

Solving recurrences

- Recurrences are a major tool for **analysis** of algorithms.
 - Divide and Conquer algorithms are analyzable by recurrences.
- Also note, recurrence relations are used to **design** algorithms for combinatorial functions and can embody dynamic programming idea.

Methods for Solving Recurrences

- Using Substitution and Mathematical Induction
- Using Recursion-tree
- Using Master Theorem

Example: Integer Multiplication

- Let $X = \boxed{A} \boxed{B}$ and $Y = \boxed{C} \boxed{D}$ where A, B, C and D are $n/2$ bit integers
- **Simple Method:** $XY = (2^{n/2}A + B)(2^{n/2}C + D)$
- **Running Time Recurrence**

$$T(n) < 4T(n/2) + 100n$$

How do we solve it?

Substitution method

The most general method:

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

Example: $T(n) = 4T(n/2) + 100n$

- [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

$$\begin{aligned}T(n) &= 4T(n/2) + 100n \\&\leq 4c(n/2)^3 + 100n \\&= (c/2)n^3 + 100n \\&= cn^3 - ((c/2)n^3 - 100n) \quad \leftarrow \text{desired} - \text{residual} \\&\leq cn^3 \quad \leftarrow \text{desired}\end{aligned}$$

whenever $(c/2)n^3 - 100n \geq 0$, for
example, if $c \geq 200$ and $n \geq 1$. residual

A tighter upper bound?

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

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

$$\begin{aligned} T(n) &= 4T(n/2) + 100n \\ &\leq cn^2 + 100n \\ &\leq cn^2 \end{aligned}$$

for **no** choice of $c > 0$. Stuck?

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 $k < n$.

$$\begin{aligned} T(n) &= 4T(n/2) + 100n \\ &\leq 4(c_1(n/2)^2 - c_2(n/2)) + 100n \\ &= c_1 n^2 - 2c_2 n + 100n \\ &= c_1 n^2 - c_2 n - (c_2 n - 100n) \\ &\leq c_1 n^2 - c_2 n \quad \text{if } c_2 > 100. \end{aligned}$$

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 is good for generating guesses for the substitution method.
 - The recursion-tree method can be unreliable, just like any method that uses ellipses (...).
 - The recursion-tree method promotes intuition, however.

