

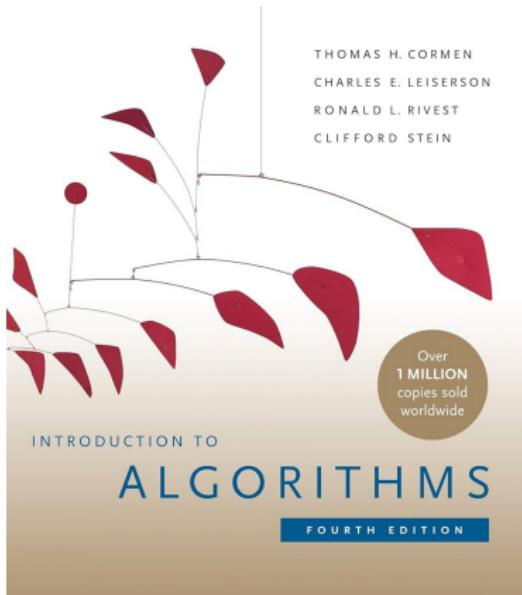
Introduction to Algorithms

Lecture 2: Asymptotic Notation

Prof. Charles E. Leiserson and Prof. Erik Demaine
Massachusetts Institute of Technology

July 29, 2025

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.

Visit <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/>.

Original slides from *Introduction to Algorithms* 6.046J/18.401J, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at <https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/>.

Plan

Asymptotic Notation

Solving recurrences

Substitution method

Recursion tree

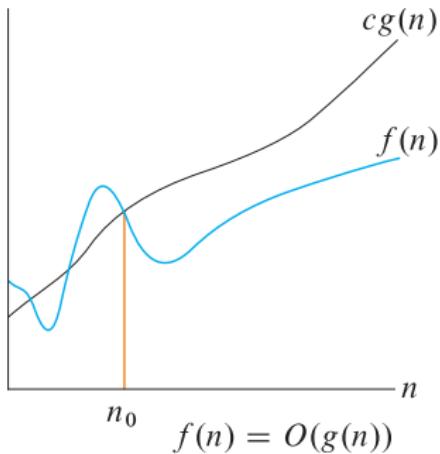
The master method

Asymptotic Notation

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

O -notation (upper bounds)

$O(g(n)) = \{f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$



O -notation (upper bounds)

$O(g(n)) = \{f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$

Example

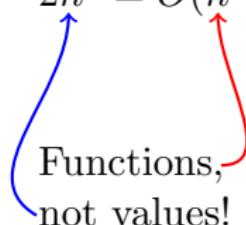
$$2n^2 = O(n^3) \quad (c = 1, n_0 = 2).$$

O -notation (upper bounds)

$O(g(n)) = \{f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$

Example

$$2n^2 = O(n^3) \quad (c = 1, n_0 = 2).$$

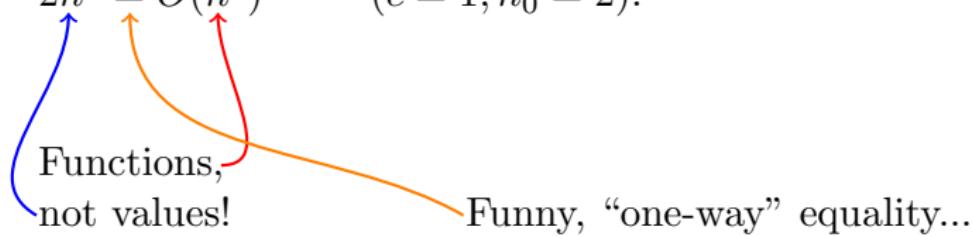


O -notation (upper bounds)

$O(g(n)) = \{f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$

Example

$$2n^2 = O(n^3) \quad (c = 1, n_0 = 2).$$



Set Definition of O -notation

$O(g(n)) = \{f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$

Set Definition of O -notation

$$\begin{aligned} O(g(n)) = \{f(n) : & \text{there exist constants} \\ & c > 0, n_0 > 0 \text{ such that} \\ & 0 \leq f(n) \leq cg(n) \\ & \text{for all } n \geq n_0\} \end{aligned}$$

Example:

$$2n^2 \in O(n^3)$$

(*Logicians*: $\lambda n.2n^2 \in O(\lambda n.n^3)$, but it's convenient to be sloppy,
as long as we understand what's really going on.)

Macro substitution

Convention:

A set in a formula represents an anonymous function in the set.

Example:

$$f(n) = n^3 + O(n^2)$$

means

$$f(n) = n^3 + h(n)$$

for some $h(n) \in O(n^2)$.

Macro substitution

Convention:

A set in a formula represents an anonymous function in the set.

Example:

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

means

for any $f(n) \in O(n)$:

$$n^2 + f(n) = h(n)$$

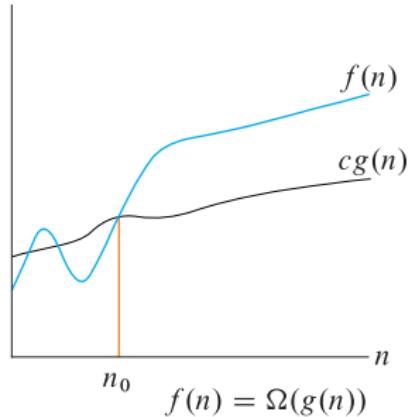
for some $h(n) \in O(n^2)$.

