

A horizontal teal brushstroke with irregular, hand-painted edges, serving as a background for the title.

Getting Started

Heejin Park

Hanyang University

Contents

- Sorting problem
- 2 sorting algorithms
 - Insertion sort
 - Merge sort

Sorting problem

keys



Input

- A sequence of n number $\langle a_1, a_2, \dots, a_n \rangle$.

Output

- A permutation (reordering) $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Ex>

- Input: $\langle 5, 2, 4, 6, 1, 3 \rangle$
- Output: $\langle 1, 2, 3, 4, 5, 6 \rangle$

Insertion sort

- **Insertion sort**
 - Description
 - Correctness
 - Performance

Description

- What is insertion sort?
 - A sorting algorithm using **insertion**.
- What is insertion?
 - Given **a key** and **a sorted list of keys**, insert the key into the sorted list preserving the sorted order.
 - ex> Insert 3 into <1, 2, 4, 5, 6>



Description

- Insertion sort uses insertion incrementally.
 - Let $A[1..n]$ denote the array storing keys.
 - Insert $A[2]$ into $A[1]$.
 - Insert $A[3]$ into $A[1..2]$.
 - Insert $A[4]$ into $A[1..3]$.
 -
 -
 -
 - Insert $A[n]$ into $A[1..n-1]$.

