Announcements (Jan 25)

- Programming assignment due Feb 1
- Homework 2 handed out, due Feb 3
- Midterm postponed to Feb 8
- Today's plan
  - Expected case analysis of Quick Sort
  - Long multiplication
  - Matrix multiplication
  - Solving recurrences using Master method

Average case analysis

What happens if we get sort-of-balanced partitions, e.g., something like:

$$T(n) = T(9n/10) + T(n/10) + O(n) \ ?$$

Still get O(n log n)

Intuition: Can divide n by c > 1 only O(log n) times before getting 1.

$$\left.\begin{array}{c} n \\ \downarrow \\ n/c \\ \downarrow \\ n/c^2 \\ \downarrow \\ \vdots \\ \downarrow \\ 1 \end{array}\right\} \text{roughly, } \log_c n \text{ steps}$$

Intuition holds even if c is very close to 1, e.g., 100/99.

Analyzing Quicksort (expected case)

- Assume the pivot is chosen at random and that S is split into S1 and S2.
- The size of the S1 subproblem is i, $i = 0,1,\ldots,n-1$, with equal probability.
- Same for the size of S2.

Notes

Solving the recurrence

$$T(n) = cn + \frac{1}{n}\sum_{i=0}^{n-1} T(i) + T(n-i-1)$$

$$T(n) = cn + \frac{2}{n}\sum_{i=0}^{n-1} T(i)$$

$$\mathrm{n}T(n) = cn^2 + 2\sum_{i=0}^{n-1} T(i)$$

$$(n-1)T(n-1) = c(n-1)^2 + 2\sum_{i=0}^{n-2} T(i)$$

$$nT(n) - (n-1)T(n-1) = 2cn - c + 2T(n-1)$$

$$nT(n) = (n+1)T(n-1) + 2cn - c$$

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

Solving (continued)

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

Dividing by $n(n+1)$,

$$\frac{T(n)}{n+1} = \frac{2c}{n+1} + \frac{T(n-1)}{n}$$

$$= \frac{2c}{n+1} + \frac{2c}{n} + \frac{T(n-2)}{n-1}$$

$$= \cdots$$

$$= 2c\sum_{k=3}^{n+1}\frac{1}{k} + \frac{T(1)}{2}$$

$$= 2c\left(H_{n+1} - \frac{1}{1} - \frac{1}{2}\right) + \frac{T(1)}{2}$$

Since, H_n is $O(\log n)$,

$$T(n) \text{ is } O(n\log n)$$

Notes

Long multiplication

- Grade school method
- $\Theta(n^2)$ operations
- Is it optimal?
- Why even bother?

Why bother?

- Why not rely on hardware?
- True for numbers that fit in one computer word.
- But what if numbers are very large.
- Cryptography (encryption, digital signatures) uses big number "keys." Typically 256 to 1024 bits long!
- $\Theta(n^2)$ multiplication too slow for such large numbers.
- Karatsuba's (1962) divide-and-conquer scheme
  - Multiplies two n bit numbers in $\Theta(n^{1.59})$ steps.

Divide and conquer

To multiply two n-bit integers a and b:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$
\begin{aligned}
a &= 2^{n/2} \cdot a_1 + a_0 \\
b &= 2^{n/2} \cdot b_1 + b_0 \\
ab &= \left(2^{n/2} \cdot a_1 + a_0\right)\left(2^{n/2} \cdot b_1 + b_0\right) = 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0
\end{aligned}
$$

Ex. $\boldsymbol{a} = \underbrace{\texttt{1000}}_{a_1}\underbrace{\texttt{1101}}_{a_0}$ $\boldsymbol{b} = \underbrace{\texttt{1110}}_{b_1}\underbrace{\texttt{0001}}_{b_0}$

What is the time complexity?

Karatsuba's method

To multiply two n-bit integers a and b:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$
\begin{aligned}
a &= 2^{n/2} \cdot a_1 + a_0 \\
b &= 2^{n/2} \cdot b_1 + b_0 \\
ab &= 2^n \cdot a_1 b_1 + 2^{n/2} \cdot \left(a_1 b_0 + a_0 b_1\right) + a_0 b_0 \\
&= 2^n \cdot \underbrace{a_1 b_1}_{1} + 2^{n/2} \cdot \left(\underbrace{(a_1 + a_0)(b_1 + b_0)}_{2} - \underbrace{a_1 b_1}_{1} - \underbrace{a_0 b_0}_{3} \right) + \underbrace{a_0 b_0}_{3}
\end{aligned}
$$

$$T(n) = 3T(n/2) + O(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

Matrix multiplication

- Multiply two n x n matrices: $C = A \times B$.
- Standard method: $C_{ij} = \sum_1^n A_{ik} B_{kj}$.
- This takes O(n) time per element of C, for the total cost of $O(n^3)$ to compute C.
- A surprising discovery by Strassen (1969) broke the n^3 asymptotic barrier.
- Method is divide and conquer, with a clever choice of submatrices to multiply.