Ω -notation (lower bounds)

O -notation is an upper-bound notation. It makes no sense to say $f(n)$ is at least $O(n^2)$

Ω -notation (lower bounds)

$\Omega(g(n)) = \{f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$



Ω -notation (lower bounds)

O -notation is an upper-bound notation. It makes no sense to say $f(n)$ is at least $O(n^2)$

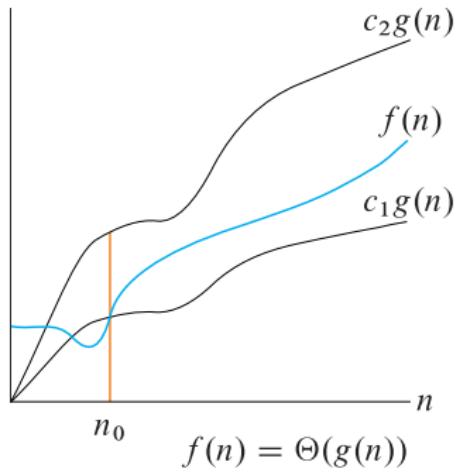
$$\begin{aligned}\Omega(g(n)) = \{f(n) : & \text{there exist constants} \\ & c > 0, n_0 > 0 \text{ such that} \\ & 0 \leq cg(n) \leq f(n) \\ & \text{for all } n \geq n_0\}\end{aligned}$$

Example:

$$\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$$

Θ -notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$



Θ -notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Example:

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

o -notation and ω -notation

O -notation and Ω -notation are like \leq and \geq .

o -notation and ω -notation are like $<$ and $>$.

$$o(g(n)) = \{f(n) : \text{for any constant } c > 0, \\ \text{there is a constant } n_0 > 0 \\ \text{such that } 0 \leq f(n) \leq cg(n) \\ \text{for all } n \geq n_0\}$$

Example:

$$2n^2 = o(n^3) \quad (n_0 = \frac{2}{c})$$

o -notation and ω -notation

O -notation and Ω -notation are like \leq and \geq .

o -notation and ω -notation are like $<$ and $>$.

$$\begin{aligned}\omega(g(n)) = \{f(n) : & \text{for any constant } c > 0, \\ & \text{there is a constant } n_0 > 0 \\ & \text{such that } 0 \leq cg(n) \leq f(n) \\ & \text{for all } n \geq n_0\}\end{aligned}$$

Example:

$$\sqrt{n} = \omega(\lg n) \quad (n_0 = 1 + \frac{1}{c})$$

Plan

Asymptotic Notation

Solving recurrences

Substitution method

Recursion tree

The master method

Solving recurrences

- ▶ The analysis of merge-sort from **Lecture 1** required us to solve a recurrence.
- ▶ Recurrences are like solving integrals, differential equations, etc.
 - ▶ Learn a few tricks.
- ▶ **Lecture 3:** Applications of recurrences to divide-and-conquer algorithms.

Substitution method

The method is based on guessing a possible solution and then verifying it using mathematical induction. It is divided into the following steps:

1. **Guess a solution:** Propose a general form of the solution $T(n)$, based on the structure of the problem.
2. **Substitute into the recurrence:** Replace the conjectured solution in the recurrence equation to check if it holds.
3. **Adjust if necessary:** If the conjecture is not valid, modify it by adding constants or additional terms.
4. **Prove by induction:** Use mathematical induction to demonstrate that the conjecture is correct.

Substitution method

The most general method:

1. **Guess** the form of the solution.
2. **Verify** by induction.
3. **Solve** for constants.

Substitution method

The most general method:

1. **Guess** the form of the solution.
2. **Verify** by induction.
3. **Solve** for constants.

Example:

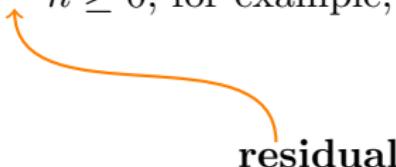
$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

- ▶ 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.

Example of substitution

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4c\left(\frac{n}{2}\right)^3 + n \\ &= \left(\frac{c}{2}\right)n^3 + n \\ &= cn^3 - \left(\left(\frac{c}{2}\right)n^3 - n\right) \leftarrow \text{desired} - \text{residual} \\ &\leq cn^3 \leftarrow \text{desired} \end{aligned}$$

whenever $\left(\frac{c}{2}\right)n^3 - n \geq 0$, for example, if $c \geq 2$ and $n \geq 1$.



Example (continued)

- ▶ We must also handle the initial conditions, that is, ground the induction with base cases.
- ▶ **Base:** $T(n) = \Theta(1)$ for all $n \leq n_0$, where n_0 is a suitable constant.
- ▶ For $1 \leq n < n_0$, we have “ $\Theta(1)$ ” $\leq cn^3$, if we pick c big enough.

Example (continued)

- ▶ We must also handle the initial conditions, that is, ground the induction with base cases.
 - ▶ **Base:** $T(n) = \Theta(1)$ for all $n \leq n_0$, where n_0 is a suitable constant.
 - ▶ For $1 \leq n < n_0$, we have “ $\Theta(1)$ ” $\leq cn^3$, if we pick c big enough.
-
-

This bound is not tight!

A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

i A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

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

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4c\left(\frac{n}{2}\right)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \end{aligned}$$

A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

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

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$\leq 4c\left(\frac{n}{2}\right)^2 + n$$

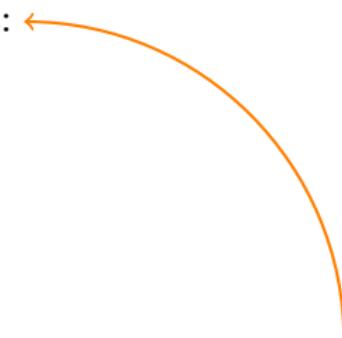
$$= cn^2 + n$$

~~$= O(n^2)$~~ **Wrong!** We must prove the I.H.

↳ A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

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

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4c\left(\frac{n}{2}\right)^2 + n \\ &= cn^2 + n \\ &= \cancel{O(n^2)} \text{ Wrong! We must prove the I.H.} \\ &= cn^2 - (-n) [\text{desired} - \text{residual}] \\ &\leq cn^2 \text{ for no choice of } c > 0. \text{ Lose!} \end{aligned}$$


|A tighter upper bound!

