

# Introduction to Algorithm Design and Analysis

[03] Recursion

Jingwei Xu

<https://ics.nju.edu.cn/~xjw/>

Institute of Computer Software  
Nanjing University

# In the Last Class ...

- Asymptotic growth rate

- $O, \Omega, \Theta$

- $o, \omega$

- Brute force algorithms

- By iteration

- By recursion

# Recursion

- **Recursion in algorithm design**
  - The divide and conquer strategy
  - Proving the correctness of recursive procedures
- **Solving recurrence equations**
  - Some elementary techniques
  - Master theorem

# Recursion in Algorithm Design

- Computing  $n!$  With  $\text{Fac}(n)$

- if  $n=1$  then return 1 else return  $\text{Fac}(n-1)*n$

**$M(1)=0$  and  $M(n)=M(n-1)+1$  for  $n>0$   
(critical operation: multiplication)**

- Hanoi Tower

- If  $n=1$  then move  $d(1)$  to peg3 else  $\text{Hanoi}(n-1, \text{peg1}, \text{peg2})$ ; move  $d(n)$  to peg3;  $\text{Hanoi}(n-1, \text{peg2}, \text{peg3})$

**$M(1)=1$  and  $M(n)=2M(n-1)+1$  for  $n>1$   
(critical operation: move)**

# Recursion in Algorithm Design

- **Counting the Number of Bits**
  - Input: a positive decimal integer  $n$
  - Output: the number of binary digits in  $n$ 's binary representation

```
int BitCounting(int n)
```

1. if( $n==1$ ) return 1;
2. else
3.     return BitCounting( $n/2$ ) + 1;

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lfloor n/2 \rfloor) + 1 & n > 1 \end{cases}$$

# Divide and Conquer

- Divide

- Divide the “big” problem to smaller ones

- Conquer

- Solve the “small” problems by **recursion**

- Combine

- Combine results of small problems, and solve the original problem

# Divide and Conquer

The general pattern

solve(I)

n=size(I);

if (n≤smallSize)

    solution=directlySolve(I);

else

    divide I into I<sub>1</sub>, ..., I<sub>k</sub>;

    for each i∈{1,...,k}

        S<sub>i</sub>=solve(I<sub>i</sub>);

    solution=combine(S<sub>1</sub>, ..., S<sub>k</sub>);

return solution

T(n)=B(n) for n≤smallSize

$$T(n) = D(n) + \sum_{i=1}^k T(\text{size}(I_i)) + C(n)$$

for n>smallSize

# Divide and Conquer

- The BF recursion

- Problem size: often decreases linearly
  - “ $n, n-1, n-2, \dots$ ”

- The D&C recursion

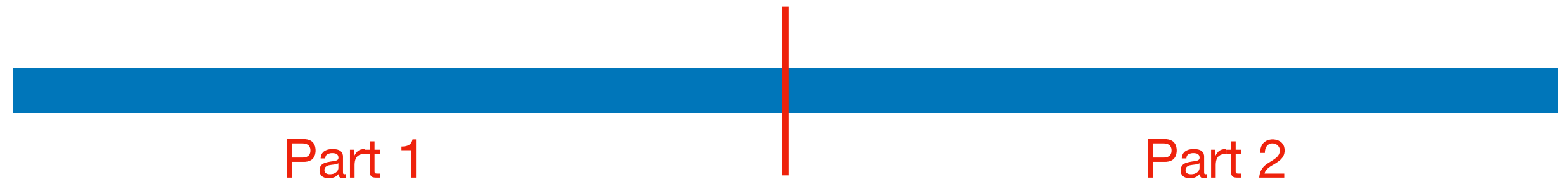
- Problem size: often decrease exponentially
  - “ $n, n/2, n/4, n/8, \dots$ ”