Description: example

5 2 4 6 1 3

2 4 5 6 1 3

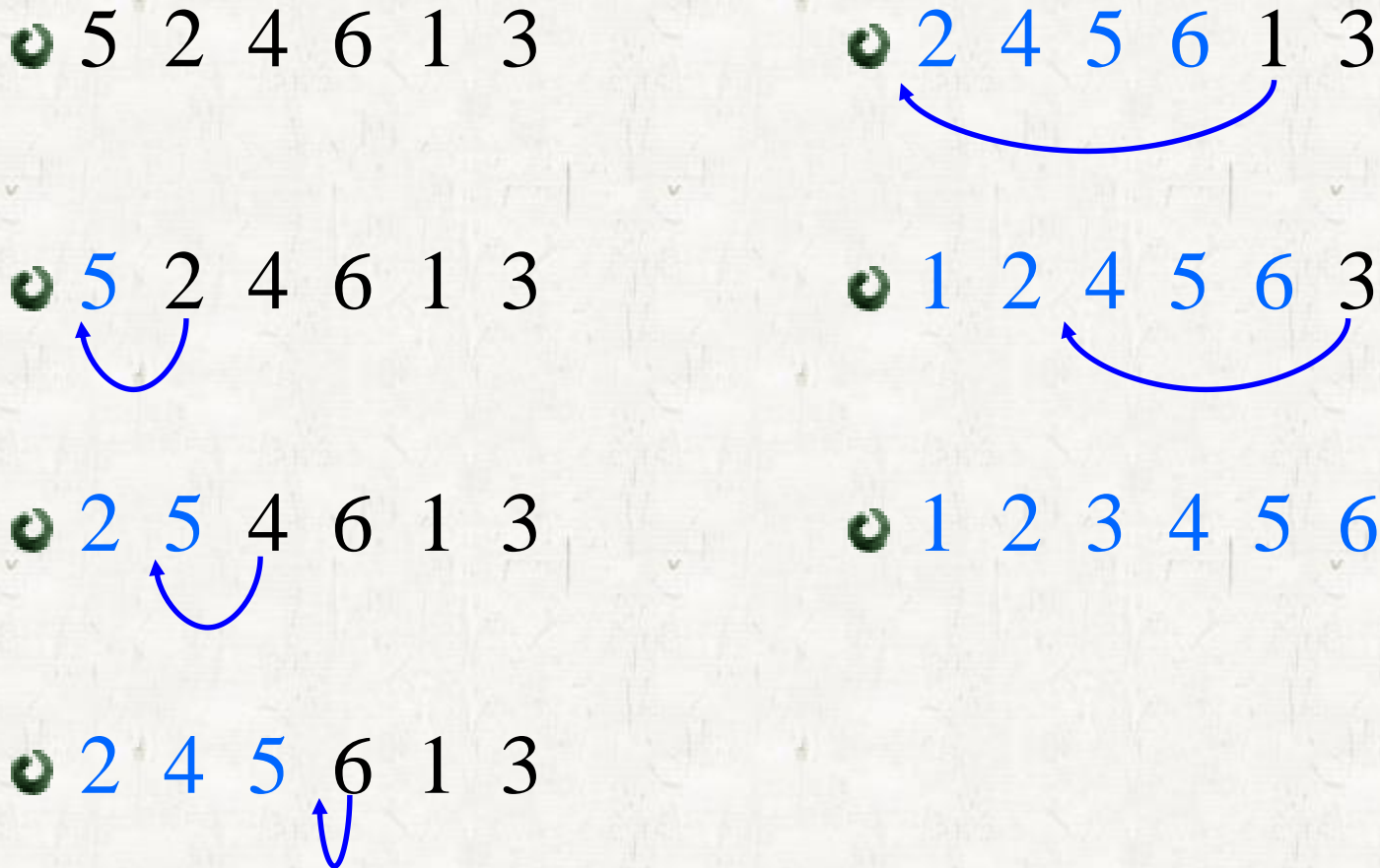
5 2 4 6 1 3

1 2 4 5 6 3

2 5 4 6 1 3

1 2 3 4 5 6

2 4 5 6 1 3



Description: pseudo code

INSERTION-SORT(A)