Multiplying blocks

C_{11} A_{11} A_{12} B_{11}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

B_{11}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Multiplying blocks (recursive)

- To multiply two n-by-n matrices A and B:
  - Divide: partition A and B into $\frac{n}{2}$ *by* $\frac{n}{2}$ blocks.
  - Conquer: multiply 8 pairs of $\frac{n}{2}$ *by* $\frac{n}{2}$ matrices, recursively.
  - Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

How?

Fast matrix multiplication: key idea

- Multiply 2-by-2 blocks with only 7 multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

- 7 multiplications and 18 (= 8+10) additions

Fast matrix multiplication: algorithm

- Strassen 1969
- To multiply two n-by-n matrices A and B:
  - Divide: partition A and B into $\frac{n}{2}$ by $\frac{n}{2}$ blocks.
  - Conquer: multiply 7 pairs of $\frac{n}{2}$ by $\frac{n}{2}$ matrices, **recursively**.
  - Combine: 7 products into 4 terms using 18 matrix additions.
- $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$
- $T(n) = \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.81})$

Fast matrix multiplication in practice

- Sparsity
- Caching effects
- Numerical stability
- Odd matrix dimensions
- Crossover to classical algorithm when n < 100 or so depending on machine specifics
- War of decimals
  - $O(n^{2.7801})$
  - $O(n^{2.7799})$
  - $O\left(n^{2.5218}\right)$
  - $O\left(n^{2.376}\right)$

Conjecture: $O(n^{2+\epsilon})$

Solving recurrences

- Studied as early as 1202 (Fibonacci)!
- Substitution method
  - Guess the form of the solution
  - Use mathematical induction to prove the solution
  - Can be used to prove both lower & upper bounds
- Recursion tree method
- Master method
- Generating functions

Usually ceiling, floor do not affect the growth rate

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

$$T(n)$$

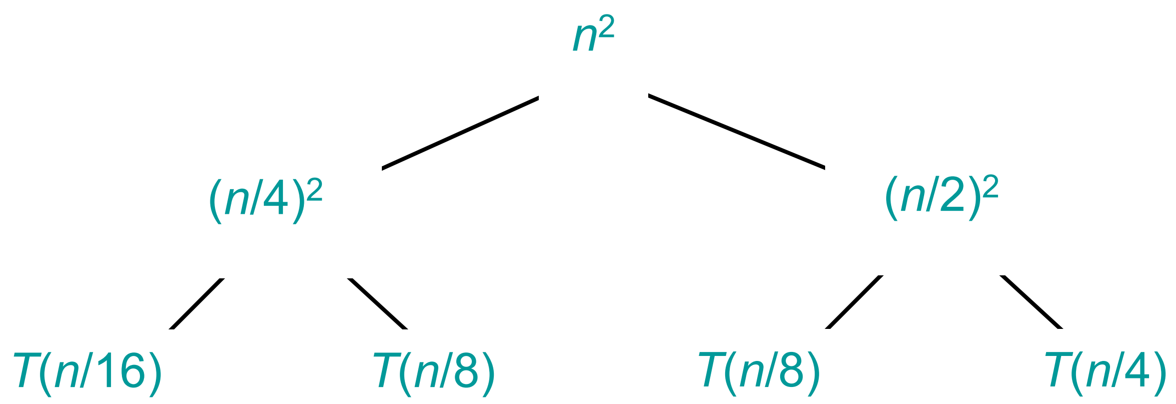
Example of recursion tree

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

$$
\begin{array}{ccc}
 & n^2 & \\
\swarrow & & \searrow \\
T(n/4) & & T(n/2)
\end{array}
$$