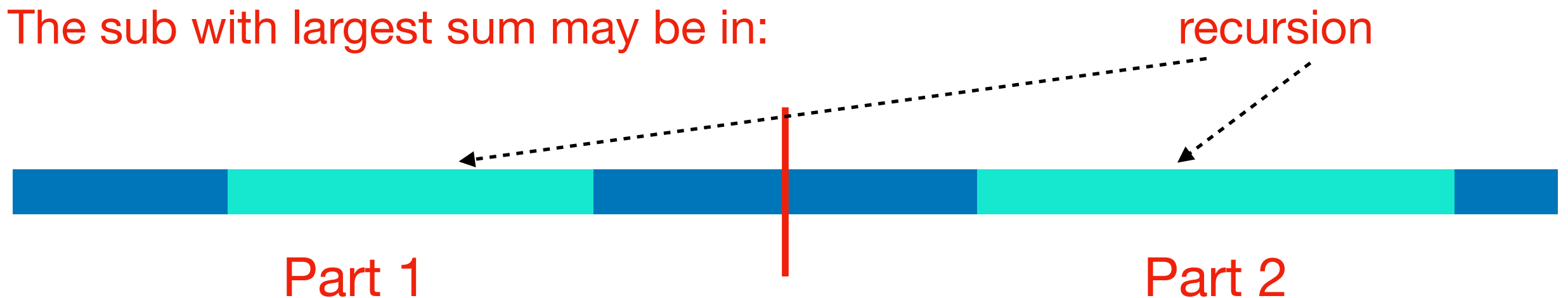
# Examples

Max sum  
subsequence

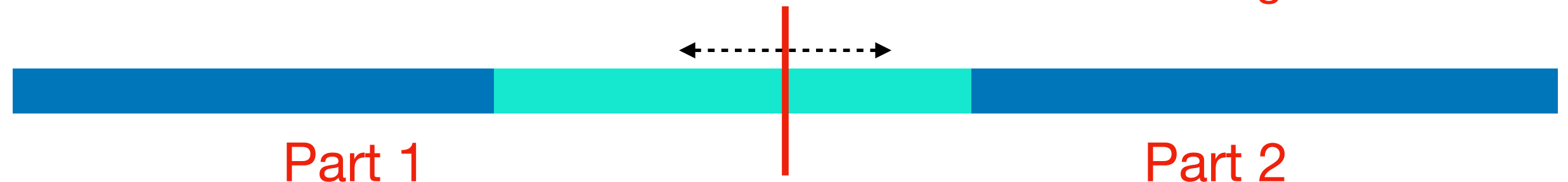
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$



The sub with largest sum may be in:



or:  The largest is the result



# Examples

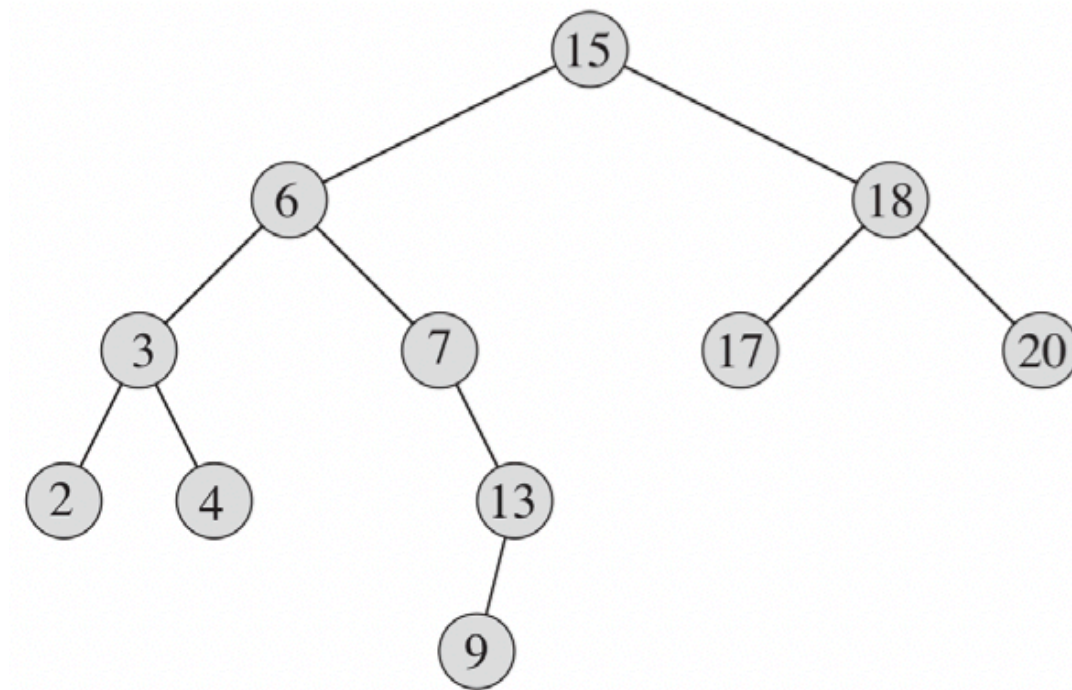
- Maxima
- Frequent element
- Multiplication
  - Integer
  - Matrix
- Nearest point pair

