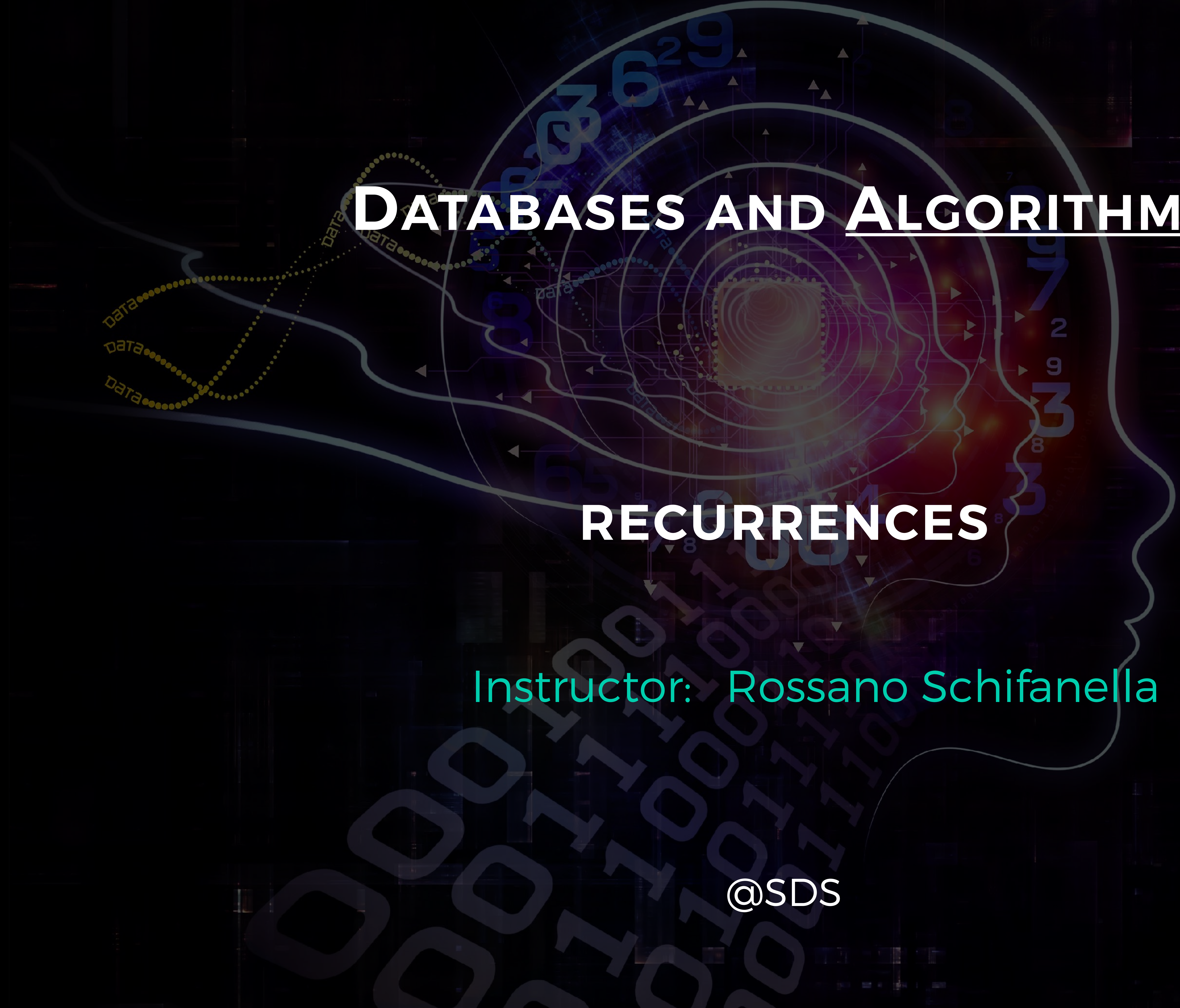


Databases and Algorithms

Recurrences

Instructor: Rossano Schifanella

@SDS

Recurrences

- Recurrences arise when an algorithm contains recursive calls to itself
- Running time is represented by an equation or inequality that describes a function in terms of its value on smaller inputs.
- $T(n) = T(n-1) + n$
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression
  - Bound the recurrence by an expression that involves n