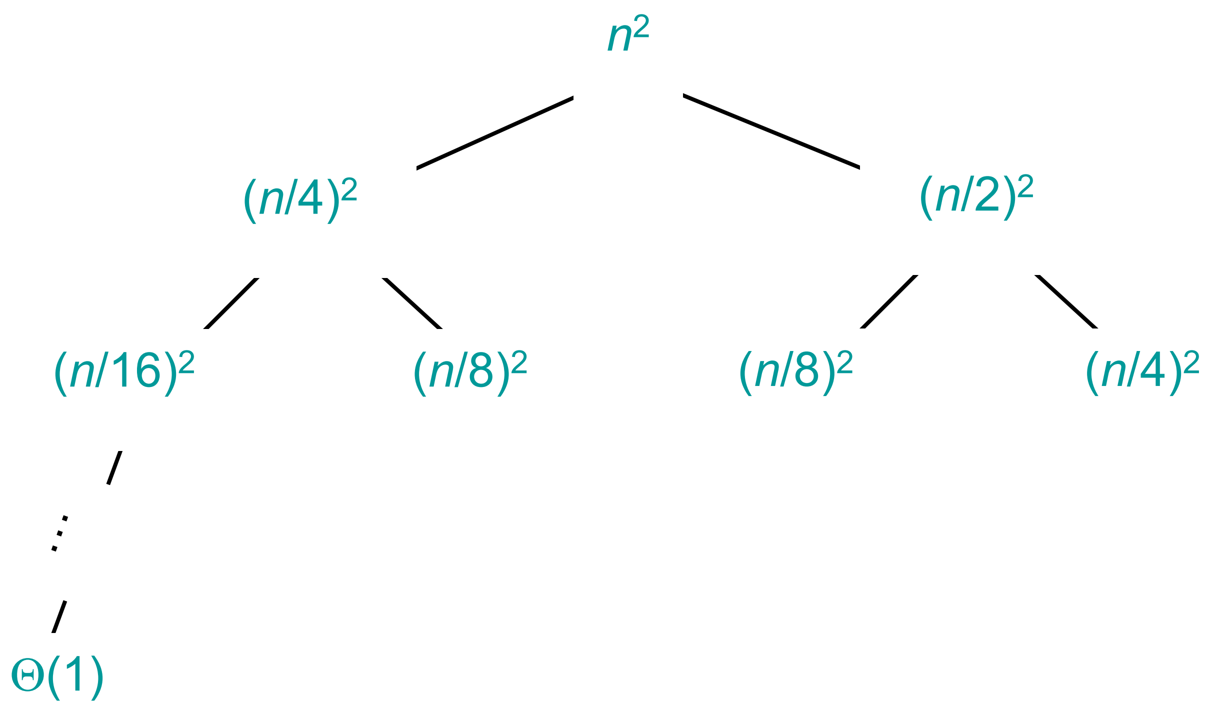
Example of recursion tree

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



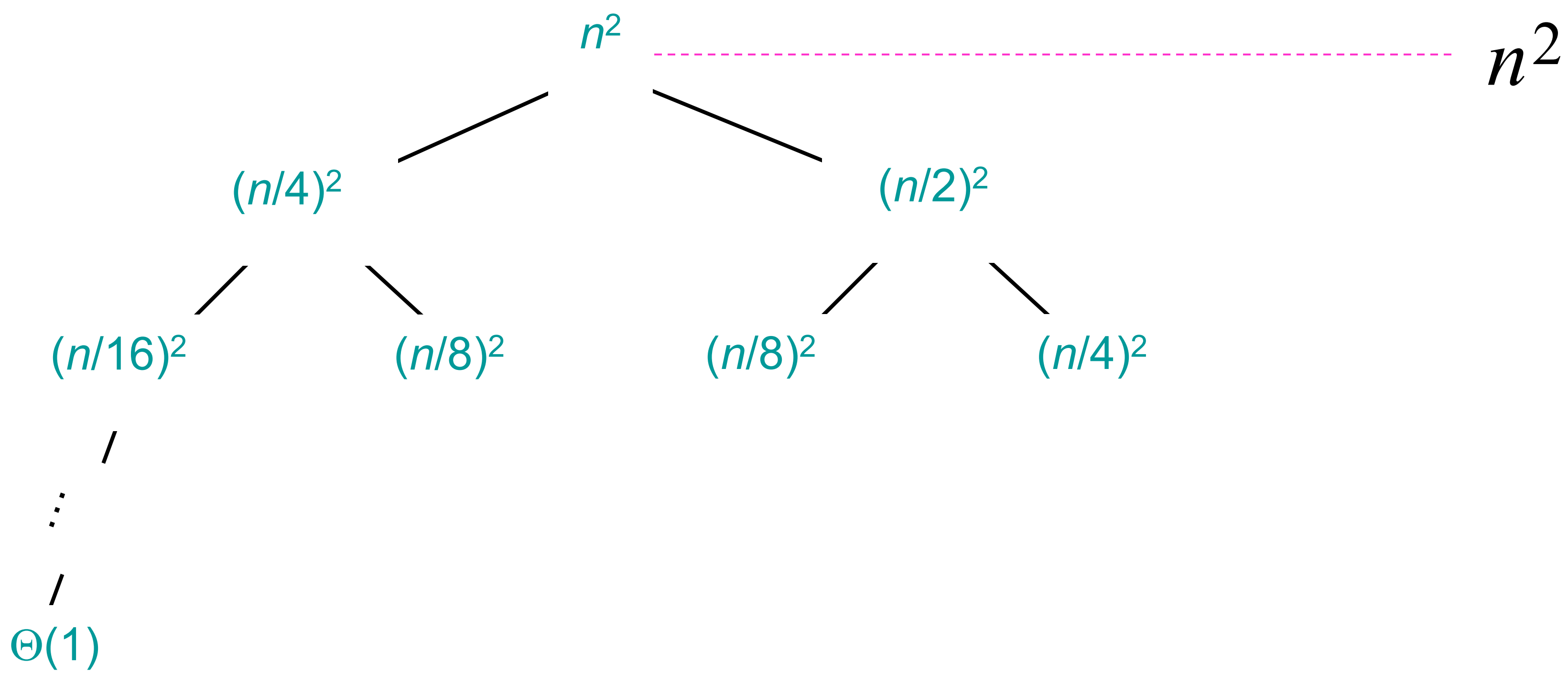
Example of recursion tree

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



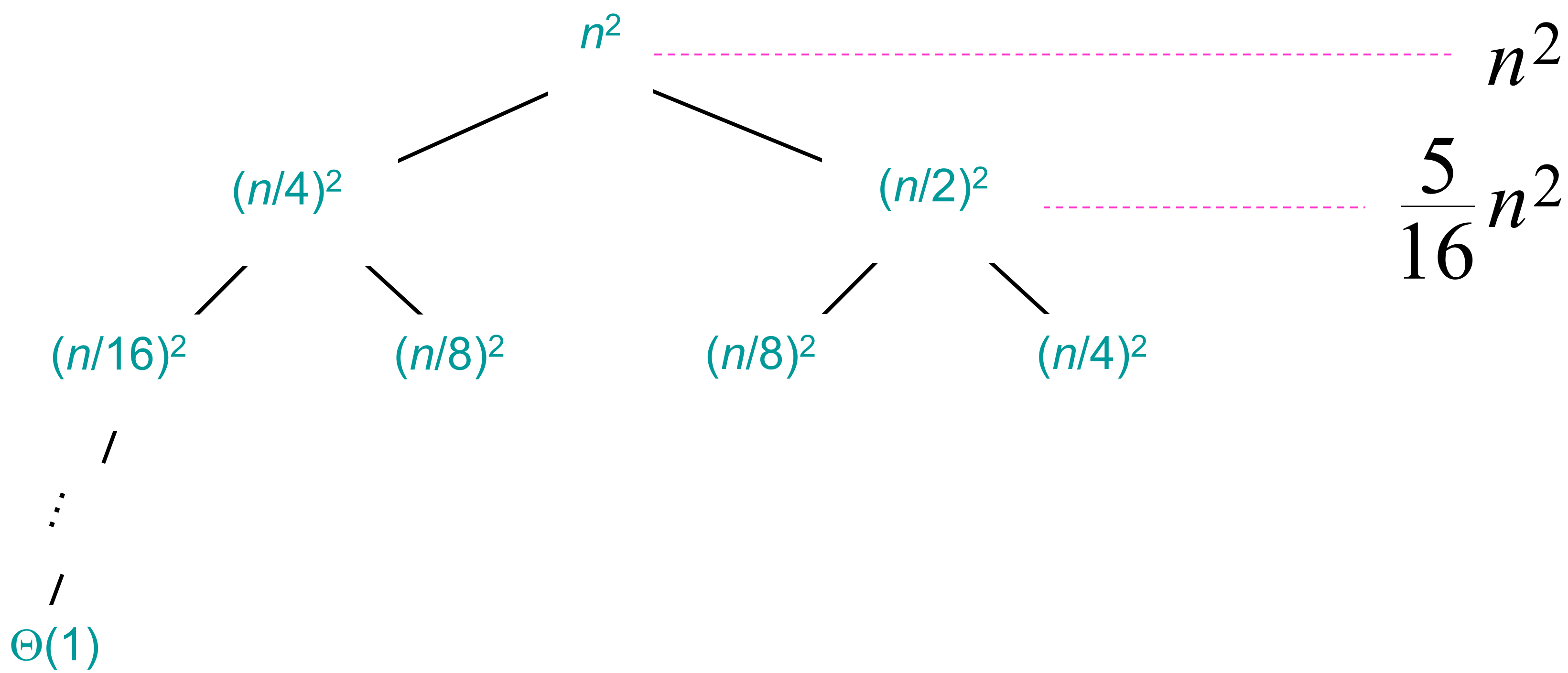
Example of recursion tree

