Design and Analysis of Algorithms

Jun Han

Professor of Computing Science

BUAA

Introduction

Outline

- Lecturer
- 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

Lecturer: Jun Han

Lecturer:
韩军

Location:
北航新主楼 G 座 1112.

Phone: 8231 6340

Email:
jun_han@buaa.edu.cn

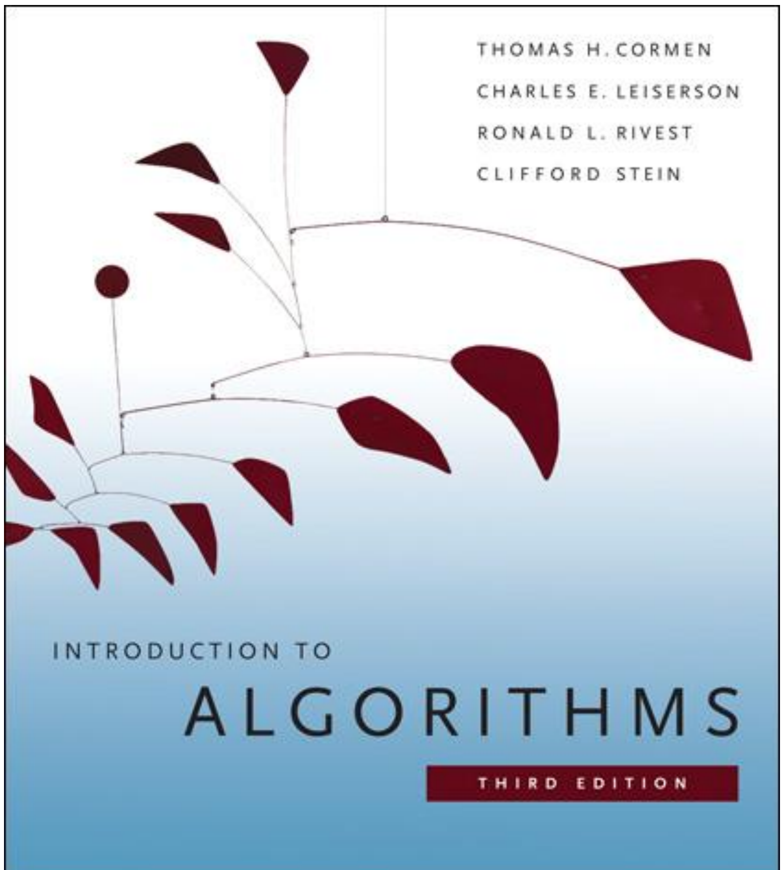
Profile:
http://www.act.buaa.edu.cn/

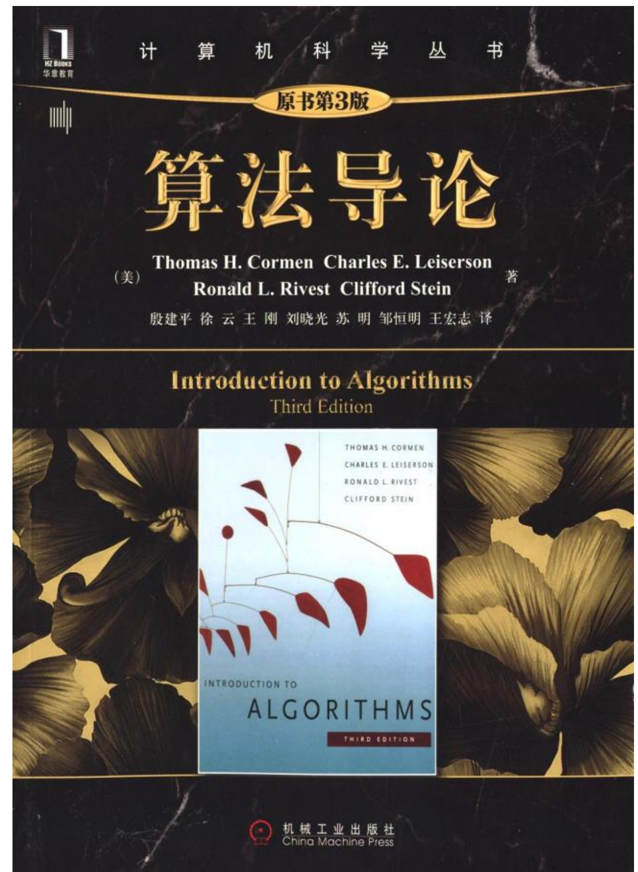
Outline

- Lecturer
- 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

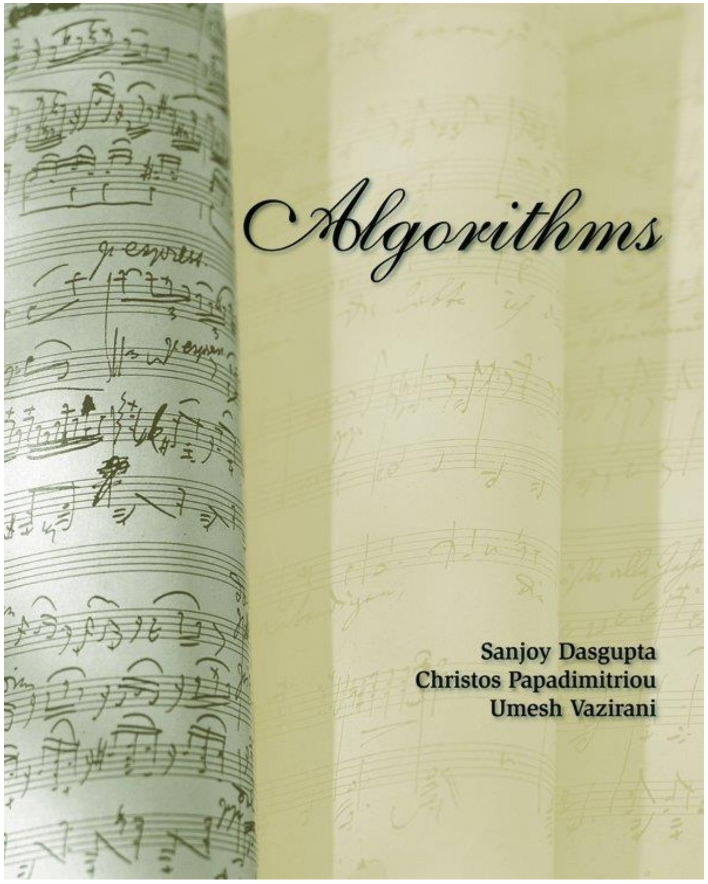
- Textbook: *Introduction to Algorithms* (3rd ed.)
  - by Cormen, Leiserson, Rivest and Stein (CLRS)
  - Prepublication version available online

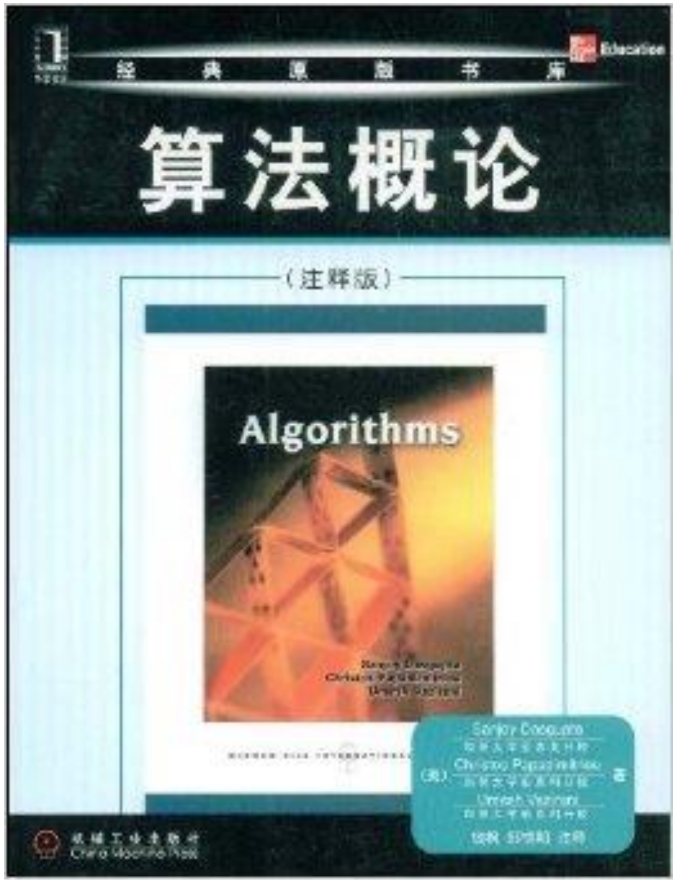




References (1)

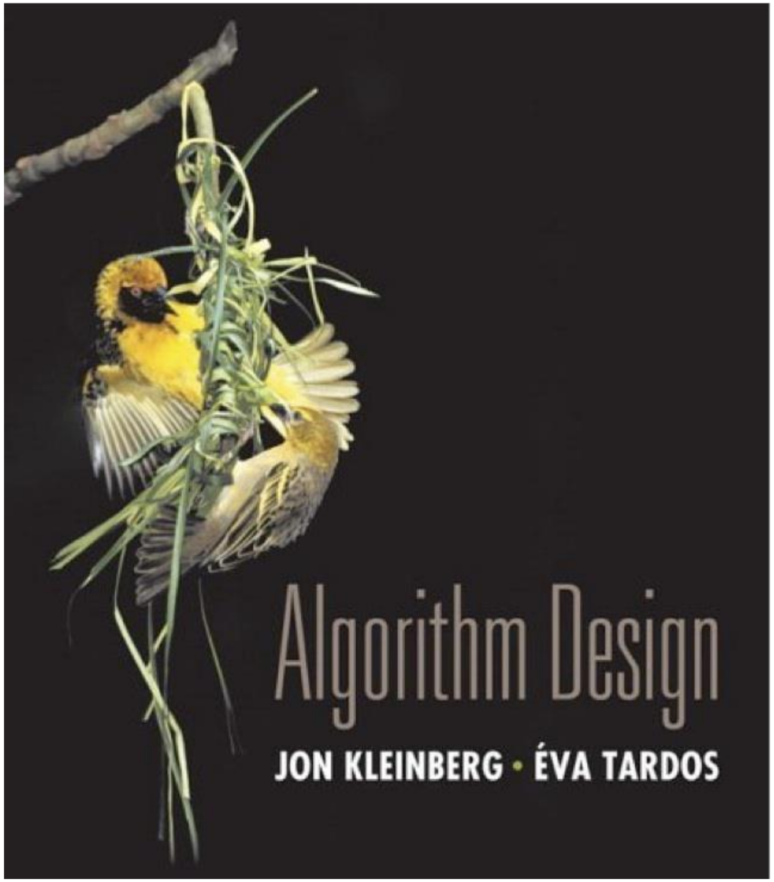
- Reference: *Algorithms*
  - by Dasgupta, Papadimitriou, and Vazirani (DPV)
  - Prepublication version available online

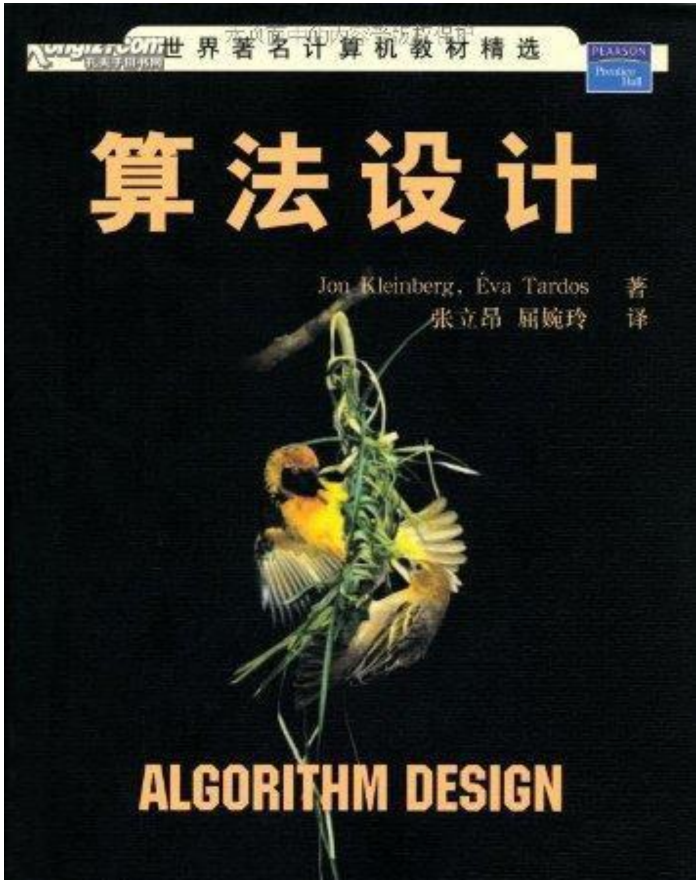




References (2)

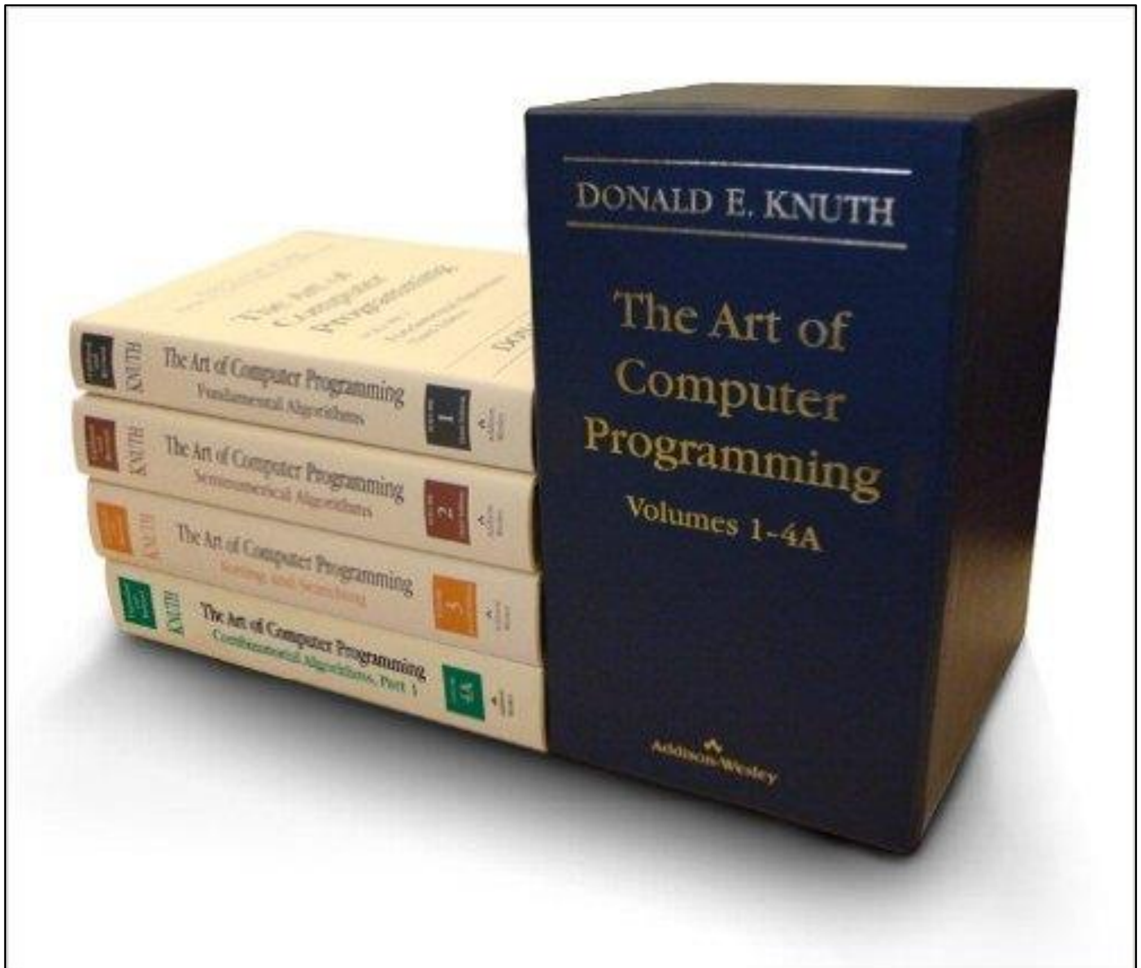
- Reference: *Algorithm Design*
  - by Kleinberg and Tardos (KT)