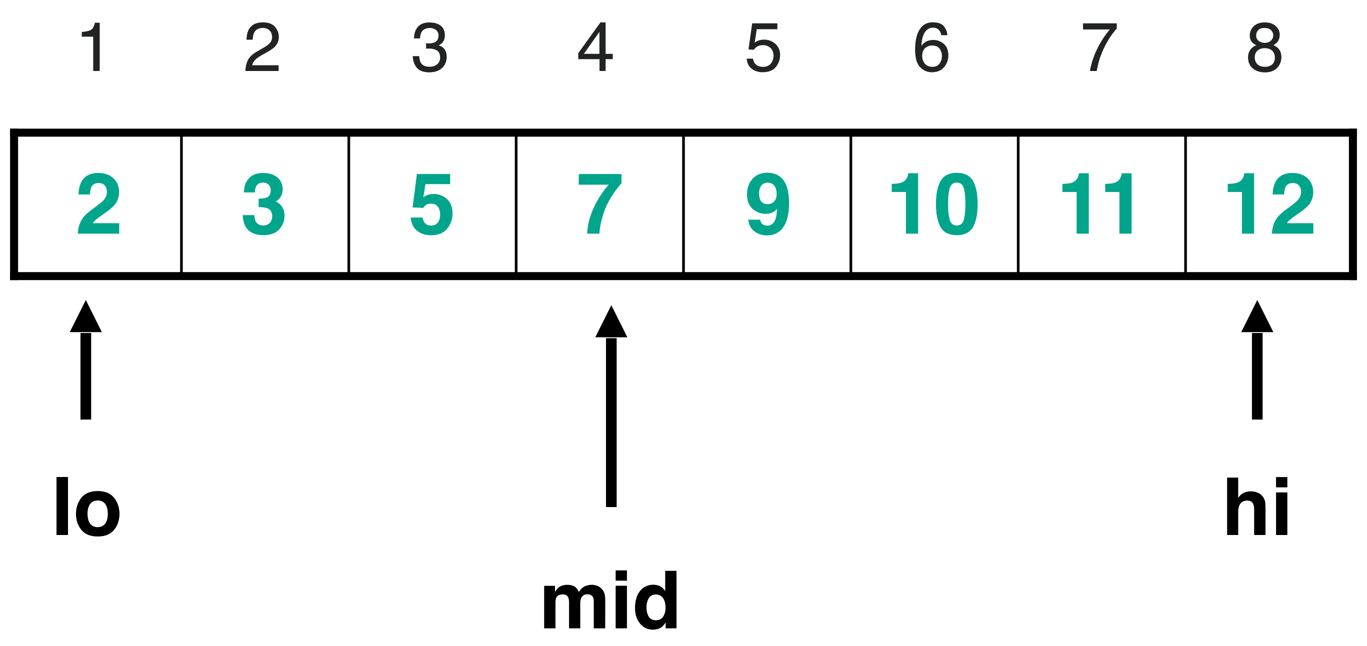
Examples

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\log n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Binary-search

```
BINARY-SEARCH (A, lo, hi, x)
    if (lo > hi)
        return FALSE
    mid ← ⌊(lo+hi)/2⌋

    if x = A[mid]
        return TRUE
    if ( x < A[mid] )
        BINARY-SEARCH (A, lo, mid-1, x)
    if ( x > A[mid] )
        BINARY-SEARCH (A, mid+1, hi, x)
```



Binary-search

$T(N) = C + T(N/2)$

```
BINARY-SEARCH (A, lo, hi, x)
    if (lo > hi)
        return FALSE
    mid ← ⌊(lo+hi)/2⌋
    if x = A[mid]
        return TRUE
    if ( x < A[mid] )
        BINARY-SEARCH (A, lo, mid-1, x)
    if ( x > A[mid] )
        BINARY-SEARCH (A, mid+1, hi, x)
```

