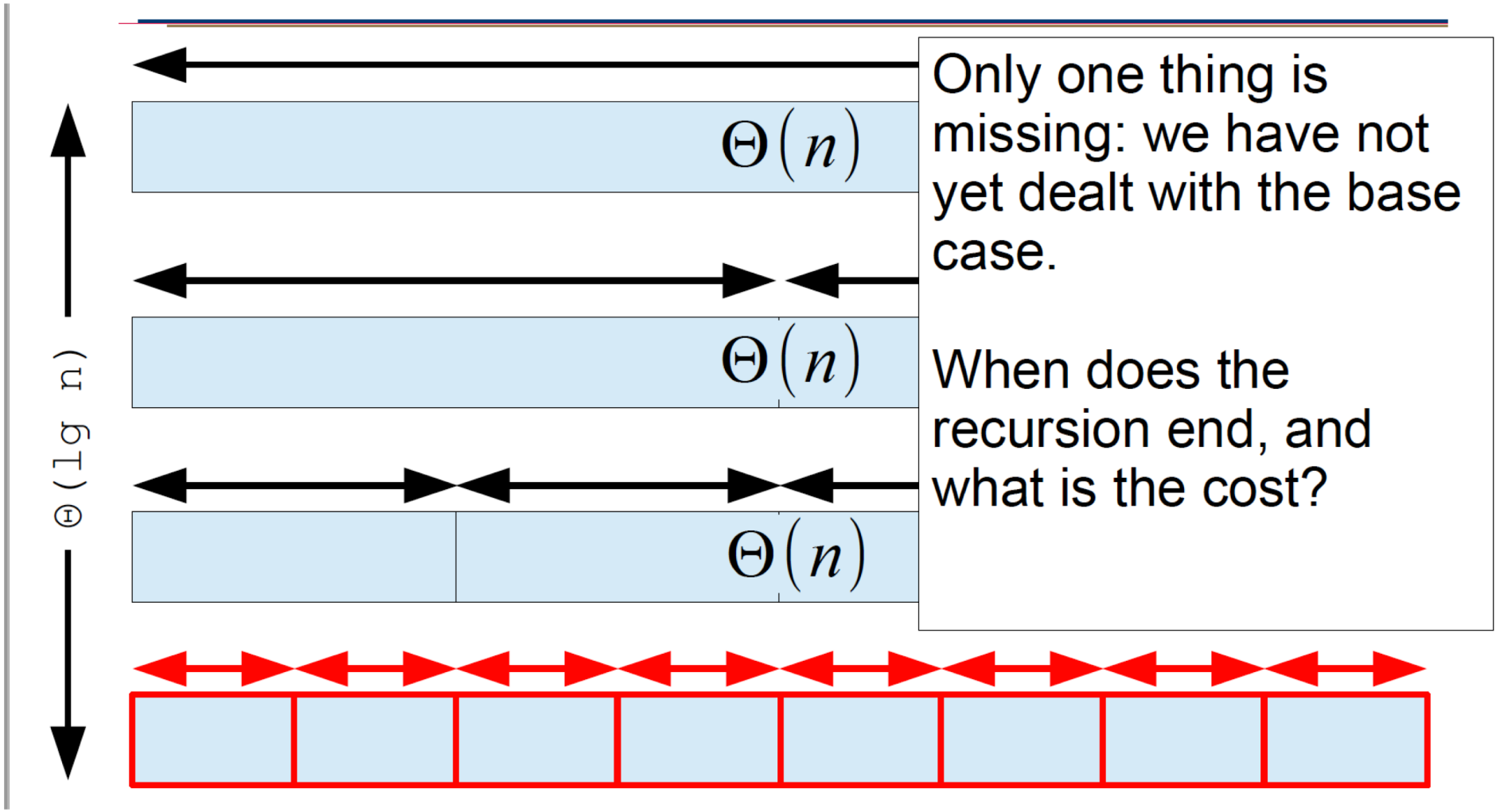
$$2^L \frac{c_0 n}{2^L} = c_0 n = \Theta(n)$$

- So every layer has the same cost
  - But only if the **number of splits** equals **the divisor**

Thinking About Recursive Calls



Thinking About Recursive Calls



Divide and Conquer

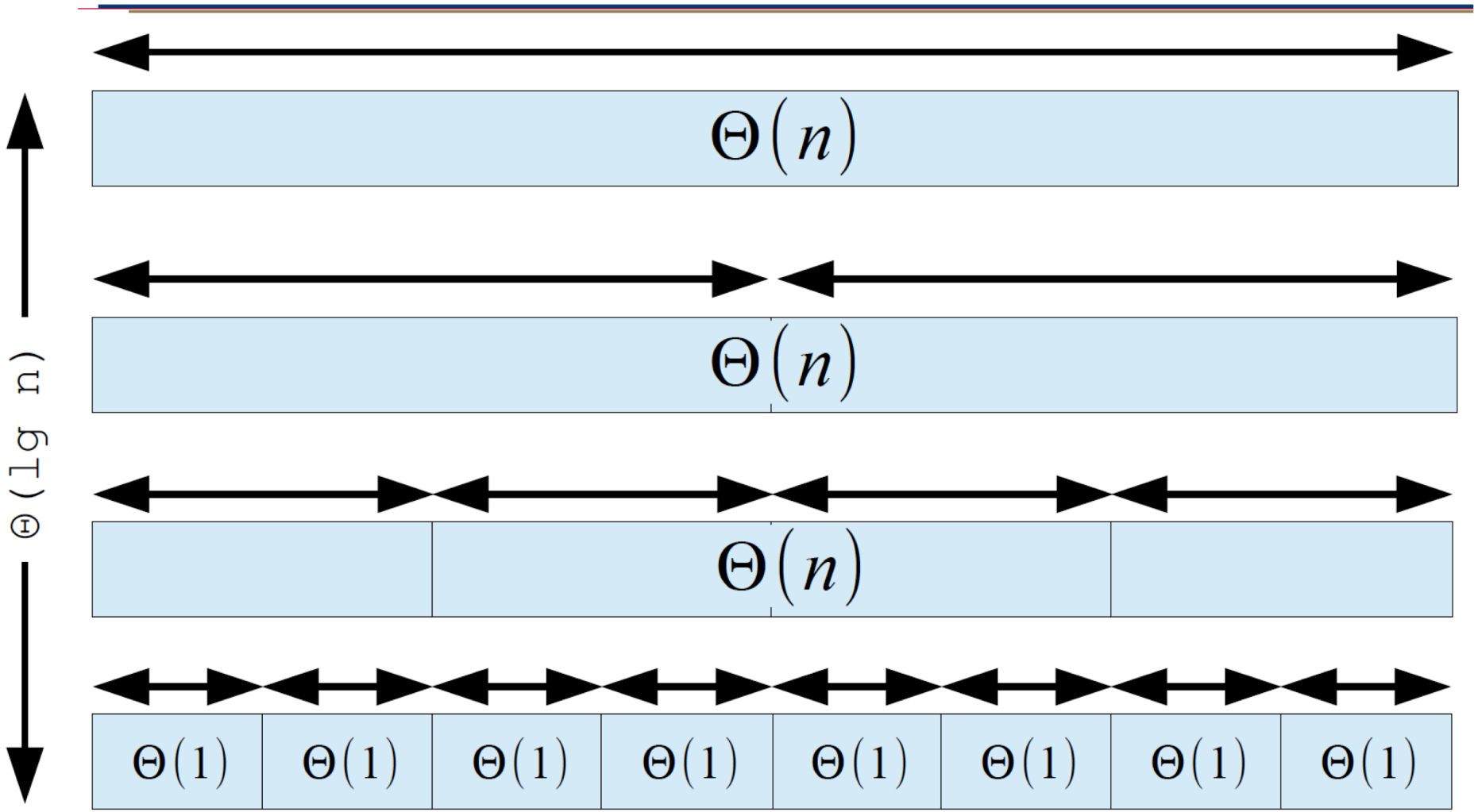
```
findMax(array):
    if (array.length <= 8)
        return specialCase(array)

    part1 = lower half of array
    part2 = upper half of
    max1 = findMa
    max2 = findMa
    return max(ma
```

