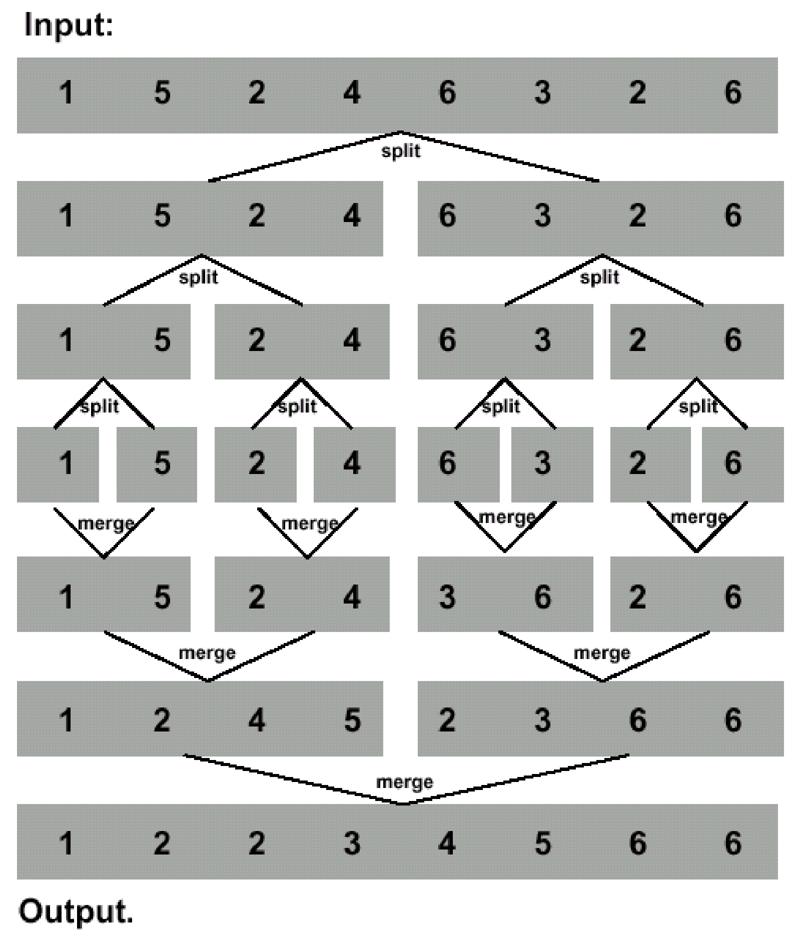
Algorithms and Data Structures
Lecture III

Merge Sort Revisited

- To sort n numbers
  - if n=1 done!
  - recursively sort 2 lists of numbers $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ elements
  - merge 2 sorted lists in $\Theta(n)$ time
- Strategy
  - break problem into similar (smaller) subproblems
  - recursively solve subproblems
  - combine solutions to answer



Merge Sort Revisited

```
Merge-Sort(A, p, r):
   if p < r then
      q←(p+r)/2
      Merge-Sort(A, p, q)
      Merge-Sort(A, q+1, r)
      Merge(A, p, q, r)
```

Merge(A, p, q, r)

Take the smallest of the two topmost elements of sequences A[p..q] and A[q+1..r] and put into the resulting sequence. Repeat this, until both sequences are empty. Copy the resulting sequence into A[p..r].

Recurrences

- Running times of algorithms with **Recursive calls** can be described using recurrences
- A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs
- Example: Merge Sort

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

Solving Recurrences

- Substitution method
  - guessing the solutions
  - verifying the solution by induction
- Iteration (recursion-tree) method
  - expansion of the recurrence
  - drawing of the recursion-tree
- Master method
  - templates for different classes of recurrences

Substitution method

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

1) Guess that $T(n) = O(n^3)$, i.e., that T of the form cn^3

2) Assume $T(k) \leq ck^3$ for $k \leq n/2$ and

3) Prove $T(n) \leq cn^3$ by induction

$$\begin{aligned}
T(n) &= 4T(n/2) + n \text{ (recurrence)} \\
&\leq 4\text{c(n/2)}^3 + n \text{ (ind. hypoth.)} \\
&= \frac{c}{2}n^3 + n \text{ (simplify)} \\
&= cn^3 - \left(\frac{c}{2}n^3 - n\right) \text{ (rearrange)} \\
&\leq cn^3 \text{ if } c \geq 2 \text{ and } n \geq 1 \text{ (satisfy)}
\end{aligned}$$

Thus $T(n) = O(n^3)$!