CONSTANT TIME: C

CONSTANT TIME: C

CONSTANT TIME: C

CONSTANT TIME: C

CONSTANT TIME: C

CONSTANT TIME: C

SAME PROBLEM OF SIZE N/2

CONSTANT TIME: C

SAME PROBLEM OF SIZE N/2

Methods for Solving Recurrences

- **Substitution** method
  - **Iteration** method
  - **Recursion-tree** method
- **Master** method

Substitution Method (CLRs 4.3)

- The substitution method is a condensed way of proving an asymptotic bound on a recurrence by induction.
- Main steps:
  - Guess the form of the solution.
  - Use mathematical induction to find the constants and show that the solution works.

Substitution Method (CLRs 4.3)

- Example, merge-sort:
  - $T(n) = 2T(n/2) + n$
  - We guess that the answer is $O(n \log_2 n)$
  - Prove it by induction
  - Can similarly show $T(n) = \Omega(n \log n)$, thus $\Theta(n \log_2 n)$

EXAMPLE

- T(n) = 2T(n/2) + n
- The substitution method requires us to prove that T(n) ≤ cn log n for an appropriate choice of the constant c > 0
- We start by assuming that this bound holds for all positive m < n, in particular for m = n/2, yielding T(n/2) ≤ c n/2 log n/2

GUESS SOLUTION O(N LOG N)

WITH C≥1

CHECK BOUNDARY CONDITIONS!

$$\begin{aligned}
T(n) &\le 2(c\lfloor n/2\rfloor \lg(\lfloor n/2\rfloor)) + n \\
&\le cn\lg(n/2) + n \\
&= cn\lg n - cn\lg 2 + n \\
&= cn\lg n - cn + n \\
&\le cn\lg n\,,
\end{aligned}$$

CONVERT THE RECURRENCE INTO A SUMMATION AND SOLVE IT USING A KNOWN SERIES

Iteration Method

Recurrence:

$$T(n) = \begin{cases} 1,\ if\ n = 1 \\ T(\frac{n}{2}) + c,\ if\ n > 1 \end{cases}$$

Iteration:

$$\begin{aligned} T(n) &= T(\frac{n}{2}) + c \\ &= T(\frac{n}{4}) + c + c = T(\frac{n}{4}) + 2c \\ &= T(\frac{n}{2^k}) + kc \end{aligned}$$

Base case:

$$\frac{n}{2^k} = 1 \implies k = log_2 n$$

Guess:

$$\begin{aligned} T(n) &= T(\frac{n}{2^{log_2 n}}) + log_2 n \cdot c \\ &= T(1) + log_2 n \cdot c \\ &= 1 + log_2 n \cdot c \\ &= O(log_2 n) \end{aligned}$$

Another example

Recurrence:

$$T(n) = \begin{cases} 1, \ if\ n = 1 \\ 2T(\frac{n}{2}) + cn, \ if\ n > 1 \end{cases}$$

Base case:

$$\frac{n}{2^k} = 1 \implies k = log_2 n$$

Iteration:

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + cn \\ &= 2(2T(\frac{n}{4}) + c(\frac{n}{2})) + cn = 4T(\frac{n}{4}) + 2cn \\ &= 2^k T(\frac{n}{2^k}) + kcn \end{aligned}$$

Guess:

$$\begin{aligned} T(n) &= 2^{log_2 n} T(\frac{n}{2^{log_2 n}}) + log_2 n \cdot cn \\ &= n \cdot T(1) + log_2 n \cdot cn \\ &= n + log_2 n \cdot cn \\ &= O(n \cdot log_2 n) \end{aligned}$$

Another example

Recurrence:

$$T(n) = \begin{cases} 1, \, if \, n < = 1 \\ T(n-1) + T(n-2), \, if \, n > 1 \end{cases}$$

Base case:

$$n - k = 1 \implies k = n - 1$$

Iteration:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) \approx 2T(n-1) \\ &= 2(2T(n-2)) \\ &= 2^k T(n-k) \end{aligned}$$

Guess:

$$\begin{aligned} T(n) &= 2^{n-1} T(n-n+1) \\ &= \frac{1}{2} \cdot 2^n \cdot T(1) \\ &= O(2^n) \end{aligned}$$

Observations

- Brute force method
- Once we “guess” the form of the solution for a recurrence relation, we need to verify it is, in fact, the solution.
- We use **induction** for this.

