Design and Analysis of Algorithms I

Master Method

Proof (Part I)

The Master Method

- If $T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$

then

$$T(n) = \begin{cases} O\left(n^d \log n\right) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Preamble

Assume : recurrence is

pattern???

I. $T(1) \leq c$

II. $T(n) \leq aT(n/b) + cn^d$

(For some constant c)

And n is a power of b.

a^j subproblems b^j size

(general case is similar, but more tedious)

Idea : generalize MergeSort analysis.

(i.e., use a recursion tree)

Ter
ver

What is the pattern ? Fill in the blanks in the following statement: at each level $j = 0,1,2,\ldots,\log_b n$, there are <blank> subproblems, each of size <blank>

# of times you can divide n by b before reaching 1

- ○ a^j and n/a^j, respectively.
- ○ a^j and n/b^j, respectively.
- ○ b^j and n/a^j, respectively.
- ○ b^j and n/b^j, respectively.

The Recursion Tree

a braches

why not b branches???

Level 0

Level 1

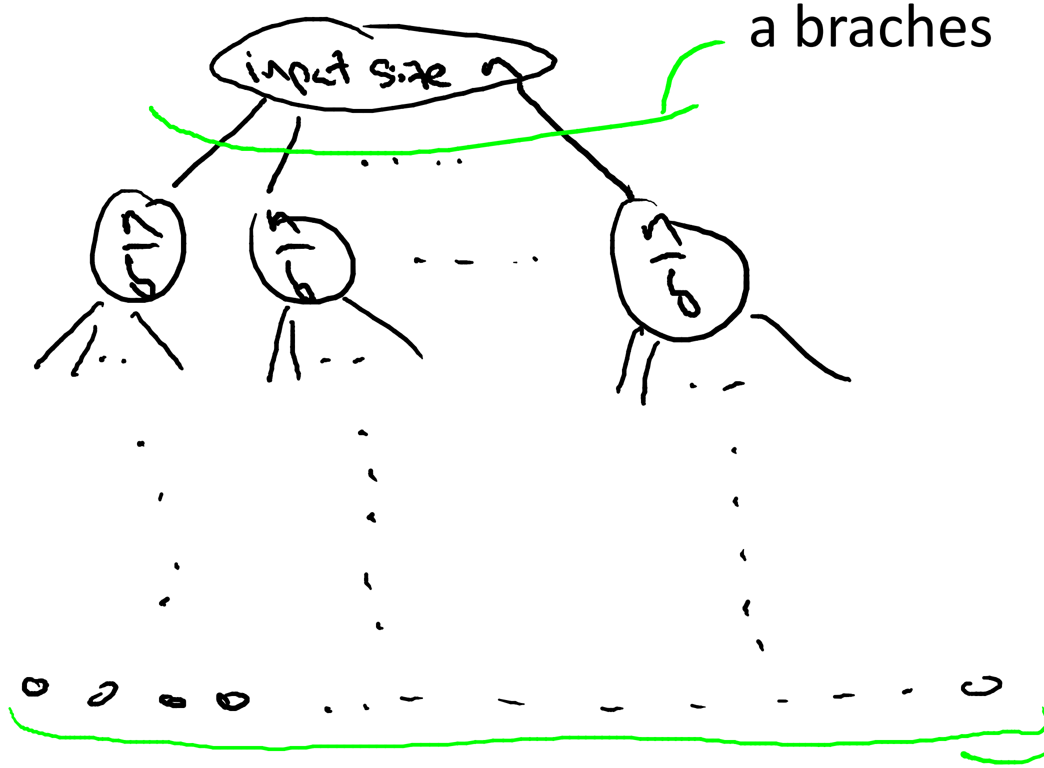
.

.

.

.

Level $\log_b n$



Base cases (size 1)

Work at a Single Level

Total work at level j [ignoring work in recursive calls]

Work per level-j subproblem

$$\le \underbrace{a^j}_{\text{\# of level-j subproblems}} \cdot c \cdot \left(\frac{n}{b^j}\right)^d = cn^d \cdot \left(\frac{a}{b^d}\right)^j$$

Size of each level-j subproblem

operations no more than constant c

Total Work

Summing over all levels j = 0,1,2,..., $\log_b n$:

Total work $\leq cn^d \cdot \sum_{j=0}^{\log_b n} (\frac{a}{b^d})^j \qquad (*)$

Tim Roughgarden