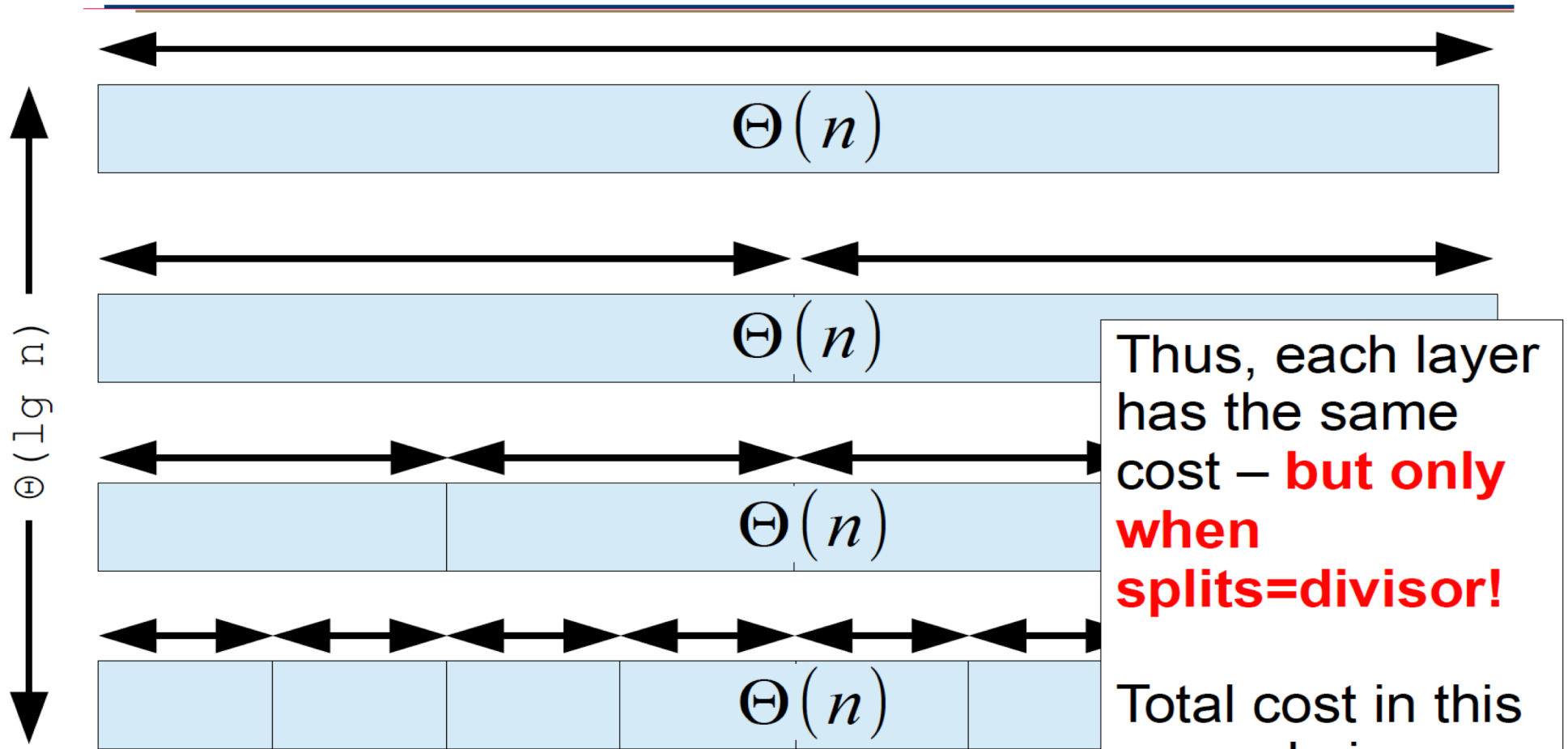
The base case generally assumes:

- Happens at fixed size
- Takes $\Theta(1)$ time

Thinking About Recursive Calls



Thinking About Recursive Calls



Thus, each layer has the same cost – **but only when splits=divisor!**

Total cost in this example is:

$\Theta(n \lg n)$

What is a Recurrence?

- A “recurrence” is:
  - An equality (or inequality) describing cost
  - Defined recursively
  - Piecewise definition to handle base case

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2\mathrm{T}\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

What is a Recurrence?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2\mathrm{T}\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Two Recursive Calls

Each Handles Half the Data

Linear Cost to Split and/or Join the Data

What is a Recurrence?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2\mathrm{T}\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Constant time...

...for the base case(s)

Generalizing

$$T(n) = 2\mathrm{T}\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$T(n) = 3\mathrm{T}\left(\frac{n}{4}\right) + \Theta(n)$$

$$T(n) = 2\mathrm{T}\left(\frac{9\mathrm{n}}{10}\right) + \Theta(lg\, n)$$

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$ (binary search)

$$T(n) = 2\mathrm{T}(n-1) + \Theta(1)$$ (boolean satifiability)

The Recursion Tree Method

- To solve a recurrence using a recursion tree:
  - Show the cost of each recursive call at each layer
  - Sum each layer
  - Sum the layers

Building a Recursion Tree

There are `n` elements.

The overall cost of the entire tree is `T(n)`.

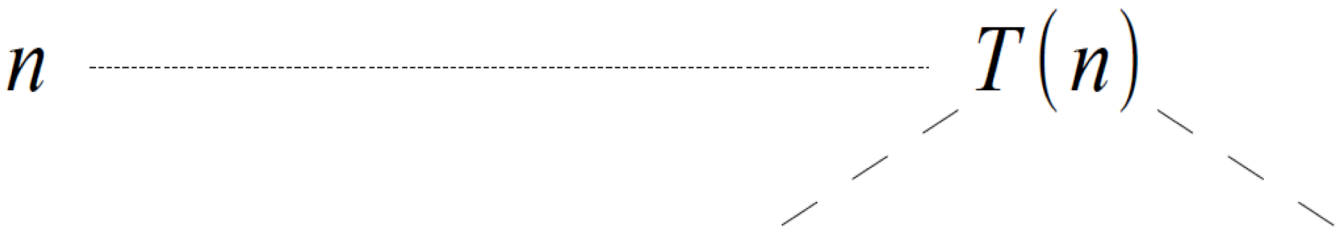
n ..... $T(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

Building a Recursion Tree

Assume that `n` is large, and thus `T(n)` will recurse.