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



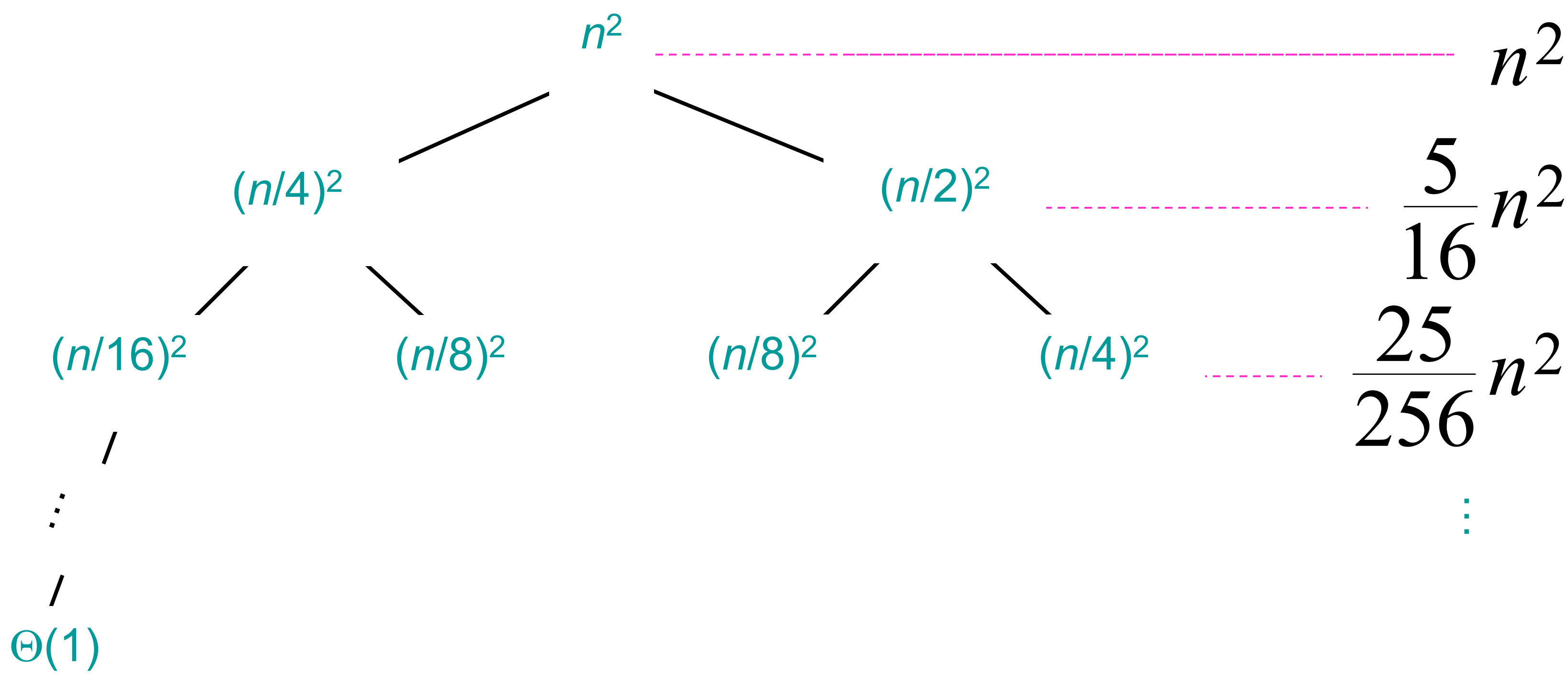
Example of recursion tree

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



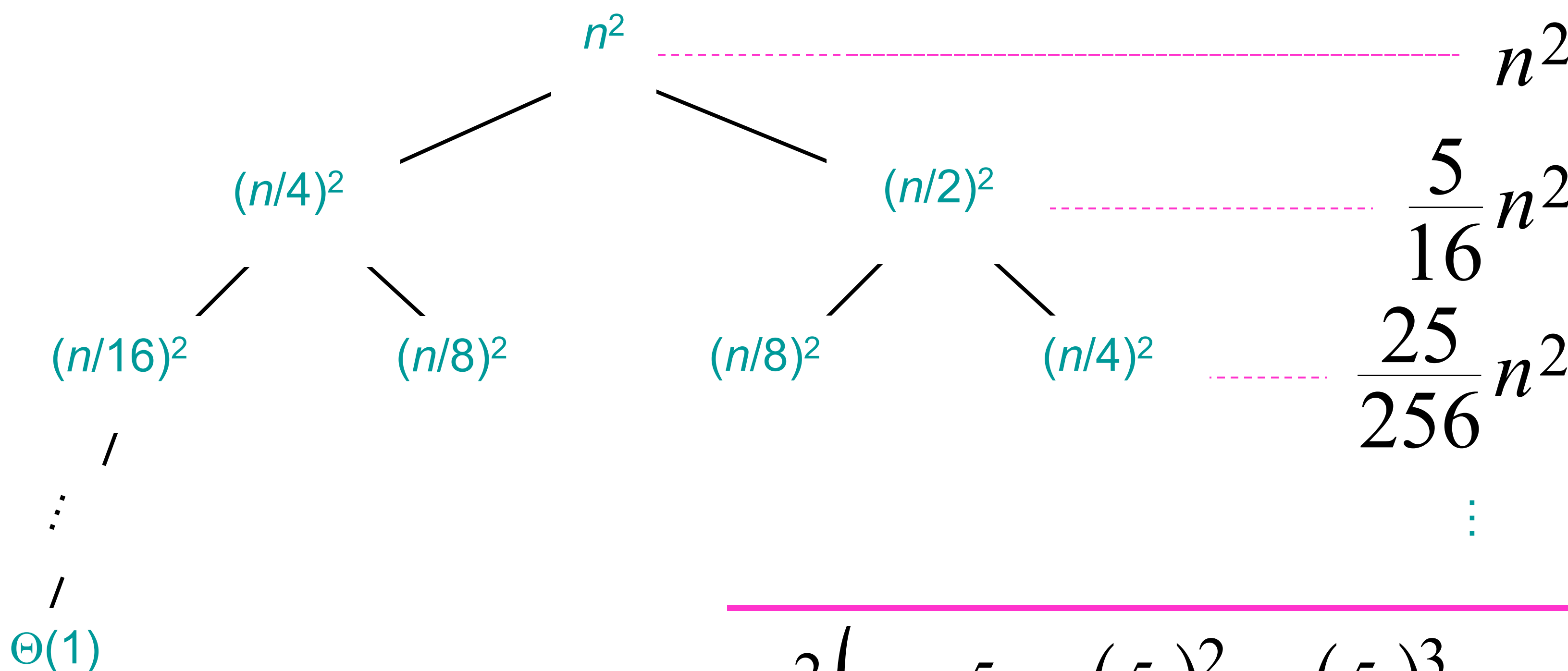
Example of recursion tree

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



