Introduction to Algorithms

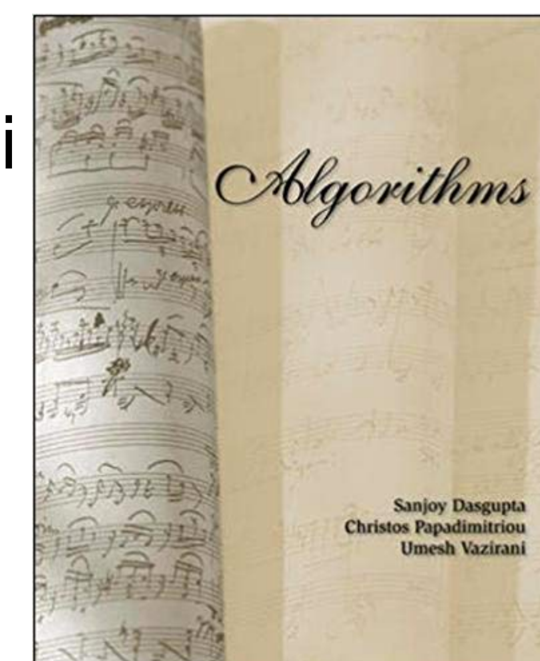
Gou Guanglei(苟光磊)
ggl@cqut.edu.cn

Teaching staff

- Instructor: Gou Guanglei(苟光磊)
  - Office room: Lab Building No.1 B209
  - Office hour: Monday 14:00-16:00
  - Email: ggl@cqut.edu.cn
- TA: None for now.

Course information

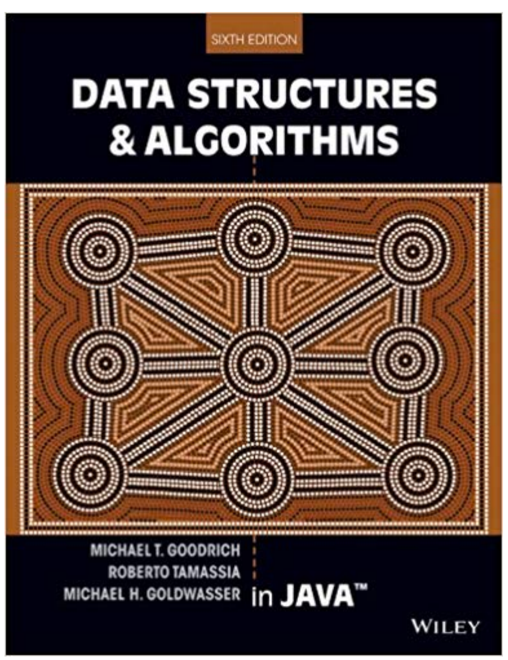
- This course introduces basic concept of design and analysis of algorithms.
- Prerequisites : discrete mathematics, data structures, basic probability
- Textbook:
  - Algorithms by Dasgupta, Papadimitriou, and Vazirani