Subtlety: Must choose c big enough to handle

$T(n) = \Theta(1)$ for $n < n_0$ for some n_0

Substitution Method

- Achieving tighter bounds

Try to show $T(n) = O(n^2)$

Assume $T(k) \le ck^2$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4c(n/2)^2 + n \\ &= cn^2 + n \\ &\le cn^2 \text{ for no choice of } c > 0. \end{aligned}$$

Substitution Method (2)

- The problem? We could not rewrite the equality

$$T(n) = cn^2 + (\text{something positive})$$

- as:

$$T(n) \le cn^2$$

- in order to show the inequality we wanted
- Sometimes to prove inductive step, try to strengthen your hypothesis
  - T(n) $\le$ (answer you want) - (something > 0)

Substitution Method (3)

- Corrected proof: the idea is to strengthen the inductive hypothesis by subtracting lower-order terms!

$$\text{Assume } T(k) \le c_1 k^2 - c_2 k \text{ for } k < n$$

$$\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4(c_1(n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n \text{ if } c_2 \ge 1
\end{aligned}$$

Iteration Method

- The basic idea is to **expand** the recurrence and **convert to a summation**!

$$
\begin{aligned}
T(n) &= n + 3T(\lfloor n/4 \rfloor) \\
&= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) \\
&= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))) \\
&= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor) \\
T(n) &= n + 3n/4 + 9n/16 + 27n/64 + \ldots + 3^{\log_4 n} T(1) \\
&\le n \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{4}\right)^i + \Theta\left(n^{\log_4 3}\right) \\
&\le 4n + o(n) \\
&\le O(n)
\end{aligned}
$$

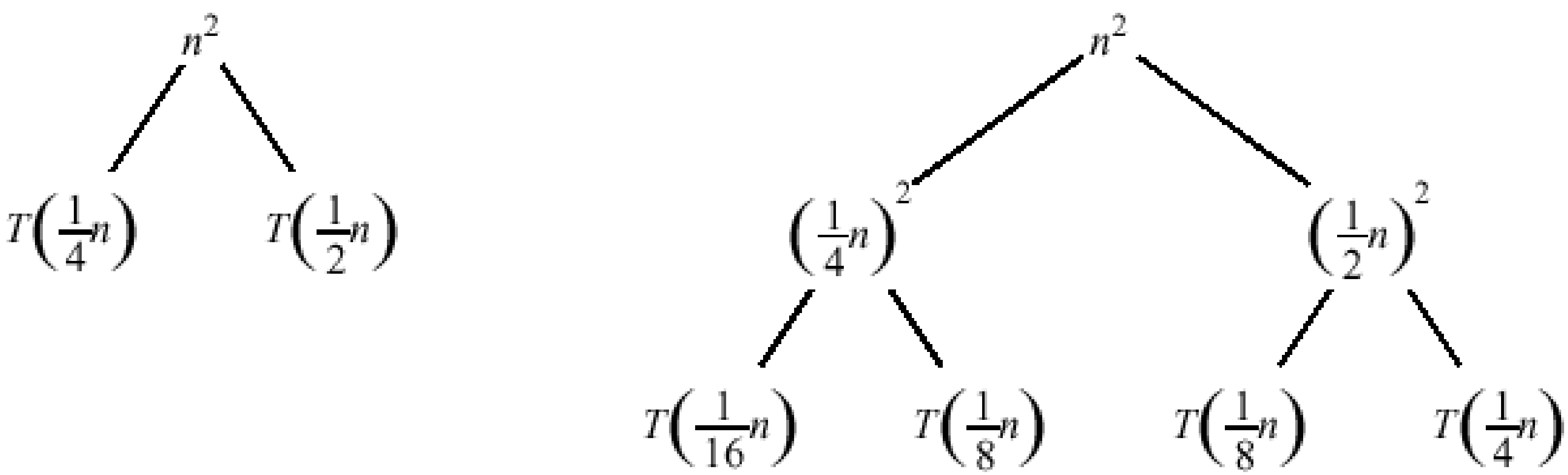
Iteration Method (2)