Example of recursion tree

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

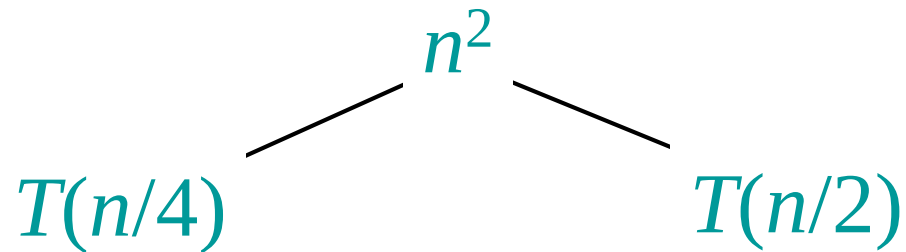
Example of recursion tree

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

$$T(n)$$

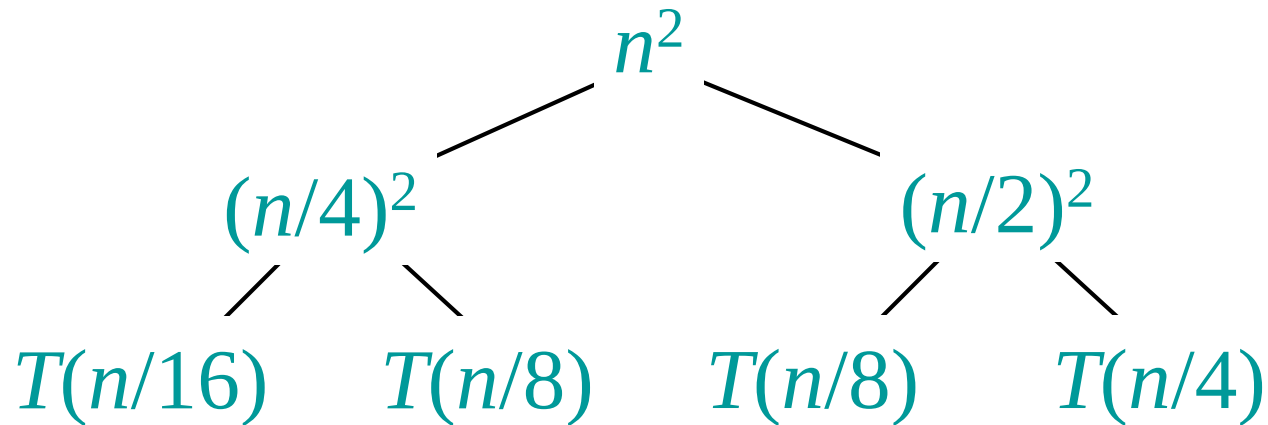
Example of recursion tree

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



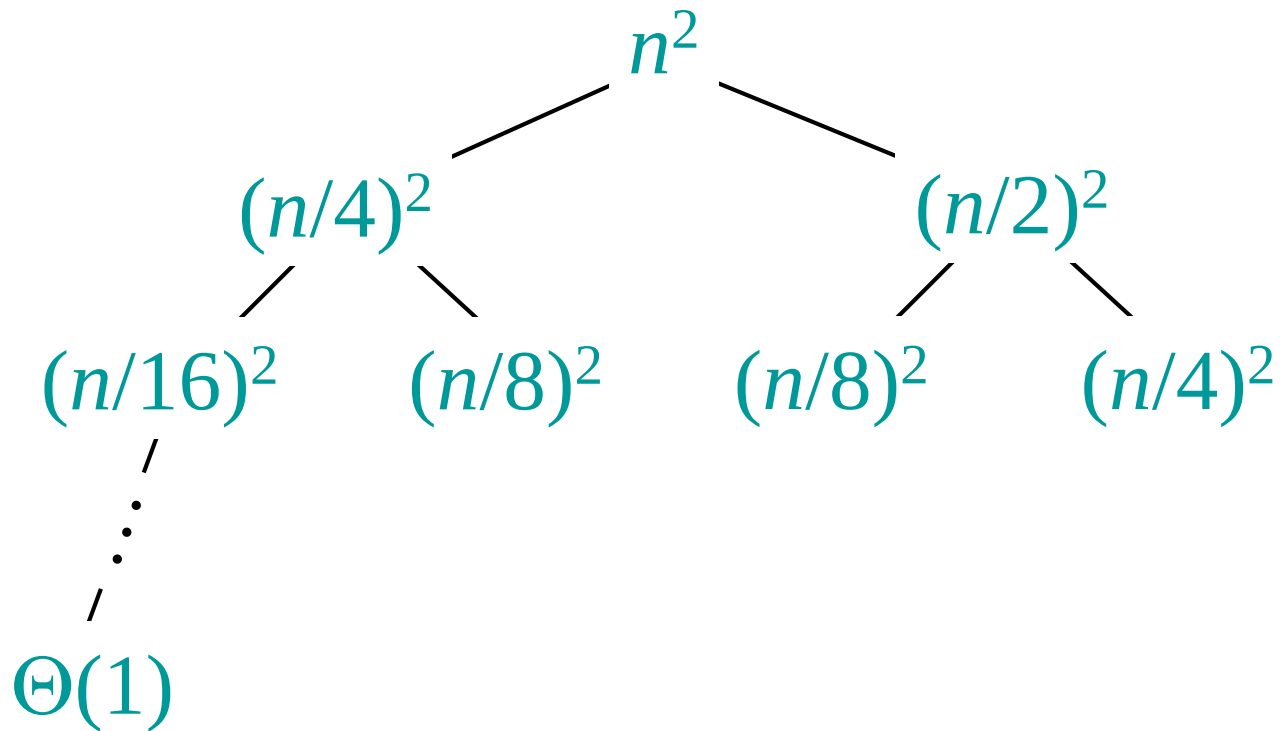
