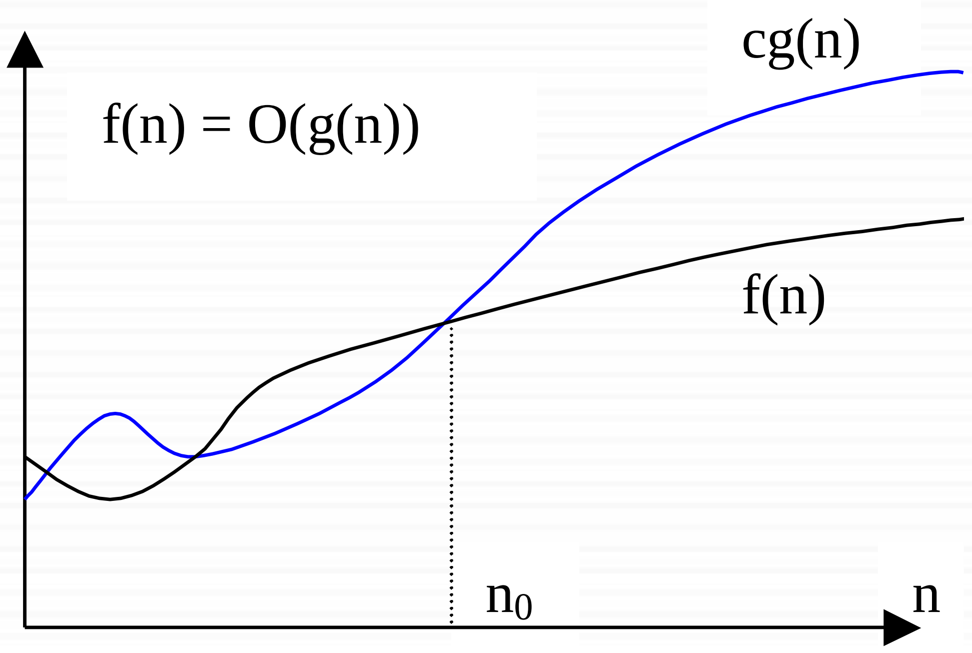
Algorithms I

Asymptotic Notation

O-notation: Asymptotic upper bound

$f(n) = O(g(n))$ if $\exists$ positive constants c, n_0 such that

$$0 \le f(n) \le cg(n), \forall n \ge n_0$$



Asymptotic running times of algorithms are usually defined by functions whose domain are N={0, 1, 2, …} (natural numbers)

Example

Show that $2n^2 = O(n^3)$

We need to find two positive constants: **c** and $\mathbf{n_0}$ such that:

$$0 \leq 2n^2 \leq cn^3 \quad \text{for all } n \geq n_0$$

Choose $c = 2$ and $n_0 = 1$

$\Rightarrow 2n^2 \leq 2n^3$ for all $n \geq 1$

Or, choose $c = 1$ and $n_0 = 2$

$\Rightarrow 2n^2 \leq n^3$ for all $n \geq 2$

Example

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

We need to find two positive constants: **c** and **n_0** such that:

$$0 \leq 2n^2 + n \leq cn^2 \text{ for all } n \geq n_0$$

$$2 + (1/n) \leq c \text{ for all } n \geq n_0$$

Choose $c = 3$ and $n_0 = 1$

➔ $2n^2 + n \leq 3n^2$ for all $n \geq 1$

O-notation

- What does $f(n) = O(g(n))$ really mean?
  - The notation is a little sloppy
  - One-way equation
    - e.g. $n^2 = O(n^3)$, but we cannot say $O(n^3) = n^2$

- $O(g(n))$ is in fact a set of functions:

$$O(g(n)) = \{f(n): \exists \text{ positive constants } c, n_0 \text{ such that } 0 \le f(n) \le cg(n), \forall n \ge n_0\}$$

O-notation

- $O(g(n)) = \{f(n): \exists$ positive constants c, n_0 such that $0 \le f(n) \le cg(n), \forall n \ge n_0\}$
- In other words: $O(g(n))$ is in fact:

  <u>the set of functions that have asymptotic upper bound g(n)</u>

- e.g. $2n^2 = O(n^3)$ *<u>means</u>* $2n^2 \in O(n^3)$

$2n^2$ is in the set of functions that have asymptotic upper bound n^3

True or False?

$10^9 n^2 = O(n^2)$ True

Choose $c = 10^9$ and $n_0 = 1$

$0 \le 10^9 n^2 \le 10^9 n^2$ for $n \ge 1$

$100 n^{1.9999} = O(n^2)$ True

Choose $c = 100$ and $n_0 = 1$

$0 \le 100 n^{1.9999} \le 100 n^2$ for $n \ge 1$

$\xcancel{10^{-9} n^{2.0001} = O(n^2)}$ False

$10^{-9} n^{2.0001} \le c n^2$ for $n \ge n_0$

$10^{-9} n^{0.0001} \le c$ for $n \ge n_0$

Contradiction

O-notation

- *O*-notation is an upper bound notation
- What does it mean if we say:

"The runtime (*T(n)*) of Algorithm A is ***at least*** $O(n^2)$"

→ says nothing about the runtime. Why?

$O(n^2)$: The set of functions with asymptotic ***upper bound*** n^2

$T(n) \geq O(n^2)$ means: $T(n) \geq h(n)$ for some $h(n) \in O(n^2)$

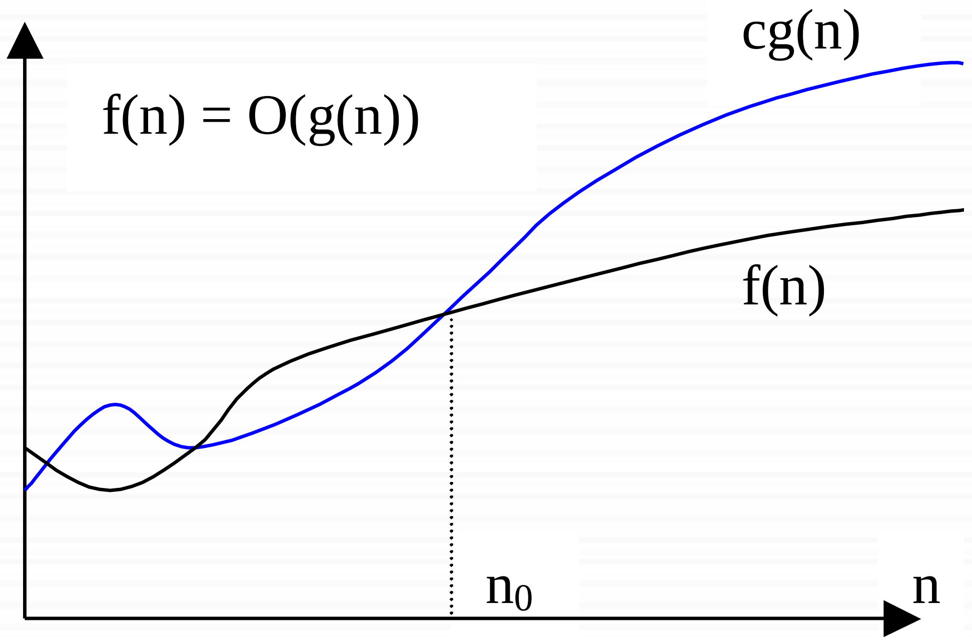
$h(n) = 0$ function is also in $O(n^2)$. Hence: $T(n) \geq 0$

runtime must be nonnegative anyway!

Summary: *O*-notation: Asymptotic upper bound

$f(n) \in O(g(n))$ if $\exists$ positive constants c, n_0 such that

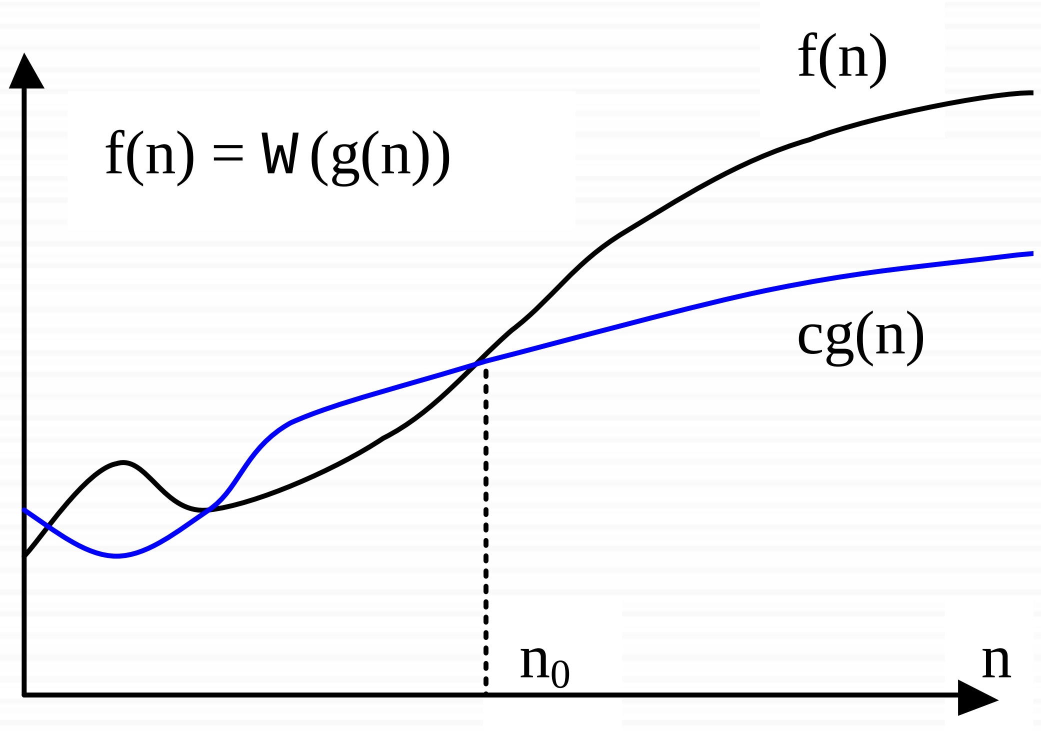
$$0 \leq f(n) \leq cg(n),\ \forall n \geq n_0$$



Ω-notation: Asymptotic lower bound

$f(n) = \Omega(g(n))$ if $\exists$ positive constants c, n_0 such that

$$0 \le cg(n) \le f(n),\ \forall n \ge n_0$$



Ω: "big Omega"

Example

Show that $2n^3 = \Omega(n^2)$

We need to find two positive constants: **c** and $\mathbf{n_0}$ such that:

$$0 \leq cn^2 \leq 2n^3 \quad \text{for all } n \geq n_0$$

Choose $c = 1$ and $n_0 = 1$

$\Rightarrow$ $n^2 \leq 2n^3$ for all $n \geq 1$

Example

Show that $\sqrt{n} = \Omega(\lg n)$

We need to find two positive constants: **c** and **n_0** such that:

$$c \lg n \leq \sqrt{n} \text{ for all } n \geq n_0$$

Choose c = 1 and $n_0 = 16$

➔ $\lg n \leq \sqrt{n}$ for all $n \geq 16$

Ω-notation: Asymptotic Lower Bound

- $\Omega(g(n)) = \{f(n)$: ∃ positive constants c, n_0 such that $0 \le cg(n) \le f(n), \forall n \ge n_0\}$

- In other words: Ω (g(n)) is in fact:

  the set of functions that have asymptotic lower bound g(n)

True or False?