- The iteration method is often used to **generate guesses** for the substitution method
- Should know rules and have intuition for arithmetic and geometric series
- Math can be messy and hard
- Focus on two parameters
  - the number of times the recurrence needs to be iterated to reach the boundary condition
  - the sum of the terms arising from each level of the iteration process

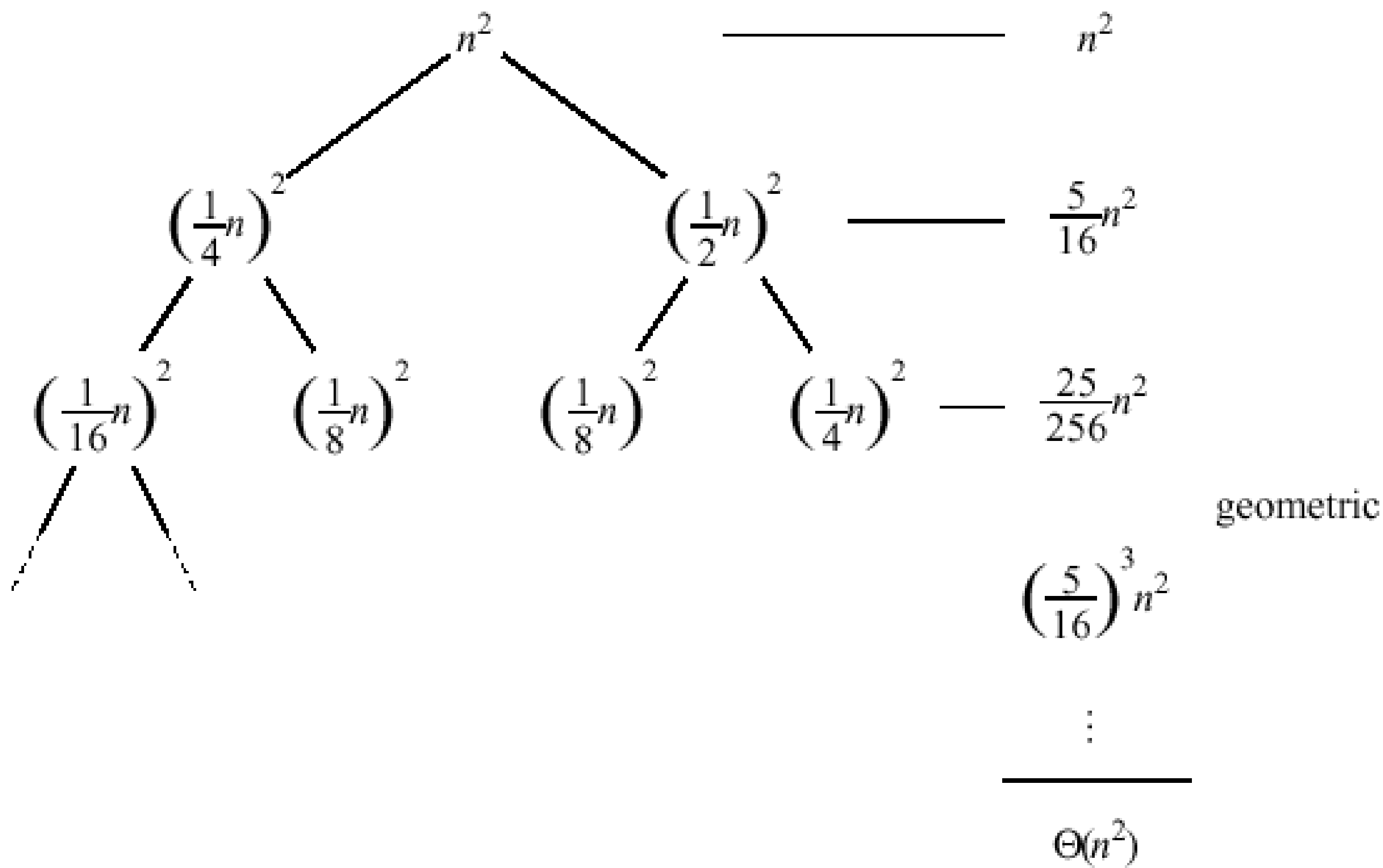
Recursion Tree

- A recursion tree is a convenient way to visualize what happens when a recurrence is iterated
- Construction of a recursion tree

