Foundations

Bin Wang

School of Software
Tsinghua University

February 27, 2018

Outline

Staff

Teacher

Name: 王斌
Email: wangbins@tsinghua.edu.cn
Telephone: 62795457
网络学堂：http://learn.cic.tsinghua.edu.cn/

TA

潘毅铃(452765625@qq.com)
潘天翔(ptx11@mails.tsinghua.edu.cn)

Prerequisites

Textbook

1. CLRS, **Introduction to Algorithms (3rd edition)**, (2009), The MIT Press.

Reference

- Anany Levitin, 算法分析与设计基础, 潘彦译, (2004), 清华大学出版社
- 王晓东, 计算机算法设计与分析, 第四版, (2012), 电子工业出版社

Prerequisites

Textbook

1. CLRS, **Introduction to Algorithms (3rd edition)**, (2009), The MIT Press.

Reference

- Donald E. Knuth(高德纳), The Art of Computer Programming (TAOCP), vol 1, 2, 3, 4A, addison-wesley publishing company.
- `http://www-cs-staff.stanford.edu/~uno/`

Prerequisites

Textbook

1. CLRS, **Introduction to Algorithms (3rd edition)**, (2009), The MIT Press.

Reference

- `http://acm.pku.edu.cn/JudgeOnline/`
- `http://en.wikipedia.org`
- `http://sourceforge.net/`

Topics

Course Schedule

1. Foundations & Divide-and-Conquer.
2. Sorting algorithms.
3. Dynamic programming.
4. Greedy algorithm.
5. Amortized analysis, Heaps.
6. String match.
7. NPC, Approximation algorithms.
8. Multithreaded algorithms

Policy

Grading Policy

- 上半学期(50%)
  - 平时作业与课堂表现(15%)
  - 期末考试(35%)，第17或18周考
- 下半学期(50%)
  - 平时作业与课堂表现(15%)
  - 大作业(35%)

Collaboration Policy

- 不能抄袭
- 引用他人成果需指明出处

Policy

Homework Policy

- 编程语言：C/C++/C #/Java/Python; 作业文档：Latex/Doc;
- 没有在规定时间内提交作业者，每迟交一天，扣10分，扣完为止;
- 交作业时漏交某些题目，每迟交一天，扣漏交题目分数的10%，扣完为止;
- 如果提交时网络学堂有故障，请在半小时内发邮件给助教，超过半小时按迟交处理;

What's algorithm?

Definition

An algorithm is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**. An algorithm is thus a sequence of computational steps that transform the input into the output.

What's algorithm?

Example

Sorting problem:

- **Input:** A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
- **Output:** A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Analysis of algorithms

Definition

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

What's more important than performance?

- correctness
- programmer time
- maintainability
- robustness
- user-friendliness

Analysis of algorithms

Definition

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

What's more important than performance?

- correctness
- programmer time
- maintainability
- robustness
- user-friendliness

Analysis of algorithms

Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Analysis of algorithms helps us to understand scalability.
- Algorithmic mathematics provides a language for talking about program behavior.
- The lessons of program performance generalize to other computing resources.

Analysis of algorithms

Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Analysis of algorithms helps us to understand scalability.
- Algorithmic mathematics provides a language for talking about program behavior.
- The lessons of program performance generalize to other computing resources.

Analysis of algorithms

Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Analysis of algorithms helps us to understand scalability.
- Algorithmic mathematics provides a language for talking about program behavior.
- The lessons of program performance generalize to other computing resources.

Analysis of algorithms

Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Analysis of algorithms helps us to understand scalability.
- Algorithmic mathematics provides a language for talking about program behavior.
- The lessons of program performance generalize to other computing resources.

Analysis of algorithms

Why study algorithms and performance?

- Performance often draws the line between what is feasible and what is impossible.
- Analysis of algorithms helps us to understand scalability.
- Algorithmic mathematics provides a language for talking about program behavior.
- The lessons of program performance generalize to other computing resources.

Analysis of algorithms

Practical Use of algorithm

- The Human Genome Project has the goals of identifying all the $100,000$ genes in human DNA, determining the sequences of the 3 billion chemical base pairs that make up human DNA, storing this information in databases, and developing tools for data analysis.

Analysis of algorithms

Practical Use of algorithm

- The Internet enables people all around the world to quickly access and retrieve large amounts of information.
- Electronic commerce enables goods and services to be negotiated and exchanged electronically.

Some questions

Given a problem, can we find an algorithm to solve it?

Not always!

Hilbert's 10th Problem

What is a good algorithm?

Time is important!

Is a "good" algorithm always exist?

Not clear now!

Some questions

Given a problem, can we find an algorithm to solve it?

Not always!

Hilbert's 10th Problem

What is a good algorithm?

Time is important!

Is a "good" algorithm always exist?

Not clear now!

Some questions

Given a problem, can we find an algorithm to solve it?

Not always!

Hilbert's 10th Problem

What is a good algorithm?

Time is important!

Is a "good" algorithm always exist?

Not clear now!

The problem of sorting

Input

A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output

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

Example

Input: $8, 2, 4, 9, 3, 6.$

Output: $2, 3, 4, 6, 8, 9.$

The problem of sorting

Input

A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output

A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence 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

INSERT-SORT(A)

```
for j = 2 to A.length
        key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1]
        i = j − 1
        while i > 0 and A[i] > key
                A[i + 1] = A[i]
                i = i − 1
        A[i + 1] = key
```

Insertion sort

Example

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

Insertion sort

Table: Analysis of INSERT-SORT

INSERT-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
do $key = A[j]$	c_2	$n - 1$
// Insert $A[j]$	0	0
$i = j - 1$	c_4	$n - 1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n - 1$

Insertion sort

Analysis of INSERT-SORT

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

Insertion sort

Best case

In INSERT-SORT, the best case occurs if the array is already sorted.

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

The time can be expressed as $an + b$; it is thus a **linear function** of n.

Insertion sort

Best case

In INSERT-SORT, the best case occurs if the array is already sorted.

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

The time can be expressed as $an + b$; it is thus a **linear function** of n.

Insertion sort

Best case

In INSERT-SORT, the best case occurs if the array is already sorted.

$$
\begin{aligned}
T(n) = {} & (c_1 + c_2 + c_4 + c_5 + c_8)n \\
& - (c_2 + c_4 + c_5 + c_8)
\end{aligned}
$$

The time can be expressed as $an + b$; it is thus a **linear function** of n.

Insertion sort

Worst-cse

If the array is in reverse sorted order, the worst case results.

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

The time can be expressed as $an^2 + bn + c$; it is thus a **quadratic function** of n.

Insertion sort

Worst-cse

If the array is in reverse sorted order, the worst case results.

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

The time can be expressed as $an^2 + bn + c$; it is thus a **quadratic function** of n.

Insertion sort

Worst-cse

If the array is in reverse sorted order, the worst case results.

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

The time can be expressed as $an^2 + bn + c$; it is thus a **quadratic function** of n.

Insertion sort

Running time

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Machine-independent time

Random-access machine(RAM) model

- No concurrent operations.
- Each instruction takes a constant amount of time.

Asymptotic Analysis

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

Machine-independent time

Random-access machine(RAM) model

- No concurrent operations.
- Each instruction takes a constant amount of time.

Asymptotic Analysis

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

Θ-notation

Definition

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 \in \mathbb{R}^+, s.t. \\ \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

We say that $g(n)$ is an **asymptotically tight bound** for $f(n)$. Denoted as $f(n) = \Theta(g(n))$ or $f(n) \in \Theta(g(n))$.

Θ-notation

Definition

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 \in \mathbb{R}^+, s.t.\ \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

We say that $g(n)$ is an **asymptotically tight bound** for $f(n)$. Denoted as $f(n) = \Theta(g(n))$ or $f(n) \in \Theta(g(n))$.

Θ-notation

Example

$$\frac{1}{2}n^2 - 3n = \Theta(n^2), \quad 0.001n^3 \neq \Theta(n^2),$$

$$c_0 = \Theta(1), \quad \sum_{i=0}^{d} a_i n^i = \Theta(n^d) \quad (a_d > 0).$$

Θ-notation

Example

For all $n \geq n_0$,

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2,$$

Dividing by n^2 yields,

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2.$$

Choosing $c_1 = 1/14$, $c_2 = 1/2$, and $n_0 = 7$.

Θ-notation

Example

For all $n \geq n_0$,

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2,$$

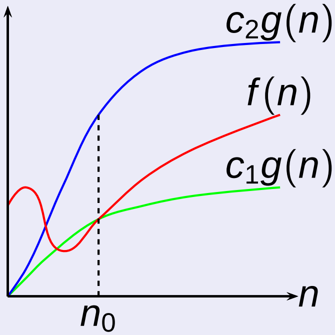
Dividing by n^2 yields,

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2.$$

Choosing $c_1 = 1/14, c_2 = 1/2$, and $n_0 = 7$.

Θ-notation

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



O-notation and Ω-notation

Definition

When we have only an **asymptotically upper bound**, we use O-notation.

$$O(g(n)) = \{f(n) : \exists c, n_0 \in \mathbb{R}^+, s.t.$$
$$\forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}$$

Denoted as $f(n) = O(g(n))$ or $f(n) \in O(g(n))$.