Example of recursion tree

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



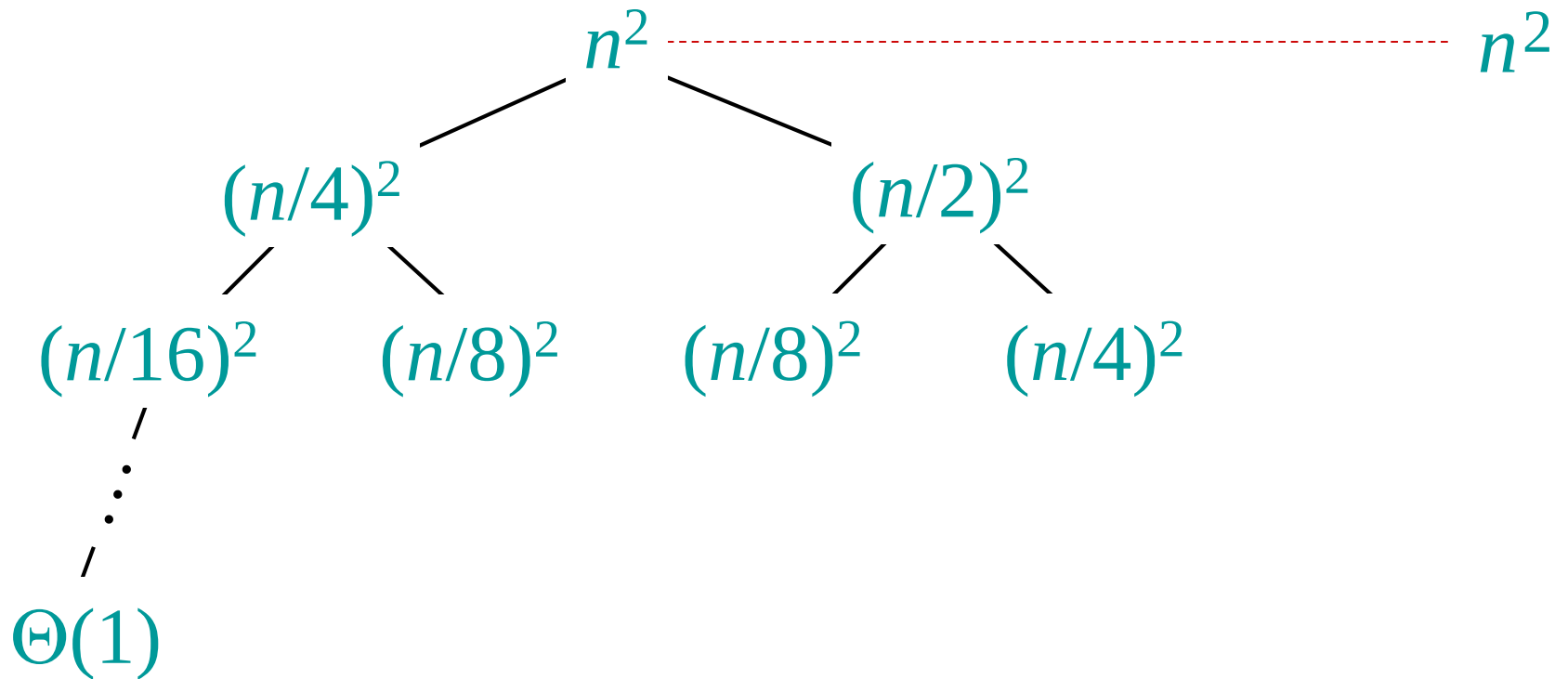
Example of recursion tree

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



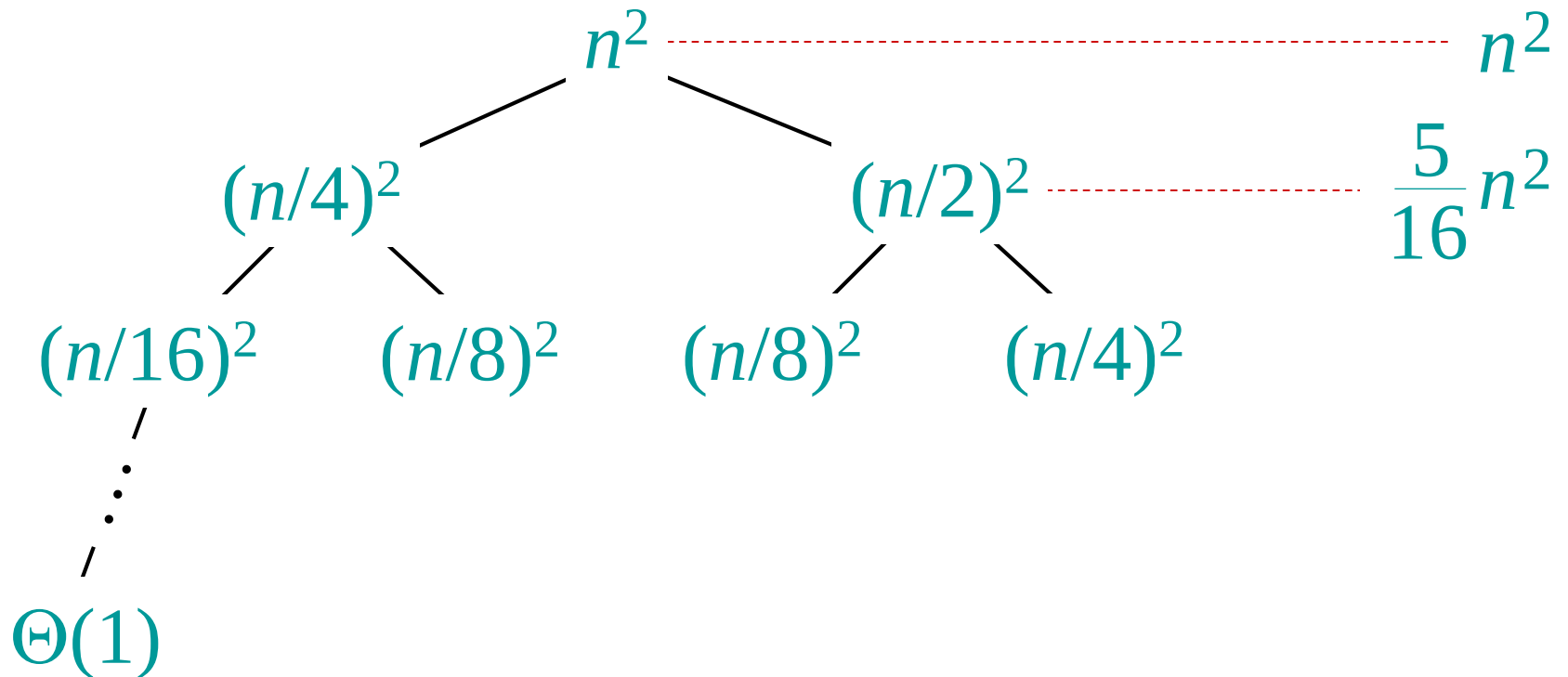
Example of recursion tree

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