$10^9n^2 = \Omega(n^2)$

True

Choose $c = 10^9$ and $n_0 = 1$

$0 \le 10^9n^2 \le 10^9n^2$ for $n \ge 1$

$\cancel{100n^{1.9999} = \Omega(n^2)}$

False

$cn^2 \le 100n^{1.9999}$ for $n \ge n_0$

$n^{0.0001} \le (100/c)$ for $n \ge n_0$

Contradiction

$10^{-9}n^{2.0001} = \Omega(n^2)$

True

Choose $c = 10^{-9}$ and $n_0 = 1$

$0 \le 10^{-9}n^2 \le 10^{-9}n^{2.0001}$ for $n \ge 1$

Summary: O-notation and Ω-notation

- $O(g(n))$: The set of functions with asymptotic upper bound $g(n)$

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

  $f(n) \in O(g(n))$ if $\exists$ positive constants c, n_0 such that

  $$0 \le f(n) \le cg(n),\ \forall n \ge n_0$$

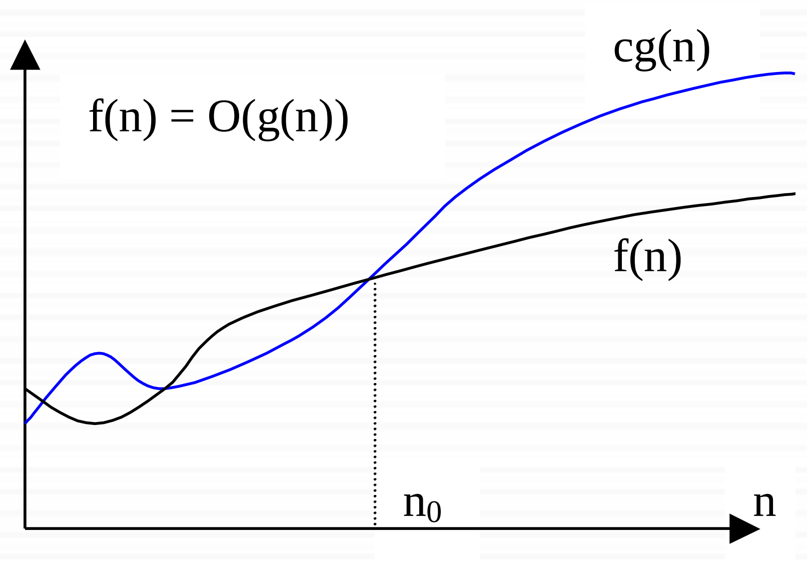
- $\Omega(g(n))$: The set of functions with asymptotic lower bound $g(n)$

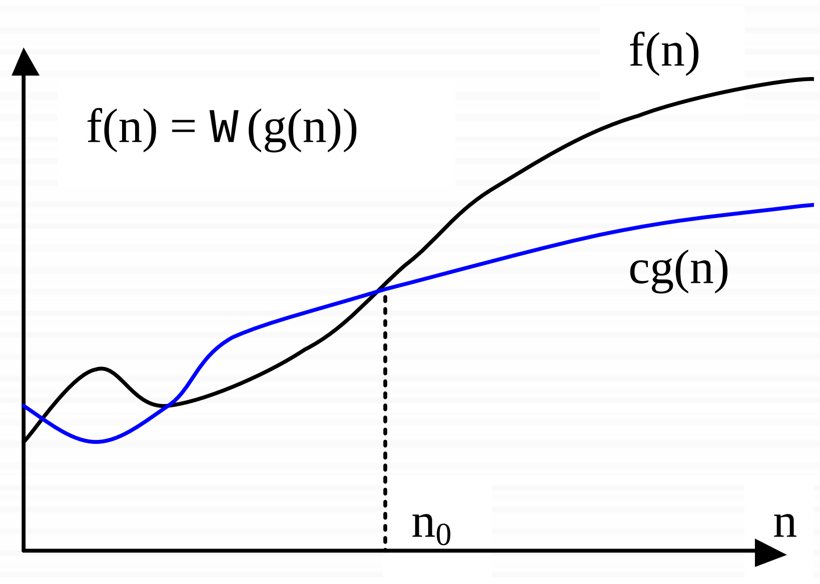
  $f(n) = \Omega(g(n))$

  $f(n) \in \Omega(g(n))$ $\exists$ positive constants c, n_0 such that

  $$0 \le cg(n) \le f(n),\ \forall n \ge n_0$$

Summary: O-notation and Ω-notation

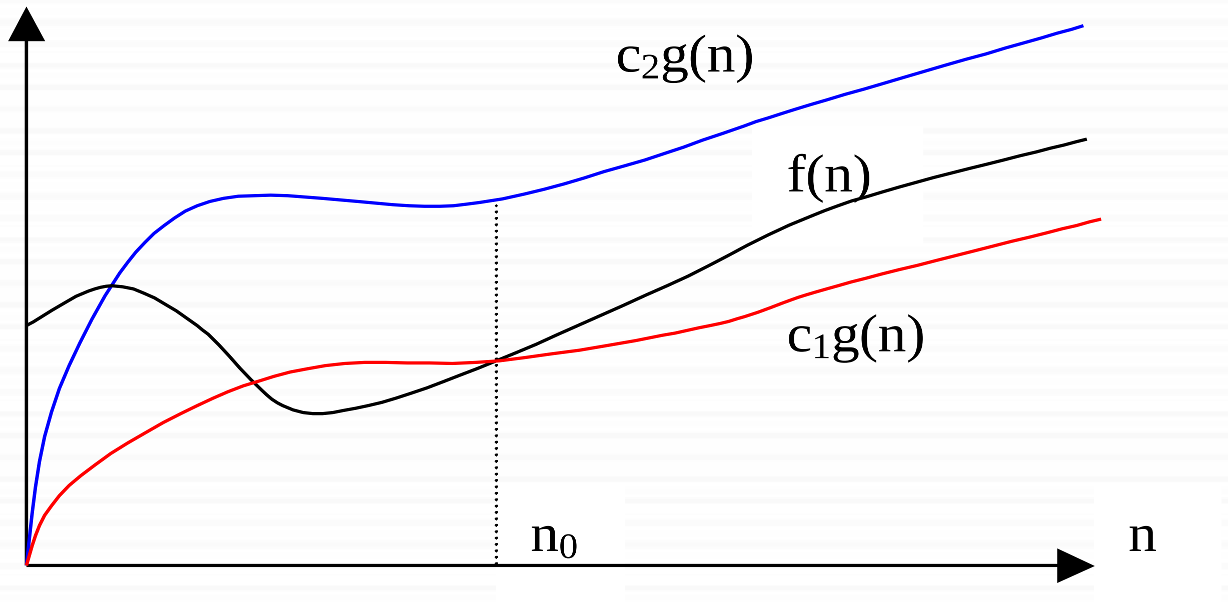




Θ-notation: Asymptotically tight bound

- $f(n)=\Theta(g(n))$ if $\exists$ positive constants c_1, c_2, n_0 such that

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n),\ \forall n \ge n_0$$



Example

Show that $2n^2 + n = \Theta(n^2)$

We need to find 3 positive constants: $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{n_0}$ such that:

$$0 \le c_1 n^2 \le 2n^2 + n \le c_2 n^2 \text{ for all } n \ge n_0$$

$$c_1 \le 2 + (1/n) \le c_2 \text{ for all } n \ge n_0$$

Choose $c_1 = 2$, $c_2 = 3$, and $n_0 = 1$

$\Rightarrow 2n^2 \le 2n^2 + n \le 3n^2$ for all $n \ge 1$

Example

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

We need to find 3 positive constants: **c_1, c_2** and **n_0** such that:

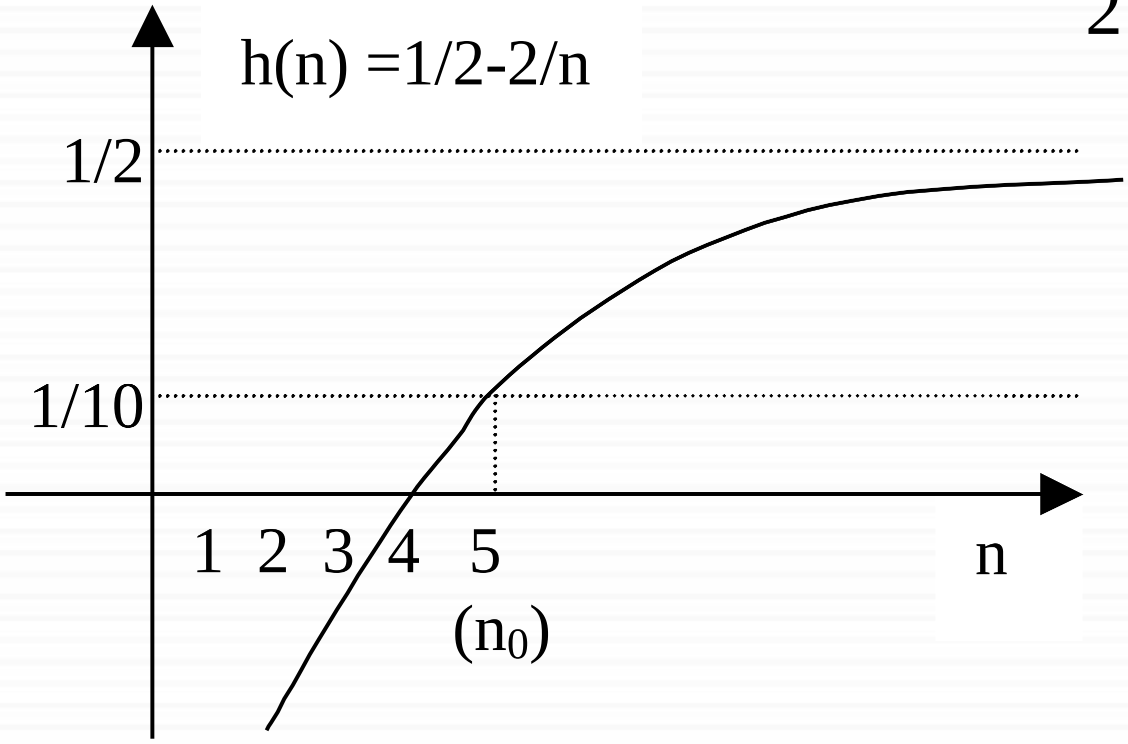
$$0 \le c_1 n^2 \le \frac{1}{2}n^2 - 2n \le c_2 n^2 \quad \text{for all } n \ge n_0$$

$$c_1 \le \frac{1}{2} - \frac{2}{n} \le c_2 \qquad \text{for all } n \ge n_0$$

Example (cont'd)

- Choose 3 positive constants: c_1, c_2, n_0 that satisfy:

$$c_1 \le \frac{1}{2} - \frac{2}{n} \le c_2 \qquad \text{for all } n \ge n_0$$



$$\frac{1}{10} \le \frac{1}{2} - \frac{2}{n} \qquad \text{for } n \ge 5$$

$$\frac{1}{2} - \frac{2}{n} \le \frac{1}{2} \qquad \text{for } n \ge 0$$

Example (cont'd)

- Choose 3 constants: c_1, c_2, n_0 that satisfy:

$$c_1 \le \frac{1}{2} - \frac{2}{n} \le c_2 \qquad \text{for all } n \ge n_0$$

$$\frac{1}{10} \le \frac{1}{2} - \frac{2}{n} \qquad \text{for } n \ge 5$$

$$\frac{1}{2} - \frac{2}{n} \le \frac{1}{2} \qquad \text{for } n \ge 0$$

Therefore, we can choose::

$$c_1 = \frac{1}{10} \qquad c_2 = \frac{1}{2} \qquad n_0 = 5$$

Θ-notation: Asymptotically tight bound

- Theorem: leading constants & low-order terms don’t matter
- Justification: can choose the leading constant large enough to make high-order term dominate other terms

True or False?

