Master method

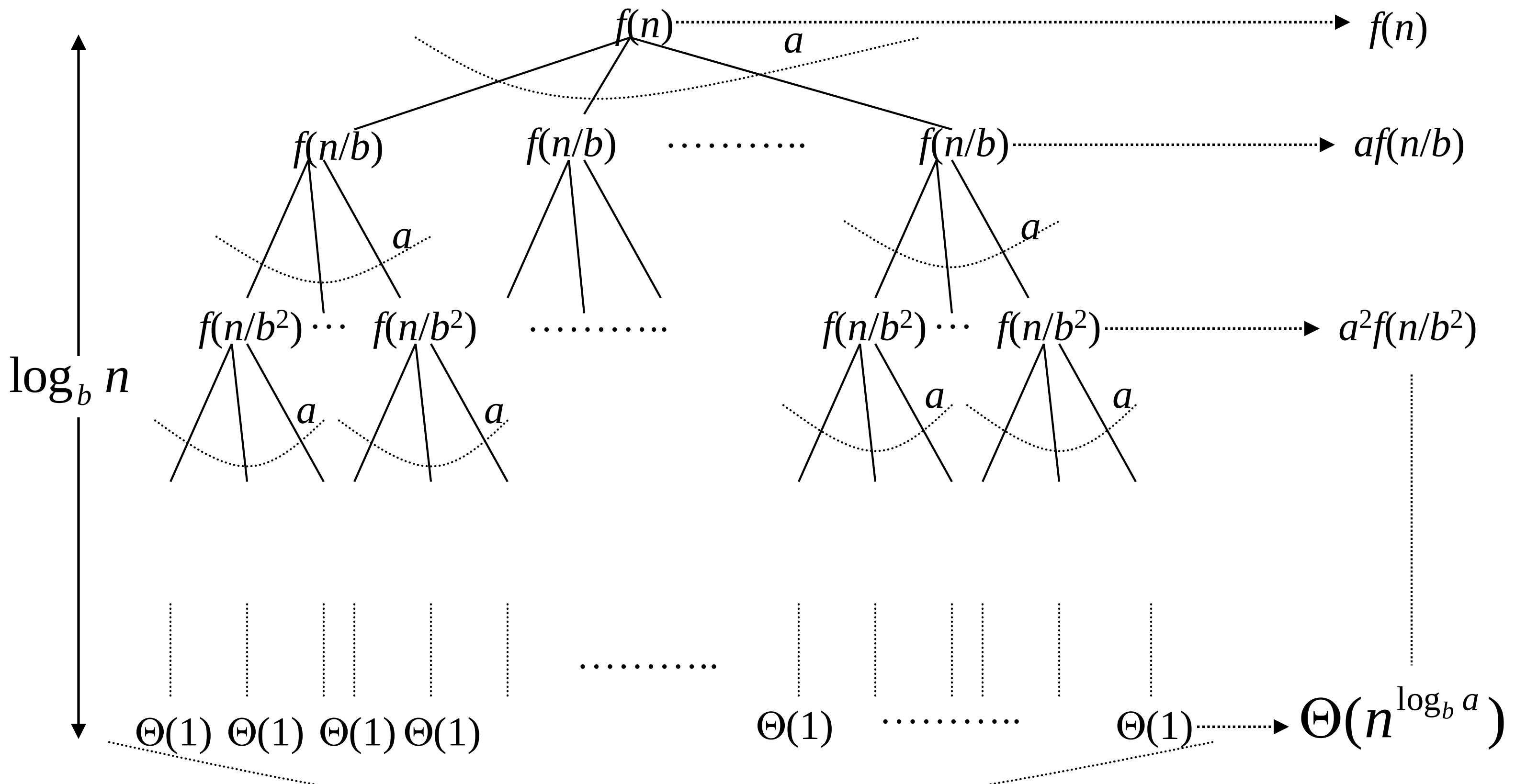
- Idea: solve *class* of recurrences of form

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

  $a \geq 0$ and $b > 0$, and f asymptotically positive.

- Abstractly: $T(n)$ is runtime for an algorithm; it's unknown, but we do know that:
  - a subproblems of size n/b are solved recursively, each in time $T(n/b)$.
  - $f(n)$ is the cost of dividing problem (beforehand) and combining the results (afterward).

Recursion tree



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

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

Analysis

- #of leaves = $n^{\log_b a} = a^{\log_b n}$
- Iterating the recurrence (expanding the tree) yields

$$
\begin{aligned}
T(n) &= f(n) + aT(n/b) \\
&= f(n) + af(n/b) + a^2T(n/b^2) \\
&= f(n) + af(n/b) + a^2f(n/b^2) + \ldots \\
&\quad + a^{\log_b n-1} f(n/b^{\log_b n-1}) + a^{\log_b n} T(1)
\end{aligned}
$$

Intuition

- Three common cases:
  - Running time dominated by cost at leaves.
  - Running time evenly distributed throughout tree
  - Running time dominated by cost at root
- Thus, to solve recurrence we need only characterize the dominant term in each case!