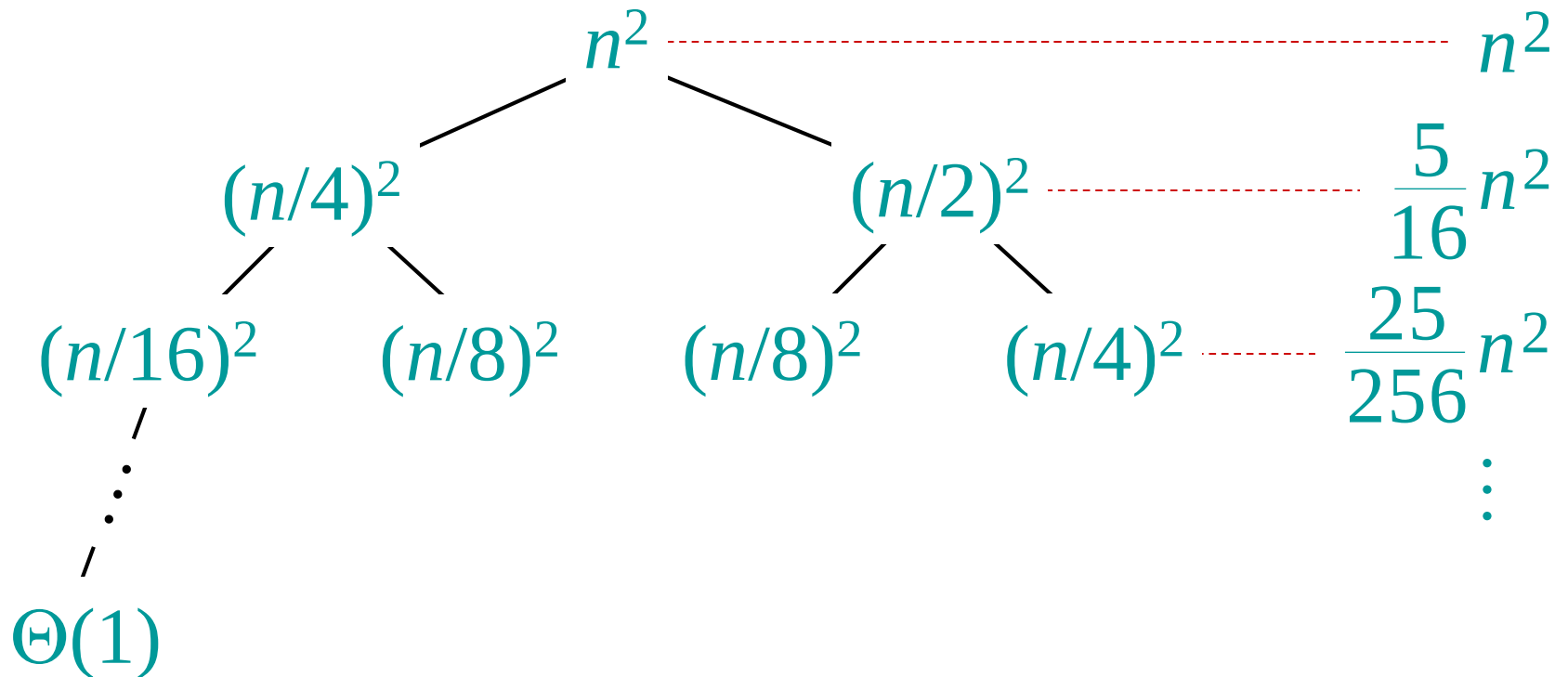
Example of recursion tree

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



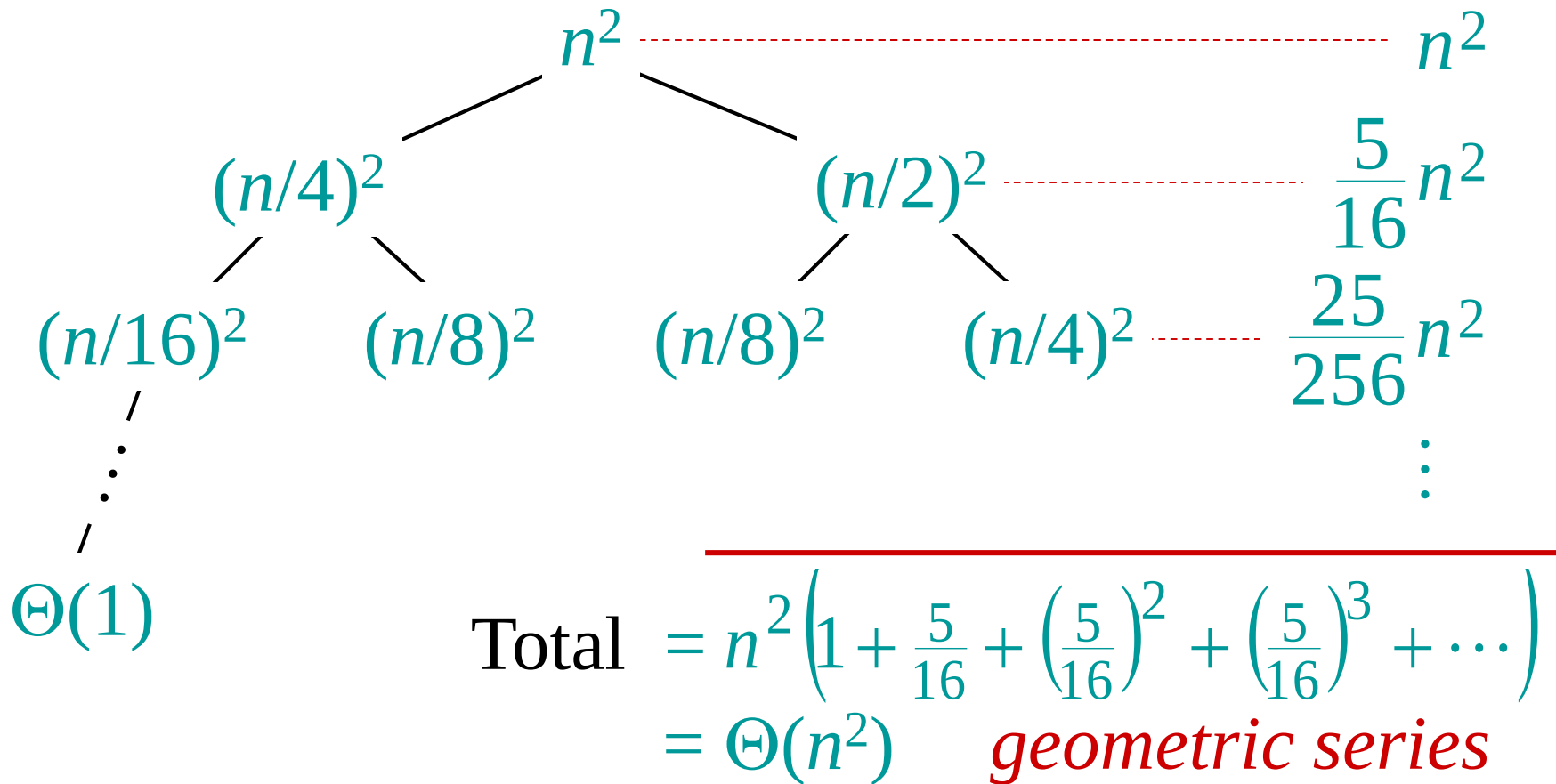
Example of recursion tree

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



Example of recursion tree

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



Appendix: geometric series

