Design and Analysis of Algorithms

Introduction

Prof. Chuhua Xian

Email: chhxian@scut.edu.cn

School of Computer Science and Engineering

South China University of Techonogy

First week

- Part I: About the course

- Part II: About algorithms
  - What are algorithms?
  - Why are they important to study?

Part I: About the course

Course Information

- Instructor: Prof. Chuhua Xian (冼楚华)
- Email: chhxian@scut.edu.cn
- Homepage： http://chuhuaxian.net

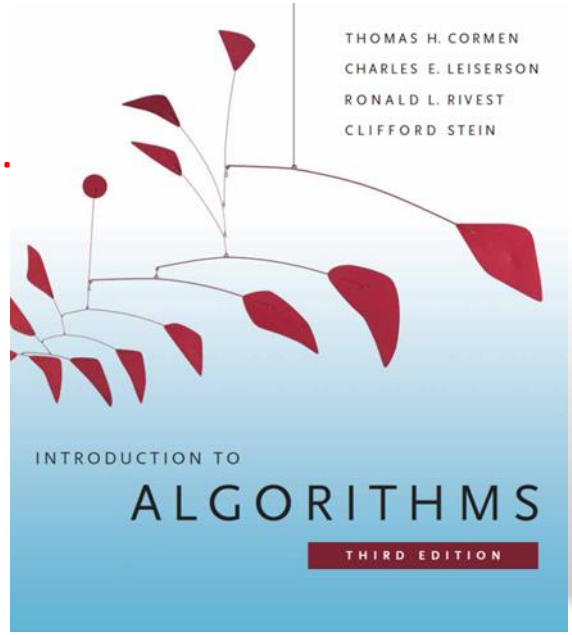
If you have any questions, please feel free to contact me by email. I will reply within two working days. Please list your name or student ID when you send me an email….

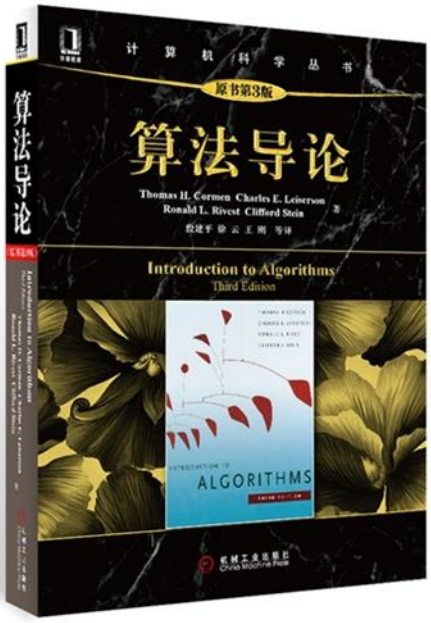
Course Information

- Teaching Assistant：
  - T.B.A

Course Information

- Website
  - http://www.chuhuaxian.net/ (后面会更新链接)
  - The slides, some answers of the homework and links of the resources will be put on this website.
- Reference
  - Introduction to Algorithms.
    Thomas H. Cormen,
    Charles E. Leiserson,
    Ronald L. Rivest,
    Clifford Stein.
    The MIT Press.





Course Information

- **Couse Time：** Week 1rd-9th and 12th-18th, 64 lessons (including 16 lessons for online experiments)
- **Experiments** Time and Room: to be announced
- **Final Grade：** Performance in class (10%) + homework and experiments (20%) + final examination (70%)

Course Information

- Online Judge：
  - http://scut.online South China University of Technology）
  - http://acm.zju.edu.cn/onlinejudge/ (Zhejiang University)
  - http://poj.org (Peking University)

About the flavor of the course

- It's more of a math flavor than a programming one.
- You will need to write pseudo-code, and implement it using C/C++, python…
- You will design and analyze, think and prove (rather than code)

Prerequisites

- Officially:
  - DISCRETE MATHEMATICS
  - PRGRAMING
  - DATA STRUCTURES
- Effectively: Basic mathematical maturity
  - functions, polynomial, exponential;
  - proof by induction;
  - basic data structure operations (stack, queue, …);
  - basic math manipulations…
- Note: As long as you'd like to learn it.

Homework and Experiment Policy

- Discussions and googling on web are allowed in general
- But you have to write down the solution yourself
- And you fully understand what you write.

Zero tolerance for cheating/plagiarism

- You may get 0 mark for this course
- Will check your codes by software; marks of both the codes provider and the copier will be 0 once the cheating/plagiarism behavior is confirmed

Suggestions/expectations/requirements

- In class:
  - Try to come on time.
  - Try your best to get more involved in the class.
  - Please don't chitchat.
    - It affects you, me, and other students.
- After class: treat homework seriously
  - The exams will be related to homework.

Awards for interactions

- Interactions in class are highly welcome.
  - Don't be shy or modest
  - Don't be afraid of asking "stupid" questions
    - There're no stupid questions; there're only stupid answers.
  - Don't be afraid of making mistakes
    - If you have to make some mistake, the earlier the better.
- I'll give marks for answering questions in class.
  - Marks directly go to your final grade.

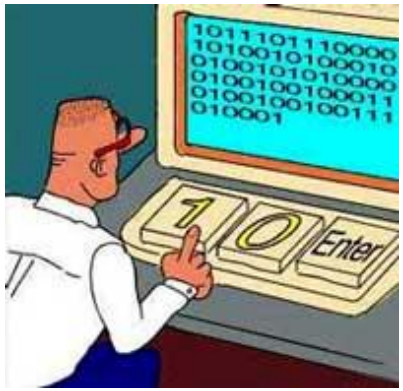
About comments

- Comments are welcome.
  - Please send to my Email
  - But please be responsible and considerate.
  - Pace: Please understand that I cannot proceed with the majority still confused.
- Any questions about the course?
- My questions:
  - What are your goals?
  - What do you like to learn from this course?
  - What excite you the most in general?

Part II: About algorithms

Factors of Programming

- Programming Languages？



BASIC

python

程序设计语言

C++

JAVA

C#

- **IDE?**

Microsoft Visual Basic 6.0

Microsoft Visual C++ 6.0

PyCharm
The Python IDE
for Professional Developers

Visual Studio

Borland C++Builder 6

eclipse JUNO

Visual C# Schritt für Schritt

Algorithms

Algorithm ?= Program

- **Algorithm. (webster.com)**

--A well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.

--Broadly: a step-step procedure for solving a problem or accomplishing some end especially by a computer.

--issues: correctness, efficiency (amount of work done and space used), storage (simplicity, clarity), optimality .etc.

Why study algorithms?

A concrete problem

- Problem: you are given n integers(there may be negative ones but not all) $a_1,a_2,\ldots,a_n$, determine i and j which maximize the sum from a_i to a_j.
- Case: 6 integers: (-2， 11， -4， 13， -5， -2)
- Ans: i=2,j=4, max sum is 20.

Importance of algorithms

- **Problem: sorting 10,000,000 integers**
  - Case 1: Computer A executes one billion instructions per second(1GHz), an algorithm taking time roughly equal to $2n^2$ to sort n integers.
  - Case 2: Computer B executes one hundred million instructions per second(100MHz), an algorithm taking time roughly equal to 50nlgn to sort n integers.
  - Case 1:

$$\frac{2\times(10^7)^2\,\text{instructions}}{10^9\,\text{instructions/second}} = 200000\text{seconds} \approx 55\text{hours}$$

  - Case 2:

$$\frac{50\times10^7\times\log 10^7\,\text{instructions}}{10^8\,\text{instructions/second}} = 105\text{seconds}$$

Importance of algorithms

Run time (nanoseconds)		$1.3\ N^3$	$10\ N^2$	$47\ N \log_2 N$	$48\ N$
Time to solve a problem of size	1000	1.3 seconds	10 msec	0.4 msec	0.048 msec
	10,000	22 minutes	1 second	6 msec	0.48 msec
	100,000	15 days	1.7 minutes	78 msec	4.8 msec
	million	41 years	2.8 hours	0.94 seconds	48 msec
	10 million	41 millennia	1.7 weeks	11 seconds	0.48 seconds
Max size problem solved in one	second	920	10,000	1 million	21 million
	minute	3,600	77,000	49 million	1.3 billion
	hour	14,000	600,000	2.4 billion	76 billion
	day	41,000	2.9 million	50 billion	1,800 billion
N multiplied by 10, time multiplied by		1,000	100	10+	10

Information Explosion

- Internet users **generate** about 2.5 quintillion bytes of **data** each day.
  In 2020, there are around 40 trillion gigabytes of data

- **Google gets over 3.5 billion searches daily.**

  - 850 million photos & 8 million videos every day (Facebook)
  - 50PB web pages, 500PB log (Baidu)

- Public Utilities
  - Health care (medical images - photos)
  - Public traffic (surveillance - videos)

- …

Research Frontier and Hot

- 《Science》: Special Online Collection: Dealing with Data
  - In this, *Science* joins with colleagues from *Science Signaling*, *Science Translational Medicine*, and *Science Careers* to provide a broad look at the issues surrounding the increasingly huge influx of research data. This collection of articles highlights both the challenges posed by the data deluge and the opportunities that can be realized if we can better organize and access the data.

Science
AAAS
11 FEBRUARY 2011 VOL 331
SCIENCE
www.sciencemag.org

- 《Nature》:

Big data, but are we ready?

Oswaldo Trelles, Pjotr Prins, Marc Snir and Ritsert C. Jansen

Data!! Data!! Data!!

What kind of problems

- Human Genome Project
  - 100,000 genes， sequences of the 3 billion chemical base pairs
- Internet
  - Finding good routes on which the data will travel
  - Search engine
- Electronic commerce
  - Public-key cryptography and digital signatures
- Manufacturing
  - Allocate scarce resources in the most beneficial way
- …

Analysis of algorithms

- **The theoretical study of computer-program performance and resource usage.**

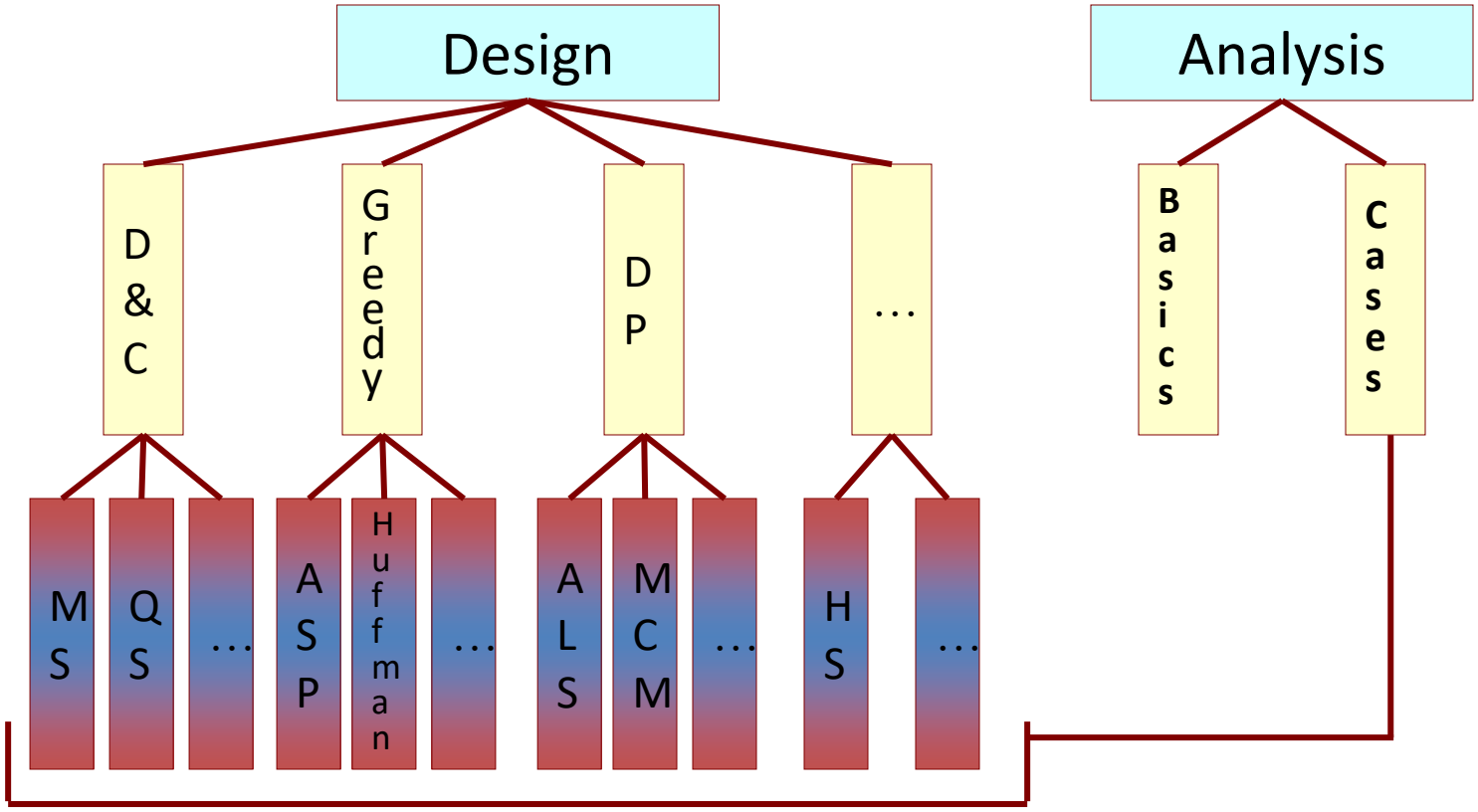
- **what's more important than performance?**

--modularity
--correctness
--maintainability
--functionality
--robustness
--user-friendliness
--programmer time
--simplicity
--extensibility
--reliability

- Objectives
  - Analyze the asymptotic performance of algorithms.
  - Demonstrate a familiarity with major algorithms and data structures.
  - Apply important algorithmic design paradigms and methods of analysis.
  - Synthesize efficient algorithms in common engineering design situations
  - Understand the difference between tractable and intractable problems, and be familiar with strategies to deal with intractability.