# Examples

- Arrays

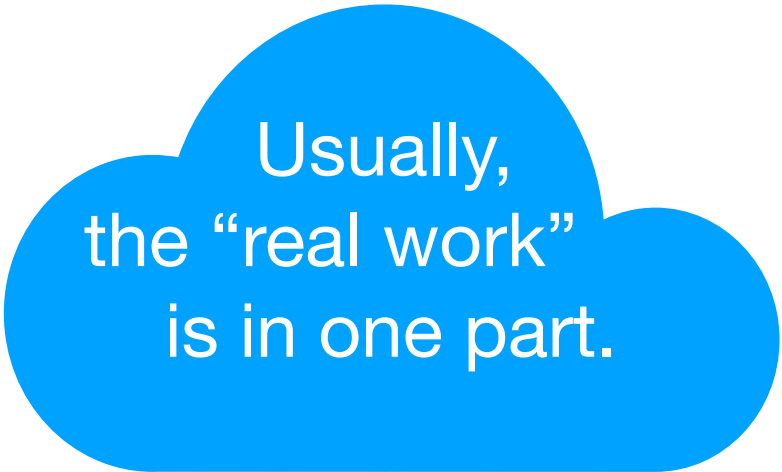
3 5 7 8 9 12 15

- Trees



# Workhorse

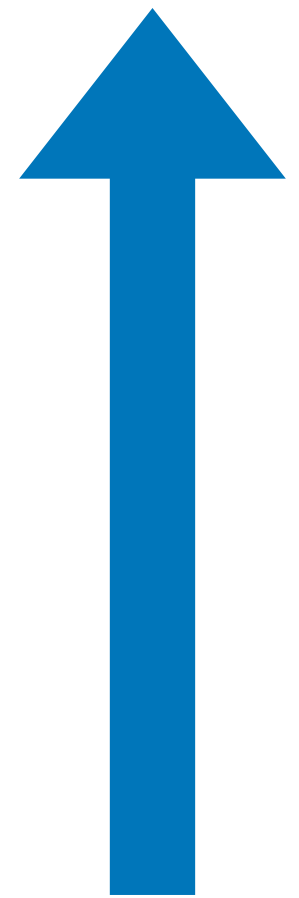
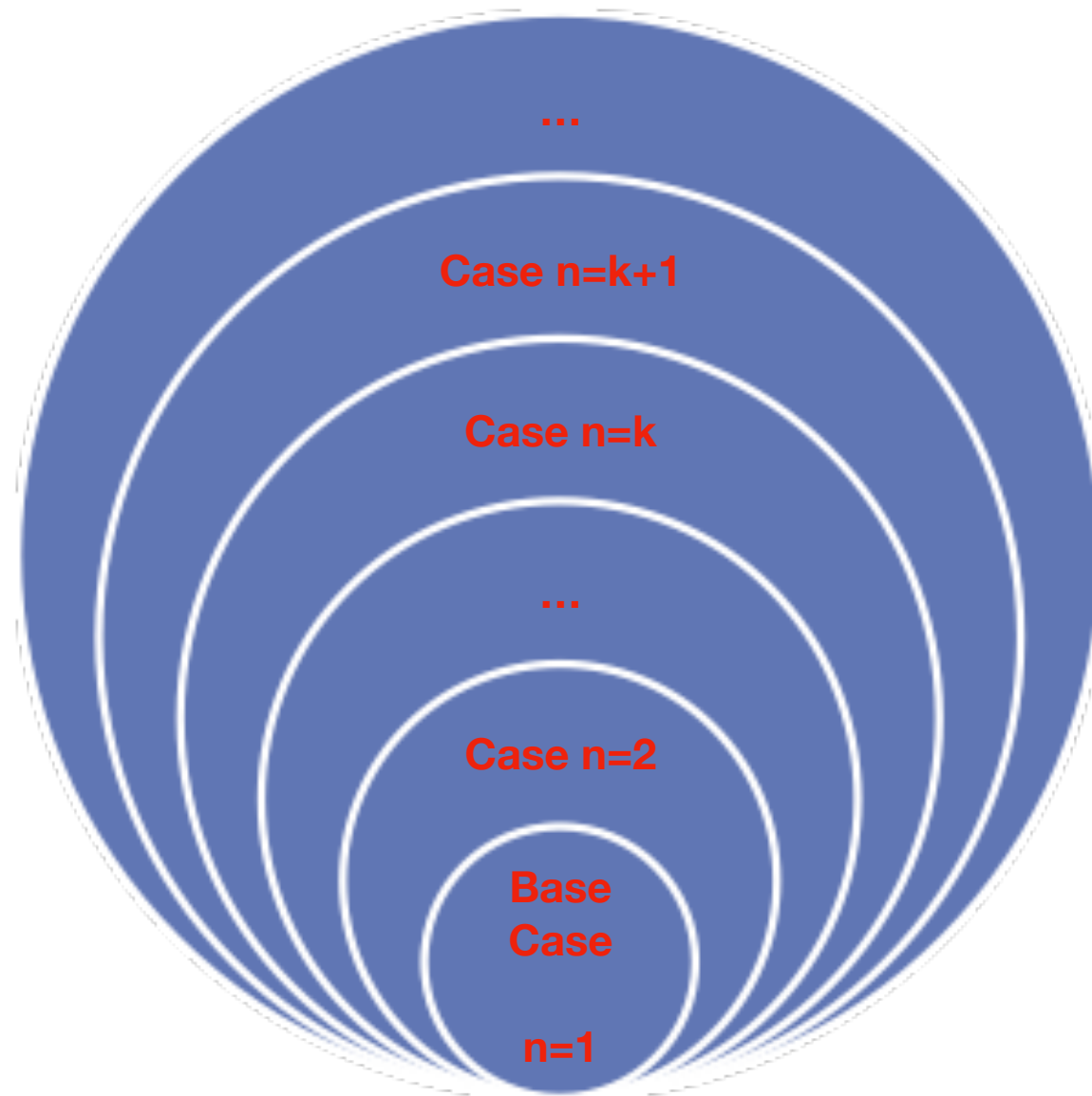
- “Hard division, easy combination”
- “Easy division, hard combination”



Usually,  
the “real work”  
is in one part.