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

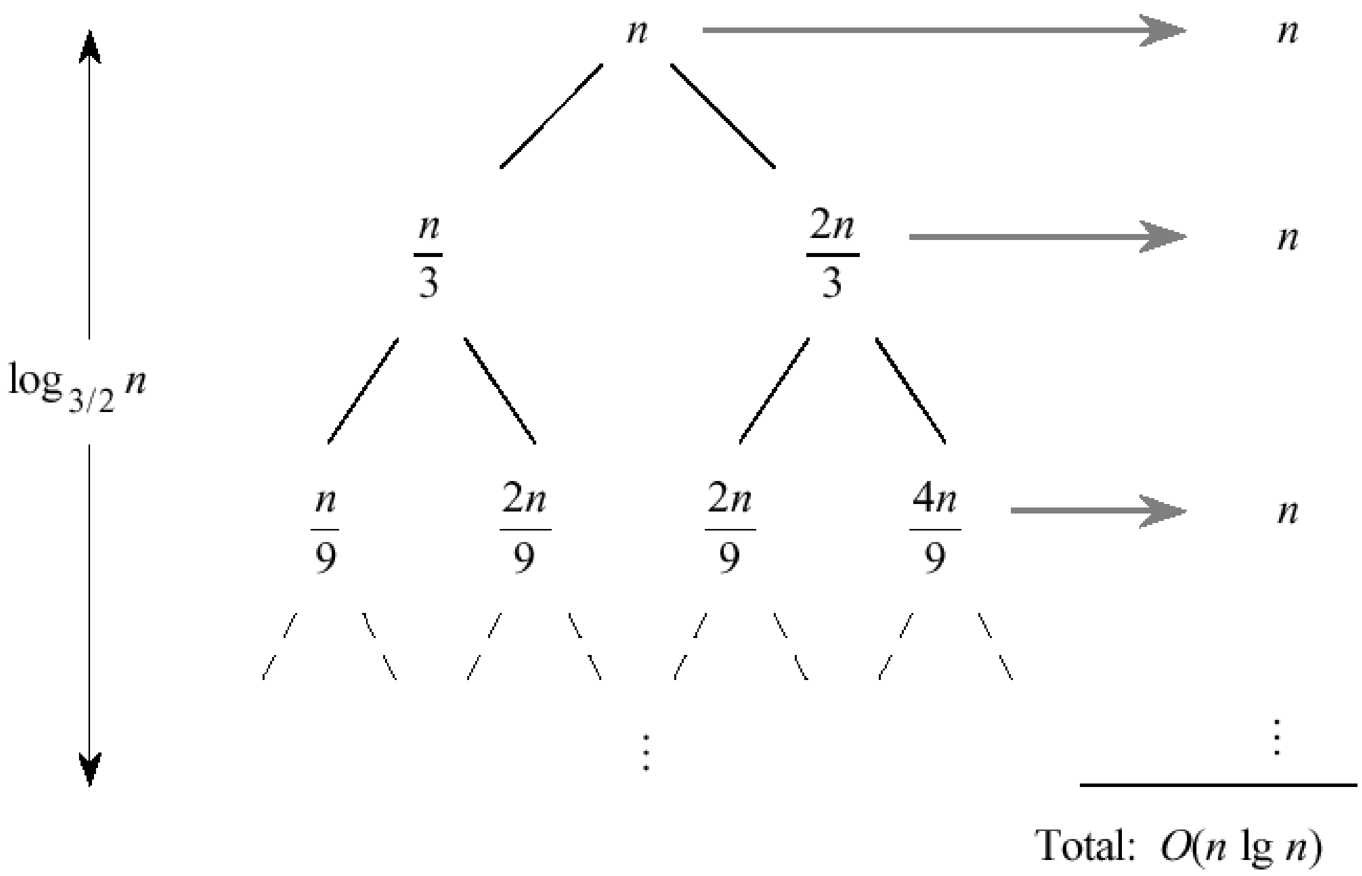


Recursion Tree (2)



