

Design and Analysis of Algorithms

Lecture 1: Introduction



Ke Xu and Yongxin Tong
(许可 与 童咏昕)

School of CSE, Beihang University

Outline

- About Us
- Course Details
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- What Are Algorithms
- What Does It Mean to Analyze An Algorithm
- Comparing Time Complexity

Instructors



Ke Xu
Professor

State Key Lab. of Software
Development Environment

Research Interests: Algorithms,
Phase Transitions in NPC, Logic
and Complexity, Data Mining



Yongxin Tong
Researcher

State Key Lab. of Software
Development Environment

Research Interests: Big Data,
Crowd Intelligence, Data Mining,
Federated/Reinforcement Learning

Contact and TAs

● Contact

● Ke Xu

- Office: Room G1131, New Main Building
- Email: kexu@nlsde.buaa.edu.cn
- Homepage: <http://sites.nlsde.buaa.edu.cn/~kexu/>

● Yongxin Tong

- Office: Room 1811, Baiyan Building
- Email: yxtong@buaa.edu.cn
- Homepage: <http://sites.nlsde.buaa.edu.cn/~yxtong/>

● TAs

● Yexuan Shi (Ph.D. Student)

- Email: skyxuan@buaa.edu.cn

● Pengfei Zheng (Master Student)

- Email: zpflyfe@163.com

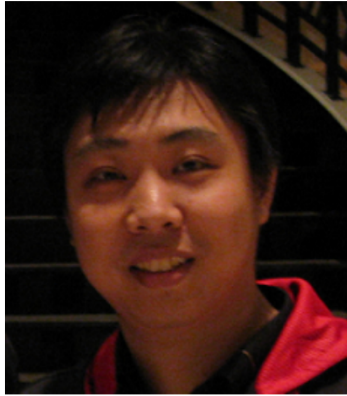
● Lipeng Zhang (Master Student)

- Email: panoszhang@outlook.com

Instructors: Yongxin Tong

- Beihang University (2015.4 - Current)
 - “Zhuoyue Program” Distinguished Researcher
 - State Key Lab. of Software Development Environment
 - Research Interests: **Big Data** and **Crowd Intelligence**
- HKUST (2010.8 – 2015.3)
 - Research Assistant Professor (2014.2 – 2015.3)
 - CSE Department, focused on big data and crowdsourcing
 - Ph.D. Student and Candidate (2010.8 – 2014.1)
 - CSE Department, focused on data mining and crowdsourcing

Instructors: Yongxin Tong



Yongxin Tong 童咏昕

Associate Professor
[State Key Laboratory of Software Development Environment](#)
[School of Computer Science and Engineering](#)
[Beihang University \(BUAA\)](#)

Office: Room 1811, Baiyan Building

E-mail: yxtong AT buaa.edu.cn or yongxintong AT gmail.com

[\[Short Bio\]](#) [\[Research\]](#) [\[Publications\]](#) [\[Awards\]](#) [\[Experiences\]](#) [\[Professional Services\]](#) [\[Misc.\]](#)

Short Biography

Yongxin Tong is an Associate Professor in the [State Key Laboratory of Software Development Environment](#) (SKLSDE) of the [School of Computer Science and Engineering](#) at [Beihang University \(BUAA\)](#). He received a Ph.D. degree in Computing Science and Engineering from the [Department of Computer Science and Engineering, The Hong Kong University of Science and Technology \(HKUST\)](#), under [Prof. Lei Chen](#)'s supervision. He also received a Master degree in Software Engineering at [Beihang University](#) and a Double Bachelor degree in Economics from [China Centre for Economic Research \(CCER\)](#) at [Peking University](#).

Research Interests

- Crowdsourcing
- Spatio-temporal Data Processing and Analysis
- Federated Learning and Distributed Learning
- Uncertain Data Mining and Management
- Social Network Analysis

Our Recent Tutorials and Surveys

- **NSM** Yongxin Tong, Lei Chen, Cyrus Shahabi. "[Spatial Crowdsourcing: Challenges, Techniques, and Applications](#)", in *Proceedings of the 43rd International Conference on Very Large Databases (VLDB 2017)*, Munich, Germany, August 28 - September 1, 2017. [[Tutorial Slides](#)]
- **NSM** Qiang Yang, Yang Liu, Tianjian Chen, **Yongxin Tong**. "[Federated Machine Learning: Concept and Applications](#)", *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2): No.12, February 2019.

Faculty Members in SKLSDE



李未教授



马殿富教授



吕卫锋教授



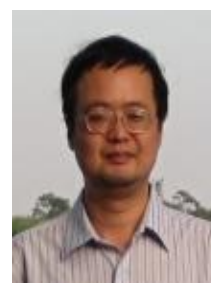
尹宝林教授



蔡维德教授



马世龙教授



张玉平教授



许可教授



张辉教授



郎波教授



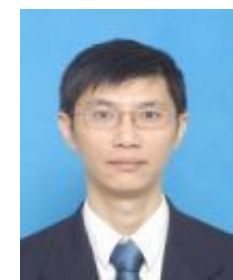
杨钦教授



吴文峻教授



朱燎副教授



诸彤宇副教授



丁嵘副教授



童咏昕副教授



刘瑞副教授



刘祥龙副教授



吕江花博士



孟宪海博士



李吉刚博士



罗杰博士



杜博文博士



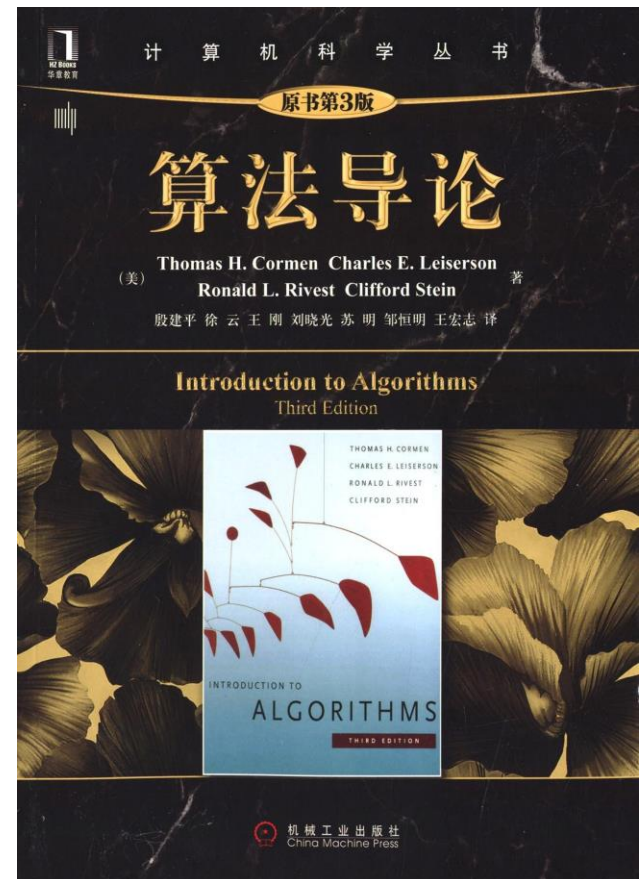
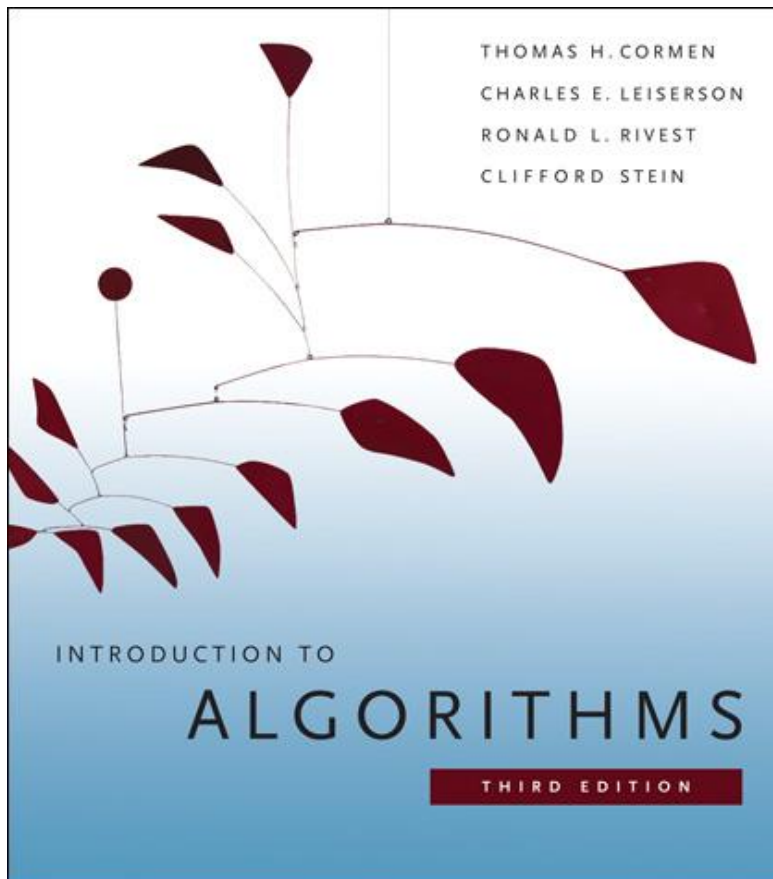
王德庆博士

Outline

- About Us
- **Course Details**
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- What Are Algorithms
- What Does It Mean to Analyze An Algorithm
- Comparing Time Complexity