$10^9 n^2 = \Theta(n^2)$ True

$\cancel{100n^{1.9999} = \Theta(n^2)}$ False

$\cancel{10^{-9} n^{2.0001} = \Theta(n^2)}$ False

Θ-notation: Asymptotically tight bound

- $\Theta(g(n))=\{f(n)$: $\exists$ positive constants c_1, c_2, n_0 such that

  $0 \le c_1 g(n) \le f(n) \le c_2 g(n),\ \forall n \ge n_0\}$

- In other words: $\Theta(g(n))$ is in fact:

  the set of functions that have asymptotically tight bound g(n)

Θ-notation: Asymptotically tight bound

- Theorem:

  $f(n) = \Theta(g(n))$ if and only if

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

- In other words:

  Θ is stronger than both O and Ω

- In other words:

  $\Theta(g(n)) \subseteq O(g(n))$ and

  $\Theta(g(n)) \subseteq \Omega(g(n))$

Example

- Prove that $10^{-8} n^2 \neq \Theta(n)$

Before proof, note that $10^{-8}n^2 = \Omega(n)$ but $10^{-8}n^2 \neq O(n)$

Proof by contradiction:

Suppose positive constants c_2 and n_0 exist such that:

$10^{-8}n^2 \leq c_2 n \quad$ for all $n \geq n_0$

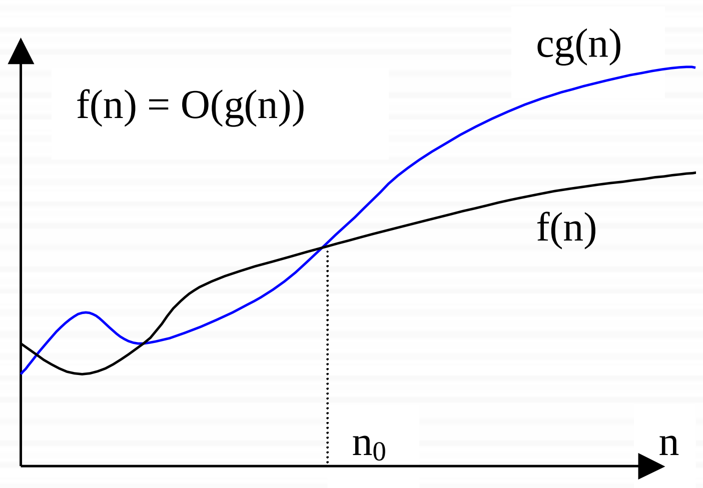
$10^{-8}n \leq c_2 \quad$ for all $n \geq n_0$

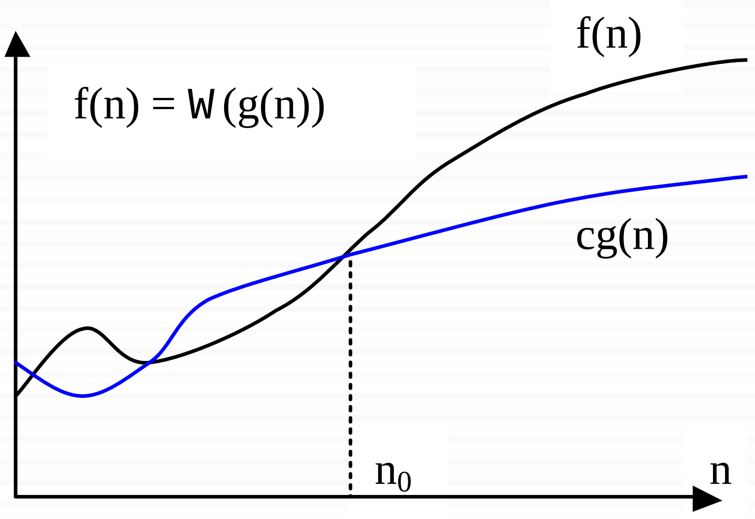
Contradiction: c_2 is a constant

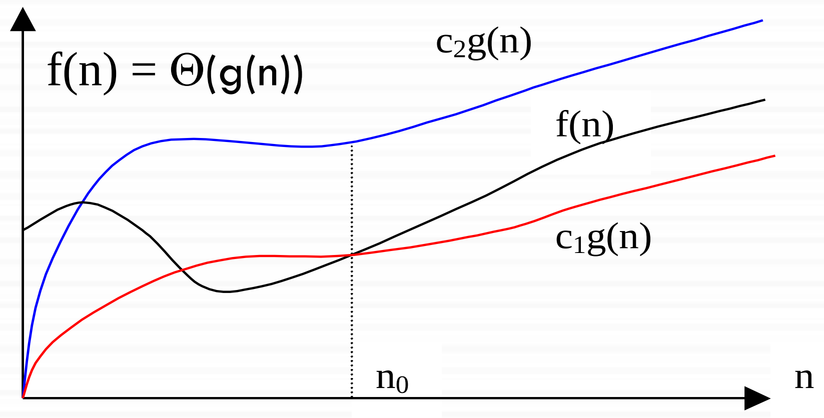
Summary: O, Ω, and Θ notations

- O(g(n)): The set of functions with asymptotic upper bound g(n)
- Ω(g(n)): The set of functions with asymptotic lower bound g(n)
- Θ(g(n)): The set of functions with asymptotically tight bound g(n)
- f(n) = Θ(g(n)) if and only if f(n) = O(g(n)) and f(n) = Ω(g(n))

Summary: O, Ω, and Θ notations







o ("small o") Notation
Asymptotic upper bound that is not tight

Reminder: Upper bound provided by O ("big O") notation can be tight or not tight:

$$\left.\begin{array}{lll} \text{e.g.} & 2n^2 = O(n^2) & \text{is asymptotically tight} \\ & 2n = O(n^2) & \text{is not asymptotically tight} \end{array}\right\} \text{both true}$$

o-Notation: An upper bound that is not asymptotically tight

o ("small o") Notation
Asymptotic upper bound that is not tight

- $o(g(n)) = \{f(n)$: for **any** constant $c > 0$,

  $\exists$ a constant $n_0 > 0$, such that

  $0 \le f(n) < cg(n), \forall n \ge n_0\}$

- Intuitively: $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

  - e.g., $2n = o(n^2)$, any positive c satisfies

    but $2n^2 \neq o(n^2)$, $c = 2$ does not satisfy

ω ("small omega") Notation
Asymptotic lower bound that is not tight

- $\omega(g(n))$ = {$f(n)$: for **any** constant $c > 0$,

  $\exists$ a constant $n_0 > 0$, such that

  $0 \le cg(n) < f(n), \forall n \ge n_0$}

- Intuitively: $$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

- e.g., $n^2/2 = \omega(n)$, any positive *c* satisfies

  but $n^2/2 \neq \omega(n^2)$, $c = 1/2$ does not satisfy

Analogy to the comparison of two real numbers

- $f(n) = O(g(n)) \leftrightarrow a \leq b$
- $f(n) = \Omega(g(n)) \leftrightarrow a \geq b$
- $f(n) = \Theta(g(n)) \leftrightarrow a = b$

- $f(n) = o(g(n)) \leftrightarrow a < b$
- $f(n) = \omega(g(n)) \leftrightarrow a > b$

True or False?

$5n^2 = O(n^2)$	True	$n^2 \lg n = O(n^2)$	False
$5n^2 = \Omega(n^2)$	True	$n^2 \lg n = \Omega(n^2)$	True
$5n^2 = \Theta(n^2)$	True	$n^2 \lg n = \Theta(n^2)$	False
$5n^2 = o(n^2)$	False	$n^2 \lg n = o(n^2)$	False
$5n^2 = \omega(n^2)$	False	$n^2 \lg n = \omega(n^2)$	True

$2^n = O(3^n)$	True		
$2^n = \Omega(3^n)$	False	$2^n = o(3^n)$	True
$2^n = \Theta(3^n)$	False	$2^n = \omega(3^n)$	False

Analogy to comparison of two real numbers

- Trichotomy property for real numbers:

  For any two real numbers a and b,

  we have either $a < b$, or $a = b$, or $a > b$

- Trichotomy property *does not hold* for asymptotic notation

  For two functions f(n) & g(n), it may be the case that ***neither*** $f(n) = O(g(n))$ ***nor*** $f(n) = \Omega(g(n))$ ***holds***

  e.g. n and $n^{1+\sin(n)}$ *cannot be compared asymptotically*

Asymptotic Comparison of Functions

(Similar to the relational properties of real numbers)

Transitivity: holds for all

e.g., $f(n) = \Theta(g(n))$ & $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

Reflexivity: holds for Θ, O, Ω

e.g., $f(n) = O(f(n))$

Symmetry: holds only for Θ

e.g., $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

Transpose symmetry: holds for $(O \leftrightarrow \Omega)$ and $(o \leftrightarrow \omega)$)

e.g., $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$

Using O-Notation to Describe Running Times

- Used to bound worst-case running times
  - Implies an upper bound runtime for arbitrary inputs as well

- Example:

  "Insertion sort has worst-case runtime of $O(n^2)$"

  Note: This $O(n^2)$ upper bound also applies to its running time on every input.

Using O-Notation to Describe Running Times

- Abuse to say "running time of insertion sort is $O(n^2)$"

- For a given n, the actual running time *depends on the particular input* of size n
  - i.e., running time is not only a function of n

- However, worst-case running time is only a function of n

Using O-Notation to Describe Running Times

- When we say:

  "*Running time of insertion sort is* $O(n^2)$",

  what we really mean is:

  "*Worst-case running time of insertion sort is* $O(n^2)$"

  or equivalently:

  "*No matter what particular input of size n is chosen, the running time on that set of inputs is* $O(n^2)$"

Using Ω-Notation to Describe Running Times

- Used to bound best-case running times
  - Implies a lower bound runtime for arbitrary inputs as well

- Example:

  "Insertion sort has best-case runtime of $\Omega(n)$"

  Note: This $\Omega(n)$ lower bound also applies to its running time on every input.

Using Ω-Notation to Describe Running Times

- When we say:

  “*Running time of algorithm A is $\Omega(g(n))$*”,

  what we mean is:

  “For any input of size n, the runtime of A is <u>at least</u> a constant times $g(n)$ for sufficiently large n”

Using □-Notation to Describe Running Times

- *Note*: It's not contradictory to say:

  "*worst-case running time of insertion sort is $\Omega(n^2)$*"

  because there exists an input that causes the algorithm to take $\Omega(n^2)$.