O-notation and Ω-notation

Definition

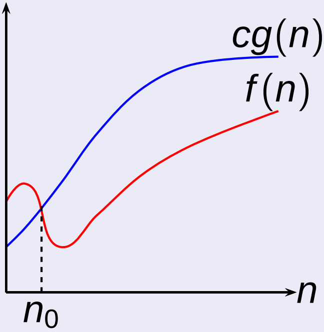
Ω-notation provides an **asymptotically lower bound**.

$$\Omega(g(n)) = \{f(n) : \exists c, n_0 \in \mathbb{R}^+, s.t. \\ \forall n \geq n_0, 0 \leq cg(n) \leq f(n)\}$$

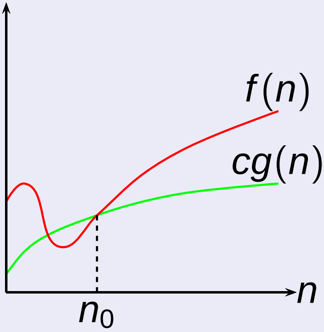
Denoted as $f(n) = \Omega(g(n))$ or $f(n) \in \Omega(g(n))$.

O-notation and Ω-notation

$f(n) = O(g(n))$



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



O-notation and Ω-notation

Example

$$n = O(n^2), \quad 2n^2 = O(n^2),$$
$$2n^2 = \Omega(n), \quad 2n^2 = \Omega(n^2).$$

O-notation and Ω-notation

Theorem 3.1

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Asymptotic notation in equations

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$$

$$\Theta(n^2) + O(n^2)$$

O-notation and Ω-notation

Theorem 3.1

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Asymptotic notation in equations

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$$

$$\Theta(n^2) + O(n^2)$$

O-notation and Ω-notation

Theorem 3.1

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Asymptotic notation in equations

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$$

$$\Theta(n^2) + O(n^2)$$

O-notation and Ω-notation

Theorem 3.1

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Asymptotic notation in equations

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2)$$

$$\Theta(n^2) + O(n^2) = \Theta(n^2)$$

o-notation and ω-notation

Definition

$$\begin{aligned} o(g(n)) &= \{f(n) : \forall c > 0, \exists n_0 > 0, \\ &\quad s.t. \forall n \geq n_0, 0 \leq f(n) < cg(n)\} \end{aligned}$$

Denoted as $f(n) = o(g(n))$. Intuitively, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$$\begin{aligned} \omega(g(n)) &= \{f(n) : \forall c > 0, \exists n_0 > 0, \\ &\quad s.t. \forall n \geq n_0, 0 \leq cg(n) < f(n)\} \end{aligned}$$

The relation $f(n) = \omega(g(n))$ implies that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

o-notation and ω-notation

Definition

$$\begin{aligned} o(g(n)) &= \{f(n) : \forall c > 0, \exists n_0 > 0, \\ &s.t. \forall n \geq n_0, 0 \leq f(n) < cg(n)\} \end{aligned}$$

Denoted as $f(n) = o(g(n))$. Intuitively, $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$$\begin{aligned} \omega(g(n)) &= \{f(n) : \forall c > 0, \exists n_0 > 0, \\ &s.t. \forall n \geq n_0, 0 \leq cg(n) < f(n)\} \end{aligned}$$

The relation $f(n) = \omega(g(n))$ implies that $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

o-notation and ω-notation

Example

$$2n = o(n^2), \quad 2n^2 \neq o(n^2),$$
$$2n^2 = \omega(n), \quad 2n^2 \neq \omega(n^2).$$

Comparison of functions

Transitivity

$f(n) = \gamma(g(n))$ and $g(n) = \gamma(h(n))$ imply
$f(n) = \gamma(h(n))$, $\gamma = \Theta, O, \Omega, o, \omega$

Reflexivity

$f(n) = \Theta(f(n)), f(n) = O(f(n)), f(n) = \Omega(f(n))$

Comparison of functions

Symmetry

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n))$$

Transpose symmetry

$$f(n) = O(g(n)) \iff g(n) = \Omega(f(n))$$
$$f(n) = o(g(n)) \iff g(n) = \omega(f(n))$$

An analogy between functions and real numbers

Asymptotic Relation between functions	Relations between real numbers
$f(n) = O(g(n))$	$a \leq b$
$f(n) = \Omega(g(n))$	$a \geq b$
$f(n) = \Theta(g(n))$	$a = b$
$f(n) = o(g(n))$	$a < b$
$f(n) = \omega(g(n))$	$a > b$

History of notation

History of noation

- O-notation was presented by P. Bachmann in 1892.
- o-notation was invented by E. Landau in 1909 for his discussion of the distribution of prime numbers.
- Ω and Θ notations were advocated by D. Knuth in 1976.

Standard notations and common functions

Floors and ceilings

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$$

For any integer n, $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$,
and for integers $a, b > 0$
$\lceil a/b \rceil \leq (a + (b-1))/b$, $\lfloor a/b \rfloor \geq ((a - (b-1))/b$

Standard notations and common functions

Logarithms

For all real $a > 0$, $b > 0$, $c > 0$, and n.

$$\log_b a = \frac{1}{\log_a b}, a^{\log_b c} = c^{\log_b a}$$

$$\frac{x}{1+x} \leq \ln(1+x) \leq x$$

Standard notations and common functions

Factorials

Stirling's approximation:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$$

Standard notations and common functions

Factorials

Stirling's approximation:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$n! = o(n^n), n! = \omega(2^n), \lg(n!) = \Theta(n \lg n)$

Standard notations and common functions

Functional iteration

$$f^{(i)}(n) = \begin{cases} n & i = 0 \\ f(f^{(i-1)}(n)) & i > 0 \end{cases}$$

The iterated logarithm function:

$$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}$$

Standard notations and common functions

Functional iteration

$$f^{(i)}(n) = \begin{cases} n & i = 0 \\ f(f^{(i-1)}(n)) & i > 0 \end{cases}$$

The iterated logarithm function:

$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}$ $\lg^* 2 = 1, \lg^* 4 = 2, \lg^* 16 = 3, \lg^* 65536 = 4, \lg^*(2^{65536}) = 5.$

Exercises

n! lgn 2^n n^4 (lgn)^2 n^(1/3) 2^n n!
2 5 6 4 7 3 1 8

Sorting the speed of growth

$(n-2)!, 5\lg(n+100)^{10}, 2^{2n}, 0.001n^4+3n^3+1, \ln^2 n, \sqrt[3]{n}, 2^n, n!$

Which is asymptotically larger

$\lg(\lg^* n)$ or $\lg^*(\lg n)$

What is recurrences?

Fibonacci numbers

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$

$$F(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

FIBONNACI(n)

if $(n = 0)$ **return** 0
if $(n = 1)$ **return** 1
return FIBONNACI($n-1$) + FIBONNACI($n-2$)

What is recurrences?

Definition

A recurrence is an equation or inequation that describes a function in terms of its value on smaller inputs.

What is recurrences?

History of recurrences

- In 1202, recurrences were studied by Leonardo Fibonacci (1170-1250).
- A. De Moivre (1667-1754) introduced the method of generating functions for solving recurrences.
- Bentley, Haken and Saxe presented the Master Theorem in 1980.

The substitution method

General method

1. **Guess** the form of the solution.
2. **Verify** by mathematical induction.

The substitution method

Example

$$T(n) = 9T(\lfloor n/3 \rfloor) + n$$

- Assume that $T(1) = \Theta(1)$
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ by induction.

The substitution method

Example

$$T(n) = 9T(\lfloor n/3 \rfloor) + n$$

- Assume that $T(1) = \Theta(1)$
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \le ck^3$ for $k < n$.
- Prove $T(n) \le cn^3$ by induction.

The substitution method

Example

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9c(n/3)^3 + n \\
&= (c/3)n^3 + n \\
&= cn^3 - ((2c/3)n^3 - n) \\
&\quad \nwarrow \textit{desired} - \textit{residual} \\
&\leq cn^3 \leftarrow \textit{desired}
\end{aligned}
$$

When $((2c/3)n^3 - n) \geq 0$, it is true.

The substitution method

Example

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9c(n/3)^3 + n \\
&= (c/3)n^3 + n \\
&= cn^3 - ((2c/3)n^3 - n) \\
&\quad \nwarrow \textcolor{red}{\textit{desired} - \textit{residual}} \\
&\leq cn^3 \leftarrow \textcolor{red}{\textit{desired}}
\end{aligned}
$$

When $((2c/3)n^3 - n) \geq 0$, it is true.

The substitution method

Example

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9c(n/3)^3 + n \\
&= (c/3)n^3 + n \\
&= cn^3 - ((2c/3)n^3 - n) \\
&\quad\ \nwarrow \textit{desired} - \textit{residual} \\
&\leq cn^3 \leftarrow \textit{desired}
\end{aligned}
$$

When $((2c/3)n^3 - n) \geq 0$, it is true. ***not tight!***

The substitution method

Example

A tighter upper bound ?

Assume $T(k) \leq ck^2$ for $k < n$

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9c(n/3)^2 + n \\
&= cn^2 + n \\
&= cn^2 - (-n) \\
&\leq cn^2
\end{aligned}
$$

We can never get $-n > 0$!

The substitution method

Example

A tighter upper bound ?

Assume $T(k) \leq ck^2$ for $k < n$

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9c(n/3)^2 + n \\
&= cn^2 + n \\
&= cn^2 - (-n) \\
&\leq cn^2 \leftarrow \textit{desired}
\end{aligned}
$$

We can never get $-n > 0$!

The substitution method

Example

A tighter upper bound ?

Assume $T(k) \leq ck^2$ for $k < n$

$$\begin{aligned} T(n) &= 9T(n/3) + n \\ &\leq 9c(n/3)^2 + n \\ &= cn^2 + n \\ &= cn^2 - (-n) \\ &\leq cn^2 \qquad\qquad \textit{Wrong}! \end{aligned}$$

We can never get $-n > 0$!

The substitution method

Example

A tighter upper bound !

Strengthen the inductive hypothesis:

Assume $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9(c_1(n/3)^2 - c_2(n/3)) + n \\
&= c_1 n^2 - 3c_2 n + n \\
&= (c_1 n^2 - c_2 n) - (2c_2 n - n) \\
&\leq c_1 n^2 - c_2 n
\end{aligned}
$$

The substitution method

Example

A tighter upper bound !