About the Course

- **Design and Analysis**
  - How can I propose an algorithm for a specific problem?
  - Is the algorithm good enough?



Why study algorithm and performance?

- Algorithms help us to understand ***scalability.***
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

- End of the slides here. Start the course using the black board.

The problem of sorting

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

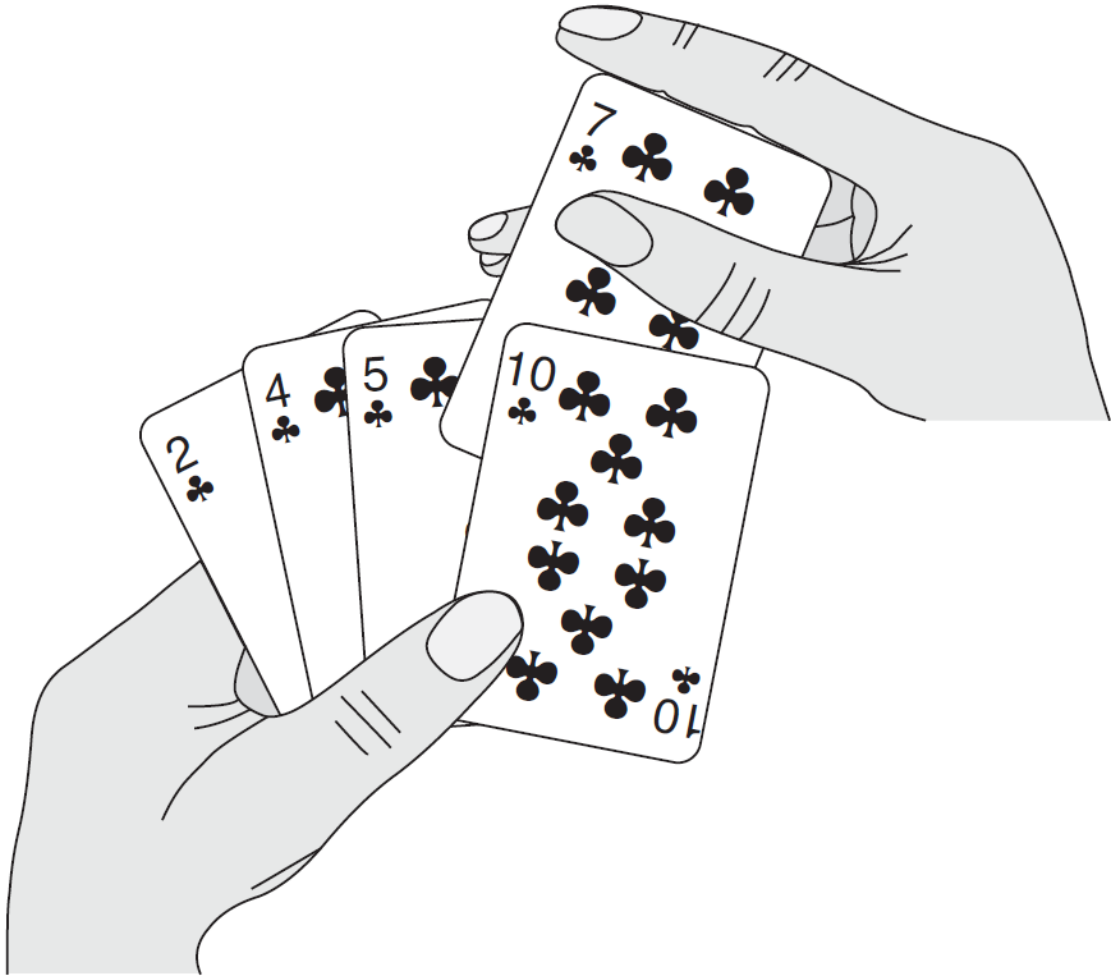
Output: 2 3 4 6 8 9

Overview

- ***Goals:***
  - Start using frameworks for describing and analyzing algorithms.
  - Examine two algorithms for sorting: insertion sort and merge sort.
  - See how to describe algorithms in pseudocode.
  - Begin using asymptotic notation to express running-time analysis.
  - Learn the technique of “divide and conquer” in the context of merge sort.

Insertion Sort

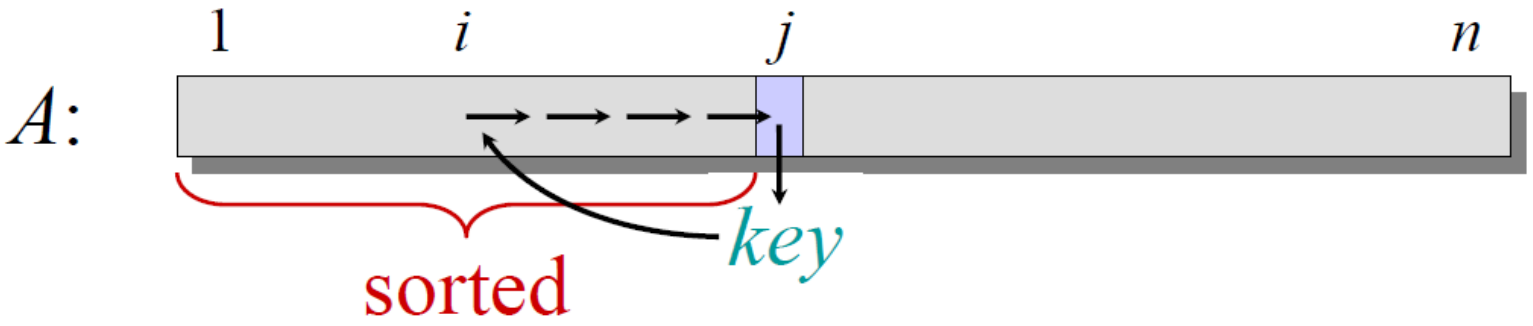
- Sorting a hand of cards using insertion sort.