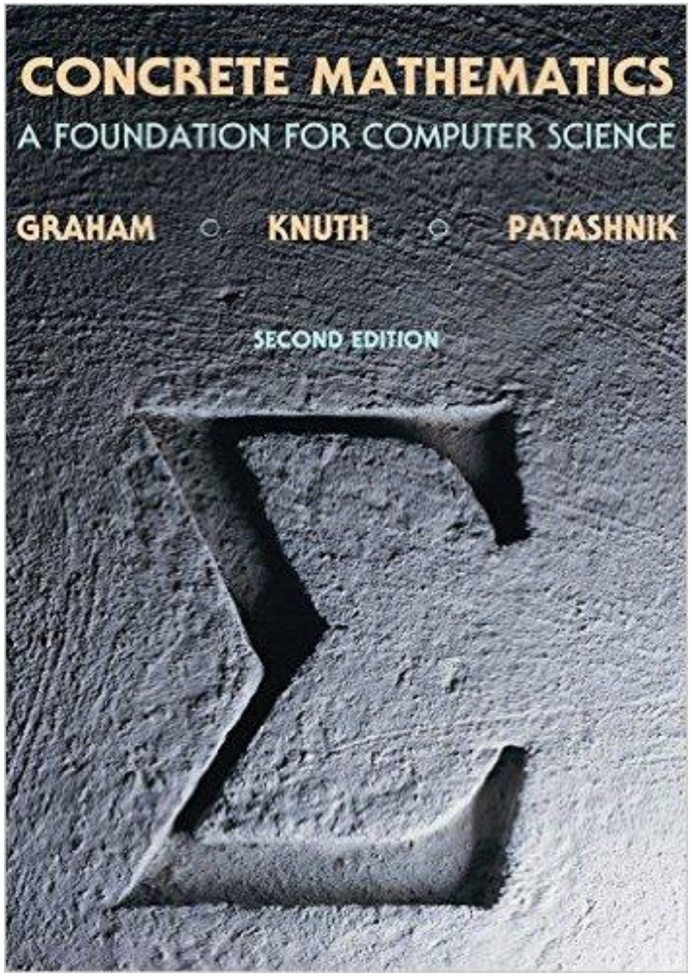
References (3)

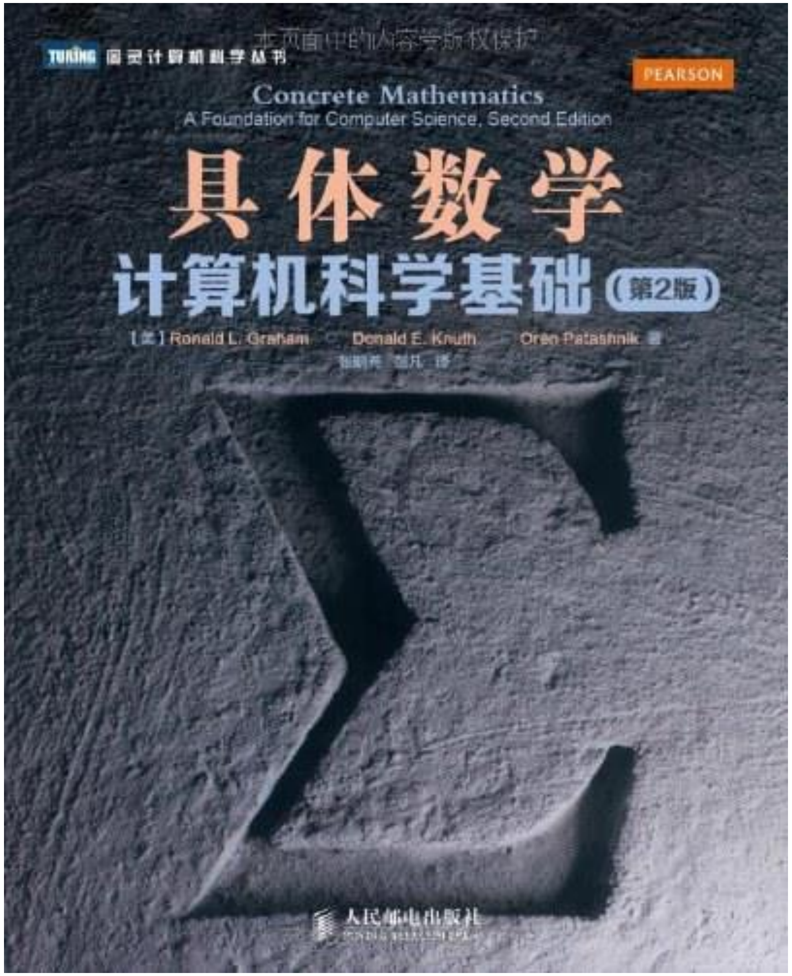
- Reference: *The Art of Computer Programming*
  - by Donald E. Knuth



References (4)

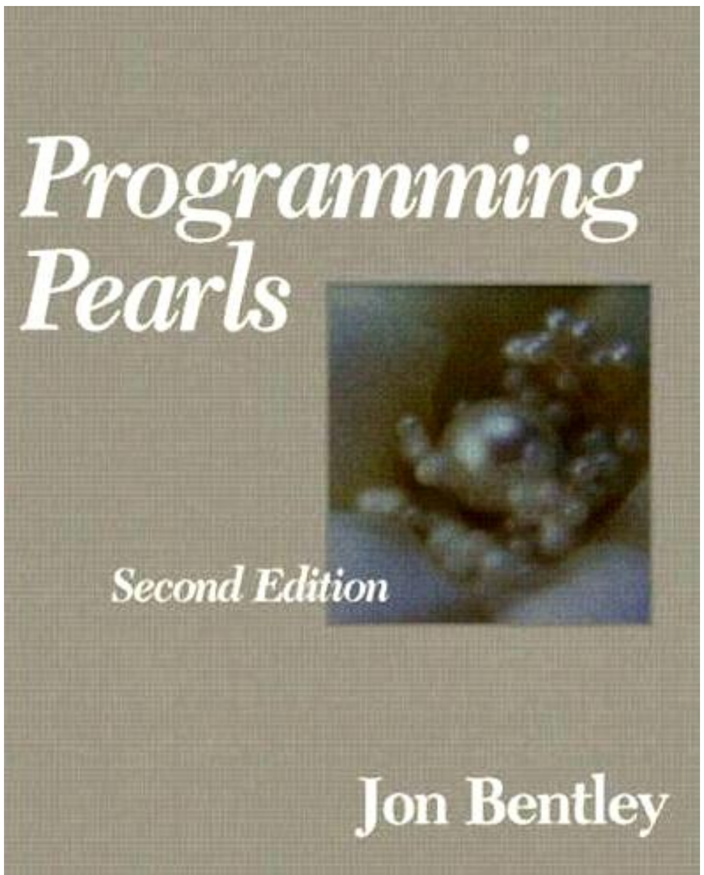
- Reference: *Concrete Mathematics* (2nd ed.)
  - by Graham, Knuth, Patashnik (GKP)

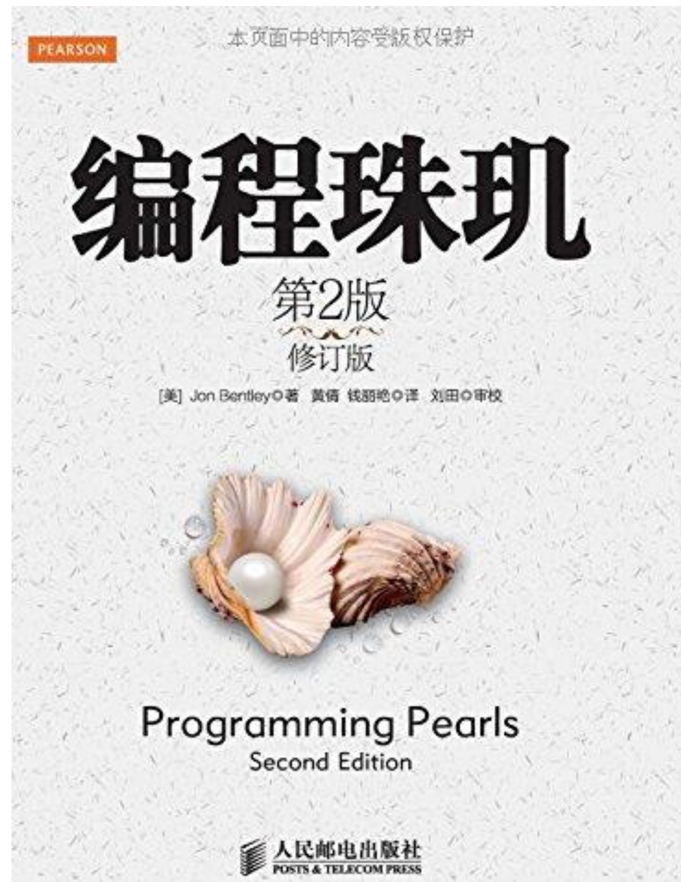




References (5)

- 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)

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 MED
- 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

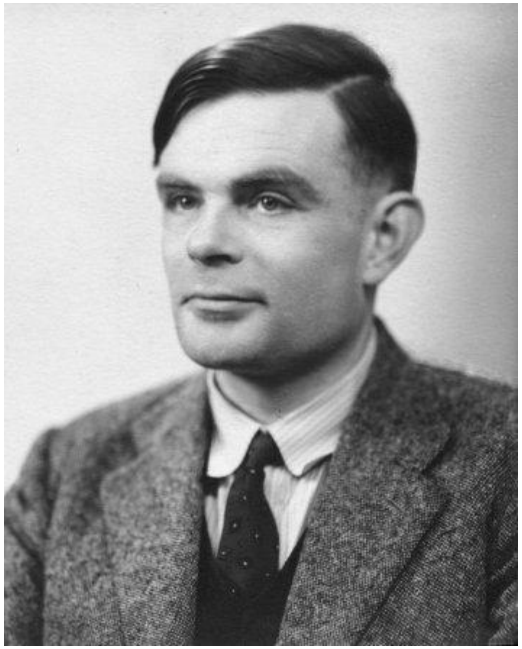
- (30%) Four Assignments
  - Each requires designing algorithms and analyzing correctness/run time.
  - Each will take 14 days.
  - After each submission due, we will post the solution and **WON'T** accept any assignment.
  - Failing to do any of these will be considered **PLAGIARISM**, and will result in a failing grade if we detect it.
- (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 middle of Oct.
- (60%) Final Exam
  - It covers entire semester's material.

Outline

- Lecturer
- 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

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.



Alan M. Turing

Nobel Prize of Computing

2020, Alfred Vaino Aho & Jeffrey David Ullman
《The Design and Analysis of Computer Algorithms》