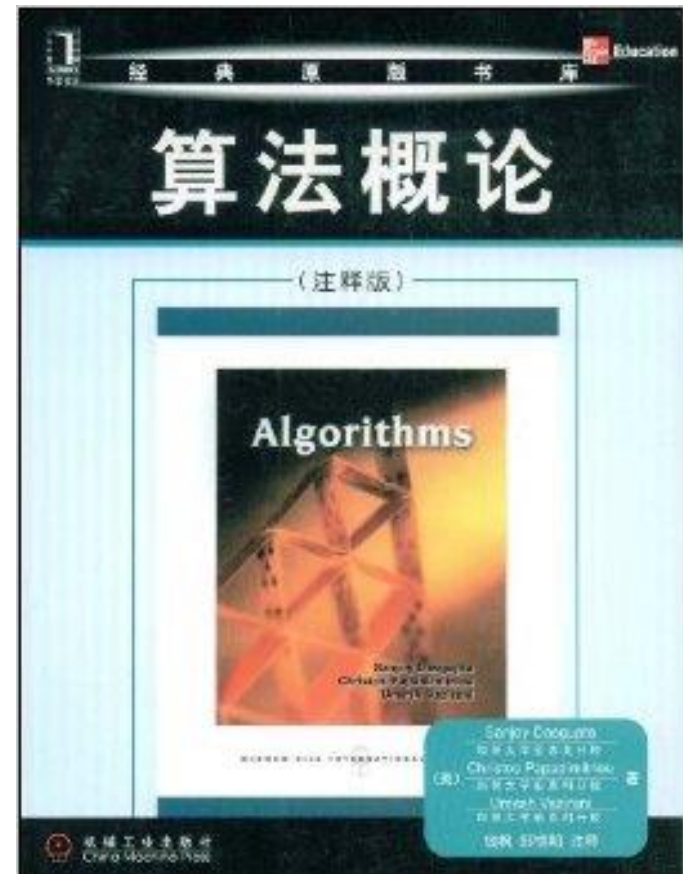
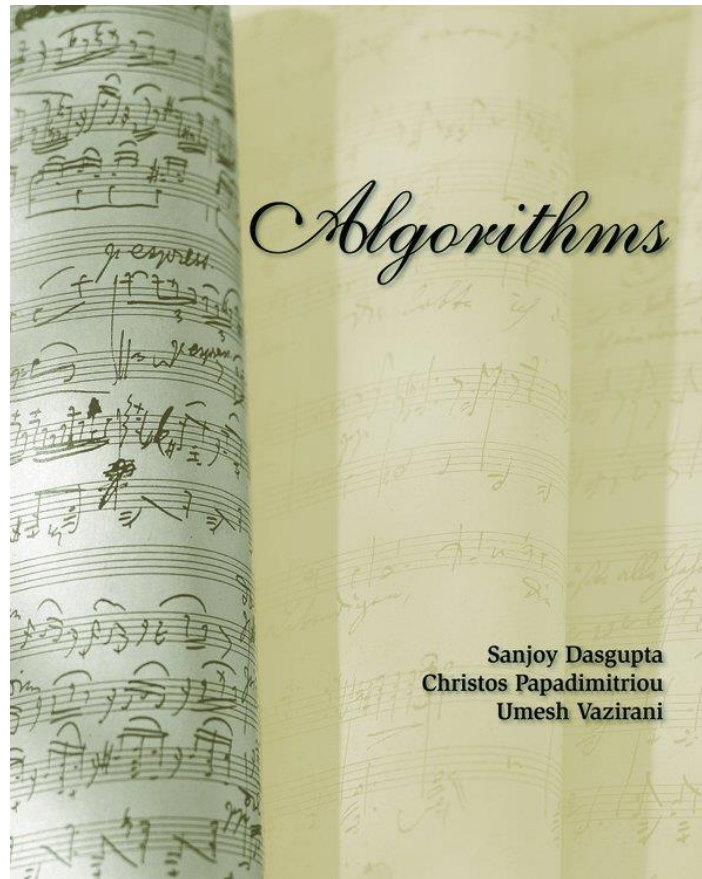
Textbook

- Textbook: *Introduction to Algorithms* (3rd ed.)
 - by Cormen, Leiserson, Rivest and Stein (CLRS)
 - Prepublication version available online



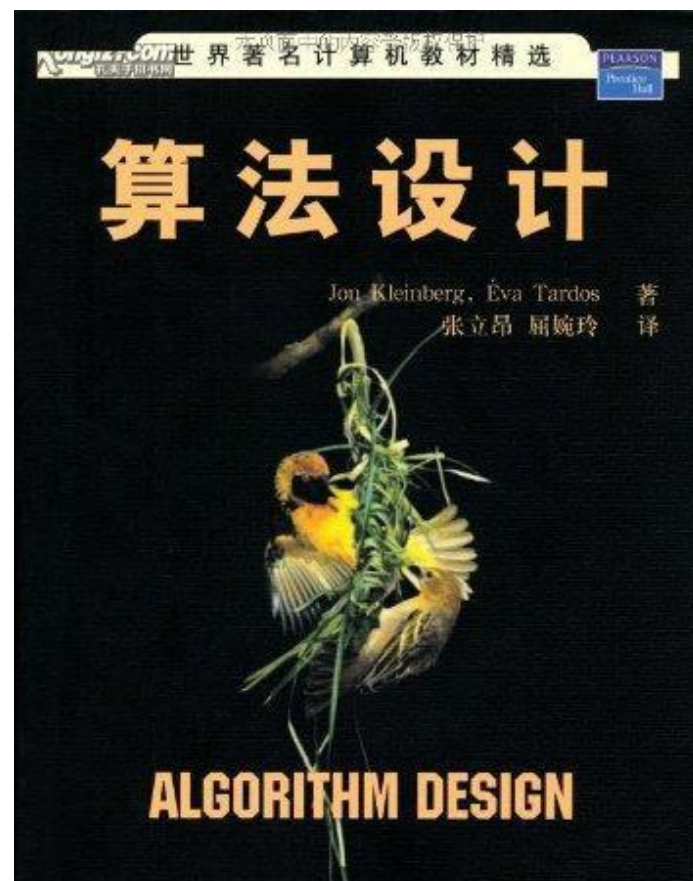
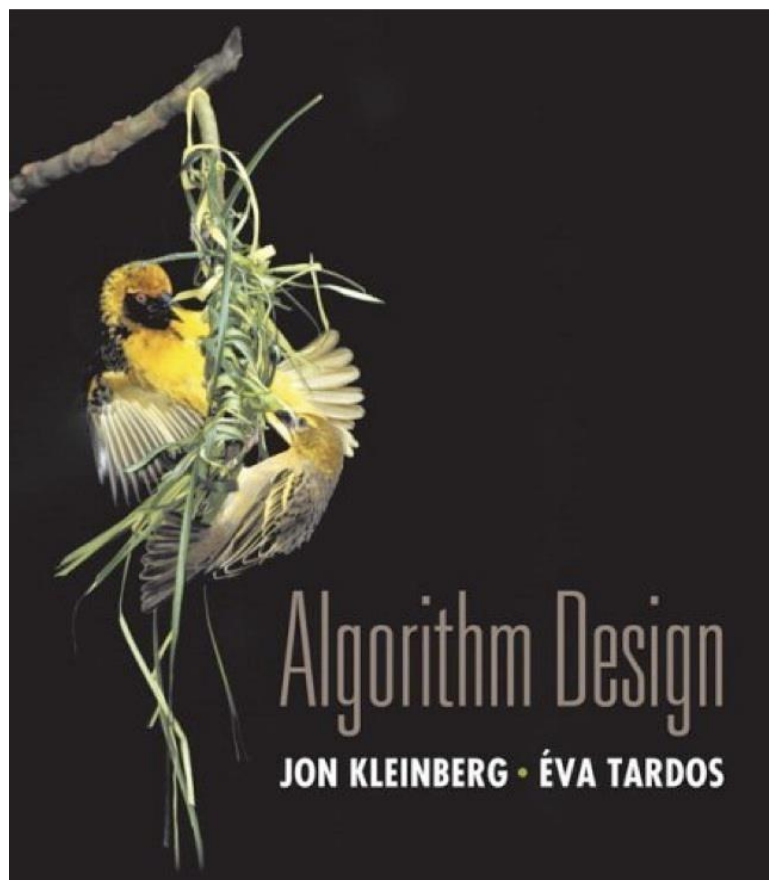
References (1)

- Reference: *Algorithms*
 - by Dasgupta, Papadimitriou, and Vazirani (DPV)
 - Prepublication version available online



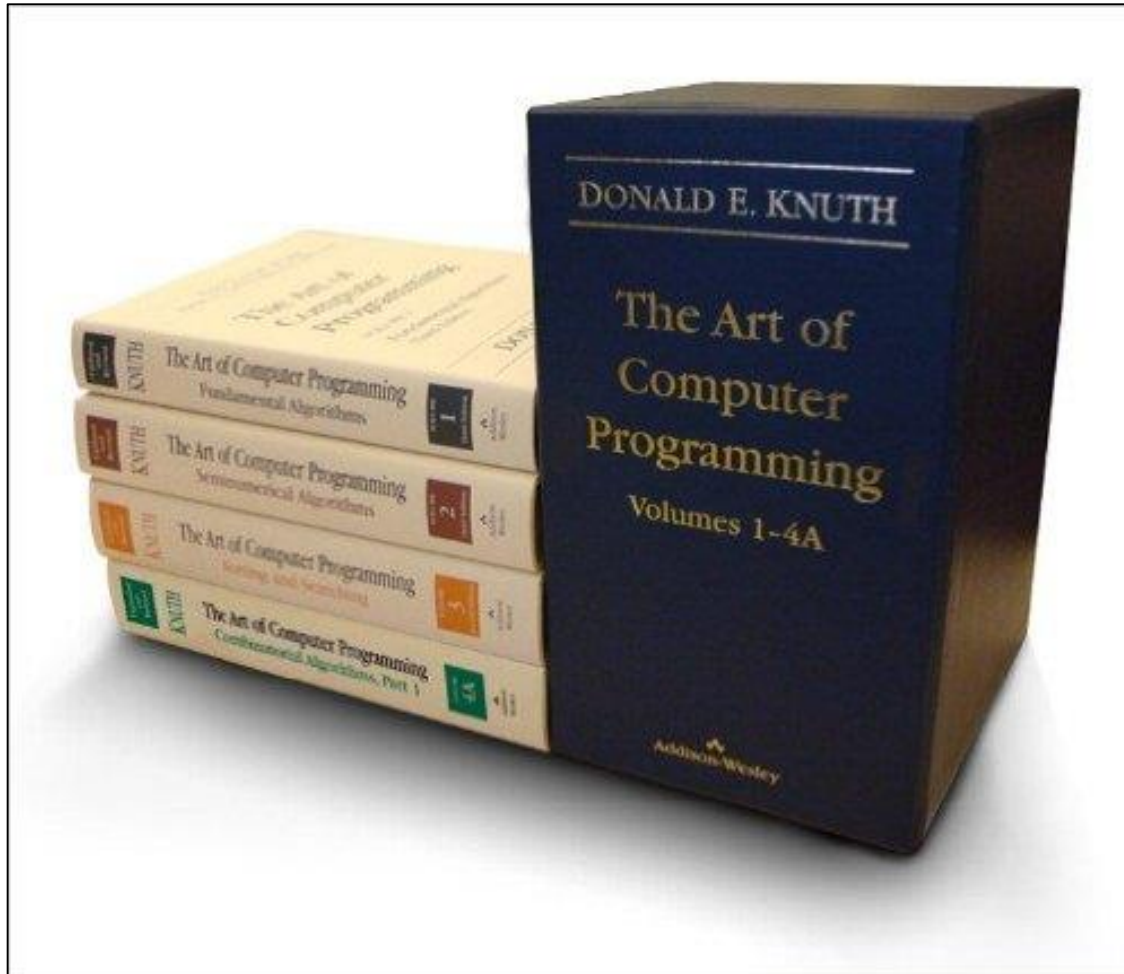
References (2)

