

# L02.01

## sorting, master theorem

50.004 Introduction to Algorithm

Gemma Roig

(slides adapted from Dr. Simon LUI)

ISTD, SUTD

# L02.01

## sorting, master theorem

Pre- or post- readings:

Introduction to Algorithms CLRS book chapters:

Chapter 2 and Chapter 4

# Objective

- Explain the sorting problem
  - Insertion sort is  $O(n^2)$
  - Merge sort is  $O(n \log n)$  (by divide and conquer)
- Analyze complexity of recursions
  - By expansion: the recursion tree method
  - By induction: the substitution method

# Sorting problem

- Input: an array  $A[0..n-1]$  of numbers
- Output:  $B$ . (a **permutation** of  $A$ ) such that

$$B[0] \leq B[1] \leq \cdots \leq B[n]$$

- in-place sort: if the sorted item occupy the same storage as the original one.
- Out-of-space sort: if the sorting algorithm used extra space to do the sorting

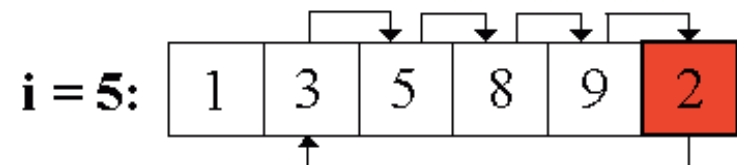
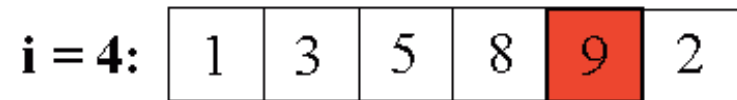
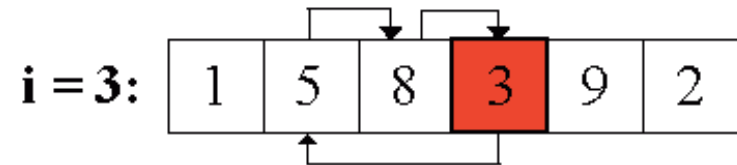
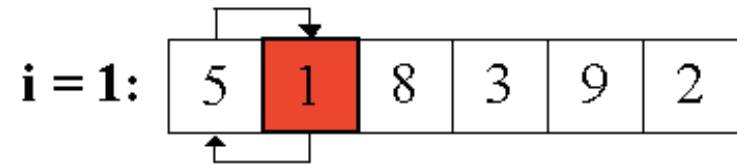
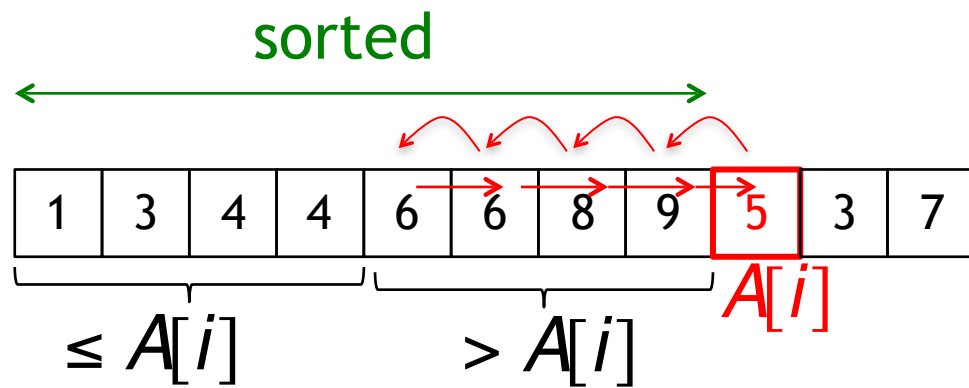
# Insertion sort

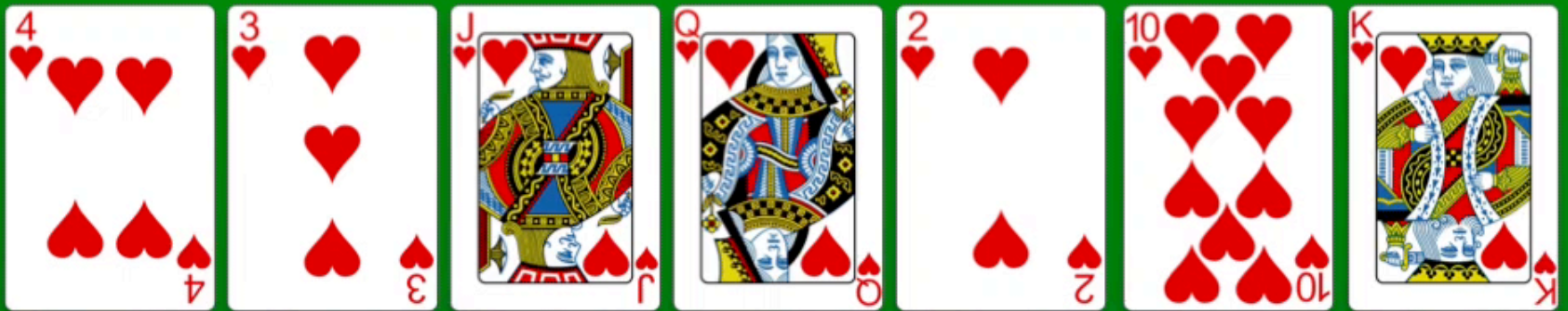
# Insertion sort

Principle:  $A[0:i-1]$  is sorted

Then, put  $A[i]$  in the right position

- For  $i$  in range(1,  $n$ ):  
  while  $A[i] < A[i-1]$ :  
    swap  $A[i]$  with  $A[i-1]$   
   $i -= 1$





# Insertion Sort

Another video example:

[https://www.youtube.com/  
watch?v=mPEBjhl6oAU](https://www.youtube.com/watch?v=mPEBjhl6oAU)