Pseudocode conventions are given in
p. 19 - 20 of the textbook.

```
1  for  $j = 2$  to  $A.length$ 
2       $key = A[j]$ 
3      // Insert  $A[j]$  into the sorted
        sequence  $A[1..j - 1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```

$n-1$ iterations of insertion.

Insert $A[j]$ into $A[1..j - 1]$.

Find a place to put $A[j]$.

Put $A[j]$.

Insertion sort

- Insertion sort
 - Description
 - Correctness
 - Performance
 - Running time
 - Space consumption

Running time

- How to analyze the running time of an algorithm?
 - Consider running the algorithm on a specific machine and measure the running time.
 - We cannot compare the running time of an algorithm on a machine with the running time of another algorithm on another machine.
 - So, we have to measure the running time of every algorithm on a specific machine, which is impossible.
 - Hence, we count the number of instructions used by the algorithm.

Instructions

• Arithmetic

- Add, Subtract, Multiply, Divide, remainder, floor, ceiling

• Data movement

- Load, store, copy

• Control

- Conditional branch
- Unconditional branch
- Subroutine call and return

Running time

- The running time of an algorithm grows with the **input size**, which is the number of items in the input.
- For example, sorting 10 keys is faster than sorting 100 keys.
- So the running time of an algorithm is described as **a function of input size n** , for example, $T(n)$.

Running time of insertion sort

| INSERTION-SORT(A) | | <i>cost</i> | <i>times</i> |
|--------------------------|--|-------------|--------------------------|
| 1 | for $j = 2$ to $A.length$ | c_1 | n |
| 2 | $key = A[j]$ | c_2 | $n - 1$ |
| 3 | // Insert $A[j]$ into the sorted sequence $A[1..j - 1]$. | 0 | $n - 1$ |
| 4 | $i = j - 1$ | c_4 | $n - 1$ |
| 5 | while $i > 0$ and $A[i] > key$ | c_5 | $\sum_{j=2}^n t_j$ |
| 6 | $A[i + 1] = A[i]$ | c_6 | $\sum_{j=2}^n (t_j - 1)$ |
| 7 | $i = i - 1$ | c_7 | $\sum_{j=2}^n (t_j - 1)$ |
| 8 | $A[i + 1] = key$ | c_8 | $n - 1$ |

- **$T(n)$:** The sum of product of *cost* and *times* of each line.

Running time of insertion sort

$$\begin{aligned}
 T(n) = & c_1 n + c_2 (n-1) + c_4 (n-1) \\
 & + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\
 & + c_7 \sum_{j=2}^n (t_j - 1) + c_8 (n-1)
 \end{aligned}$$

| <i>cost</i> | <i>times</i> |
|-------------|--------------------------|
| c_1 | n |
| c_2 | $n - 1$ |
| c_4 | $n - 1$ |
| c_5 | $\sum_{j=2}^n t_j$ |
| c_6 | $\sum_{j=2}^n (t_j - 1)$ |
| c_7 | $\sum_{j=2}^n (t_j - 1)$ |
| c_8 | $n - 1$ |

- $T(n)$: The sum of product of *cost* and *times* of each line.

Running time of insertion sort

- t_i : The number of times the **while** loop test is executed for j .
- Note that **for**, **while** loop test is executed one time more than the loop body.

Running time of insertion sort

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j-1) \\ + c_7 \sum_{j=2}^n (t_j-1) + c_8(n-1)$$

- Although the size of the input is the same, we have
 - best case
 - average case, and
 - worst case.

Running time of insertion sort

Best case

- If $A[1..n]$ is already sorted, $t_j = 1$ for $j = 2, 3, \dots, n$.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) \\ &\quad + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1) \\ &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

- This running time can be expressed as $an+b$ for constants a and b ; it is thus a **linear function** of n .

Running time of insertion sort

Worst case

- If $A[1..n]$ is sorted in reverse order, $t_j = j$ for $j = 2, 3, \dots, n$.

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1 \quad \text{and} \quad \sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

- This running time can be expressed as $an^2 + bn + c$ for constants a , b , and c ; it is thus a **quadratic function** of n .

Running time of insertion sort

- Only the degree of leading term is important.
 - Because we are only interested in the *rate of growth* or *order of growth*.
 - For example, a quadratic function grows faster than any linear function.
- The degree of leading term is expressed as Θ -notation.
 - The worst-case running time of insertion sort is $\Theta(n^2)$.

Space consumption of insertion sort

- $\Theta(n)$ space.
- Moreover, the input numbers are *sorted in place*.
 - $n + c$ space for some constant c .

Self-study on Insertion Sort

- **Exercise 2.1-1**

- **Exercise 2.1-2**

Content

- Sorting problem
- Sorting algorithms
 - Insertion sort - $\Theta(n^2)$.
 - Merge sort - $\Theta(n \lg n)$.

Merge

- What is merge sort?
 - A sorting algorithm using merge.
- What is merge?
 - Given two sorted lists of keys, generate a sorted list of the keys in the given sorted lists.
 - $\langle 1, 5, 6, 8 \rangle \langle 2, 4, 7, 9 \rangle \rightarrow \langle 1, 2, 4, 5, 6, 7, 8, 9 \rangle$

Merge

Merging example

- $\langle 1, 5, 6, 8 \rangle \langle 2, 4, 7, 9 \rangle \rightarrow \langle 1 \rangle$
- $\langle 5, 6, 8 \rangle \langle 2, 4, 7, 9 \rangle \rightarrow \langle 1, 2 \rangle$
- $\langle 5, 6, 8 \rangle \langle 4, 7, 9 \rangle \rightarrow \langle 1, 2, 4 \rangle$
- $\langle 5, 6, 8 \rangle \langle 7, 9 \rangle \rightarrow \langle 1, 2, 4, 5 \rangle$
- $\langle 6, 8 \rangle \langle 7, 9 \rangle \rightarrow \langle 1, 2, 4, 5, 6 \rangle$
- $\langle 8 \rangle \langle 7, 9 \rangle \rightarrow \langle 1, 2, 4, 5, 6, 7 \rangle$
- $\langle 8 \rangle \langle 9 \rangle \rightarrow \langle 1, 2, 4, 5, 6, 7, 8 \rangle$
- $\langle \rangle \langle 9 \rangle \rightarrow \langle 1, 2, 4, 5, 6, 7, 8, 9 \rangle$

Merge

MERGE(A, p, q, r)