- Reference: *Algorithm Design*
 - by Kleinberg and Tardos (KT)



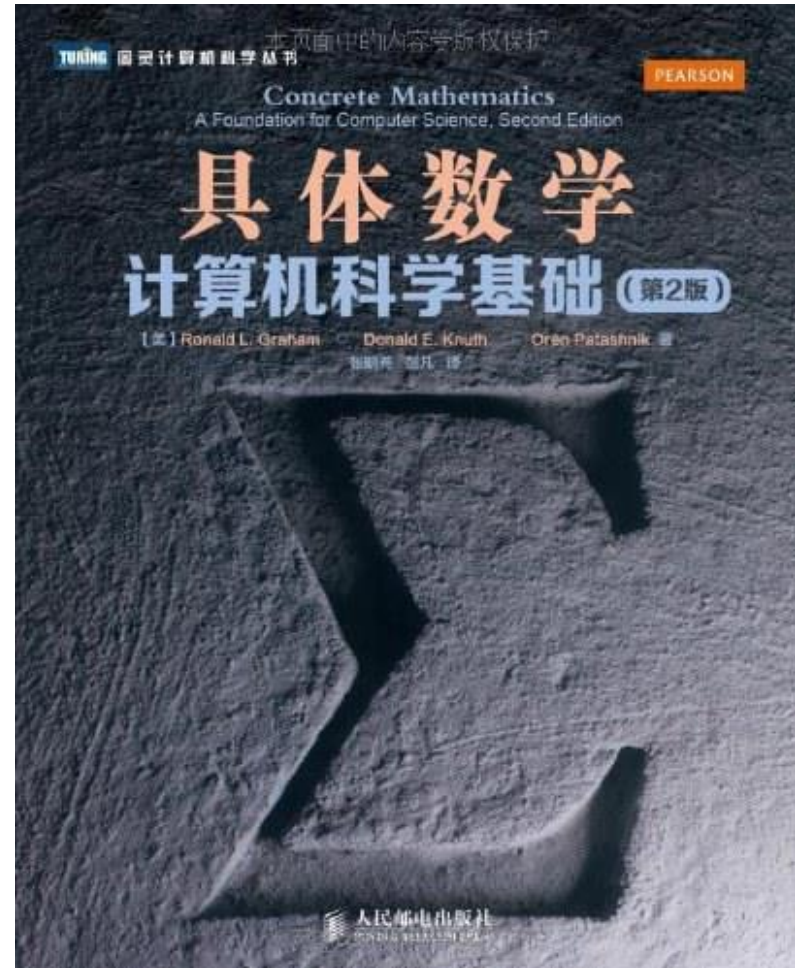
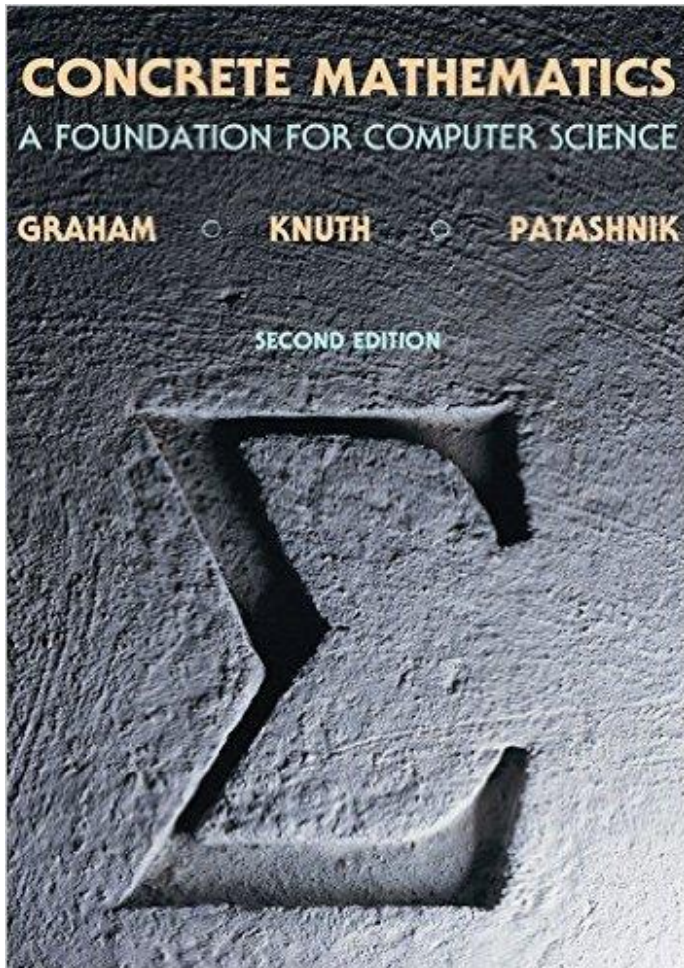
References (3)

- Reference: *The Art of Computer Programming*
 - by Donald E. Knuth



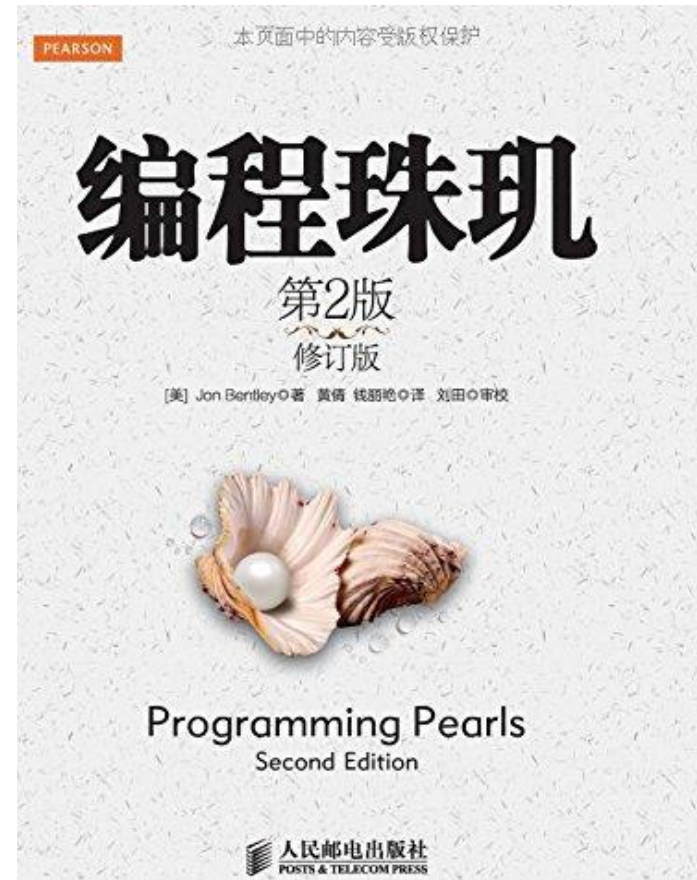
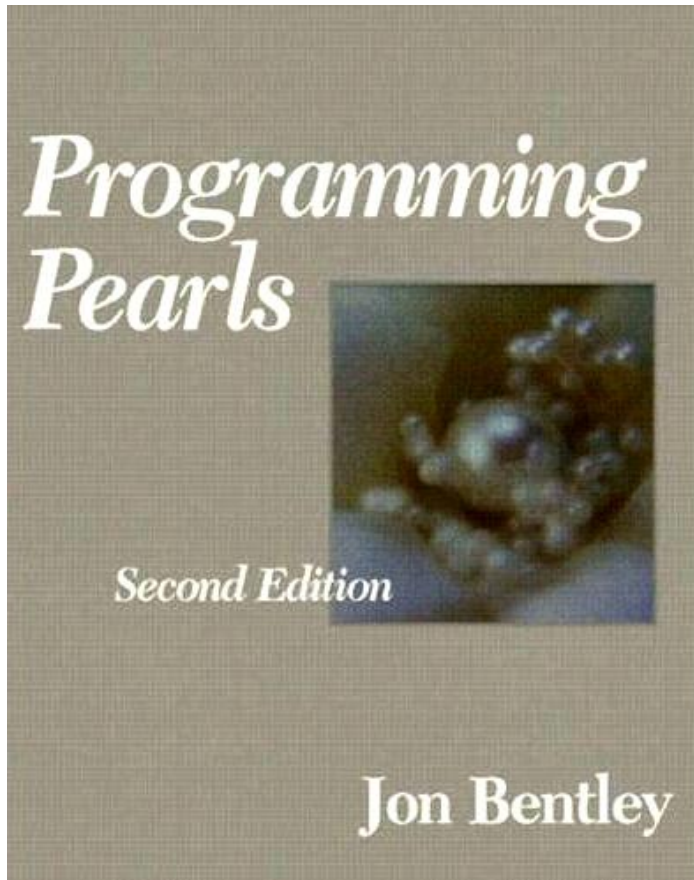
References (4)

- Reference: *Concrete Mathematics* (2nd ed.)
 - by Graham, Knuth, Patashnik (GKP)



References (5)

- Reference: *Programming Pearls* (2nd ed.)
 - by Jon Bentley



Prerequisites

- We assume you know:
 - Linked Lists, Stacks, Queues
 - Binary Search Trees
 - Traversals
 - Searching (but not analysis)
- What have you learnt previously?
 - Graph algorithms
 - Breadth-first search (BFS)
 - Depth-first search (DFS)
 - Topological sort (TS)
 - Minimum Spanning Trees (MST)
 - Dijkstra's shortest path algorithm (SP)

Tentative Syllabus

- Basics

- Asymptotic Notations and Recurrences

- Divide and Conquer Algorithms

- MCS Problem, PM Problem, and Quicksort

- Dynamic Programming Algorithms

- 0-1 Knapsack, Rod-Cutting, CMM, LCS, and MDE

- Greedy Algorithms

- Huffman Coding and Fractional Knapsack

- Graph Algorithms

- BFS, DFS, SP, MST, Max Flow and Matching

- Dealing with Hard Problems

- Problem Classes (P, NP, NPC) and Approximation Alg.