Min 1:03



# Complexity of insertion sort

- **Worst-case** running time  $T(n)$  on an input of size  $n$

- $T(n)$  =  $i$  comparisons and swaps at step  $i$

$$= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)$$

# Divide and Conquer - revision

# Divide and conquer solution

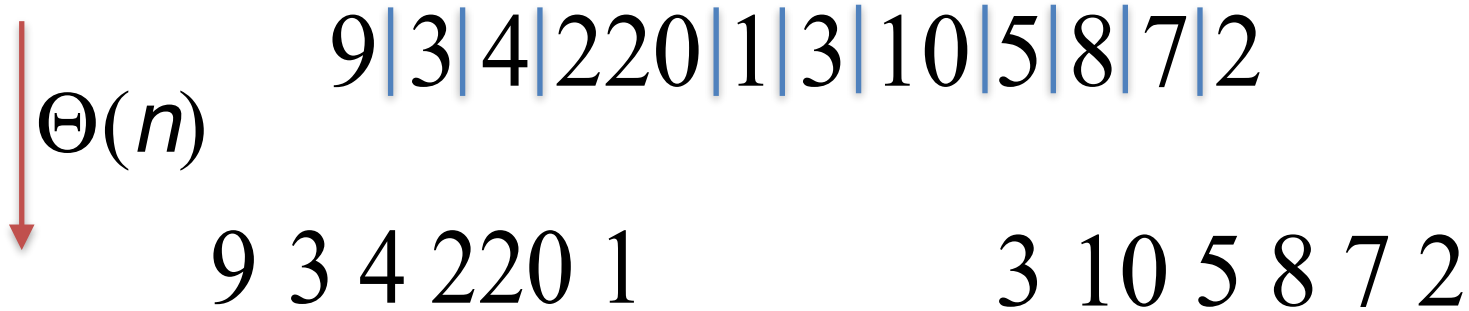
- Key idea:
  - **Divide** input into parts (smaller problems)
  - **Conquer** (solve) each part recursively
  - **Combine** results to obtain solution of original

$$\begin{aligned} T(n) = & \text{divide time} \\ & + T(n_1) + T(n_2) + \dots + T(n_k) \\ & + \text{combine time} \end{aligned}$$

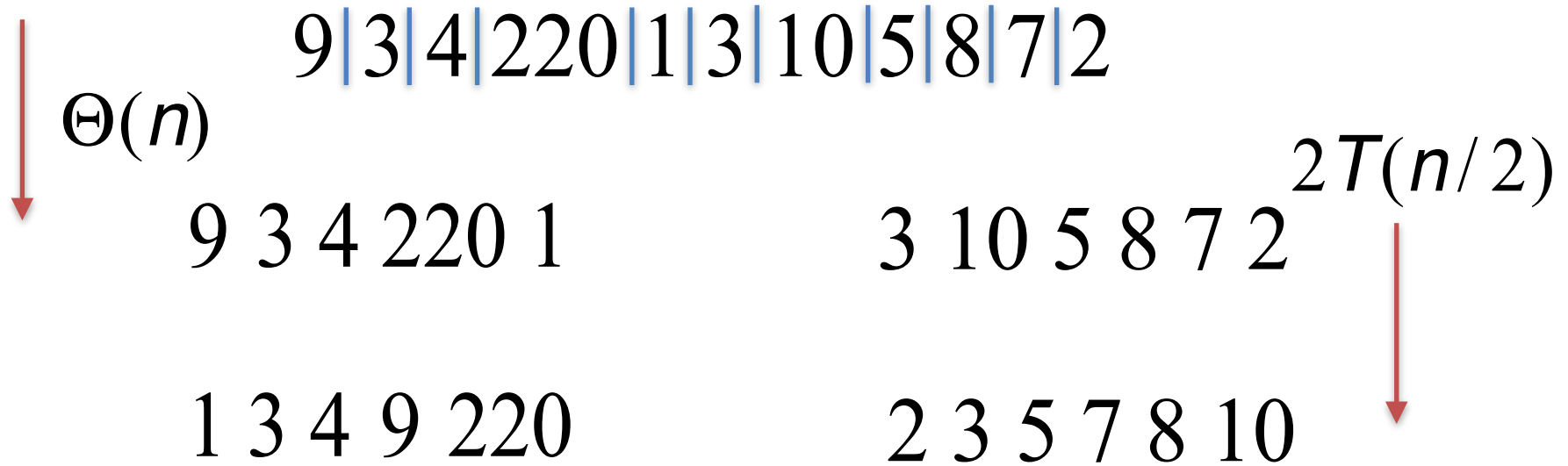
# Lets construct the recursion!

9 3 4 220 1 3 10 5 8 7 2

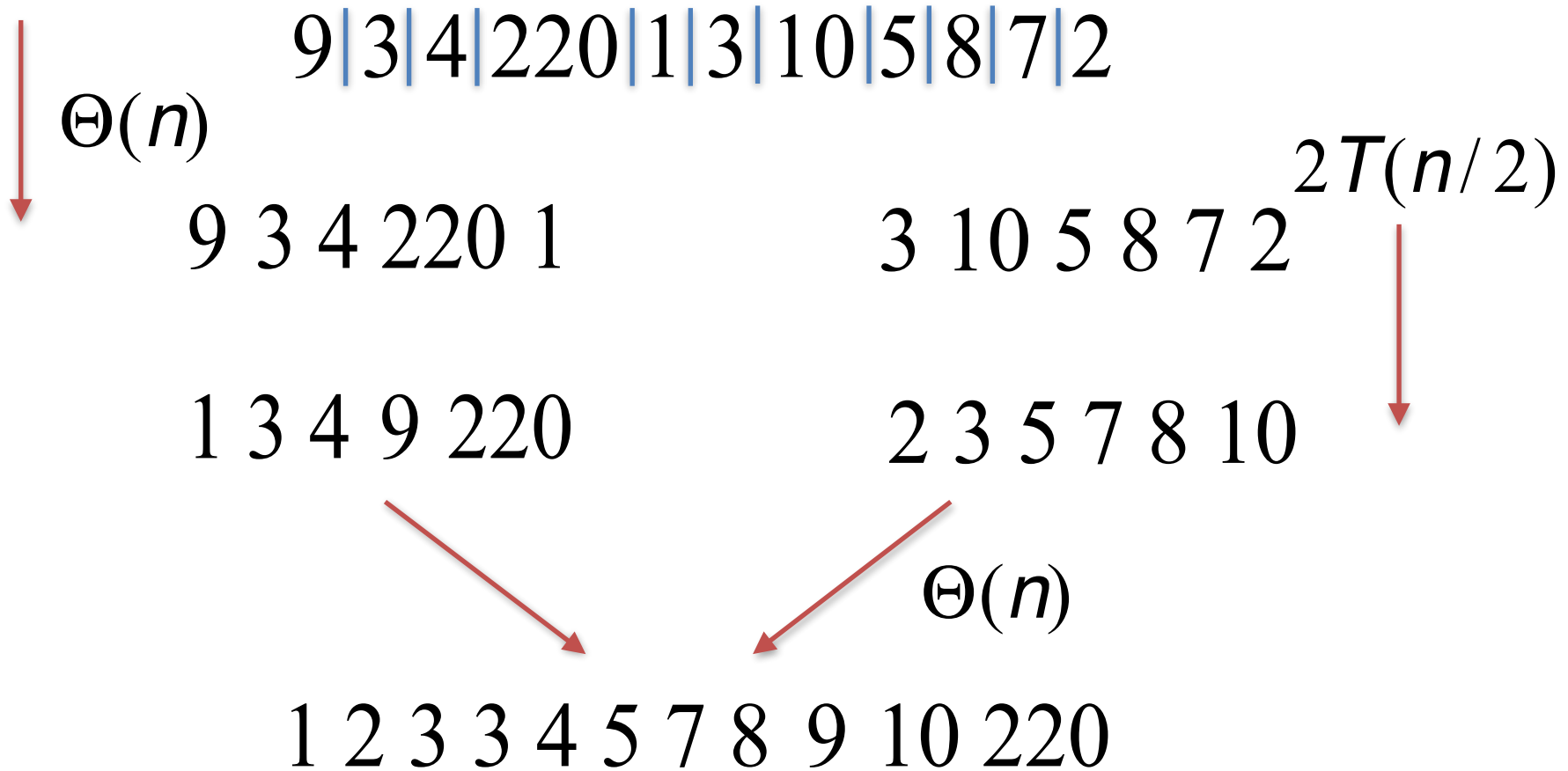
# Lets construct the recursion!



