Foundations

Bin Wang

School of Software
Tsinghua University

September 20, 2019

Outline

Staff

Teacher

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

TA

潘天翔(ptx11@mails.tsinghua.edu.cn)
李思宇(lisy14liz@163.com)

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://poj.org/`
- `http://en.wikipedia.org/`
- `http://www.github.com/`

Topics

Course Schedule

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

Policy

Grading Policy

- 考勤(10%)
- 平时作业(30%)
- 期末考试(60%)

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

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

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.

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

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

Insertion sort

Worst-cse

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

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

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.

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

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.

$$T(n) = (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

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 $n \geq n_0$,

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

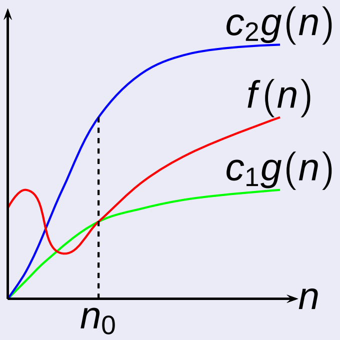
Dividing by n^2 yields,

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

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

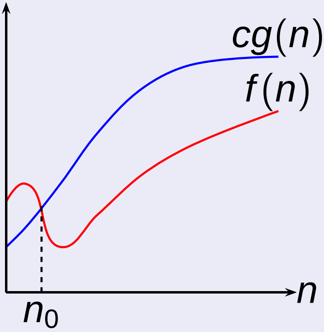
Ω-notation provides an **asymptotically lower bound**.

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

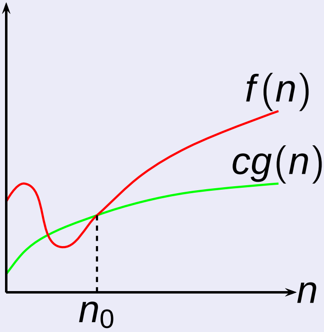
Denoted as $f(n) = \Omega(g(n))$ or $f(n) \in \Omega(g(n))$.

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

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

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

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

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

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

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

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

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

FIBONNACCI(n)

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

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 $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ 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

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

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

The substitution method

Example

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

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

The substitution method

Example

A tighter upper bound ?

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

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

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

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

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

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

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

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

$$\begin{array}{ccc} & cn^2 & \\ \diagup & & \diagdown \\ T(n/4) & & T(n/2) \end{array}$$

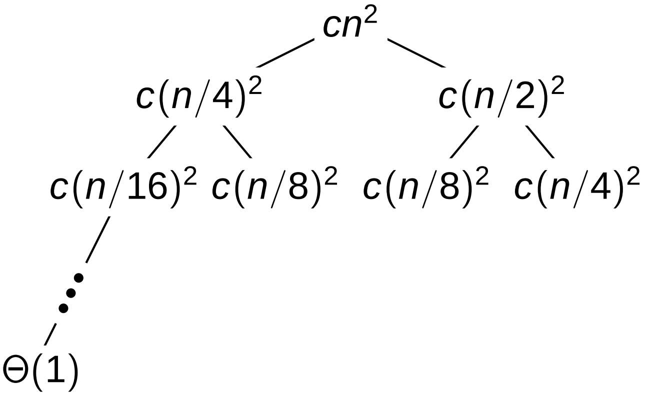
The recursion-tree method

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

$$\begin{array}{ccccccc}
 & & & cn^2 & & & \\
 & & \swarrow & & \searrow & & \\
 & c(n/4)^2 & & & & c(n/2)^2 & \\
 \swarrow & & \searrow & & \swarrow & & \searrow \\
T(n/16) & & T(n/8) & & T(n/8) & & T(n/4)
\end{array}$$