Insertion sort

"pseudocode"

```
INSERTION-SORT (A, n)      ▷ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```



Example of insertion sort

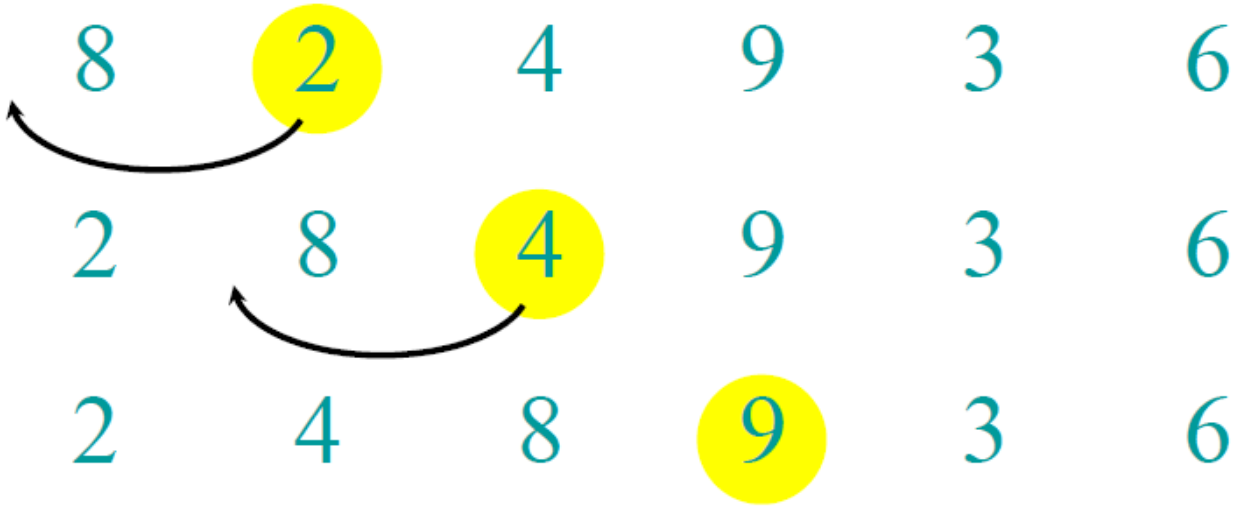
8 2 4 9 3 6

Example of insertion sort

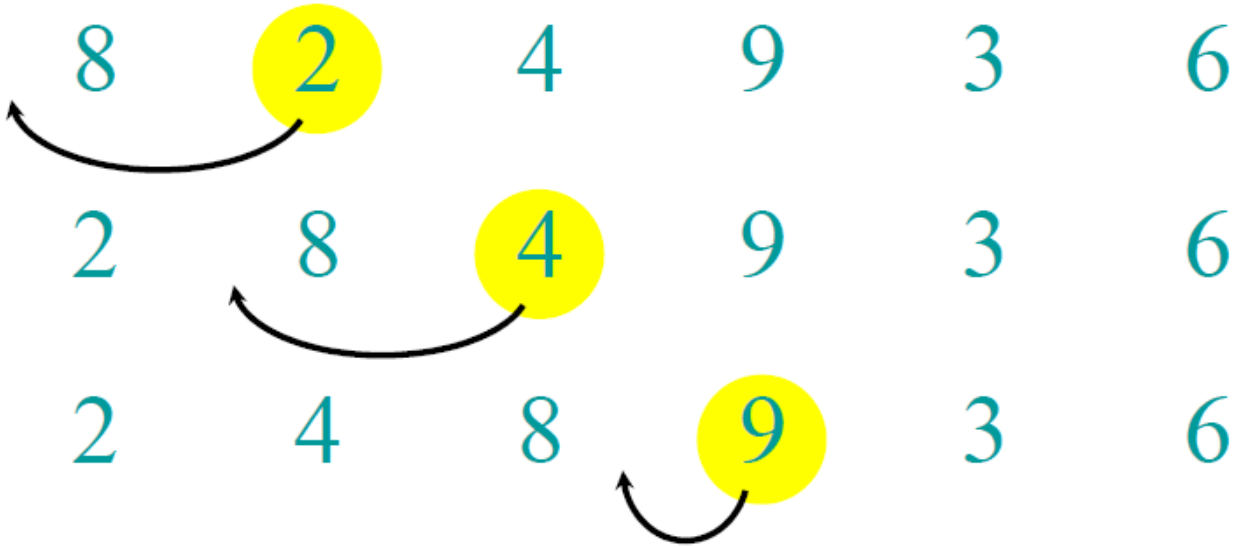
Example of insertion sort

Example of insertion sort

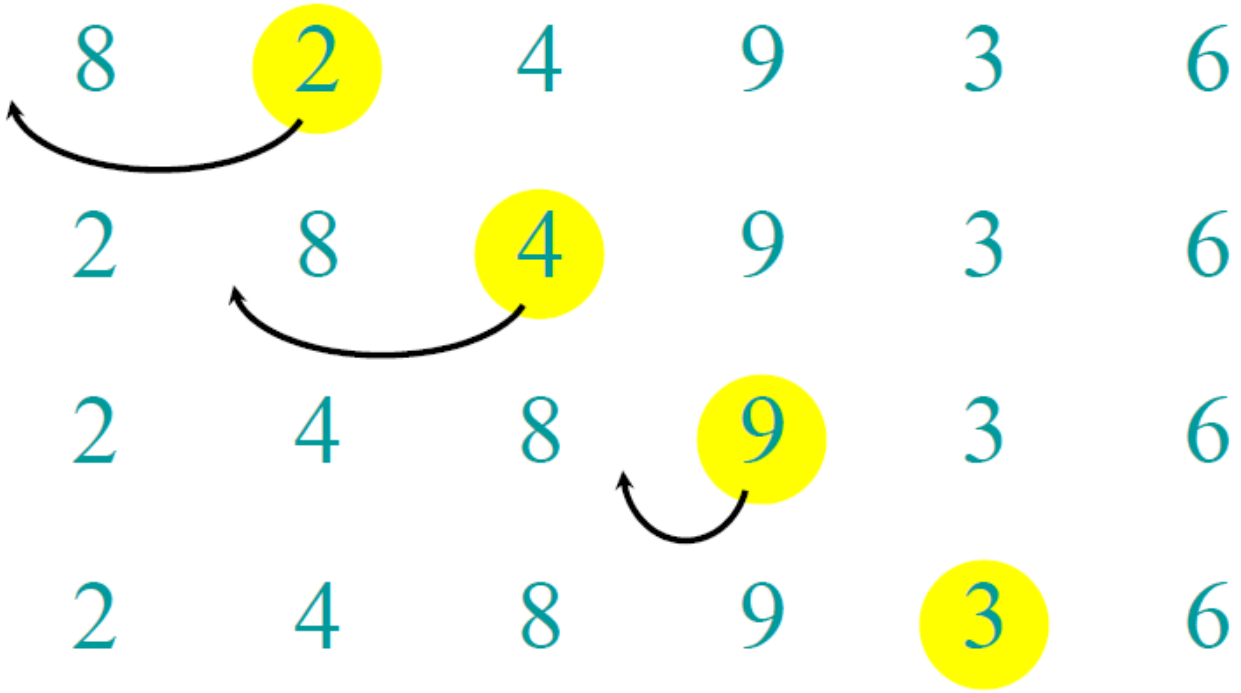
Example of insertion sort



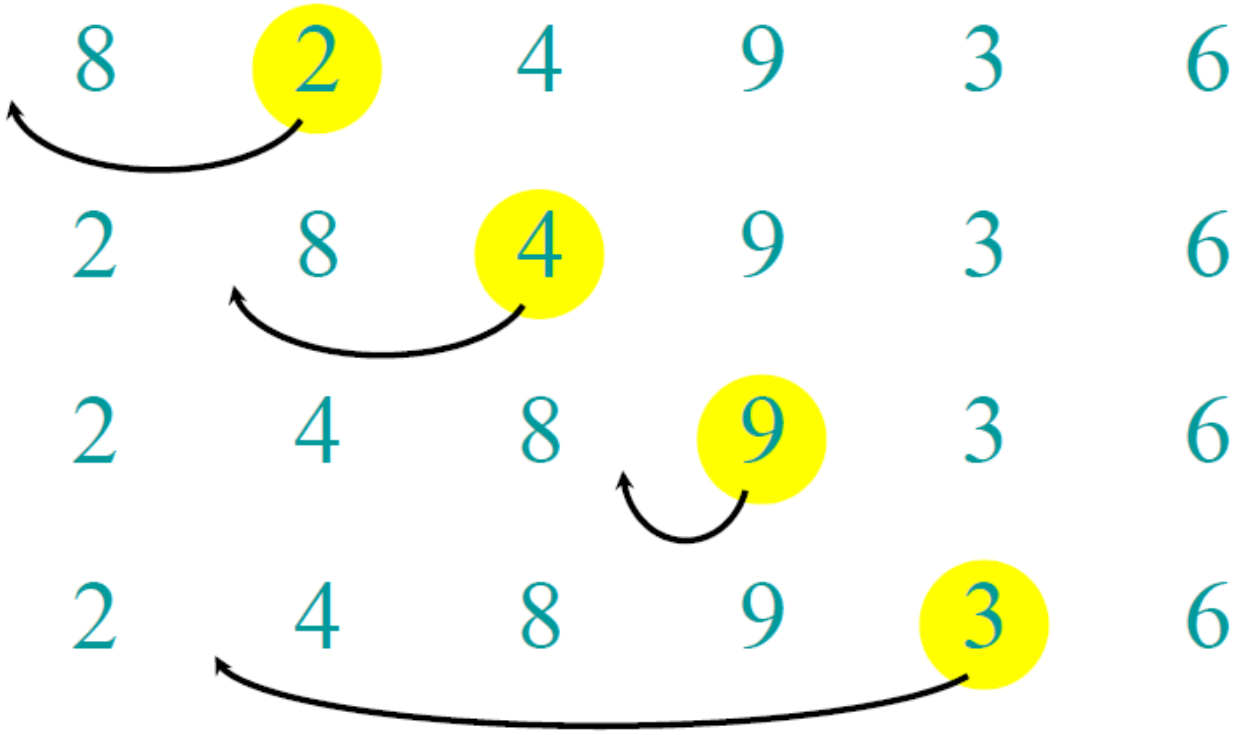
Example of insertion sort



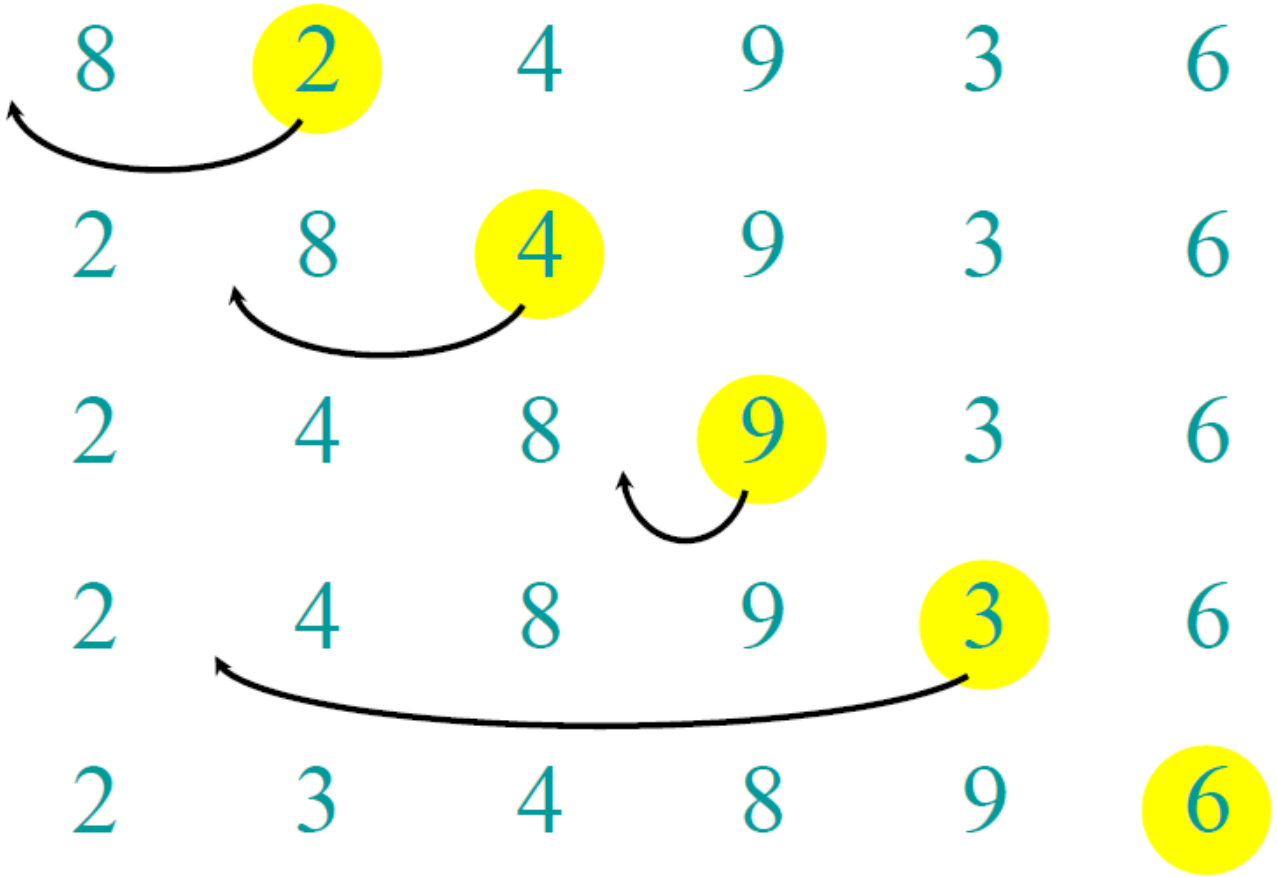
Example of insertion sort



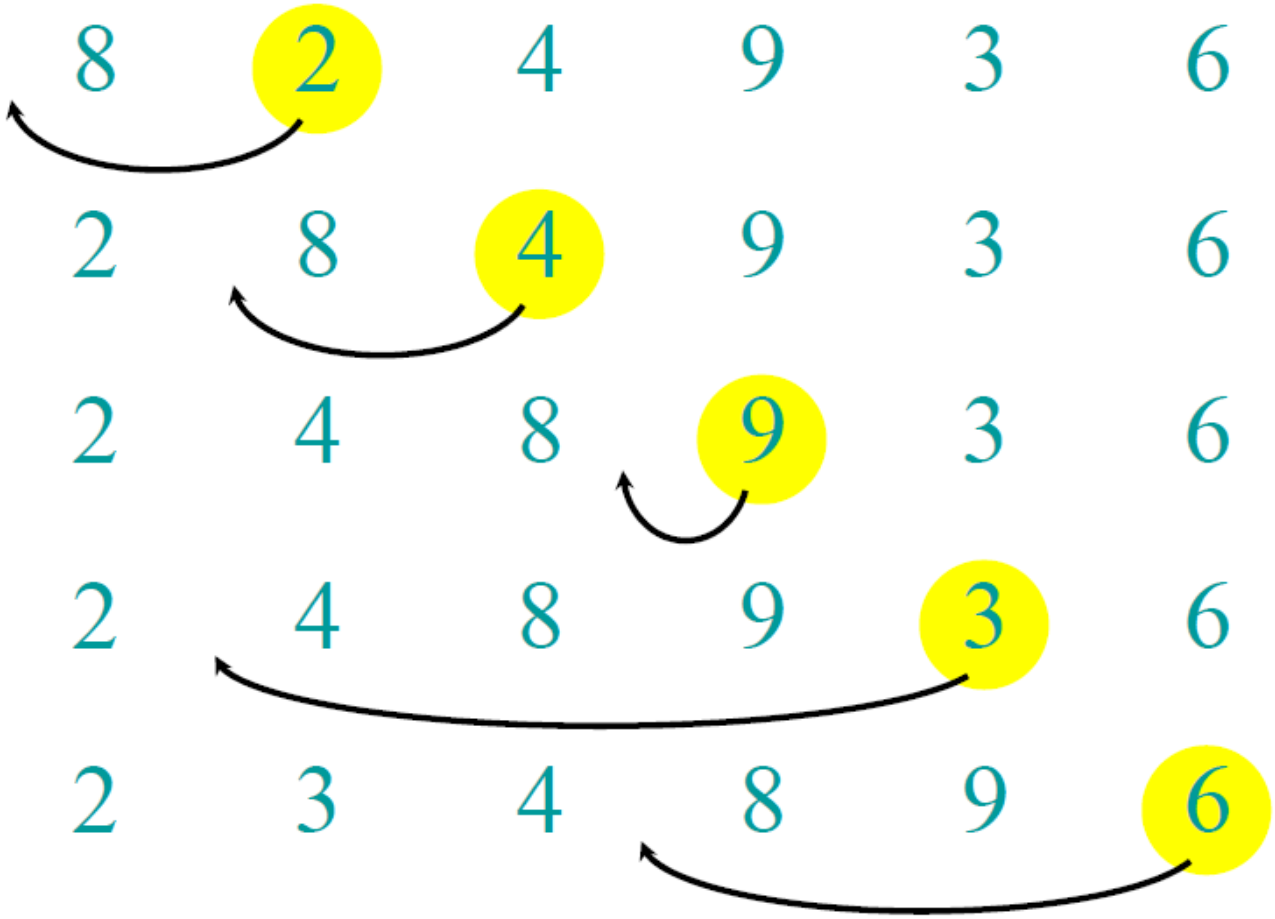
Example of insertion sort



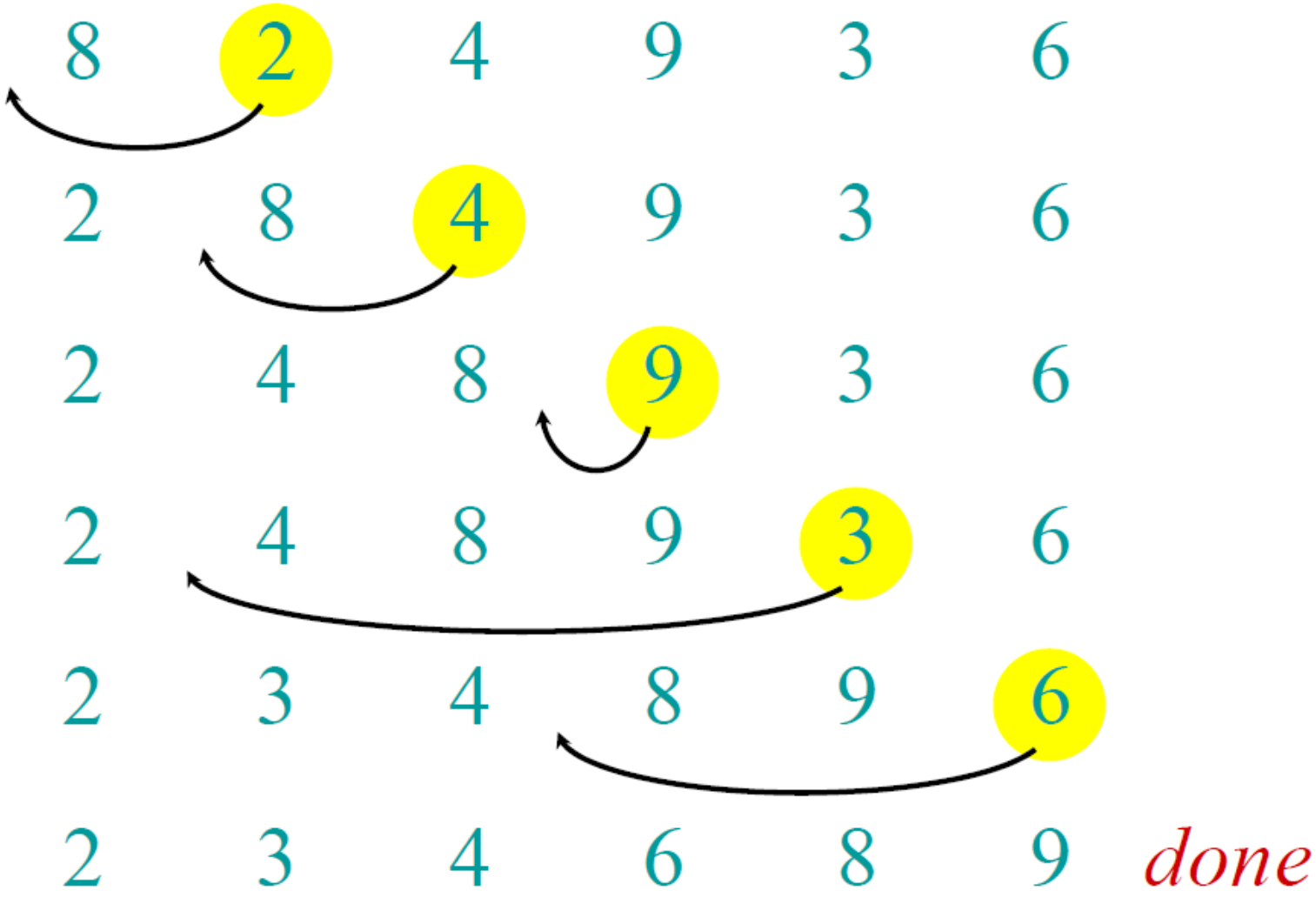
Example of insertion sort



Example of insertion sort

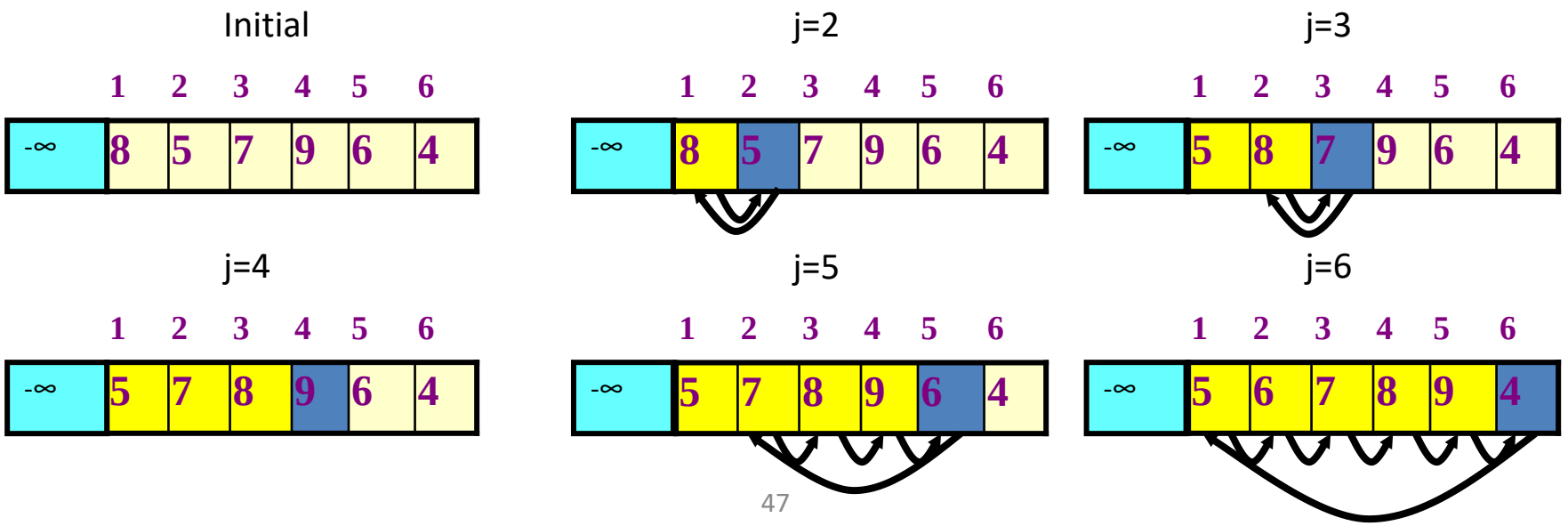


Example of insertion sort



Insertion Sort (another example)