Recursion-tree method (CLRS 4.4)

- Convert the recurrence into a <u>tree</u>:
  - Each node represents the cost incurred at various levels of the recursion
  - Sum up the costs of all levels
  - Serves as a straightforward way to devise a good guess
  - A recursion tree is best used to generate a good guess, which you can then verify by the substitution method

Recurrence:

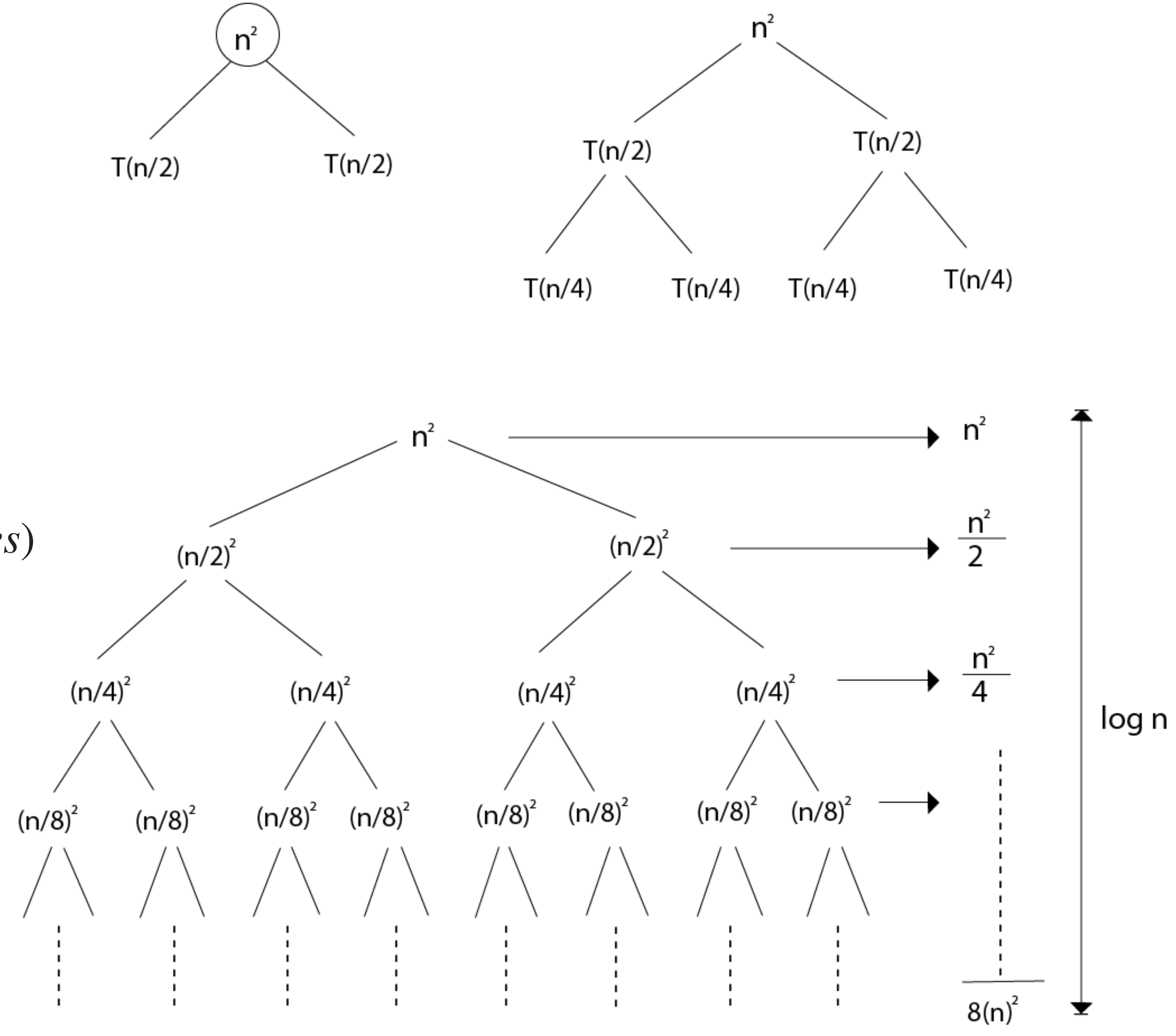
$$T(n) = \begin{cases} 1, \ if\ n <= 1 \\ 2T\left(\frac{n}{2}\right) + n^2, \ if\ n > 1 \end{cases}$$