A.M. Turing Award Winners for Algorithms

Donald E. Knuth
1974, USA

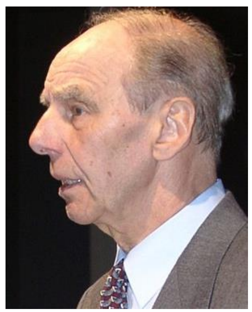
Robert W. Floyd
1978, USA

Stephen A. Cook
1982, USA

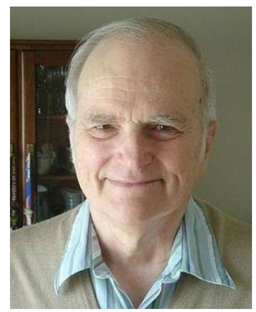
Richard M. Karp
1985, USA

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

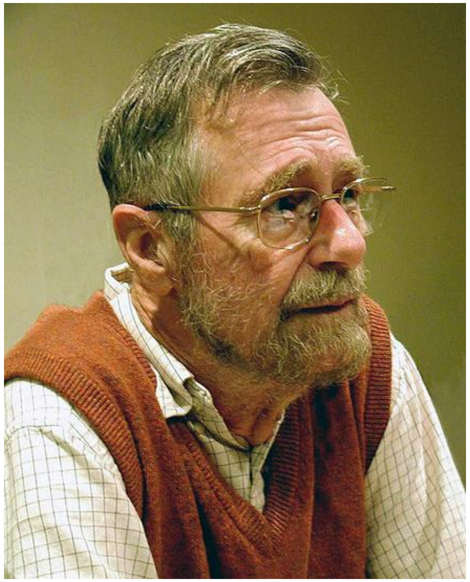
Silvio Micali
2012, Italy

Shafi Goldwasser
2012, USA

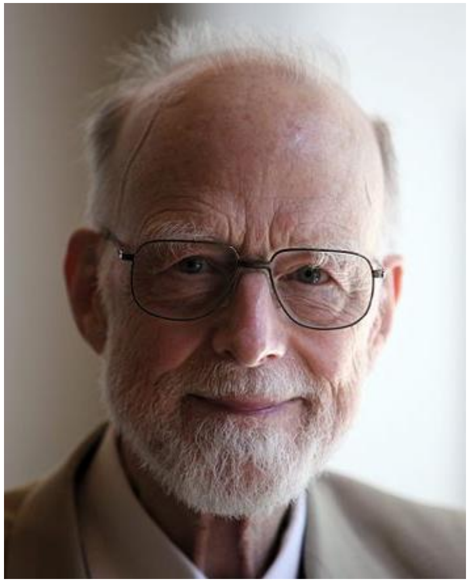
Martin Hellman
2015, USA

Whitfield Diffie
2015, USA

Other Related A.M. Turing Award Winners

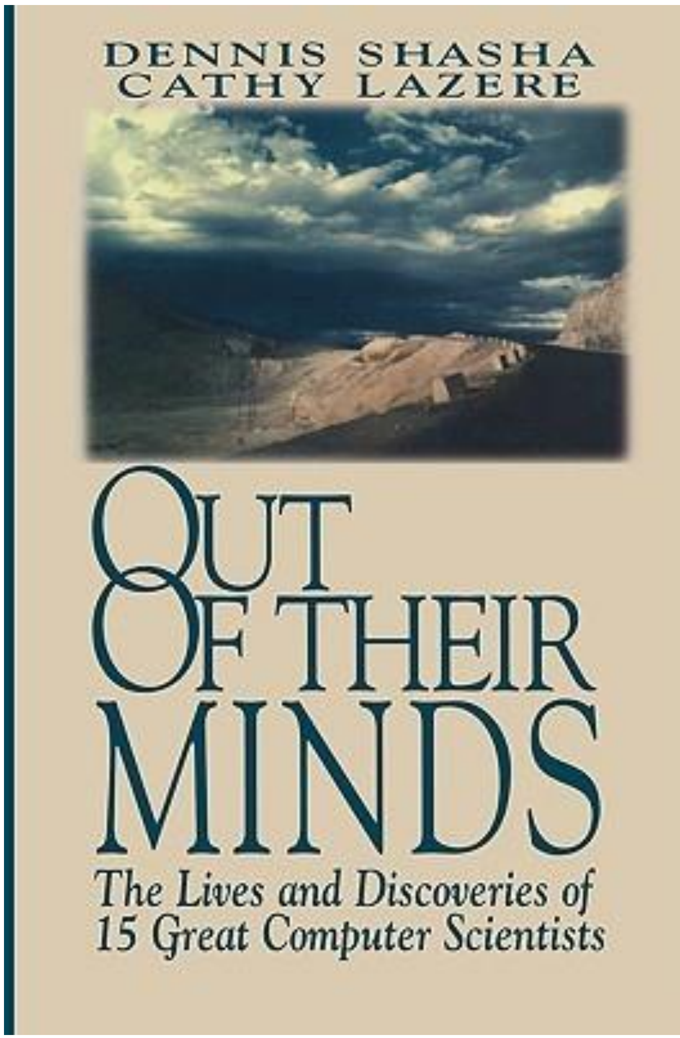


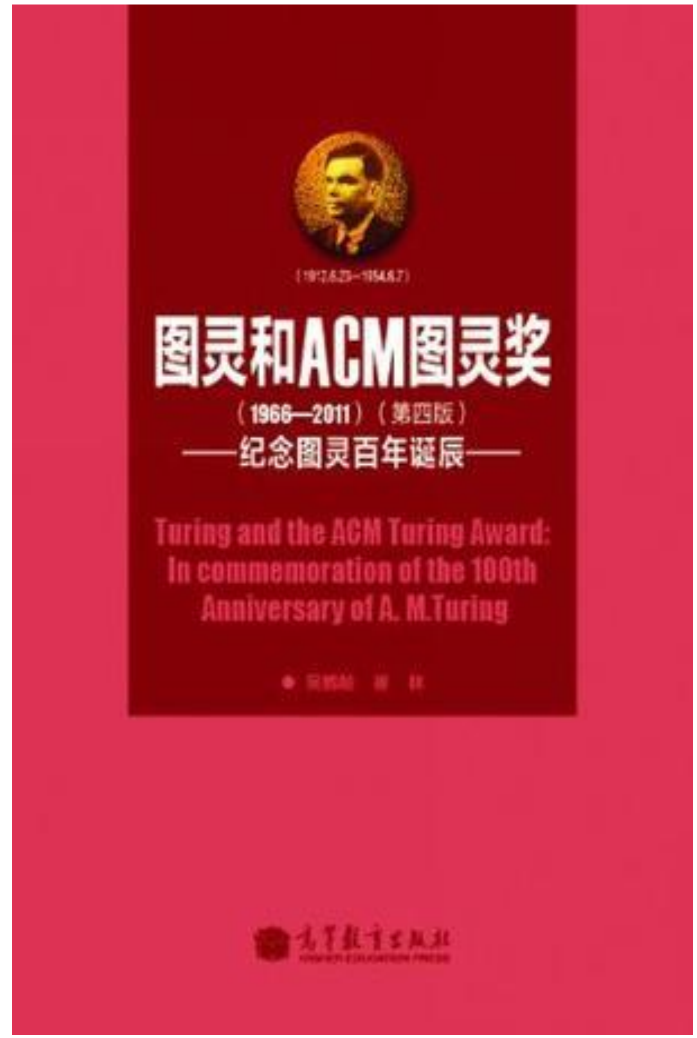
Edsger W. Dijkstra
The Recipient in 1972,
Netherlands,
Contributions: ALGOL Father,
Related Work: Dijkstra Algorithm



Tony Hoare
The Recipient in 1980,
UK,
Contributions: Hoare logic,
Related Work: QuickSort

Books of A.M. Turing Award Winners





Outline

- Lecturer
- 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

- Lecturer
- 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 $\langle a_1, \cdots, a_n \rangle$
- Output: Permutation (reordering)

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

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

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**
  key $\leftarrow A[j]$;
  $i \leftarrow j - 1$;
  while $i \geq 1$ *and* $A[i] > key$ **do**
    $A[i+1] \leftarrow A[i]$;
    $i \leftarrow i - 1$;
  end
  $A[i+1] \leftarrow$ key;
end

	key	

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

- Lecturer
- 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)

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

Three Kinds of Analysis: I

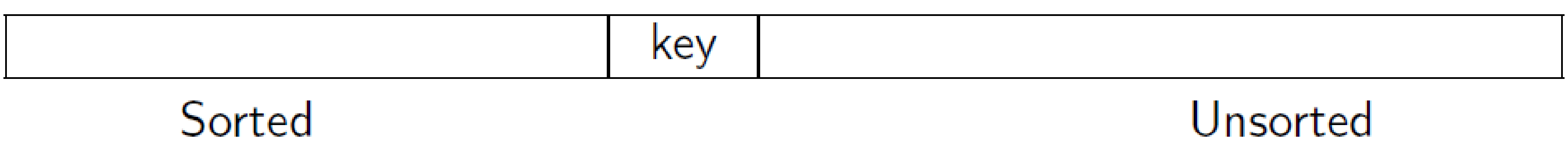
Best Case: An instance for a given size n that results in the fastest possible running time.

Example (Insertion sort)

$$A[1] \le A[2] \le A[3] \le \cdots \le A[n]$$

The number of comparisons needed is equal to

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



"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)

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

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)

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

The number of comparisons needed is equal to

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

	key	

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)

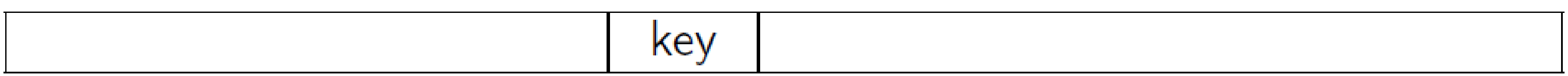
$\Theta(n^2)$, assuming that each of the $n!$ instances is equally likely (uniform distribution).

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).



Sorted Unsorted

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

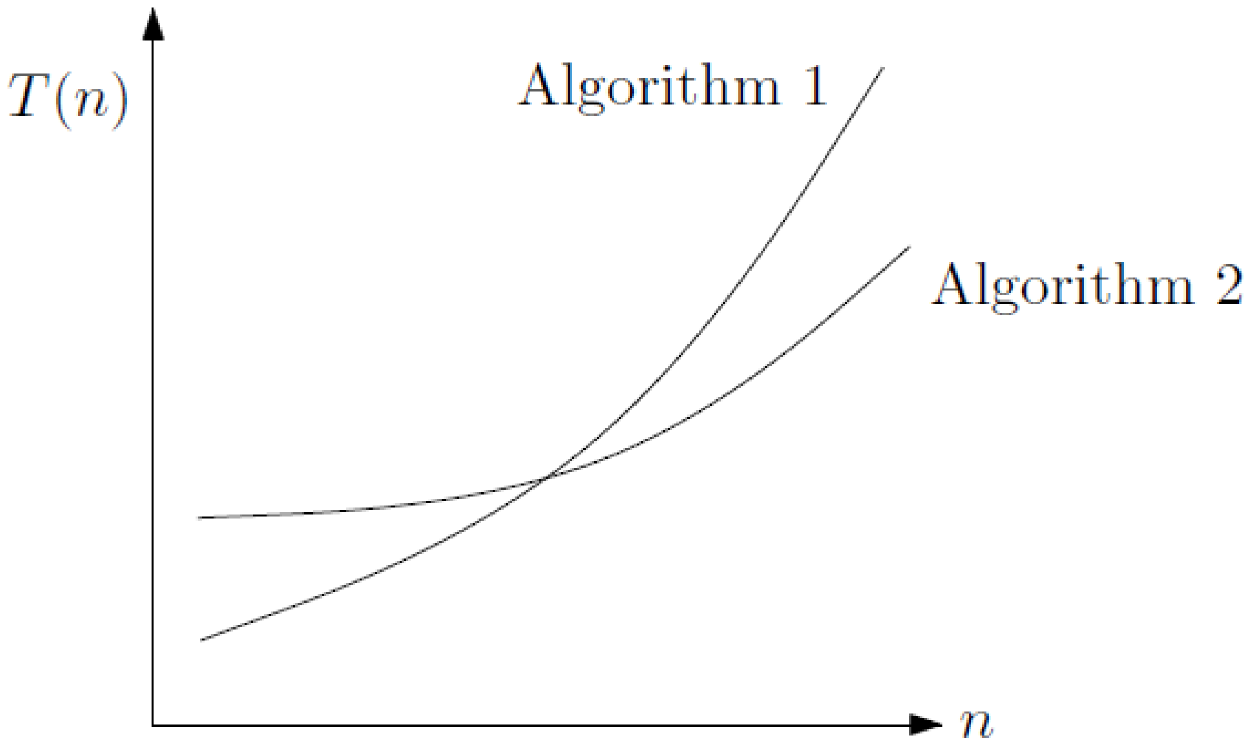
Three Kinds of Analysis

- 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

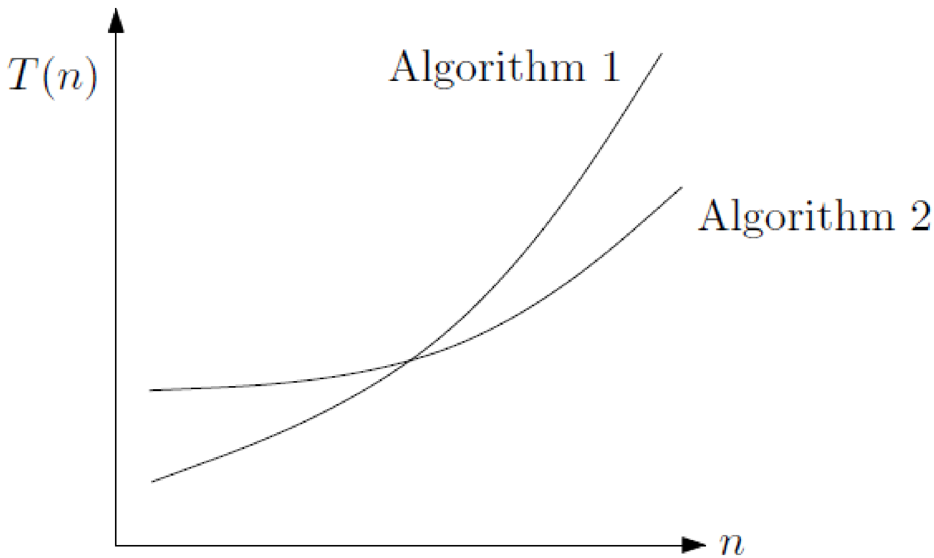
- Lecturer
- Course Details
- A.M. Turing Award Winners for Algorithms
- What Is This Course About
- What Are Algorithms
- What Does It Mean to Analyze An Algorithm
- Comparing Time Complexity

Comparing Time Complexity



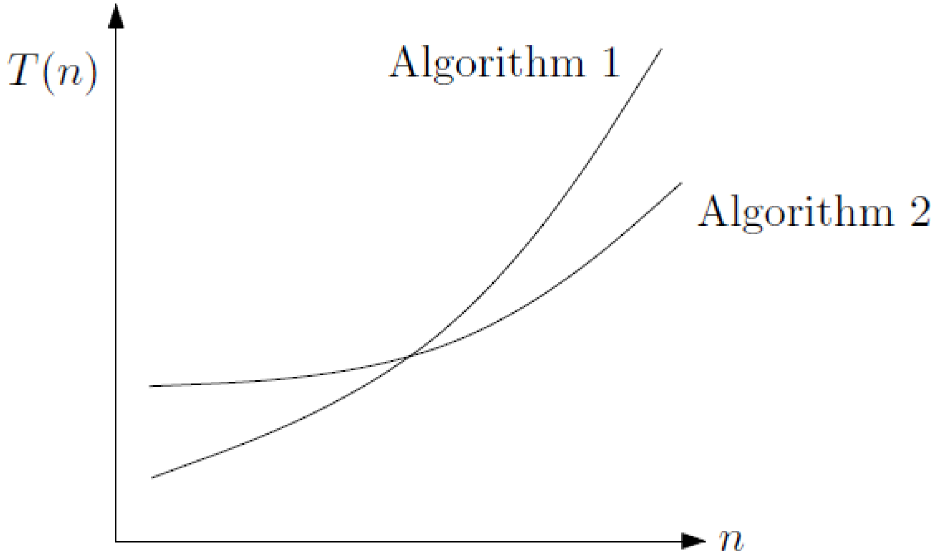
- Which algorithm is superior for large n?
  - T(n) for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17$
  - T(n) for Algorithm 2 is $7n^2 - 8n + 20$
- Clearly, Algorithm 2 is superior.