# Correctness of Recursion

Recursion



Induction

# Analysis of Recursion

- Solving recurrence equations
- E.g., Bit counting
  - Critical operation: add
  - The recurrence relation

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\lfloor n/2 \rfloor) + 1 & n > 1 \end{cases}$$

# Analysis of Recursion

- Backward substitutions

By the recursion equation:  $T(n) = T(\lfloor \frac{n}{2} \rfloor) + 1$

For simplicity, let  $n=2^k$  ( $k$  is a nonnegative integer),  
that is,  $k = \log n$

$$T(n) = T(\frac{n}{2}) + 1 = T(\frac{n}{4}) + 1 + 1 = T(\frac{n}{8}) + 1 + 1 + 1 = \dots$$

$$T(n) = T(\frac{n}{2^k}) + \log n = \log n \quad (T(1) = 0)$$

# Smooth Functions

- $f(n)$ 
  - Nonnegative eventually non-decreasing function defined on the set of natural numbers
- $f(n)$  is called smooth
  - If  $f(2n) \in \Theta(f(n))$
- Examples of smooth functions
  - $\log n$ ,  $n$ ,  $n \log n$ , and  $n^a$  ( $a \geq 0$ )
  - E.g.,  $2n \log 2n = 2n(\log n + \log 2) \in \Theta(n \log n)$



# Even Smoother

- Let  $f(n)$  be a smooth function, then, for any fixed integer  $b \geq 2$ ,  $f(bn) \in \Theta(f(n))$ 
  - That is, there exist positive constants  $c_b$  and  $d_b$  and a nonnegative integer  $n_0$  such that

$$d_b f(n) \leq f(bn) \leq c_b f(n) \quad \text{for } n \geq n_0$$

It is easy to prove that the result holds for  $b=2^k$ ,  
For the second inequality:

$$f(2^k n) \leq c_2^k f(n) \quad \text{for } k=1,2,3\dots \text{ and } n \geq n_0$$

For an arbitrary integer  $b \geq 2$ ,  $2^{k-1} \leq b \leq 2^k$

Then,  $f(bn) \leq f(2^k n) \leq c_2^k f(n)$ , we can use  $c_2^k$  as  $c_b$ .

# Smoothness Rule

- Let  $T(n)$  be an eventually non-decreasing function and  $f(n)$  be a smooth function.
  - If  $T(n) \in \Theta(f(n))$  for values of  $n$  that are powers of  $b$  ( $b \geq 2$ ), then  $T(n) \in \Theta(f(n))$ .

Just proving the big - Oh part:

By the hypothesis:  $T(b^k) \leq cf(b^k)$  for  $b^k \geq n_0$

By the prior result:  $f(bn) \leq c_b f(n)$  for  $n \geq n_0$

Let  $n_0 \leq b^k \leq n \leq b^{k+1}$

$$T(n) \leq T(b^{k+1}) \leq cf(b^{k+1}) = cf(bb^k) \leq cc_b f(b^k) \leq cc_b f(n)$$



# Guess and Prove

- Example:  $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Guess

- $T(n) \in O(n)$ ?

Try to prove  $T(n) \leq cn$ :

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \leq 2c(\lfloor n/2 \rfloor) + n \\ &\leq 2c(n/2) + n = (c+1)n, \text{ **Fail!**} \end{aligned}$$

- $T(n) \leq cn$ , to be proved for  $c$  large enough

- $T(n) \in O(n^2)$ ?

However:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \geq 2c\lfloor n/2 \rfloor + n \\ &\geq 2c[(n-1)/2] + n = cn + (n-c) \geq cn \end{aligned}$$

- $T(n) \leq cn^2$ , to be proved for  $c$  large enough

- **Or maybe**,  $T(n) \in O(n \log n)$ ?

- $T(n) \leq cn \log n$ , to be proved for  $c$  large enough

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\ &\leq cn \log(n/2) + n \\ &= cn \log n - cn \log 2 + n \\ &= cn \log n - cn + n \\ &\leq cn \log n \quad \text{for } c \geq 1 \end{aligned}$$