Example of recursion tree

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



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$

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

The Master method

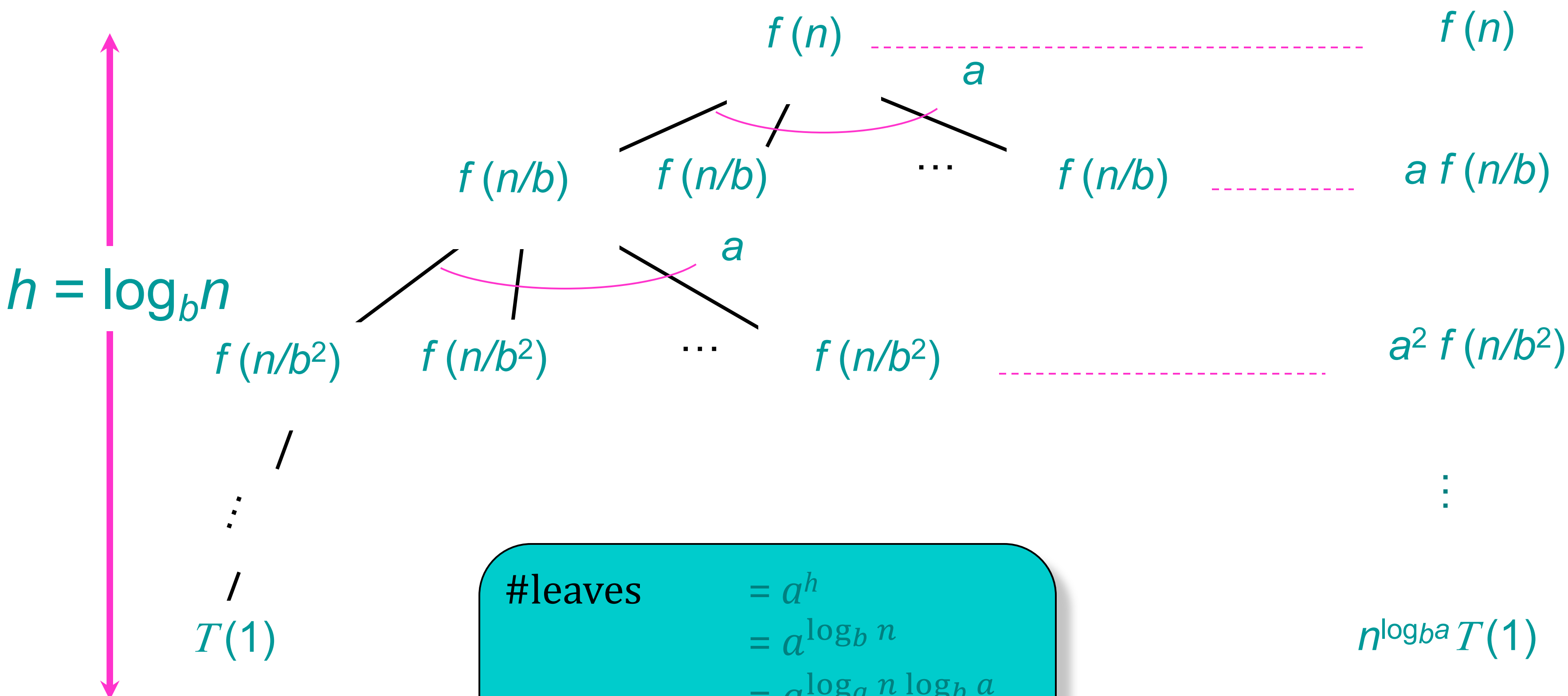
The master method applies to recurrences of the form

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

where $a \geq 1$, $b > 1$.

Idea of Master theorem

Recursion tree:



#leaves

$$
\begin{aligned}
&= a^h \\
&= a^{\log_b n} \\
&= a^{\log_a n \log_b a} \\
&= \left(a^{\log_a n}\right)^{\log_b a} \\
&= n^{\log_b a}
\end{aligned}
$$