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

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

Recursion Tree of $T(n)=T(n/3)+ T(2n/3)+O(n)$

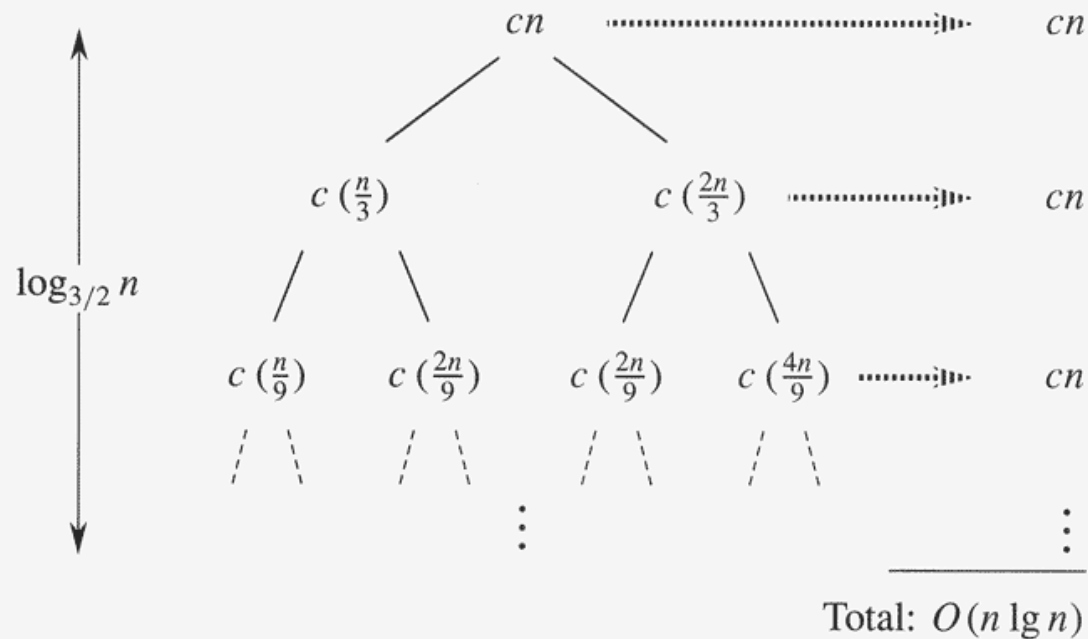


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Master Theorem

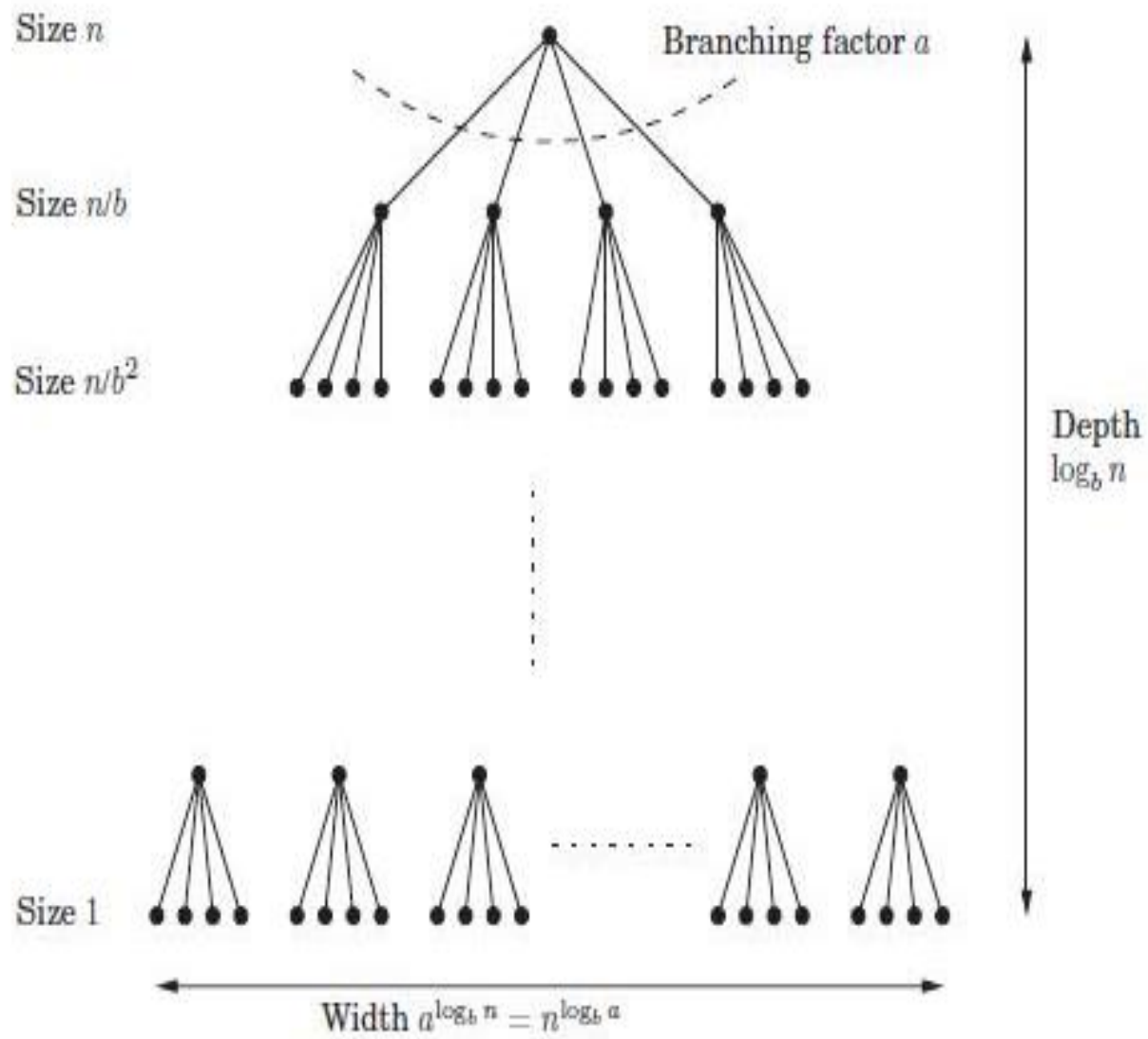
- Let $T(n)$ be a monotonically increasing function that satisfies