Recursion Tree (3)

$$T(n) = T(n/3) + T(2n/3) + n$$



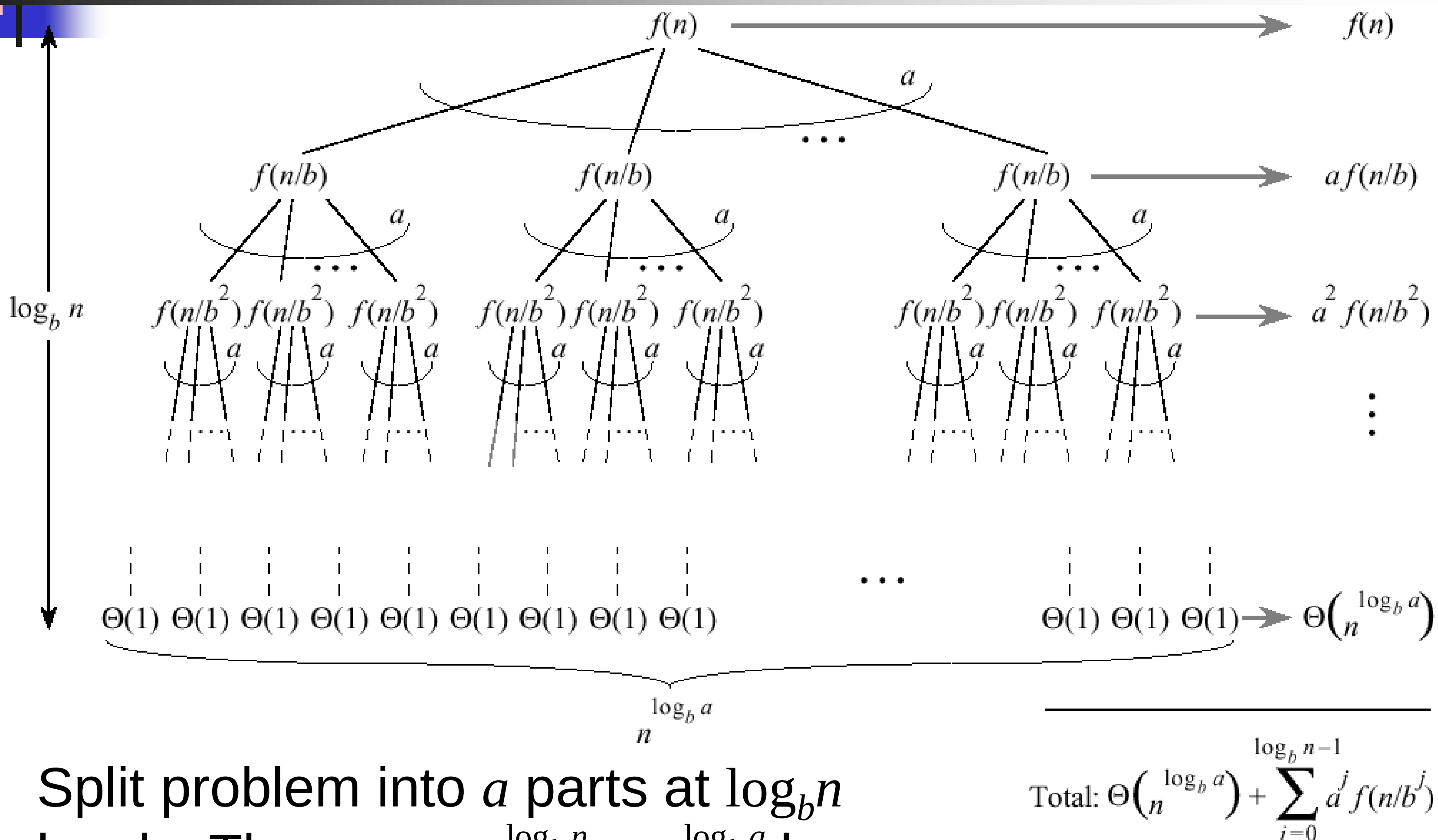
Master Method

- The idea is to solve a class of recurrences that have the form

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

- $a >= 1$ and $b > 1$, and f is asymptotically positive!
- Abstractly speaking, $T(n)$ is the runtime for an algorithm and we know that
  - a subproblems of size n/b are solved recursively, each in time $T(n/b)$
  - $f(n)$ is the cost of dividing the problem and combining the results. In merge-sort $T(n) = 2T(n/2) + \Theta(n)$

Master Method (2)



Split problem into a parts at $\log_b n$ levels. There are $a^{\log_b n} = n^{\log_b a}$ leaves

Master Method (3)

- Number of leaves: $a^{\log_b n} = n^{\log_b a}$
- Iterating the recurrence, expanding the tree yields