IDEA:

- ▶ Strengthen the inductive hypothesis.
- ▶ **Subtract** a low-order term.
- ▶ *Inductive hypothesis:* $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$.

|A tighter upper bound!

IDEA:

- ▶ Strengthen the inductive hypothesis.
- ▶ **Subtract** a low-order term.
- ▶ *Inductive hypothesis:* $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$.

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &= 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\left(\frac{n}{2}\right)\right) + n \\ &= c_1n^2 - 2c_2n + n \\ &= c_1n^2 - c_2n - c_2n + n \\ &= c_1n^2 - c_2n - (c_2n - n) \\ &\leq c_1n^2 - c_2n \text{ if } c_2 \geq 1. \end{aligned}$$

|A tighter upper bound!

IDEA:

- ▶ Strengthen the inductive hypothesis.
- ▶ **Subtract** a low-order term.
- ▶ *Inductive hypothesis:* $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$.

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &= 4\left(c_1\left(\frac{n}{2}\right)^2 - c_1\left(\frac{n}{2}\right)\right) + n \\ &= c_1n^2 - 2c_2n + n \\ &= c_1n^2 - c_2n - c_2n + n \\ &= c_1n^2 - c_2n - (c_2n - n) \\ &\leq c_1n^2 - c_2n \text{ if } c_2 \geq 1. \end{aligned}$$

Pick c_1 big enough to handle the initial conditions.

Recursion-tree method

- ▶ A recursion tree models the costs (time) of a recursive execution of an algorithm.
- ▶ The recursion-tree method can be unreliable, just like any method that uses ellipses (...).
- ▶ The recursion-tree method promotes intuition, however.
- ▶ The recursion-tree method is good for generating guesses for the substitution method.

Steps of the recurrence-tree method

1. **Expand the recurrence** over multiple levels until a general pattern emerges.
2. **Determine the cost at each level**, which usually depends on the number of subproblems and their size.
3. **Calculate the depth of the tree**, which is the total number of levels until reaching base cases.
4. **Sum the costs of all levels** to obtain the overall cost.