$$T(n) = a T(n/b) + f(n)$$

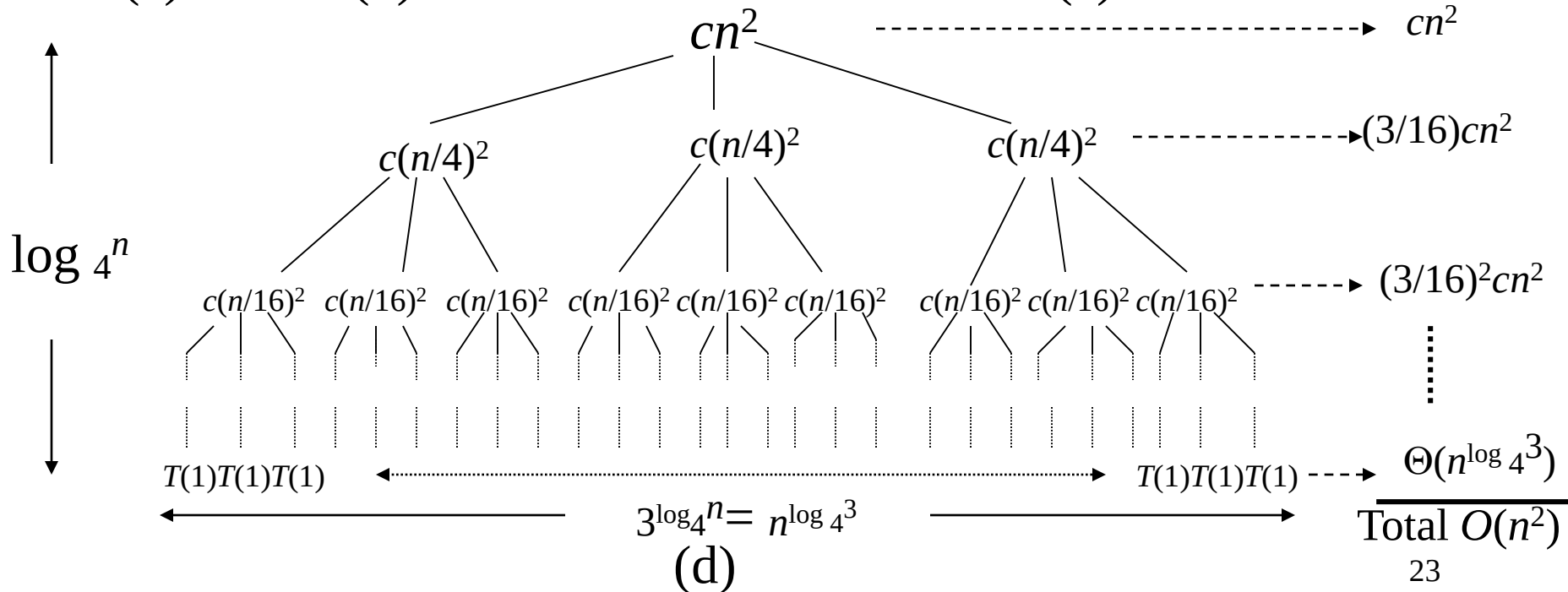
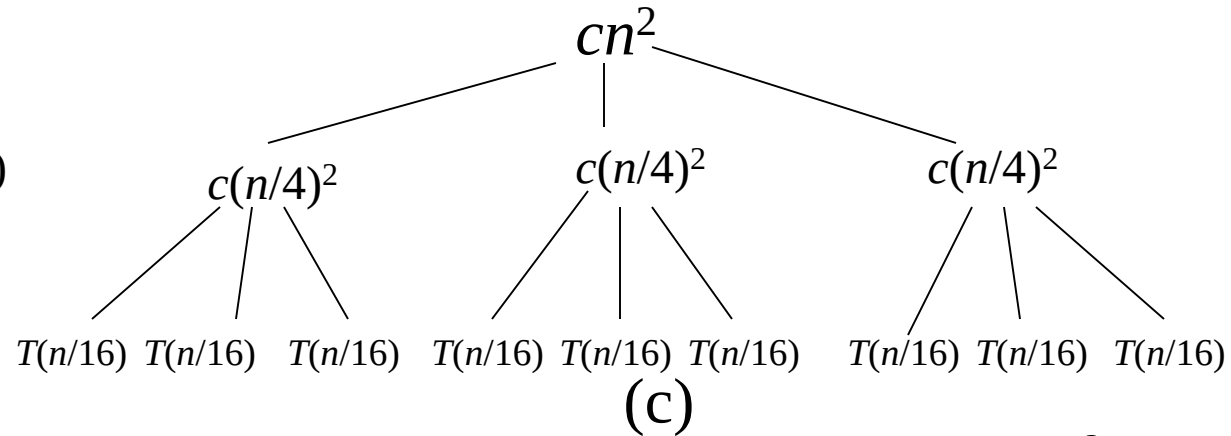
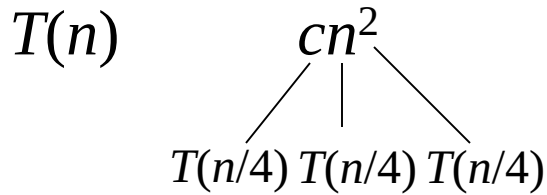
$$T(1) = c$$

where $a \geq 1$, $b \geq 2$, $c > 0$. If $f(n)$ is $\Theta(n^d)$ where $d \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } \log_b a < d \text{ or } a < b^d \\ \Theta(n^d \log n) & \text{if } \log_b a = d \text{ or } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } \log_b a > d \text{ or } a > b^d \end{cases}$$