Lectures and Tutorials

- Lectures
 - Slides will be available on course web page.
- Tutorials (补充练习)
 - There will be 12 tutorials in this semester.
 - The tutorials will provide more examples to illustrate the material you learnt in class.
 - The first tutorial will be released on next week.

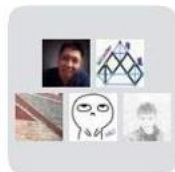
Grading Scheme

- (30%) Four Assignments
 - Each requires designing algorithms and analyzing correctness/running time.
 - Each takes 14 days. The first one will be released on Sep. 27.
 - After each submission due, we will post the solution and **WON'T** accept any assignment.
 - Failing to do any of these will be considered **PLAGIARISM**, and may result in a failing grade in the course.
- (10%) Project
 - Each project is completed by a group.
 - Each group needs to submit a final report and codes.
 - The topics of the project will be released by the end of Sep.
- (60%) Final Exam
 - It covers the materials of the entire semester.

Classroom Etiquette

- **No roll-call in our class !**
- Turn off cell phone ringers.
 - No phone conversations in room.
- Latecomers should enter quietly.
- No LOUD talking among selves during lectures.

WeChat Group



算法设计-6系-2019秋季



该二维码7天内(9月12日前)有效, 重新进入将更新

Outline

- About Us
- Course Details
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- What Are Algorithms
- What Does It Mean to Analyze An Algorithm
- Comparing Time Complexity

A.M. Turing Award



Alan M. Turing

From 2007 to 2013, the award was accompanied by a prize of US \$250,000 by Intel and Google. Since 2014, the award has been accompanied by a prize of US \$1 million by Google.



Nobel Prize of Computing

Since 1966, there have been 70 recipients of A.M. Turing Award!
This year is the 53rd anniversary of A.M. Turing Award!

A.M. Turing Award Winners for Algorithms



Donald E. Knuth
1974, USA



Robert W. Floyd
1978, USA



Stephen A. Cook
1982, USA



Richard M. Karp
1985, USA



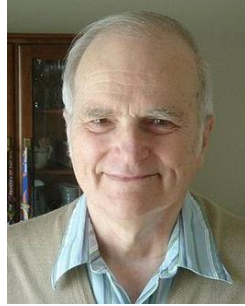
John Hopcroft
1986, USA



Robert Tarjan
1986, USA



Juris Hartmanis
1993, Latvia



Richard E. Stearns
1993, USA



Manuel Blum
1995, Venezuela



Andrew Yao
2000, China



Leslie G. Valiant
2010, Hungarian



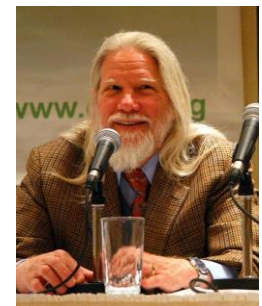
Silvio Micali
2012, Italy



Shafi Goldwasser
2012, USA

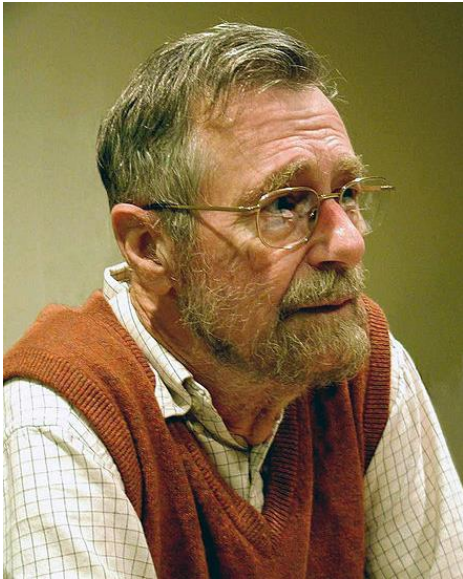


Martin Hellman
2015, USA



Whitfield Diffie
2015, USA

Other Related A.M. Turing Award Winners



Edsger W. Dijkstra

**The Recipient in 1972,
Netherlands,**

**Contributions: ALGOL Father,
Related Work: Dijkstra Algorithm**

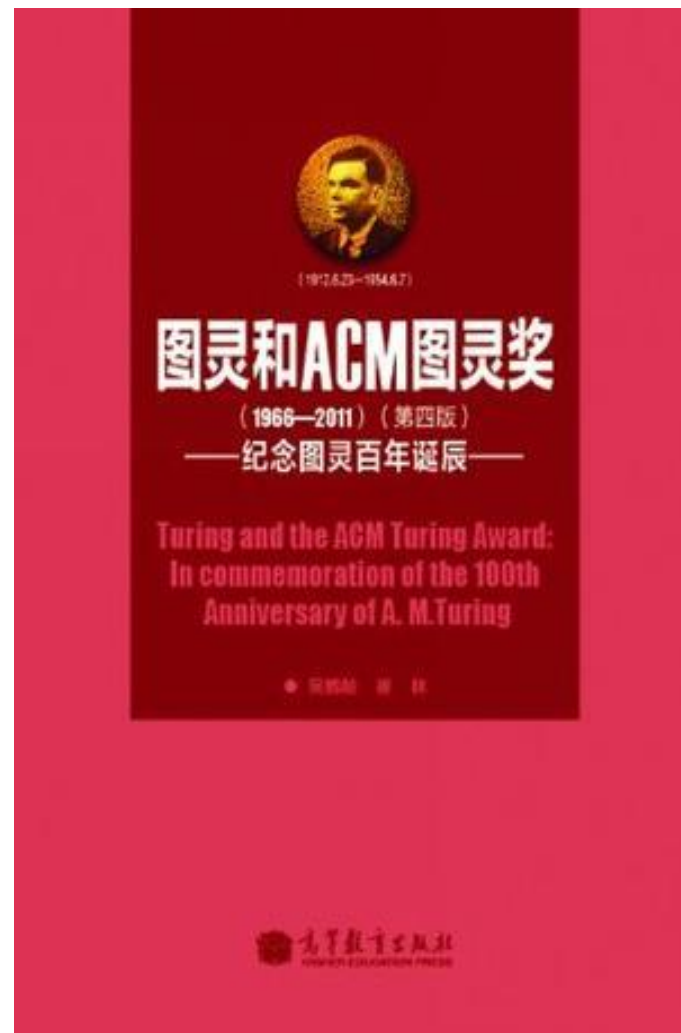
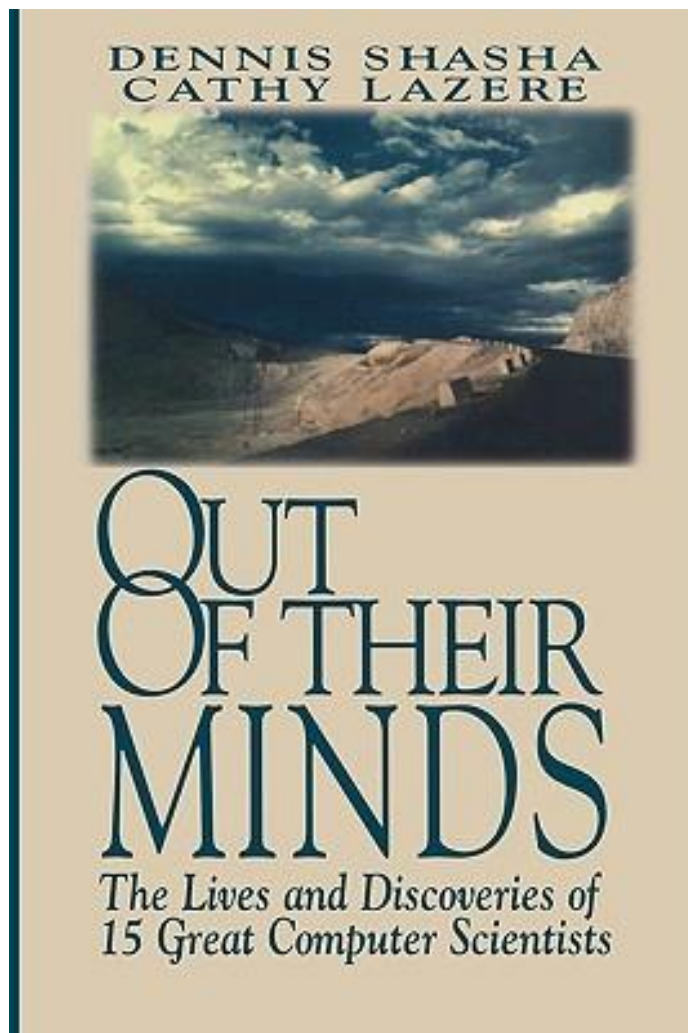


Tony Hoare

**The Recipient in 1980,
UK,**

**Contributions: Hoare logic,
Related Work: QuickSort**

Books of A.M. Turing Award Winners



Outline

- About Us
- Course Details
- A.M. Turing Award Winners for Algorithms
- **What Is This Course About**
- What Are Algorithms
- What Does It Mean to Analyze An Algorithm
- Comparing Time Complexity

What is this course about?