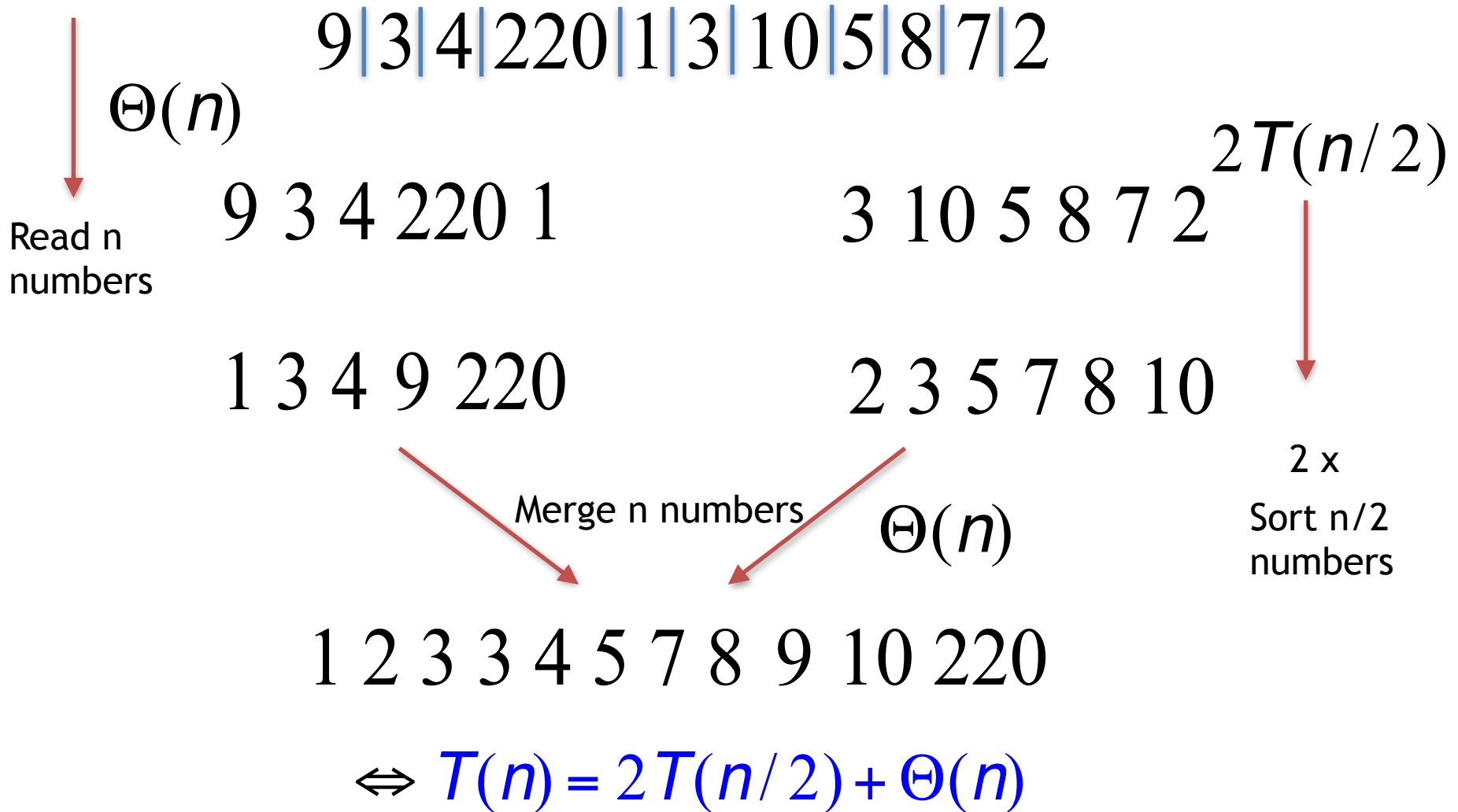
# Lets construct the recursion!