Example of recursion tree

Solve $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$

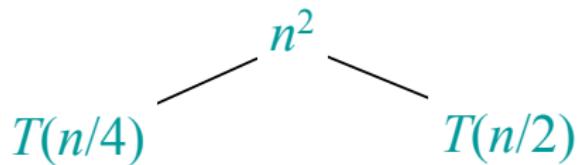
Example of recursion tree

Solve $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$

$\textcolor{teal}{T(n)}$

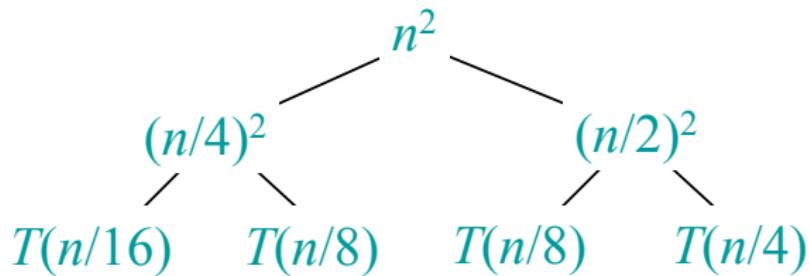
Example of recursion tree

Solve $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$



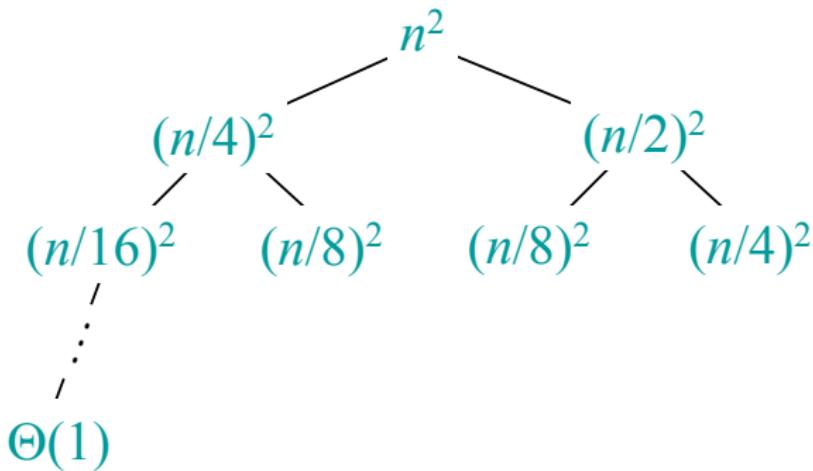
Example of recursion tree

Solve $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$



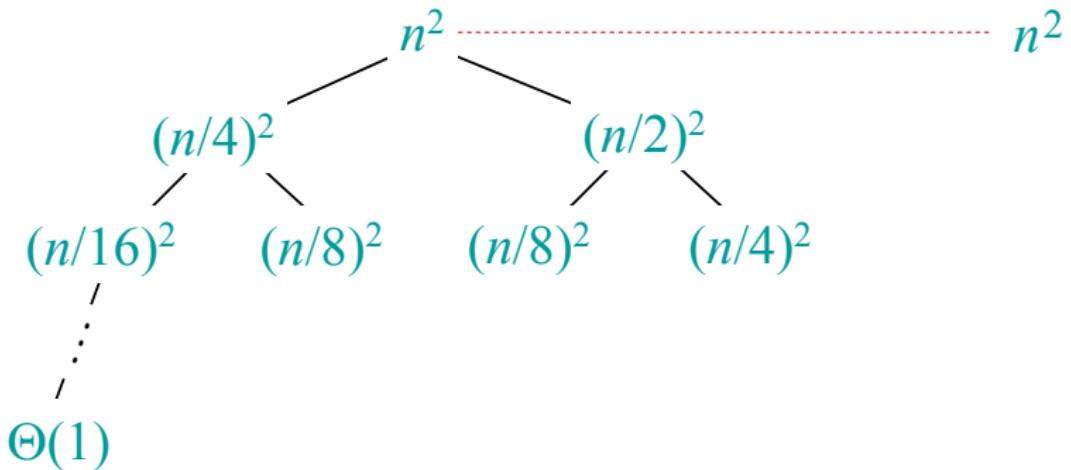
Example of recursion tree

Solve $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$



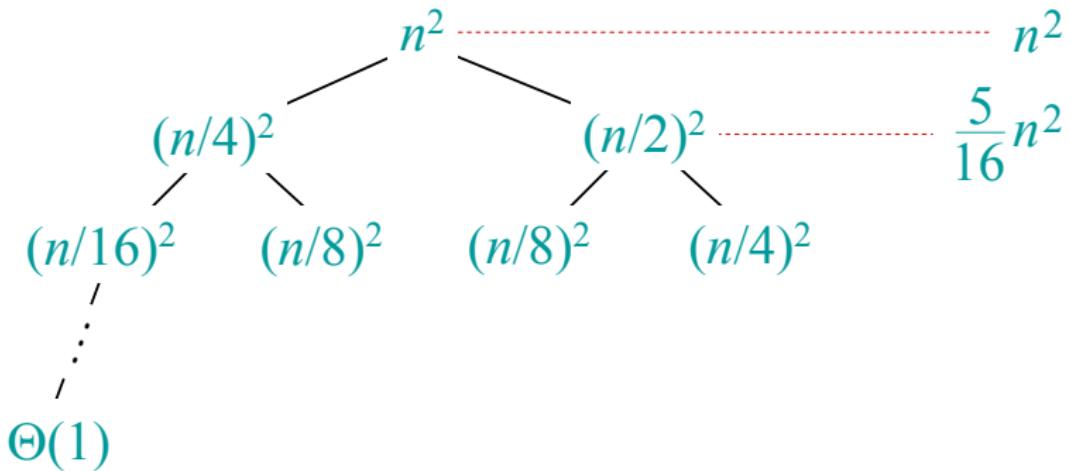
Example of recursion tree

$$\text{Solve } T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$$



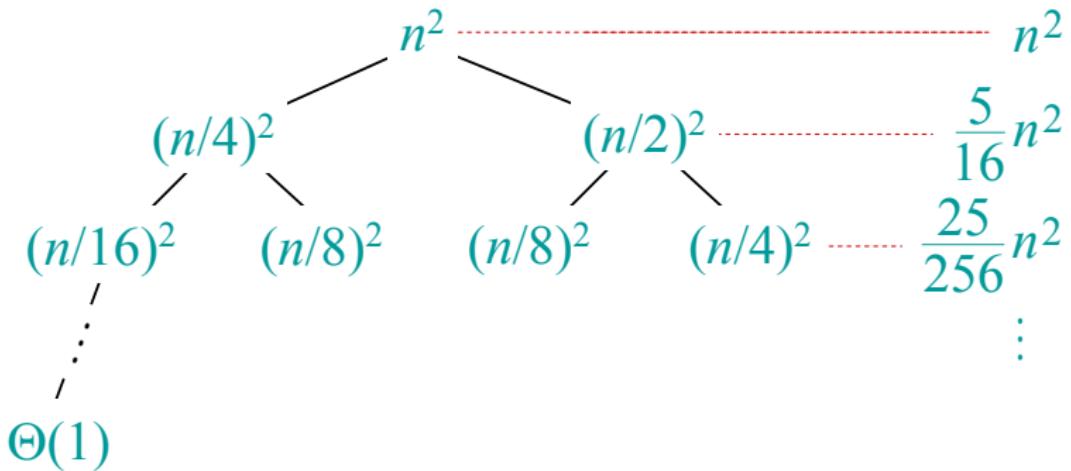
Example of recursion tree

$$\text{Solve } T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$$



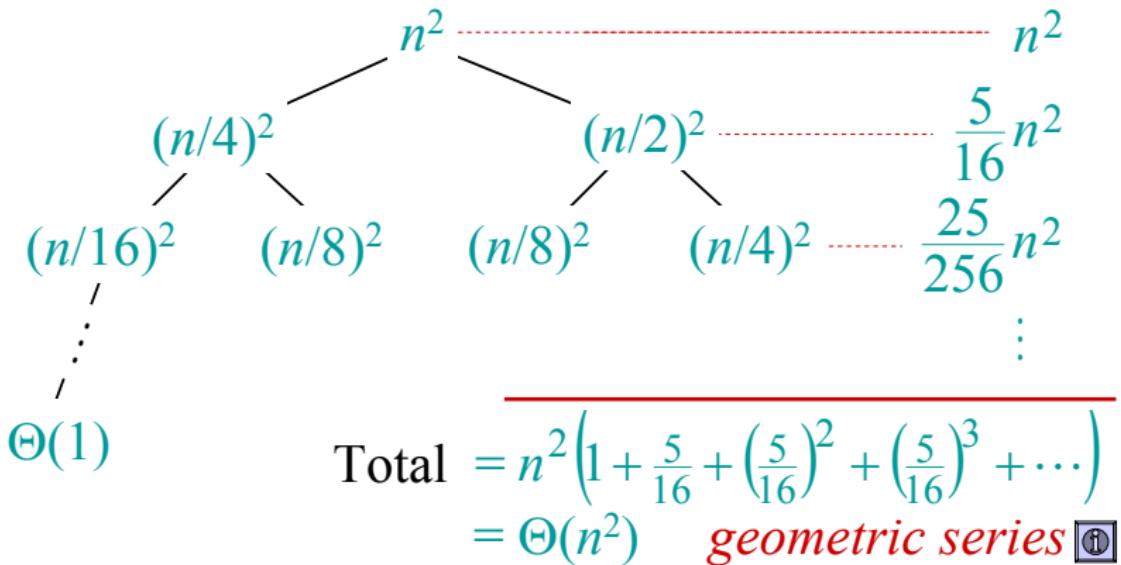
Example of recursion tree

$$\text{Solve } T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$$



Example of recursion tree

$$\text{Solve } T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$$



The master method

The master method applies to recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

You have a subproblems.