Strengthen the inductive hypothesis:

Assume $T(k) \leq c_1k^2 - c_2k$ for $k < n$

$$\begin{aligned} T(n) &= 9T(n/3) + n \\ &\leq 9(c_1(n/3)^2 - c_2(n/3)) + n \\ &= c_1n^2 - 3c_2n + n \\ &= (c_1n^2 - c_2n) - (2c_2n - n) \\ &\leq c_1n^2 - c_2n \leftarrow \textit{desired} \end{aligned}$$

The substitution method

Example

A tighter upper bound !

Strengthen the inductive hypothesis:

Assume $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$

$$
\begin{aligned}
T(n) &= 9T(n/3) + n \\
&\leq 9(c_1(n/3)^2 - c_2(n/3)) + n \\
&= c_1 n^2 - 3c_2 n + n \\
&= (c_1 n^2 - c_2 n) - (2c_2 n - n) \\
&\leq c_1 n^2 - c_2 n \qquad \text{Pick } c_2 > 1/2
\end{aligned}
$$

The substitution method

Changing variables

考试的时候较少出现

$$T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n$$

- Let $m = \lg n$, then $T(2^m) = 2T(2^{m/2}) + m$.
- Let $S(m) = T(2^m)$, then $S(m) = 2S(m/2) + m$.
- $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$.

The substitution method

Changing variables

$$T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n$$

- Let $m = \lg n$, then $T(2^m) = 2T(2^{m/2}) + m$.
- Let $S(m) = T(2^m)$, then $S(m) = 2S(m/2) + m$.
- $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$.

The recursion-tree method

Definition

- A **recursion tree** models the costs of a execution of an recursive algorithm.
- Each node of a recursion tree represents the cost of a single subproblem.
- A recursion tree is good for generating a good guess, which is then verified by the substitution method.

Example

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

The recursion-tree method

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

$$T(n)$$

The recursion-tree method

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

$$cn^2$$

$$T(n/4) \qquad\qquad T(n/2)$$

The recursion-tree method

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

$$cn^2$$

$$c(n/4)^2 \qquad\qquad c(n/2)^2$$

$$T(n/16) \quad T(n/8) \quad T(n/8) \quad T(n/4)$$

The recursion-tree method

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

cn^2

$c(n/4)^2$ $c(n/2)^2$

$c(n/16)^2$ $c(n/8)^2$ $c(n/8)^2$ $c(n/4)^2$

$\vdots$

$\Theta(1)$

The recursion-tree method

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

$$cn^2 \dashrightarrow cn^2$$

$$c(n/4)^2 \quad c(n/2)^2 \dashrightarrow 5cn^2/16$$

$$c(n/16)^2 \quad c(n/8)^2 \quad c(n/8)^2 \quad c(n/4)^2 \dashrightarrow 25cn^2/256$$

$$\vdots$$

$$\Theta(1)$$

$$T(n) = cn^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \ldots\right)$$

$$= \Theta(n^2)$$

The recursion-tree method

$$T(n) = T(\lfloor n/4 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n^2)$$

$$cn^2 \dashrightarrow cn^2$$

$$c(n/4)^2 \quad c(n/2)^2 \dashrightarrow 5cn^2/16$$

$$c(n/16)^2 \quad c(n/8)^2 \quad c(n/8)^2 \quad c(n/4)^2 \dashrightarrow 25cn^2/256$$

$$\vdots$$

$$\Theta(1)$$

$$T(n) = cn^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \ldots\right)$$

$$= \Theta(n^2)$$

The master method

The master method applies to recurrences of the form

$$T(n) = aT(n/b) + f(n),$$

where $a \geq 1, b > 1$, and f is asymptotically positive.

The master method

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

The master method

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

The master method

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

The master method

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

The master method

Example

- $T(n) = 9T(n/3) + n$
  We have $a = 9, b = 3, f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_3 9} = n^2$. Since $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply **case 1**. The solution is $T(n) = \Theta(n^2)$.
- $T(n) = T(2n/3) + 1$
  $a = 1, b = 3/2, f(n) = 1, f(n) = \Theta(n^{\log_b a}) = \Theta(1)$.
  Case 2 applies, $T(n) = \Theta(\lg n)$.

The master method

Example

- $T(n) = 9T(n/3) + n$
  We have $a = 9, b = 3, f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_3 9} = n^2$. Since $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply **case 1**. The solution is $T(n) = \Theta(n^2)$.
- $T(n) = T(2n/3) + 1$
  $a = 1, b = 3/2, f(n) = 1, f(n) = \Theta(n^{\log_b a}) = \Theta(1)$.
  Case 2 applies, $T(n) = \Theta(\lg n)$.

The master method

Example

- $T(n) = 9T(n/3) + n$
  We have $a = 9, b = 3, f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_3 9} = n^2$. Since $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply **case 1**. The solution is $T(n) = \Theta(n^2)$.
- $T(n) = T(2n/3) + 1$
  $a = 1, b = 3/2, f(n) = 1, f(n) = \Theta(n^{\log_b a}) = \Theta(1)$.
  Case 2 applies, $T(n) = \Theta(\lg n)$.

The master method

Example

- $T(n) = 9T(n/3) + n$
  We have $a = 9, b = 3, f(n) = n$, and thus we have that $n^{\log_b a} = n^{\log_3 9} = n^2$. Since $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply **case 1**. The solution is $T(n) = \Theta(n^2)$.
- $T(n) = T(2n/3) + 1$
  $a = 1, b = 3/2, f(n) = 1, f(n) = \Theta(n^{\log_b a}) = \Theta(1)$.
  Case 2 applies, $T(n) = \Theta(\lg n)$.

The master method

Example

- $T(n) = 3T(n/4) + n\lg n$
  $a = 3, b = 4, f(n) = n\lg n, f(n) = \Omega(n^{\log_4 3+\epsilon})$, where $\epsilon \approx 0.2$. For sufficiently large n,
  $af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n\lg n$ for $c = 3/4$.
  By **case 3**, $T(n) = \Theta(n\lg n)$.

The master method

Example

- $T(n) = 3T(n/4) + n \lg n$
  $a = 3, b = 4, f(n) = n \lg n, f(n) = \Omega(n^{\log_4 3+\epsilon})$, where $\epsilon \approx 0.2$. For sufficiently large n,
  $af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n$ for $c = 3/4$.
  By **case 3**, $T(n) = \Theta(n \lg n)$.

The master method

Example

- $T(n) = 3T(n/4) + n \lg n$
  $a = 3, b = 4, f(n) = n \lg n, f(n) = \Omega(n^{\log_4 3+\epsilon})$, where $\epsilon \approx 0.2$. For sufficiently large n,
  $af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n$ for $c = 3/4$.
  By **case 3**, $T(n) = \Theta(n \lg n)$.

The master method

Is master method omnipotent?

Master Theorem fails in the following cases:

- When $f(n)$ is smaller than $n^{\log_b a}$ but not **polynomially** smaller. This is a gap between cases 1 and 2.
- When $f(n)$ is larger than $n^{\log_b a}$ but not **polynomially** larger. This is a gap between cases 2 and 3.
- When the regularity condition in case 3 fails to hold.

The master method

Is master method omnipotent?

Master Theorem fails in the following cases:

- When $f(n)$ is smaller than $n^{\log_b a}$ but not **polynomially** smaller. This is a gap between cases 1 and 2.
- When $f(n)$ is larger than $n^{\log_b a}$ but not **polynomially** larger. This is a gap between cases 2 and 3.
- When the regularity condition in case 3 fails to hold.

The master method

Is master method omnipotent?

Master Theorem fails in the following cases:

- When $f(n)$ is smaller than $n^{\log_b a}$ but not **polynomially** smaller. This is a gap between cases 1 and 2.
- When $f(n)$ is larger than $n^{\log_b a}$ but not **polynomially** larger. This is a gap between cases 2 and 3.
- When the regularity condition in case 3 fails to hold.

The master method

Is master method omnipotent?

Master Theorem fails in the following cases:

- When $f(n)$ is smaller than $n^{\log_b a}$ but not **polynomially** smaller. This is a gap between cases 1 and 2.
- When $f(n)$ is larger than $n^{\log_b a}$ but not **polynomially** larger. This is a gap between cases 2 and 3.
- When the regularity condition in case 3 fails to hold.

The master method

Example

$$T(n) = 2T(n/2) + n\lg n$$

$a = 2, b = 2, f(n) = n\lg n$, and $n^{\log_b a} = n$.
$f(n) = n\lg n$ is asymptotically larger than n, but not **polynomially** larger. The ratio $f(n)/n = \lg n$ is asymptotically less than n^{ϵ} for any positive constant ϵ.