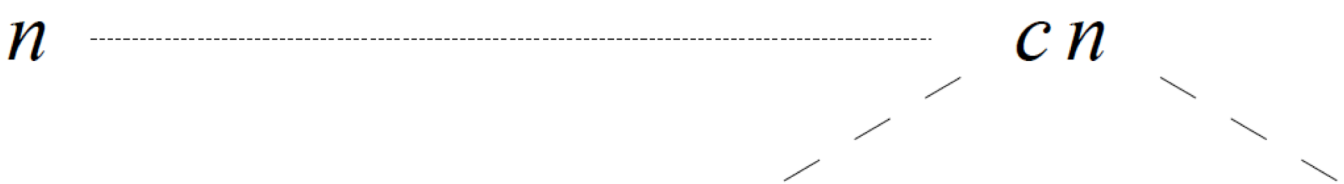
$$T(n) = 2\mathrm{T}\left(\frac{n}{2}\right) + c\,n$$



Building a Recursion Tree

The per-level cost is `c n`

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

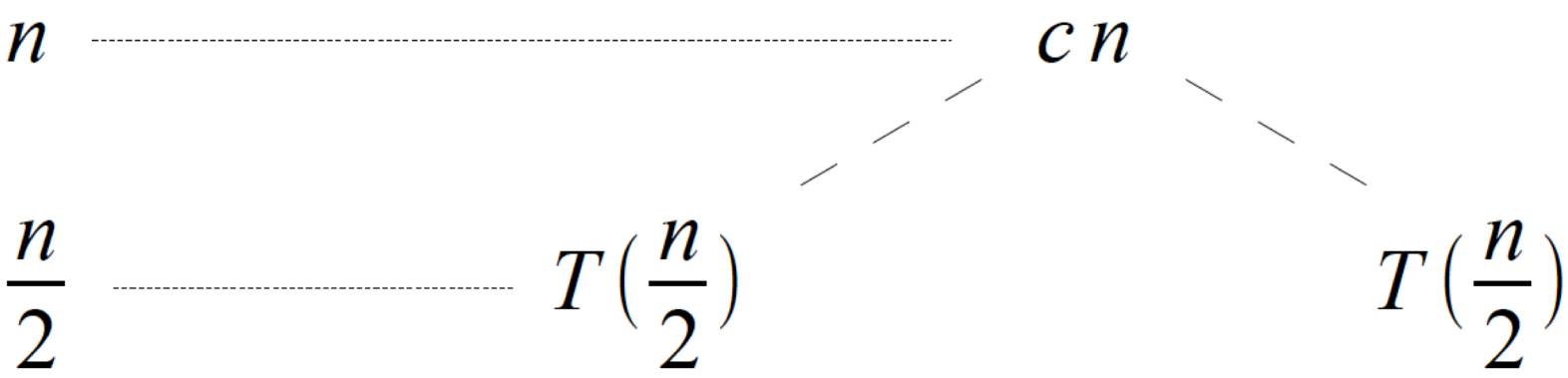


Building a Recursion Tree

There are two recursive calls.

Each call has size `n/2`

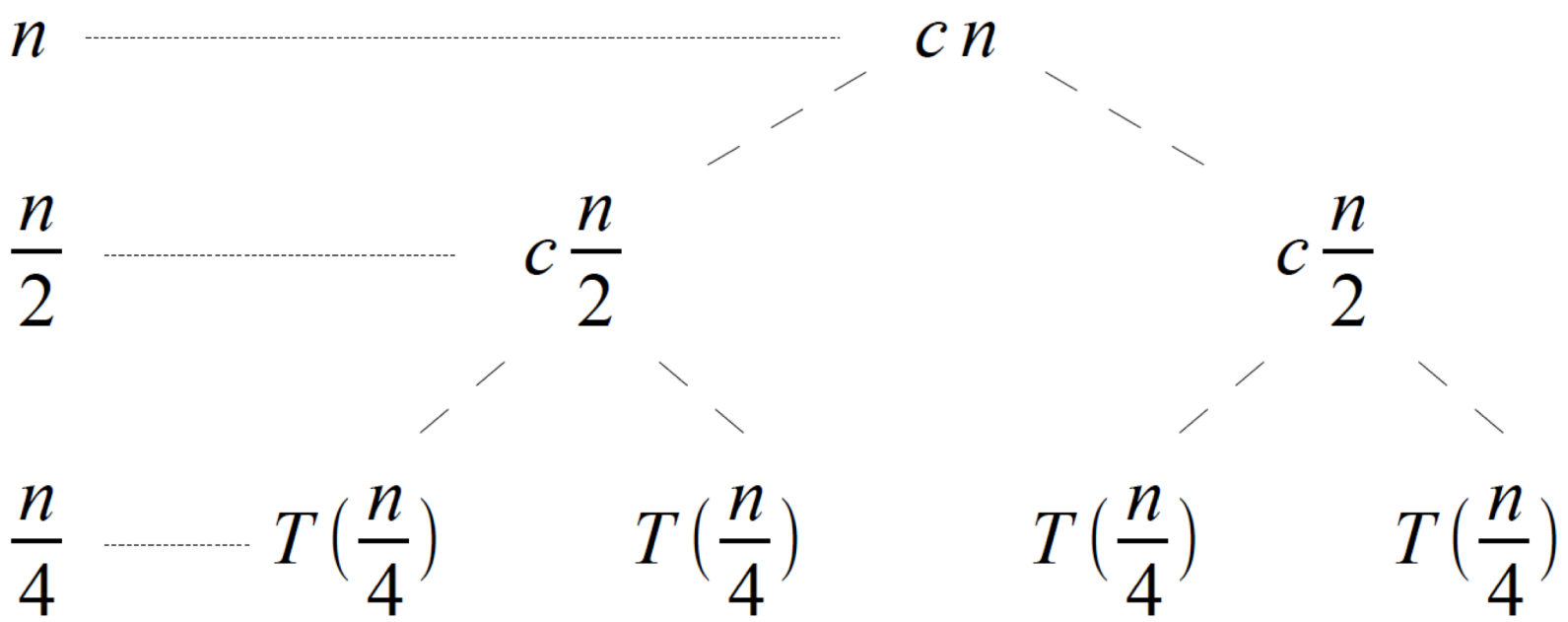
$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$



Building a Recursion Tree

We recurse to a third level, using the same logic.