The master method

The master method applies to recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

You have a subproblems.

Each of them is of size $\frac{n}{b}$.

The master method

The master method applies to recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

You have a subproblems.

Each of them is of size $\frac{n}{b}$.

Then you're doing $f(n)$ nonrecursive work.

The master method

The master method applies to recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

You have a subproblems.

Each of them is of size $\frac{n}{b}$.

Then you're doing $f(n)$ nonrecursive work.

The master method

The master method applies to recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $a \geq 1$, $b > 1$, and $f(n)$ is *asymptotically positive*.

The master method

The master method applies to recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $a \geq 1$, $b > 1$, and $f(n)$ is *asymptotically positive*.

Note

asymptotically positive means $f(n) > 0$ for $n \geq n_0$.

Three common cases

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

Three common cases

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

Note:

$n^{\log_b a}$ = The number of leaves in the recursion tree.

Three common cases

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

Note:

$n^{\log_b a}$ = The number of leaves in the recursion tree.

Case 1 $f(n) < n^{\log_b a}$

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

Case 3 $f(n) > n^{\log_b a}$

Three common cases

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

Three common cases

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

1 $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

- ▶ $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^ε factor, polynomially smaller).

▶ **Solution:**

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

Three common cases

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

1 $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

- ▶ $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^ε factor, polynomially smaller).

▶ **Solution:**

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

2 $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.

- ▶ $f(n)$ and $n^{\log_b a}$ grow at similar rates, up to poly log factor.

▶ **Solution:**

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n).$$

Three common cases

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

1 $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.

- ▶ $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^ε factor, polynomially smaller).

▶ **Solution:**

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

2 $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.

- ▶ $f(n)$ and $n^{\log_b a}$ grow at similar rates, up to poly log factor.
- ▶ **Solution:**

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n).$$

3 $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.

- ▶ $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^ε factor, polynomially faster),
and $f(n)$ satisfies the ***regularity condition*** that
 $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$.

▶ **Solution:**

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

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n.$$

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$
$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n.$$

CASE 1:

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n.$$

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n.$$

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

$$\therefore T(n) = \Theta(n^2).$$

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^2.$$

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^2.$$

CASE 2:

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^2.$$

CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^2.$$

CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$$\therefore T(n) = \Theta(n^2 \lg n).$$

Examples

Ex.

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

Examples

Ex.

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

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^3.$$

Examples

Ex.

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

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^3.$$

CASE 3:

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$
$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^3.$$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$.

Examples

Ex.

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

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^3.$$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$.

and $4\left(\frac{n}{2}\right)^3 \leq cn^3$ (reg. cond.) for $c = \frac{1}{2}$.

Examples

Ex.

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

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = n^3.$$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$.

and $4\left(\frac{n}{2}\right)^3 \leq cn^3$ (reg. cond.) for $c = \frac{1}{2}$.

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

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\lg n}$$

Examples

Ex.

$$T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\lg n}$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = \frac{n^2}{\lg n}.$$

Examples

Ex.

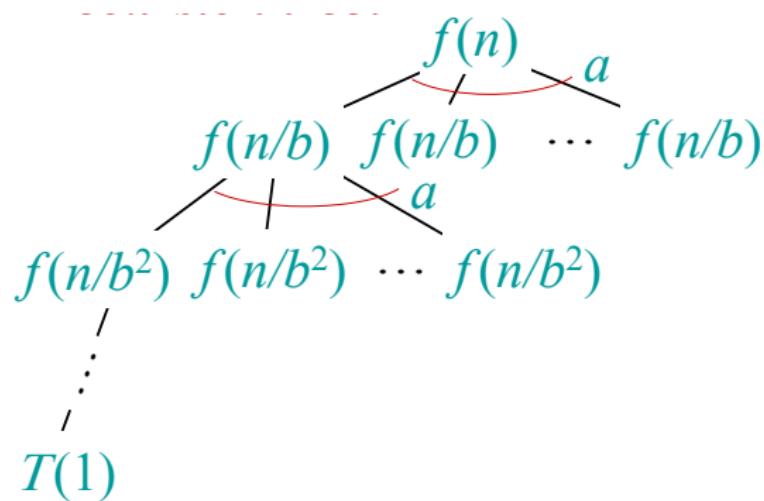
$$T(n) = 4T\left(\frac{n}{2}\right) + \frac{n^2}{\lg n}$$

$$a = 4, b = 2 \implies n^{\log_b a} = n^2 ; f(n) = \frac{n^2}{\lg n}.$$

- ▶ $f(n) = \frac{n^2}{\lg n}$. Have $f(n) = o(n)$, so that $f(n)$ grows more slowly than n , it doesn't grow polynomially slower.
- ▶ In terms of the master theorem, have $f(n) = n^2 \lg^{-1} n$, so that $k = -1$.
- ▶ Master theorem holds only for $k \geq 0$, so case 2 does not apply.
- ▶ Master method does not apply.