The master method

知道就行

A more general method

In 1998, Mohamad Akra and Louay Bazzi presented a more general master method:

$$T(n) = \sum_{i=1}^{k} a_i T(\lfloor n/b_i \rfloor) + f(n)$$

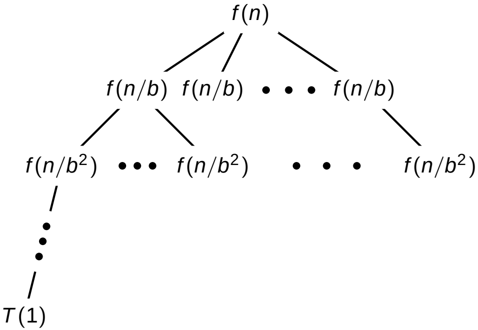
The master method

A more general method

This method would work on a recurrence such as $T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + O(n)$. We first find the value of p such that $\sum_{i=1}^{p} a_i b_i^{-p} = 1$. The solution to the recurrence is then

$$T(n) = \Theta(n^p) + \Theta(n^p \int_{n'}^{n} \frac{f(x)}{x^{p+1}} dx)$$

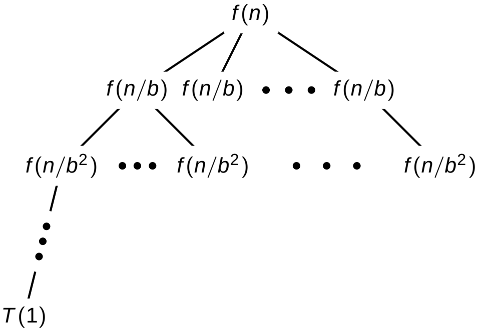
Idea of master theorem



Number of leaves

$$a^h = a^{\log_b n} = n^{\log_b a}$$

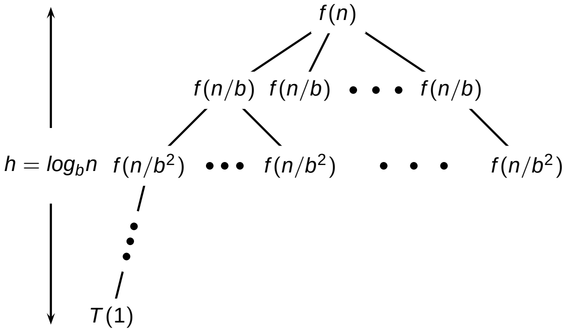
Idea of master theorem



Number of leaves

$$a^h = a^{\log_b n} = n^{\log_b a}$$

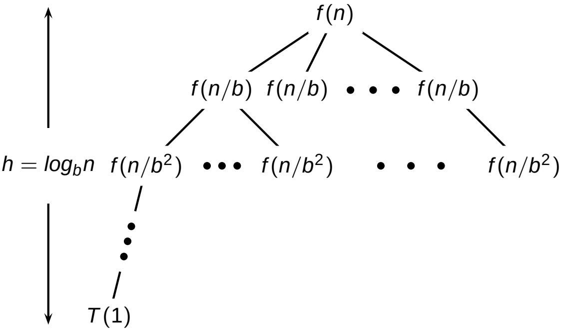
Idea of master theorem



Number of leaves

$$a^h = a^{\log_b n} = n^{\log_b a}$$

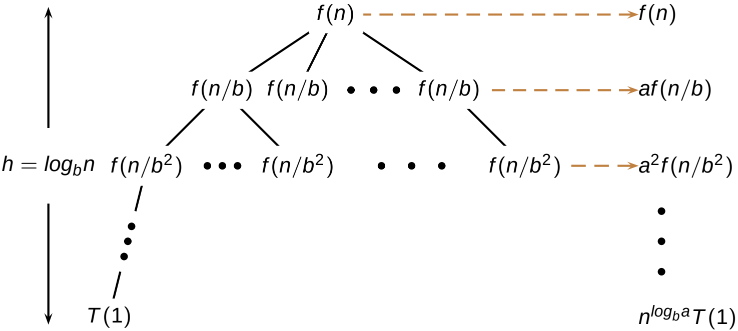
Idea of master theorem



Number of leaves

$$a^h = a^{\log_b n} = n^{\log_b a}$$

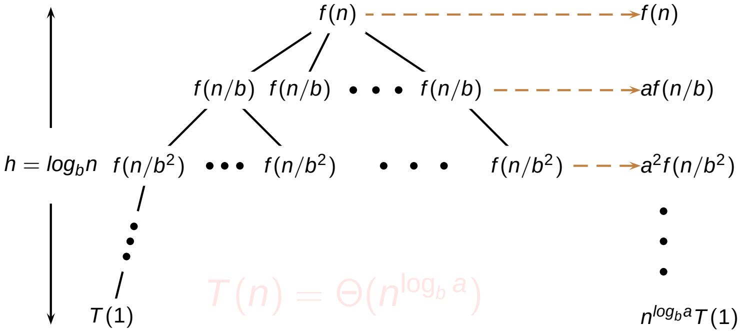
Idea of master theorem



Number of leaves

$$a^h = a^{\log_b n} = n^{\log_b a}$$

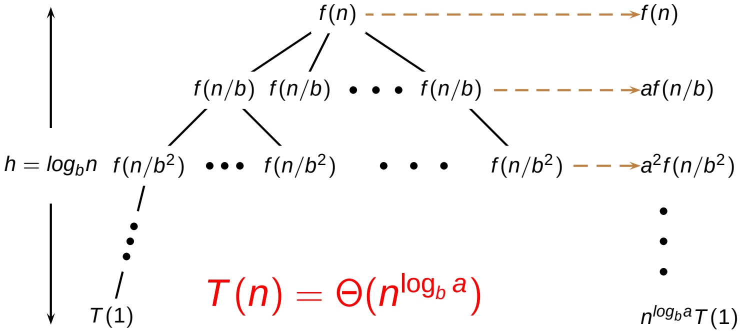
Idea of master theorem



Case 1

The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Idea of master theorem

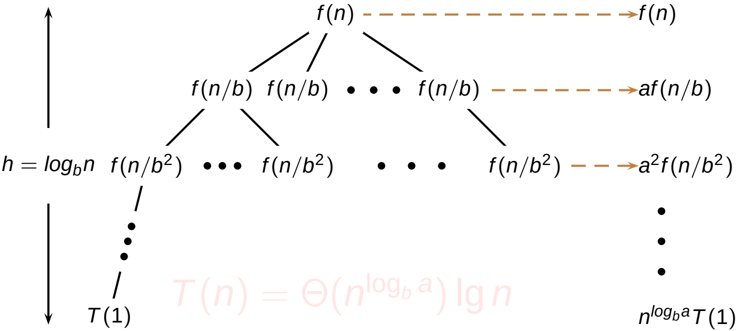


$$T(n) = \Theta(n^{\log_b a})$$

Case 1

The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

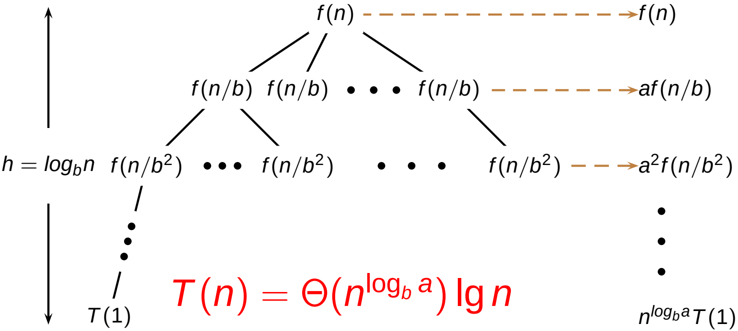
Idea of master theorem



Case 2

The weight is approximately the same on each of the $\log_b n$ levels.

Idea of master theorem

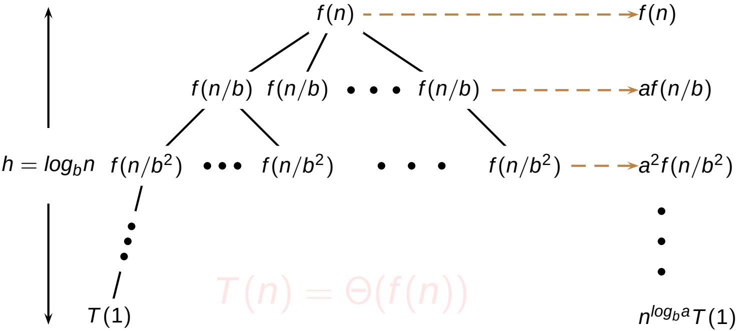


$$T(n) = \Theta(n^{\log_b a}) \lg n$$

Case 2

The weight is approximately the same on each of the $\log_b n$ levels.

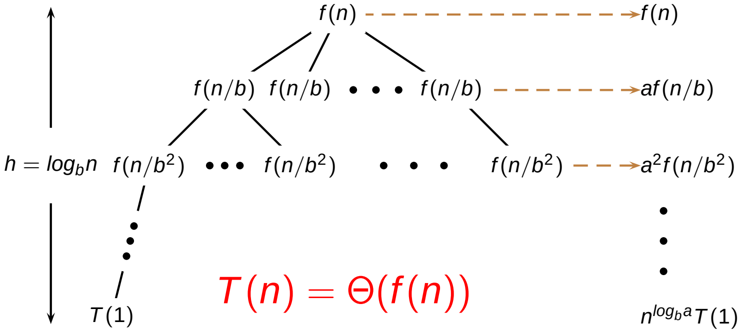
Idea of master theorem



Case 3

The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Idea of master theorem

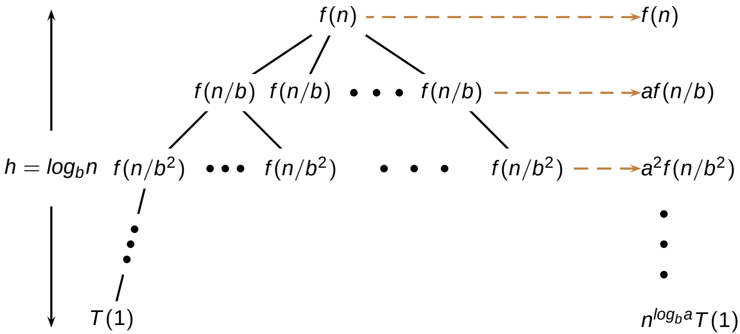


$$T(n) = \Theta(f(n))$$

Case 3

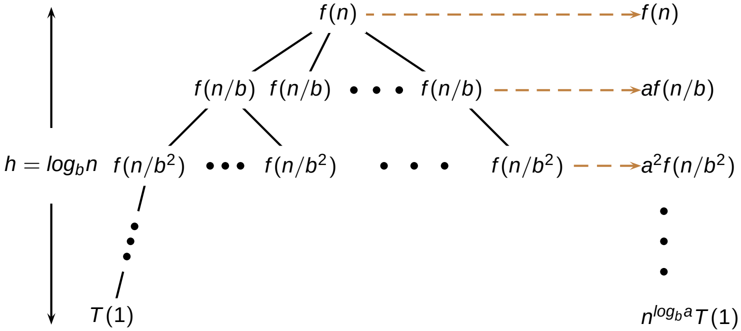
The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Proof of master theorem