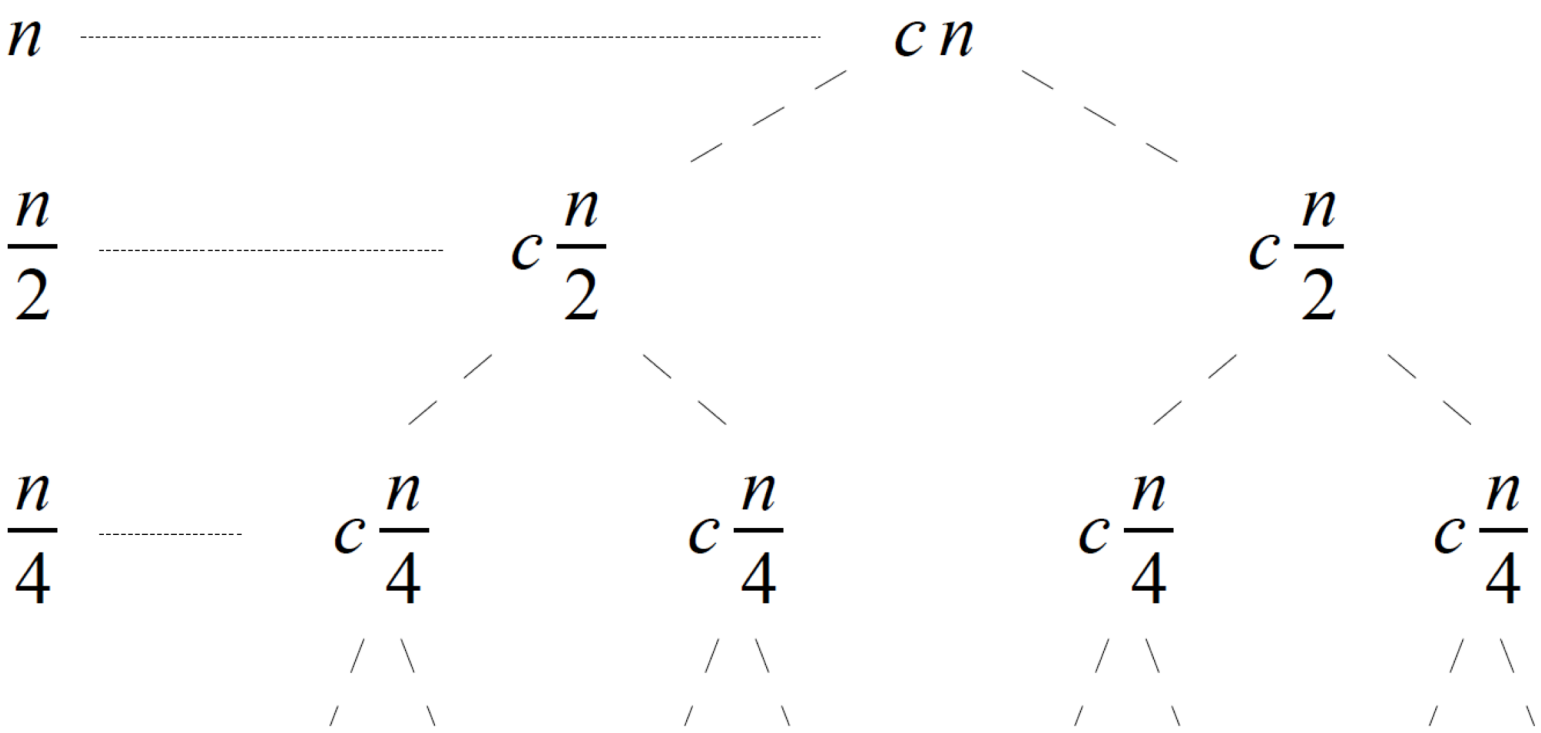
$$T(n)=2\mathrm{T}(\frac{n}{2})+c\,n$$



Building a Recursion Tree

Recursion continues, down many levels.

$$T(n) = 2\mathrm{T}\left(\frac{n}{2}\right) + c\,n$$

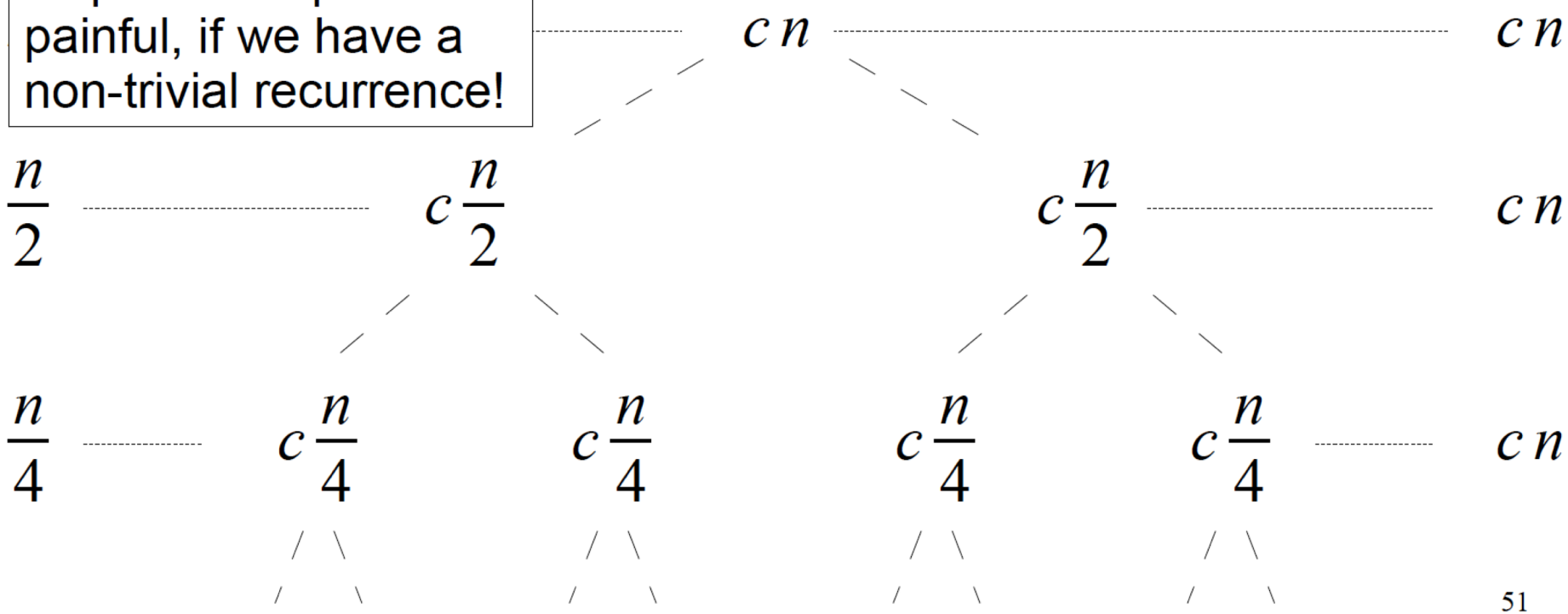


Building a Recursion Tree

$$T(n) = 2\mathrm{T}\left(\frac{n}{2}\right) + c\,n$$

Total up each layer.

The algebra at this step can be quite painful, if we have a non-trivial recurrence!