```
INSERTION-SORT (A, n) ⊳ A[1 . . n]
1  for j ← 2 to n
2      do key ← A[j ]
3         i ← j – 1
4         while i > 0 and A[i] > key
5             do A[i + 1] ← A[i]
6                i ← i – 1
7         A[i + 1] = key
```



Running time

- **The running time depends on the input: an already sorted sequence is easier to sort.**

- **Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.**

- **Generally, we seek upper bounds on the running time, because everybody likes a guarantee.**

Kinds of analysis

- **Worst-case: (usually)**
  --T(n) = maximum time of algorithm on any input of size n.

- **Average-case: (sometimes)**
  --T(n) = expected time of algorithm over all inputs of size n.
  --Need assumption of statistics distribution of inputs.

- **Best-case: (bogus)**
  --Cheat with a slow algorithm that works fast on some input.

Machine-independent time

- **What is insertion sort's worst-case time?**
  --It depends on the speed of our computer:
  >>relative speed (on the same machine)
  >>absolute speed (on different machine)

- **BIG IDEA:**
  --Ignore machine-dependent constants.
  --Look at *growth* of $T(n)$ as $n \to \infty$

"Asymptotic Analysis"

Θ-notation

Math:

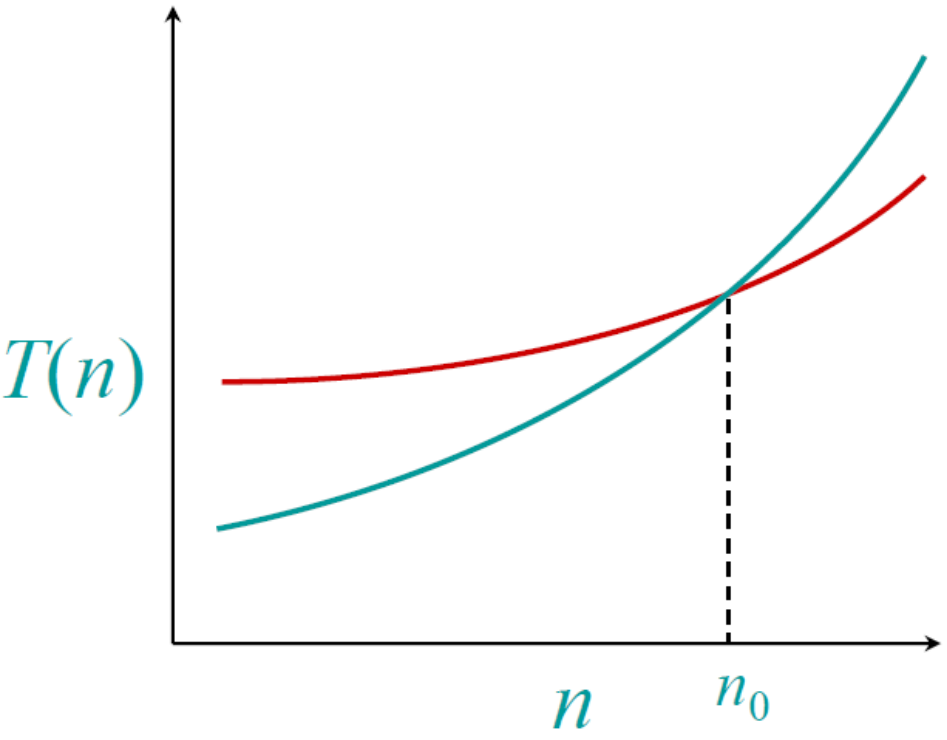
$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$

Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \,..\, n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j\,]$		
3	$i \leftarrow j - 1$		
4	**while** $i > 0$ and $A[i] > key$		
5	**do** $A[i + 1] \leftarrow A[i]$		
6	$i \leftarrow i - 1$		
7	$A[i + 1] = key$		

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	*cost*	*times*
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$		
4	**while** $i > 0$ and $A[i] > key$		
5	**do** $A[i+1] \leftarrow A[i]$		
6	$i \leftarrow i-1$		
7	$A[i+1] = key$		

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$		
5	**do** $A[i+1] \leftarrow A[i]$		
6	$i \leftarrow i-1$		
7	$A[i+1] = key$		

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5	**do** $A[i+1] \leftarrow A[i]$		
6	$i \leftarrow i-1$		
7	$A[i+1] = key$		

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n} (t_i - 1)$
6	$i \leftarrow i-1$		
7	$A[i+1] = key$		

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6	$i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7	$A[i+1] = key$		

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	*cost*	*times*
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_j$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_j - 1)$
6	$i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(tj - 1)$
7	$A[i+1] = key$	c_7	$n-1$

Analysis of INSERTION-SORT

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	*cost*	*times*
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6	$i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7	$A[i+1] = key$	c_7	$n-1$

Let $T(n)$ = running time of **INSERTION-SORT**.

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$$

	INSERTION-SORT $(A, n) \triangleright A[1..n]$	cost	times
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_j$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_j-1)$
6	$i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(tj-1)$
7	$A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$$

Best-case: The array is already sorted.

- Always find that **A[i] ≤ key** upon the first time the while loop test is run (when i = j – 1).
- All t_j are 1.
- Running time is

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

- Can express $T(n)$ as $an + b$ for constants a and b (that depend on the statement costs c_i) $\Rightarrow T(n)$ is a linear function of n. $\Rightarrow T(n) = \Theta(n)$

INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1 for $j \leftarrow 2$ **to** n	c_1	n
2 do $key \leftarrow A[j]$	c_2	$n-1$
3 $i \leftarrow j-1$	c_3	$n-1$
4 while $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5 do $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6 $i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7 $A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$$

Worst-case: The array is in reverse sorted order.

- Always find that ***A[i] > key*** in while loop test.

INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1 for $j \leftarrow 2$ **to** n	c_1	n
2 do $key \leftarrow A[j]$	c_2	$n-1$
3 $i \leftarrow j-1$	c_3	$n-1$
4 while $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5 do $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6 $i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7 $A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n}(t_j - 1) + c_6 \sum_{j=2}^{n}(t_j - 1) + c_7(n-1)$$

Worst-case: The array is in reverse sorted order.

- Have to compare ***key*** with all elements to the left of the *j* th position ⇒ compare with *j* − *1* elements.

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6	$i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7	$A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_i + c_5 \sum_{j=2}^{n}(t_i - 1) + c_6 \sum_{j=2}^{n}(t_i - 1) + c_7(n-1)$$

Worst-case: The array is in reverse sorted order.

- Since the while loop exits because i reaches 0, there's one additional test after the $j - 1$ tests $\Rightarrow t_j = j$.

INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	cost	times
1 for $j \leftarrow 2$ **to** n	c_1	n
2 do $key \leftarrow A[j]$	c_2	$n-1$
3 $i \leftarrow j-1$	c_3	$n-1$
4 while $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5 do $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6 $i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7 $A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_i + c_5 \sum_{j=2}^{n}(t_i - 1) + c_6 \sum_{j=2}^{n}(t_i - 1) + c_7(n-1)$$

Worst-case: The array is in reverse sorted order.

- $\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j \; and \; \sum_{j=2}^{n}(t_j - 1) = \sum_{j=2}^{n}(j-1)$

INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	*cost*	*times*
1 for $j \leftarrow 2$ **to** n	c_1	n
2 do $key \leftarrow A[j]$	c_2	$n-1$
3 $i \leftarrow j-1$	c_3	$n-1$
4 while $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5 do $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6 $i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7 $A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_i + c_5 \sum_{j=2}^{n}(t_i - 1) + c_6 \sum_{j=2}^{n}(t_i - 1) + c_7(n-1)$$

Worst-case: The array is in reverse sorted order. Running time:

- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2} - 1\right) + c_7(n-1)$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 - \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) + c_7\right) n - (c_1 + c_3 + c_4 + c_7)$$

a ***b*** ***c***

	INSERTION-SORT $(A, n) \triangleright A[1 \ldots n]$	***cost***	***times***
1	**for** $j \leftarrow 2$ **to** n	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$i \leftarrow j-1$	c_3	$n-1$
4	**while** $i > 0$ and $A[i] > key$	c_4	$\sum_{j=2}^{n} t_i$
5	**do** $A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n}(t_i - 1)$
6	$i \leftarrow i-1$	c_6	$\sum_{j=2}^{n}(ti - 1)$
7	$A[i+1] = key$	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_i + c_5 \sum_{j=2}^{n}(t_i - 1) + c_6 \sum_{j=2}^{n}(t_i - 1) + c_7(n-1)$$

Worst-case: The array is in reverse sorted order.

- $Can\ express\ T\ (n)\ as\ an^2 + bn + c\ for\ constants\ a, b, c\ (that\ again\ depend\ on\ statement\ costs) \Rightarrow$
  $T\ (n)\ is\ a\ quadratic\ function\ of\ n. \Rightarrow\ T\ (n) = \Theta(n^2)$

We usually concentrate on finding the **worst-case** running time: the longest running time for any input of size n.

Reasons:

- The worst-case running time gives a guaranteed upper bound on the running time for any input.
- For some algorithms, the worst case occurs often. For example, when searching, the worst case often occurs when the item being searched for is not present, and searches for absent items may be frequent.
- Why not analyze the average case? Because it's often about as bad as the worst case.

Order of Growth

- **We will only consider order of growth of running time:**

-- We can ignore the lower-order terms, since they are relatively insignificant for very large n.

-- We can also ignore leading term's constant coefficients, since they are not as important for the rate of growth in computational efficiency for very large n.

--We just said that best case was linear in n and worst/average case quadratic in n.

Designing Algorithms

- **We discussed insertion sort**

--We analyzed insertion sort

--We discussed how we are normally only interested in growth of running time:

>>Best-case linear in O(n), worst-case quadratic in $O(n^2)$

-- Can we design better than n^2 sorting algorithms?

-- We will do so using one of the most powerful algorithm design techniques.

Divide - and - Conquer

- **To solve P:**
  - **-- Divide P into smaller problems P_1, P_2, …, P_k.**
  - **-- Conquer by solving the (smaller) subproblems recursively.**
  - **-- Combine the solutions to P_1, P_2, …, P_k into the solution for P.**

Merge Sort Algorithm

- **Using divide-and-conquer, we can obtain a merge-sort algorithm**
  -- Divide: Divide the n elements into two subsequences of n/2 elements each.
  -- Conquer: Sort the two subsequences recursively.
  -- Combine: Merge the two sorted subsequences to produce the sorted answer.

- **Assume we have procedure MERGE (A, p,q,r) which merges sorted A[p…q] with sorted A[q+1…r] in (r - p) time.**

Merge-Sort (A, p, r)

- **INPUT: a sequence of n numbers stored in array A**

- **OUTPUT: an ordered sequence of n numbers**