# Lets construct the recursion!



# Lets construct the recursion!





# ... Syntax for your reference

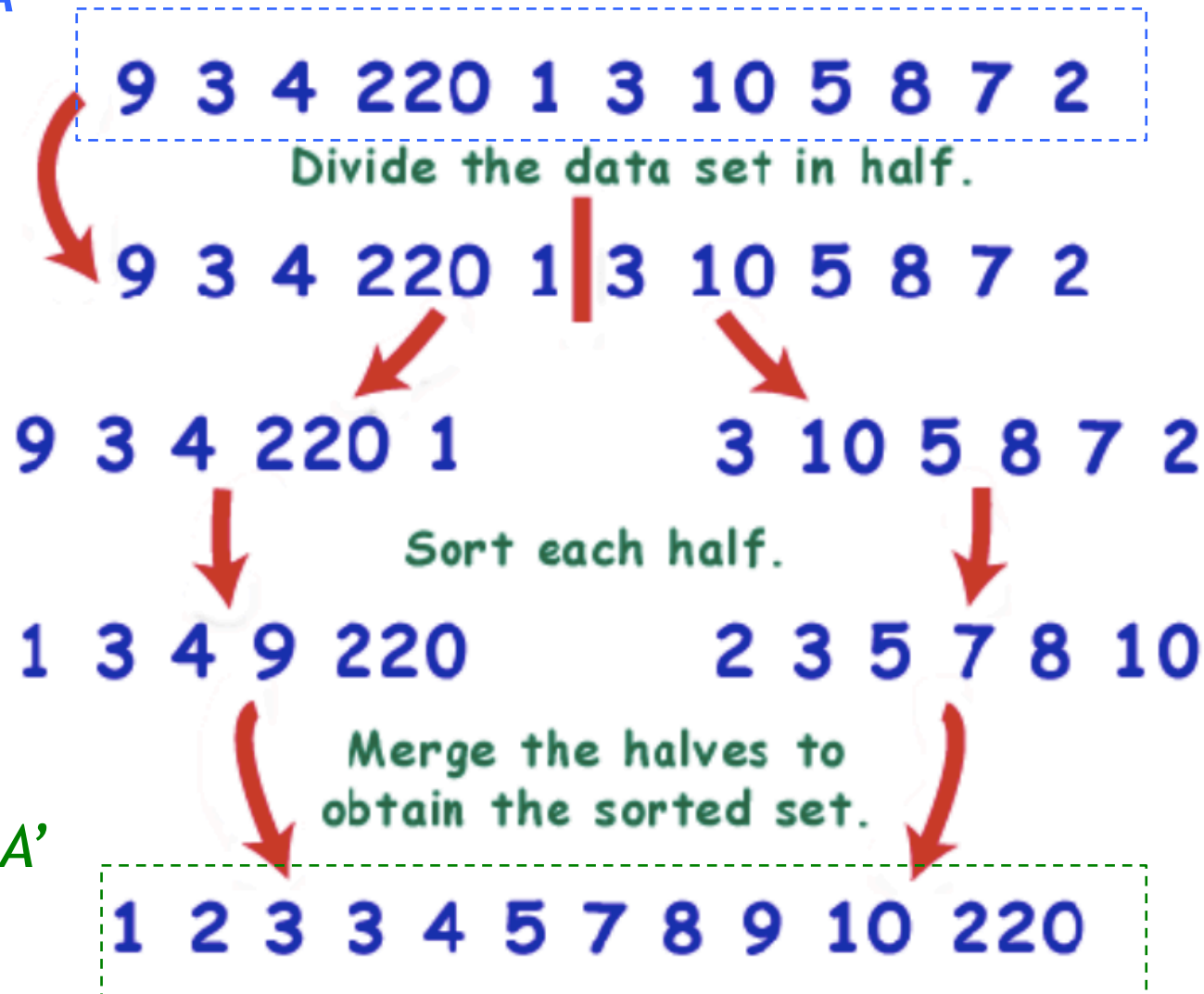
$A = [2, 4, 6, 8, 10]$

- $A[] = [2, 4, 6, 8, 10]$
- $A[0] = [2]$
- $A[1] = [4]$
- $A[1:3] = [4, 6, 8]$
- $A[0:2] = [2, 4, 6]$
- $A[:2] = [2, 4, 6]$
- $A[1:] = [4, 6, 8, 10]$

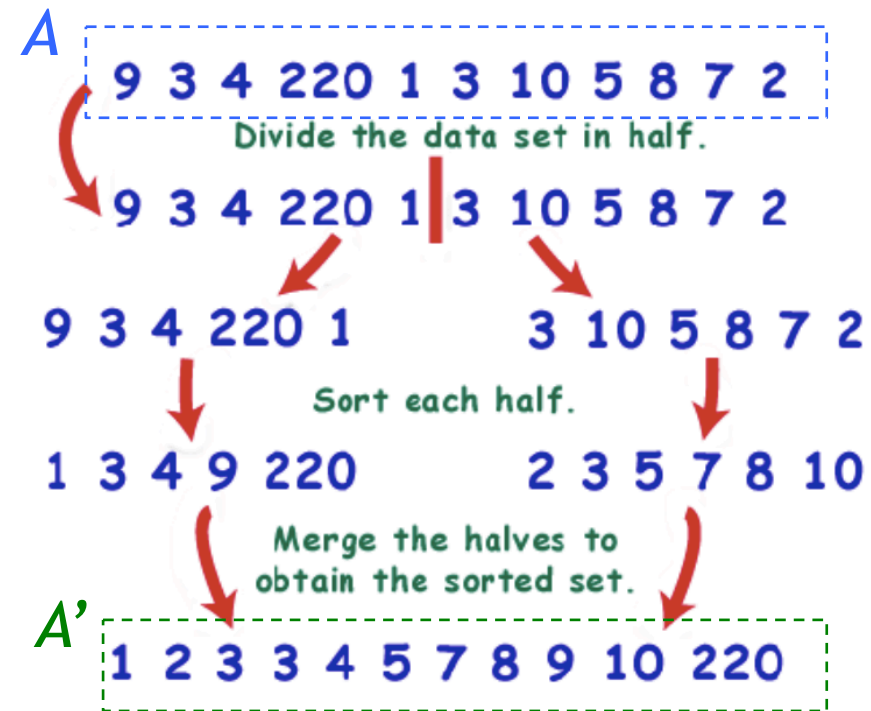
# Merge sort

# Merge sort

A



# Merge sort



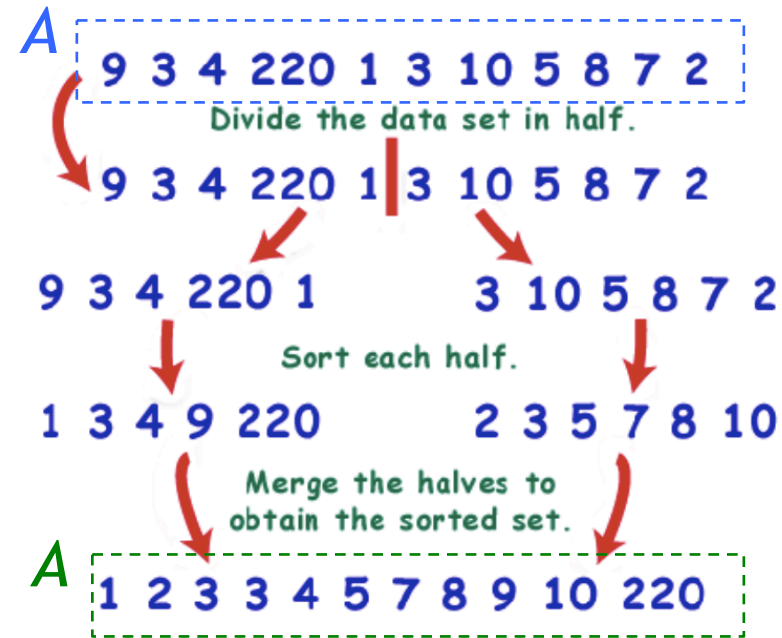
MergeSort(A):