- Prove

- by substitution

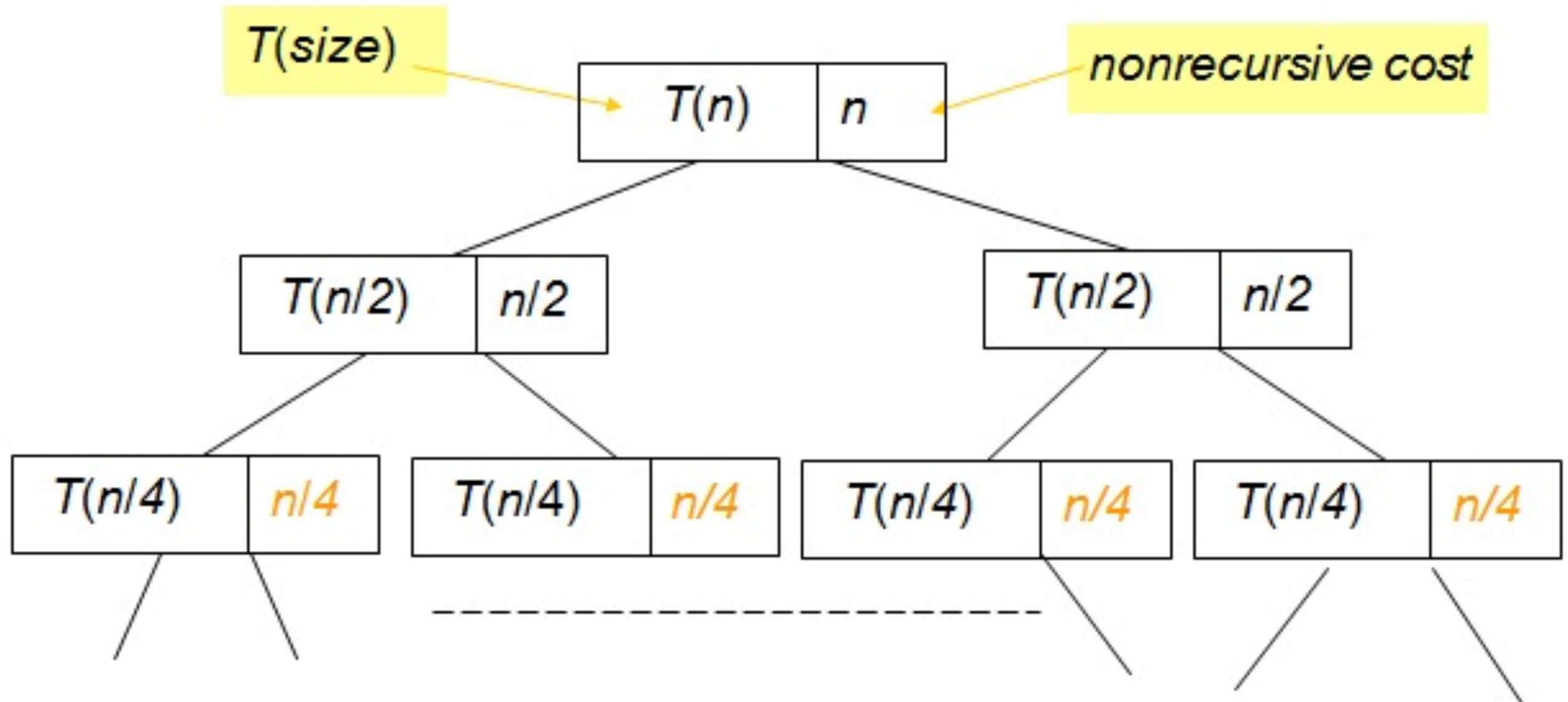
# Divide and Conquer Recursions

- Divide and conquer
  - **Divide** the “big” problem to small ones
  - **Solve** the “small” problems by recursion
  - **Combine** results of small problems, and solve the original problem
- Divide and conquer recursion