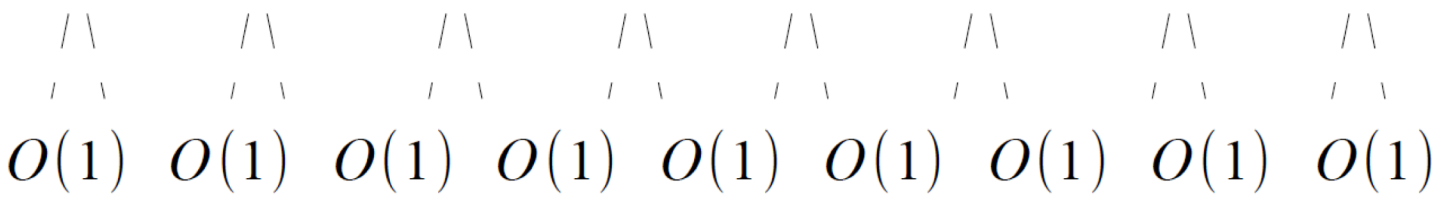


Building a Recursion Tree

Now we need to consider the leaves of the tree!

Each leaf costs `O(1)`. But how many are there? (Don't assume `n` – that only works in special cases!)

$$T(n) = 2\mathrm{T}\left(\frac{n}{2}\right) + c\,n$$

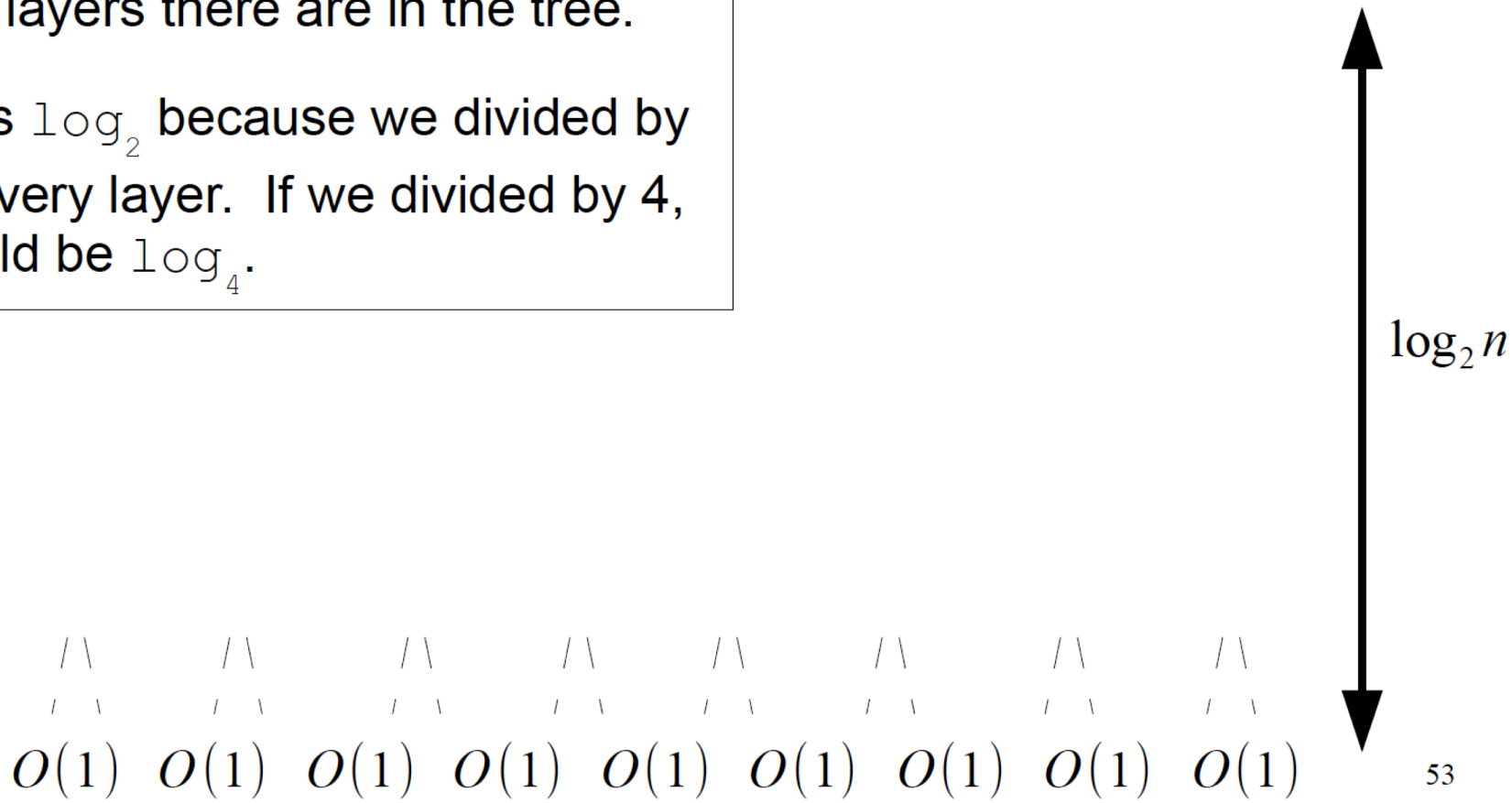


Building a Recursion Tree

Use a logarithm to figure out how many layers there are in the tree.

This is $\log_2$ because we divided by 2 at every layer. If we divided by 4, it would be $\log_4$.