```
1       $n_1 = q - p + 1$ 
2       $n_2 = r - q$ 
3      let  $L[1 \dots n_1 + 1]$  and  $R[1 \dots n_2 + 1]$  be new arrays
4      for  $i = 1$  to  $n_1$ 
5           $L[i] = A[p + i - 1]$ 
6      for  $j = 1$  to  $n_2$ 
7           $R[j] = A[q + j]$ 
8       $L[n_1 + 1] = \infty$ 
9       $R[n_2 + 1] = \infty$ 
10      $i = 1$ 
11      $j = 1$ 
12     for  $k = p$  to  $r$ 
13         if  $L[i] \leq R[j]$ 
14              $A[k] = L[i]$ 
15              $i = i + 1$ 
16         else  $A[k] = R[j]$ 
17              $j = j + 1$ 
```

Merge

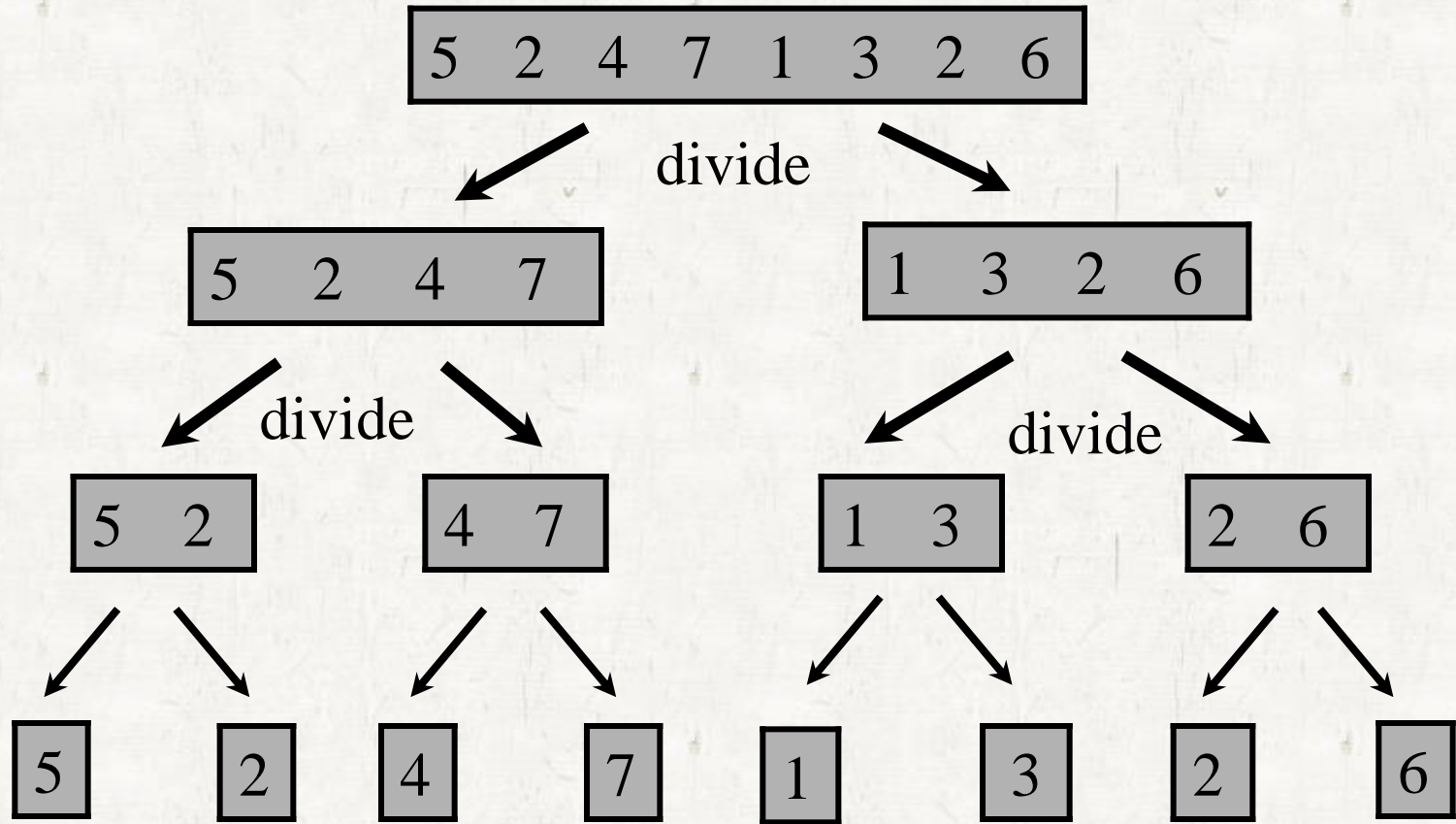
Running time of merge

- Let n_1 and n_2 denote the lengths of two sorted lists.
- $\Theta(n_1 + n_2)$ time.
 - Main operations: **compare** and **move**
 - $\# \text{comparison} \leq \# \text{movement}$
 - Obviously, $\# \text{movement} = n_1 + n_2$
 - So, $\# \text{comparison} \leq n_1 + n_2$
 - Hence, $\# \text{comparison} + \# \text{movement} \leq 2(n_1 + n_2)$
 - which means $\Theta(n_1 + n_2)$.

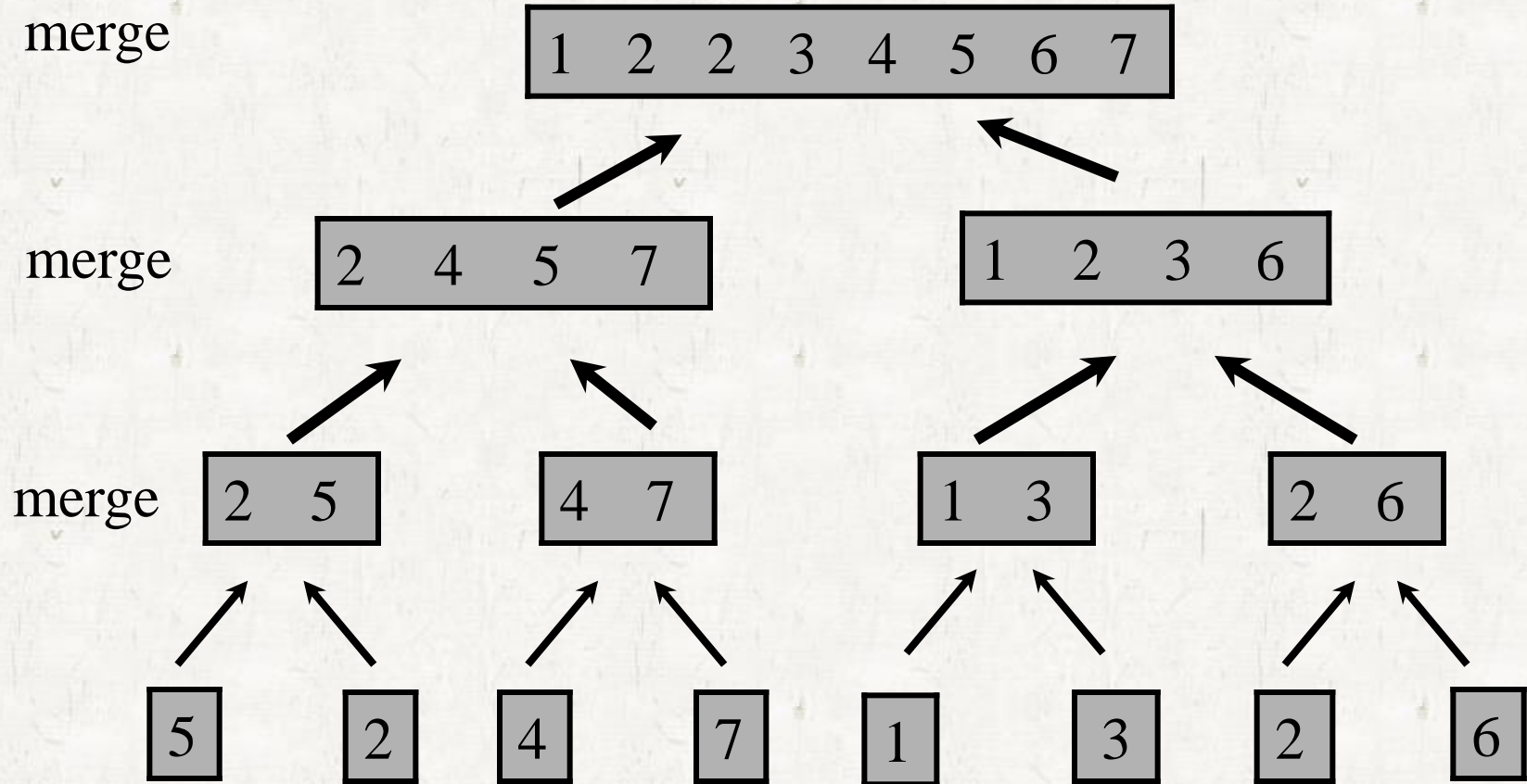
Merge sort

- A divide-and-conquer approach