Using Θ-Notation to Describe Running Times

- Consider 2 cases about the runtime of an algorithm:

- Case 1: Worst-case and best-case not asymptotically equal
  - ➔ Use Θ-notation to bound worst-case and best-case runtimes separately

- Case 2: Worst-case and best-case asymptotically equal
  - ➔ Use Θ-notation to bound the runtime for any input

Using Θ-Notation to Describe Running Times Case 1

- Case 1: Worst-case and best-case not asymptotically equal
  → Use Θ-notation to bound the worst-case and best-case runtimes separately

- We can say:
  - "The worst-case runtime of insertion sort is $\Theta(n^2)$"
  - "The best-case runtime of insertion sort is $\Theta(n)$"

- But, we can't say:
  - "The runtime of insertion sort is $\Theta(n^2)$ for every input"

- A Θ-bound on worst-/best-case running time does not apply to its running time on arbitrary inputs

Using Θ-Notation to Describe Running Times Case 2

- Case 2: Worst-case and best-case asymptotically equal
  - ➔ Use Θ-notation to bound the runtime for any input

- e.g. For merge-sort, we have:

$$\left.\begin{array}{l} T(n) = O(n\lg n) \\ T(n) = \Omega(n\lg n) \end{array}\right\} \quad T(n) = \Theta(n\lg n)$$

Using Asymptotic Notation to Describe Runtimes Summary

- "The worst case runtime of Insertion Sort is $O(n^2)$"
  - Also implies: "The runtime of Insertion Sort is $O(n^2)$"

- "The best-case runtime of Insertion Sort is $\Omega(n)$"
  - Also implies: "The runtime of Insertion Sort is $\Omega(n)$"

- "The worst case runtime of Insertion Sort is $\Theta(n^2)$"
  - But: "The runtime of Insertion Sort is not $\Theta(n^2)$"

- "The best case runtime of Insertion Sort is $\Theta(n)$"
  - But: "The runtime of Insertion Sort is not $\Theta(n)$"

Using Asymptotic Notation to Describe Runtimes Summary

- "The worst case runtime of Merge Sort is $\Theta(nlgn)$"
- "The best case runtime of Merge Sort is $\Theta(nlgn)$"
- "The runtime of Merge Sort is $\Theta(nlgn)$"
    - *This is true, because the best and worst case runtimes have asymptotically the same tight bound* $\Theta(nlgn)$

Asymptotic Notation in Equations

- Asymptotic notation appears alone on the RHS of an equation:
  - implies set membership

    e.g., $n = O(n^2)$ means $n \in O(n^2)$

- Asymptotic notation appears on the RHS of an equation
  - stands for some anonymous function in the set

    e.g., $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means:

    $2n^2 + 3n + 1 = 2n^2 + h(n)$, for some $h(n) \in \Theta(n)$

    i.e., $h(n) = 3n + 1$

Asymptotic Notation in Equations

- Asymptotic notation appears on the LHS of an equation:
  - stands for any anonymous function in the set

    e.g., $2n^2 + \Theta(n) = \Theta(n^2)$ means:

    for any function $g(n) \in \Theta(n)$

    $\exists$ some function $h(n) \in \Theta(n^2)$

    such that $2n^2+g(n) = h(n)$

- RHS provides coarser level of detail than LHS

Algorithms I

Solving Recurrences

Solving Recurrences

- Reminder: Runtime (T(n)) of *MergeSort* was expressed as a recurrence

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

- Solving recurrences is like solving differential equations, integrals, etc.
    - *Need to learn a few tricks*

Recurrences

- Recurrence: *An equation or inequality that describes a function in terms of its value on smaller inputs.*

Example:

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

Recurrence - Example

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

- *Simplification: Assume $n = 2^k$*
- Claimed answer: $T(n) = \lg n + 1$
- Substitute claimed answer in the recurrence:

$$\lg n + 1 = \begin{cases} 1 & \text{if } n = 1 \\ (\lg(\lceil n/2 \rceil) + 2) & \text{if } n > 1 \end{cases}$$

True when $n = 2^k$

Technicalities: Floor/Ceiling

- Technically, should be careful about the floor and ceiling functions (as in the book).
- e.g. For merge sort, the recurrence should in fact be:

$$T(n) = \begin{cases} \Theta(1) & \text{if n = 1} \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if n > 1} \end{cases}$$

- But, it's usually ok to:
  - ignore floor/ceiling
  - solve for exact powers of 2 (or another number)

Technicalities: Boundary Conditions

- Usually assume: $T(n) = \Theta(1)$ for sufficiently small n
  - Changes the exact solution, but usually the asymptotic solution is not affected (e.g. if polynomially bounded)

- For convenience, the boundary conditions generally implicitly stated in a recurrence

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

  assuming that

  $T(n) = \Theta(1)$ for sufficiently small n

Example: When Boundary Conditions Matter

- Exponential function: $T(n) = (T(n/2))^2$
- Assume $T(1) = c$ (where c is a positive constant).

  $T(2) = (T(1))^2 = c^2$

  $T(4) = (T(2))^2 = c^4$

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

- e.g. $\left.\begin{array}{l} T(1) = 2 \Rightarrow T(n) = \Theta(2^n) \\ T(1) = 3 \Rightarrow T(n) = \Theta(3^n) \end{array}\right\} \textit{However} \quad \Theta(2^n) \neq \Theta(3^n)$

- Difference in solution more dramatic when:

$$T(1) = 1 \Rightarrow T(n) = \Theta(1^n) = \Theta(1)$$

Solving Recurrences

- We will focus on 3 techniques in this lecture:

1. Substitution method

1. Recursion tree approach

1. Master method

Substitution Method

- The most general method:
  1. Guess
  2. Prove by induction
  3. Solve for constants

Substitution Method: Example

Solve $T(n) = 4T(n/2) + n$ *(assume $T(1) = \Theta(1)$)*

1. Guess $T(n) = O(n^3)$ (need to prove O and Ω separately)

2. Prove by induction that $T(n) \leq cn^3$ for large n (i.e. $n \geq n_0$)

Inductive hypothesis: $T(k) \leq ck^3$ for any $k < n$

Assuming ind. hyp. holds, prove $T(n) \leq cn^3$

Substitution Method: Example – cont'd

Original recurrence: $T(n) = 4T(n/2) + n$

From inductive hypothesis: $T(n/2) \leq c(n/2)^3$

Substitute this into the original recurrence:

$$\begin{aligned} T(n) &\leq 4c\,(n/2)^3 + n \\ &= (c/2)\,n^3 + n \\ &= cn^3 - \underbrace{((c/2)n^3 - n)}_{\text{desired - residual}} \\ &\leq cn^3 \end{aligned}$$

when $((c/2)n^3 - n) \geq 0$

Substitution Method: Example – cont'd

- So far, we have shown:

  $T(n) \leq cn^3$ when $((c/2)n^3 - n) \geq 0$

- We can choose $c \geq 2$ and $n_0 \geq 1$
- But, the proof is not complete yet.
- Reminder: Proof by induction:
  1. Prove the base cases $\longrightarrow$ *haven't proved the base cases yet*
  2. Inductive hypothesis for smaller sizes
  3. Prove the general case

Substitution Method: Example – cont'd

- We need to prove the base cases

Base: $T(n) = \Theta(1)$ for small n (e.g. for $n = n_0$)

- We should show that:

  "$\Theta(1)$" $\leq cn^3$ for $n = n_0$

  This holds if we pick c big enough

- So, the proof of $T(n) = O(n^3)$ is complete.
- But, is this a tight bound?

Example: A tighter upper bound?

- Original recurrence: $T(n) = 4T(n/2) + n$
- Try to prove that $T(n) = O(n^2)$,

  i.e. $T(n) \leq cn^2$ for all $n \geq n_0$

- Ind. hyp: Assume that $T(k) \leq ck^2$ for $k < n$
- Prove the general case: $T(n) \leq cn^2$

Example (cont'd)

- Original recurrence: $T(n) = 4T(n/2) + n$
- Ind. hyp: Assume that $T(k) \leq ck^2$ for $k < n$
- Prove the general case: $T(n) \leq cn^2$

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

Wrong! We must prove exactly

Example (cont'd)

- Original recurrence: $T(n) = 4T(n/2) + n$
- Ind. hyp: Assume that $T(k) \leq ck^2$ for $k < n$
- Prove the general case: $T(n) \leq cn^2$

- So far, we have:

$T(n) \leq cn^2 + n$

No matter which positive c value we choose,

this does not show that $T(n) \leq cn^2$

Proof failed?

Example (cont'd)

- What was the problem?
  - *The inductive hypothesis was not strong enough*
- Idea: Start with a stronger inductive hypothesis
  - *Subtract* a low-order term
- Inductive hypothesis: $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$
- Prove the general case: $T(n) \leq c_1 n^2 - c_2 n$

Example (cont'd)

- Original recurrence: $T(n) = 4T(n/2) + n$
- Ind. hyp: Assume that $T(k) \leq c_1k^2 - c_2k$ for $k < n$
- Prove the general case: $T(n) \leq c_1n^2 - c_2n$

$$\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\leq 4\,(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1n^2 - 2c_2n + n \\
&= c_1n^2 - c_2n - (c_2n - n) \\
&\leq c_1n^2 - c_2n \qquad \text{for } n(c_2n - 1) \geq 0
\end{aligned}$$

choose $c_2 \geq 1$

Example (cont'd)

- We now need to prove

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

  for the base cases.

  $T(n) = \Theta(1)$ for $1 \le n \le n_0$ (implicit assumption)

  "$\Theta(1)$" $\le c_1 n^2 - c_2 n$ for n small enough (e.g. $n = n_0$)

  We can choose c_1 large enough to make this hold

- We have proved that $T(n) = O(n^2)$

Substitution Method: Example 2

- For the recurrence $T(n) = 4T(n/2) + n$,

  prove that $T(n) = \Omega(n^2)$

  i.e. $T(n) \geq cn^2$ for any $n \geq n_0$
- Ind. hyp: $T(k) \geq ck^2$ for any $k < n$
- Prove general case: $T(n) \geq cn^2$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\geq 4c\,(n/2)^2 + n \\
&= cn^2 + n \\
&\geq cn^2 \qquad \text{since } n > 0
\end{aligned}
$$

Proof succeeded – no need to strengthen the ind. hyp as in the last example

Example 2 (cont'd)

- We now need to prove that

  $T(n) \geq cn^2$

  for the base cases

  $T(n) = \Theta(1)$ for $1 \leq n \leq n_0$ (implicit assumption)

  "$\Theta(1)$" $\geq cn^2$ for $n = n_0$

  n_0 is sufficiently small (i.e. constant)

  We can choose c small enough for this to hold

- We have proved that $T(n) = \Omega(n^2)$

Substitution Method - Summary

1. Guess the asymptotic complexity

1. Prove your guess using induction
   1. Assume inductive hypothesis holds for $k < n$
   2. Try to prove the general case for n

      Note: MUST prove the EXACT inequality

      CANNOT ignore lower order terms

      If the proof fails, strengthen the ind. hyp. and try again
   3. Prove the base cases (usually straightforward)

Recursion Tree Method

- A recursion tree models the runtime costs of a recursive execution of an algorithm.
- The recursion tree method is good for generating guesses for the substitution method.
- The recursion-tree method can be unreliable.
    - Not suitable for formal proofs
- The recursion-tree method promotes intuition, however.