Asymptotic Analysis



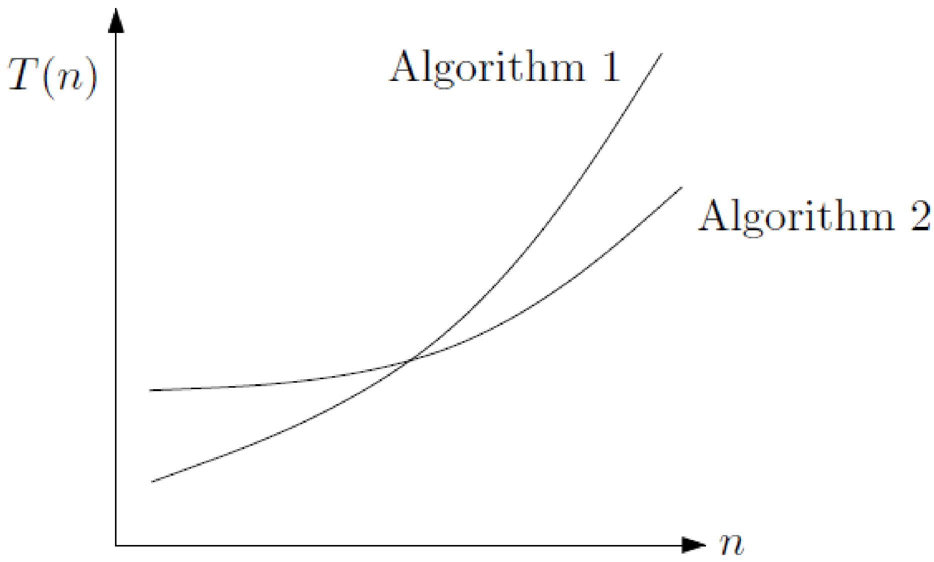
- T(n) for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$

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

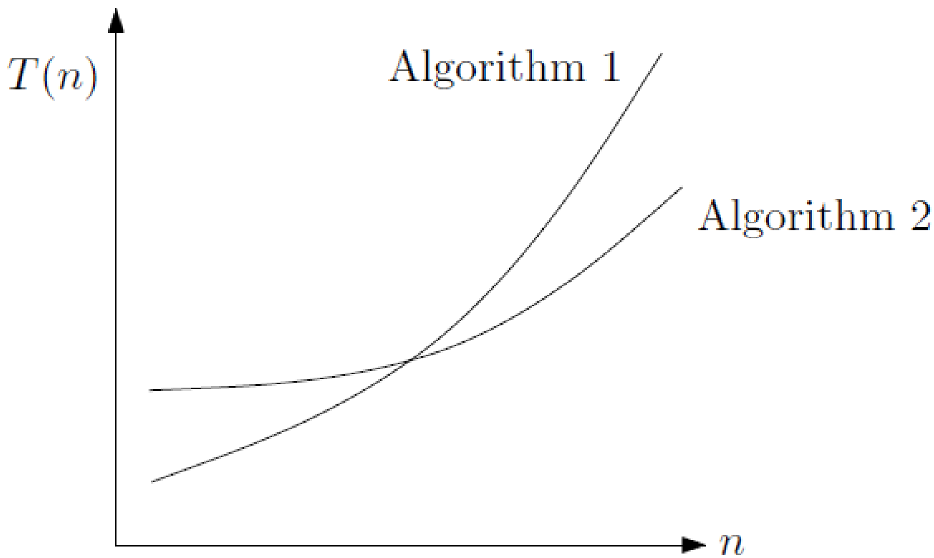


- T(n) for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- T(n) for Algorithm 2 is $7n^2 - 8n + 20 = \Theta(n^2)$

Θ-notation

- Drop low-order terms; ingore leading constants

Asymptotic Analysis

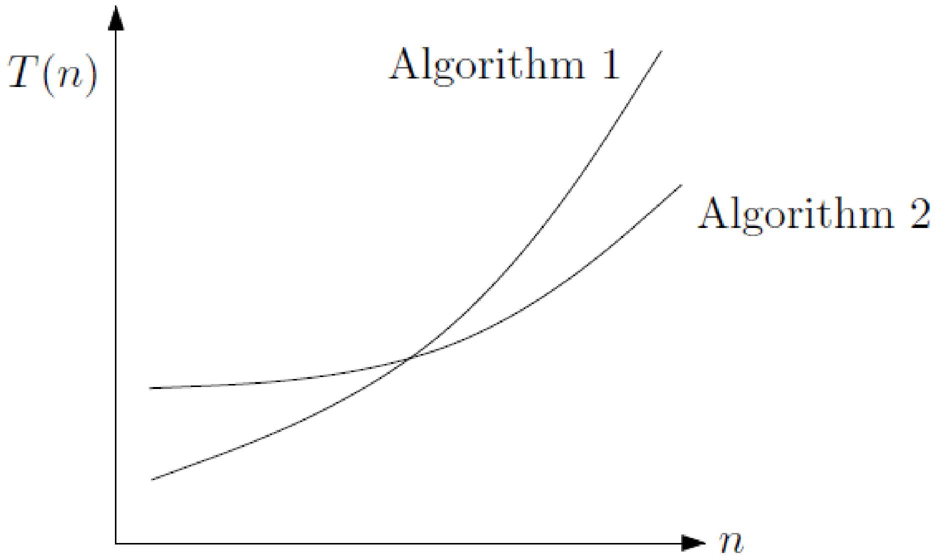


- T(n) for Algorithm 1 is $3n^3 + 6n^2 - 4n + 17 = \Theta(n^3)$
- T(n) for Algorithm 2 is $7n^2 - 8n + 20 = \Theta(n^2)$