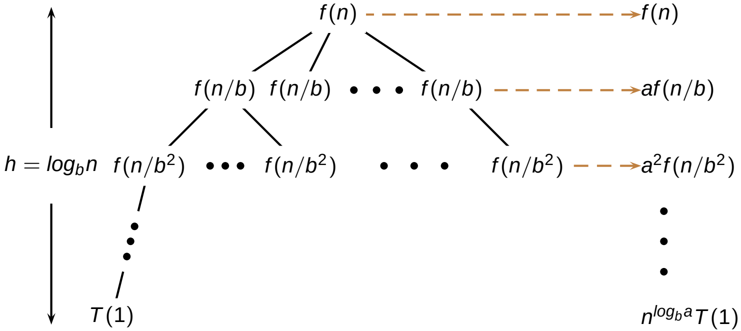
$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

Proof of master theorem



$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

Proof of master theorem



$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

Since $f(n/b^j) = O((n/b^j)^{\log_b a - \epsilon})$, then

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$= O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{ab^{\epsilon}}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^{\epsilon})^j$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = n^{\log_b a - \epsilon}\left(\frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}\right)$$

$$= n^{\log_b a - \epsilon}\left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right)$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = n^{\log_b a - \epsilon}\left(\frac{b^{\epsilon \log_b n} - 1}{b^{\epsilon} - 1}\right)$$

$$= n^{\log_b a - \epsilon}\left(\frac{n^{\epsilon} - 1}{b^{\epsilon} - 1}\right)$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$g(n) = O\left(n^{\log_b a - \epsilon}\left(\frac{n^{\epsilon} - 1}{b^{\epsilon} - 1}\right)\right)$$

$$= O\left(n^{\log_b a - \epsilon} n^{\epsilon}\right)$$

$$= O(n^{\log_b a})$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$g(n) = O\left(n^{\log_b a - \epsilon}\left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right)\right)$$

$$= O\left(n^{\log_b a - \epsilon} n^\epsilon\right)$$

$$= O(n^{\log_b a})$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$\begin{aligned} g(n) &= O\left(n^{\log_b a - \epsilon}\left(\frac{n^{\epsilon} - 1}{b^{\epsilon} - 1}\right)\right) \\ &= O\left(n^{\log_b a - \epsilon} n^{\epsilon}\right) \\ &= O(n^{\log_b a}) \end{aligned}$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$= \Theta(n^{\log_b a}) + g(n)$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a})$$

$$= \Theta(n^{\log_b a})$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a-\epsilon})$

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

$$= \Theta(n^{\log_b a}) + g(n)$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a})$$

$$= \Theta(n^{\log_b a})$$

Proof of master theorem

Case 1: $f(n) = O(n^{\log_b a-\epsilon})$

$$
\begin{aligned}
T(n) &= \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \\
&= \Theta(n^{\log_b a}) + g(n) \\
&= \Theta(n^{\log_b a}) + O(n^{\log_b a}) \\
&= \Theta(n^{\log_b a})
\end{aligned}
$$

Proof of master theorem

Case 2: $f(n) = \Theta(n^{\log_b a})$

We have $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$, then

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

$$= \Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

Proof of master theorem

Case 2: $f(n) = \Theta(n^{\log_b a})$

We have $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$, then

$$\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} = n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n-1} 1$$

$$= n^{\log_b a} \log_b n$$

Proof of master theorem

Case 2: $f(n) = \Theta(n^{\log_b a})$

We have $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$, then

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} = n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1$$

$$= n^{\log_b a} \log_b n$$

Proof of master theorem

Case 2: $f(n) = \Theta(n^{\log_b a})$

We have $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$, then

$$
\begin{aligned}
\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} &= n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j \\
&= n^{\log_b a} \sum_{j=0}^{\log_b n-1} 1 \\
&= n^{\log_b a} \log_b n
\end{aligned}
$$

Proof of master theorem

Case 2: $f(n) = \Theta(n^{\log_b a})$

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

$$= \Theta(n^{\log_b a}) + g(n)$$

$$= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log_b n)$$

$$= \Theta(n^{\log_b a} \lg n)$$

Proof of master theorem

Case 2: $f(n) = \Theta(n^{\log_b a})$

$$
\begin{aligned}
T(n) &= \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \\
&= \Theta(n^{\log_b a}) + g(n) \\
&= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log_b n) \\
&= \Theta(n^{\log_b a} \lg n)
\end{aligned}
$$

Proof of master theorem

Case 3: $f(n) = \Omega(n^{\log_b a+\epsilon})$

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

$$\leq \sum_{j=0}^{\log_b n-1} c^j f(n) \quad (\text{By } af(n/b) \leq cf(n))$$

$$\leq f(n) \sum_{j=0}^{\infty} c^j = f(n)\left(\frac{1}{1-c}\right)$$

Proof of master theorem

Case 3: $f(n) = \Omega(n^{\log_b a+\epsilon})$

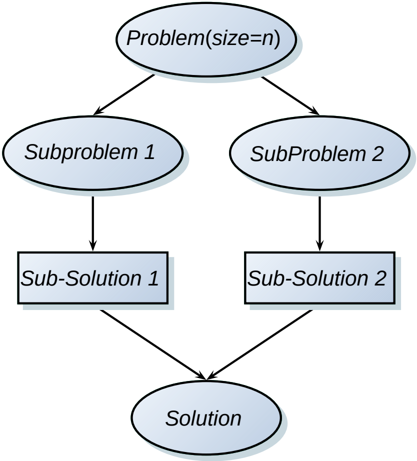
$$
\begin{aligned}
T(n) &= \Theta(n^{\log_b a}) + g(n) \\
&= \Theta(n^{\log_b a}) + \Theta(f(n)) \\
&= \Theta(f(n))
\end{aligned}
$$

Proof of master theorem

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a}) + g(n) \\ &= \Theta(n^{\log_b a}) + \Theta(f(n)) \\ &= \Theta(f(n)) \end{aligned}$$

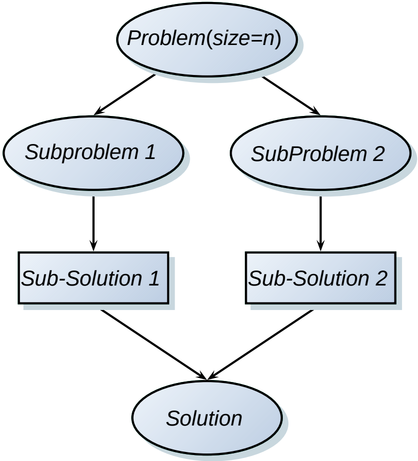
What is Divide and Conquer?



Design paradigm

1. **Divide** the problem (instance) into subproblems.
2. **Conquer** subproblems by solving them recursively.
3. **Combine** subproblems solutions.

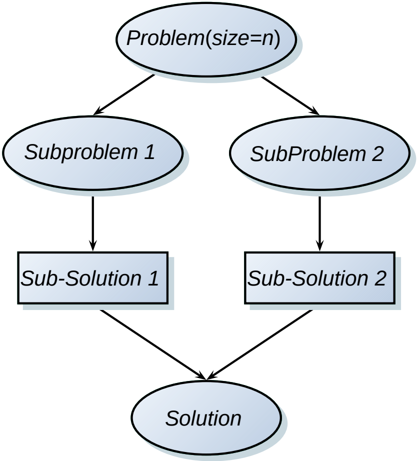
What is Divide and Conquer?



Design paradigm

1. **Divide** the problem (instance) into subproblems.
2. **Conquer** subproblems by solving them recursively.
3. **Combine** subproblems solutions.

What is Divide and Conquer?



Design paradigm

1. **Divide** the problem (instance) into subproblems.
2. **Conquer** subproblems by solving them recursively.
3. **Combine** subproblems solutions.

MergeSort

Design paradigm

1. **Divide:** Trivial! We get two $n/2$-size subarrays.
2. **Conquer:** Recursively sort the two subarrays.
3. **Combine:** Linear-time merge.

MergeSort

Design paradigm

1. **Divide:** Trivial! We get two $n/2$-size subarrays.
2. **Conquer:** Recursively sort the two subarrays.
3. **Combine:** Linear-time merge.

MergeSort

Design paradigm

1. **Divide:** Trivial! We get two $n/2$-size subarrays.
2. **Conquer:** Recursively sort the two subarrays.
3. **Combine:** Linear-time merge.