```
MERGE-SORT (A, p, r)
1    if p < r
2        then q ← ⌊(p + r)/2⌋
3            MERGE-SORT(A, p, q)
4            MERGE-SORT(A, q + 1, r)
5            MERGE(A, p, q, r)
```

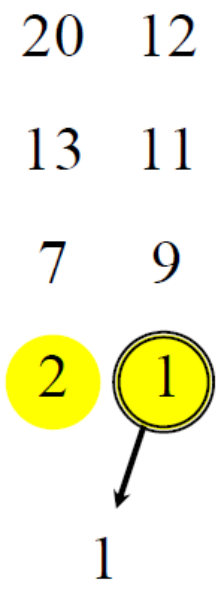
Merge (A, p, q, r)

```
MERGE (A, p, q, r)
1   n1 ← q − p + 1
2   n2 ← r − q
3   create arrays L[1..n1 + 1] and R[1..n2 + 1]
4   for i ← 1 to n1
5         do L[i] ← A[p + i - 1]
6   for j ← 1 to n2
7         do R[j] ← A[q + j]
8   L[n1 + 1] ← ∞
9   R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13        do if L[i] ≤ R[j]
14              then A[k] ← L[i]
15                   i ← i + 1
16              else A[k] ← R[j]
17                   j ← j + 1
```

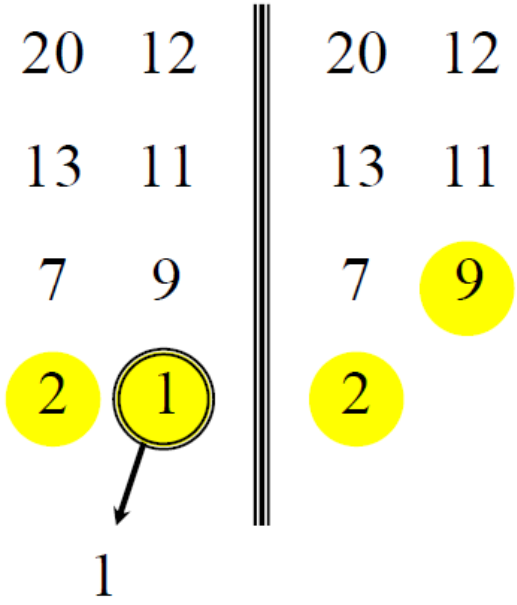
Merging two sorted arrays

20	12
13	11
7	9
2	1

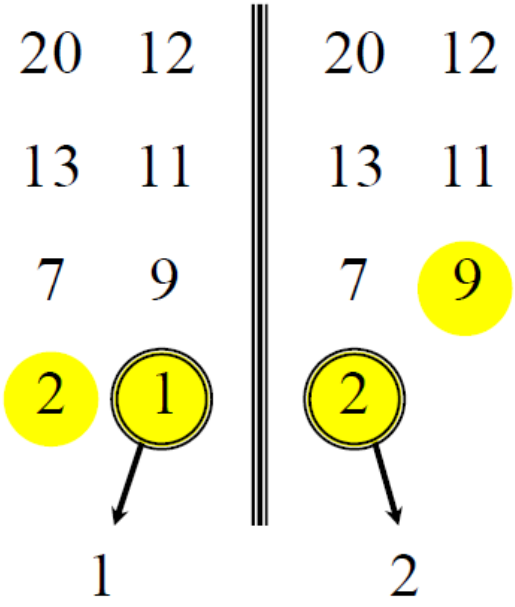
Merging two sorted arrays



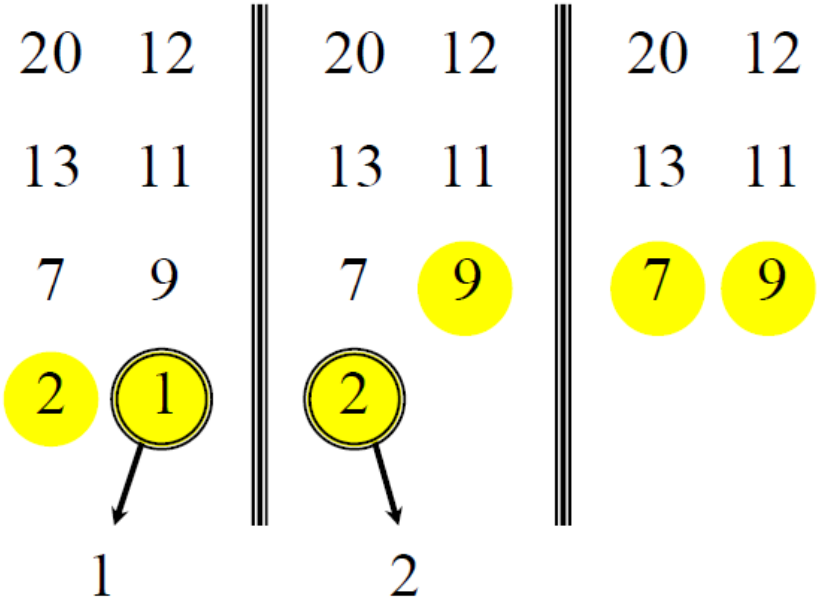
Merging two sorted arrays



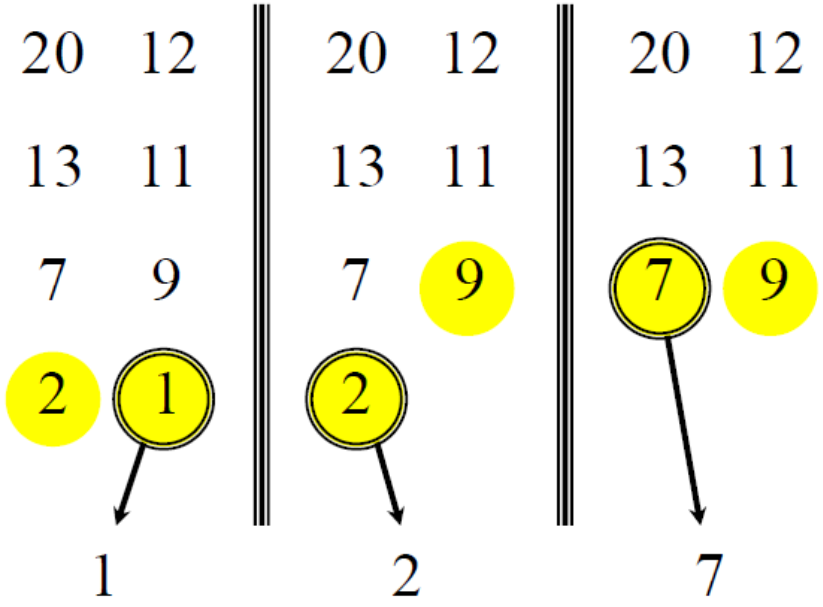