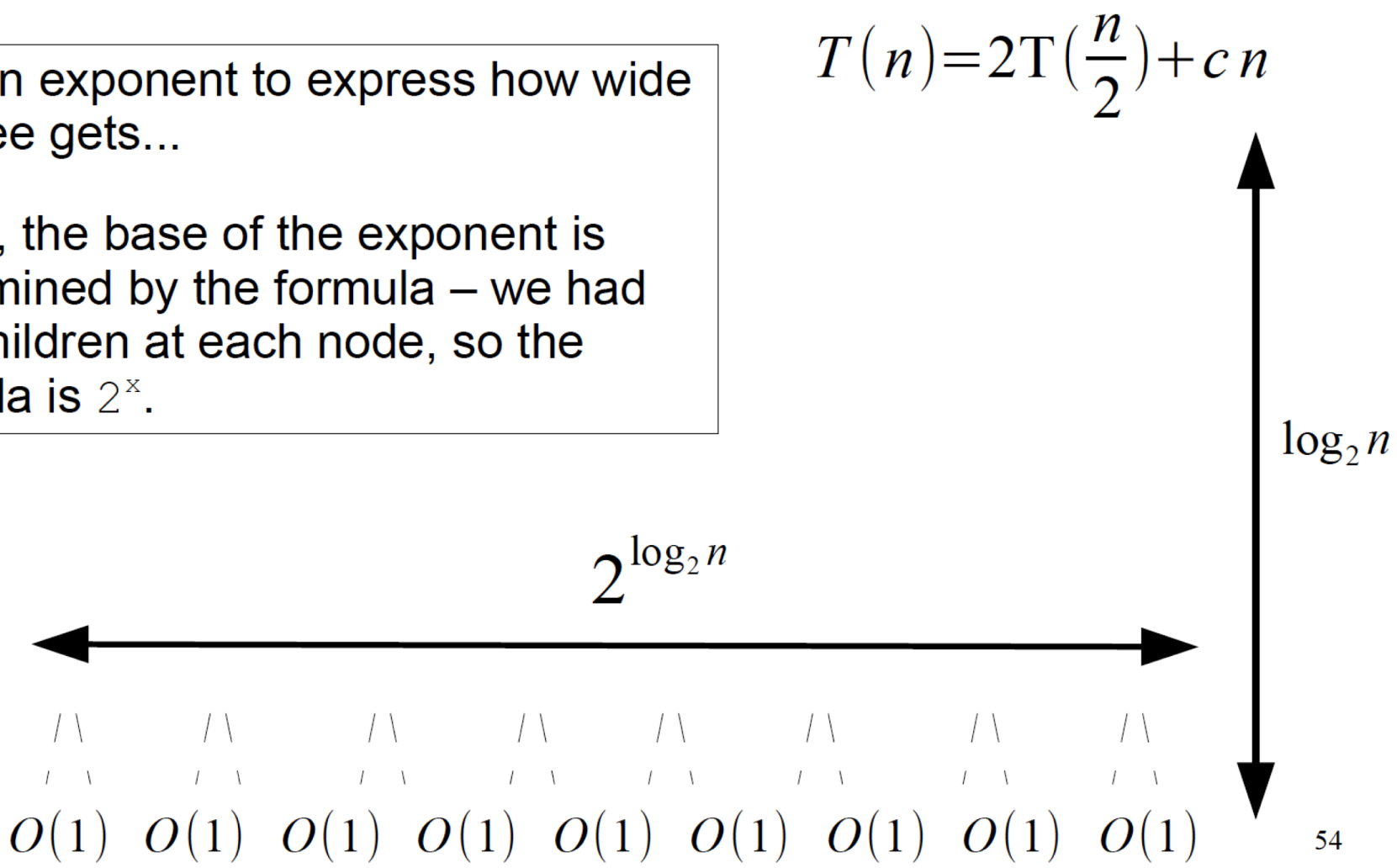
$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$



Building a Recursion Tree

Use an exponent to express how wide the tree gets...

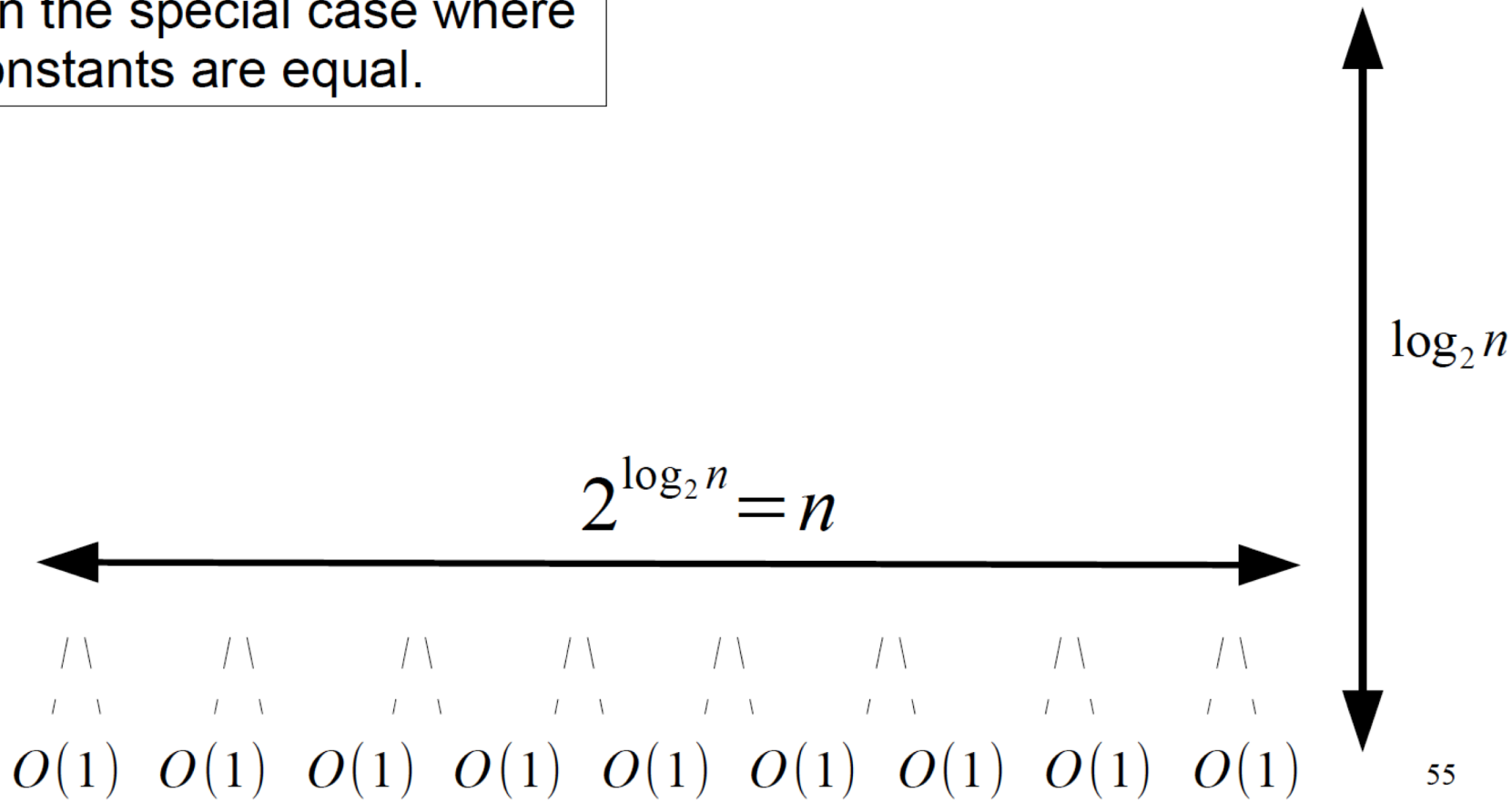
Again, the base of the exponent is determined by the formula – we had two children at each node, so the formula is 2^x.