MergeSort

```
MERGE-SORT(A,p,r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A,p,q)
    MERGE-SORT(A,q+1,r)
    MERGE(A,p,q,r)
```

Merging two sorted arrays

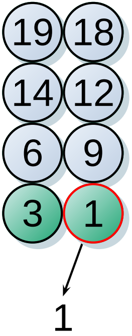
Example

$$19, 3, 6, 14, 1, 9, 18, 12$$

Merging two sorted arrays

Example

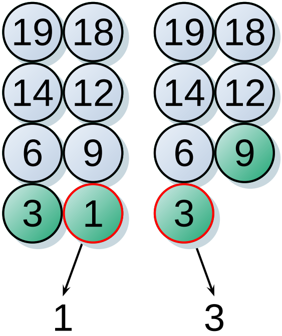
19, 3, 6, 14, 1, 9, 18, 12



Merging two sorted arrays

Example

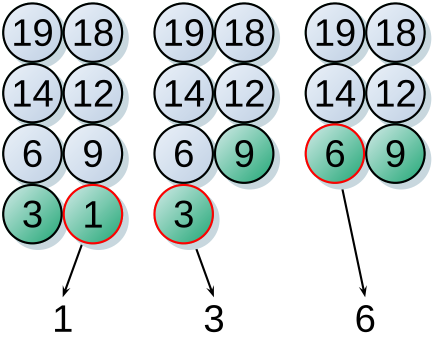
19, 3, 6, 14, 1, 9, 18, 12



Merging two sorted arrays

Example

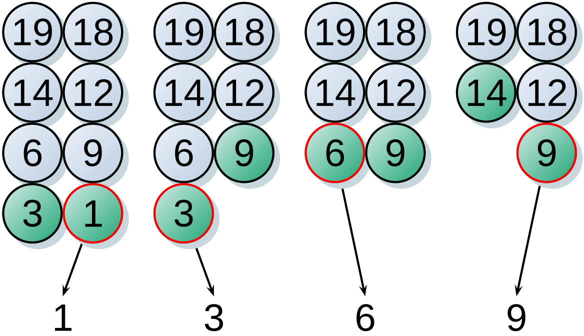
19, 3, 6, 14, 1, 9, 18, 12



Merging two sorted arrays

Example

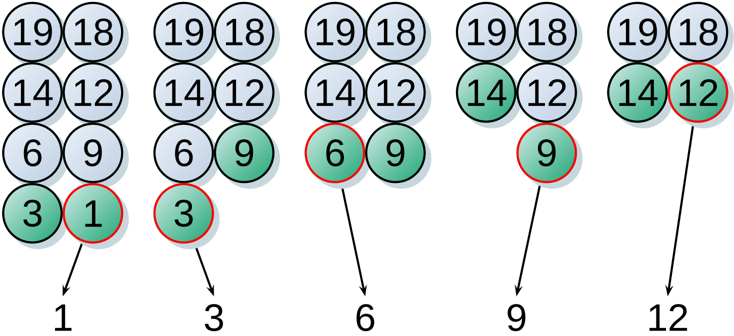
19, 3, 6, 14, 1, 9, 18, 12



Merging two sorted arrays

Example

19, 3, 6, 14, 1, 9, 18, 12



Analysis of MergeSort

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Two subarrays = $2T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

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

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(n)$

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

Analysis of MergeSort

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Two subarrays = $2T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

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

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(n)$

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

Analysis of MergeSort

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Two subarrays = $2T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

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

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(n)$

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

Analysis of MergeSort

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Two subarrays = $2T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

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

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(n)$

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

Analysis of MergeSort

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Two subarrays = $2T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

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

Master theorem, Case 2: $\Theta(n^{log_b a}) = \Theta(n)$

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

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1	4	7	9	10	15	20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1	4	7	9	10	15	20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1	4	7	9	10	15	20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1	4	7	9	10	15	20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1	4	7	9	10	15	20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1	4	7	9	10	15	20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1 4 7 9 $\boxed{10}$ 15 20

Binary search

Design paradigm

1. **Divide:** Trivial! Check the middle element.
2. **Conquer:** Recursively search one subarray.
3. **Combine:** Return the position.

Example

Search 10 in the following array:

1 4 7 9 $\boxed{10}$ 15 20

Analysis of Binary search

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Only search one subarray = $T(n/2)$.
3. **Combine:** $\Theta(1)$.

$$T(n) = 1T(n/2) + \Theta(1)$$

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(1)$

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

Analysis of Binary search

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Only search one subarray = $T(n/2)$.
3. **Combine:** $\Theta(1)$.

$$T(n) = 1T(n/2) + \Theta(1)$$

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(1)$

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

Analysis of Binary search

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Only search one subarray = $T(n/2)$.
3. **Combine:** $\Theta(1)$.

$$T(n) = 1T(n/2) + \Theta(1)$$

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(1)$

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

Analysis of Binary search

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Only search one subarray = $T(n/2)$.
3. **Combine:** $\Theta(1)$.

$$T(n) = 1T(n/2) + \Theta(1)$$

Master theorem, Case 2: $\Theta(n^{log_b a}) = \Theta(1)$

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

Analysis of Binary search

Analysis paradigm

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** Only search one subarray = $T(n/2)$.
3. **Combine:** $\Theta(1)$.

$$T(n) = 1T(n/2) + \Theta(1)$$

Master theorem, Case 2: $\Theta(n^{\log_b a}) = \Theta(1)$

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

Fibonacci numbers

Naive recursive algorithm

$\text{F{\small IBONNACI}}(n)$

if $(n = 0)$ **return** 0 ;
if $(n = 1)$ **return** 1 ;
return $\text{F{\small IBONNACI}}(n-1)$
$\quad + \text{F{\small IBONNACI}}(n-2)$;

Fibonacci numbers

Naive recursive algorithm

$$T(n) = T(n-1) + T(n-2)$$

$$T(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

$$T(n) = \Omega(\phi^n), \phi = (1+\sqrt{5})/2$$

Fibonacci numbers

Naive recursive algorithm

$$T(n) = T(n-1) + T(n-2)$$

$$T(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

$$T(n) = \Omega(\phi^n), \phi = (1+\sqrt{5})/2$$

Fibonacci numbers

Naive recursive squaring

$F_n = \phi^n/\sqrt{5}$ rounded to the nearest integer.

- $T(n) = T(n/2) + \Theta(1) \Longrightarrow T(n) = \Theta(\lg n)$.
- ***Unreliable!***

Bottom-up

- Compute $F_0, F_1, \ldots, F_n$.
- $T(n) = \Theta(n)$.

Fibonacci numbers

Naive recursive squaring

$F_n = \phi^n/\sqrt{5}$ rounded to the nearest integer.

- $T(n) = T(n/2) + \Theta(1) \Longrightarrow T(n) = \Theta(\lg n)$.
- ***Unreliable!***

Bottom-up

- Compute $F_0, F_1, \ldots, F_n$.
- $T(n) = \Theta(n)$.

Fibonacci numbers

Naive recursive squaring

$F_n = \phi^n/\sqrt{5}$ rounded to the nearest integer.

- $T(n) = T(n/2) + \Theta(1) \Longrightarrow T(n) = \Theta(\lg n)$.
- ***Unreliable!***

Bottom-up

- Compute $F_0, F_1, \ldots, F_n$.
- $T(n) = \Theta(n)$.

Fibonacci numbers

Naive recursive squaring

$F_n = \phi^n/\sqrt{5}$ rounded to the nearest integer.

- $T(n) = T(n/2) + \Theta(1) \implies T(n) = \Theta(\lg n)$.
- ***Unreliable!***

Bottom-up

- Compute $F_0, F_1, \ldots, F_n$.
- $T(n) = \Theta(n)$.

Fibonacci numbers

Naive recursive squaring

$F_n = \phi^n / \sqrt{5}$ rounded to the nearest integer.

- $T(n) = T(n/2) + \Theta(1) \Longrightarrow T(n) = \Theta(\lg n)$.
- ***Unreliable!***

Bottom-up

- Compute $F_0, F_1, \ldots, F_n$.
- $T(n) = \Theta(n)$.

Fibonacci numbers

Recursive squaring

Theorem:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

Fibonacci numbers

Proof.

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

Inductive step$(n \geq 2)$:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$\square$

Fibonacci numbers

Proof.

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

Inductive step($n \geq 2$) :

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$\square$

Fibonacci numbers

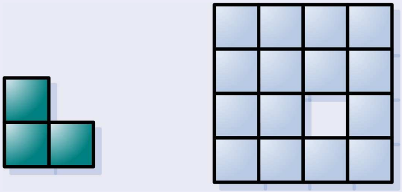
Recursive squaring

1. **Divide:** $n/2$
2. **Conquer:** Calculate $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2}$
3. **Combine:** $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2}$

$T(n) = T(n/2) + \Theta(1) \Longrightarrow T(n) = \Theta(\lg n)$.

Exercises

1. 用分治法分析两个长度均为n的大整数乘法
2. Triomino拼图，用一个L型瓦片（含三个方块）覆盖一个缺少了一个方块的$2^n \times 2^n$的棋盘。设计此问题的分治算法并分析复杂度。



Big integers multiplication

A simple example

$$
\begin{aligned}
23 * 14 = (2 \cdot 10^1 + 3 \cdot 10^0) \\
* (1 \cdot 10^1 + 4 \cdot 10^0) \\
= (2 * 1)10^2 + (3 * 1 + 2 * 4)10^1 \\
+ (3 * 4)10^0
\end{aligned}
$$

$$
\begin{aligned}
(3 * 1 + 2 * 4) = (2 + 3) * (1 + 4) \\
- (2 * 1) - (3 * 4)
\end{aligned}
$$

Big integers multiplication

A simple example

$$
\begin{aligned}
23 * 14 &= (2 \cdot 10^1 + 3 \cdot 10^0) \\
&\quad * (1 \cdot 10^1 + 4 \cdot 10^0) \\
&= (2 * 1)10^2 + (3 * 1 + 2 * 4)10^1 \\
&\quad + (3 * 4)10^0 \\
(3 * 1 + 2 * 4) &= (2 + 3) * (1 + 4) \\
&\quad - (2 * 1) - (3 * 4)
\end{aligned}
$$

Big integers multiplication

A general example

$$
\begin{aligned}
c = a * b &= (a_1 10^{n/2} + a_0) * (b_1 10^{n/2} + b_0) \\
&= (a_1 * b_1) 10^n + (a_1 * b_0 + a_0 * b_1) 10^{n/2} \\
&\quad + (a_0 * b_0) \\
&= c_2 10^n + c_1 10^{n/2} + c_0
\end{aligned}
$$

$$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$$

Big integers multiplication

A general example

$$
\begin{aligned}
c = a * b &= (a_1 10^{n/2} + a_0) * (b_1 10^{n/2} + b_0) \\
&= (a_1 * b_1) 10^n + (a_1 * b_0 + a_0 * b_1) 10^{n/2} \\
&\qquad + (a_0 * b_0) \\
&= c_2 10^n + c_1 10^{n/2} + c_0 \\
c_1 &= (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)
\end{aligned}
$$