Example (Chain Matrix Multiplication)

$$A = C = \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}.$$

$$B = D = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

Want: $ABCD = ?$

- Method 1: $(AB)(CD)$
- Method 2: $A((BC)D)$

Method 1 is much more efficient than Method 2.
(Expand the expression on board)

What is this course about?

- There is usually more than one algorithm for solving a problem.
- Some algorithms are more efficient than others.
- We want the most efficient algorithm.

What is this course about?

- If we have a number of alternative algorithms for solving a problem, how do we know which is the most efficient?
- To do so, we need to analyze each of them to determine its **efficiency**.
- Of course, we must also make sure the algorithm is **correct**.

What is this course about?

- In this course, we will discuss **fundamental techniques** for:
 - Designing efficient algorithms,
 - Proving the correctness of algorithms,
 - Analyzing the running times of algorithms

What is this course about?

- In this course, we will discuss **fundamental techniques** for:
 - Designing efficient algorithms,
 - Proving the correctness of algorithms,
 - Analyzing the running times of algorithms
- Note:
 - Analysis and design go hand-in-hand:
By analyzing the running times of algorithms, we will know how to design fast algorithms

Outline

- About Us
- Course Details
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- **What Are Algorithms**
- What Does It Mean to Analyze An Algorithm
- Comparing Time Complexity

Computational Problem

Definition

A **computational problem** is a **specification** of the desired input-output relationship

Computational Problem

Definition

A **computational problem** is a **specification** of the desired input-output relationship

Example (Computational Problem)

Sorting

- **Input:** Sequence of n numbers $\langle a_1, \dots, a_n \rangle$
- **Output:** Permutation (reordering)

$$\langle a'_1, a'_2, \dots, a'_n \rangle$$

such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$

Instance

Definition

A **problem instance** is any valid input to the problem.

Instance

Definition

A **problem instance** is any valid input to the problem.

Example (Instance of the Sorting Problem)

$\langle 8, 3, 6, 7, 1, 2, 9 \rangle$

Algorithm

Definition

An **algorithm** is a well defined **computational procedure** that transforms inputs into outputs, achieving the desired input-output relationship

Algorithm

Definition

An **algorithm** is a well defined **computational procedure** that transforms inputs into outputs, achieving the desired input-output relationship

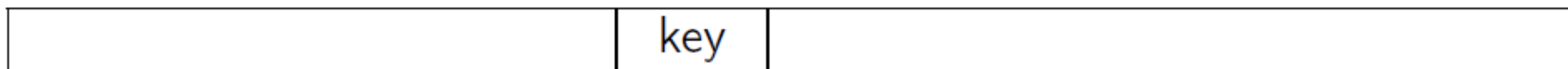
Definition

A **correct algorithm** **halts** with the correct output for every input instance. We can then say that the algorithm **solves** the problem

Example: Insertion Sort

Pseudocode:

```
Input:  $A[1 \dots n]$  is an array of numbers  
for  $j \leftarrow 2$  to  $n$  do  
     $\text{key} \leftarrow A[j];$   
     $i \leftarrow j - 1;$   
    while  $i \geq 1$  and  $A[i] > \text{key}$  do  
         $A[i + 1] \leftarrow A[i];$   
         $i \leftarrow i - 1;$   
    end  
     $A[i + 1] \leftarrow \text{key};$   
end
```



Sorted

Unsorted

Where in the sorted part to put "key"?

How Does It Work?

- An incremental approach: To sort a given array of length n , at the i th step it sorts the array of the first i items by making use of the sorted array of the first $i - 1$ items

Example

Sort $A = \langle 6, 3, 2, 4, 5 \rangle$ with insertion sort

Step 1: $\langle 6, 3, 2, 4, 5 \rangle$

Step 2: $\langle 3, 6, 2, 4, 5 \rangle$

Step 3: $\langle 2, 3, 6, 4, 5 \rangle$

Step 4: $\langle 2, 3, 4, 6, 5 \rangle$

Step 5: $\langle 2, 3, 4, 5, 6 \rangle$

Outline

- About Us
- Course Details
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- What Are Algorithms
- **What Does It Mean to Analyze An Algorithm**
- Comparing Time Complexity

Analyzing Algorithms

- Predict resource utilization
 - Memory (**space complexity**)
 - Running time (**time complexity**) -- focus of this course

Analyzing Algorithms

- Predict resource utilization
 - Memory (**space complexity**)
 - Running time (**time complexity**) -- focus of this course
 - depends on the speed of the computer

Analyzing Algorithms

- Predict resource utilization
 - Memory (**space complexity**)
 - Running time (**time complexity**) -- focus of this course
 - depends on the speed of the computer
 - depends on the implementation details

Analyzing Algorithms

- Predict resource utilization
 - Memory (**space complexity**)
 - Running time (**time complexity**) -- focus of this course
 - depends on the speed of the computer
 - depends on the implementation details
 - depends on the input, especially on the size of the input

Analyzing Algorithms

- Predict resource utilization
 - Memory (**space complexity**)
 - Running time (**time complexity**) -- focus of this course
 - depends on the speed of the computer
 - depends on the implementation details
 - depends on the input, especially on the size of the input
- In light of the above factors, how can we compare different algorithms in terms of their running times?

Analyzing Algorithms

- Predict resource utilization
 - Memory (**space complexity**)
 - Running time (**time complexity**) -- focus of this course
 - depends on the speed of the computer
 - depends on the implementation details
 - depends on the input, especially on the size of the input
- In light of the above factors, how can we compare different algorithms in terms of their running times?
- We want to find a way of measuring running times that is mathematically elegant and machine-independent.

Machine-independent running time

- We will measure the running time as the number of **primitive operations** (e.g., addition, multiplication, comparisons) used by the algorithm

Machine-independent running time

- We will measure the running time as the number of **primitive operations** (e.g., addition, multiplication, comparisons) used by the algorithm
- We will measure the running time as a function of the input size. Let n denote the input size and let $T(n)$ denote the running time for input of size n .

Machine-independent running time

- We will measure the running time as the number of **primitive operations** (e.g., addition, multiplication, comparisons) used by the algorithm
- We will measure the running time as a function of the input size. Let n denote the input size and let $T(n)$ denote the running time for input of size n .
- **Input size n** : rigorous definition given later
 - Sorting: number of items to be sorted

Machine-independent running time

- We will measure the running time as the number of **primitive operations** (e.g., addition, multiplication, comparisons) used by the algorithm
- We will measure the running time as a function of the input size. Let n denote the input size and let $T(n)$ denote the running time for input of size n .
- **Input size n** : rigorous definition given later
 - Sorting: number of items to be sorted
 - Graphs: number of vertices and edges

Three Kinds of Analysis: I

Best Case: An instance for a given size n that results in the fastest possible running time.

Three Kinds of Analysis: I

Best Case: An instance for a given size n that results in the fastest possible running time.

Example (Insertion sort)

$$A[1] \leq A[2] \leq A[3] \leq \dots \leq A[n]$$

Three Kinds of Analysis: I

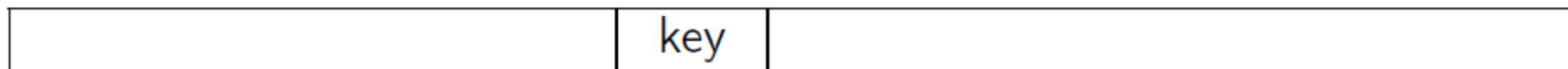
Best Case: An instance for a given size n that results in the fastest possible running time.

Example (Insertion sort)

$$A[1] \leq A[2] \leq A[3] \leq \dots \leq A[n]$$

The number of comparisons needed is equal to

$$\underbrace{1 + 1 + 1 + \dots + 1}_{n-1} = n - 1 = \Theta(n)$$



Sorted

Unsorted

“key” is compared to only the element right before it.

Three Kinds of Analysis: II

Worst Case: An instance for a given size n that results in the slowest possible running time.

Three Kinds of Analysis: II

Worst Case: An instance for a given size n that results in the slowest possible running time.

Example (Insertion sort)

$$A[1] \geq A[2] \geq A[3] \geq \dots \geq A[n]$$

Three Kinds of Analysis: II

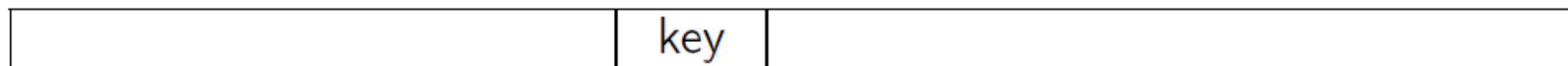
Worst Case: An instance for a given size n that results in the **slowest** possible running time.

Example (Insertion sort)

$$A[1] \geq A[2] \geq A[3] \geq \dots \geq A[n]$$

The number of comparisons needed is equal to

$$1 + 2 + \dots + (n - 1) = \frac{n(n - 1)}{2} = \Theta(n^2)$$



Sorted

Unsorted

“key” is compared to everything element before it.

Three Kinds of Analysis: III

Average Case: Running time averaged over **all possible** instances for the given size, assuming some probability distribution on the instances.

Three Kinds of Analysis: III

Average Case: Running time averaged over **all possible** instances for the given size, assuming some probability distribution on the instances.