Building a Recursion Tree

The number of leaves is n **only** in the special case where the constants are equal.

$$T(n)=2\mathrm{T}\left(\frac{n}{2}\right)+c\,n$$

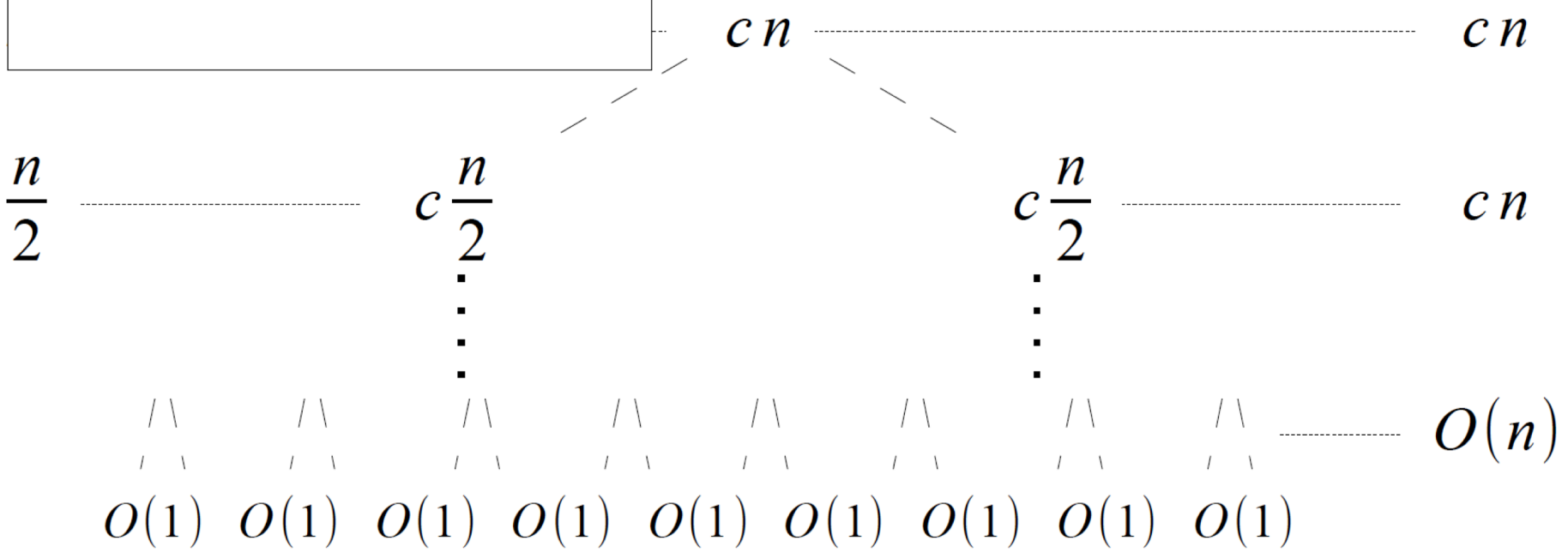


Building a Recursion Tree

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

Now, we sum the layers:

- Each of the upper layers
- Plus the leaves



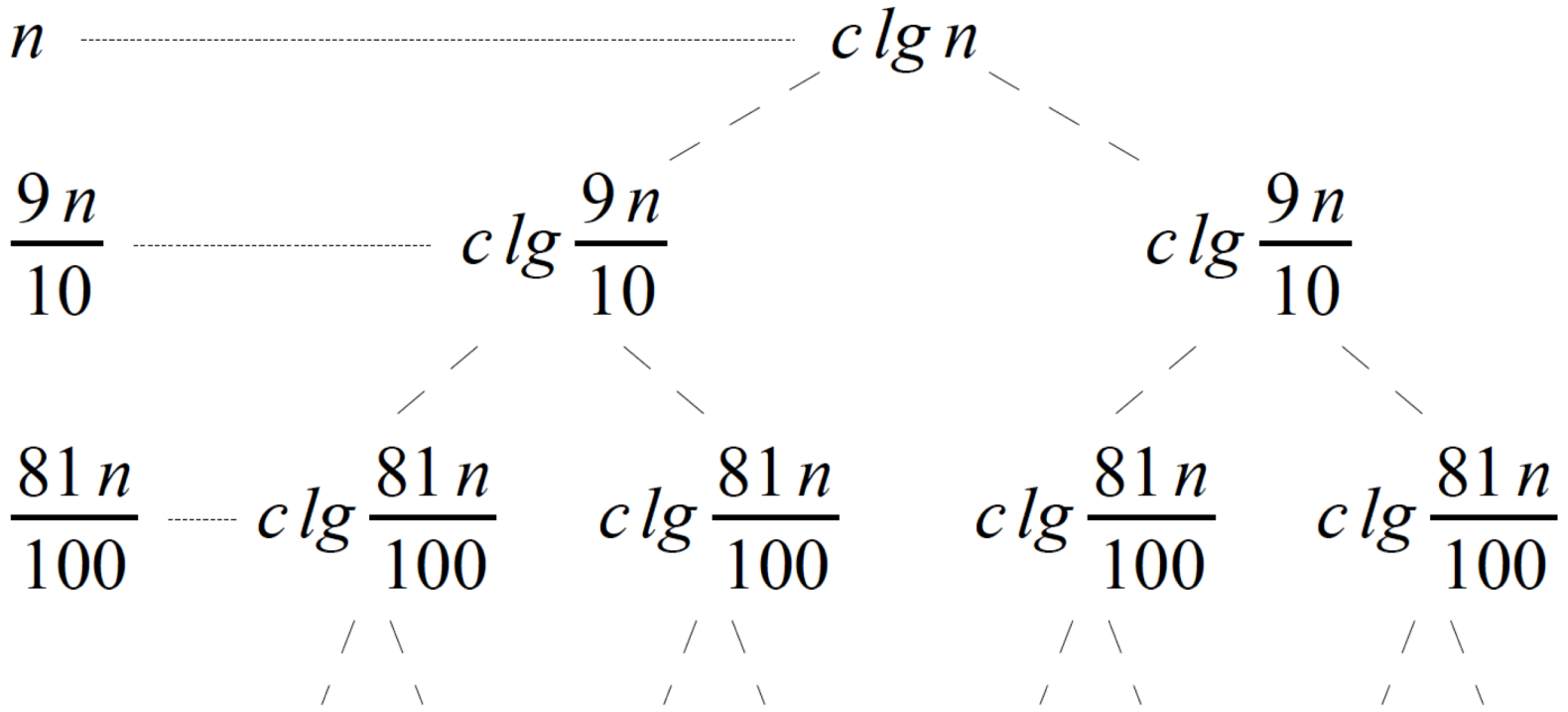
Overall total: $O(n \lg n)$

Building a Recursion Tree

$$T(n) = 2\mathrm{T}\left(\frac{9\mathrm{n}}{10}\right) + \Theta(lg\, n)$$

Building a Recursion Tree

$$T(n) = 2\mathrm{T}\left(\frac{9\mathrm{n}}{10}\right) + c\,lg\,n$$

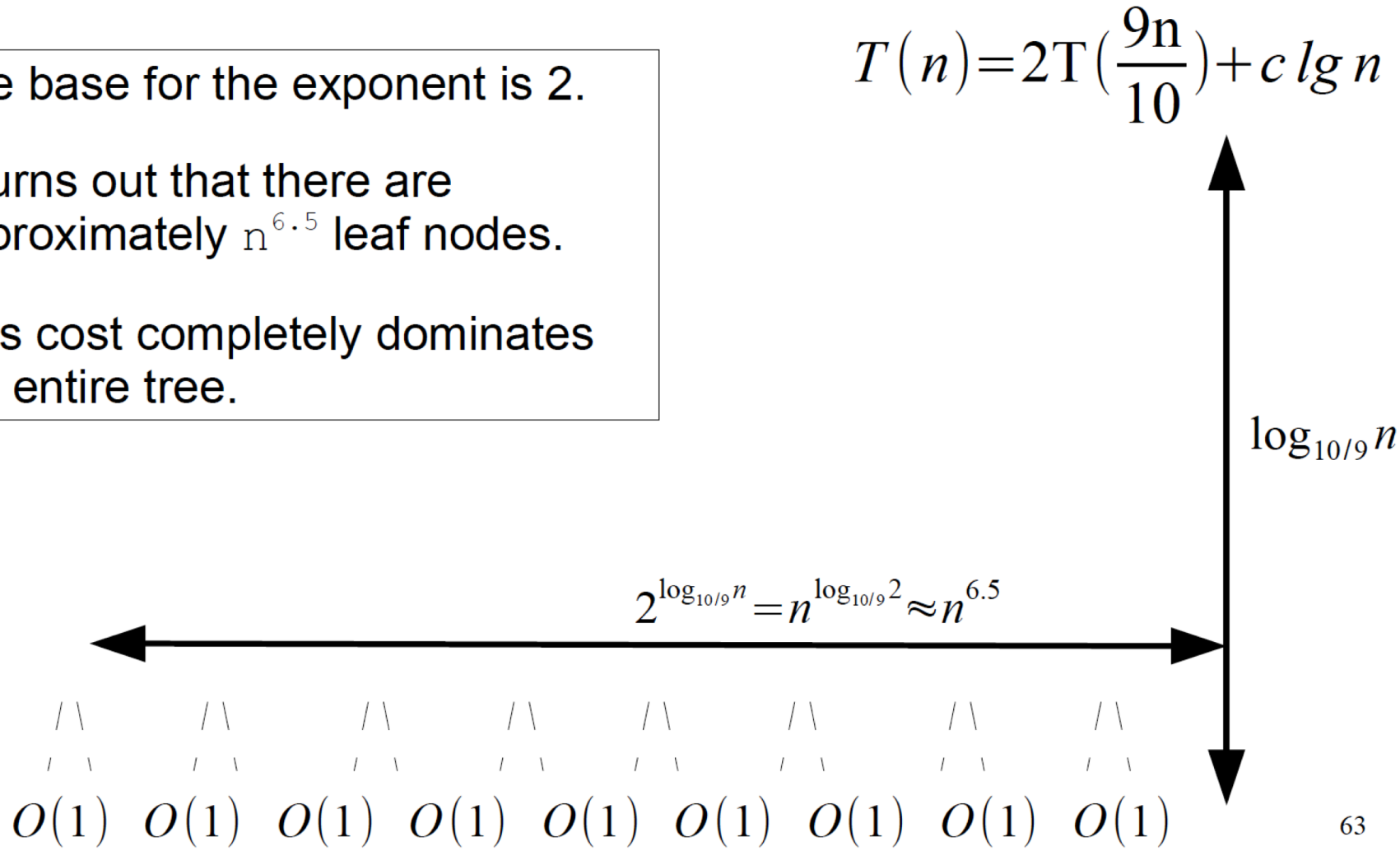


Building a Recursion Tree

The base for the exponent is 2.

It turns out that there are approximately $n^{6.5}$ leaf nodes.

This cost completely dominates the entire tree.



Building a Recursion Tree

$$T(n) = 2\mathrm{T}\left(\frac{9\mathrm{n}}{10}\right) + c\,lg\,n$$

$$T(n) = \Theta\left(n^{\log_{10/9} 2}\right)$$

Is a Recursion Tree Enough?

- We used an approximation in our recursion tree
- Do we trust our result?
- We could solve it formally

The Master Method

- Recursion trees and induction is hard!
- Isn't there some easier way?
- Welcome to the Master Method!

The Master Method

- Master Method is a theorem which gives mechanical answers for many common recurrences
- Requires that the recurrence be in a standard form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1$$

- Has three cases, to handle common scenarios

The Master Method

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

- Step 1: Calculate $\log_b a$
- Step 2: Compare $f(n)$ to $n^{\log_b a}$
  - Hopefully, they are equal – or different by a polynomial factor.
- Step 3: Find the appropriate case

Master Method

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

- Case 1: $f(n) = O\left(n^{\log_b a - \epsilon}\right), \epsilon > 0$

- Solution: $T(n) = \Theta\left(n^{\log_b a}\right)$

`f(n)` is smaller than the target function, by **at least** a polynomial factor.

Thus, the leaves dominate.

Master Method

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

- Case 2: $f(n) = \Theta\left(n^{\log_b a}\right)$

- Solution: $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

The leaves and upper nodes are balanced, and thus contribute equally.

The cost is the cost of each layer, times the number of layers.

Master Method

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

`f(n)` is **larger** than the target function, by **at least** a polynomial factor.

- Case 3: $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right), \epsilon > 0$
  $a\,f(n/b) \le c\,f(n), c < 1, \textit{for all large } n$

`f(n)` gets smaller as you reduce `n` .

- Solution: $T(n) = \Theta(f(n))$

Thus, `f(n)` dominates the recurrence.

Master Method Examples

$$T(n)=2\mathrm{T}\left(\frac{n}{2}\right)+n^2$$

$$a=?$$
$$b=?$$
$$\log_b a=?$$
$$n^{\log_b a}=?$$

Which case? $?$

$$T(n)=?$$

Master Method Examples

$$T(n)=2\mathrm{T}\left(\frac{n}{2}\right)+n^2$$

$$a=2$$
$$b=2$$
$$\log_b a=1$$
$$n^{\log_b a}=n$$

Case 3 $$f(n)=\Omega\left(n^{1+\epsilon}\right),\ \epsilon>0$$

$$T(n)=\Theta\left(n^2\right)$$

Master Method Examples

$$T(n) = 3\mathrm{T}\left(\frac{n}{4}\right) + n$$

$$a = ?$$
$$b = ?$$
$$\log_b a = ?$$
$$n^{\log_b a} = ?$$

Which case?

$$?$$

$$T(n) = ?$$

Master Method Examples

$$T(n)=3\mathrm{T}\left(\frac{n}{4}\right)+n$$

$$a=3$$
$$b=4$$
$$\log_b a=\log_4 3\approx .79$$
$$n^{\log_b a}\approx n^{.79}$$

Case 3

$$f(n)=\Omega\left(n^{.79+\epsilon}\right),\epsilon>0$$

$$T(n)=\Theta(n)$$

Master Method Examples

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

$$a = ?$$
$$b = ?$$
$$\log_b a = ?$$
$$n^{\log_b a} = ?$$

Which case? $?$

$$T(n) = ?$$

Master Method Examples

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

$$a = 1$$
$$b = 2$$
$$\log_b a = 0$$
$$n^{\log_b a} = n^0 = 1$$

Case 2 $$f(n) = \Theta(n^0)$$

$$T(n) = \Theta(n^0 \lg n) = \Theta(\lg n)$$