Solve Recurrence: $T(n) = 2T\ (n/2) + \Theta(n)$

$$\begin{tikzcd}
 & \Theta(n) \arrow[dl] \arrow[dr] & \\
T(n/2) & & T(n/2)
\end{tikzcd}$$

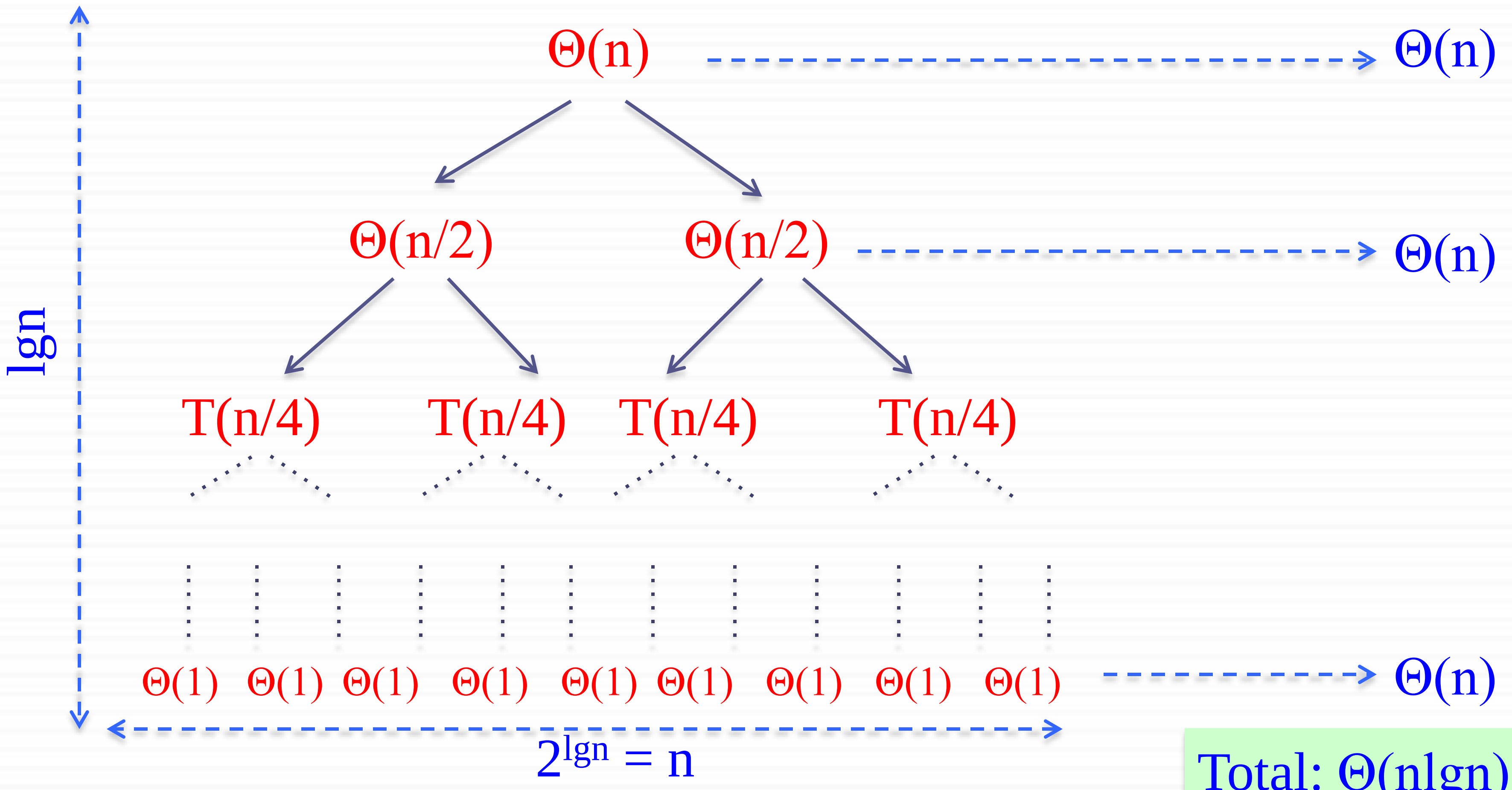
Solve Recurrence: $T(n) = 2T(n/2) + \Theta(n)$

$$\begin{tikzcd}
 & & & \Theta(n) \arrow[dl] \arrow[dr] & & & \\
 & & \Theta(n/2) \arrow[dl] \arrow[d] & & \Theta(n/2) \arrow[d] \arrow[dr] & & \\
 & T(n/4) & T(n/4) & & T(n/4) & T(n/4) &
\end{tikzcd}$$

2x subprobs $\downarrow$

$\downarrow$ each size halved

Solve Recurrence: $T(n) = 2T(n/2) + \Theta(n)$



Total: $\Theta(n\lg n)$

Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

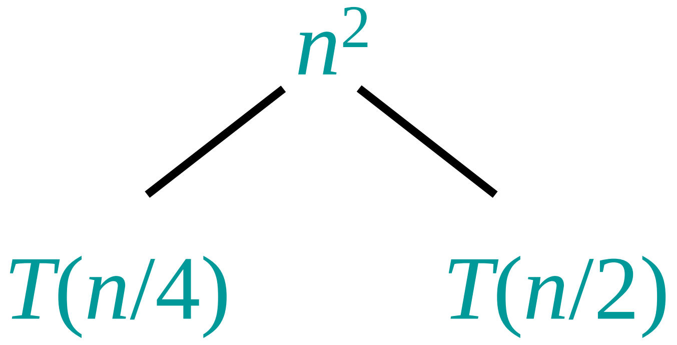
Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$T(n)$$

Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



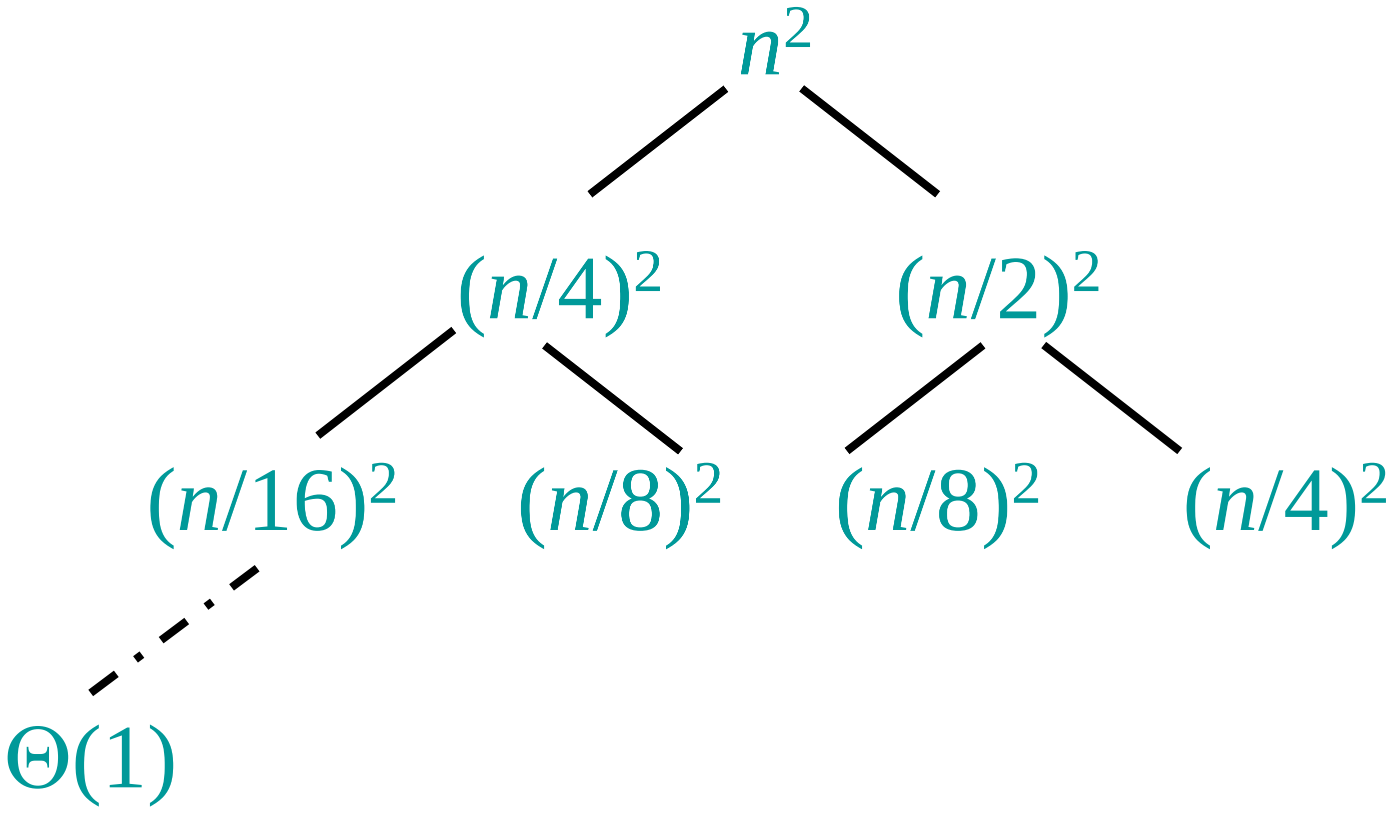
Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$\begin{tikzcd}
 & & & n^2 \arrow[dll, dash] \arrow[drr, dash] & & & \\
 & (n/4)^2 \arrow[dl, dash] \arrow[dr, dash] & & & & (n/2)^2 \arrow[dl, dash] \arrow[dr, dash] & \\
\mathrm{T}(n/16) & & \mathrm{T}(n/8) & & \mathrm{T}(n/8) & & \mathrm{T}(n/4)
\end{tikzcd}$$

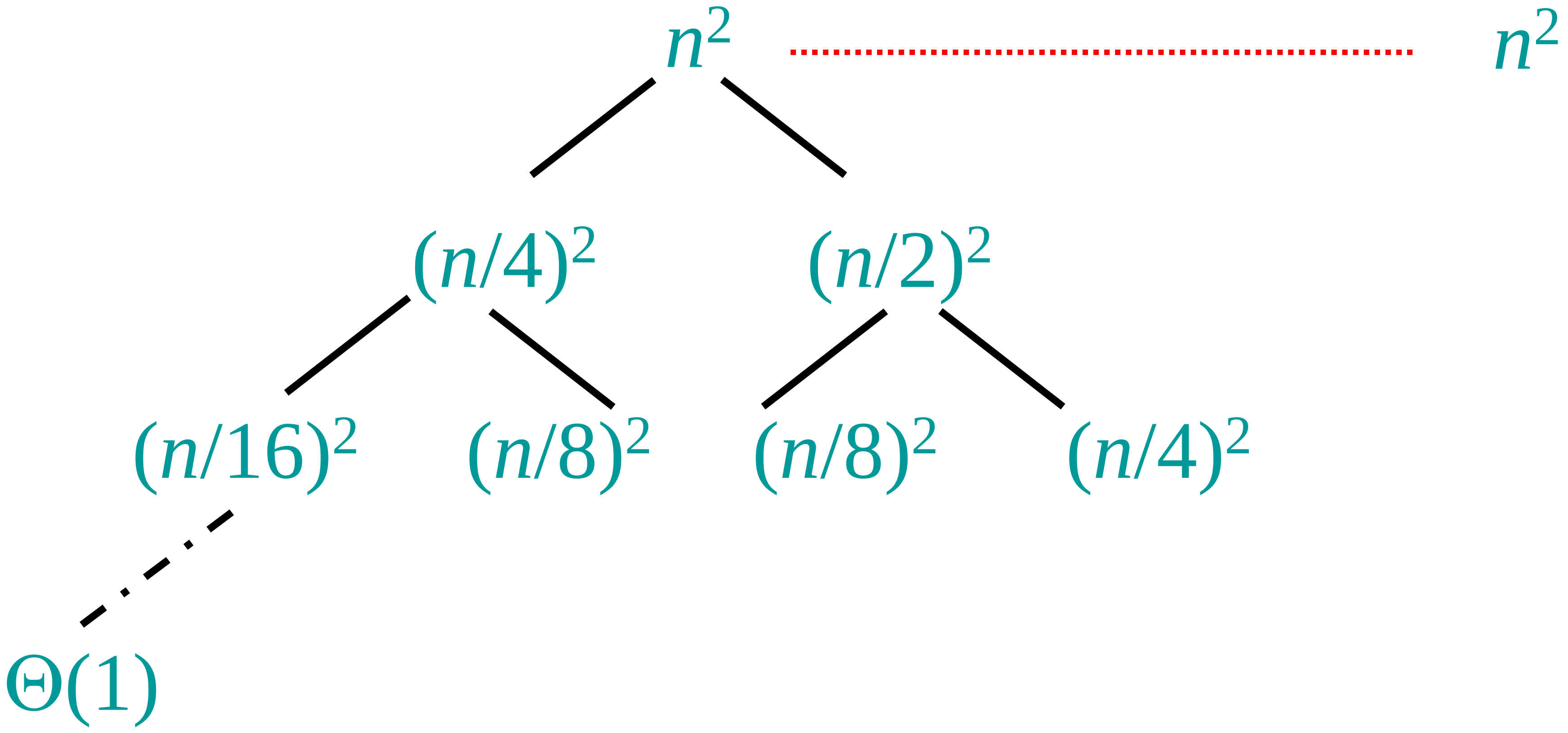
Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