Example (Insertion sort)

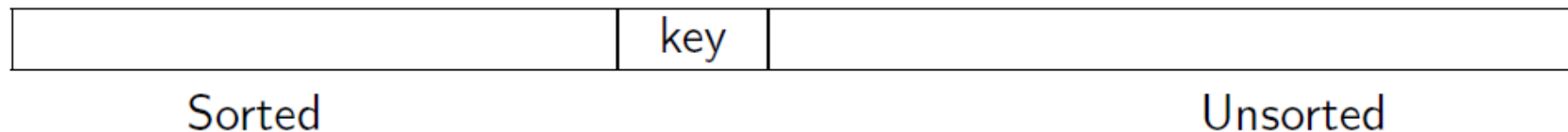
$\Theta(n^2)$, assuming that each of the $n!$ instances is equally likely (uniform distribution).

Three Kinds of Analysis: III

Average Case: Running time averaged over **all possible** instances for the given size, assuming some probability distribution on the instances.

Example (Insertion sort)

$\Theta(n^2)$, assuming that each of the $n!$ instances is equally likely (uniform distribution).



On average, “key” is compared to half of the elements before it.

Three Kinds of Analysis

- Best case: Clearly useless

Three Kinds of Analysis

- Best case: Clearly useless
- **Worst case**: Commonly used, will also be used in this course
 - Gives a running time guarantee no matter what the input is
 - Fair comparison among different algorithms

Three Kinds of Analysis

- Best case: Clearly useless
- **Worst case**: Commonly used, will also be used in this course
 - Gives a running time guarantee no matter what the input is
 - Fair comparison among different algorithms
- Average case: Used sometimes
 - Need to assume some distribution: real-world inputs are seldom uniformly random!
 - Analysis is complicated

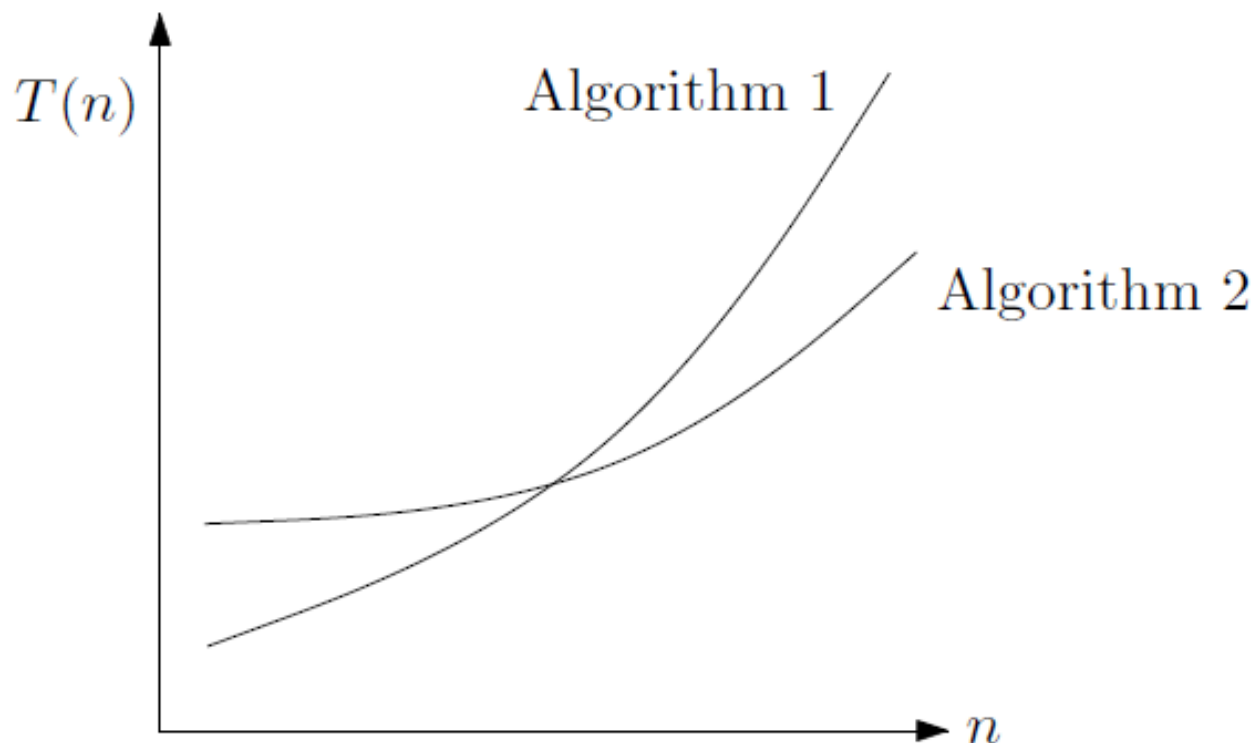
Three Kinds of Analysis

- Best case: Clearly useless
- **Worst case**: Commonly used, will also be used in this course
 - Gives a running time guarantee no matter what the input is
 - Fair comparison among different algorithms
- Average case: Used sometimes
 - Need to assume some distribution: real-world inputs are seldom uniformly random!
 - Analysis is complicated
 - Will not be used in this course

Outline

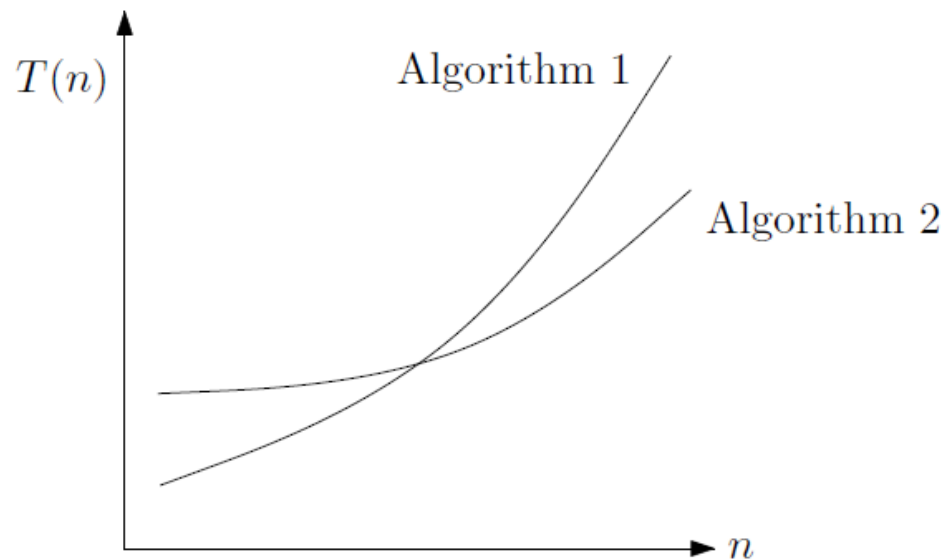
- About Us
- Course Details
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- What Are Algorithms
- What Does It Mean to Analyze An Algorithm
- **Comparing Time Complexity**

Comparing Time Complexity



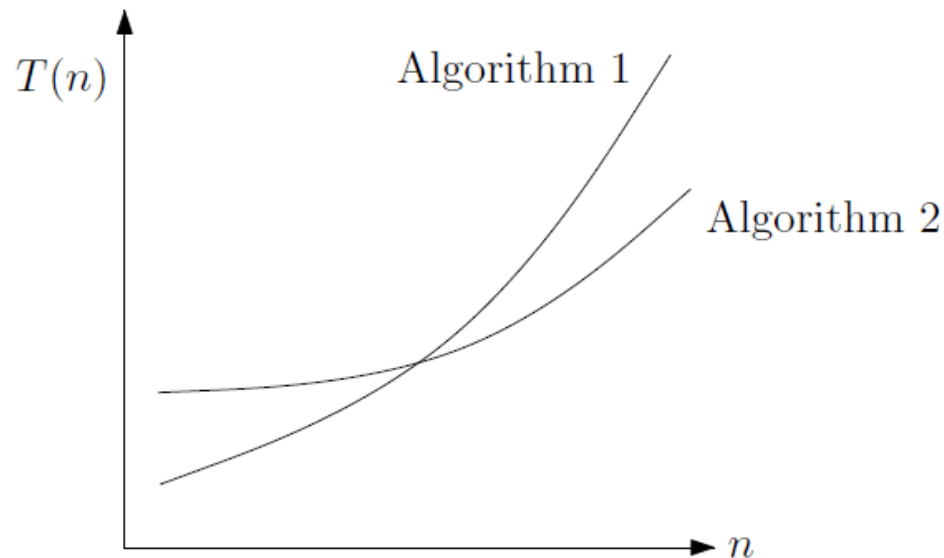
- Which algorithm is superior for large n ?
 - $T(n)$ for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17$
 - $T(n)$ for Algorithm 2 is $7n^2 - 8n + 20$
- Clearly, Algorithm 2 is superior.

Asymptotic Analysis



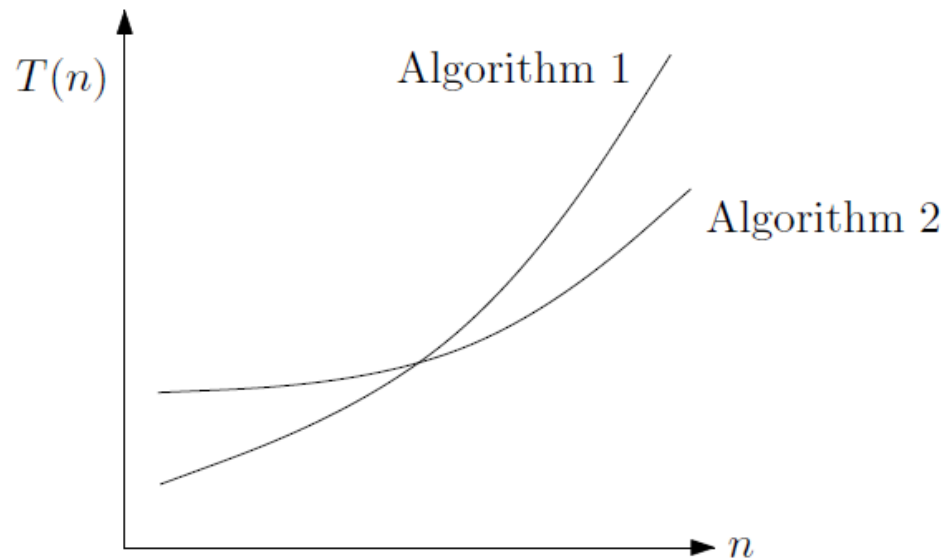
- $T(n)$ for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$