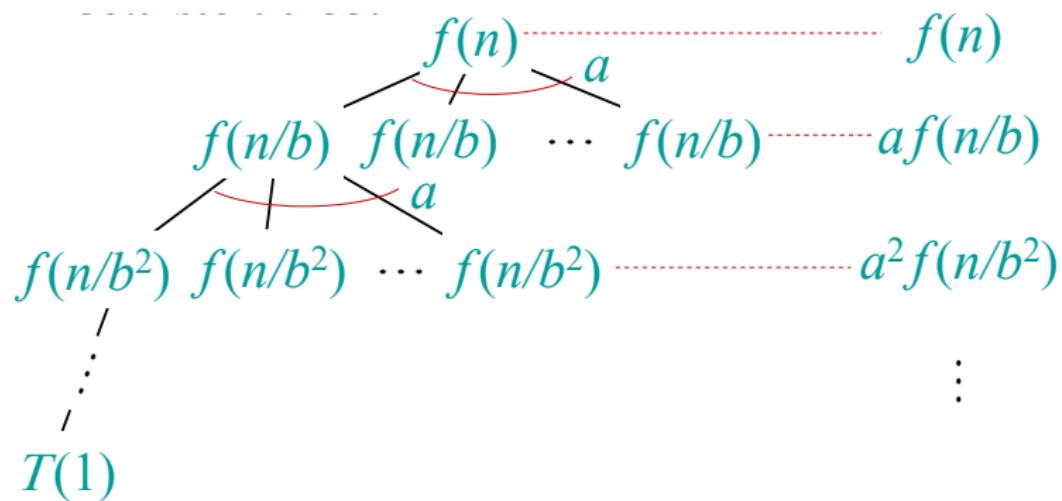
Intuition behind of master theorem

Recursion tree:



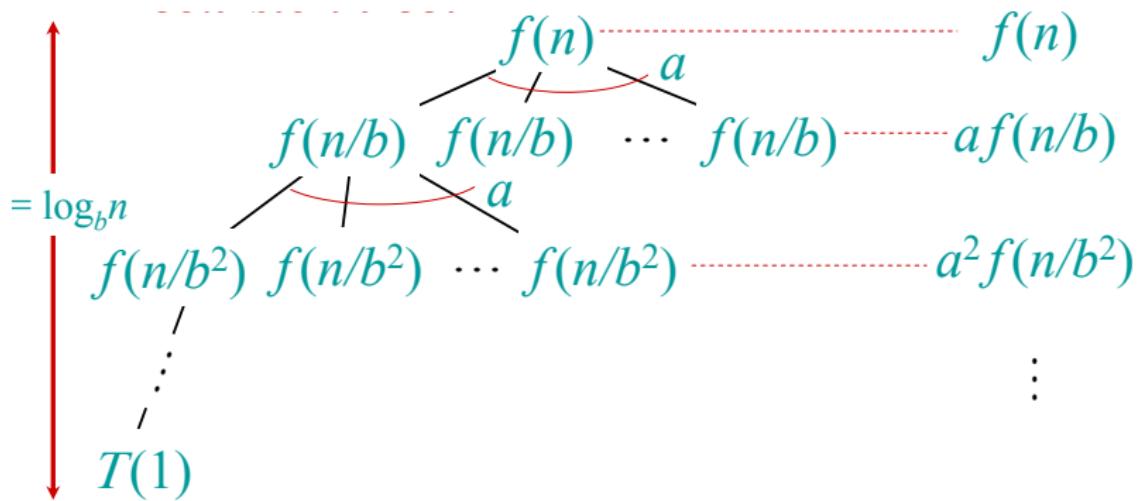
Intuition behind of master theorem

Recursion tree:



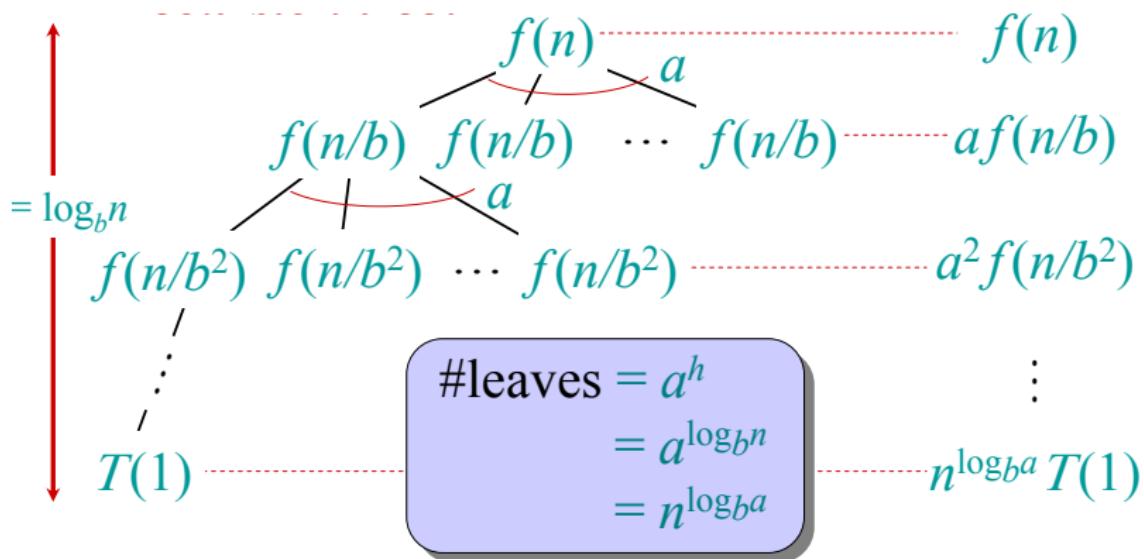
Intuition behind of master theorem

Recursion tree:



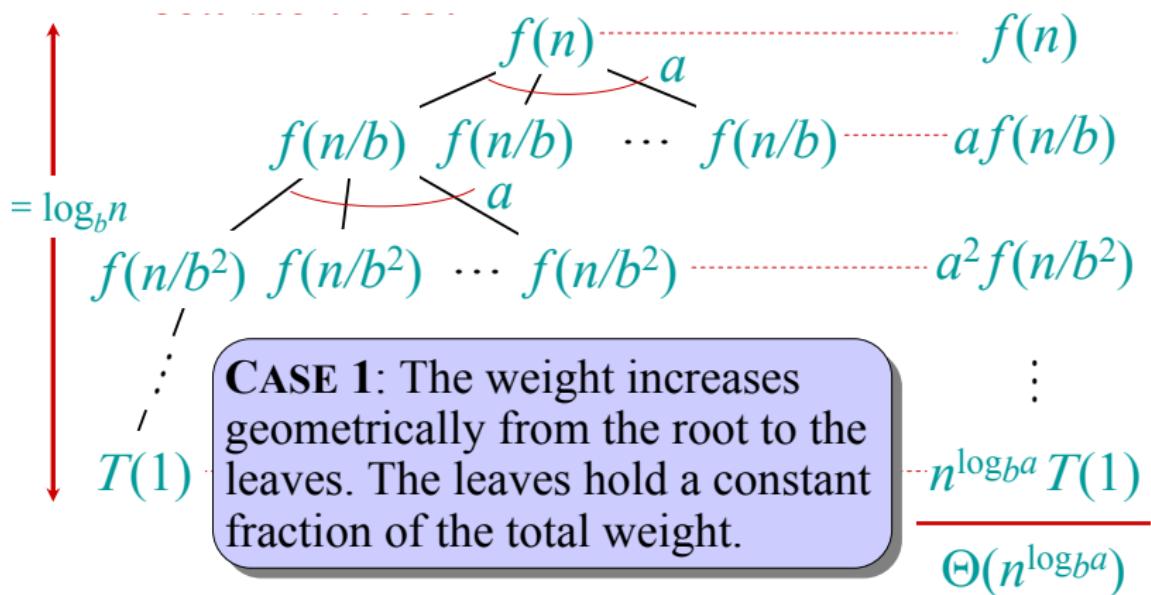
Intuition behind of master theorem

Recursion tree:



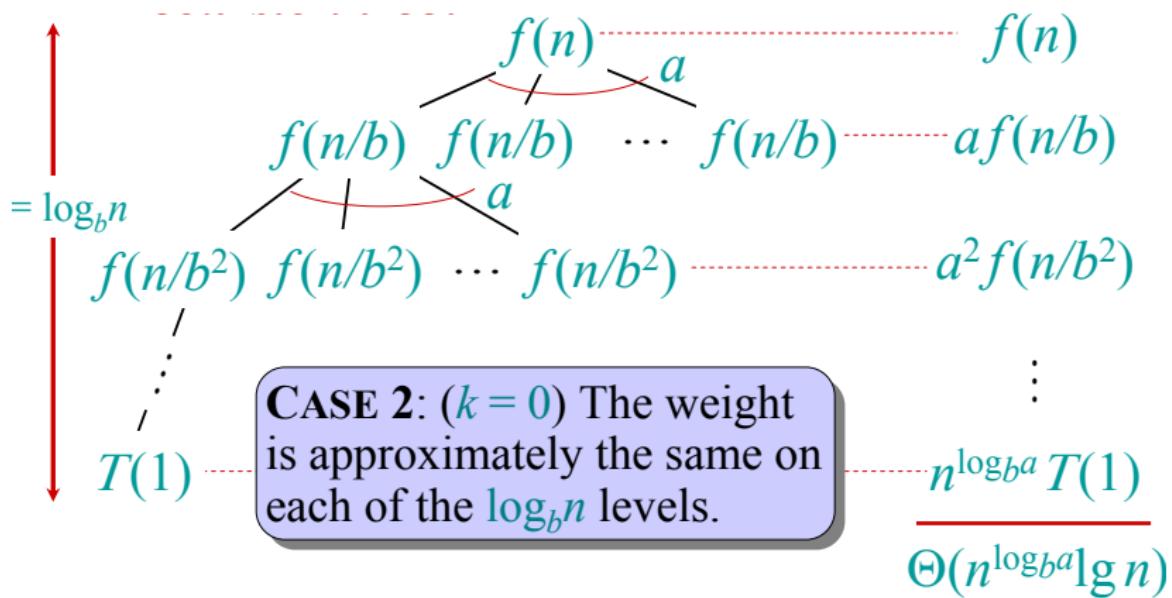
Intuition behind of master theorem

Recursion tree:



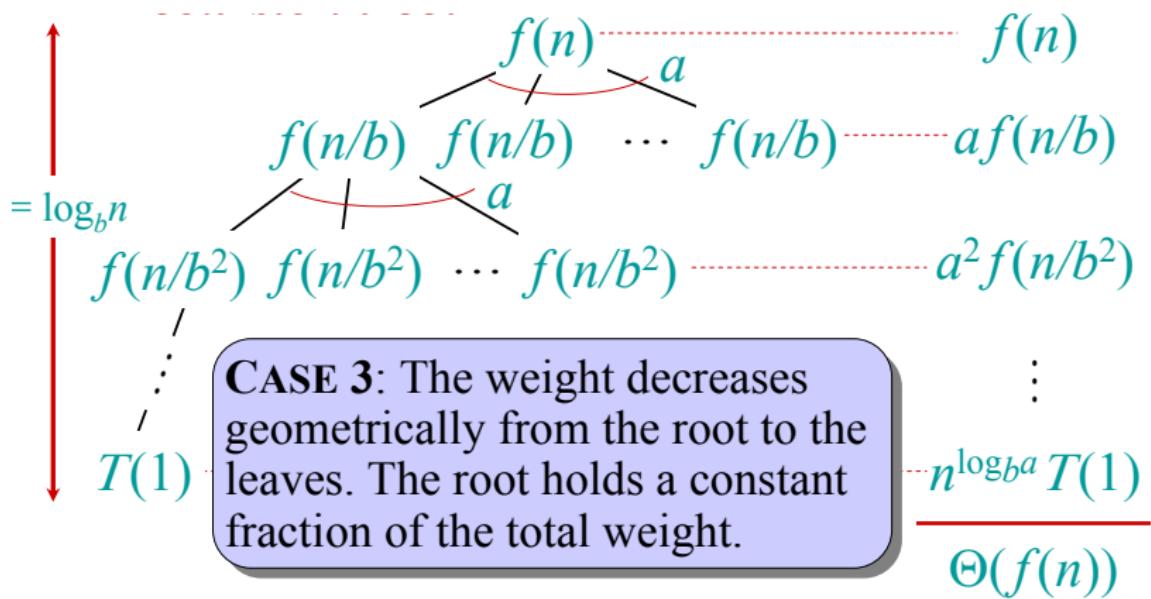
Intuition behind of master theorem

Recursion tree:



Intuition behind of master theorem

Recursion tree:



Appendix: geometric series

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x} \text{ for } x \neq 1$$

$$1 + x + x^2 + \cdots = \frac{1}{1 - x} \text{ for } |x| < 1$$

End of Lecture 2.