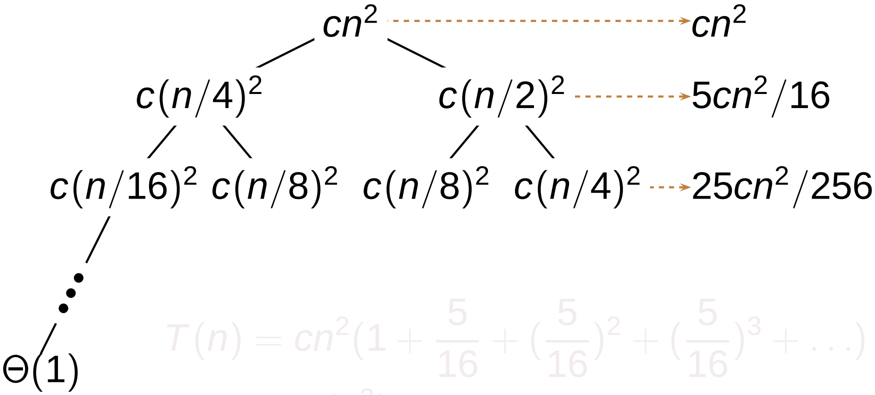
The recursion-tree method

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



The recursion-tree method

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



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

The recursion-tree method

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

$cn^2 \dashrightarrow cn^2$

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

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

$\vdots$

$\Theta(1)$

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

Changing Variables

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 $S(m) = 2S(m/2) + m$.
- $T(n) = T(2^m) = S(m) = \Theta(m \lg m) = \Theta(\lg n \lg \lg n)$.

Changing Variables

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
  $S(m) = 2S(m/2) + m$.
- $T(n) = T(2^m) = S(m) = \Theta(m \lg m) = \Theta(\lg n \lg \lg n)$.

Changing Variables

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 $S(m) = 2S(m/2) + m$.
- $T(n) = T(2^m) = S(m) = \Theta(m \lg m) = \Theta(\lg n \lg \lg n)$.

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 $af(n/b) \le cf(n)$ 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