- if  $n=1$ : done -> return
- recursively sort  $A[:n/2]$  -> L
- recursively sort  $A[n/2:]$  -> R
- merge L & R -> output A'

# Merge sort

MergeSort(A):

- if  $n=1$ : **done** -> **return**
- recursively sort  $A[:n/2]$  ->  $L$
- recursively sort  $A[n/2:]$  ->  $R$
- merge  $L$  &  $R$  -> **output  $A'$**



Time:

1. **divide:**  $\Theta(n)$

2. **recursion:**  $n_1=n_2=n/2$

$$\text{time} = T(n_1) + T(n_2) = 2T(n/2)$$

3. **merge:**  $\Theta(n)$

$$\text{total: } T(n) = 2T(n/2) + \Theta(n)$$

# Merge sort

[https://www.youtube.com/  
watch?v=mPEBjhl6oAU](https://www.youtube.com/watch?v=mPEBjhl6oAU)

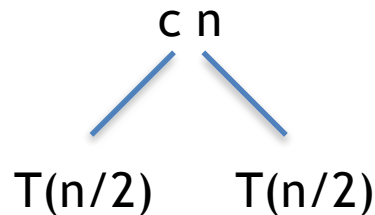
Min 1:53

Solve the recurrence formula

# Solving recurrences by expansion (recursion tree)

- The recursion tree = sum of cost of nodes
- Example 1, Let's draw the tree of:

$$T(n) = 2T(n/2) + cn \quad c \text{ is a constant}$$

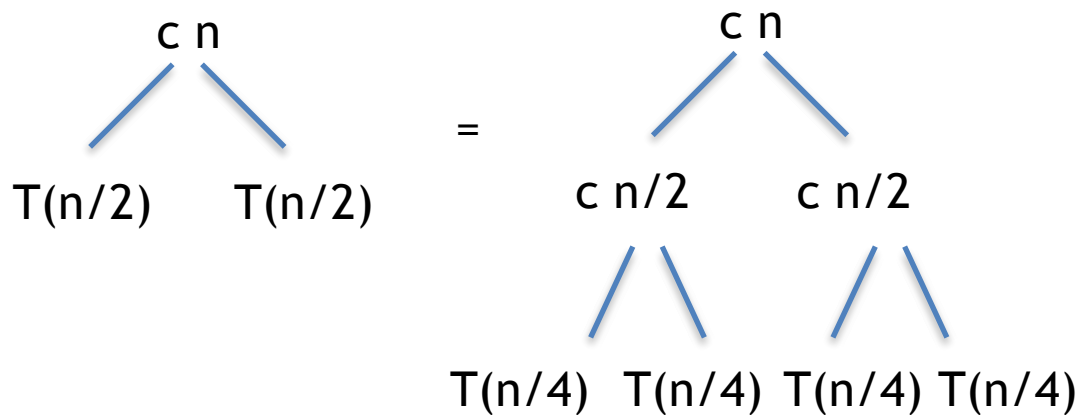




# Solving recurrences by expansion (recursion tree)

- The recursion tree = sum of cost of nodes
- Example 1, Let's draw the tree of:

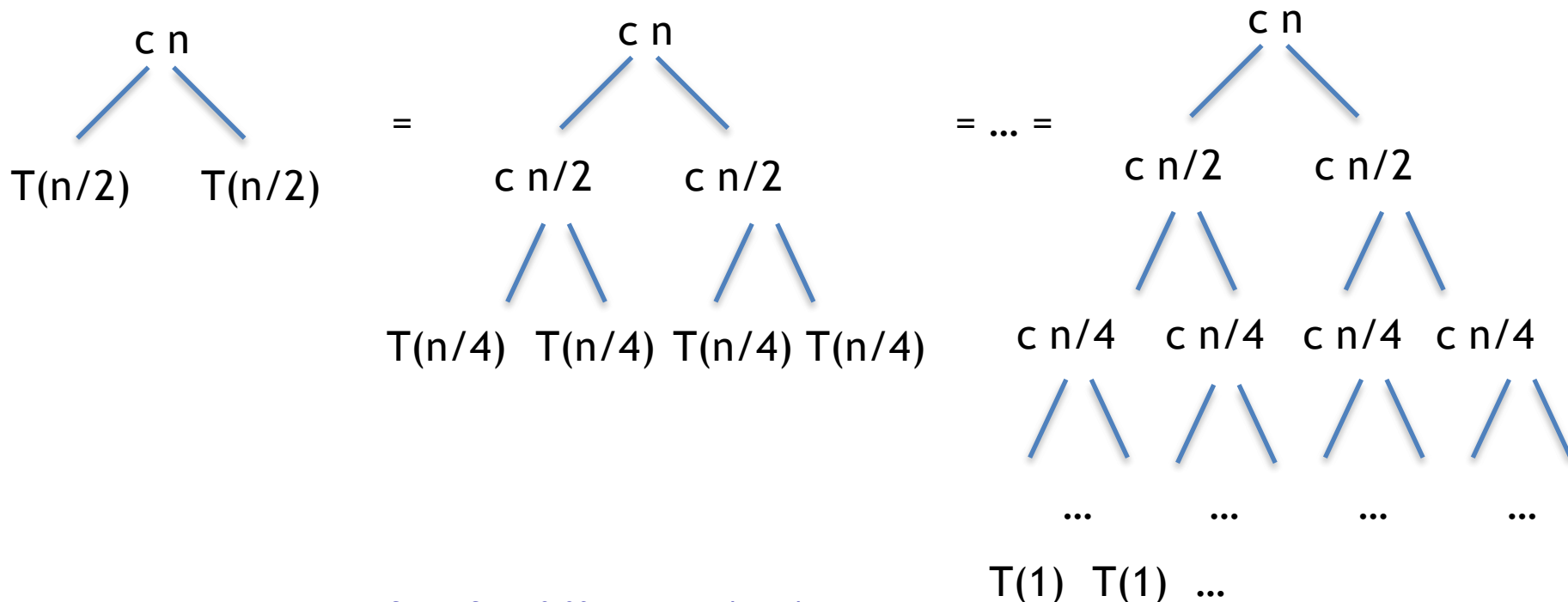
$$T(n) = 2T(n/2) + cn \quad c \text{ is a constant}$$