TDT5FTOTTC



Tim Duncan in Wikipedia.

TDT5FTOTTC



Top 5 Fundamental Takeaways

Top 5 Fundamental Takeaways

- 5 **Solving Recurrences:** Common methods to solve recurrences include substitution, recursion trees, and the master theorem, each providing different approaches to analyze recursive complexity.

Top 5 Fundamental Takeaways

- 5 **Solving Recurrences:** Common methods to solve recurrences include substitution, recursion trees, and the master theorem, each providing different approaches to analyze recursive complexity.
- 4 **Tightening Bounds Using Substitution:** Strengthening inductive hypotheses by subtracting lower-order terms helps refine bounds when standard methods provide loose approximations.

Top 5 Fundamental Takeaways

- 5 **Solving Recurrences:** Common methods to solve recurrences include substitution, recursion trees, and the master theorem, each providing different approaches to analyze recursive complexity.
- 4 **Tightening Bounds Using Substitution:** Strengthening inductive hypotheses by subtracting lower-order terms helps refine bounds when standard methods provide loose approximations.
- 3 **Recursion Tree Intuition:** A recursion tree models the breakdown of recursive calls, where the total complexity is derived by summing work across all levels.

Top 5 Fundamental Takeaways

- 5 **Solving Recurrences:** Common methods to solve recurrences include substitution, recursion trees, and the master theorem, each providing different approaches to analyze recursive complexity.
- 4 **Tightening Bounds Using Substitution:** Strengthening inductive hypotheses by subtracting lower-order terms helps refine bounds when standard methods provide loose approximations.
- 3 **Recursion Tree Intuition:** A recursion tree models the breakdown of recursive calls, where the total complexity is derived by summing work across all levels.
- 2 **Master Theorem Cases:** The master theorem classifies recurrences into three cases based on how $f(n)$ compares to $n^{\log_b(a)}$, determining whether recursion, work per level, or additional growth dominates.

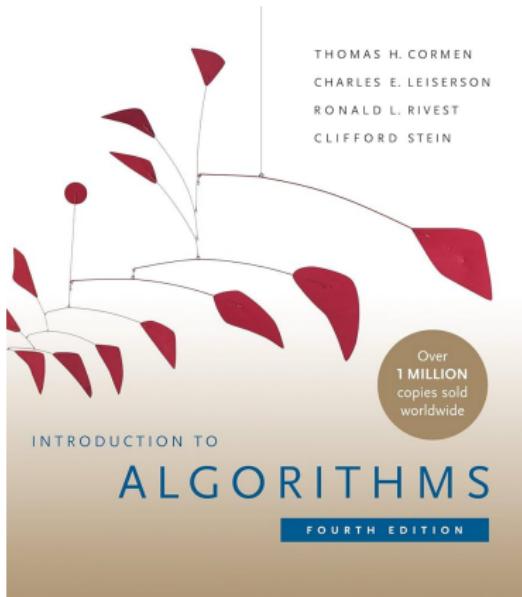
Top 5 Fundamental Takeaways

- 5 **Solving Recurrences:** Common methods to solve recurrences include substitution, recursion trees, and the master theorem, each providing different approaches to analyze recursive complexity.
- 4 **Tightening Bounds Using Substitution:** Strengthening inductive hypotheses by subtracting lower-order terms helps refine bounds when standard methods provide loose approximations.
- 3 **Recursion Tree Intuition:** A recursion tree models the breakdown of recursive calls, where the total complexity is derived by summing work across all levels.
- 2 **Master Theorem Cases:** The master theorem classifies recurrences into three cases based on how $f(n)$ compares to $n^{\log_b(a)}$, determining whether recursion, work per level, or additional growth dominates.
- 1 **O , Ω , and Θ** notations describe upper, lower, and tight bounds on algorithm growth, while o and ω represent strict bounds.

Top 5 Fundamental Takeaways

- 5 **Solving Recurrences:** Common methods to solve recurrences include substitution, recursion trees, and the master theorem, each providing different approaches to analyze recursive complexity.
- 4 **Tightening Bounds Using Substitution:** Strengthening inductive hypotheses by subtracting lower-order terms helps refine bounds when standard methods provide loose approximations.
- 3 **Recursion Tree Intuition:** A recursion tree models the breakdown of recursive calls, where the total complexity is derived by summing work across all levels.
- 2 **Master Theorem Cases:** The master theorem classifies recurrences into three cases based on how $f(n)$ compares to $n^{\log_b(a)}$, determining whether recursion, work per level, or additional growth dominates.
- 1 **O , Ω , and Θ** notations describe upper, lower, and tight bounds on algorithm growth, while o and ω represent strict bounds.

Introduction to Algorithms



Content has been extracted from *Introduction to Algorithms*, Fourth Edition, by Cormen, Leiserson, Rivest, and Stein. MIT Press. 2022.

Visit <https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/>.

Original slides from *Introduction to Algorithms* 6.046J/18.401J, Fall 2005 Class by Prof. Charles Leiserson and Prof. Erik Demaine. MIT OpenCourseWare Initiative available at <https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/>.