Big integers multiplication

Analysis

1. **Divide:** $D(n) = \Theta(1)$.
2. **Conquer:** $3T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

Master theorem

$$T(n) = 3T(n/2) + \Theta(n)$$

Case 1: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{1.585})$

Matrix multiplication

Matrix multiplication

- **Input:** $A = [a_{ij}], B = [b_{ij}]$.
- **Output:** $C = [c_{ij}] = A \cdot B \quad i, j = 1, 2, \cdots, n$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ c_{21} & \cdots & c_{2n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ b_{21} & \cdots & b_{2n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

Matrix multiplication

$\textsc{Matrix-Multiply}(A, B)$

```
n = A.rows
let C be an n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

$T(n) = \Theta(n^3)$

Matrix multiplication

MATRIX-MULTIPLY(A, B)

$n = A.rows$
let C be an $n \times n$ matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        $c_{ij} = 0$
        for $k = 1$ **to** n
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

$T(n) = \Theta(n^3)$

Matrix multiplication

Idea of Divide and Conquer

Divide a $n \times n$ matrix multiplication into 2×2 $(n/2) \times (n/2)$ submatrix multiplication.

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\begin{aligned} r &= ae + bg, & s &= af + bh, \\ t &= ce + dg, & u &= cf + dh. \end{aligned}$$

Matrix multiplication

Analysis

$$T(n) = T(n/2) +$$

Master theorem

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) = O(n^{3-\epsilon})$$

Case 1: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

No improvment !?

Matrix multiplication

Analysis

$$T(n) = 8T(n/2) +$$

Master theorem

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) = O(n^{3-\epsilon})$$

$$\text{Case 1:} \quad T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$$

No improvement !?

Matrix multiplication

Analysis

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Master theorem

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) = O(n^{3-\epsilon})$$

Case 1: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

No improvment !?

Matrix multiplication

Analysis

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Master theorem

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) = O(n^{3-\epsilon})$$

Case 1: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

No improvment !?

Matrix multiplication

Analysis

$$T(n) = 8T(n/2) + \Theta(n^2)$$

Master theorem

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) = O(n^{3-\epsilon})$$

Case 1: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

No improvment !?

Matrix multiplication

Strassen's idea

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs

Matrix multiplication

Strassen's idea

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs

Matrix multiplication

Strassen's idea

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
s &= P_1 + P_2 \\
t &= P_3 + P_4 \\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

7 mults, 18 adds/subs

Matrix multiplication

Strassen's idea

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad - bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$

Matrix multiplication

Strassen's Divide and Conquer

1. **Divide:** Partition A and B into $(n/2) \times (n/2)$ submatrices.
2. **Conquer:** Perform 7 multiplications of $(n/2) \times (n/2)$ submatrices recursively.
3. **Combine:** Form C using $+$ and $-$ on $(n/2) \times (n/2)$ submatrices.

Matrix multiplication

Master theorem

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) \approx O(n^{2.81-\epsilon})$$

Case 1: $T(n) = \Theta(n^{\log_b a}) \approx \Theta(n^{2.81})$

Matrix multiplication

Master theorem

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon}) \approx O(n^{2.81-\epsilon})$$

Case 1: $T(n) = \Theta(n^{\log_b a}) \approx \Theta(n^{2.81})$

Matrix multiplication

Discussion

- The "crossover point" of Strassen's algorithm on various systems ranging from $n = 400$ to $n = 2150$.
- In 1971, Hopcroft and Kerr proved that 7 multiplications is the best for 2×2 partition.
- **Best to date** (of theoretical interest only): $\Theta(n^{2.376\ldots})$.

Matrix multiplication

Discussion

- Strassen's algorithm is often not the method of choice for matrix multiplication:
  - The constant factor hidden in the running time is larger than the simple procedure.
  - For sparse matrices, we have better algorithms.
  - Strassen's algorithm is not quite numerically stable.
  - It uses too much memories.

Finding the closest pair of points

Problem

Given a set P of $n \geq 2$ points, we now consider the problem of finding the closest pair of points in the set.
Closest refers to the usual euclidean distance: the distance between points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Brute-force algorithm

Simply look at all $\Theta(n^2)$ pairs of points.

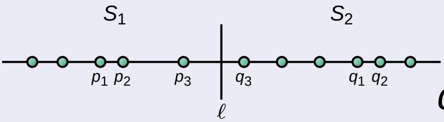
Finding the closest pair of points

Divide-and-conquer algorithm

1. **Divide:** Find a vertical line l that bisects the point set P into two sets $|P_L| = \lceil |P|/2 \rceil$, $|P_R| = \lfloor |P|/2 \rfloor$.
2. **Conquer:** Find the closest pairs of points in P_L and P_R.
3. **Combine:** How?

Finding the closest pair of points

One dimension example



$$S_1 = \{x \in S \mid x \leq m\}$$
$$S_2 = \{x \in S \mid x > m\}$$

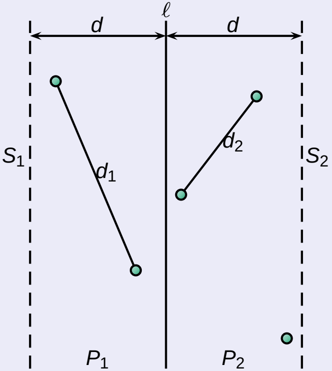
$$d = \min\{|p_1 - p_2|,\ |q_1 - q_2|\}$$
$$d_{\min} = \min\{d, |p_3 - q_3|\}$$

Finding the closest pair of points

Two dimension example

$$S_1 = \{p \in S \mid x(p) \leq m\}$$
$$S_2 = \{p \in S \mid x(p) > m\}$$

$$d = \min\{d_1, d_2\}$$



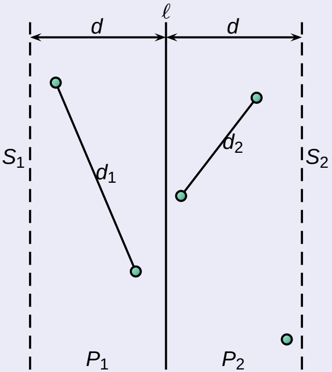
Finding the closest pair of points

Two dimension example

$S_1 = \{p \in S \mid x(p) \leq m\}$
$S_2 = \{p \in S \mid x(p) > m\}$

$d = \min\{d_1, d_2\}$

$d_{\min} = \min\{d,$ closest pair in ℓ neighborhood $\}$



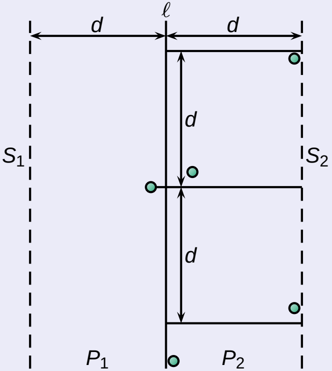
Finding the closest pair of points

Two dimension example

$$S_1 = \{p \in S \mid x(p) \leq m\}$$
$$S_2 = \{p \in S \mid x(p) > m\}$$

$$d = \min\{d_1, d_2\}$$

$d_{\min} = \min\{d,$ closest pair in ℓ neighborhood $\}$

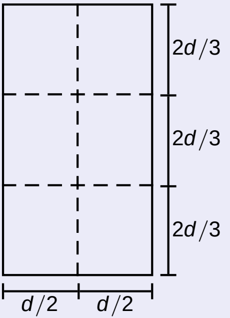


Finding the closest pair of points

How many points in the region?

6 is the maximum!

$$(x(u)-x(v))^2+(y(u)-y(v))^2$$
$$\leq (d/2)^2+(2d/3)^2$$
$$=25d^2/36$$

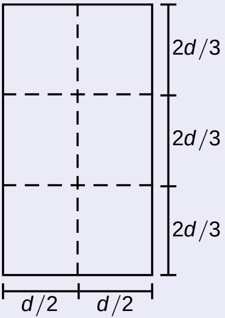


Finding the closest pair of points

How many points in the region?

6 is the maximum!

$$(x(u)-x(v))^2+(y(u)-y(v))^2 \leq (d/2)^2+(2d/3)^2 = 25d^2/36$$

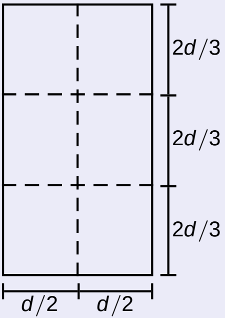


Finding the closest pair of points

How many points in the region?

6 is the maximum!

$$
\begin{aligned}
&(x(u) - x(v))^2 + (y(u) - y(v))^2 \\
&\leq (d/2)^2 + (2d/3)^2 \\
&= 25d^2/36 \\
&\leq d^2
\end{aligned}
$$

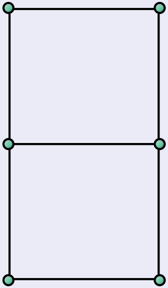


Finding the closest pair of points

How many points in the region?

6 is the maximum!

$$(x(u) - x(v))^2 + (y(u) - y(v))^2 \leq (d/2)^2 + (2d/3)^2 = 25d^2/36 \leq d^2$$



Finding the closest pair of points

Analysis

1. **Divide:** $D(n) = \Theta(n)$.
2. **Conquer:** $2T(n/2)$.
3. **Combine:** $C(n) = \Theta(n)$.

Master theorem

$$T(n) = 2T(n/2) + \Theta(n)$$

Case 2: $T(n) = \Theta(n \lg n)$

Hiring Problem

Hiring Problem

Suppose that you need to hire a new office assistant from n candidates. After the interview, you must decide whether to hire him (her) or not. If you hire somebody, you have to pay some money.

Hiring Problem

HIRE-ASSISTANT(n)

$best = 0$
for $i = 1$ **to** n
    interview candidate i
    if candidate i is better than candidate $best$
        $best = i$
        hire candidate i

Hiring Problem

Worst-case analysis

- We actually hire every candidate that we interview.
- If every hiring cost is c_h, the total hiring cost is $O(nc_h)$.

Hiring Problem

Probabilistic analysis

- ***Why?*** More practical!
- In order to perform a probabilistic analysis, we must use knowledge of, or make assumption about, the distribution of the inputs.
- We must have greater control over the order in which we interview the candidates.

Review of probability knowledge

Expectation

The ***expected value(expectation)*** of a discrete random variable X is

$$E[X] = \sum_{x} x Pr\{X = x\}.$$

Its variance is

$$V[X] = E[X - E[X]]^2 = E[X^2] - [E[X]]^2.$$

Review of probability knowledge

Expectation

Some properties:

$$E[X+Y] = E[X] + E[Y] \qquad E[aX] = aE[X]$$

$$E[XY] = E[X]E[Y] \qquad V[aX] = a^2V[X]$$

$$V[X+Y] = V[X] + V[Y]$$

Review of probability knowledge

Conditional probability

The ***conditional probability*** of an event A given that another event B occurs is defined to be

$$Pr\{A|B\} = \frac{Pr\{A \cap B\}}{Pr\{B\}}$$

Hence we have $Pr\{A \cap B\} = Pr\{A|B\}Pr\{B\}$

Indicator Random Variable

Definition

Give a sample space S and an event A, the ***indicator random variable*** $I\{A\}$ associated with event A is defined as

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

Lemma 5.1

Given a sample space S and an event A in the sample space S , let $X_A = I\{A\}$. Then $E[X_A] = Pr\{A\}$.

Indicator Random Variable

Definition

Give a sample space S and an event A, the ***indicator random variable*** $I\{A\}$ associated with event A is defined as

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

Lemma 5.1

Given a sample space S and an event A in the sample space S , let $X_A = I\{A\}$. Then $E[X_A] = Pr\{A\}$.

Indicator Random Variable

Flip a coin

$$E[X_H] = E[I\{Y = H\}] = Pr\{Y = H\} = 1/2.$$

Flip n coins

$$E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} 1/2 = n/2$$

Indicator Random Variable

Flip a coin

$$
\begin{aligned}
E[X_H] &= E[I\{Y = H\}] \\
&= Pr\{Y = H\} = 1/2.
\end{aligned}
$$

Flip n coins

$$
\begin{aligned}
E[X] = E[\sum_{i=1}^{n} X_i] &= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} 1/2 = n/2
\end{aligned}
$$

Analysis of the hiring problem

Definition

Let X be the number of hired persons and let X_i be

$$X_i = I\{\text{candidate } i \text{ is hired}\} = \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{if candidate } i \text{ is not hired} \end{cases}$$

and

$$X = X_1 + X_2 + \cdots + X_n$$

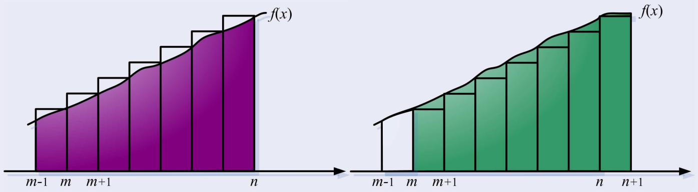
Analysis of the hiring problem

Hired expectation

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} 1/i
\end{aligned}
$$

Analysis of the hiring problem

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$



Analysis of the hiring problem

Hired expectation

When $f(k)$ is a monotonically decreasing function:

$$\int_{m}^{n+1} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx$$

$$\ln(n+1) = \int_{1}^{n+1} \frac{dx}{x} \leq \sum_{k=1}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} + 1 = \ln n + 1$$

Analysis of the hiring problem

Hired expectation

When $f(k)$ is a monotonically decreasing function:

$$\int_{m}^{n+1} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx$$

$$\ln(n+1) = \int_{1}^{n+1} \frac{dx}{x} \leq \sum_{k=1}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} + 1 = \ln n + 1$$

Analysis of the hiring problem

Hired expectation

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} 1/i \\
&= \ln n + O(1)
\end{aligned}
$$

Analysis of the hiring problem

Lemma 5.2

Assuming that the candidates are presented in a random order, algorithm HIRE-ASSISTANT has a total hiring cost of $O(c_h \ln n)$.

Randomized algorithms

RANDOMIZED-HIRE-ASSISTANT(n)