Case 1

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

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

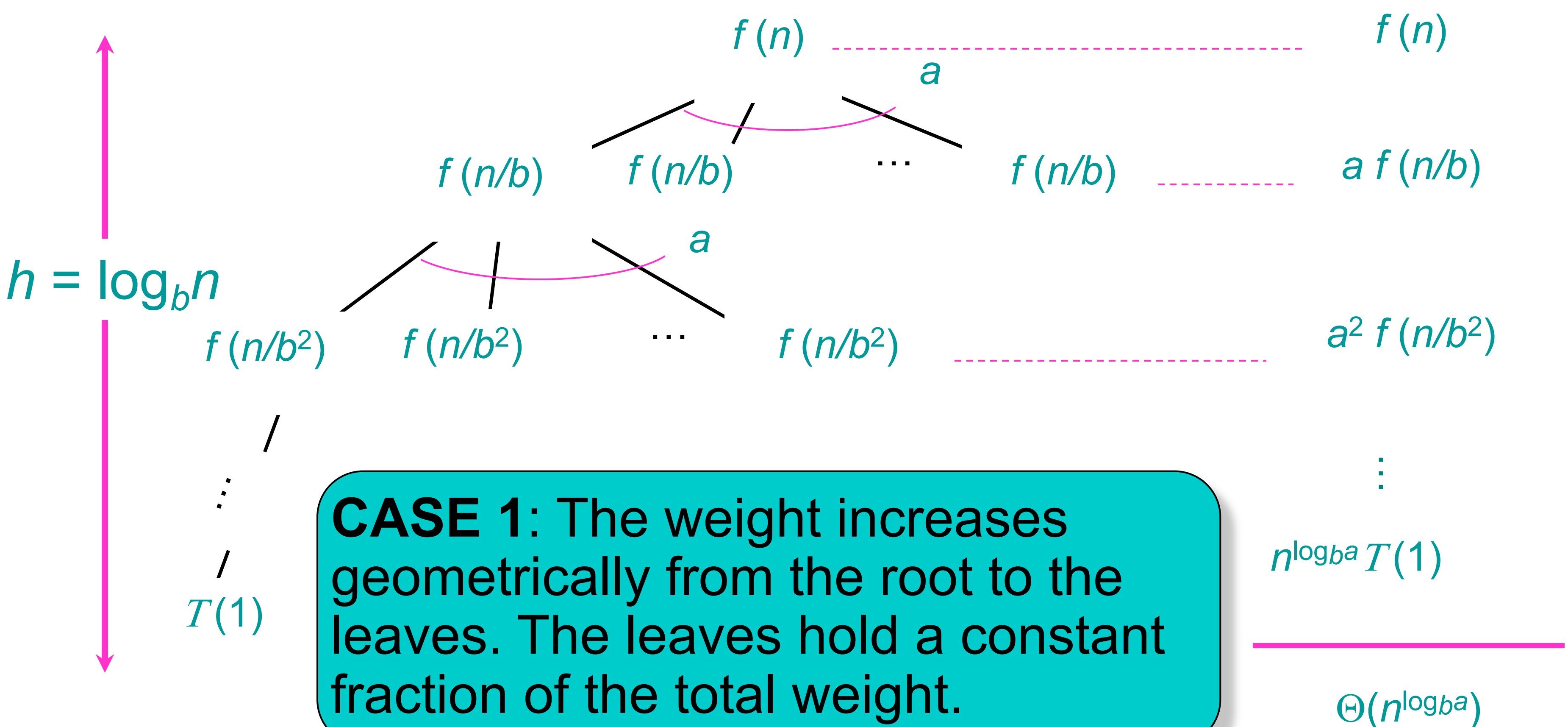
   $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

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

#leaves decides the function asymptotics.

Case 1

Recursion tree:



CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Example

Case 2

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

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

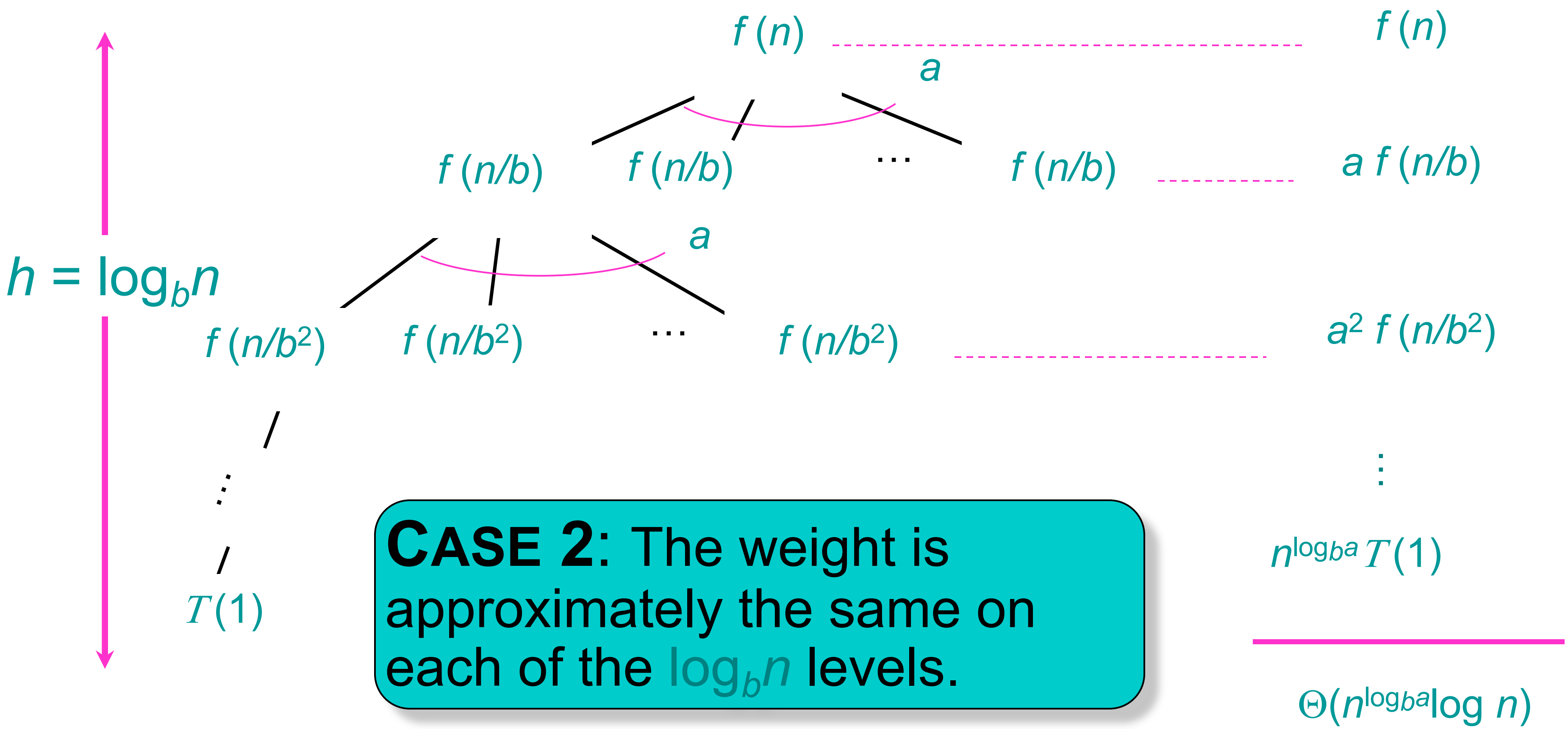
   $f(n)$ and $n^{\log_b a}$ grow at similar rates.

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

Entire tree decides the function asymptotics.

Case 2

Recursion tree:



CASE 2: The weight is approximately the same on each of the $\log_b n$ levels.

Example

Case 3

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

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),

   and $f(n)$ satisfies the ***regularity condition*** that

   $a f(n/b) \le c f(n)$ for some constant $c < 1$ and large n.

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

Root decides the function asymptotics.

Most of the polynomial functions we work with
satisfy the regularity condition.

Case 3

Recursion tree:

$f(n)$ .... $f(n)$

a

$f(n/b)$ $f(n/b)$ $\cdots$ $f(n/b)$ .... $a\,f(n/b)$

a

$h = \log_b n$

$f(n/b^2)$ $f(n/b^2)$ $\cdots$ $f(n/b^2)$ .... $a^2\,f(n/b^2)$

$\vdots$

$T(1)$ $\quad$ $n^{\log_b a}\,T(1)$

CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

$\Theta(f(n))$

Example

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: $f(n) = \Theta(n^2)$.

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

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 $\varepsilon = 1$; $4(n/2)^3 \le (1/2)\, n^3$.

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

Notes

Breakout on applying Master method

Proof of Master theorem: Case 1

Assume n is a power of b, results generalize to floors and ceilings

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

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

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

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

$$\mathrm{So,\ T(n)} = \Theta\left(\mathrm{n}^{\log_\mathrm{b} \mathrm{a}}\right)$$

Proof of Master theorem: Case 2

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

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

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

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

$$\text{So, } \mathrm{T(n)} = \Theta\left(\mathrm{n}^{\log_{\mathrm{b}} \mathrm{a}} \log_{\mathrm{b}} \mathrm{n}\right)$$

Proof of Master theorem: Case 3

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

$$af\left(\frac{n}{b}\right) \le cf(n),\ f\left(\frac{n}{b}\right) \le \left(\frac{c}{a}\right) f(n),\ f\left(\frac{n}{b^j}\right) \le \left(\frac{c}{a}\right)^j f(n),\ a^j f\left(\frac{n}{b^j}\right) \le c^j f(n)$$

$$\sum_{j=0}^{\log_b n - 1} a^j \, f\left(\frac{n}{b^j}\right) \le \sum_{j=0}^{\log_b n - 1} c^j f(n) \le f(n) \sum_{j=0}^{\infty} c^j = O\left(f(n)\right)$$

Note c < 1

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

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

$$\text{So, } \mathrm{T(n)} = \Theta\left(n^{\log_b a}\right) + \Theta(\mathrm{f(n)})$$

$$\text{Since } \mathrm{f(n)} = \Omega\left(n^{\log_b a + \epsilon}\right), \mathrm{T(n)} = \Theta(\mathrm{f(n)})$$

Notes