$$T(n) = n^2 + \frac{n^2}{2} + \frac{n^4}{4} + \cdots + \ (log_2 n\ times)$$

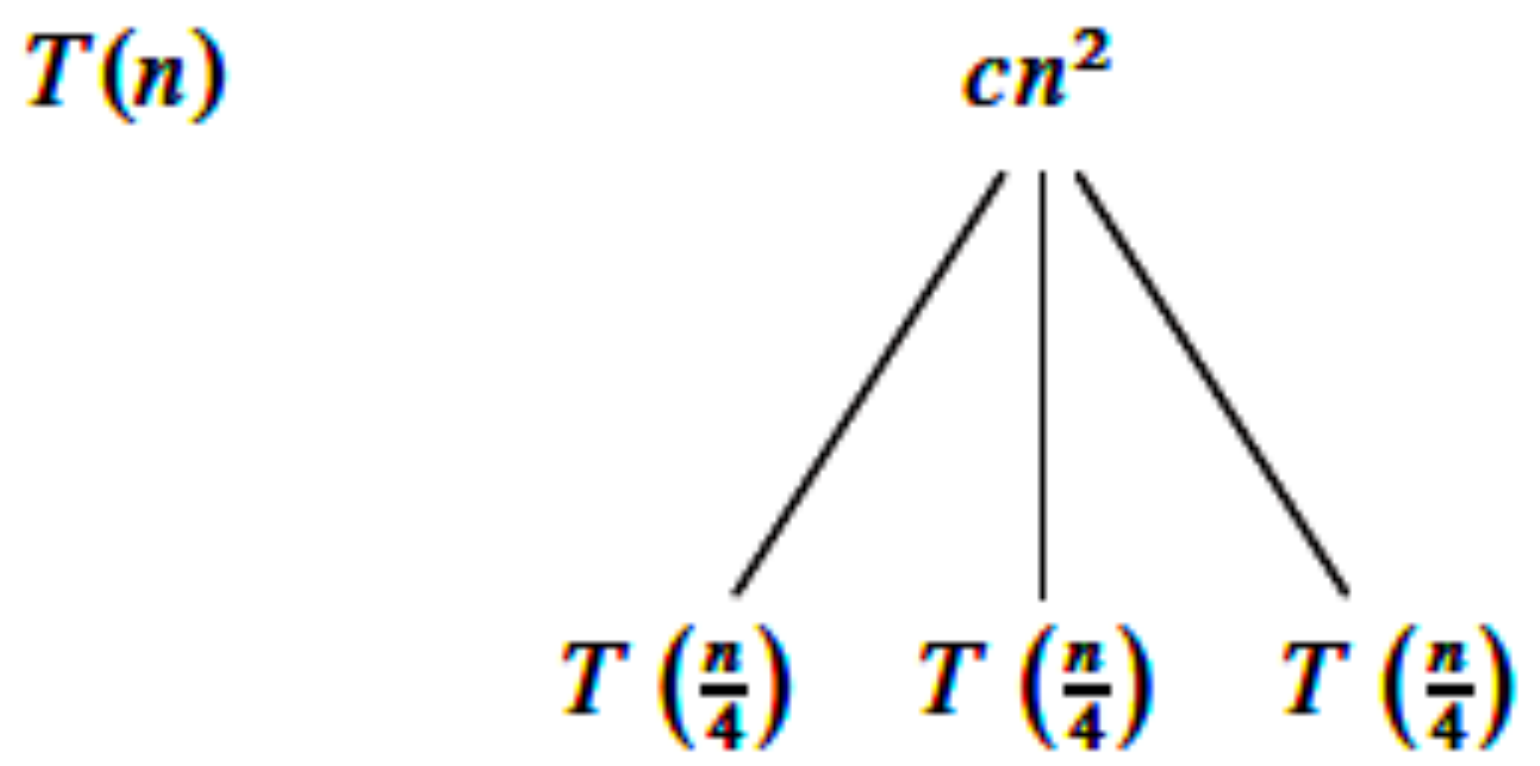
$$\leq n^2 \sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$= n^2 \cdot 1$$