Merging two sorted arrays



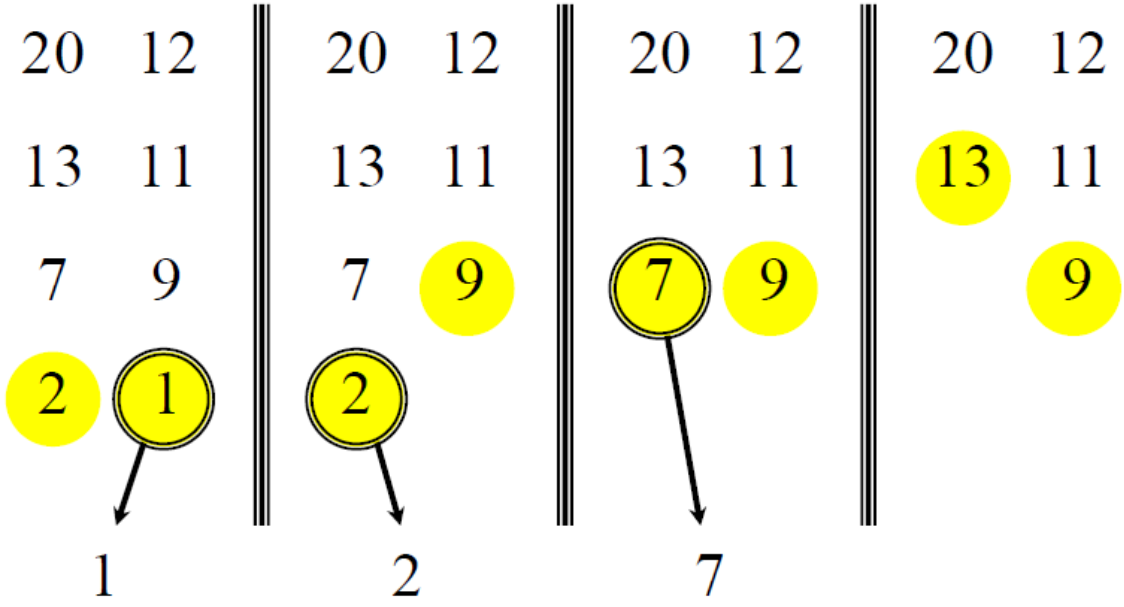
Merging two sorted arrays



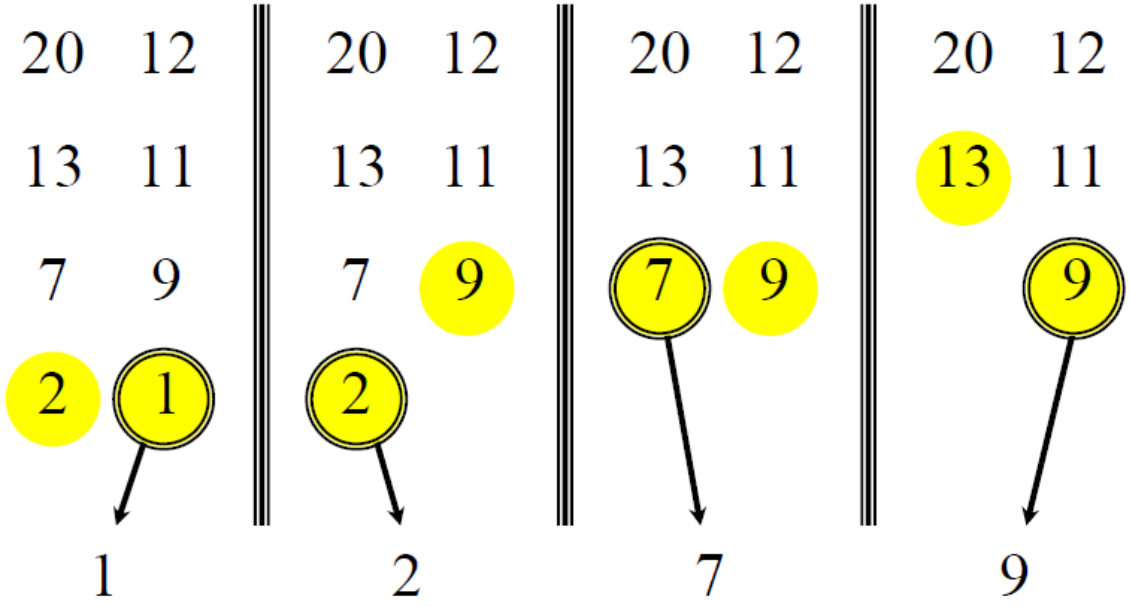
Merging two sorted arrays



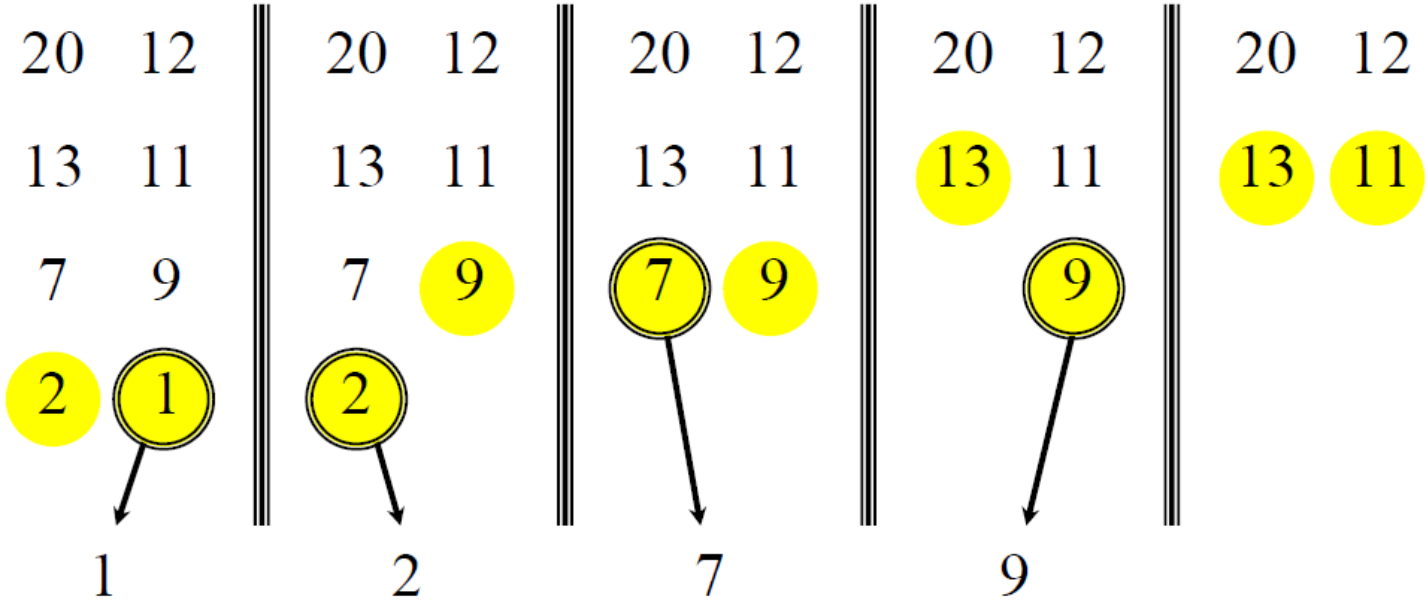
Merging two sorted arrays



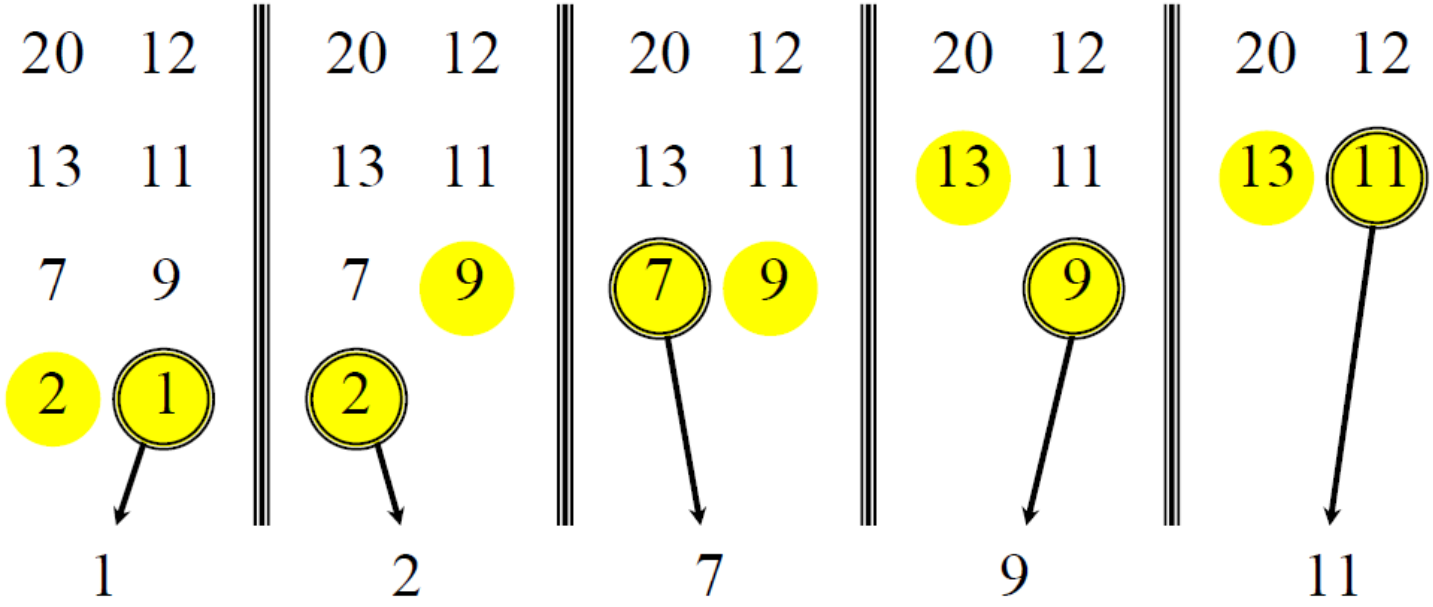
Merging two sorted arrays



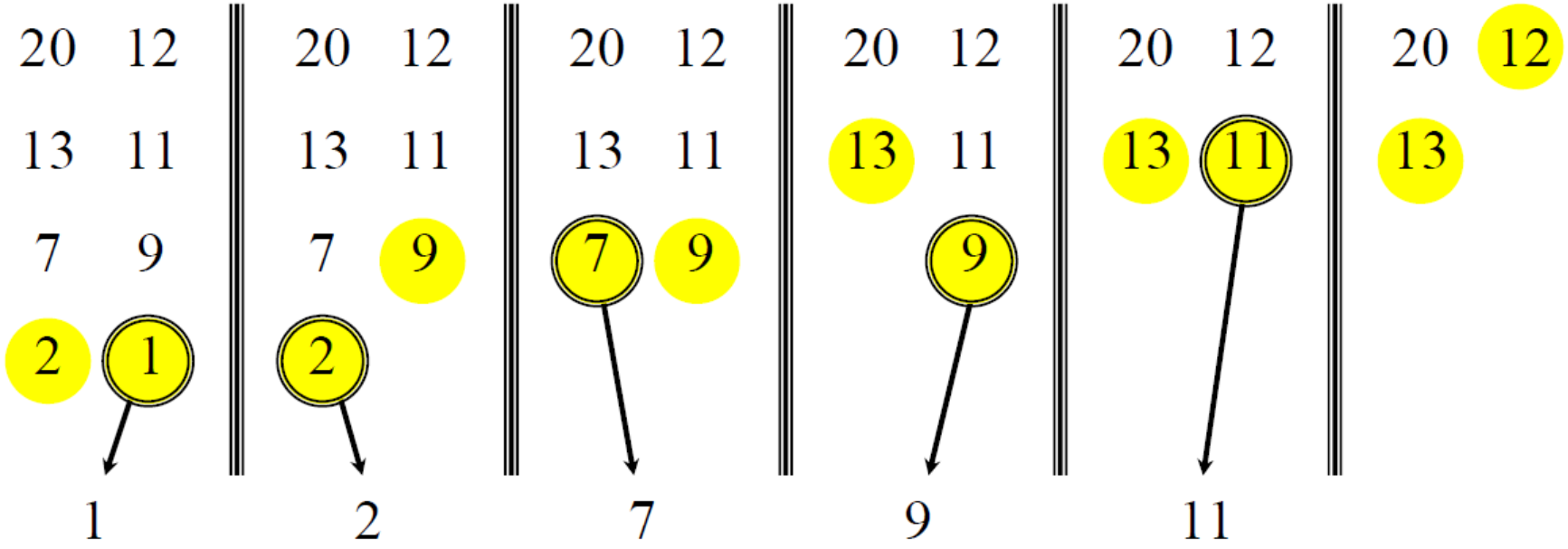
Merging two sorted arrays



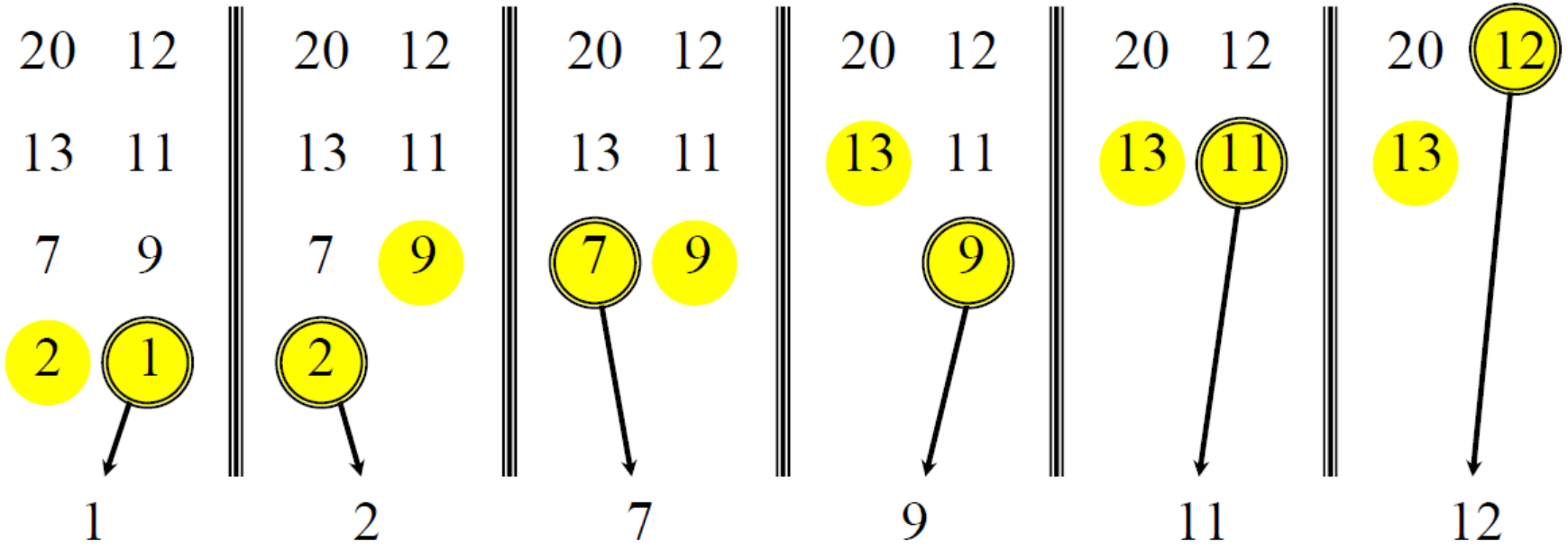
Merging two sorted arrays

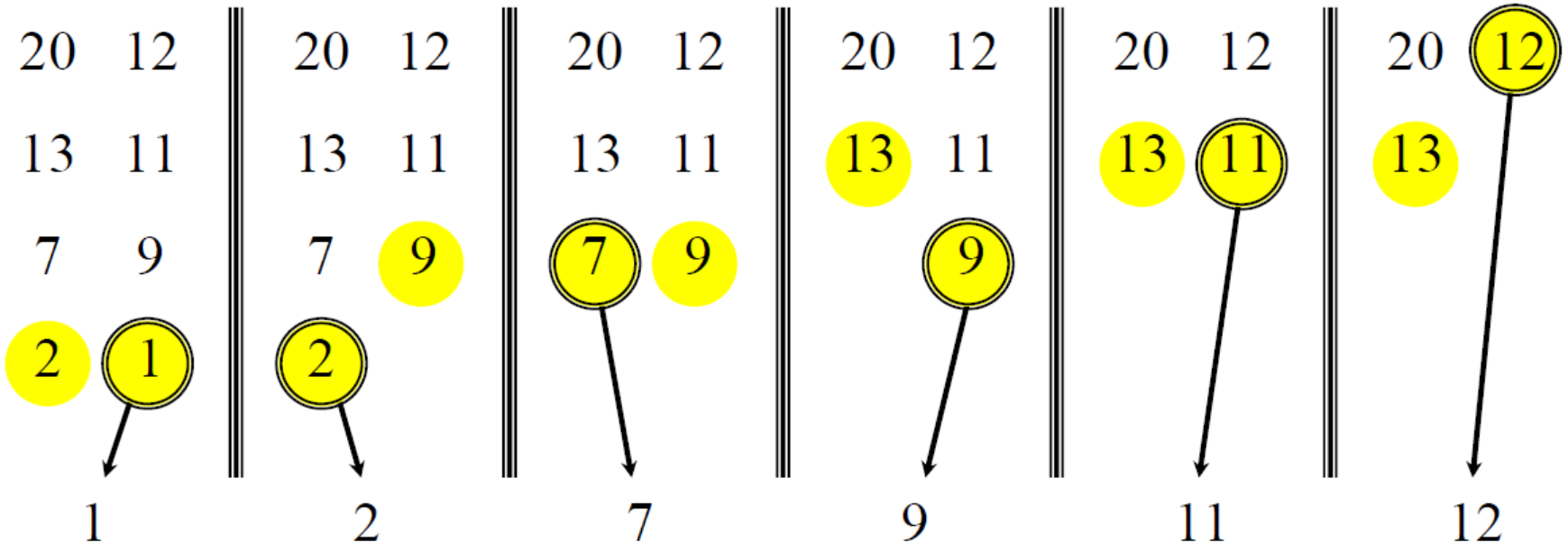


Merging two sorted arrays



Merging two sorted arrays





Time = $\Theta(n)$ to merge a total of n elements (linear time).

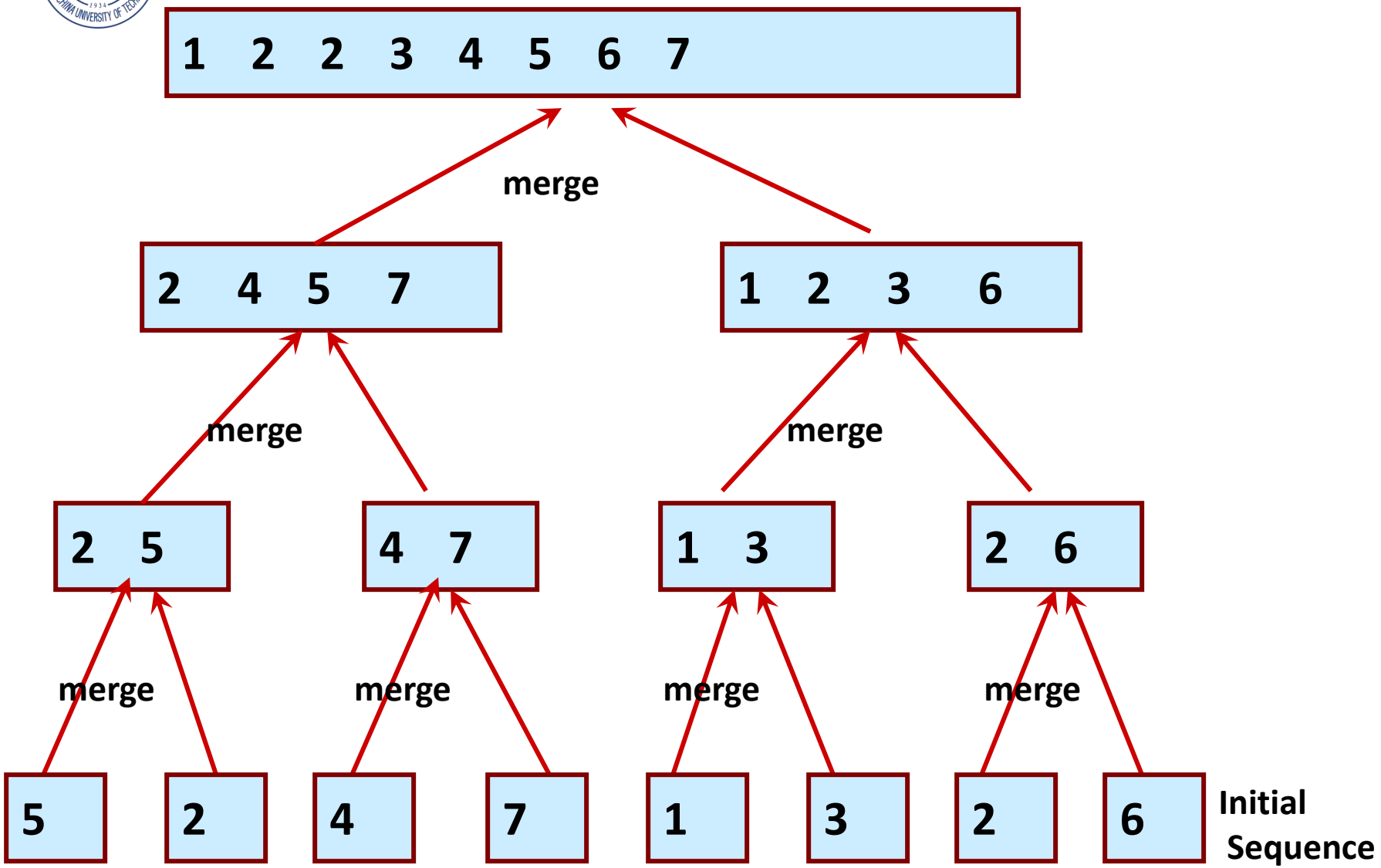
Analyzing merge sort

	MERGE-SORT $A[1 \ldots n]$
$T(n)$	
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[1 \ldots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \ldots n]$.
$\Theta(n)$	***3. "Merge"*** the 2 sorted lists

Abuse

Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.

Action of Merge Sort



Analyzing Merge Sort

- **How long does merge sort take?**

  -- Bottleneck = merging (and copying).

  >> merging two files of size *n/2* requires *n* comparisons

  -- *T(n)* = comparisons to merge sort *n* elements.

  >>to make analysis cleaner, assume n is a power of 2

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{Sorting both halves}} + \underbrace{\Theta(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

- **Claim. $T(n) = n \lg_2 n$**

  -- Note: same number of comparisons for ANY file.

  >> even already sorted

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

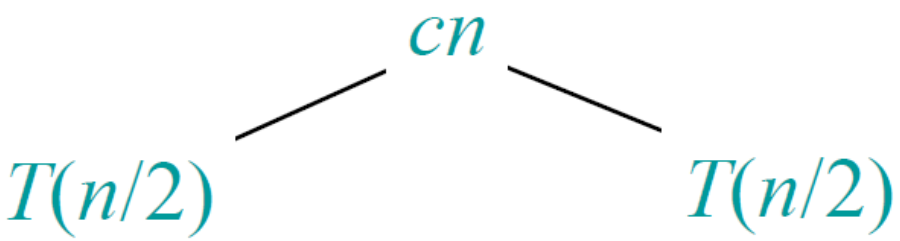
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$T(n)$

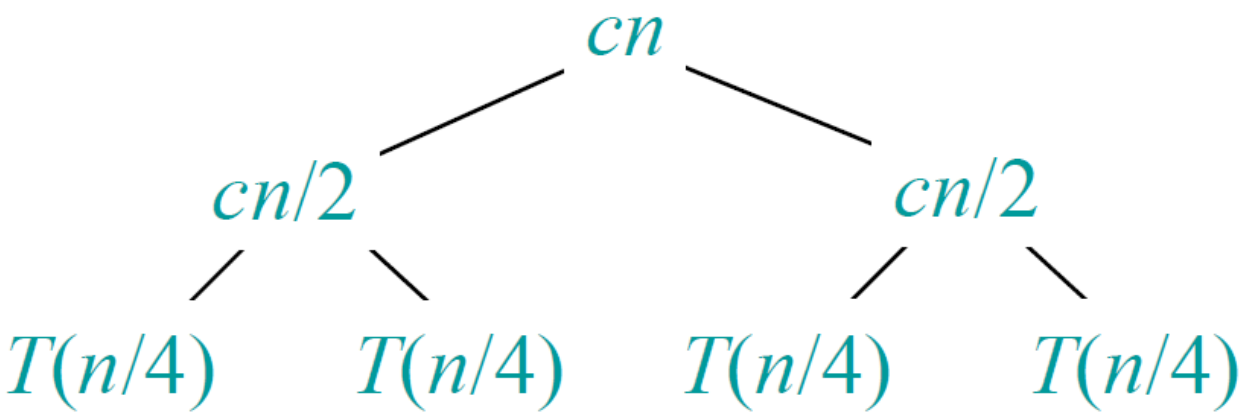
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



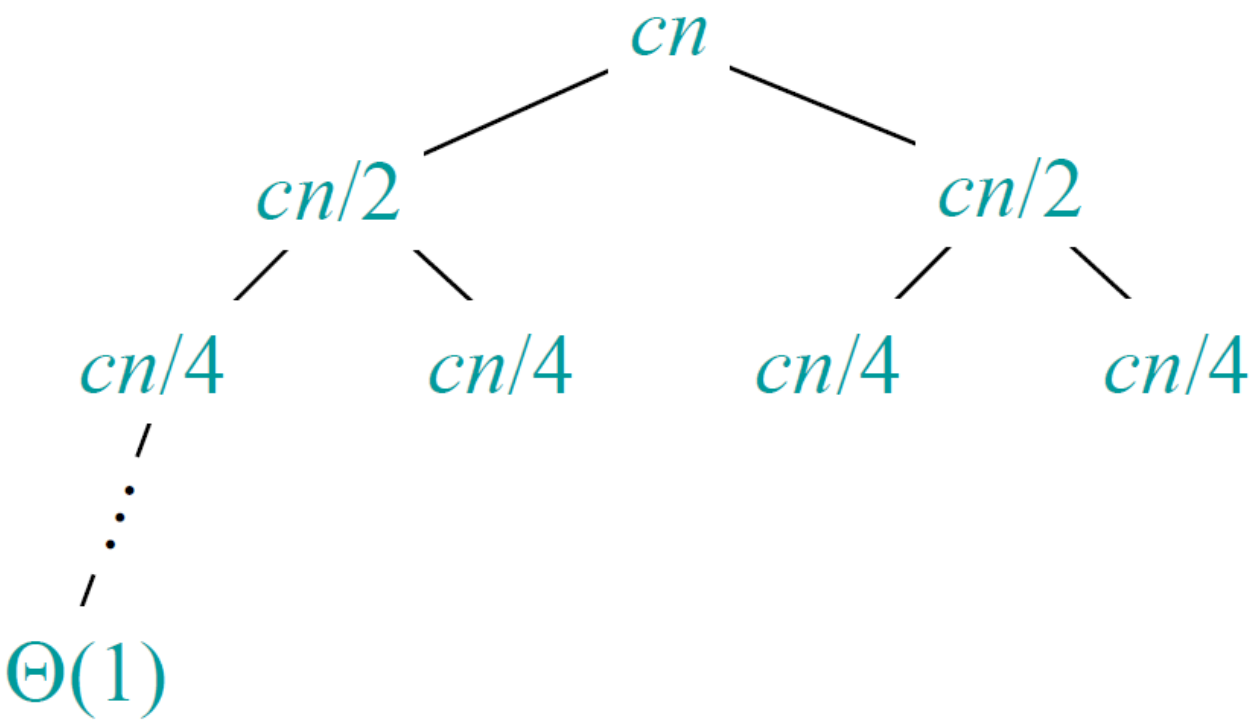
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