# Solving recurrences by expansion (recursion tree)

- The recursion tree = sum of cost of nodes
- Example 1, Let's draw the tree of:

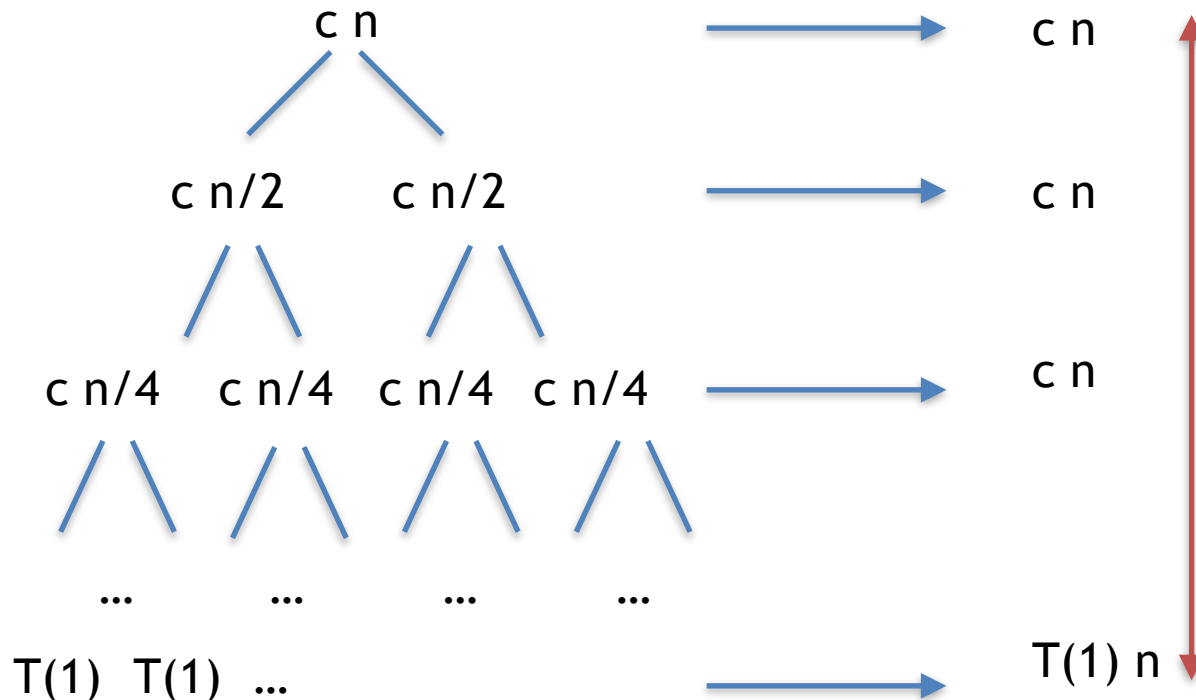
$$T(n) = 2T(n/2) + cn \quad c \text{ is a constant}$$



# Solving recurrences by expansion (recursion tree)

- The recursion tree = sum of cost of nodes
- Example 1, Let's draw the tree of:

$$T(n) = 2T(n/2) + cn \quad c \text{ is a constant}$$



Height of the tree:  
 $\log(n)$  levels

Sum each level,  
then sum all levels:

$$\begin{aligned} T(n) &= nT(1) + cn \log n \\ &= \Theta(n \log n) \end{aligned}$$

# Solution by expansion (recursion tree)

- Example 2, The recurrence tree of

$$T(n) = T(n/2) + cn$$



$$\begin{aligned} T(n) &= cn + cn/2 + cn/4 + \dots + 1 \\ &= cn (1 + 1/2 + 1/4 + \dots + 1/2^{\log n}) \\ &= cn (2) \\ &= \Theta(n) \end{aligned}$$

# Master Theorem

# Can we **generalize** our observations?

There are three cases

- Each level's value is the same
  - Total value =  $\Theta(\text{number of levels} \times \text{sum of each level})$
- Each level's value is decreasing
  - total value =  $\Theta(\text{root node's value})$
- Each level's value is increasing
  - total value =  $\Theta(\text{number of leaves})$ 
    - Since each level node is  $\Theta(1)$

# What is master theorem

Consider  $T(n) = aT(n/b) + f(n)$

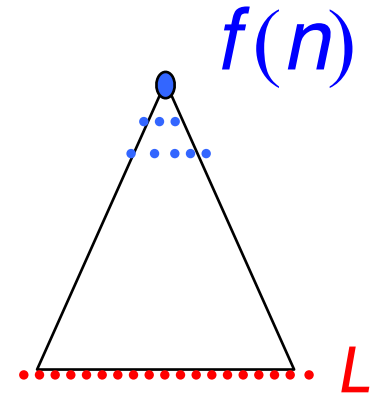
- The master Theorem can find the complexity of  $T(n)$ , according to  $f(n)$