 - **Divide:** Divide the n keys into two lists of $n/2$ keys.
 - **Conquer:** Sort the two lists recursively using merge sort.
 - **Combine:** Merge the two sorted lists.

Merge sort



Merge sort



Pseudo code

MERGE-SORT(A, p, r)

1 **if** $p < r$

2 $q = \lfloor (p + r)/2 \rfloor$

3 MERGE-SORT(A, p, q)

4 MERGE-SORT($A, q + 1, r$)

5 MERGE(A, p, q, r)

Running time

• **Divide:** $\Theta(1)$

- The divide step just computes the middle of the subarray, which takes constant time.

• **Conquer:** $2T(n/2)$

- We recursively solve two subproblems, each of size $n/2$.

• **Combine:** $\Theta(n)$

- We already showed that merging two sorted lists of size $n/2$ takes $\Theta(n)$ time.

Running time

- $T(n)$ can be represented as a recurrence.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Running time

- where the constant c represents the time required to solve problems of size 1 as well as the time per array element of the divide and combine steps.

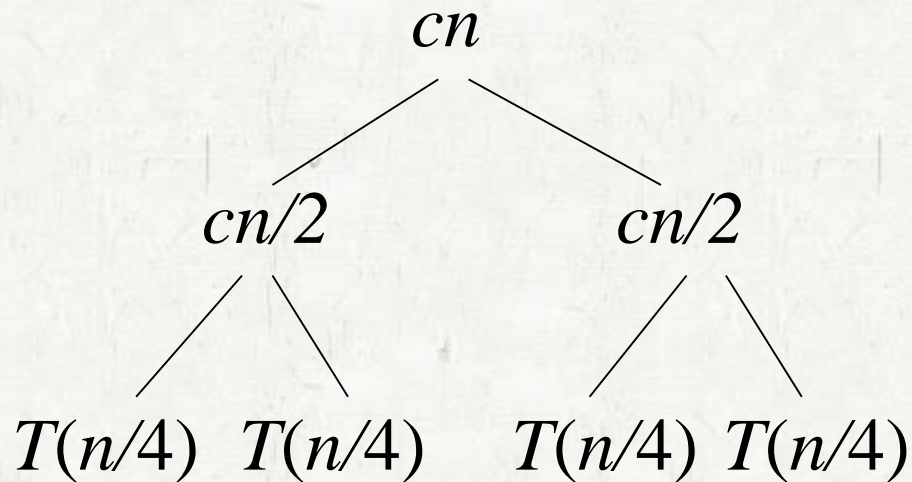
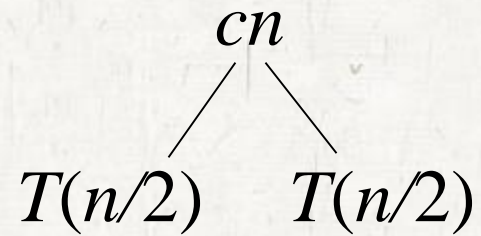
$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