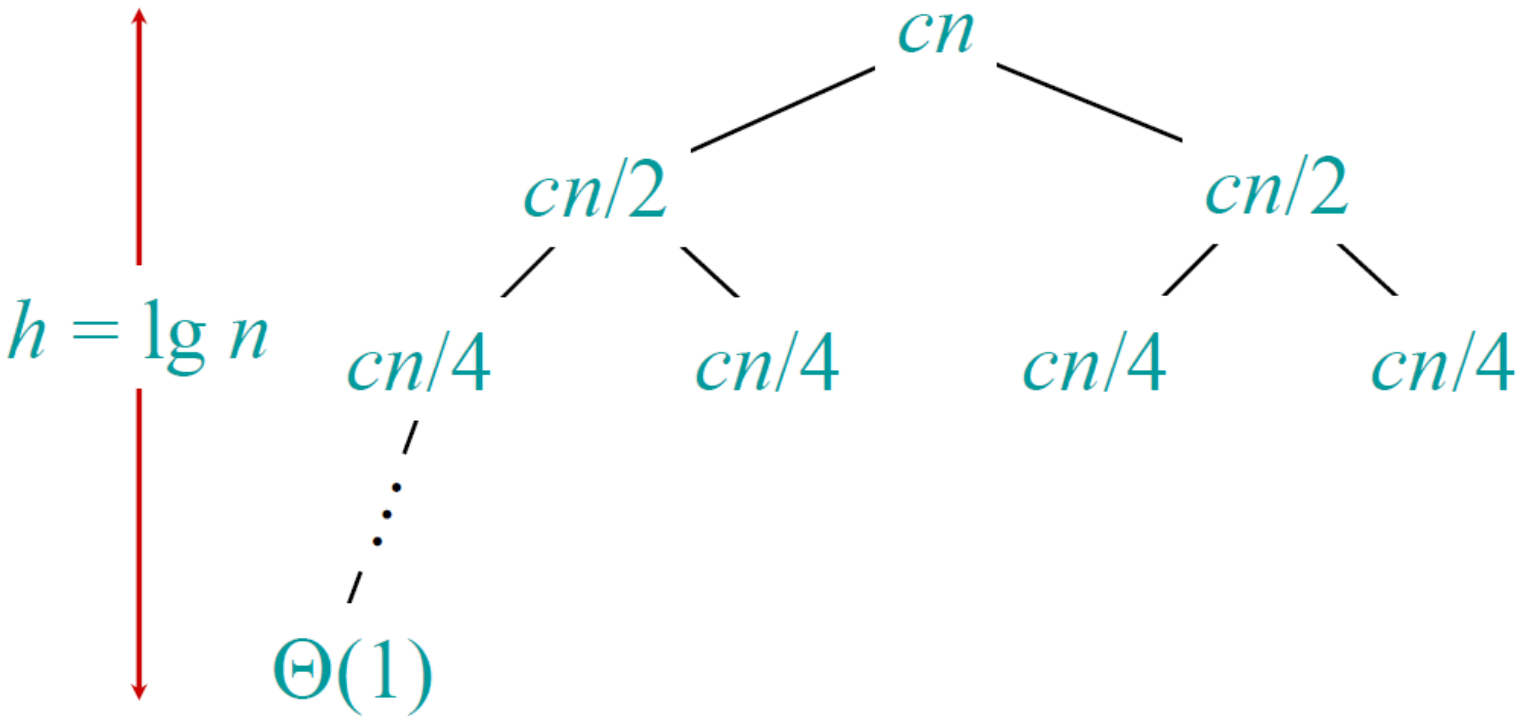
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



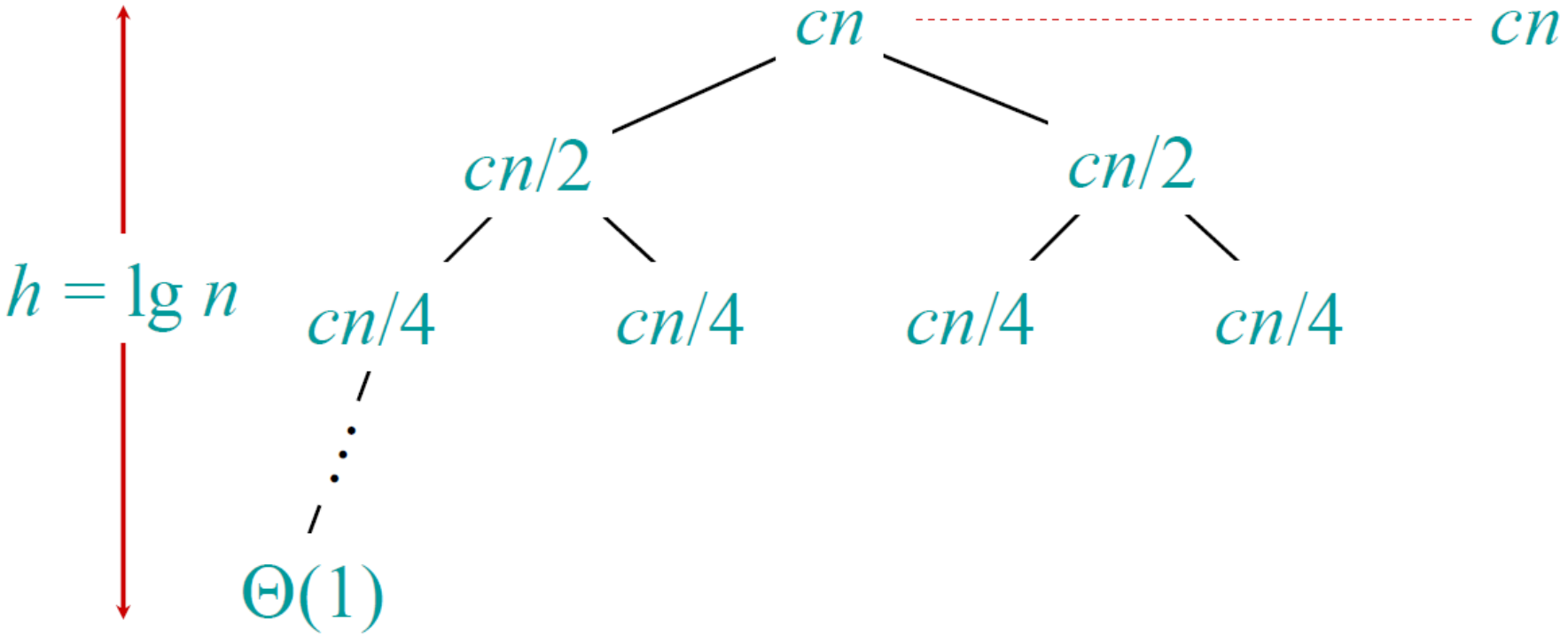
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



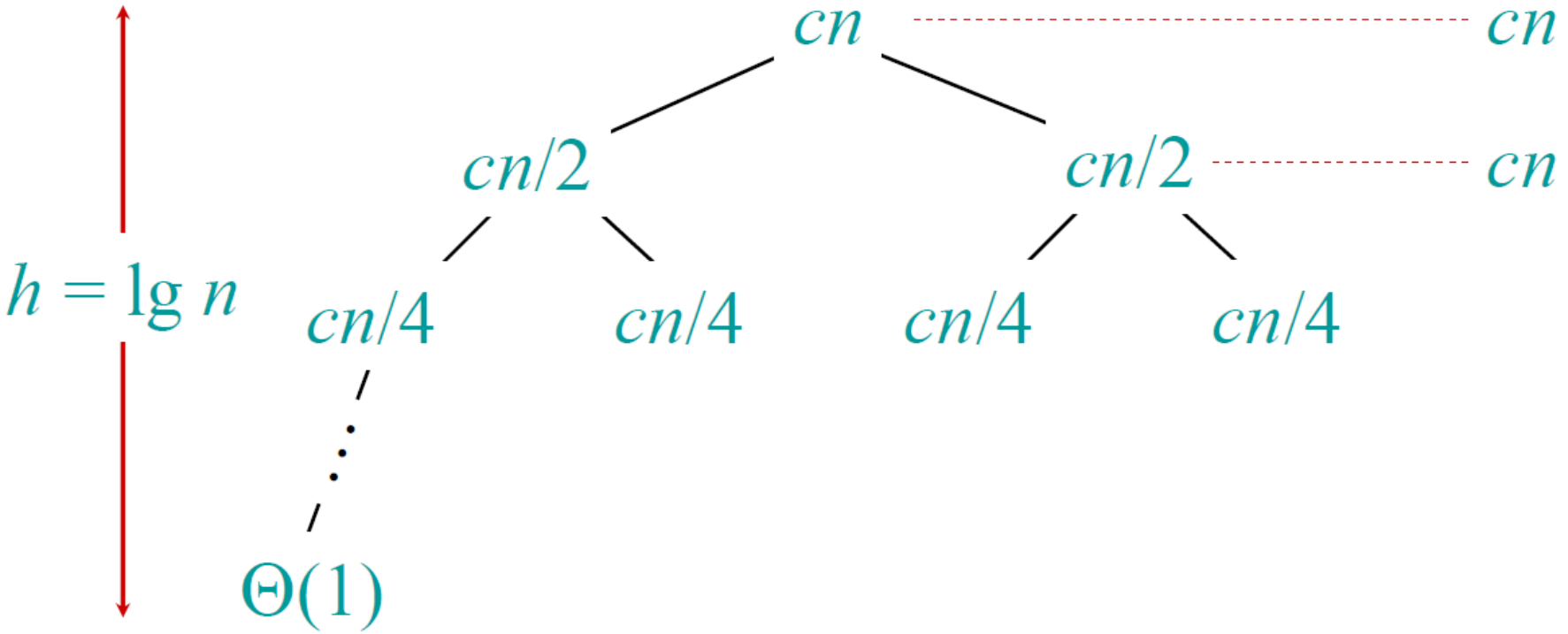
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



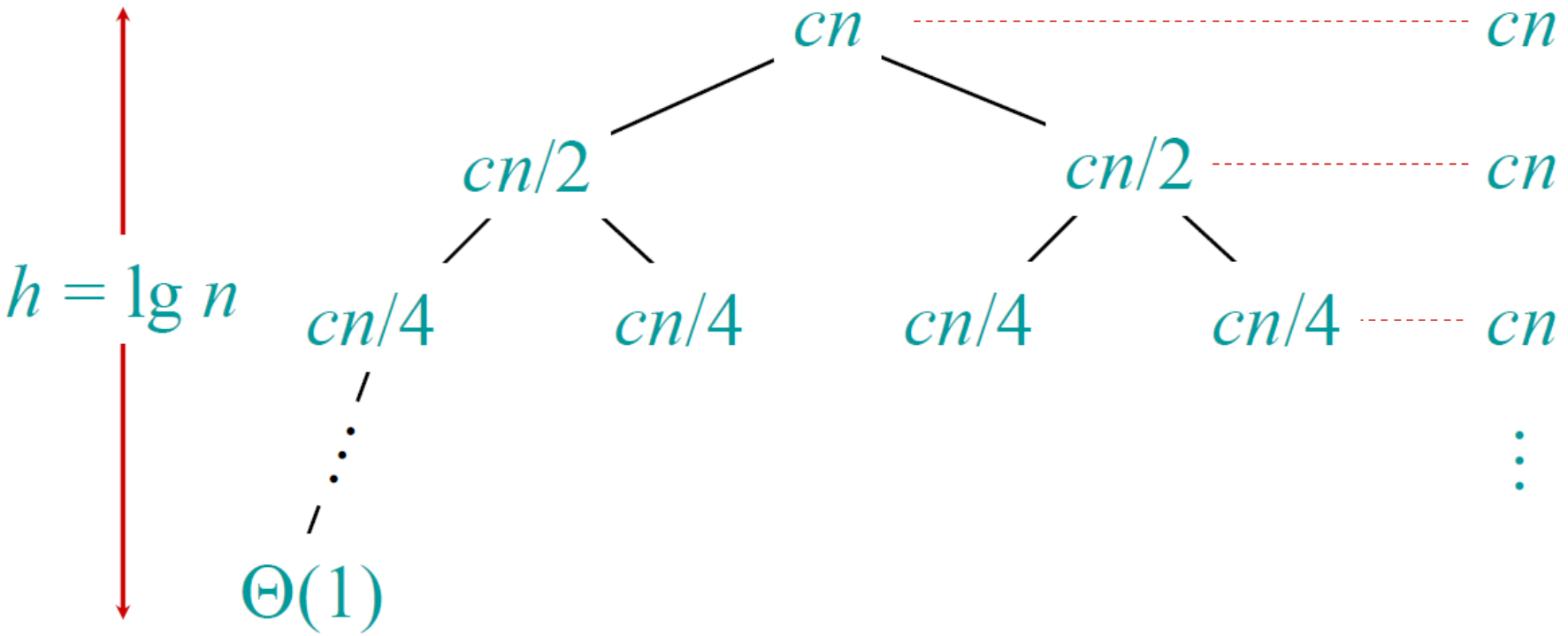
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



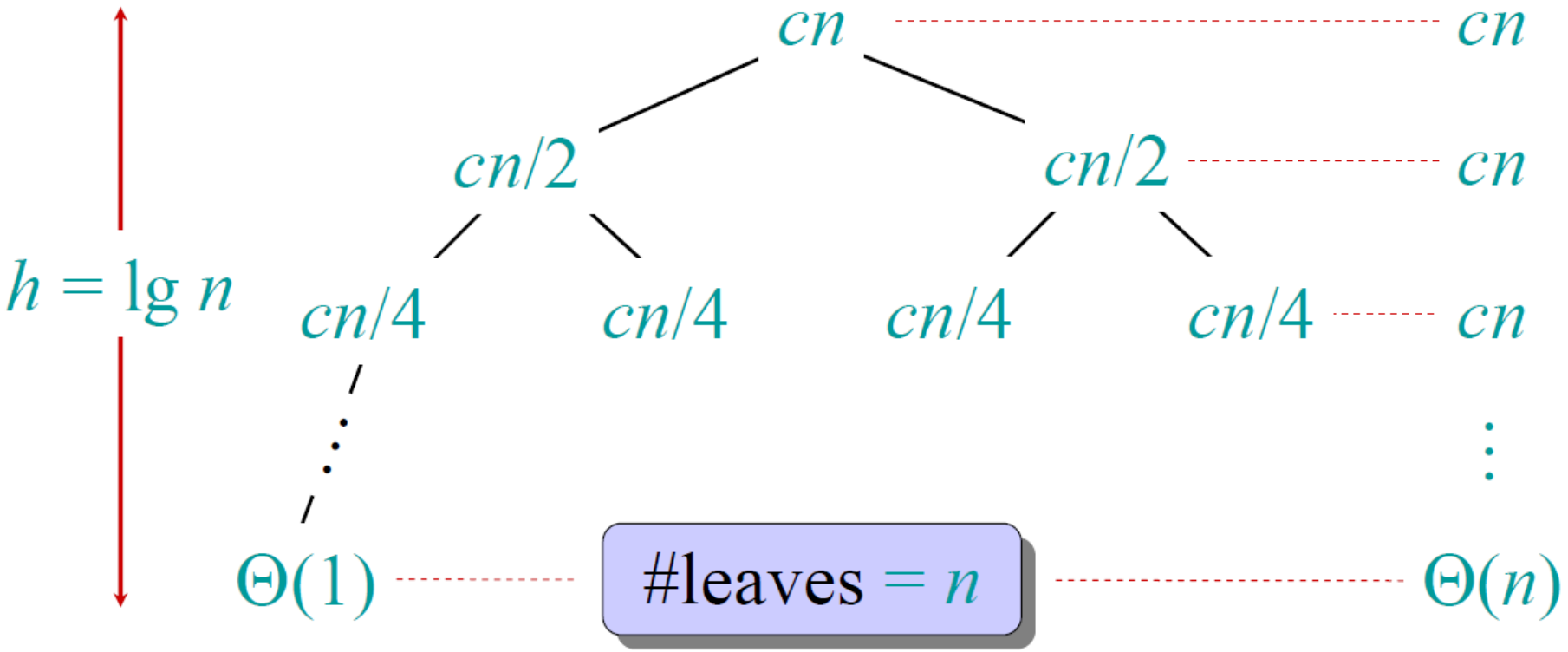
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



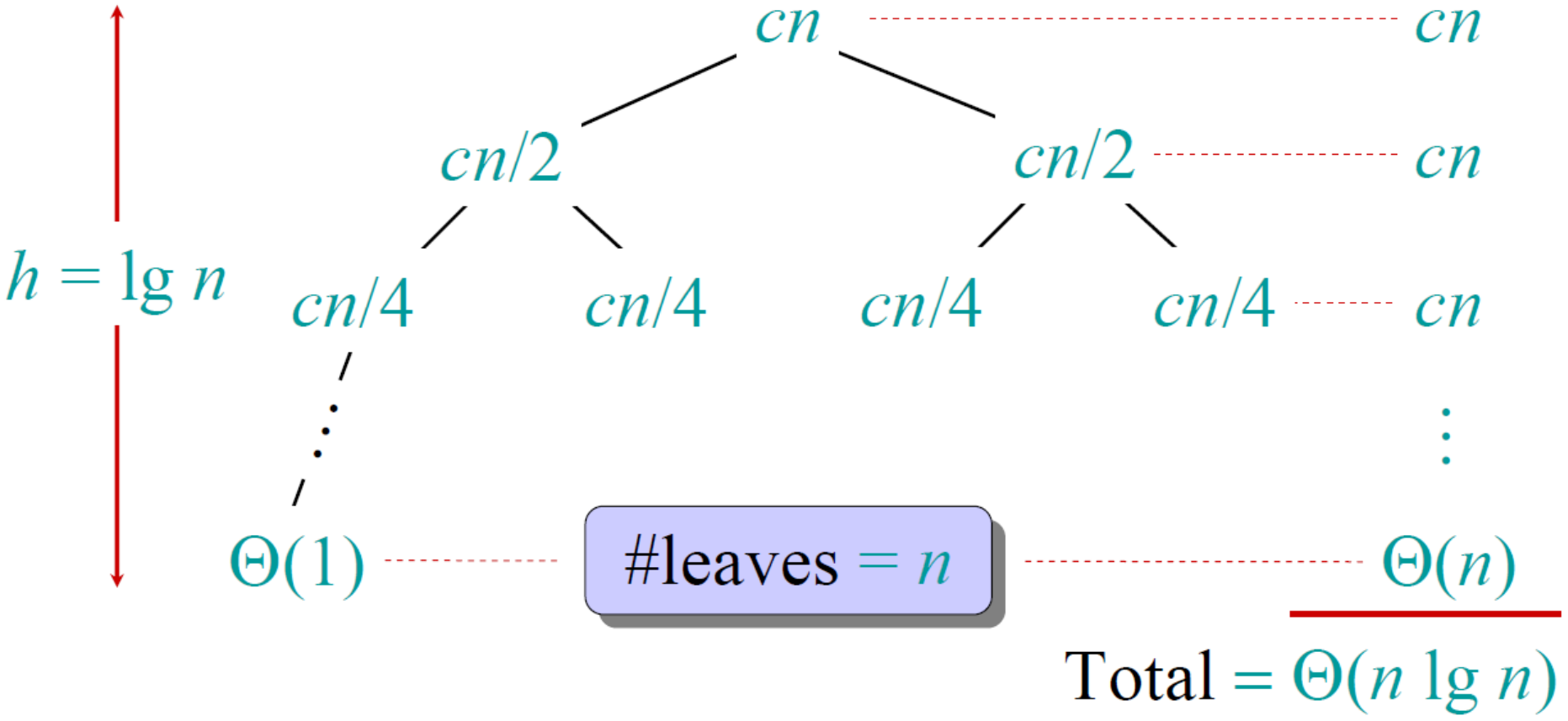
Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Total = $\Theta(n \lg n)$

Conclusions

- Θ(nlgn) grows more slowly than $\Theta(n^2)$
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for ***n > 30*** or so.

- **Homework**
  - Implement the insertion sort algorithm and merging sort algorithm.
  - Test the sorting time with different values of ***n***, then draw the diagram of the time.