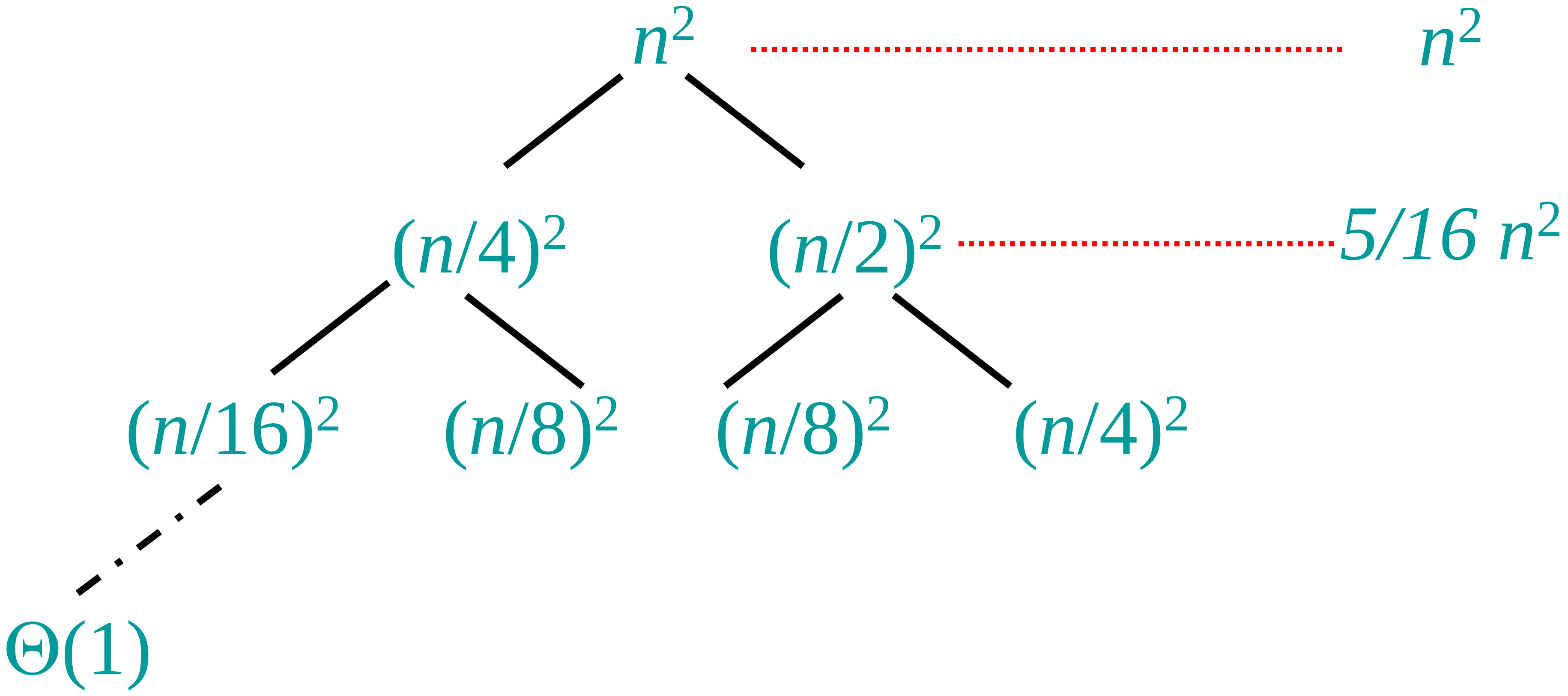
Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



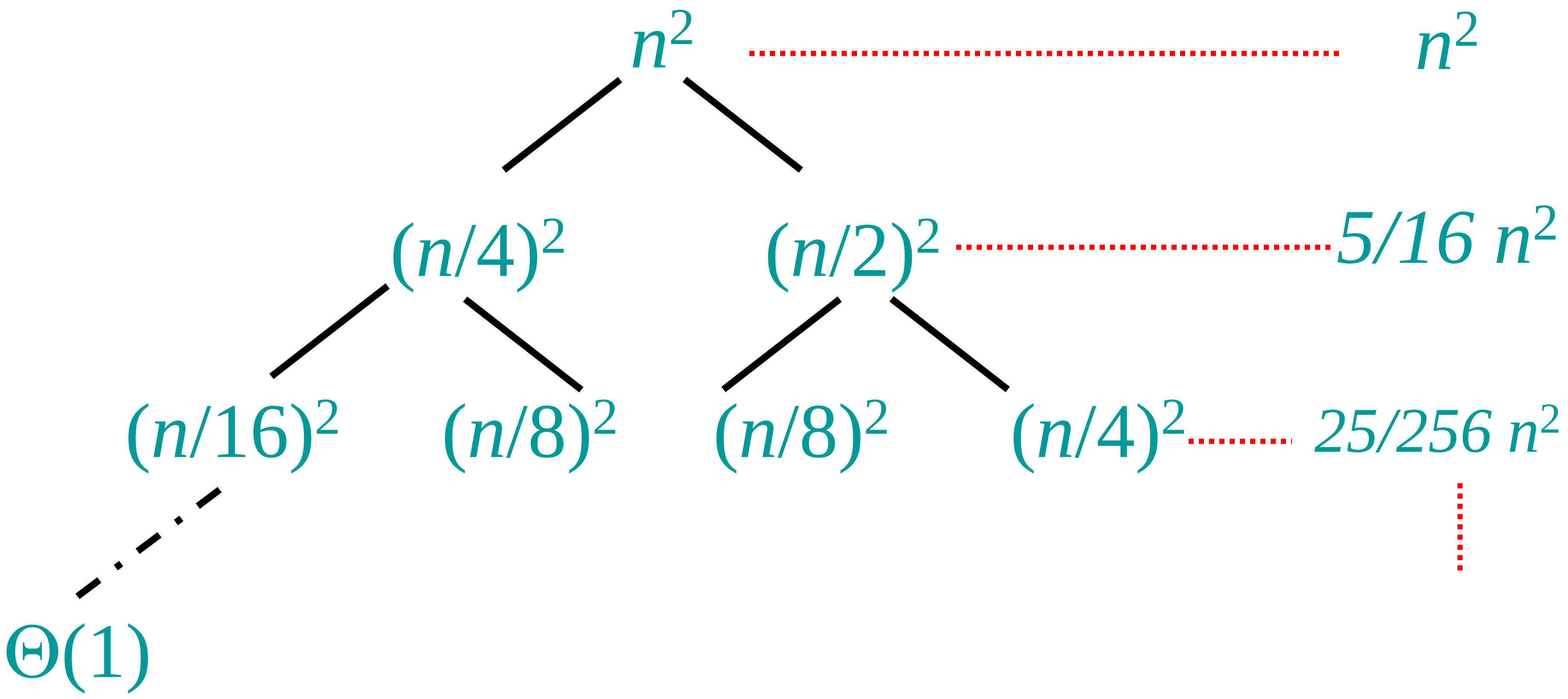
Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



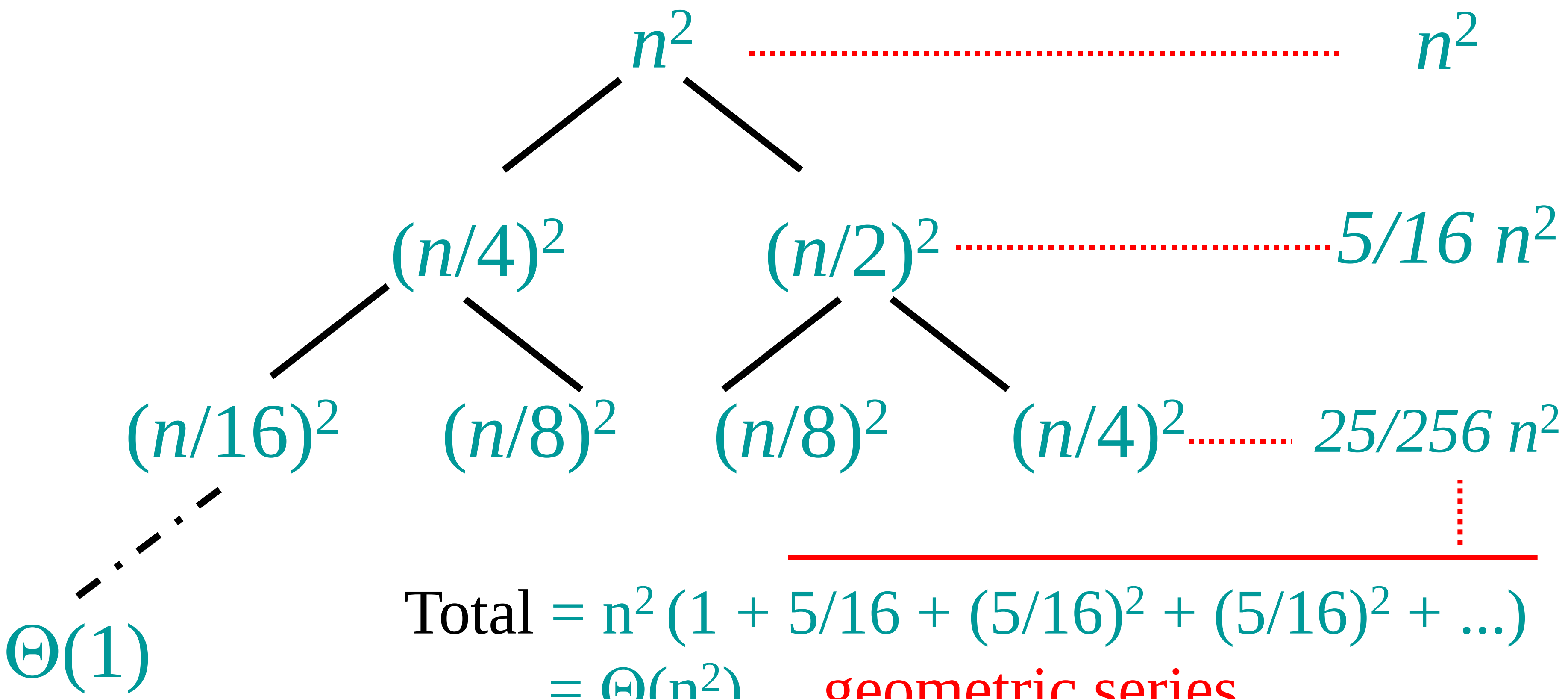
Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of Recursion Tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Total $= n^2 (1 + 5/16 + (5/16)^2 + (5/16)^2 + \ldots)$

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

The Master Method

- A powerful black-box method to solve recurrences.