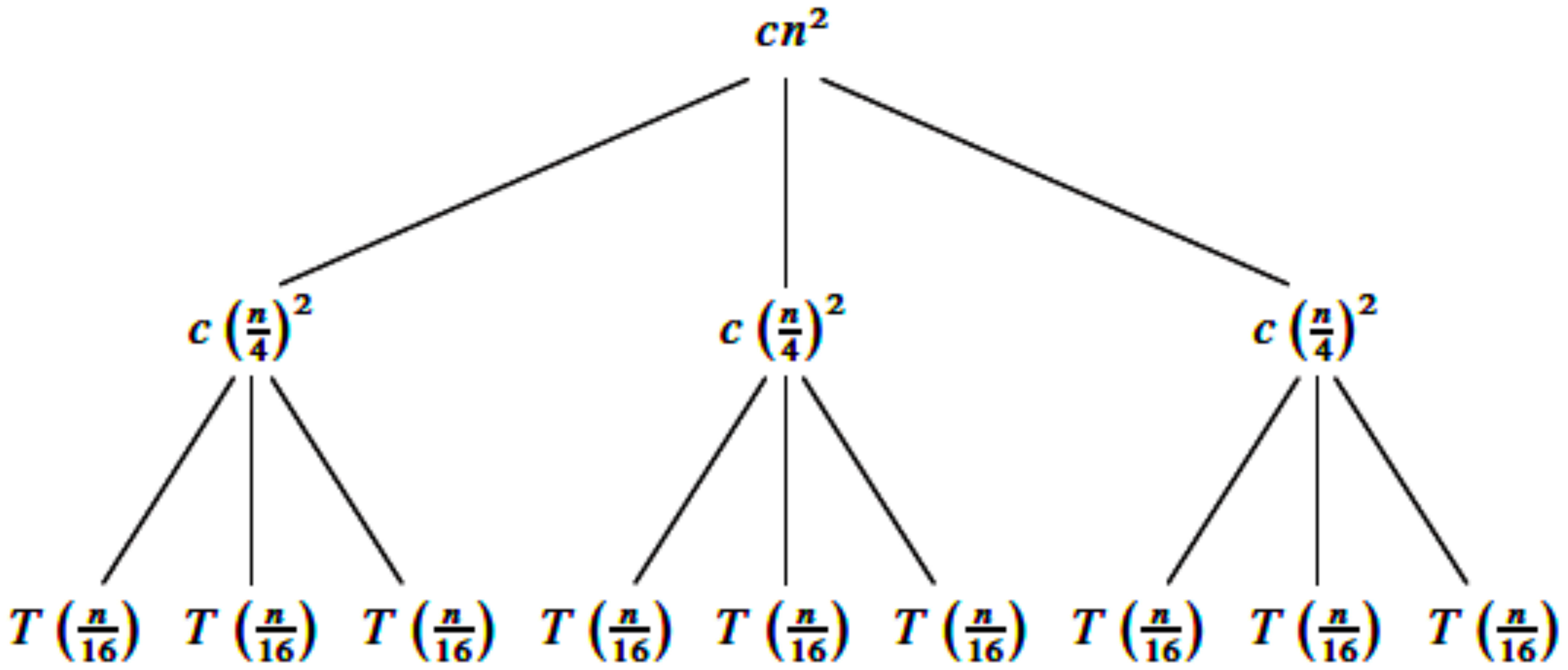
$$= O(n^2)$$



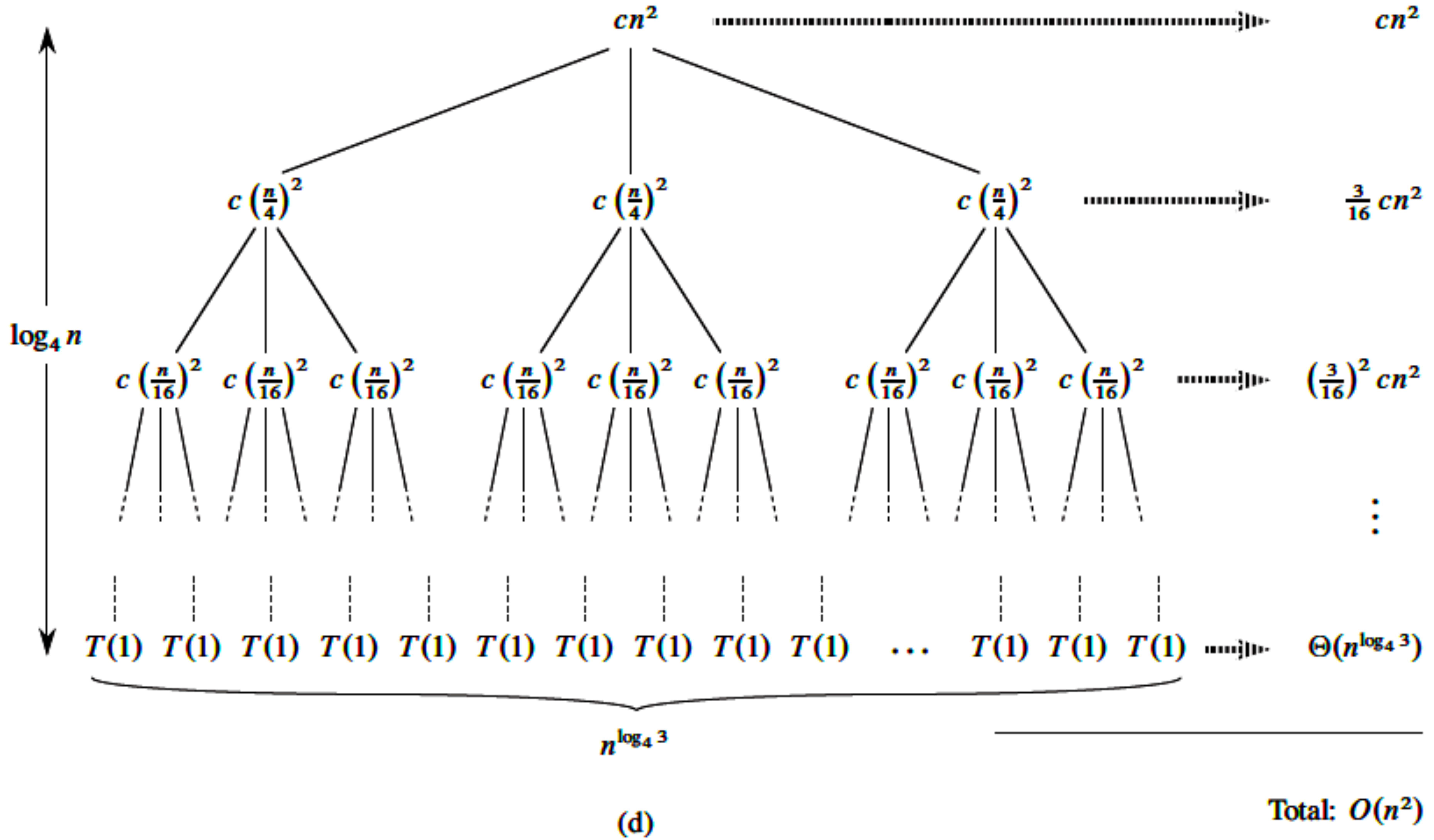
Example 1 $T(n) = 3T(n/4) + cn^2$



Example 1 $T(n) = 3T(n/4) + cn^2$



Example 1 $T(n) = 3T(n/4) + cn^2$



(d)