Θ-notation

- Drop low-order terms; ingore 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 ← ⌊(left + right)/2⌋;
    Mergesort(A, left, center);
    Mergesort(A, center+1, right);
    "Merge" the two sorted arrays;
end
```

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

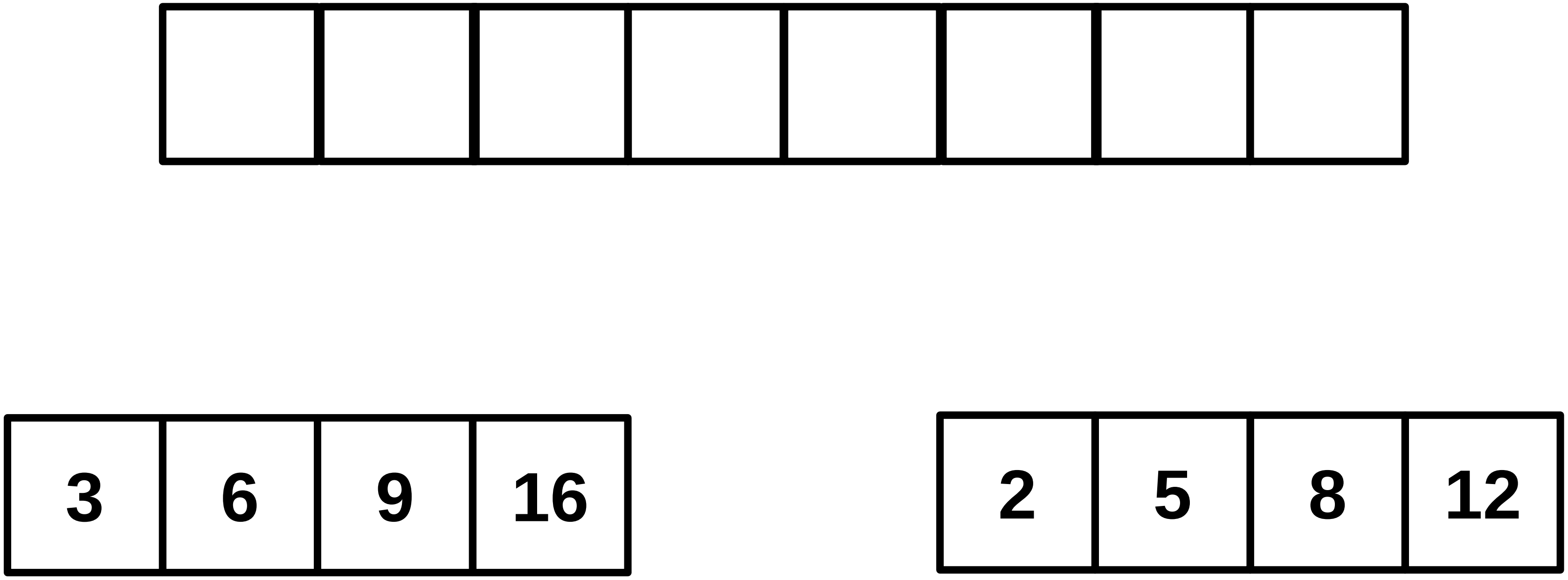
Merge Sort

Mergesort(A, left, right)

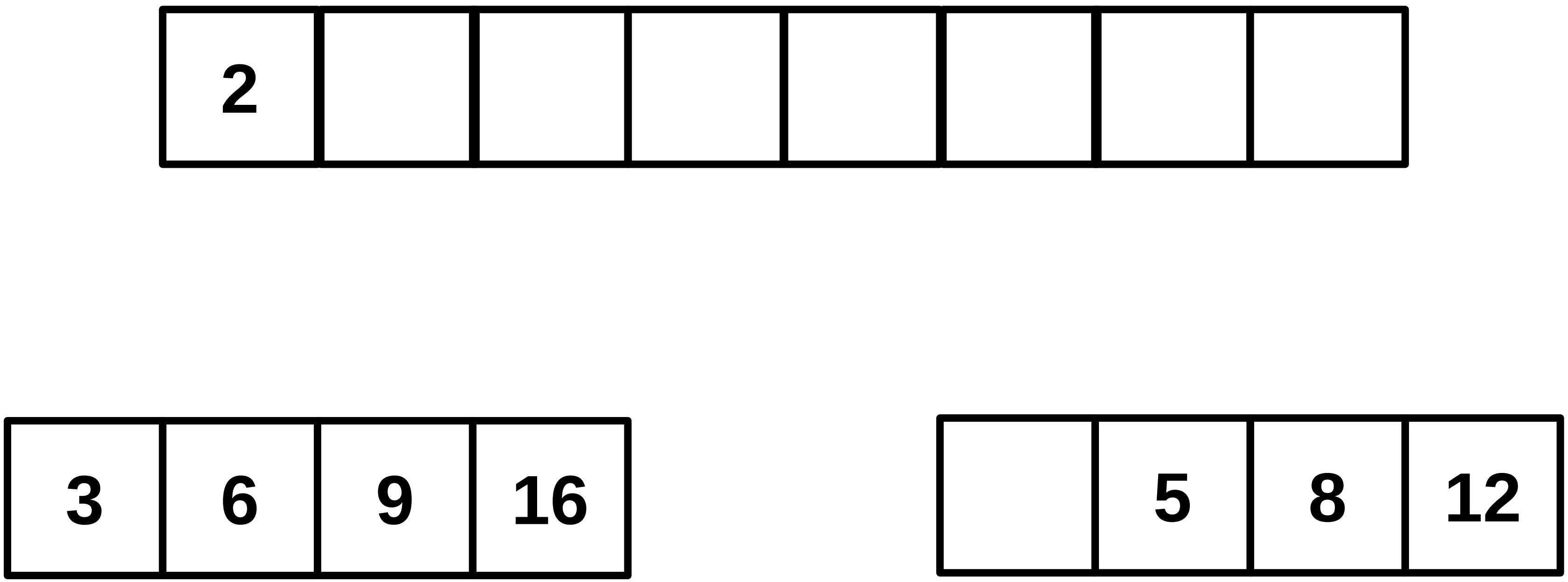
if *left* $<$ *right* **then**
  center $\leftarrow \lfloor$(left + right)/2$\rfloor$;
  Mergesort(A, left, center);
  Mergesort(A, center+1, right);
  "Merge" the two sorted arrays;
end

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

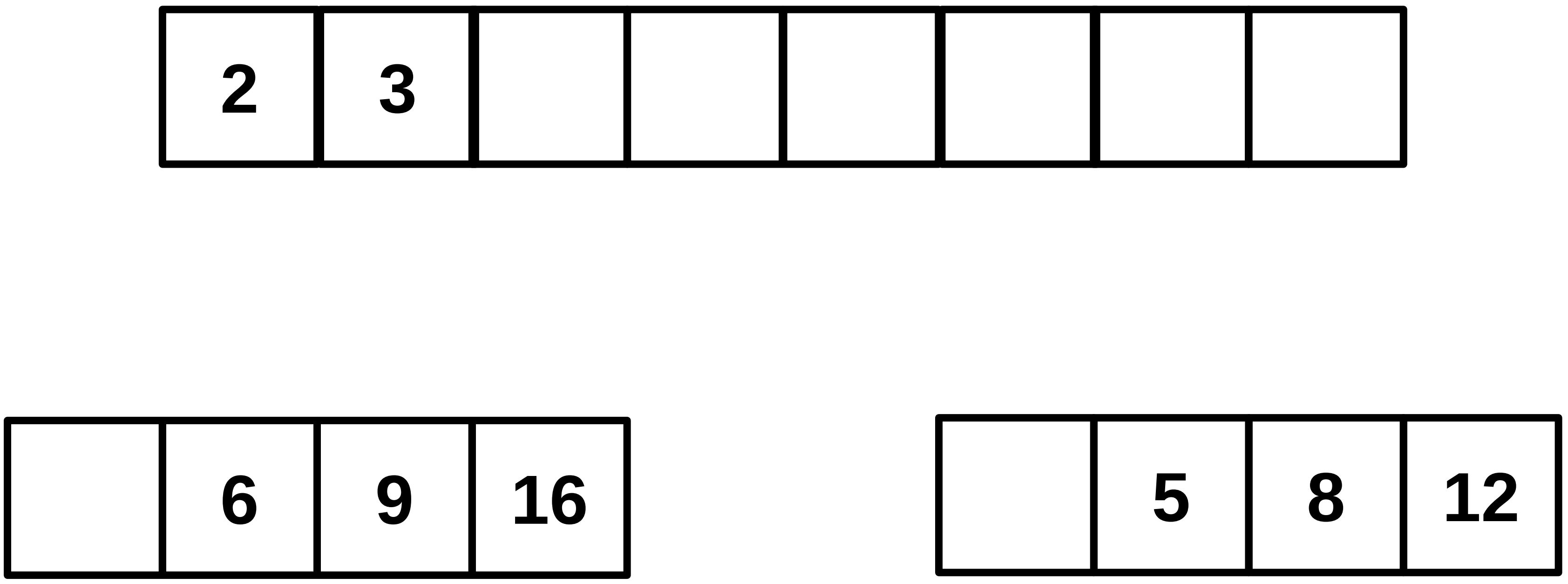
Merge two sorted arrays



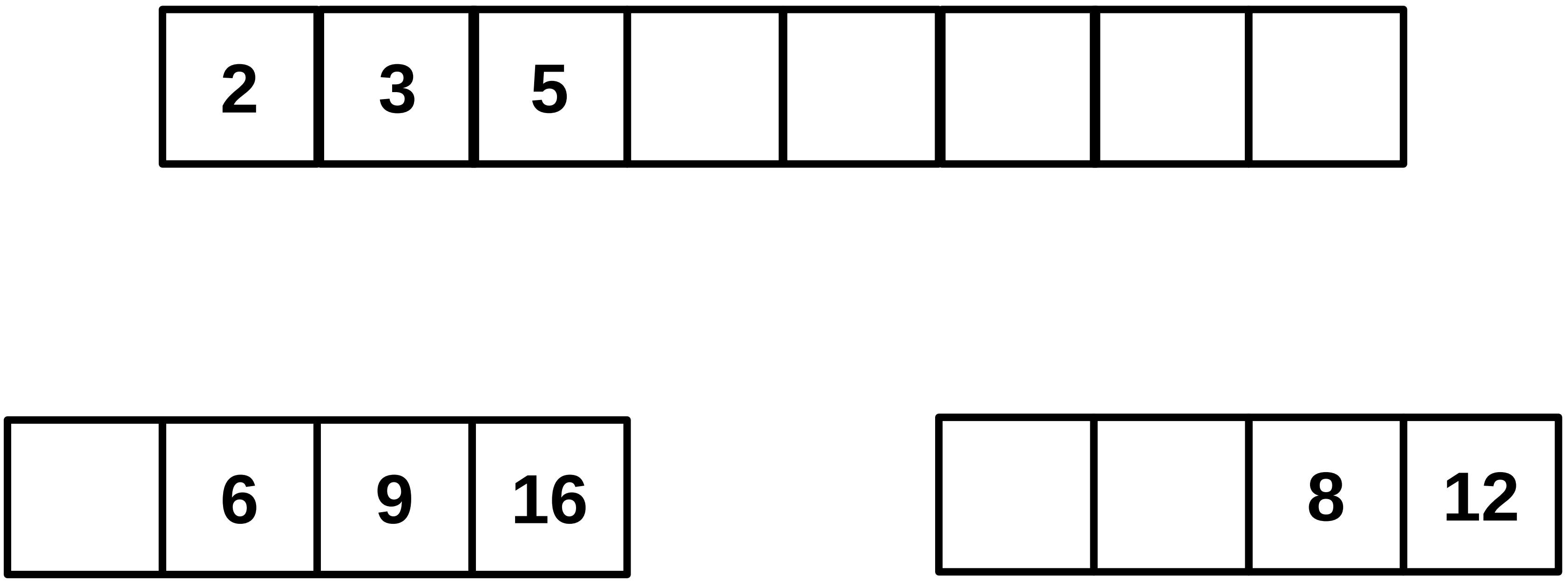
Merge two sorted arrays



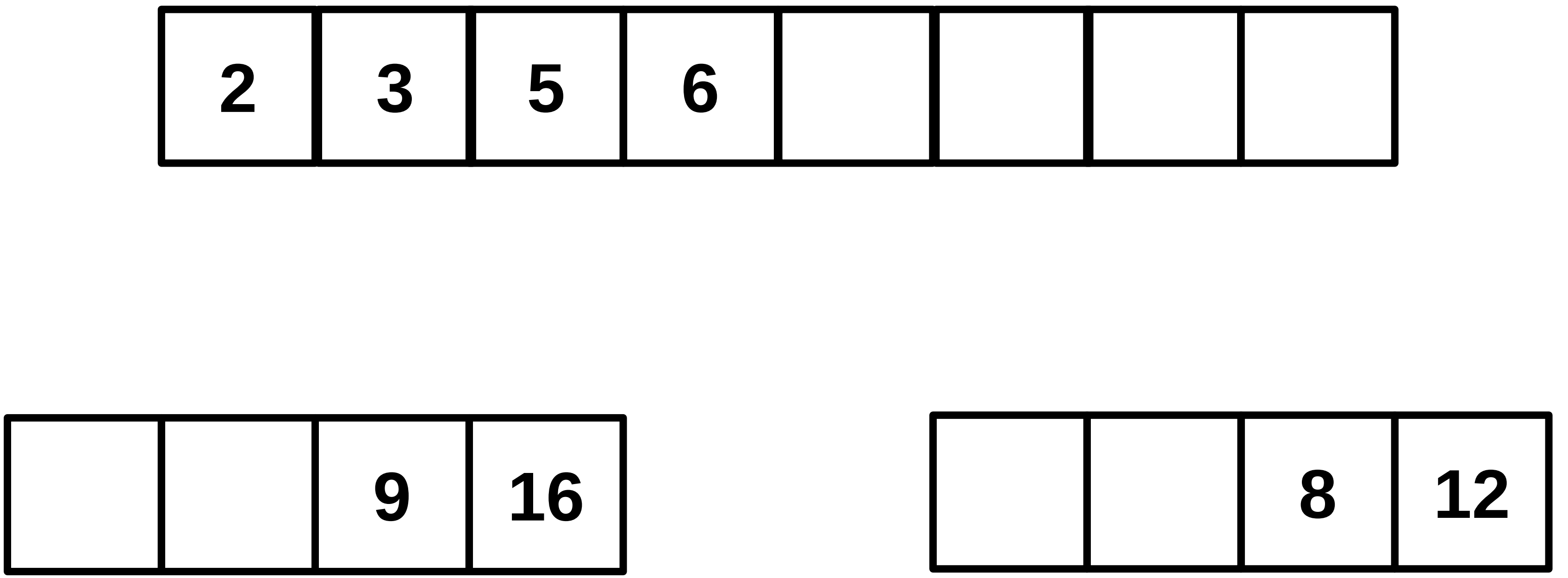
Merge two sorted arrays



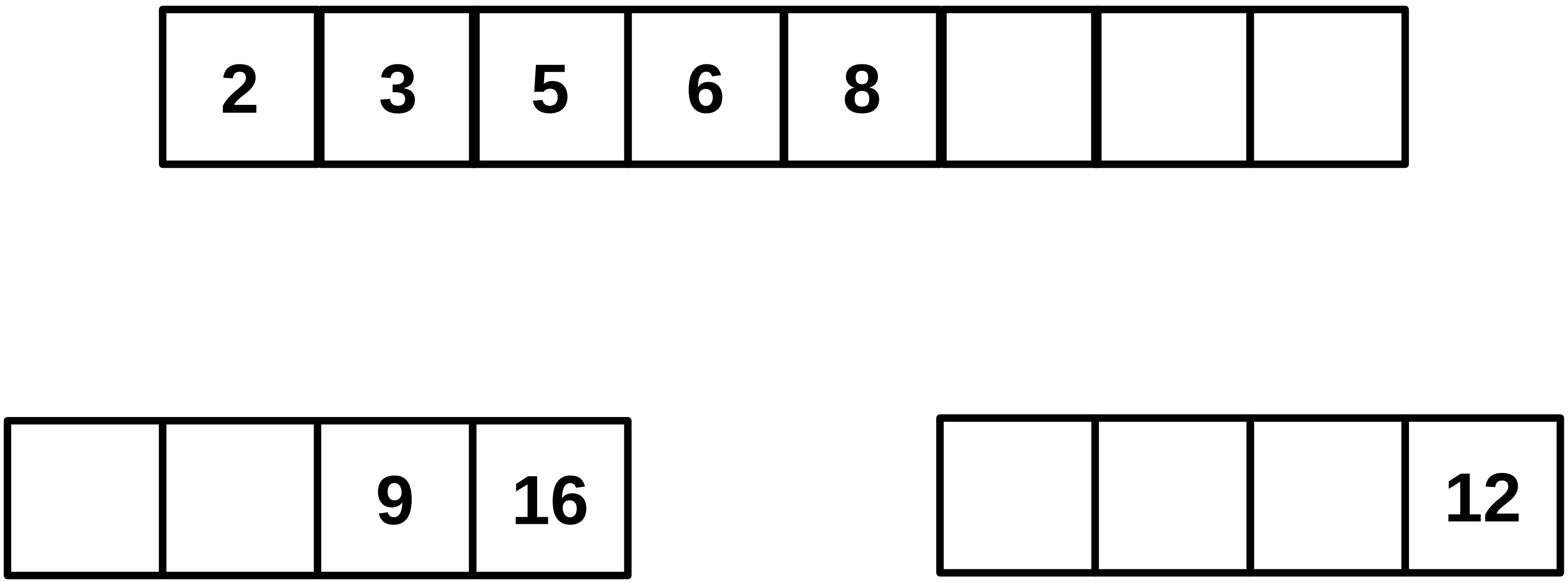
Merge two sorted arrays



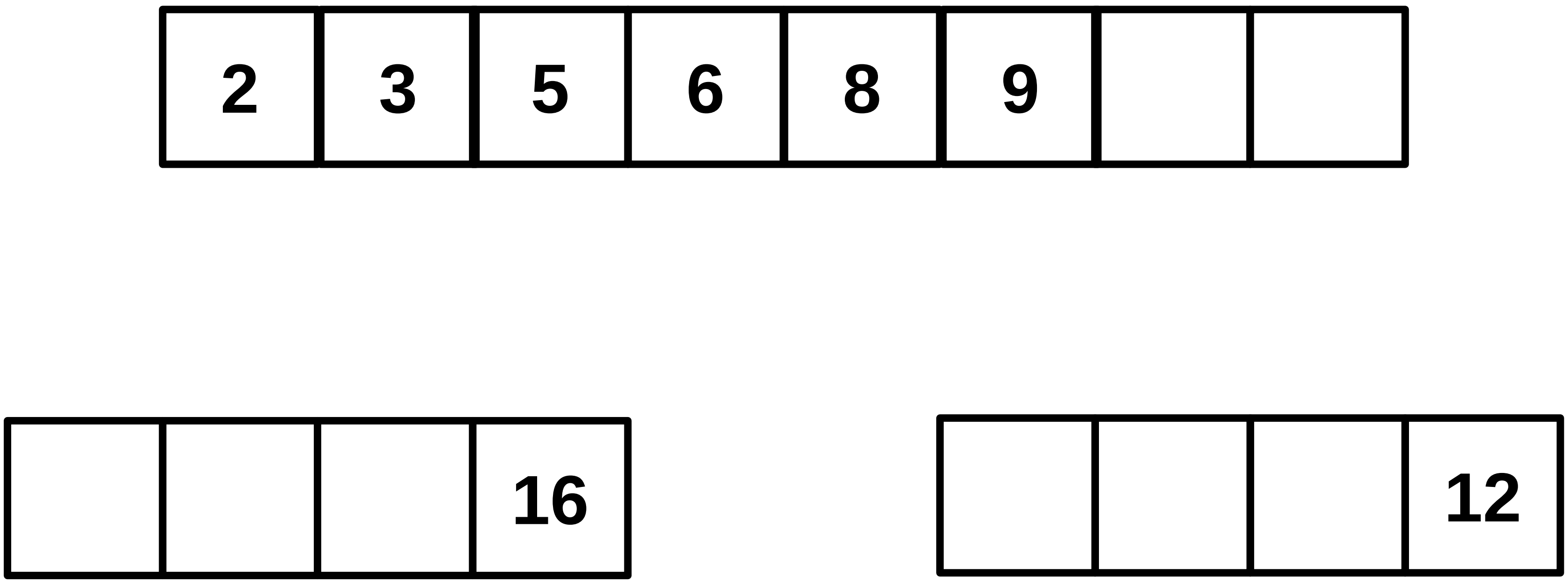
Merge two sorted arrays



Merge two sorted arrays



Merge two sorted arrays

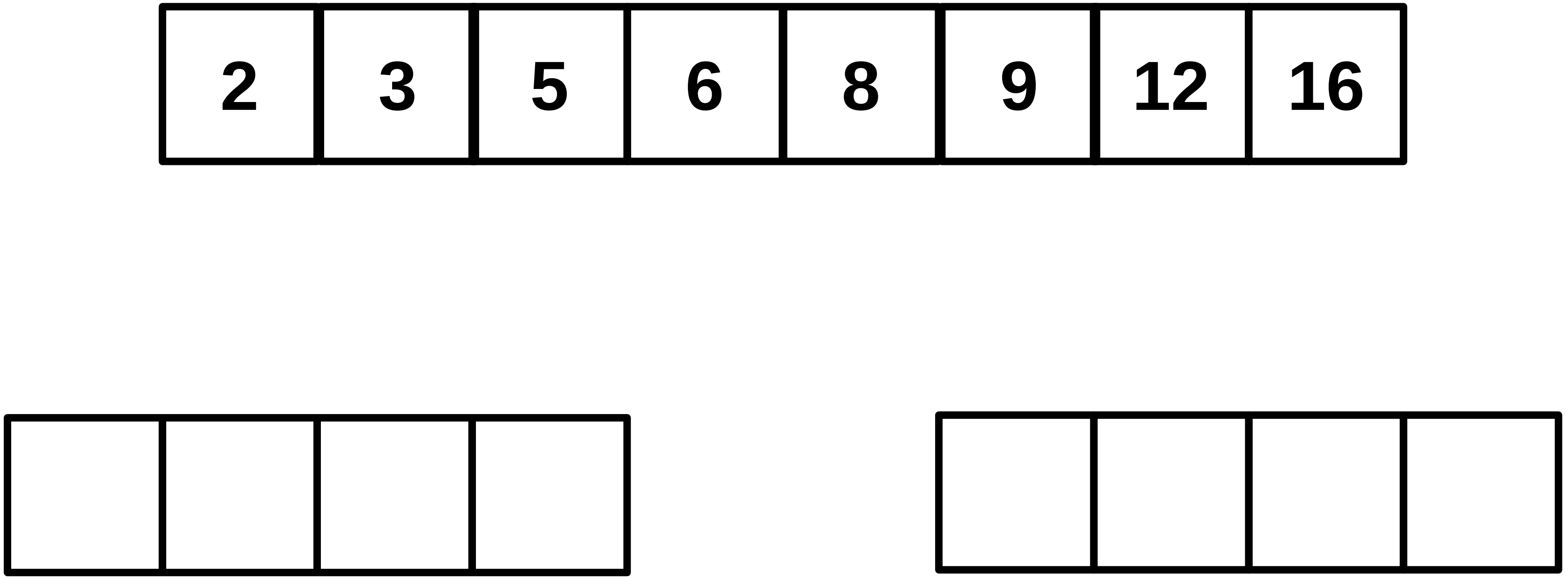


Merge two sorted arrays

2	3	5	6	8	9	12	

			16

Merge two sorted arrays



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 $\mathit{left} < \mathit{right}$ **then**
    center $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$;
    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 $\leftarrow \lfloor$(left $+$ 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 $\leftarrow \lfloor(\text{left} + \text{right})/2\rfloor$;
    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; // Θ(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

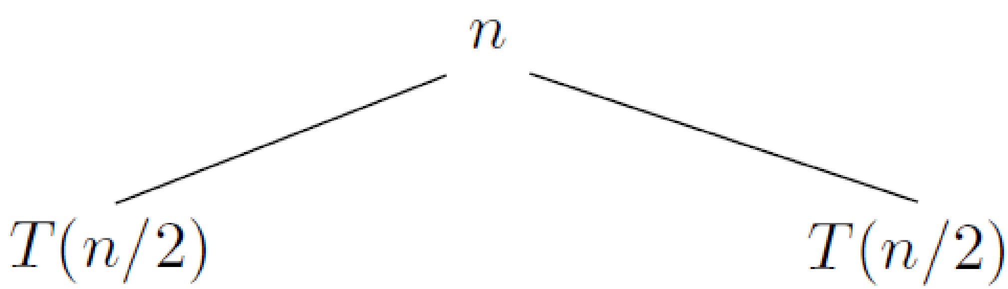
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

$$T(n)$$

Three Kinds of Analysis

Solve

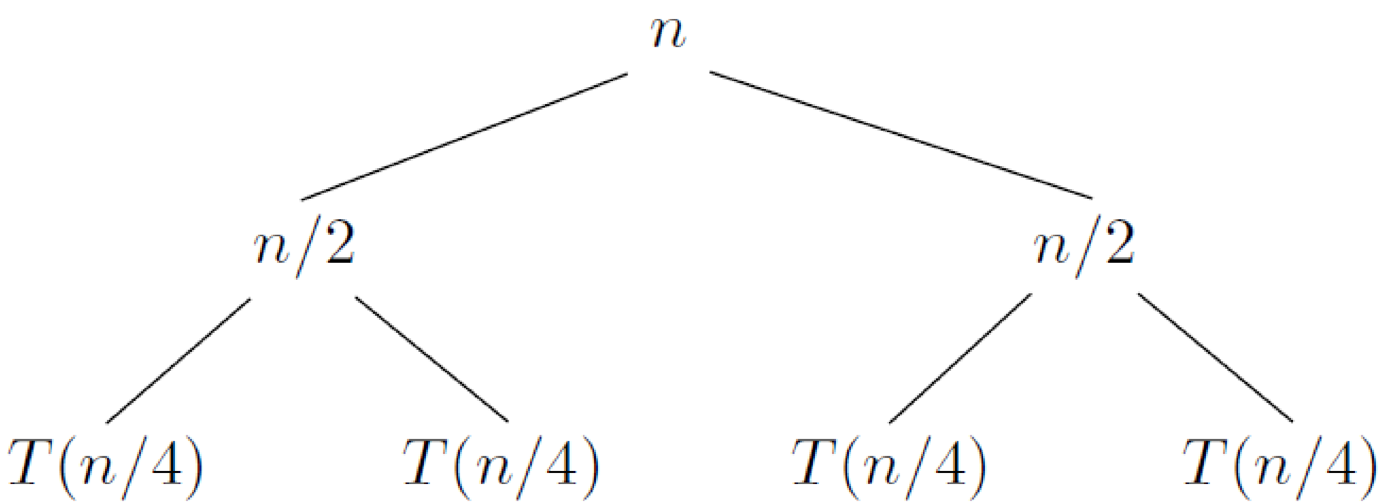
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Three Kinds of Analysis

Solve

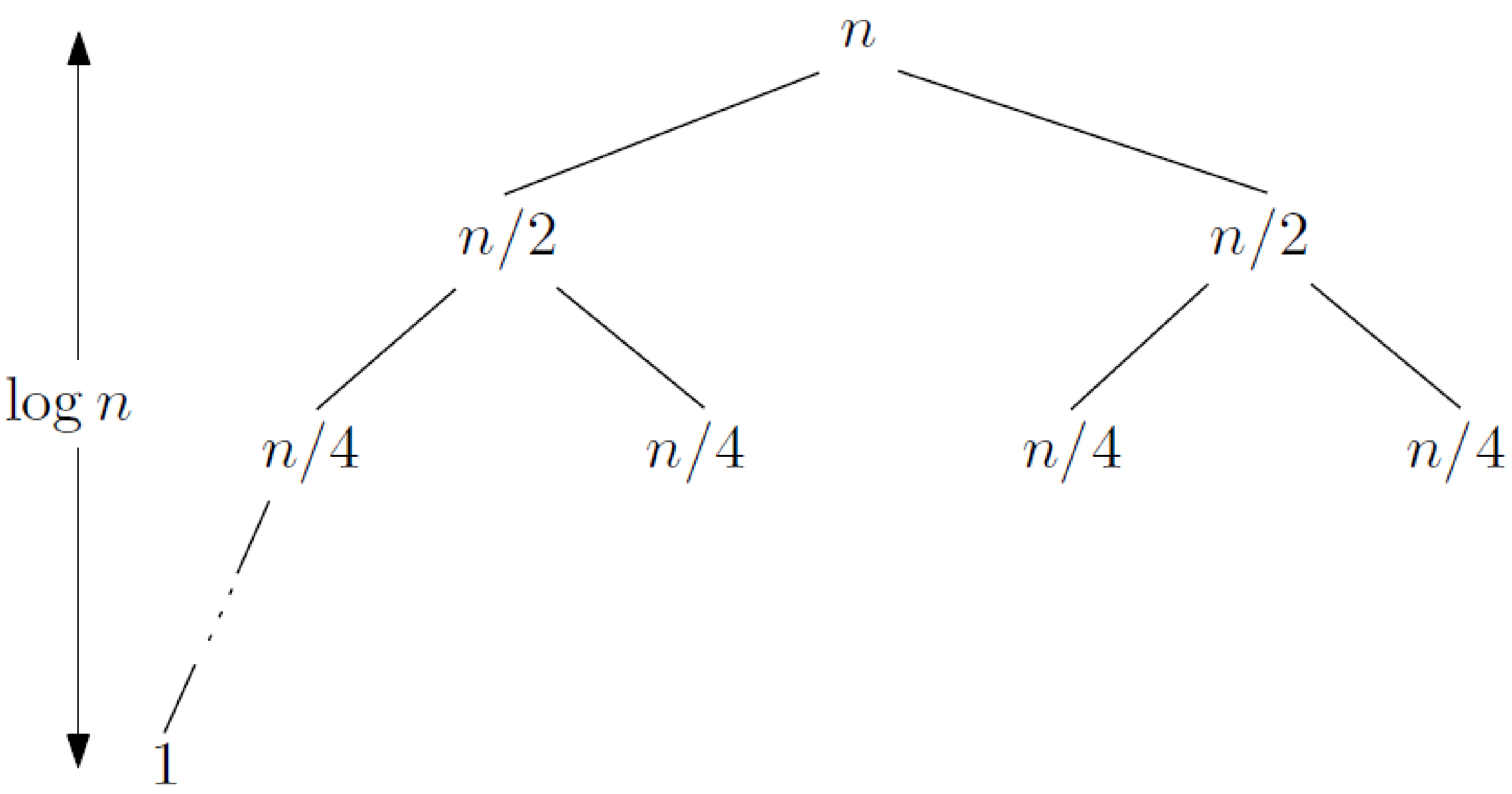
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Three Kinds of Analysis

Solve

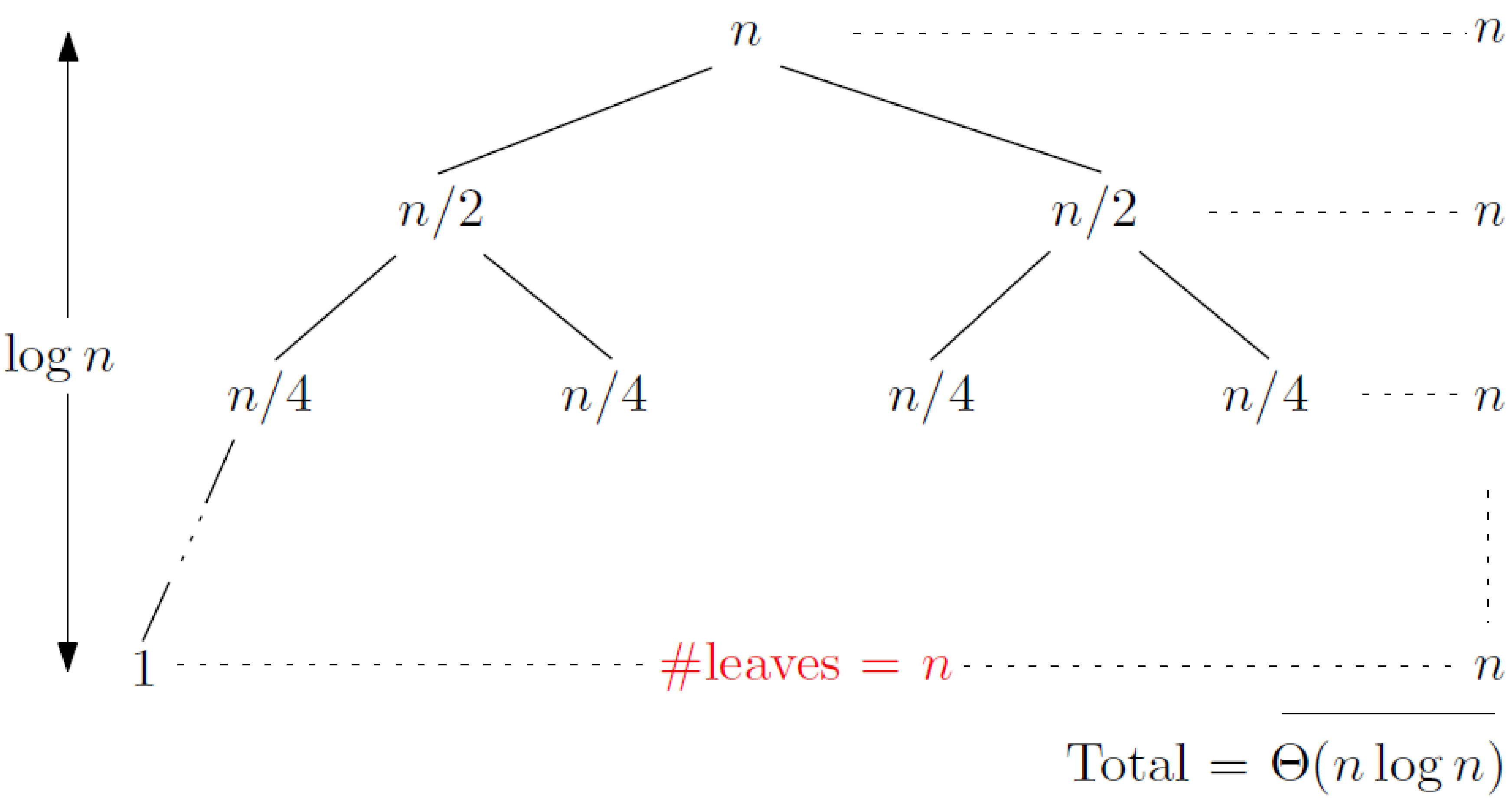
$$T(n) = \begin{cases} 2T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Three Kinds of Analysis

Solve

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



Asymptotic
Notations and Recurrences

Outline

- Asymptotic Notations（渐近记号）
  - Big-Oh
  - Big-Omega
  - Big-Theta
  - Algorithm Design and Algorithm Tuning
- Solving Recurrences
  - Recursion-tree Method（递归树法）
  - Substitution Method（代入法/替代法）
  - Master Method and Master Theorem (主方法)

Outline

- Asymptotic Notations（渐近记号）
  - Big-Oh
  - Big-Omega
  - Big-Theta
  - Algorithm Design and Algorithm Turing
- Solving Recurrences
  - Recursion-tree Method（递归树法）
  - Substitution Method（代入法/替代法）
  - Master Method and Master Theorem (主方法)

Big-Oh

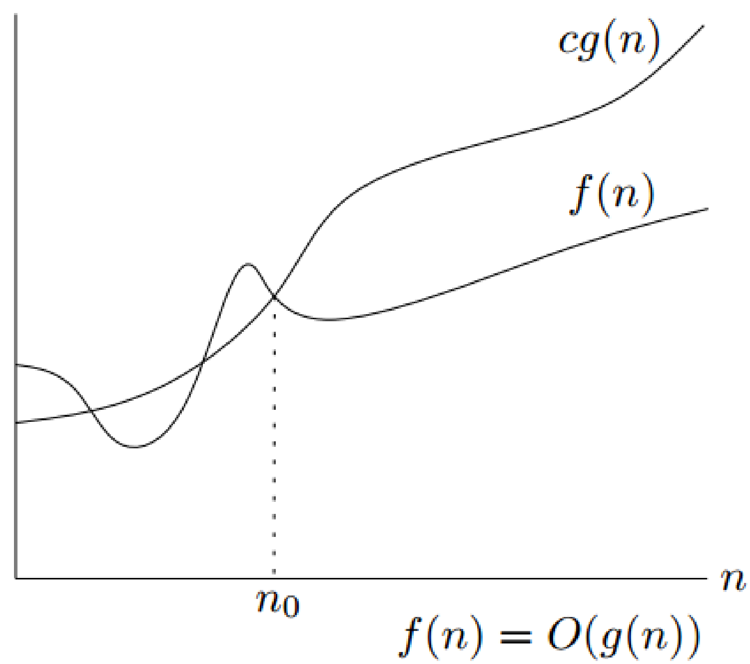
Asymptotic upper bound

Definition (big-Oh)

$f(n) = O(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \leq c \cdot g(n)$ for $n \geq n_0$

When estimating the growth rate of $T(n)$ using big-Oh:

- ignore the low order terms
- ignore the constant coefficient of the most significant term



Big-Oh: Example

Definition (big-Oh)

$f(n) = O(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \leq c \cdot g(n)$ for $n \geq n_0$