$$T(n) = bT(n/c) + f(n)$$



# Recursion Tree

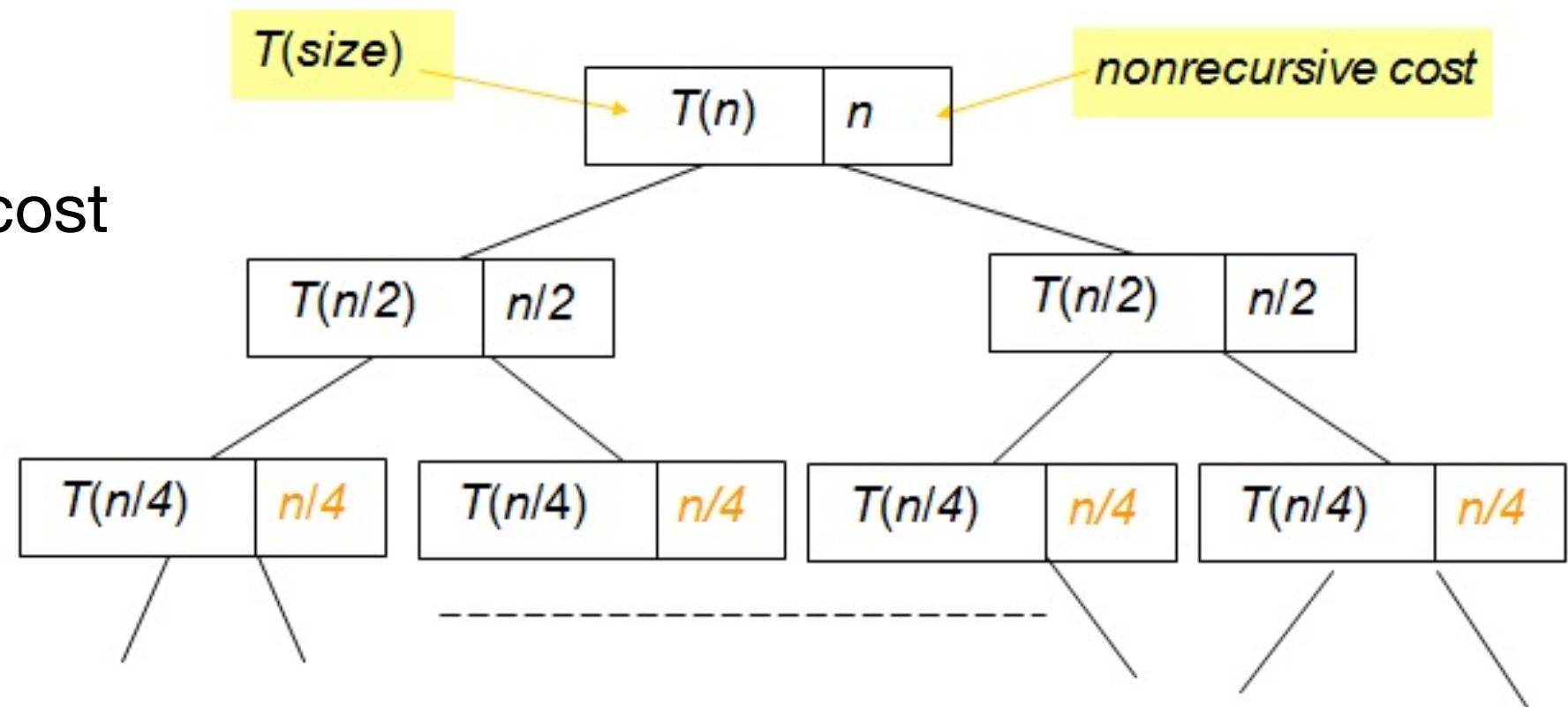


The recursion tree for  $T(n) = T(n/2) + T(n/2) + n$

# Recursion Tree

- Node

- Non-leaf
  - Non-recursive cost
  - Recursive cost



The recursion tree for  $T(n) = T(n/2) + T(n/2) + n$

- Leaf
  - Base case
- Edge
  - Recursion

# Recursion Tree

Recursive cost

Non-recursive cost

$$T(n) = \underbrace{3T(\lfloor n/4 \rfloor)}_{\text{\# of sub-problems}} + \underbrace{\Theta(n^2)}_{\text{size of sub-problems}}$$