- 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: 3 Cases

- Recurrence: $T(n) = aT(n/b) + f(n)$

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

Case 1: $f(n)$ grows *polynomially slower* than $n^{\log_b a}$

Case 2: $f(n)$ grows *at the same rate* as $n^{\log_b a}$

Case 3: $f(n)$ grows *polynomially faster* than $n^{\log_b a}$

The Master Method: Case 1

- Recurrence: $T(n) = aT(n/b) + f(n)$

<u>Case 1</u>: $$\frac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon})$$ for some constant $\varepsilon > 0$

i.e., $f(n)$ grows polynomialy slower than $n^{\log_b a}$

(by an n^{ε} factor).

<u>Solution</u>: $T(n) = \Theta(n^{\log_b a})$

The Master Method: Case 2 (simple version)

- Recurrence: $T(n) = aT(n/b) + f(n)$

Case 2: $\dfrac{f(n)}{n^{\log_b a}} = \Theta(1)$

i.e., $f(n)$ and $n^{\log_b a}$ grow at similar rates

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

The Master Method: Case 3

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

i.e., $f(n)$ grows polynomialy faster than $n^{\log_b a}$ (by an n^{ε} factor).

and the following regularity condition holds:

$a\, f(n/b) \leq c\, f(n)$ for some constant $c < 1$

Solution: $T(n) = \Theta(\, f(n)\,)$

Example: T(n) = 4T(n/2) + n

a = 4

b = 2

f(n) = n

$$n^{\log_b a} = n^2$$

$\Rightarrow$

f(n) grows *polynomially* slower than $n^{\log_b a}$

$$\frac{n^{\log_b a}}{f(n)} = \frac{n^2}{n} = n = \Omega(n^{\varepsilon})$$

for ε = 1

$\Rightarrow$ CASE 1

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

$$\boxed{T(n) = \Theta(n^2)}$$

Example: $T(n) = 4T(n/2) + n^2$

$a = 4$

$b = 2$

$f(n) = n^2$

$n^{\log_b a} = n^2$

$\Rightarrow$

f(n) grows at similar rate as $n^{\log_b a}$

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

$\Rightarrow$ CASE 2

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

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

Example: T(n) = 4T(n/2) + n^3

a = 4

b = 2

f(n) = n^3

$$n^{\log_b a} = n^2$$

$\Rightarrow$ f(n) grows *polynomially* faster than $n^{\log_b a}$

$$\frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n^2} = n = \Omega(n^{\varepsilon})$$

for ε = 1

$\Rightarrow$ seems like CASE 3, but need to check the regularity condition

$\Rightarrow$ Regularity condition: $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$

$\Rightarrow$ $4(n/2)^3 \le cn^3$ for c = 1/2

$\Rightarrow$ CASE 3 $\Rightarrow$ T(n) = Θ(f(n)) $\Rightarrow$ T(n) = Θ(n^3)

Example: $T(n) = 4T(n/2) + n^2/\lg n$

$a = 4$

$b = 2$

$f(n) = n^2/\lg n$

$$n^{\log_b a} = n^2$$

$\Rightarrow$ f(n) grows slower than $n^{\log_b a}$

but is it polynomially slower?

$$\frac{n^{\log_b a}}{f(n)} = \frac{n^2}{\frac{n^2}{\lg n}} = \lg n \neq \Omega(n^{\varepsilon})$$

for any ε > 0

$\Rightarrow$ is **<u>not</u>** CASE 1

$\Rightarrow$ Master method does not apply!

The Master Method: Case 2 (general version)

- Recurrence: $T(n) = aT(n/b) + f(n)$

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

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

General Method (Akra-Bazzi)

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

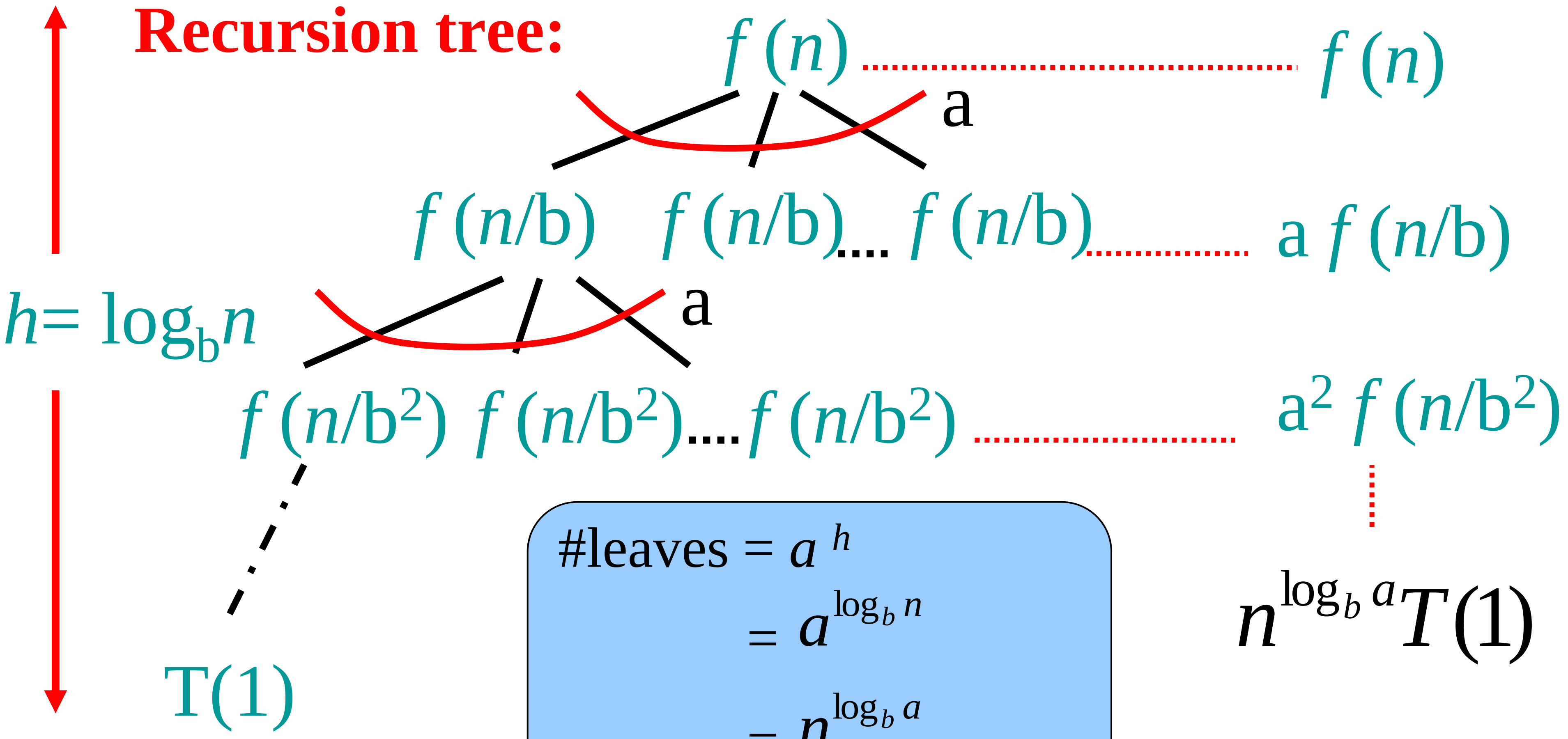
Let p be the unique solution to

$$\sum_{i=1}^{k} (a_i / b^p{}_i) = 1$$

Then, the answers are the same as for the master method, but with n^p instead of $n^{\log_b a}$

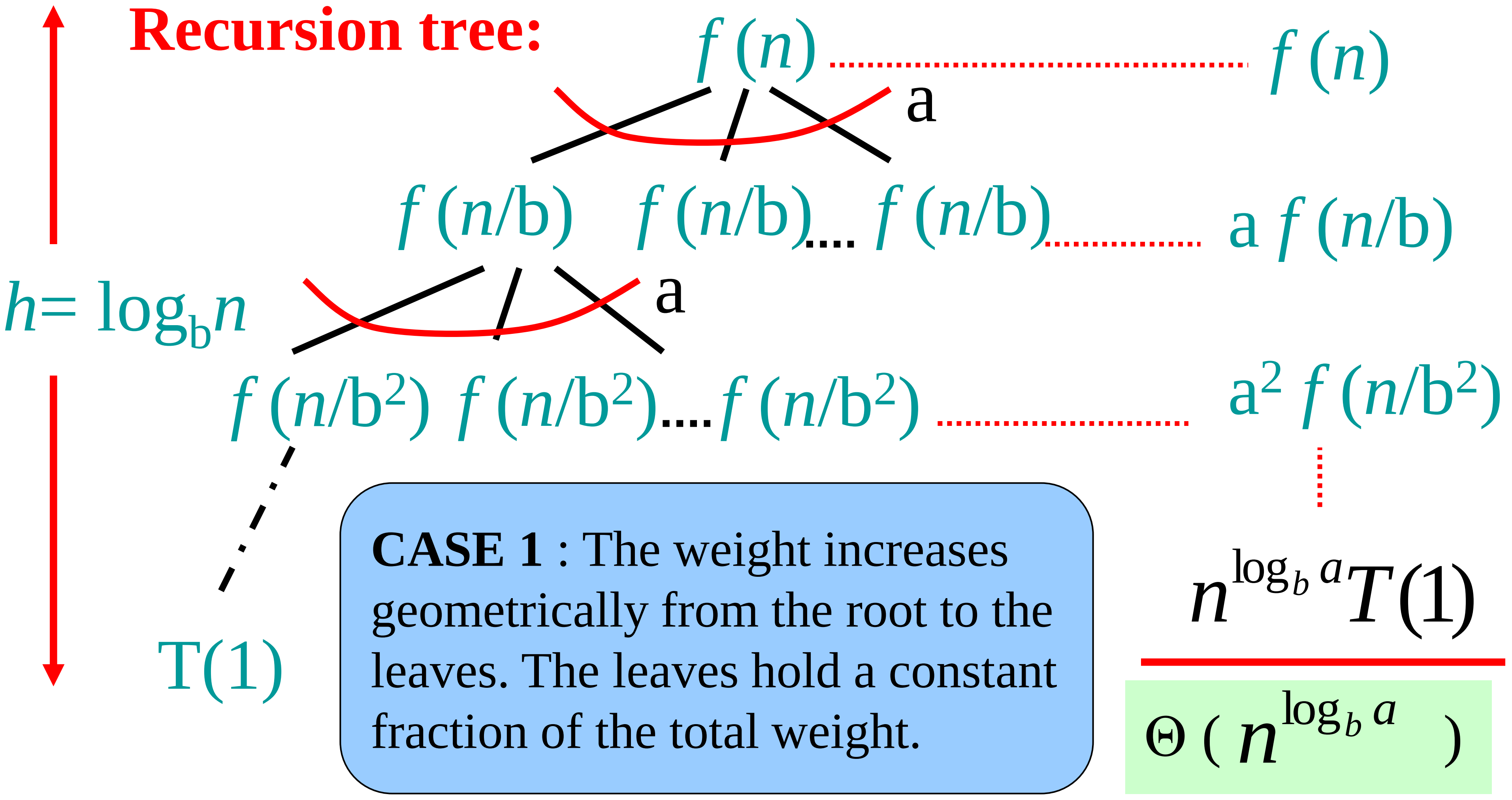
(*Akra and Bazzi also prove an even more general result.*)