# The Master Theorem

Consider  $T(n) = aT(n/b) + f(n)$

$\Rightarrow h = \# \text{ of levels} = \log_b n = \Theta(\log n)$

$\Rightarrow L = \# \text{ of leaves} = a^h = a^{\log_b n} = n^{\log_b a}$



1) if  $f(n) = O(L^{1-\epsilon}) = O(n^{\log_b a - \epsilon})$

$\Rightarrow T(n) = \Theta(L) = \Theta(n^{\log_b a})$

i.e. value of  $f(n)$  is geometrically increasing down the tree

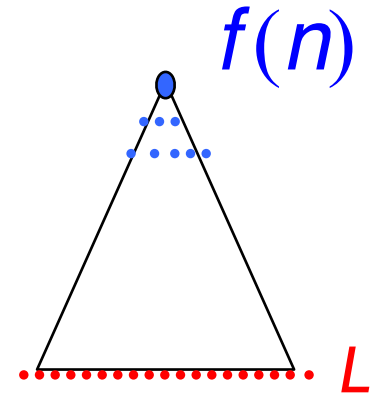


# The Master Theorem

Consider  $T(n) = aT(n/b) + f(n)$

$\Rightarrow h = \# \text{ of levels} = \log_b n = \Theta(\log n)$

$\Rightarrow L = \# \text{ of leaves} = a^h = a^{\log_b n} = n^{\log_b a}$



2) if  $f(n) = \Theta(L) = \Theta(n^{\log_b a})$

$\Rightarrow T(n) = \Theta(L \log n) = \Theta(n^{\log_b a} \log n)$

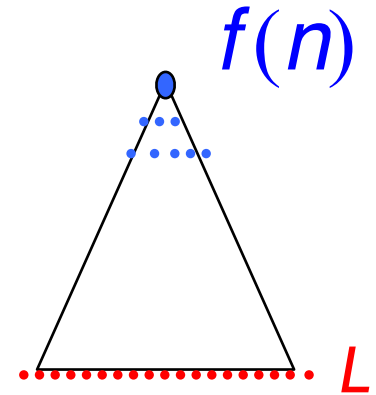
i.e value of each level  
are (roughly) equal

# The Master Theorem

Consider  $T(n) = aT(n/b) + f(n)$

$\Rightarrow h = \# \text{ of levels} = \log_b n = \Theta(\log n)$

$\Rightarrow L = \# \text{ of leaves} = a^h = a^{\log_b n} = n^{\log_b a}$



3) if  $f(n) = \Omega(L^{1+\epsilon}) = \Omega(n^{\log_b a + \epsilon})$

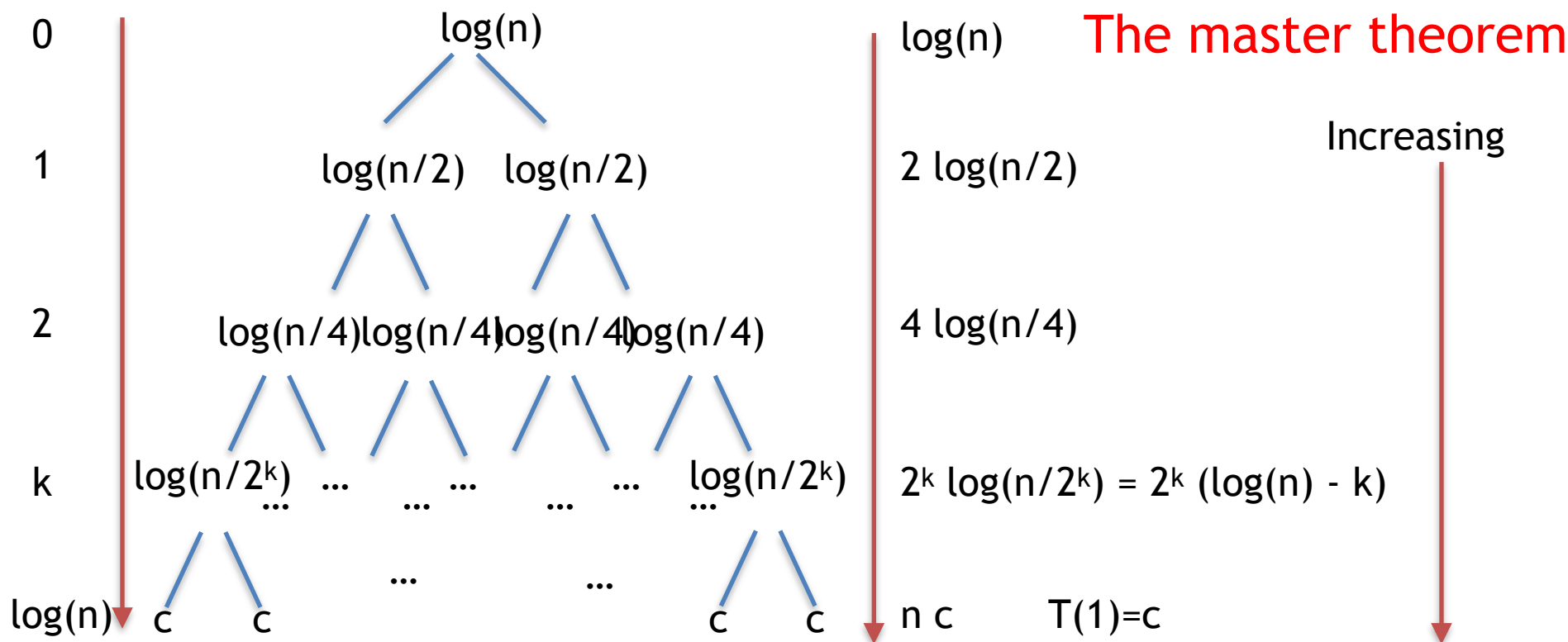
$\Rightarrow T(n) = \Theta(f(n))$

i.e. the value of  $f(n)$   
Is geometrically decreasing  
down the tree

# Examples

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

$$1) \text{ if } f(n) = O(L^{1-\varepsilon}) = O(n^{\log_b a - \varepsilon}) \\ \Rightarrow T(n) = \Theta(L) = \Theta(n^{\log_b a})$$



$$f(n) = \log n, a = 2, b = 2$$

The values of each level are geometrically increasing down the tree. So,  $T(n) = \Theta(n)$