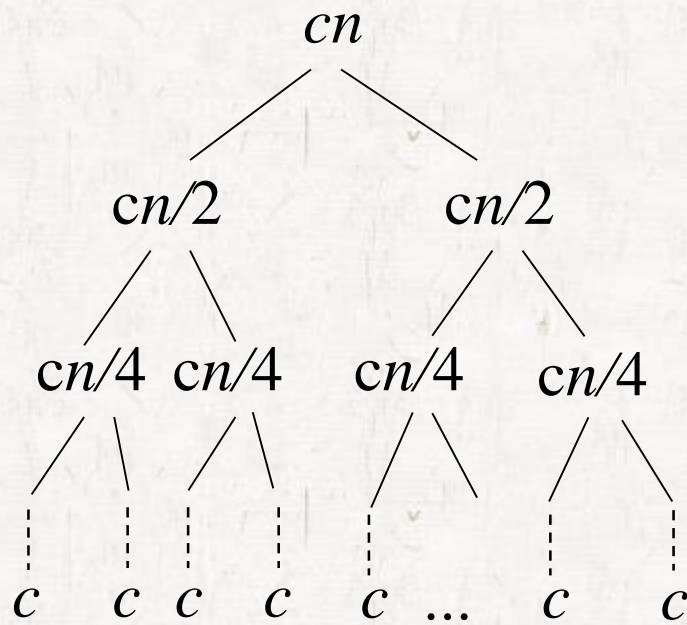
$$T(n) = \begin{cases} c & \text{if } n=1, \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

Recursion tree

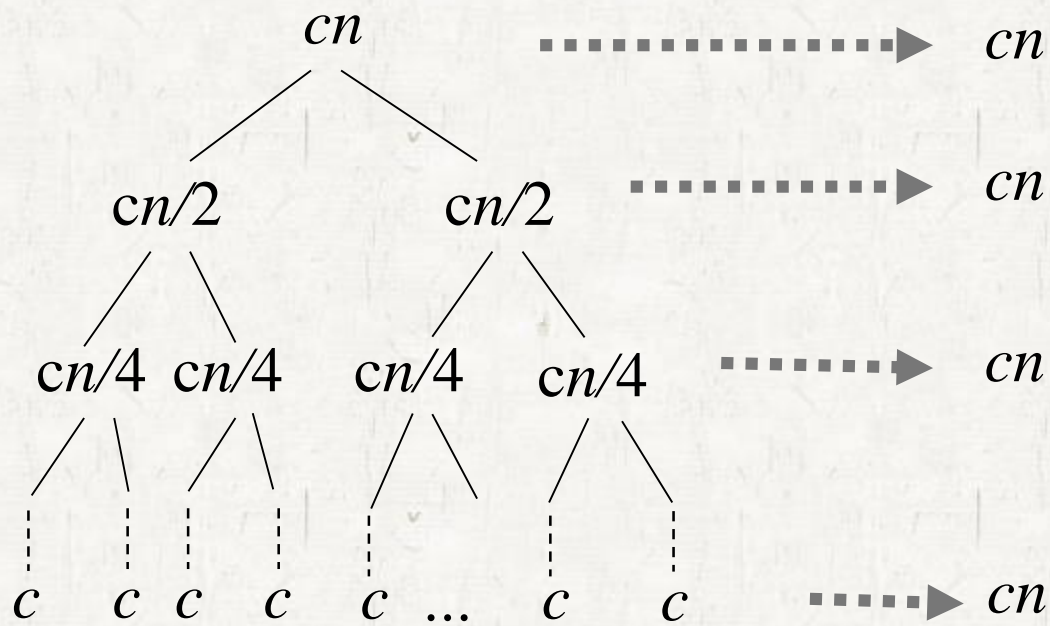
$T(n)$



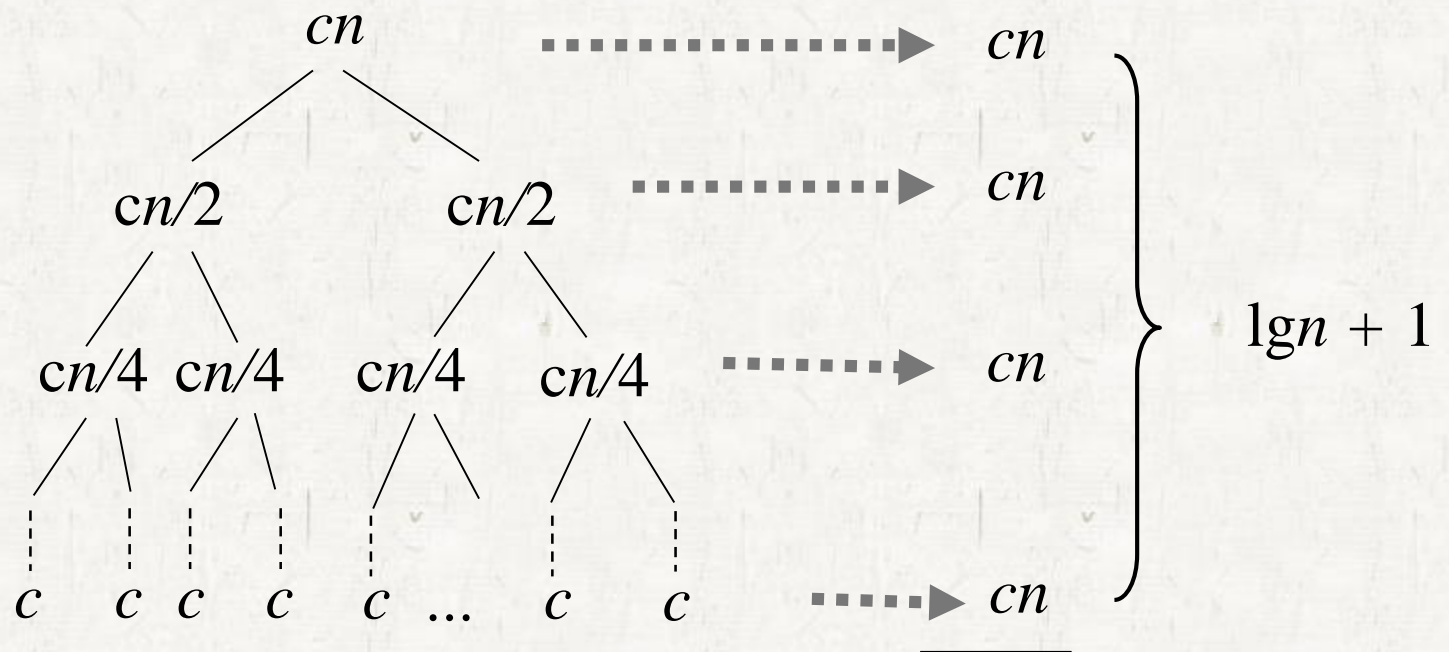
Recursion tree



Recursion tree



Recursion tree



Total : $cn \lg n + cn = \Theta(n \lg n)$

Self-study

● Merge sort

- Exercise 2.3-1
- Exercise 2.3-2

● Horner's rule

- Problem 2-3 (a) (b)

More (sorting) algorithms

- **Binary Search**

- Exercise 2.3-5

- **Selection sort**

- Exercise 2.2-2