Idea of Master Theorem



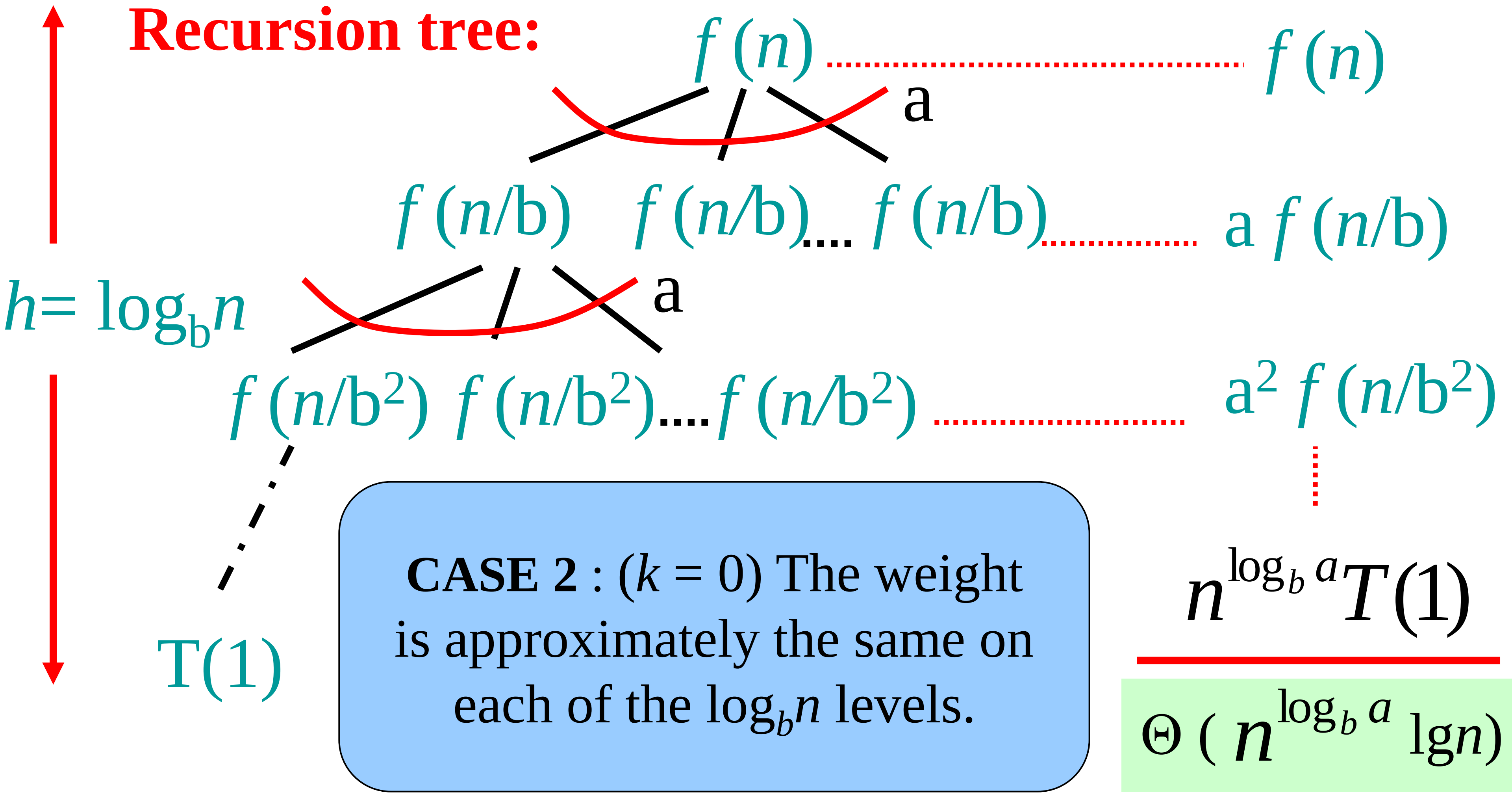
$$\#\text{leaves} = a^h = a^{\log_b n} = n^{\log_b a}$$

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.

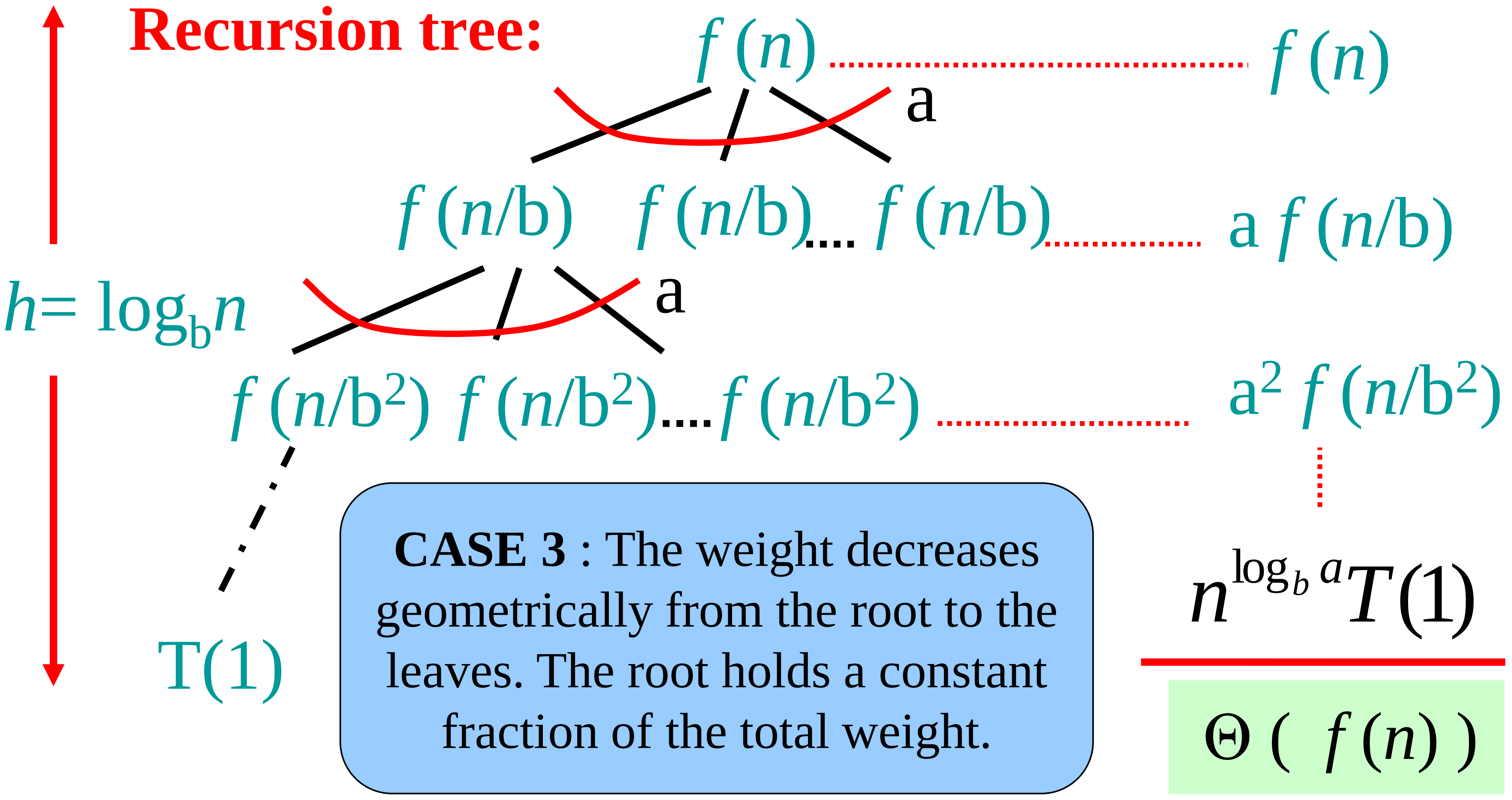
Idea of Master Theorem



CASE 2 : ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

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

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: Case 1 and Case 2

- Recall from the recursion tree (note $h = \lg_b n$=tree height)

$$T(n) = \underbrace{\Theta(n^{\log_b a})}_{\text{Leaf cost}} + \underbrace{\sum_{i=0}^{h-1} a^i f(n/b^i)}_{\text{Non-leaf cost} = g(n)}$$

Proof of Case 1

- $\dfrac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon})$ for some ε > 0
- $\dfrac{n^{\log_b a}}{f(n)} = \Omega(n^{\varepsilon}) \Rightarrow \dfrac{f(n)}{n^{\log_b a}} = O(n^{-\varepsilon}) \Rightarrow f(n) = O(n^{\log_b a-\varepsilon})$
- $g(n) = \sum_{i=0}^{h-1} a^i O\left((n/b^i)^{\log_b a-\varepsilon}\right) = O\left(\sum_{i=0}^{h-1} a^i (n/b^i)^{\log_b a-\varepsilon}\right)$
- $= O\left(n^{\log_b a-\varepsilon} \sum_{i=0}^{h-1} a^i b^{i\varepsilon} / b^{i \log_b a}\right)$

Case 1 (cont')

$$\sum_{i=0}^{h-1} \frac{a^i b^{i\varepsilon}}{b^{i\log_b a}} = \sum_{i=0}^{h-1} a^i \frac{(b^\varepsilon)^i}{(b^{\log_b a})^i} = \sum a^i \frac{b^{\varepsilon i}}{a^i} = \sum_{i=0}^{h-1} (b^\varepsilon)^i$$

= An increasing geometric series since b > 1

$$= \frac{b^{\varepsilon h} - 1}{b^\varepsilon - 1} = \frac{(b^h)^\varepsilon - 1}{b^\varepsilon - 1} = \frac{(b^{\log_b n})^\varepsilon - 1}{b^\varepsilon - 1} = \frac{n^\varepsilon - 1}{b^\varepsilon - 1} = O(n^\varepsilon)$$

Case 1 (cont')

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

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

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

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

Q.E.D.

Proof of Case 2 (limited to *k=0*)

- $$\frac{f(n)}{n^{\log_b a}} = \Theta(\lg^0 n) = \Theta(1) \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow f(n/b^i) = \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b a} \right)$$
- $$\therefore g(n) = \sum_{i=0}^{h-1} a^i \Theta\left((n/b^i)^{\log_b a} \right)$$

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

$$= \Theta\left(n^{\log_b a} \sum_{i=0}^{\log_b n - 1} 1 \right) = \Theta\left(n^{\log_b a} \log_b n \right) = \Theta\left(n^{\log_b a} \lg n \right)$$

- $$T(n) = n^{\log_b a} + \Theta(n^{\log_b a} \lg n)$$
  $$= \Theta\left(n^{\log_b a} \lg n \right)$$

Q.E.D.