Example 1 $\quad T(n) = 3T(n/4) + cn^2$

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n-1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \qquad \text{(by equation (A.5))} .
\end{aligned}
$$

Example 1 T(n) = 3T(n/4) + cn²

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2) .
\end{aligned}
$$

Example 1 $T(n) = 3T(n/4) + cn^2$

- Now we can use the substitution method to verify that our guess was correct, that is, $T(n) = O(n^2)$ is an upper bound for the recurrence. We want to show that $T(n) \leq dn^2$ for some constant $d > 0$.

- Using the same constant $c > 0$ as before, we have

$$\begin{aligned} T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \end{aligned}$$

WITH $d \geq (16/13)\,c$

Master Theorem

- Let T(n) be a monotonically increasing function that satisfies

$$T(n) = a\,T(n/b) + f(n)$$

$$T(1) = c$$

where $a \geq 1$, $b \geq 2$, $c>0$. If f(n) is $\Theta(n^d)$ where $d \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^d) & : a < b^d \\ \Theta(n^d log\, n) & : a = b^d \\ \Theta(n^{log_b a}) & : a > b^d \end{cases}$$

Master Theorem: Pitfalls

- You cannot use the Master Theorem if
  - **T(n) is not monotone**, e.g. T(n) = sin(x)
  - **f(n) is not a polynomial**, e.g., $T(n)=2T(n/2)+2^n$
  - **b cannot be expressed as a constant**, e.g.

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

- Note that the Master Theorem does not solve the recurrence equation

Master Theorem: Example 1

Let $T(n) = T(n/2) + \frac{1}{2} n^2 + n$. What are the parameters?

$a = 1$

$b = 2$

$d = 2$

Therefore, which condition applies?

$b^d = 4$ so $a < b^d$ => case 1 applies

$$T(n) \in \Theta(n^d) = \Theta(n^2)$$

Master Theorem: Example 2

Let $T(n) = 2\,T(n/4) + \sqrt{n} + 42$. What are the parameters?

$a = 2$

$b = 4$

$d = 1/2$

Therefore, which condition applies?

$b^d = 4^{1/2} = 2$ so $a = b^d$ => case 2 applies

$$T(n) \in \Theta(n^d log\, n) = \Theta(log\, n\sqrt{n})$$

Master Theorem: Example 3

Let T(n)= 3 T(n/2) + 3/4n + 1. What are the parameters?

a = 3

b = 2

d = 1

Therefore, which condition applies?

b^d=2 so a>b^d => case 3 applies

$$T(n) \in \Theta(n^{log_b a}) = \Theta(n^{log_2 3})$$

Fourth Condition

- Recall that we cannot use the Master Theorem if f(n), the non-recursive cost, is not a polynomial
- There is a limited 4th condition of the Master Theorem that allows us to consider polylogarithmic functions
- Corollary: If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some k≥0 then

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$$

- This final condition is fairly limited and we present it merely for sake of completeness.. Relax ☺️

Fourth Condition

- Recall that we cannot use the Master Theorem if f(n), the non-recursive cost, is not a polynomial
- There is a limited 4th condition of the Master Theorem that allows us to consider polylogarithmic functions

If $$f(n) \in \Theta(n^{\log_b a} \log^k n)$$

for some k≥0 then

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$$

This final condition is fairly limited and we present it merely for sake of completeness. Relax! ☺️

‘Fourth’ Condition: Example

Say we have the following recurrence relation

$$T(n)= 2\ T(n/2) + n \log n$$

Clearly, a=2, b=2, but f(n) is not a polynomial. However, we have $f(n) \in \Theta(n \log n)$, k=1

Therefore by the 4th condition of the Master Theorem we can say that

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n^{\log_2 2} \log^2 n) = \Theta(n \log^2 n)$$

Questions?

@rschifan

schifane@di.unito.it

http://www.di.unito.it/~schifane