In each case, compare $f(n)$ and $n^{\log_b a}$

- Case 1:

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

  - $f(n)$ grows polynomially slower than $n^{\log_b a}$
  - Weight of each level increases geometrically from root to leaves $(1, a^1, a^2, \ldots)$
  - Leaves contain constant fraction of total weight; i.e., total is constant $\times$ #of leaves.
- the work at leaves dominates.

Analysis of Case 1

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

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

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

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

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

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

Case 2

- $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant k≥0
- $f(n)$ and $n^{\log_b a}$ same to within a polylogarithmic factor (log to a power)
- Weight decreases from root to leaves

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

  Work is distributed evenly throughout tree

Case 3

- $f(n) = \Omega(n^{\log_b a - \varepsilon})$ for some constant ε>0
- Also, need "regularity condition":

  $\exists c<1$ and $n_0>0$ such that

  $$af(n/b) \le cf(n) \ \forall n > n_0$$
- $f(n)$ grows polynomially faster than $n^{\log_b a}$
- Weight is geometrically decreasing from root to leaves
- Root contains constant fraction of total weight
- Thus, $\boxed{T(n) = \Theta(f(n))}$ Cost at root dominates.

Master Theorem, Summarized

Master Theorem: $T(n) = aT(n/b) + f(n)$

1. $f(n) = O(n^{\log_b a - \varepsilon}) \Rightarrow T(n) = O(n^{\log_b a})$

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

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and $af(n/b) \le cf(n)$
$\Rightarrow T(n) = \Theta(f(n))$

MT Strategy

Basic strategy:

1. Extract a, b, and $f(n)$ from given recurrence
2. Determine $n^{\log_b a}$
3. Compare $f(n)$ and $n^{\log_b a}$ asymptotically
4. Determine appropriate MT case (if any), and apply it.

MT examples (cont.)

$T(n) = 7T(n/2) + \Theta(n^2)$

$$n^{\log_b a} = n^{\log_2 7} = n^{\lg 7}$$

$$f(n)/n^{\log_b a} = \Theta(n^2)/n^{\lg 7} = \Theta(n^{2-\lg 7}) = O(n^{-0.8})$$

$\Rightarrow$ Case 1, applies

$$T(n) = \Theta(n^{\lg 7})$$

MT examples (cont.)

- Binary search (not sort!):

$$T(n) = T(n/2) + \Theta(1); \ (a=1, b=2)$$

$$n^{\lg 1} = n^0 = 1$$

$$\Theta(1)/1 = \Theta(\lg^0 n) = 1$$

$\Rightarrow$ Case 2,

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

MT examples (cont.)

$T(n) = 4T(n/2) + n^3$; (a=4, b=2)

$$n^3 / n^{\lg 4} = n \qquad \Rightarrow \text{Case 3}$$

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

Are we done?

No… Need to check regularity condition:

$$4f(n/2) \le cf(n)$$

$$4n^3/8 \le cn^3$$

$$n^3/2 \le cn^3$$

$$c=3/4<1$$

MT examples (cont.)

One more recurrence:

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

$$\frac{n^2/\lg n}{n^2} = 1/\lg n = \begin{cases} O(n^{-\varepsilon}) & \varepsilon > 0 ? \lg n = \Omega(n^{\varepsilon}) ? \\ \Theta(\lg^k n) & k \geq 0 ? No \\ \Omega(n^{\varepsilon}) & \varepsilon > 0 ? No \end{cases}$$

No MT case applies!

Solution: $T(n) = \Theta(n^2 \lg\lg n)$ – substitution.

Merge sort analysis

For merge sort:

$T(n) = 2T(n/2) + \Theta(n)$; a=2, b=2

$$n^{\log_b a} = n^{\log_2 2} = n$$

$$f(n)/n^{\log_b a} = \Theta(n)/n = \Theta(1) = \Theta(\lg^0 n)$$

$\Rightarrow$Case 2, with k=0

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

Binary search

Search for a key in a *sorted* array

Example: Given array:

3 5 7 8 9 12 15

Find key 9

Binary search

Binary search as Divide-and-conquer algorithm:

1. *Divide*: Check middle element.
2. *Conquer*: Search one subarray.
3. *Combine*: Trivial.

Recurrence for binary search

$$T(n) = \underbrace{1}_{\#\text{subproblems}} \; T(\overbrace{n/2}^{\text{subproblem size}}) + \underbrace{\Theta(1)}_{\text{work dividing \& combining}}$$

$$T(n) = T(n/2) + \Theta(1)$$

Binary search analysis

solve using MT:

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

a=1 and b=2 $\Rightarrow O(n^{\log_b a}) = n^{\log_2 1} = 1$

$\Rightarrow f(n) / O(n^{\log_b a}) = \Theta(1)$

$\Rightarrow$ Case 2 of MT $\Rightarrow T(n) = \Theta(\lg n)$