```
randomly permute the list of candidates
best = 0
for i = 1 to n
        interview candidate i
        if candidate i is better than
                candidate best
            best = i
            hire candidate i
```

Randomized algorithms

Lemma 5.3

The expected hiring cost of the procedure RANDOMIZED-HIRE-ASSISTANT is $O(c_h \ln n)$.

Randomly permuting arrays

$\textsc{Permute-By-Sorting}(A)$

```
n = A.length
for i = 1 to n
        P[i] = RANDOM(1, n^3)
        sort A, using P as sort keys.
        return A
```

Example

Let $A = (1, 2, 3, 4)$ and choose random priorities $P = (10, 2, 57, 21)$, then the new A is $(2, 1, 4, 3)$.

Uniform Random Permutation

Lemma 5.4

Procedure PERMUTE-BY-SORTING produces ***a uniform random permutation*** of the input, assuming that all priorities are distinct.

Uniform Random Permutation

Proof.

$$
\begin{aligned}
& Pr\{X_1 \cap X_2 \cap \cdots \cap X_{n-1} \cap X_n\} \\
= & Pr\{X_1\} \cdot Pr\{X_2 | X_1\} \cdots Pr\{X_n | X_{n-1} \cap \cdots \cap X_1\} \\
= & \left(\frac{1}{n}\right)\left(\frac{1}{n-1}\right) \cdots \left(\frac{1}{2}\right)\left(\frac{1}{1}\right) \\
= & \frac{1}{n!}
\end{aligned}
$$

$\square$

A Better Random Permutation

RANDOMIZE-IN-PLACE(A)

```
1   n = A.length
2   for i = 1 to n
3       swap A[i] ↔ A[RANDOM(i, n)]
```

Lemma 5.5

Procedure RANDOMIZE-IN-PLACE computes a uniform random permutation.

A Better Random Permutation

Proof.

We use the following loop invariant:
Just prior to the ith iteration of the **for** loop of lines $2-3$, for each possible $i-1$-permutation, the subarray $A[1..i-1]$ contains this $i-1$-permutation with probability $(n-i+1)!/n!$. $\square$

The on-line hiring problem

ON-LINE-MAXIMUM(k, n)

```
bestscore = −∞
for i = 1 to k
        if score(i) > bestscore
                bestscore = score(i)
for i = k + 1 to n
        if score(i) > bestscore
                return i
return n
```

The on-line hiring problem

Analysis

$$Pr\{S\} = \sum_{i=k+1}^{n} Pr\{S_i\}$$

In order to succeed when the best-qualified applicant is the ith one, two things must happen.

The on-line hiring problem

Analysis

- The best-qualified applicant must be in position i, an event which we denote by B_i.
- The algorithm must ***not*** select any of the applicants in positions $k+1$ through $i-1$. We use O_i to denote the event.

The on-line hiring problem

Analysis

$$Pr\{S_i\} = Pr\{Bi \cap O_i\} = Pr\{B_i\}Pr\{O_i\}.$$

$$Pr\{S\} = \sum_{i=k+1}^{n} Pr\{S_i\} = \sum_{i=k+1}^{n} \frac{k}{n(i-1)}$$

$$= \frac{k}{n}\sum_{i=k+1}^{n} \frac{1}{i-1} = \frac{k}{n}\sum_{i=k}^{n-1} \frac{1}{i}$$

The on-line hiring problem

Analysis

We have

$$\int_{k}^{n} \frac{1}{x}dx \leq \sum_{i=k}^{n-1} \frac{1}{i} \leq \int_{k-1}^{n-1} \frac{1}{x}dx$$

Such that

$$\frac{k}{n}(\ln n - \ln k) \leq Pr\{S\} \leq \frac{k}{n}(\ln(n-1) - \ln(k-1))$$

The on-line hiring problem

Analysis

We have

$$\int_{k}^{n} \frac{1}{x} dx \leq \sum_{i=k}^{n-1} \frac{1}{i} \leq \int_{k-1}^{n-1} \frac{1}{x} dx$$

Such that

$$\frac{k}{n}(\ln n - \ln k) \leq Pr\{S\} \leq \frac{k}{n}(\ln(n-1) - \ln(k-1))$$

The on-line hiring problem

Analysis

We get

$$\frac{d(\frac{k}{n}(\ln n - \ln k))}{dk} = \frac{1}{n}(\ln n - \ln k - 1).$$

When $\frac{1}{n}(\ln n - \ln k - 1) = 0$, $Pr\{S\}$ is maximized. Thus if we implement our strategy with $k = n/e$, we will succeed in hiring our best-qualified applicant with the probability at least $1/e$.

The on-line hiring problem

Analysis

We get

$$\frac{d(\frac{k}{n}(\ln n - \ln k))}{dk} = \frac{1}{n}(\ln n - \ln k - 1).$$

When $\frac{1}{n}(\ln n - \ln k - 1) = 0$, $Pr\{S\}$ is maximized. Thus if we implement our strategy with $k = n/e$, we will succeed in hiring our best-qualified applicant with the probability at least $1/e$.