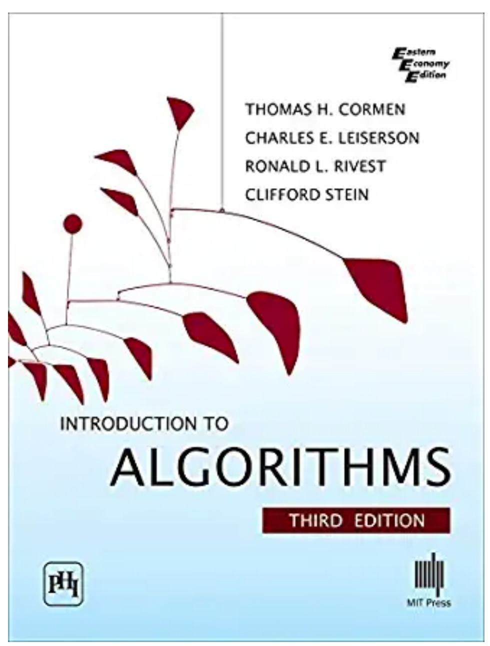


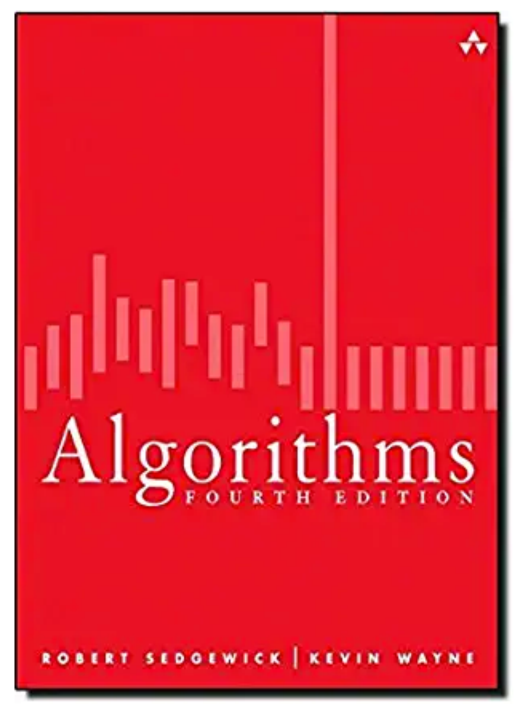
Course material

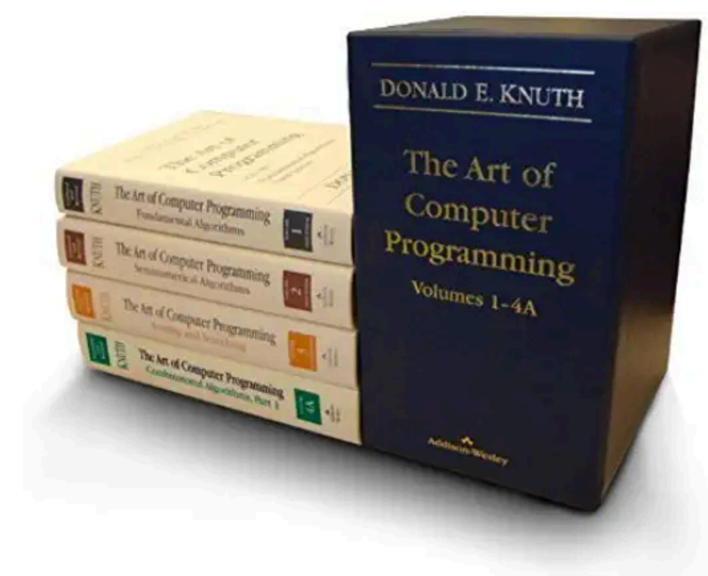
- Web Site (youku, baidu, opencourse)

- Some recommended books









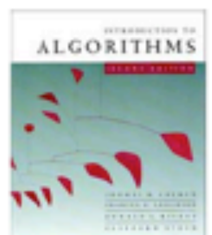
Evaluation

- Participation(attendance, quiz): 10%
- Homework: 20%
- Presentation: 20%
- Final term: 50%

Why

- **Programming == Data Structures + Algorithms**

- **Thinking and solving problems like computer scientist**



Why study algorithms 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.
- Performance is the ***currency*** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

How to study

- Understanding lectures is not enough
- Doing exercises on your own solutions.
- Teaching is best way to learn. Try to explain your idea to your friends.
- Make study groups to discuss problems.

Contents

- Design paradigms
  - **Divide and conquer**
  - **Dynamic programming**
  - **Greedy algorithms**
  - Randomized algorithms
- Analysis techniques
  - **Recurrences**
  - **Asymptotic analysis**
  - Probabilities analysis
- Graph algorithms
  - **Minimum spanning tree**
  - **Shortest path**
- **NP-completeness**

Algorithms

- **Definition**: A well-defined computational procedure to solve a computational ***problem*** (to transform some input into a desired output).

- Statement of the ***problem*** specifies the desired ***input/ output relationship***.

- Algorithm describes a specific computational procedure for achieving that input/output relationship.

Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

$0 \quad 1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad \cdots$

```
function fib1(n)
if n = 0:  return 0
if n = 1:  return 1
return fib1(n - 1) + fib1(n - 2)
```

Analysis

- Let T(n) denote computer steps needed to compute fib1(n).
- $T(n) \leq 2$ for $n \leq 1$
- $T(n) = T(n-1) + T(n-2) + 3$ for $n > 1$
- $T(n) \geq Fn$
- $Fn \approx 2^{0.694n}$

- T(n) is exponential in n ! – very slow except for very small n

- Can we do better ?

-
```
function fib2(n)
if n = 0 return 0
create an array f[0...n]
f[0] = 0, f[1] = 1
for i = 2...n:
    f[i] = f[i-1] + f[i-2]
return f[n]
```

- More carefully analysis
    - We counted the number of *basic computer steps* executed by each algorithm assuming that each step takes a constant amount of time.
    - Basic computer steps : branching, loading, storing, comparisons, simple arithmetic, and so on
    - This is a very useful simplification.

The problem of sorting

Input: sequence $\langle a1, a2, \ldots, an \rangle$ of numbers.

Output: permutation $\langle a'1, a'2, \ldots, a'n \rangle$ such that $a'1 \leq a'2 \leq \ldots \leq a'n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Insertion sort

INSERTION-SORT (A, n) $\quad\quad \triangleright A[1\,.\,.\,n]$

for $j \leftarrow 2$ **to** n

  do $key \leftarrow A[j]$

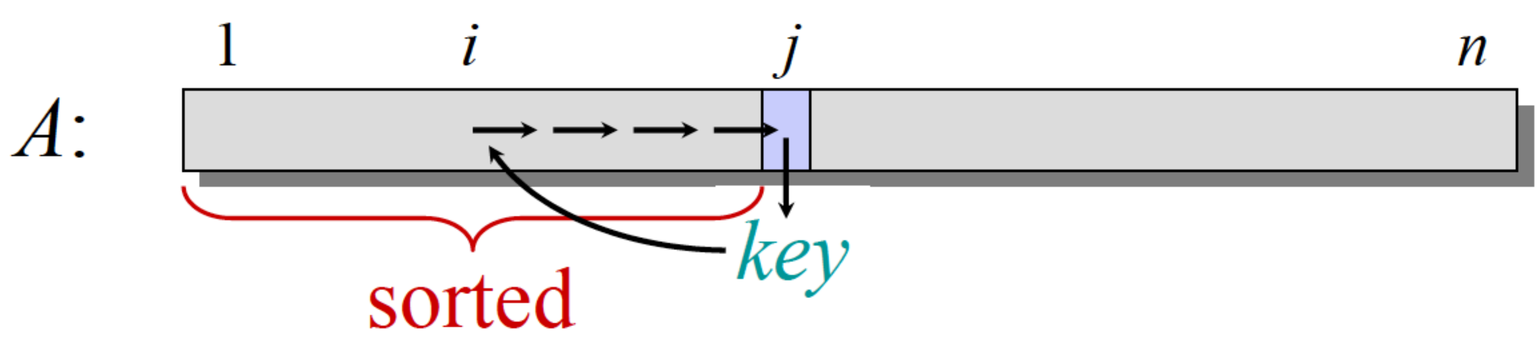
  $i \leftarrow j - 1$

  while $i > 0$ and $A[i] > key$

   do $A[i+1] \leftarrow A[i]$

    $i \leftarrow i - 1$

  $A[i+1] = key$



8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

http://algs4.cs.princeton.edu

2.1 INSERTION SORT DEMO

Insert A[i] right after the largest thing that's still smaller than A[i] .

This sounds like a job for…

Proof By Induction!

- **Inductive hypothesis.** After iteration i of the outer loop, `A[:i+1]` is sorted.
- **Base case.** When $i = 0$, `A[:1]` contains only one element, and this is sorted.

- **Inductive step.** Suppose that the inductive hypothesis holds for $i - 1$, so `A[:i]` is sorted after the $i - 1$'st iteration. We want to show that `A[:i+1]` is sorted after the i'th iteration.