Example

Let $T(n) = 3n^2 + 4n + 5$. Prove that $T(n) = O(n^2)$.

Big-Oh: Example

Definition (big-Oh)

$f(n) = O(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \leq c \cdot g(n)$ for $n \geq n_0$

Example

Let $T(n) = 3n^2 + 4n + 5$. Prove that $T(n) = O(n^2)$.

Proof.

$$\begin{aligned} T(n) &= 3n^2 + 4n + 5 \\ &\leq 3n^2 + 4n^2 + 5n^2 \\ &= 12n^2. \end{aligned}$$

Thus, $T(n) \leq 12n^2$ for all $n \geq 1$. Setting $n_0 = 1$ and $c = 12$ in the definition, we have that $T(n) = O(n^2)$. $\square$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O(n^2)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O(n^2)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \leq n \cdot n^2 =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \leq n \cdot n^2 = O\left(n^3\right)$
- $\sum_{i=1}^{n} i$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \le n \cdot n^2 = O\left(n^3\right)$
- $\sum_{i=1}^{n} i \le n \cdot n =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \leq n \cdot n^2 = O\left(n^3\right)$
- $\sum_{i=1}^{n} i \leq n \cdot n = O\left(n^2\right)$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \leq n \cdot n^2 = O\left(n^3\right)$
- $\sum_{i=1}^{n} i \leq n \cdot n = O\left(n^2\right)$
- $\log(n!) =$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \leq n \cdot n^2 = O\left(n^3\right)$
- $\sum_{i=1}^{n} i \leq n \cdot n = O\left(n^2\right)$
- $\log(n!) = \log(n) + \cdots + log 1 = O(n \log n)$

Big-Oh: More Examples

- $\frac{n^2}{2} - 3n = O\left(n^2\right)$
- $1 + 4n = O(n)$
- $\log_{10} n = \frac{\log_2 n}{\log_2 10} = O(\log_2 n) = O(\log n)$
- $\sin n = O(1), 10 = O(1), 10^{10} = O(1)$
- $\sum_{i=1}^{n} i^2 \leq n \cdot n^2 = O\left(n^3\right)$
- $\sum_{i=1}^{n} i \leq n \cdot n = O\left(n^2\right)$
- $\log(n!) = \log(n) + \cdots + log 1 = O(n \log n)$
- $\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$ (Harmonic Series, 调和级数)

Big-Oh: More Examples

The Asymptotic Upper Bound of Harmonic Series:

$$\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$$

Big-Oh: More Examples

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

Big-Oh: More Examples

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$< \frac{1}{1} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \cdots + \frac{1}{\mathrm{n}/2} + \frac{1}{\mathrm{n}}$$

Big-Oh: More Examples

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$< \frac{1}{1} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \cdots + \frac{1}{\mathrm{n}/2} + \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + 2 \cdot \left(\frac{1}{2}\right) + 4 \cdot \left(\frac{1}{4}\right) + 8 \cdot \left(\frac{1}{8}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n/2}\right) + \frac{1}{n}$$

Big-Oh: More Examples

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$< \frac{1}{1} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \cdots + \frac{1}{\mathrm{n}/2} + \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + 2 \cdot \left(\frac{1}{2}\right) + 4 \cdot \left(\frac{1}{4}\right) + 8 \cdot \left(\frac{1}{8}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n/2}\right) + \frac{1}{n}$$

$$= 1/n + \sum_{j=0}^{\log n - 1} 1$$

Big-Oh: More Examples