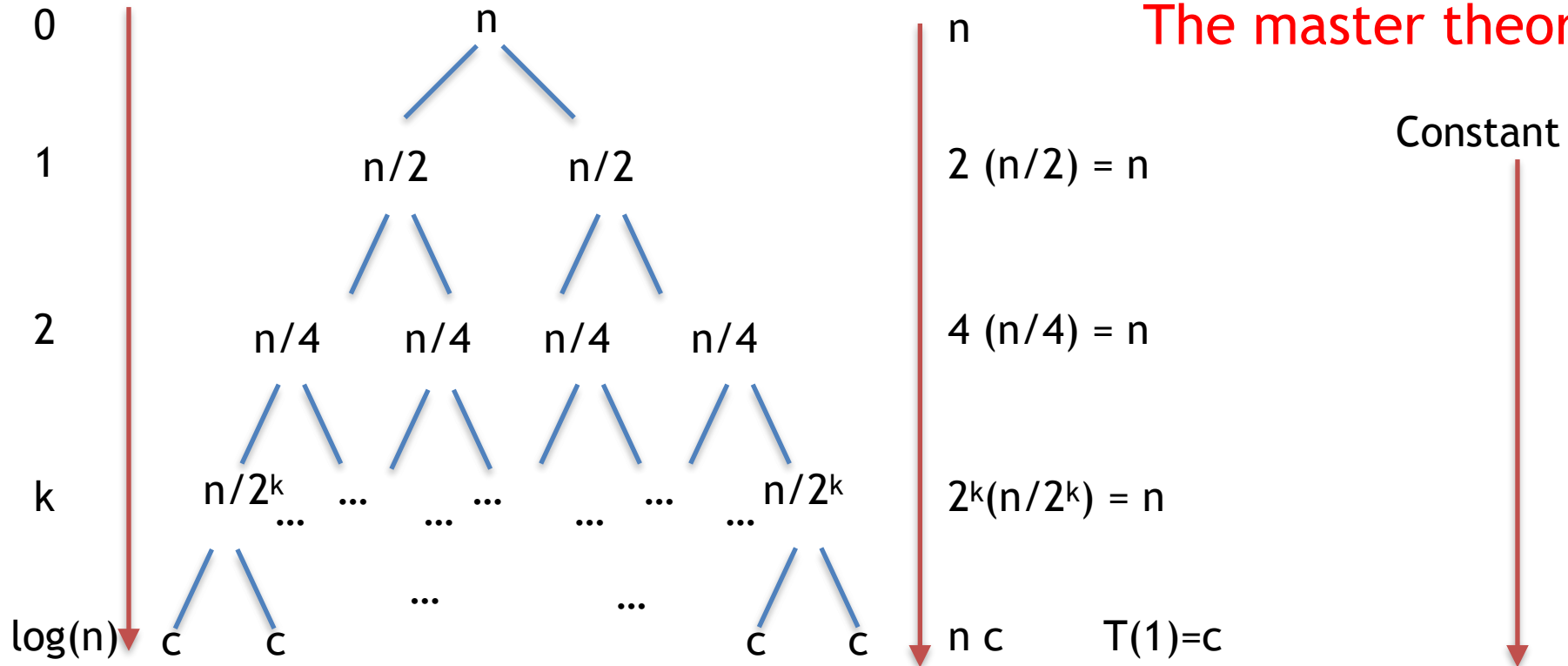
# Examples

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

$$2) \text{ if } f(n) = \Theta(L) = \Theta(n^{\log_b a})$$

$$\Rightarrow T(n) = \Theta(L \log n) = \Theta(n^{\log_b a} \log n)$$

The master theorem



$$f(n) = n, a = 2, b = 2$$

The values of each level are geometrically equal

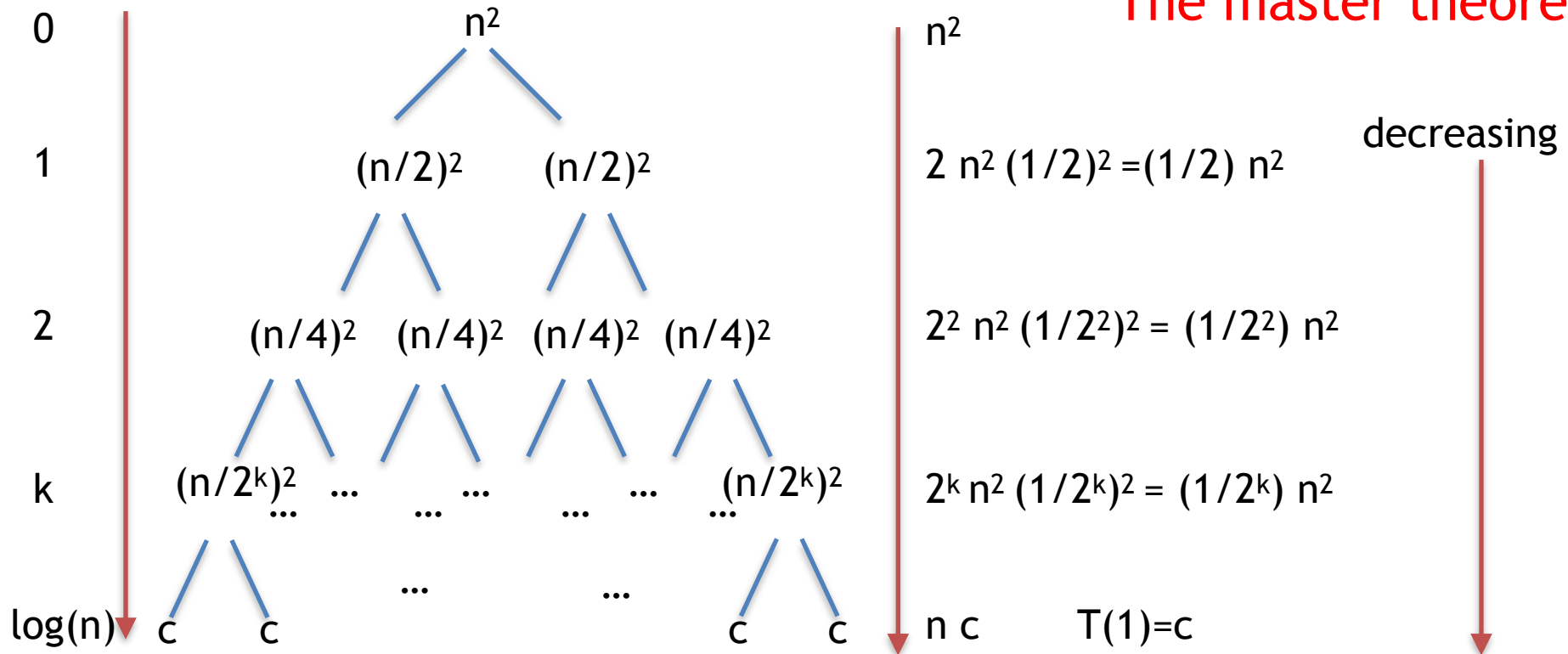
$$\text{So, } T(n) = \Theta(n \log n)$$

# Examples

$$T(n) = 2T(n/2) + n^2$$

$$3) \text{ if } f(n) = \Omega(L^{1+\varepsilon}) = \Omega(n^{\log_b a + \varepsilon}) \\ \Rightarrow T(n) = \Theta(f(n))$$

The master theorem



$$f(n) = n^2, a = 2, b = 2$$

The values of each level are geometrically decreasing down the tree

So,  $T(n) = \Theta(n^2)$