  Suppose that j^* is the largest integer in $\{0, \ldots, i-1\}$ so that `A[j*] < A[i]`. Then the effect of the inner loop is to turn

  $$[A[0], A[1], \ldots, A[j^*], \ldots, A[i-1], A[i]]$$

  into

  $$[A[0], A[1], \ldots, A[j^*], A[i], A[j^* + 1], \ldots, A[i-1]].$$

  We claim that this latter list is sorted. This is because $A[i] > A[j^*]$, and by the inductive hypothesis, we have $A[j^*] \geq A[j]$ for all $j \leq j^*$, and so $A[i]$ is larger than everything that is positioned before it. Similarly, by the choice of j^* we have $A[i] \leq A[j^* + 1] \leq A[j]$ for all $j \geq j^* + 1$, so $A[i]$ is smaller than everything that comes after it. Thus, $A[i]$ is in the right place. All of the other elements were already in the right place, so this proves the claim.

  Thus, after the i'th iteration completes, `A[:i+1]` is sorted, and this establishes the inductive hypothesis for i.

- What does the running time depend on?
- What is best/worst running time?

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: short sequences are easier to sort than long ones.
- Usually, interested in the worst-case running time because
  - – It gives an upper bound (because everybody likes a guarantee)
  - – For some algorithms, the worst case occurs often.
  - – Average case is often as bad as the worst case.

Kinds of analyses

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 statistical distribution of inputs.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on someinput.

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 machines).

BIG IDEA:

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

"Asymptotic Analysis"

Asymptotic Analysis

- **Look only at the leading term of the formula for running time.**

Example : for insertion sort, the worst-case running time is $an^2 + bn + c$.

It grows like n^2 .

Asymptotic notation

O-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $\mathbf{0 \leq f(n) \leq cg(n)}$ for all $n \geq n_0$.

For example

$$T(n) = 3n^2 + 17$$

$$T(n) = O(n^2) \quad \surd$$

$$T(n) = O(n^3) \quad \surd$$

Ω-notation (lower bounds)

Ω(g(n))= { f(n) :there exist constants c > 0, n0 > 0 such that **0 ≤ cg(n) ≤ f(n)** for all n ≥ n0}

For example

$$T(n) = 3n^2 - 2n$$

$$T(n) = \Omega(n^2) \quad \surd$$

$$T(n) = \omega(nlogn) \quad \surd$$

Θ-notation (tight bounds)

$\Theta(g(n)) = \{ f(n)$: there exist positive constants $c1$, $c2$, and $n0$ such that $\mathbf{0 \le c1\, g(n) \le f(n) \le c2\, g(n)}$ for all $n \ge n0\}$

- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

- o-notation

We write $f(n) = o(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $\mathbf{0 \le f(n) < cg(n)}$ for all $n \ge n_0$.

- ω-notation

- $\omega(g(n)) = \{ f(n)$:there exist constants $c > 0$, $n0 > 0$ such that $\mathbf{0 \le cg(n) < f(n)}$ for all $n \ge n0\}$

Theta	$f(n) = \theta(g(n))$	$f(n) \approx c\, g(n)$
BigOh	$f(n) = O(g(n))$	$f(n) \leq c\, g(n)$
Omega	$f(n) = \Omega(g(n))$	$f(n) \geq c\, g(n)$
Little Oh	$f(n) = o(g(n))$	$f(n) < c\, g(n)$
Little Omega	$f(n) = \omega(g(n))$	$f(n) > c\, g(n)$

Theorem

$$f(n) = \Theta(n) \quad \text{if and only if}$$

$$f(n) = O(n) \ \text{ and } \ f(n) = \Omega(n)$$

Properties

- P1: Transitivity $f \in O(g)$ **and** $g \in O(h) \Rightarrow f \in O(h)$

  How about Ω, θ, o, ω ?

- P2: Duality $f \in O(g) \Leftrightarrow g \in \Omega(f)$
- P3: $f \in \Theta(g) \Rightarrow g \in \Theta(f)$
- P4: $O(f + g) = O(max\{f, g\})$