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$< \frac{1}{1} + \frac{1}{2} + \frac{1}{\color{red}{2}} + \frac{1}{4} + \frac{1}{\color{red}{4}} + \frac{1}{\color{red}{4}} + \frac{1}{\color{red}{4}} + \frac{1}{8} + \frac{1}{\color{red}{8}} + \frac{1}{\color{red}{8}} + \cdots + \frac{1}{\mathrm{n}/2} + \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + 2 \cdot \left(\frac{1}{2}\right) + 4 \cdot \left(\frac{1}{4}\right) + 8 \cdot \left(\frac{1}{8}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n/2}\right) + \frac{1}{n}$$

$$= 1/n + \sum_{j=0}^{\log n - 1} 1$$

$$= \log n + \frac{1}{n}$$

Big-Oh: More Examples

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i=1}}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$< \frac{1}{1} + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \cdots + \frac{1}{\mathrm{n}/2} + \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + 2 \cdot \left(\frac{1}{2}\right) + 4 \cdot \left(\frac{1}{4}\right) + 8 \cdot \left(\frac{1}{8}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n/2}\right) + \frac{1}{n}$$

$$= 1/n + \sum_{j=0}^{\log n - 1} 1$$

$$= \log n + \frac{1}{n}$$

Thus, $\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$

Big-Oh: Examples of Complexity Analysis

algorithm scan(v)

1. **for** $i = 1$ **to** n **do**
2.   **if** $S[i] = v$ **then**
3.   **return** *yes*
4. **return** *no*

Big-Oh: Examples of Complexity Analysis

algorithm scan(v)

1. **for** $i = 1$ **to** n **do**
2. **if** $S[i] = v$ **then**
3. **return** *yes*
4. **return** *no*

$\left.\begin{array}{l} \textbf{if } S[i] = v \textbf{ then} \\ \quad \textbf{return } yes \end{array}\right\} O(1)$

Big-Oh: Examples of Complexity Analysis

algorithm scan(v)

1. **for** $i = 1$ **to** n **do**
2. **if** $S[i] = v$ **then**
3. **return** *yes*
4. **return** *no*

$$\left.\begin{array}{l}\text{for } i = 1 \text{ to } n \text{ do} \\ \left.\begin{array}{l}\text{if } S[i] = v \text{ then} \\ \text{return yes}\end{array}\right\} O(1)\end{array}\right\} n \cdot O(1) = O(n)$$

Big-Oh: Examples of Complexity Analysis

algorithm scan(v)

1. **for** $i = 1$ **to** n **do**
2.   **if** $S[i] = v$ **then**
3.   **return** *yes*
4. **return** *no*

$O(1)$

$n \cdot O(1) = O(n)$

Although Lines 2-3 may be executed less than n times, we are considering the worst-case complexity

Big-Oh: Examples of Complexity Analysis

algorithm CountingInversedPairs($A[1..n]$)

1. $ans = 0$
2. **for** $i = 1$ **to** n **do**
3.   **for** $j = i + 1$ **to** n **do**
4.   **if** $A[i] > A[j]$ **then**
5. $ans = ans + 1$
6. **return** ans

What's the worst-case complexity of this program?

Big-Oh: Examples of Complexity Analysis

algorithm CountingInversedPairs($A[1..n]$)

1. $ans = 0$
2. **for** $i = 1$ **to** n **do**
3. **for** $j = i + 1$ **to** n **do**
4. **if** $A[i] > A[j]$ **then**
5. $ans = ans + 1$
6. **return** ans

Big-Oh: Examples of Complexity Analysis

algorithm CountingInversedPairs($A[1..n]$)

1. $ans = 0$
2. **for** $i = 1$ **to** n **do**
3. **for** $j = i + 1$ **to** n **do**
4. **if** $A[i] > A[j]$ **then**
5. $ans = ans + 1$
6. **return** ans

$\left.\begin{array}{l} \textbf{if } A[i] > A[j] \textbf{ then} \\ \quad ans = ans + 1 \end{array}\right\} O(1)$

Big-Oh: Examples of Complexity Analysis

algorithm CountingInversedPairs($A[1..n]$)