Asymptotic Analysis



- $T(n)$ for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- $T(n)$ for Algorithm 2 is $7n^2 - 8n + 20 = \Theta(n^2)$

Asymptotic Analysis

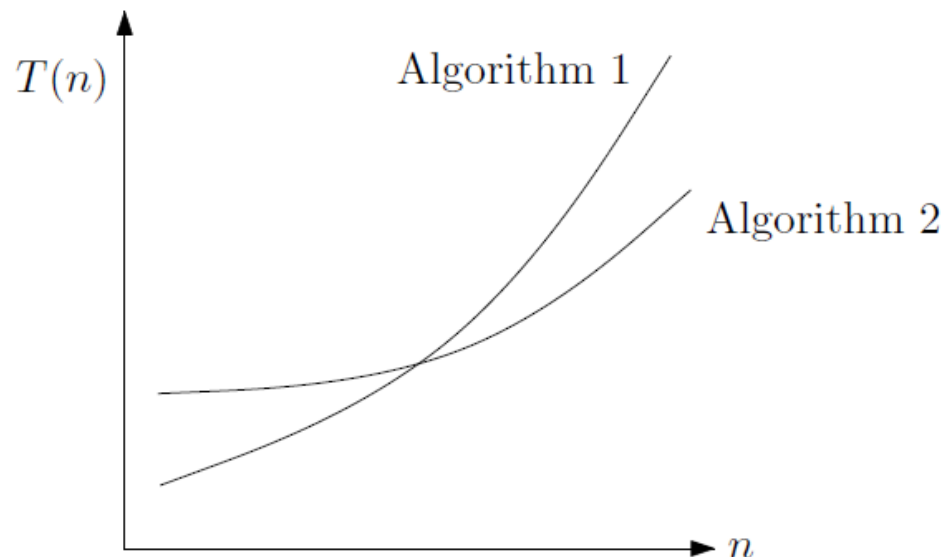


- $T(n)$ for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- $T(n)$ for Algorithm 2 is $7n^2 - 8n + 20 = \Theta(n^2)$

Θ -notation

- Drop low-order terms; ignore leading constants

Asymptotic Analysis

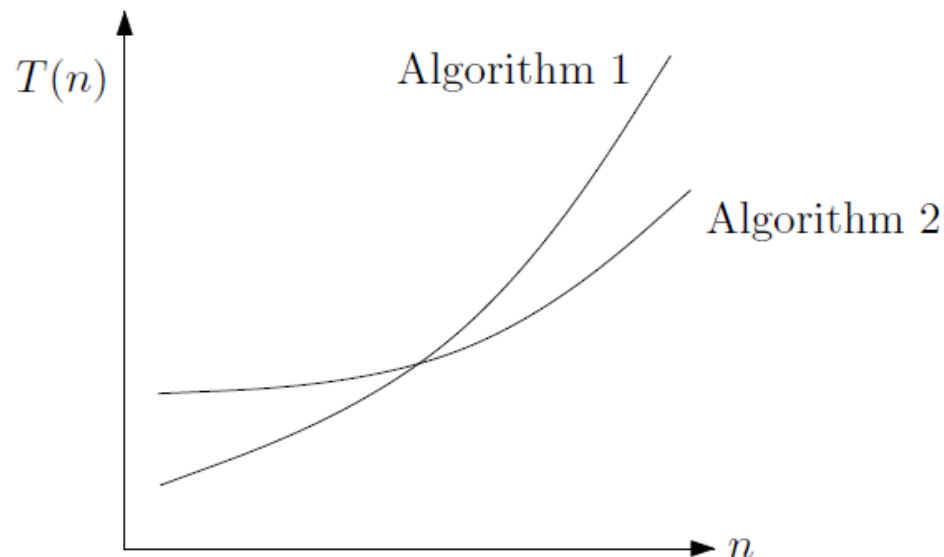


- $T(n)$ for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- $T(n)$ for Algorithm 2 is $7n^2 - 8n + 20 = \Theta(n^2)$

Θ -notation

- Drop low-order terms; ignore leading constants
- Look at growth of $T(n)$ as $n \rightarrow \infty$

Asymptotic Analysis



- $T(n)$ for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- $T(n)$ for Algorithm 2 is $7n^2 - 8n + 20 = \Theta(n^2)$

Θ -notation

- Drop low-order terms; ignore leading constants
- Look at growth of $T(n)$ as $n \rightarrow \infty$
- When n is large enough, a $\Theta(n^2)$ algorithm **always** beats a $\Theta(n^3)$ algorithm

Merge Sort

Mergesort(*A*, *left*, *right*)

```
if left < right then  
    center  $\leftarrow \lfloor (\text{left} + \text{right}) / 2 \rfloor$ ;  
    Mergesort(A, left, center);  
    Mergesort(A, center+1, right);  
    “Merge” the two sorted arrays;  
end
```

- To sort the entire array $A[1 \dots n]$, we make the initial call Mergesort(*A*, 1, *n*).

Merge Sort

Mergesort(*A*, *left*, *right*)

```
if left < right then  
    center  $\leftarrow \lfloor (\text{left} + \text{right})/2 \rfloor$ ;  
    Mergesort(A, left, center);  
    Mergesort(A, center+1, right);  
    “Merge” the two sorted arrays;  
end
```

- To sort the entire array $A[1 \dots n]$, we make the initial call Mergesort(*A*, 1, *n*).
- Key subroutine: “Merge”

Merge two sorted arrays

--	--	--	--	--	--	--	--

3	6	9	16
---	---	---	----

2	5	8	12
---	---	---	----

Merge two sorted arrays

2							
---	--	--	--	--	--	--	--

3	6	9	16
---	---	---	----

	5	8	12
--	---	---	----

Merge two sorted arrays

2	3						
---	---	--	--	--	--	--	--

	6	9	16
--	---	---	----

	5	8	12
--	---	---	----

Merge two sorted arrays

2	3	5					
---	---	---	--	--	--	--	--

	6	9	16
--	---	---	----

		8	12
--	--	---	----

Merge two sorted arrays

2	3	5	6				
---	---	---	---	--	--	--	--

		9	16
--	--	---	----

		8	12
--	--	---	----

Merge two sorted arrays

2	3	5	6	8			
---	---	---	---	---	--	--	--

		9	16
--	--	---	----

			12
--	--	--	----

Merge two sorted arrays

2	3	5	6	8	9		
---	---	---	---	---	---	--	--

			16
--	--	--	----

			12
--	--	--	----

Merge two sorted arrays

2	3	5	6	8	9	12	
---	---	---	---	---	---	----	--

			16
--	--	--	----

--	--	--	--

Merge two sorted arrays

2	3	5	6	8	9	12	16
---	---	---	---	---	---	----	----

--	--	--	--

--	--	--	--

Three Kinds of Analysis

- $T(n)$: time needed to run Mergesort($A, 1, n$)
- Assume n is a power of 2 for simplicity

Mergesort(A , left, right)

```
if left < right then  
    center  $\leftarrow \lfloor (left + right)/2 \rfloor$ ;  
    Mergesort( $A$ , left, center); //  $T(n/2)$ 
```

Three Kinds of Analysis

- $T(n)$: time needed to run Mergesort($A, 1, n$)
- Assume n is a power of 2 for simplicity

Mergesort(A , left, right)

```
if left < right then
  center  $\leftarrow \lfloor (\text{left} + \text{right}) / 2 \rfloor$ ;
  Mergesort( $A$ , left, center); //  $T(n/2)$ 
  Mergesort( $A$ , center+1, right); //  $T(n/2)$ 
```

Three Kinds of Analysis

- $T(n)$: time needed to run Mergesort($A, 1, n$)
- Assume n is a power of 2 for simplicity

Mergesort(A , left, right)

```
if left < right then
  center ← ⌊(left + right)/2⌋;
  Mergesort( $A$ , left, center); //  $T(n/2)$ 
  Mergesort( $A$ , center+1, right); //  $T(n/2)$ 
  "Merge" the two sorted arrays; //  $\Theta(n)$ 
end
```

Three Kinds of Analysis

- $T(n)$: time needed to run Mergesort($A, 1, n$)
- Assume n is a power of 2 for simplicity

Mergesort(A , left, right)

```
if left < right then
  center ← ⌊(left + right)/2⌋;
  Mergesort( $A$ , left, center); //  $T(n/2)$ 
  Mergesort( $A$ , center+1, right); //  $T(n/2)$ 
  “Merge” the two sorted arrays; //  $\Theta(n)$ 
end
```