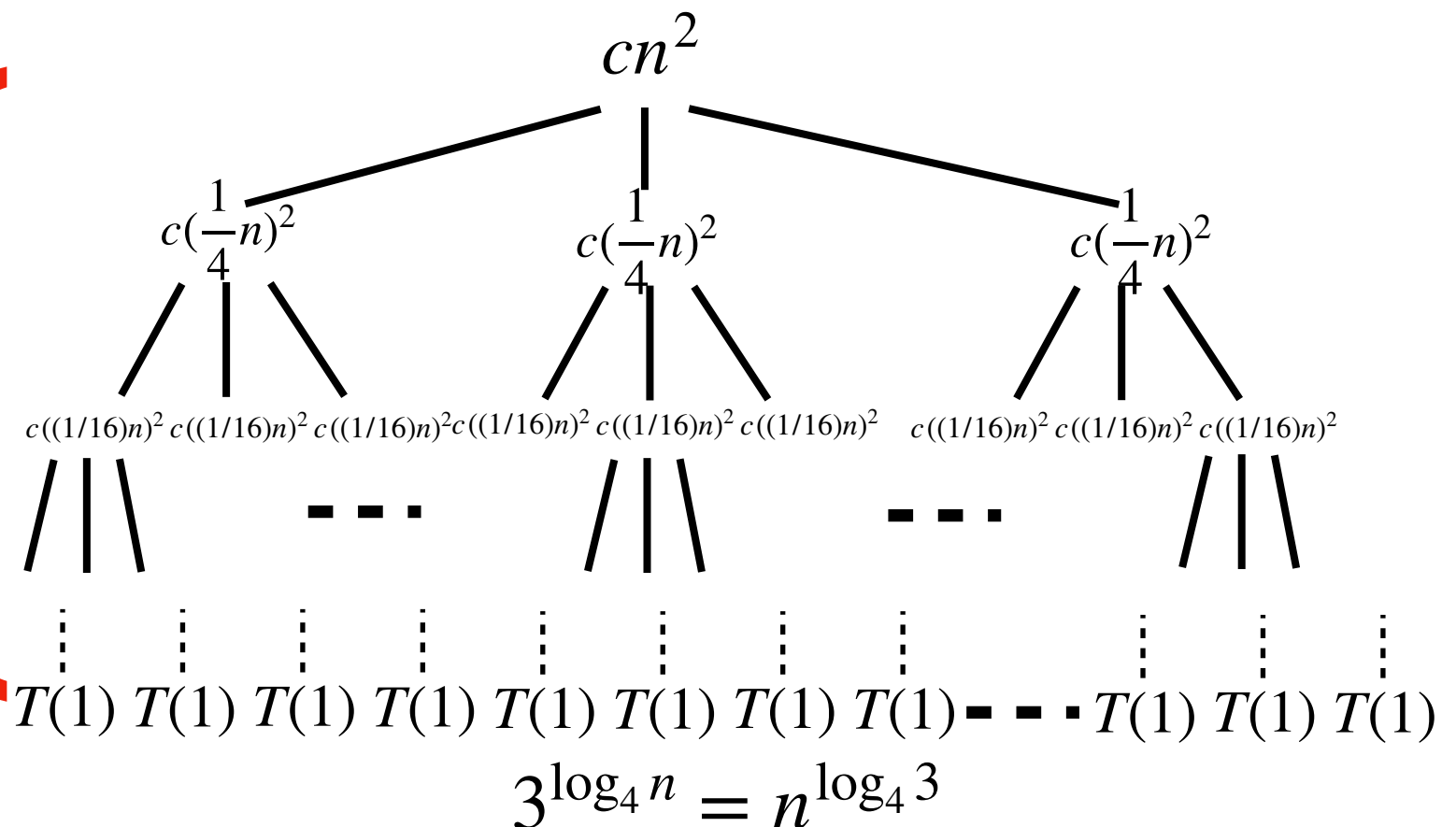
# of sub-problems

size of sub-problems

Total cost

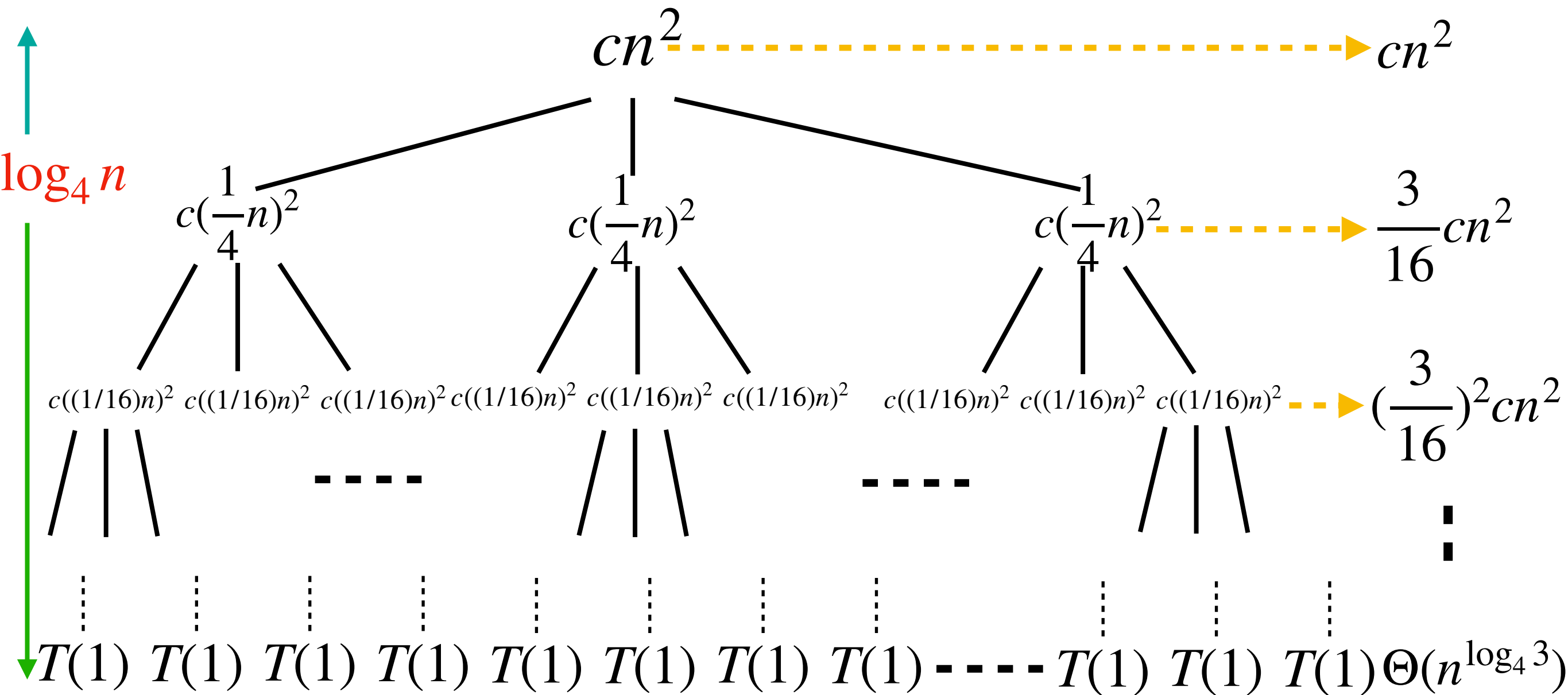
$\Sigma$

Sum of row sums



# Recursion Tree for

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$



Note:  $3^{\log_4 n} = n^{\log_4 3}$

**Total?**

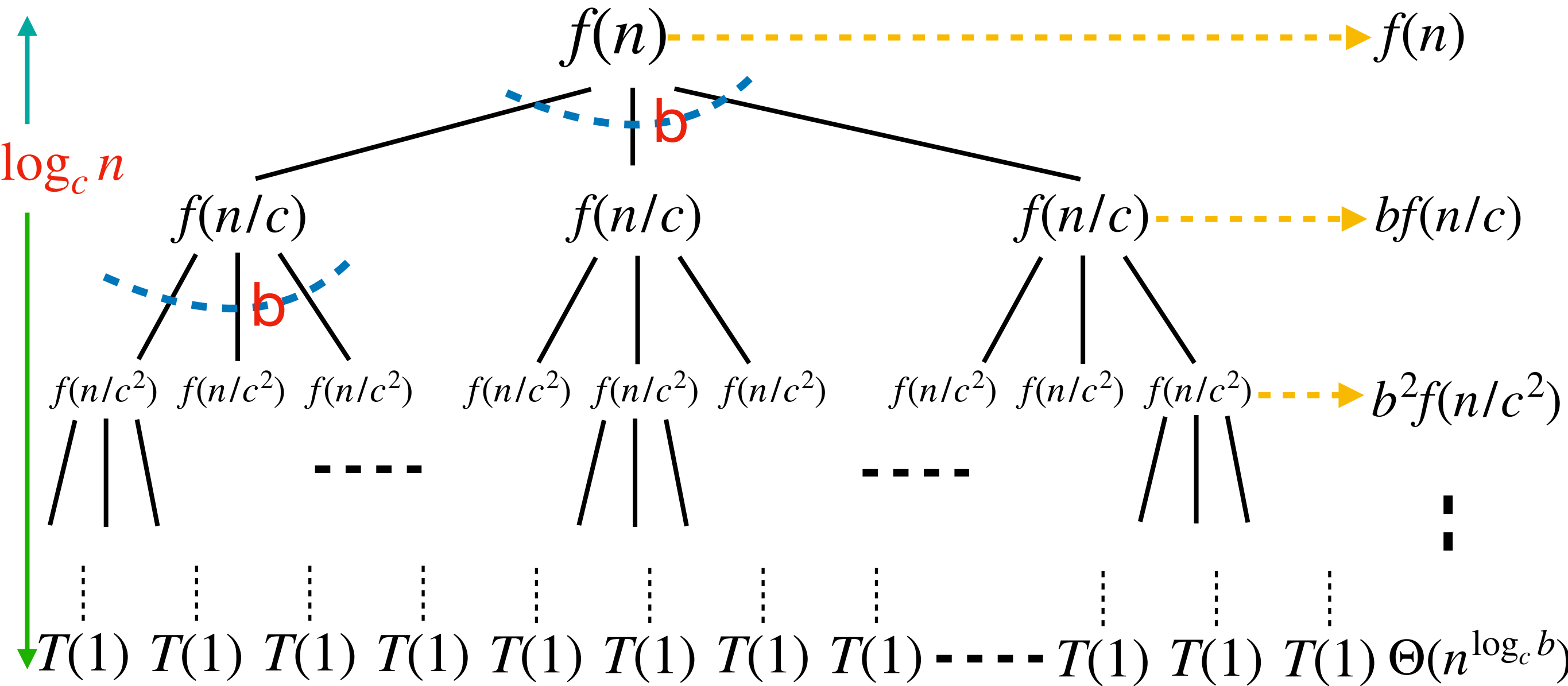


# Solving the Divide-and-Conquer Recurrence

- The recursion equation for divide-and-conquer, the general case:  $T(n) = bT(n/c) + f(n)$
- Observations:
  - Let base-cases occur at depth  $D$  (leaf), then  $n/c^D = 1$ , that is  $D = \log(n)/\log(c)$
  - Let the number of leaves of the tree be  $L$ , then  $L = b^D$ , that is  $L = b^{\log(n)/\log(c)}$
  - By a little algebra:  $L = n^E$ , where  $E = \log(b)/\log(c)$ , called **critical exponent**.

# Recursion Tree for

$$T(n) = bT(n/c) + f(n)$$



**Note:**  $b^{\log_c n} = n^{\log_c b}$

# Total?

# Divide-and-Conquer - the Solution

- The solution of divide-and-conquer equation is the non-recursive costs of all nodes in the tree, which is the sum of the row-sums
  - The recursion tree has depth  $D = \log(n)/\log(c)$ , so there are about that many row-sums.
- The  $0^{\text{th}}$  row-sum
  - is  $f(n)$ , the non recursive cost of the root.
- The  $D^{\text{th}}$  row-sum
  - is  $n^E$ , assuming base cases 1, or  $\Theta(n^E)$  in any event.

# Solution by Row-sums

- [Little Master Theorem] Row-sums decide the solution of the equation for divide-and-conquer:
  - Increasing geometric series:  $T(n) \in \Theta(n^E)$