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

Thus,

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

  - The first term is a division/recombination cost (totaled across all levels of the tree)
  - The second term is the cost of doing all $n^{\log_b a}$ subproblems of size 1 (total of all work pushed to leaves)

MM Intuition

- Three common cases:
  - Running time dominated by cost at leaves
  - Running time evenly distributed throughout the tree
  - Running time dominated by cost at root
- Consequently, to solve the recurrence, we need only to characterize the dominant term
- In each case compare $f(n)$ with $O(n^{\log_b a})$

MM Case 1

- $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$
  - $f(n)$ grows polynomially (by factor n^{ε}) slower than $n^{\log_b a}$
- **The work at the leaf level dominates**
  - Summation of recursion-tree levels $O(n^{\log_b a})$
  - Cost of all the leaves $\Theta(n^{\log_b a})$
  - Thus, the overall cost $\Theta(n^{\log_b a})$

MM Case 2

- $f(n) = \Theta(n^{\log_b a} \lg n)$
  - $f(n)$ and $n^{\log_b a}$ are asymptotically the same
- **The work is distributed equally throughout the tree** $T(n) = \Theta(n^{\log_b a} \lg n)$
  - (level cost) $\times$ (number of levels)

MM Case 3

- $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$
  - Inverse of Case 1
  - $f(n)$ grows polynomially faster than $n^{\log_b a}$
  - Also need a regularity condition

    $\exists c < 1 \text{ and } n_0 > 0 \text{ such that } af(n/b) \le cf(n) \ \forall n > n_0$
- **The work at the root dominates**

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

Master Theorem Summarized

- Given a recurrence of the form $T(n) = aT(n/b) + f(n)$

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

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

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

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

3. $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ and $af(n/b) \leq cf(n)$, for some $c < 1, n > n_0$

   $\Rightarrow T(n) = \Theta\left(f(n)\right)$

- The master method cannot solve every recurrence of this form; there is a gap between cases 1 and 2, as well as cases 2 and 3

Strategy

- Extract *a*, *b*, and *f*(*n*) from a given recurrence
- Determine $n^{\log_b a}$
- Compare *f*(*n*) and $n^{\log_b a}$ asymptotically
- Determine appropriate MT case, and apply
- Example merge sort

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

$$a = 2,\ b = 2;\ n^{\log_b a} = n^{\log_2 2} = n = \Theta(n)$$

$$\text{Also } f(n) = \Theta(n)$$

$$\Rightarrow \text{Case 2: } T(n) = \Theta\left(n^{\log_b a} \lg n\right) = \Theta\left(n \lg n\right)$$

Examples

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

$a = 1, b = 2;\ n^{\log_2 1} = 1$

also $f(n) = 1,\ f(n) = \Theta(1)$

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

```
Binary-search(A, p, r, s):
    q←(p+r)/2
    if A[q]=s then return q
    else if A[q]>s then
          Binary-search(A, p, q-1, s)
    else Binary-search(A, q+1, r, s)
```

$T(n) = 9T(n/3) + n$

$a = 9, b = 3;$

$f(n) = n,\ f(n) = O(n^{\log_3 9 - \varepsilon})$ with $\varepsilon = 1$

$\Rightarrow$ Case 1: $T(n) = \Theta\left(n^2\right)$

Examples (2)

$T(n) = 3T(n/4) + n \lg n$

$a = 3, b = 4;\ n^{\log_4 3} = n^{0.793}$

$f(n) = n \lg n,\ f(n) = \Omega(n^{\log_4 3+\varepsilon})$ with $\varepsilon \approx 0.2$

$\Rightarrow$ Case 3:

Regularity condition

$af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$

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

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

$a = 2, b = 2;\ n^{\log_2 2} = n^1$

$f(n) = n \lg n,\ f(n) = \Omega(n^{1+\varepsilon})$ with ε?

also $n \lg n / n^1 = \lg n$

$\Rightarrow$ neither Case 3 nor Case 2!

Examples (3)

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

$$a = 4, b = 2;\ n^{\log_2 4} = n^2$$

$$f(n) = n^3;\ f(n) = \Omega(n^2)$$

$\Rightarrow$ Case 3: $T(n) = \Theta\left(n^3\right)$

Checking the regularity condition

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

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

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

$$c = 3/4 < 1$$

Next lecture

- Sorting
  - QuickSort
  - HeapSort