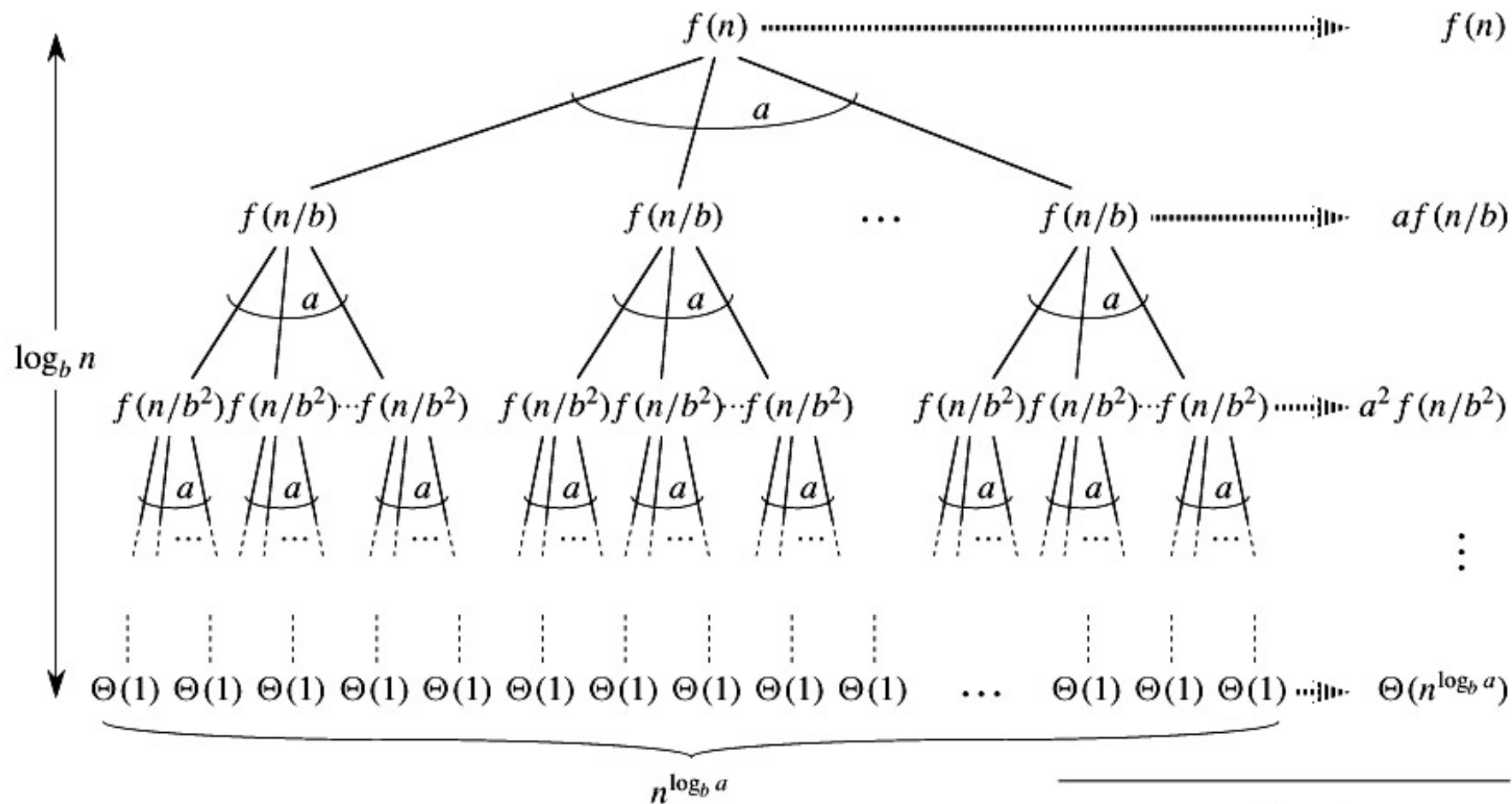
Recursion Tree for $T(n)=3T(\lfloor n/4 \rfloor)+\Theta(n^2)$



Solution to $T(n)=3T(\lfloor n/4 \rfloor)+\Theta(n^2)$

- The height is $\log_4 n$,
- #leaf nodes = $3^{\log_4 n} = n^{\log_4 3}$. Leaf node cost: $T(1)$.
- Total cost $T(n)=cn^2+(3/16)cn^2+(3/16)^2cn^2+\dots+(3/16)^{\log_4 n-1}cn^2+\Theta(n^{\log_4 3})$
 $= (1+3/16+(3/16)^2+\dots+(3/16)^{\log_4 n-1})cn^2+\Theta(n^{\log_4 3})$
 $< (1+3/16+(3/16)^2+\dots+(3/16)^m+\dots)cn^2+\Theta(n^{\log_4 3})$
 $= (1/(1-3/16))cn^2+\Theta(n^{\log_4 3})$
 $= 16/13cn^2+\Theta(n^{\log_4 3})$
 $= O(n^2).$


$$T(n)=3T(\lfloor n/4 \rfloor)+\Theta(n^2)$$



Basis problems cost

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

Divide and combine cost

Master Theorem: Pitfalls

- You **cannot** use the Master Theorem if
 - $T(n)$ is not monotone, e.g. $T(n) = \sin(n)$
 - $f(n)$ is not a polynomial, e.g., $T(n) = 2T(n/2) + 2^n$
 - b cannot be expressed as a constant, e.g.

$$T(n) = T(\sqrt{n})$$

Master Theorem: Example 1

- Let $T(n) = T(n/2) + \frac{1}{2} n^2 + n$. What are the parameters?

$$a = 1$$

$$b = 2$$

$$d = 2$$

Therefore, which condition applies?

$$0 < 2 \quad \text{or} \quad 1 < 2^2, \quad \text{case 1 applies} \quad (\text{if } \log_b a < d \text{ or } a < b^d)$$

- We conclude that

$$T(n) \in \Theta(n^d) = \Theta(n^2)$$

Master Theorem: Example 2

- Let $T(n) = 2T(n/4) + \sqrt{n} + 42$. What are the parameters?

$$a = 2$$

$$b = 4$$

$$d = 1/2$$

Therefore, which condition applies?

$$\frac{1}{2} = \frac{1}{2} \quad \text{or} \quad 2 = 4^{1/2}, \text{ case 2 applies (if } \log_b a = d \text{ or } a = b^d)$$

- We conclude that

$$T(n) \in \Theta(n^d \log n) = \Theta(\log n \sqrt{n})$$

Master Theorem: Example 3

- Let $T(n) = 3T(n/2) + 3/4n + 1$. What are the parameters?

$$a = 3$$

$$b = 2$$

$$d = 1$$

Therefore, which condition applies?

$\log_2 3 > 1$ or $3 > 2^1$, case 3 applies (if $\log_b a > d$ or $a > b^d$)

- We conclude that

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$$

- Note that $\log_2 3 \approx 1.584...$, can we say that $T(n) \in \Theta(n^{1.584})$

No, because $\log_2 3 \approx 1.5849...$ and $n^{1.584} \notin \Theta(n^{1.5849})$

‘Fourth’ Condition

- Recall that we cannot use the Master Theorem if $f(n)$, the non-recursive cost, is not a polynomial.
- There is a limited 4th condition of the Master Theorem that allows us to consider polylogarithmic functions.
- **Corollary:** If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then
$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$$
- This final condition is fairly limited and we present it for sake of completeness.