  - Constant:  $T(n) \in \Theta(f(n)\log n)$
  - Decreasing geometric series:  $T(n) \in \Theta(f(n))$

This can be generalized to get a result not using explicitly row-sums.

# Master Theorem

- Loosening the restrictions on  $f(n)$

- Case 1:  $f(n) \in O(n^{E-\varepsilon})$ , ( $\varepsilon > 0$ ), then:

$$T(n) \in \Theta(n^E)$$

- Case 2:  $f(n) \in \Theta(n^E)$ , as all node depth contribute about equally:

$$T(n) \in \Theta(f(n)\log(n))$$

- Case 3:  $f(n) \in \Omega(n^{E+\varepsilon})$ , ( $\varepsilon > 0$ ), and of  $bf(n/c) \leq \theta f(n)$  for some constant  $\theta < 1$  and all sufficiently large  $n$ , then:

$$T(n) \in \Theta(f(n))$$

**The positive  $\varepsilon$  is critical, resulting gaps between cases as well.**

# Using Master Theorem

- Example 1:  $T(n) = 9T(\frac{n}{3}) + n$   
 $b = 9, c = 3, E = 2, f(n) = n = O(n^{E-1})$   
Case 1 applies:  $T(n) = \Theta(n^2)$
- Example 2:  $T(n) = T(\frac{2}{3}n) + 1$   
 $b = 1, c = \frac{3}{2}, E = 0, f(n) = 1 = \Theta(n^E)$   
Case 2 applies:  $T(n) = \Theta(\log n)$
- Example 3:  $T(n) = 3T(\frac{n}{4}) + n \log n$   
 $b = 3, c = 4, E = \log_4 3, f(n) = n \log n = \Omega(n^{E+\epsilon})$   
Case 3 applies:  $T(n) = \Theta(n \log n)$

# Using Master Theorem

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

Does Case 3 apply? Why?

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

- The gap between the 3 cases
  - Often, non of the 3 cases apply
  - Your task: design more non-solvable recursions

Thank you!

Q & A