```
ans = 0
for i = 1 to n do
    for j = i + 1 to n do
        if A[i] > A[j] then                } (n − i)O(1)
            ans = ans + 1                  } O(1)
return ans
```

Big-Oh: Examples of Complexity Analysis

algorithm CountingInversedPairs($A[1..n]$)

1. $ans = 0$
2. **for** $i = 1$ **to** n **do**
3. **for** $j = i + 1$ **to** n **do**
4. **if** $A[i] > A[j]$ **then**
5. $ans = ans + 1$
6. **return** ans

$$\left.\begin{array}{l}\text{for } i = 1 \text{ to } n \text{ do}\\ \left.\begin{array}{l}\text{for } j = i+1 \text{ to } n \text{ do}\\ \left.\begin{array}{l}\text{if } A[i] > A[j] \text{ then}\\ \quad ans = ans + 1\end{array}\right\} O(1)\end{array}\right\} (n-i)O(1)\end{array}\right\} ??$$

Big-Oh: Examples of Complexity Analysis

algorithm CountingInversedPairs($A[1..n]$)

$ans = 0$

$$\left.\begin{array}{l} \textbf{for } i = 1 \textbf{ to } n \textbf{ do} \\ \left.\begin{array}{l} \quad \textbf{for } j = i+1 \textbf{ to } n \textbf{ do} \\ \left.\begin{array}{l} \quad\quad \textbf{if } A[i] > A[j] \textbf{ then} \\ \quad\quad\quad ans = ans + 1 \end{array}\right\} O(1) \end{array}\right\} (n-i)O(1) \end{array}\right\} ??$$

return ans

$$\begin{aligned} ?? &= (n-1)O(1) + (n-2)O(1) + \cdots + (n-n)O(1) \\ &= n(n-1)/2(O(1)) \\ &= O(n^2) \end{aligned}$$

Outline

- Asymptotic Notations（渐近记号）
  - Big-Oh
  - Big-Omega
  - Big-Theta
  - Algorithm Design and Algorithm Turing
- Solving Recurrences
  - Recursion-tree Method（递归树法）
  - Substitution Method（代入法/替代法）
  - Master Method and Master Theorem (主方法)

Big-Omega

Asymptotic lower bound

Definition (big-Omega)

$f(n) = \Omega(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.

Big-Omega

Asymptotic lower bound

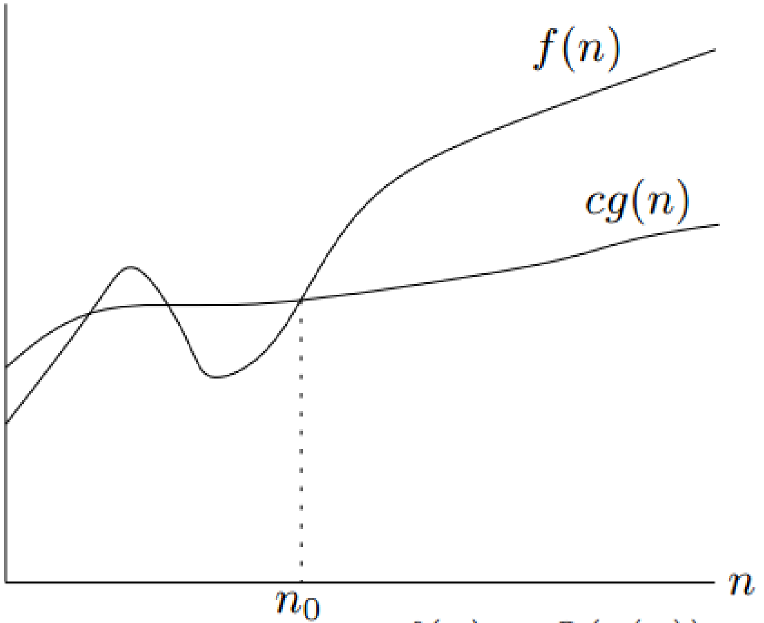
Definition (big-Omega)

$f(n) = \Omega(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.

It is easy to show that

$\frac{n^2}{2} - 3n \geq \frac{n^2}{4}$ for all $n \geq 12$.

Thus, $n^2/2 - 3n = \Omega(n^2)$.



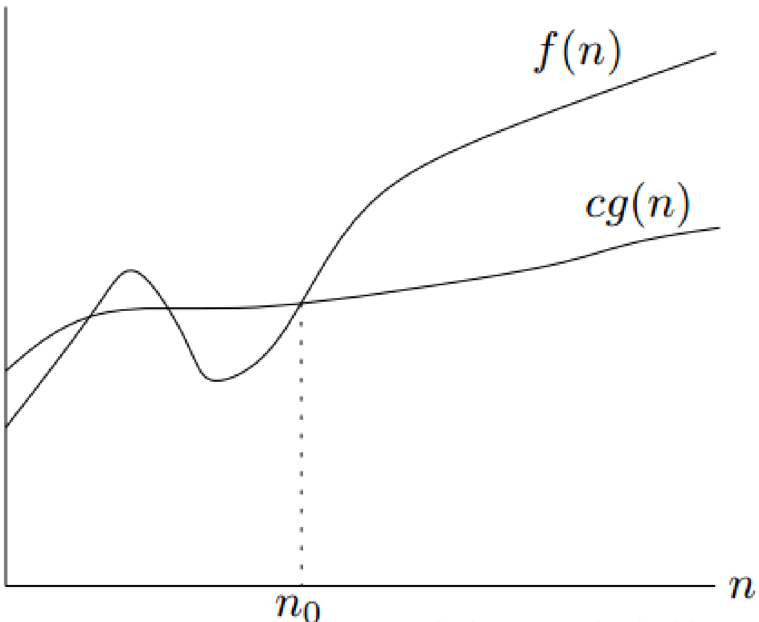
$f(n) = \Omega(g(n))$

Big-Omega

Asymptotic lower bound

Definition (big-Omega)

$f(n) = \Omega(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.



$f(n) = \Omega(g(n))$

It is easy to show that

$\frac{n^2}{2} - 3n \geq \frac{n^2}{4}$ for all $n \geq 12$.

Thus, $n^2/2 - 3n = \Omega(n^2)$.

Example

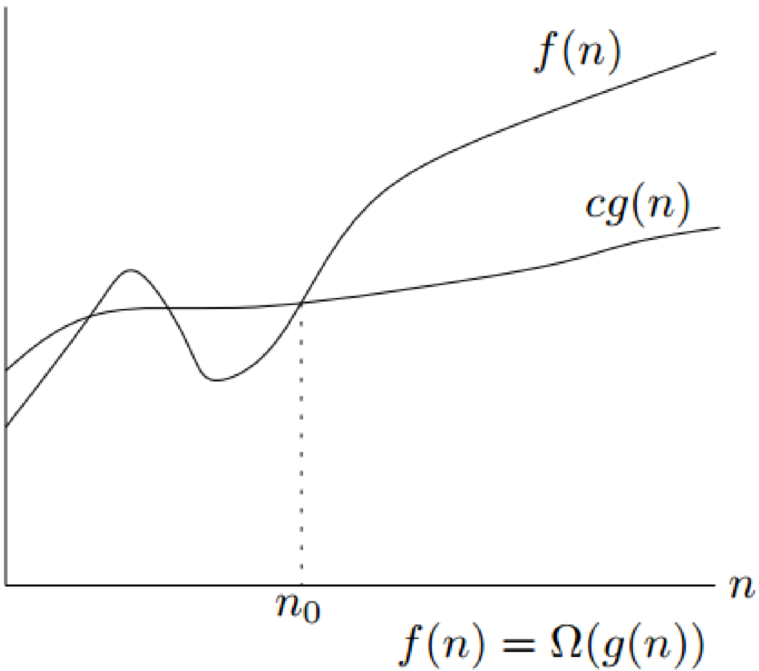
$$\log(n!) \quad = \quad \log(n) + \log(n-1) + \cdots + \log 1$$

Big-Omega

Asymptotic lower bound

Definition (big-Omega)

$f(n) = \Omega(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.



It is easy to show that

$\frac{n^2}{2} - 3n \geq \frac{n^2}{4}$ for all $n \geq 12$.

Thus, $n^2/2 - 3n = \Omega(n^2)$.

Example

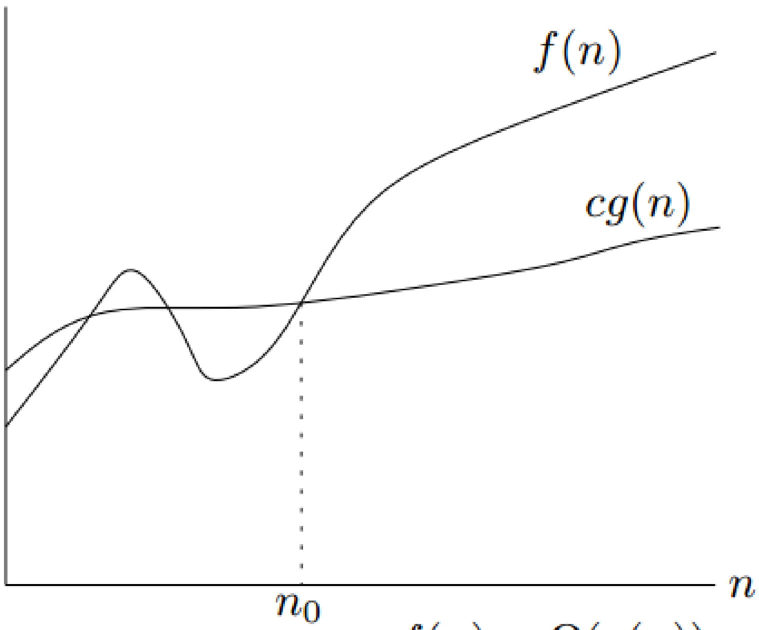
$$\begin{aligned} \log(n!) &= \log(n) + \log(n-1) + \cdots + \log 1 \\ &\geq \log(n) + \log(n-1) + \cdots + \log(n/2) \end{aligned}$$

Big-Omega

Asymptotic lower bound

Definition (big-Omega)

$f(n) = \Omega(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.



$f(n) = \Omega(g(n))$

It is easy to show that

$\frac{n^2}{2} - 3n \geq \frac{n^2}{4}$ for all $n \geq 12$.

Thus, $n^2/2 - 3n = \Omega(n^2)$.

Example

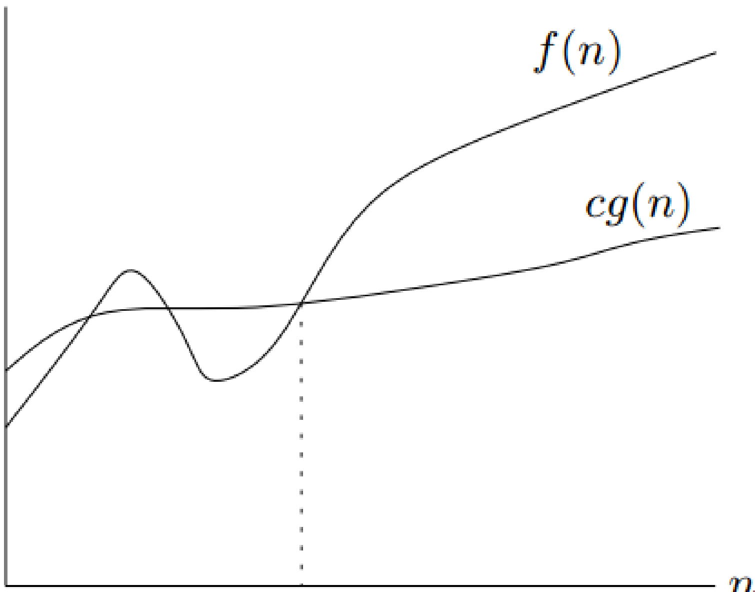
$$
\begin{aligned}
\log(n!) &= \log(n) + \log(n-1) + \cdots + \log 1 \\
&\geq \log(n) + \log(n-1) + \cdots + \log(n/2) \\
&\geq n/2 \cdot \log(n/2)
\end{aligned}
$$

Big-Omega

Asymptotic lower bound

Definition (big-Omega)

$f(n) = \Omega(g(n))$: There exists constant $c > 0$ and n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$.



It is easy to show that

$\frac{n^2}{2} - 3n \geq \frac{n^2}{4}$ for all $n \geq 12$.

Thus, $n^2/2 - 3n = \Omega(n^2)$.

Example

$$\begin{aligned}
\log(n!) &= \log(n) + \log(n-1) + \cdots + \log 1 \\
&\geq \log(n) + \log(n-1) + \cdots + \log(n/2) \\
&\geq n/2 \cdot \log(n/2) \\
&= n/2 \cdot (\log n - 1) = \Omega(n \log n).
\end{aligned}$$

Big-Omega: Harmonic Series

The Asymptotic Lower Bound of Harmonic Series:

$$\sum_{i=1}^{n} \frac{1}{i} = \Omega(\log n)$$

Big-Omega: Harmonic Series

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

Big-Omega: Harmonic Series

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$> \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \cdots \frac{1}{\mathrm{n}}$$

Big-Omega: Harmonic Series

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$> \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \cdots \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + \frac{1}{2} + 2 \cdot \left(\frac{1}{4}\right) + 4 \cdot \left(\frac{1}{8}\right) + 8 \cdot \left(\frac{1}{16}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n}\right)$$

Big-Omega: Harmonic Series

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$> \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \cdots \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + \frac{1}{2} + 2 \cdot \left(\frac{1}{4}\right) + 4 \cdot \left(\frac{1}{8}\right) + 8 \cdot \left(\frac{1}{16}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n}\right)$$

$$= 1 + \sum_{j=1}^{\log n} \frac{1}{2}$$

Big-Omega: Harmonic Series

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$> \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \cdots \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + \frac{1}{2} + 2 \cdot \left(\frac{1}{4}\right) + 4 \cdot \left(\frac{1}{8}\right) + 8 \cdot \left(\frac{1}{16}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n}\right)$$

$$= 1 + \sum_{j=1}^{\log n} \frac{1}{2}$$

$$= 1 + \frac{1}{2} \log n$$

Big-Omega: Harmonic Series

Proof: Assume that n is a power of two, then

$$\sum_{\mathrm{i}=1}^{n} \frac{1}{i}$$

$$= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \frac{1}{9} + \frac{1}{10} + \cdots + \frac{1}{n}$$

$$> \frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} + \cdots \frac{1}{\mathrm{n}}$$

$$= \frac{1}{1} + \frac{1}{2} + 2 \cdot \left(\frac{1}{4}\right) + 4 \cdot \left(\frac{1}{8}\right) + 8 \cdot \left(\frac{1}{16}\right) + \cdots + \frac{n}{2}\left(\frac{1}{n}\right)$$

$$= 1 + \sum\nolimits_{j=1}^{\log n} \frac{1}{2}$$

$$= 1 + \frac{1}{2} \log n$$

Thus, $\sum_{i=1}^{n} \frac{1}{i} = \Omega(\log n)$

Outline

- Asymptotic Notations（渐近记号）
  - Big-Oh
  - Big-Omega
  - Big-Theta
  - Algorithm Design and Algorithm Turing
- Solving Recurrences
  - Recursion-tree Method（递归树法）
  - Substitution Method（代入法/替代法）
  - Master Method and Master Theorem (主方法)

Big-Theta

Asymptotic tight bound

Definition (big-Theta)

$f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Big-Theta

Asymptotic tight bound

Definition (big-Theta)

$f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

We have shown that

$$n^2/2 - 3n = O(n^2),$$

and

$$n^2/2 - 3n = \Omega(n^2).$$

Therefore, we have that $n^2/2 - 3n = \Theta(n^2)$.

Big-Theta

Asymptotic tight bound

Definition (big-Theta)

$f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

We have shown that

$$n^2/2 - 3n = O(n^2),$$

and

$$n^2/2 - 3n = \Omega(n^2).$$

Therefore, we have that $n^2/2 - 3n = \Theta(n^2)$.

Usually (and in this course), it is sufficient to show only upper bounds (big-Oh), though we should try to make these as tight as we can.

Asymptotic Notations

Upper bounds. T(n)=O(f(n)) if there exist constants c > 0 and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \leq c \cdot f(n)$.

Equivalent definition: $\lim_{n \longrightarrow \infty} \frac{T(n)}{f(n)} < \infty$.

Lower bounds. T(n)=Ω(f(n)) if there exist constants c> 0 and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \geq c \cdot f(n)$.

Equivalent definition: $\lim_{n \longrightarrow \infty} \frac{T(n)}{f(n)} > 0$.

Tight bounds. $T(n) = \Theta\big(f(n)\big)$ if $T(n) = O\big(f(n)\big)$ and $T(n) = \Omega(f(n))$.

Note: Here "=" means "is", not equal. The more mathematically correct way should be $T(n) \in O(f(n))$.

For example, for the harmonic series,

we have: $\sum_{i=1}^{n} \frac{1}{n} = \mathrm{O}(\log n) = \Omega(\log n) = \Theta(\log n)$

Examples

- $100n^2 = O(n^3)$?

- $100n^2 = \Omega(n^3)$?

- $10n^2 - 100n = O(n^2)$?
- $10n^2 - 100n = \Omega(n^2)$?
- $10n^2 - 100n = \Theta(n^2)$?
- $\log(2n) = O(\log n)$?
- $(2n)^{10} = O(n^{10})$?
- $2^{2n} = O(2^n)$?

Examples

- $100n^2 = O(n^3)$?
  Answer: Yes. $C = 1$ and $n_0 = 100$. Then $\forall n \geq n_0$, $100n^2 \leq C \cdot n^3$.
- $100n^2 = \Omega(n^3)$?
- $10n^2 - 100n = O(n^2)$?
- $10n^2 - 100n = \Omega(n^2)$?
- $10n^2 - 100n = \Theta(n^2)$?
- $\log(2n) = O(\log n)$?
- $(2n)^{10} = O(n^{10})$?
- $2^{2n} = O(2^n)$?

Examples

- $100n^2 = O(n^3)$?
  Answer: Yes. $C = 1$ and $n_0 = 100$. Then $\forall n \geq n_0$, $100n^2 \leq C \cdot n^3$.
- $100n^2 = \Omega(n^3)$?
  Answer: No. $\forall\, C > 0$, $n_0 > 0$, there exists $n > n_0$ ($n = n_0 + 100/C$) such that $100n^2 < C \cdot n^3$.
- $10n^2 - 100n = O(n^2)$?
- $10n^2 - 100n = \Omega(n^2)$?
- $10n^2 - 100n = \Theta(n^2)$?
- $\log(2n) = O(\log n)$?
- $(2n)^{10} = O(n^{10})$?
- $2^{2n} = O(2^n)$?

Examples

- $100n^2 = O(n^3)$?
  Answer: Yes. $C = 1$ and $n_0 = 100$. Then $\forall n \geq n_0$, $100n^2 \leq C \cdot n^3$.
- $100n^2 = \Omega(n^3)$?
  Answer: No. $\forall\, C > 0$, $n_0 > 0$, there exists $n > n_0$ ($n = n_0 + 100/C$) such that $100n^2 < C \cdot n^3$.
- $10n^2 - 100n = O(n^2)$? √
- $10n^2 - 100n = \Omega(n^2)$? √
- $10n^2 - 100n = \Theta(n^2)$? √
- $\log(2n) = O(\log n)$? √
- $(2n)^{10} = O(n^{10})$? √
- $2^{2n} = O(2^n)$? ×

Solutions

- $10n^2 - 100n = O(n^2)$?

- $10n^2 - 100n = \Omega(n^2)$?

- $10n^2 - 100n = \Theta(n^2)$?

Solutions

- $10n^2 - 100n = O(n^2)$?
  Answer: **Yes**. $\forall\, n > 0,\ 10n^2 - 100n \le 10n^2$.

- $10n^2 - 100n = \Omega(n^2)$?

- $10n^2 - 100n = \Theta(n^2)$?

Solutions

- $10n^2 - 100n = O(n^2)$?
  Answer: **Yes**. $\forall\ n > 0,\ 10n^2 - 100n \leq 10n^2$.

- $10n^2 - 100n = \Omega(n^2)$?
  Answer: **Yes**. $\forall\ n \geq 20,\ 10n^2 - 100n \geq 5n^2$.

- $10n^2 - 100n = \Theta(n^2)$?

Solutions

- $10n^2 - 100n = O(n^2)$?
  Answer: **Yes**. $\forall$ n > 0, $10n^2 - 100n \le 10n^2$.

- $10n^2 - 100n = \Omega(n^2)$?
  Answer: **Yes**. $\forall$ n ≥ 20, $10n^2 - 100n \ge 5n^2$.

- $10n^2 - 100n = \Theta(n^2)$?
  Answer: **Yes**. Because

$$10n^2 - 100n = O(n^2)$$

and

$$10n^2 - 100n = \Omega(n^2)$$

Solutions

- $\log(2n) = O(\log n)$?

- $(2n)^{10} = O(n^{10})$?

- $2^{2n} = O(2^n)$?

An interesting fact about logarithm

$$\boldsymbol{log_{b_1} n = O(log_{b_2} n)}$$

For any constant $b_1 > 1$ and $b_2 > 1$.

For example, let us verify $\log_2 n = O(\log_3 n)$.

Notice that

$$\log_3 n = \frac{\log_2 n}{\log_2 3} \Longrightarrow \log_2 n = \log_2 3 \cdot \log_3 n$$

Hence, we can set $c_1 = \log_2 3$ and $c_2 = 1$, which makes

$$\log_2 n \leq c_1 \log_3 n$$

Hold for all $n \geq c_2$.

An interesting fact about logarithm

$$\boldsymbol{log_{b_1} n = O(log_{b_2} n)}$$

For any constant $b_1 > 1$ and $b_2 > 1$.

Because of the above, in computer science, we omit all the constant logarithm bases in big-O. For example, instead of $O(\log_2 n)$, we will simply write O(log n)

- Essentially, this says that "you are welcome to put any constant base there, and it will be the same asymptotically".
- Obviously, Ω, Θ also have this property.

Outline

- Asymptotic Notations（渐近记号）
  - Big-Oh
  - Big-Omega
  - Big-Theta
  - Algorithm Design and Algorithm Turing
- Solving Recurrences
  - Recursion-tree Method（递归树法）
  - Substitution Method（代入法/替代法）
  - Master Method and Master Theorem (主方法)

Some Thoughts on Algorithm Design

- Algorithm Design, as taught in this class, is mainly about designing algorithms that have big-Oh running times.
- As n gets larger and larger, O(nlogn) algorithms will run faster than $O(n^2)$ ones and O(n) algorithms will beat O(nlogn) ones.
- Good algorithm design & analysis allows you to identify the hard parts of your problem and deal with them effectively.
- Too often, programmers try to solve problems using brute force techniques and end up with slow complicated code!
- A few hours of abstract thought devoted to algorithm design often results in faster, simpler, and more general solutions.

Algorithm Tuning

- After algorithm design one can continue on to Algorithm tuning
  - concentrate on improving algorithms by cutting down on the constants in the big O() bounds.
  - needs a good understanding of both algorithm design principles and efficient use of data structures.
- In this course we will not go further into algorithm tuning
  - For a good introduction, see chapter 9 in Programming Pearls, 2nd ed by Jon Bentley