A more general method

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

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

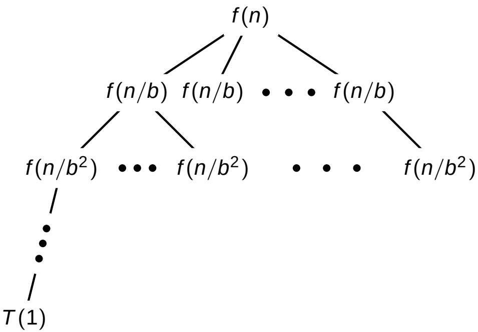
The master method

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

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

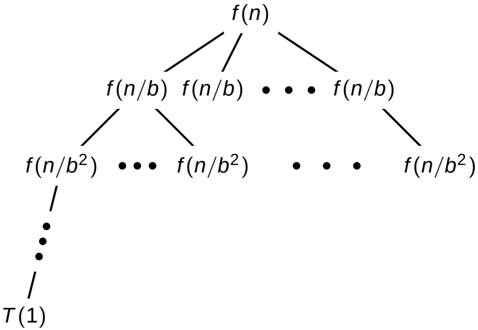
Idea of master theorem



Number of leaves

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

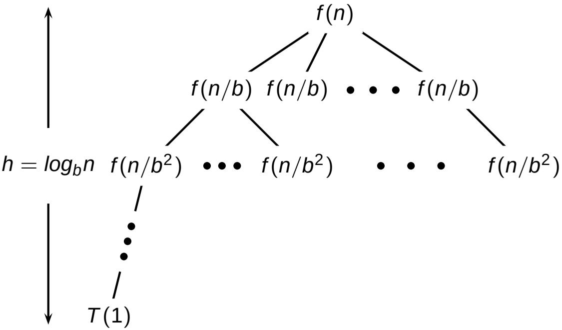
Idea of master theorem



Number of leaves

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

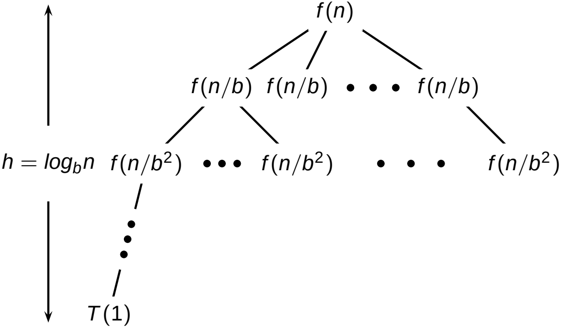
Idea of master theorem



Number of leaves

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

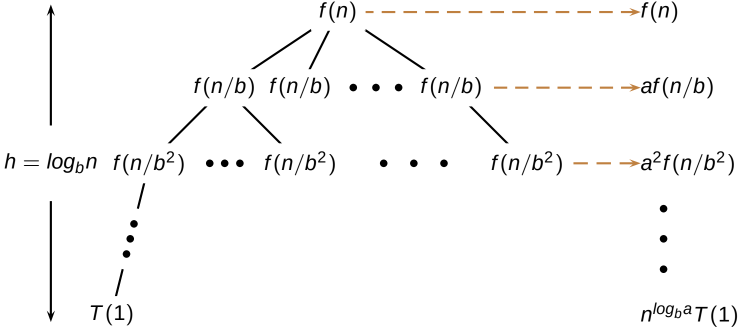
Idea of master theorem



Number of leaves

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

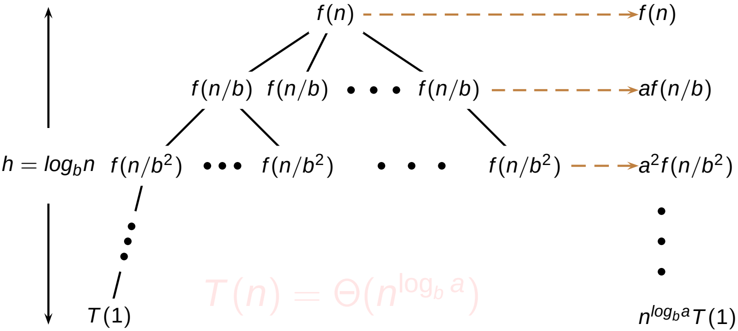
Idea of master theorem



Number of leaves

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

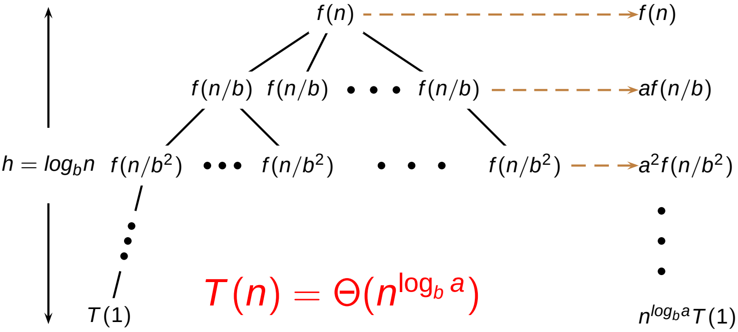
Idea of master theorem



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

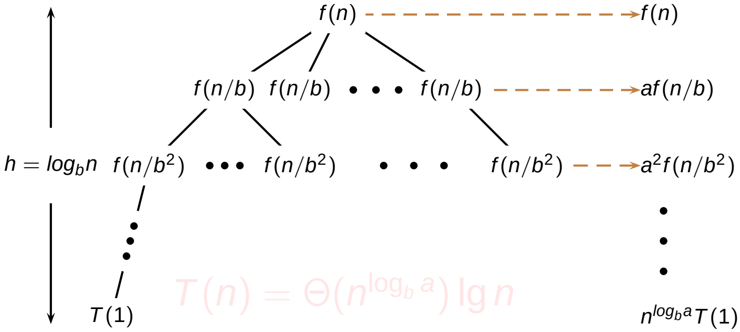


$$T(n) = \Theta(n^{\log_b a})$$

Case 1

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

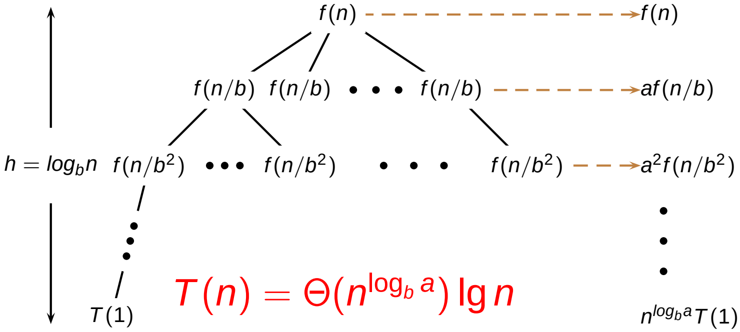
Idea of master theorem



Case 2

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

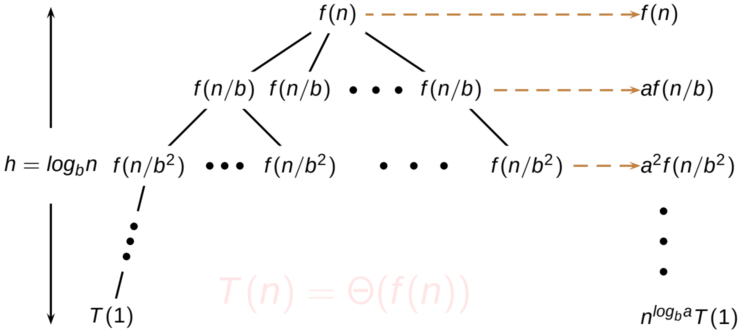
Idea of master theorem



Case 2

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

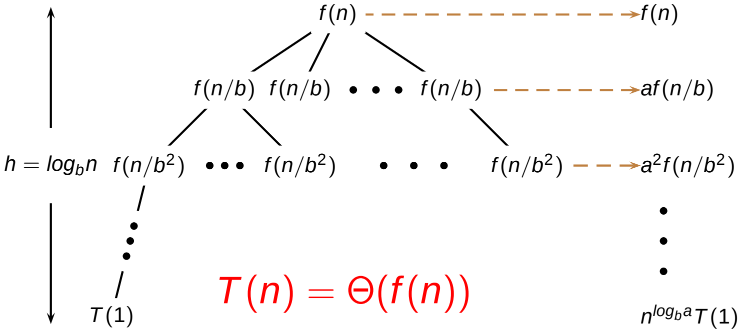
Idea of master theorem



Case 3

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

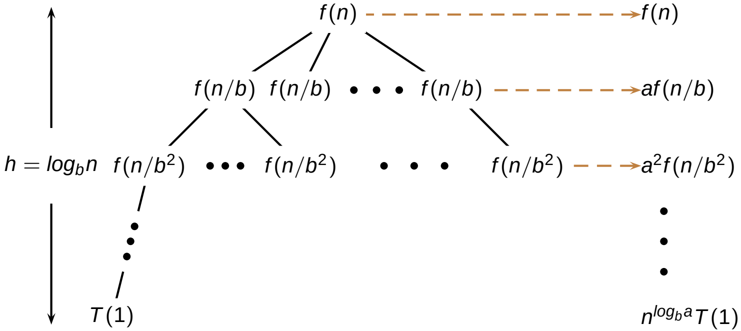
Idea of master theorem



Case 3