‘Fourth’ Condition: Example

- Say we have the following recurrence relation

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

- Clearly, $a=2$, $b=2$, but $f(n)$ is not a polynomial. However, we have $f(n) \in \Theta(n \log n)$, $k=1$
- Therefore, by the 4th condition of the Master Theorem, we can say that

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n^{\log_2 2} \log^2 n) = \Theta(n \log^2 n)$$

Examples

Ex. $T(n) = 4T(n/2) + n$

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

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

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

Ex. $T(n) = 4T(n/2) + n^2$

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

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

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

Examples

Ex. $T(n) = 4T(n/2) + n^3$

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

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

and $4(cn/2)^3 \leq cn^3$ (reg. cond.) for $c = 1/2$.

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

Ex. $T(n) = 4T(n/2) + n^2/\lg n$

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

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $n^\varepsilon = \Omega(\lg n)$.

Special Cases: Changing Variables

- Suppose $T(n)=2T(\sqrt{n})+\lg n$.
- Rename $m=\lg n$. So $T(2^m)=2T(2^{m/2})+m$.
- Domain transformation:

$$S(m)=T(2^m)$$

$$S(m)=2S(m/2)+m.$$

- So the solution is $S(m)=O(m \lg m)$.
- Changing back to $T(n)$ from $S(m)$, the solution is

$$T(n) = T(2^m) = S(m)$$

$$=O(m \lg m)$$

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

Using Iteration

$$T(n) = T(\sqrt{n}) + 1$$

$$T(2) = \Theta(1)$$

$$T(n) = 1 + T(n^{1/2})$$

$$= 1 + 1 + T(n^{1/4})$$

$$= i + \dots T(n^{1/2^i})$$

$$= \log \log(n) + T(2)$$

$$= \log \log(n) + \Theta(1)$$

$$= \Theta(\log \log(n))$$

until....

$$n^{(1/2^i)} = 2 \quad //(log)$$

$$\Rightarrow 1/2^i \log n = \log 2 = 1$$

$$\Rightarrow 2^i = \log n$$

$$\Rightarrow i = \log \log n$$

Using Iteration:

Master Theorem *does not* apply

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

Would yield the following summation (assuming n is a power of 2):

$$\begin{aligned} T(n) &= \frac{n}{\log n} + 2\frac{\frac{n}{2}}{\log \frac{n}{2}} + 4\frac{\frac{n}{4}}{\log \frac{n}{4}} + \dots \\ &= \frac{n}{\log n} + \frac{n}{\log \frac{n}{2}} + \frac{n}{\log \frac{n}{4}} + \dots + 1 \\ &= n \left[\frac{1}{\log n} + \frac{1}{\log(n) - 1} + \frac{1}{\log(n) - 2} + \dots + 1 \right] \\ &= n \sum_{i=0}^{\log n - 1} \frac{1}{\log(n) - i} \\ &= n \cdot H_{\log n} \end{aligned}$$

Where H_n is the n th [Harmonic number](#). It's [known](#) that $H_n = \Theta(\log n)$.

We can thus conclude

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

Solving: (Second-degree) Linear Homogeneous Recurrence Relations

- **Assumptions**

- Recurrence relation: $a_n = c_1 a_{n-1} + c_2 a_{n-2}$
- Characteristic equation $r^2 - c_1 r - c_2 = 0$ has potentially two **distinct** roots r_1, r_2 .

- **Theorem**

$\{a_n\}$ is a solution of the recurrence $a_n = c_1 a_{n-1} + c_2 a_{n-2}$

if and only if

$\{a_n\}$ is of the form $a_n = b_1 r_1^n + b_2 r_2^n$

for all $n \geq 0$, and for some constants b_1, b_2

Example solution

- $a_n = a_{n-1} + 2a_{n-2},$ $a_0=2$ and $a_1=7$

- Characteristic equation

$$r^2 - r - 2 = 0$$

- The quadratic formula for $ax^2+bx+c = 0,$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Roots: $r_1=(1-(3))/2= -1$ and $r_2=(1+(3))/2 = 2$

Example solution

- Therefore, solving for coefficients b_1, b_2 using initial values a_0, a_1 in

$$a_n = b_1 r_1^n + b_2 r_2^n$$

$$b_2 = \frac{a_1 - a_0 r_1}{r_2 - r_1} = \frac{7 - 2 \cdot (-1)}{2 - (-1)} = \frac{9}{3} = 3$$

$$b_1 = \frac{a_0 r_2 - a_1}{r_2 - r_1} = \frac{2 \cdot 2 - 7}{2 - (-1)} = \frac{-3}{3} = -1$$

$$a_n = b_1 r_1^n + b_2 r_2^n = -1 \cdot (-1)^n + 3 \cdot 2^n = 3 \cdot 2^n - (-1)^n$$

Summary: Master and Muster Theorem

Master Theorem (for divide and conquer recurrences):

Let $T(n)$ be a function defined on positive n , and having the property

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n/b) + f(n), & n > 1, \end{cases}$$

for some constants $c, a > 0, b > 1, d \geq 0$, and function $f(n)$. If $f(n)$ is in $O(n^d)$, then

$$T(n) \text{ is in } \begin{cases} O(n^d), & \text{if } d > \log_b(a), \\ O(n^d \log(n)), & \text{if } d = \log_b(a), \\ O(n^{\log_b(a)}), & \text{if } d < \log_b(a). \end{cases}$$

Muster Theorem (for subtract and conquer recurrences):

Let $T(n)$ be a function defined on positive n , and having the property

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases},$$

for some constants $c, a > 0, b > 0, d \geq 0$, and function $f(n)$. If $f(n)$ is in $O(n^d)$, then

$$T(n) \text{ is in } \begin{cases} O(n^d), & \text{if } a < 1, \\ O(n^{d+1}), & \text{if } a = 1, \\ O(n^d a^{n/b}), & \text{if } a > 1. \end{cases}$$