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

Three Kinds of Analysis

Solve

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

Three Kinds of Analysis

Solve

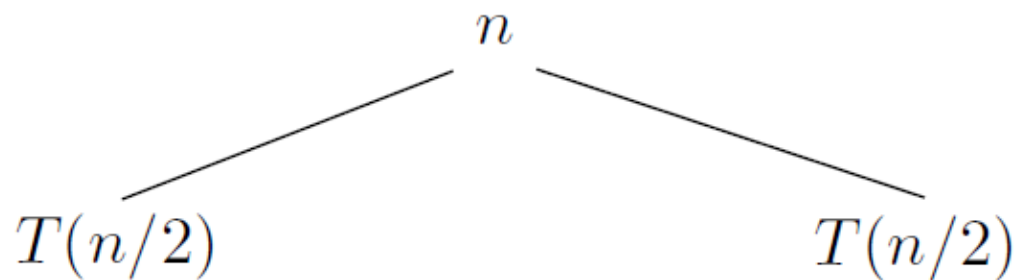
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

$$T(n)$$

Three Kinds of Analysis

Solve

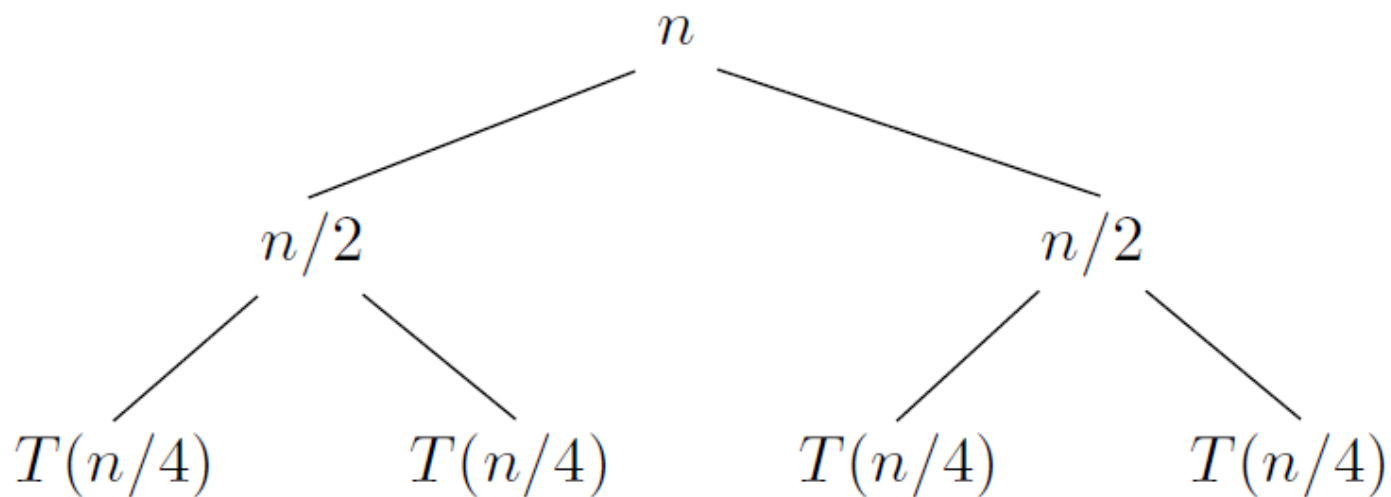
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Three Kinds of Analysis

Solve

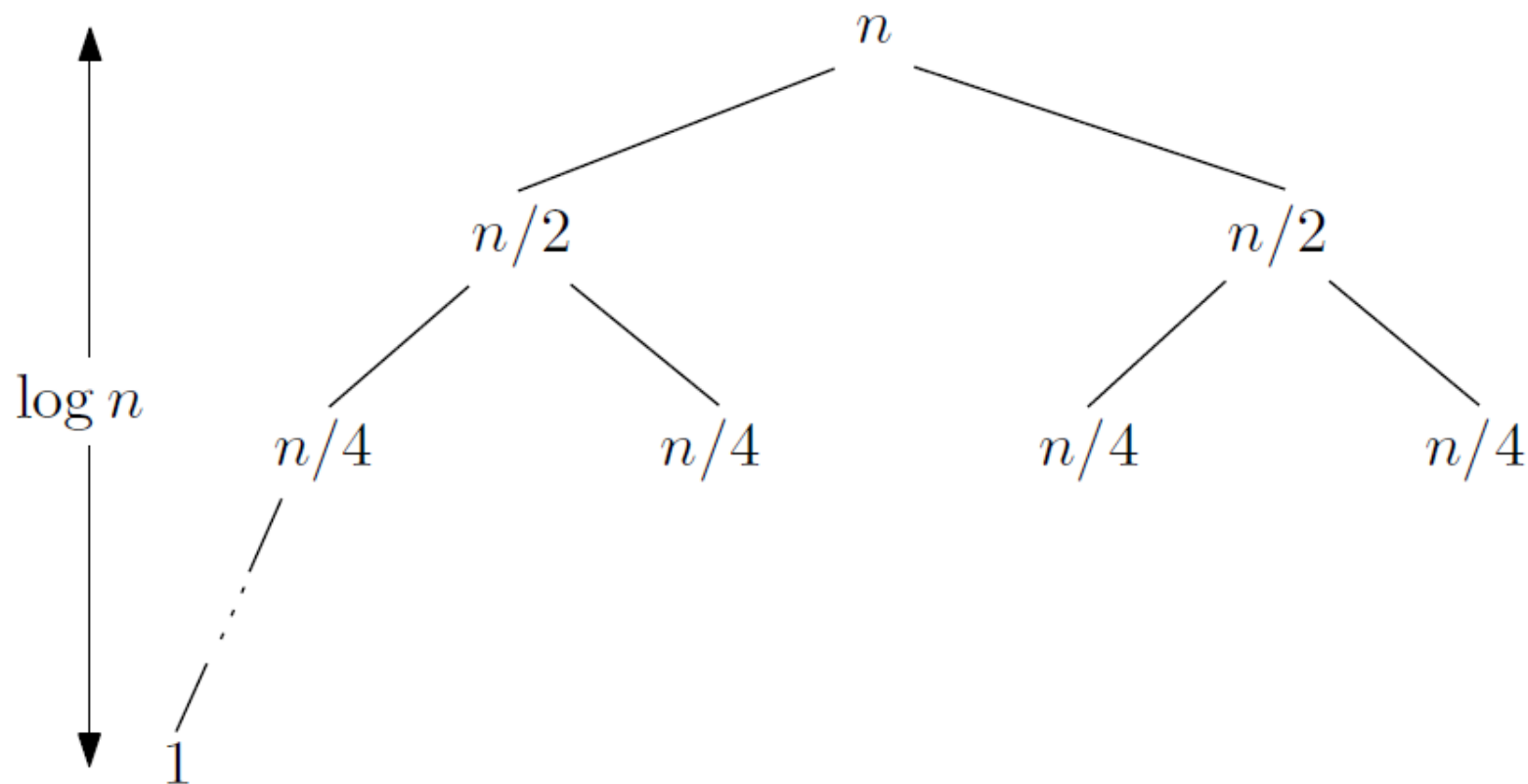
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Three Kinds of Analysis

Solve

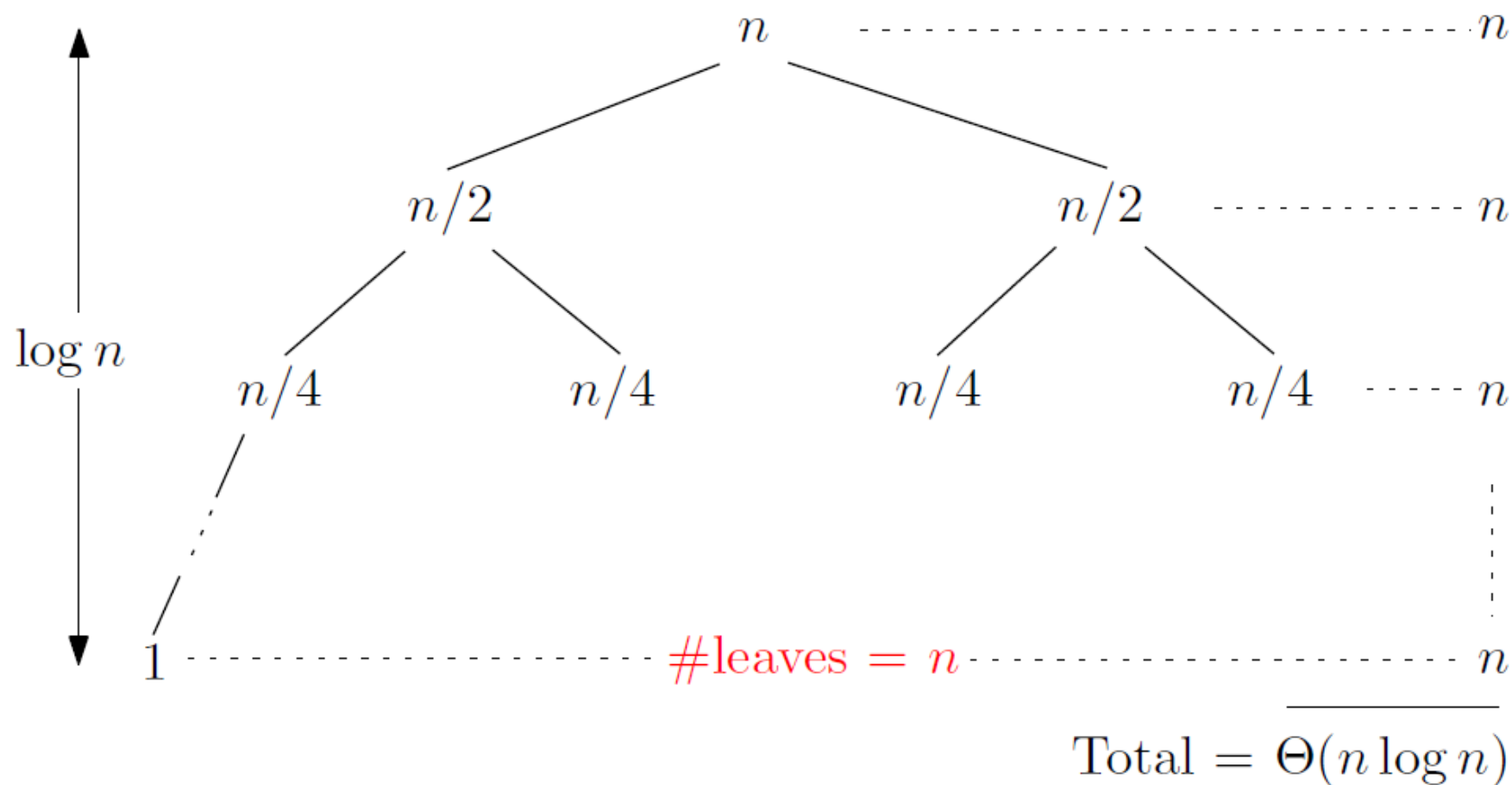
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Three Kinds of Analysis

Solve

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



dank u
ju faleminderit
Tack
Asante 谢谢 Tak mulțumesc
kiitos
Salamat! Gracias
Terima kasih Aliquam
Merci
Dankie Obrigado
ありがとう köszönöm grazie
Aliquam Go raibh maith agat
děkuii Thank you