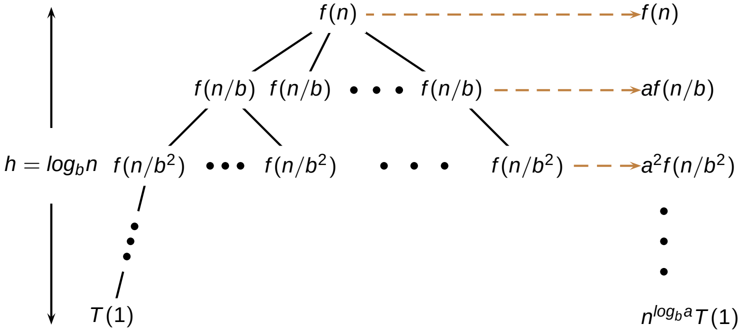
The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Proof of master theorem



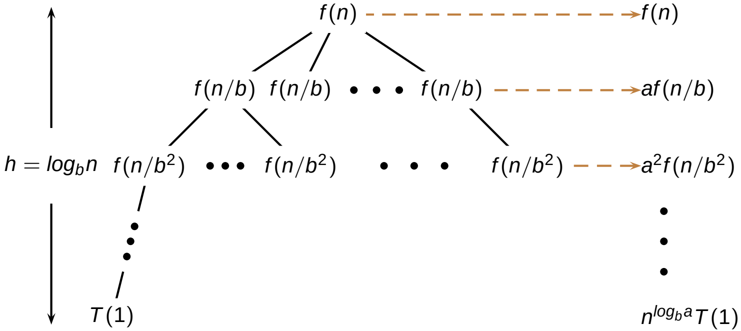
$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

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

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

Proof of master theorem

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

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

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

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

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

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 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

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

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

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

Proof of master theorem

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

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

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

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

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

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

$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \\
&\le \sum_{j=0}^{\log_b n-1} c^j f(n) \quad (\text{By } af(n/b) \le cf(n)) \\
&\le f(n)\sum_{j=0}^{\infty} c^j = f(n)\left(\frac{1}{1-c}\right)
\end{aligned}
$$

Proof of master theorem

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

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

Proof of master theorem

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

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