Master Method

Proof (Part 1)

Design and Analysis
of Algorithms 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

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.

(general case is similar, but more tedious)

Idea : generalize MergeSort analysis.

(i.e., use a recursion tree)

NEXTCORE AI

What is the pattern ? Fill in the blanks in the following statement: at each level j = 0,1,2,…,$\log_b n$, 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

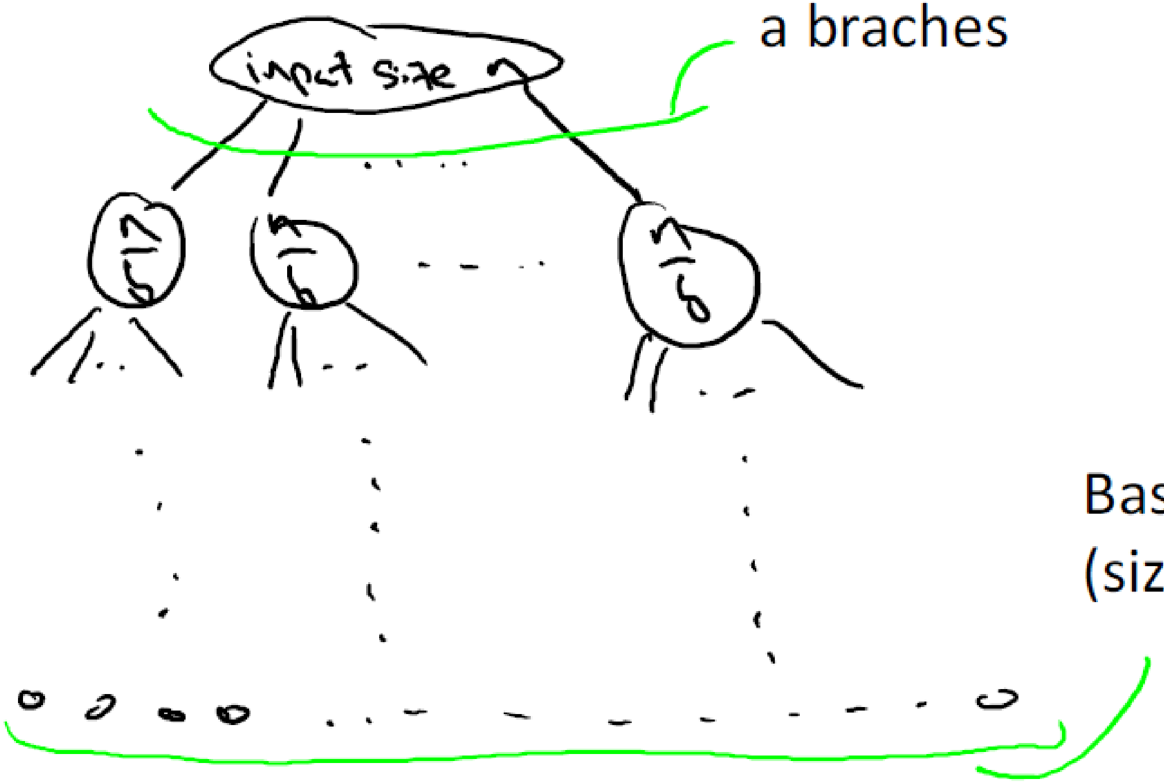
a braches

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]

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

Work per level-j subproblem: $c \cdot \left(\frac{n}{b^j}\right)^d$

\# of level-j subproblems: a^j

Size of each level-j subproblem: $\frac{n}{b^j}$

Total Work

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

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

Master Method

Intuition for the 3 Cases

Design and Analysis of Algorithms I

Nextcore AI - Gopal Shangari

HOW TO THINK ABOUT (*)

Our upper bound on the work at level j:

$$cn^d \times (\frac{a}{b^d})^j$$

Interpreta3on

a = rate of subproblem prolifera3on (RSP)

b^d = rate of work shrinkage (RWS)

(per subproblem)

Which of the following statements are true?
(Check all that apply.)

- ☐ If RSP < RWS, then the amount of work is decreasing with the recursion level j.
- ☐ If RSP > RWS, then the amount of work is increasing with the recursion level j.
- ☐ No conclusions can be drawn about how the amount of work varies with the recursion level j unless RSP and RWS are equal.
- ☐ If RSP and RWS are equal, then the amount of work is the same at every recursion level j.

INTUITION FOR THE 3 CASES

Upper bound for level j: $cn^d \times (\frac{a}{b^d})^j$

1. RSP = RWS => Same amount of work each level (like Merge Sort) [expect $O(n^d \log(n))$]
2. RSP < RWS => less work each level => most work at the root [might expect $O(n^d)$]
3. RSP > RWS => more work each level => most work at the leaves [might expect O(# leaves)]

Master Method

Proof (Part II)

Design and Analysis of Algorithms I

THE STORY SO FAR/CASE 1

Total work: $\leq cn^d \times \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$ (*)

= 1 for all j

=1

= $(\log_b n + 1)$

$If \quad a = b^d, \; then$

$$(*) = cn^d(\log_b n + 1)$$

$$= O(n^d \log n)$$

[end Case 1]

Basic Sums Fact

For $r \neq 1$, we have

$$1 + r + r^2 + r^3 + \ldots + r^k = \frac{r^{k+1} - 1}{r - 1}$$

Proof : by induction (you check)

Upshot:

Independent of k

1. If r<1 is constant, RHS is <= $\frac{1}{1-r}$ = a constant
   I.e., 1st term of sum dominates
2. If r>1 is constant, RHS is <= $r^k \cdot (1 + \frac{1}{r-1})$
   I.e., last term of sum dominates

Case 2

Total work: $\leq c n^d \times \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$ (*)

:= r

<= a constant

(independent of n)

[by basic sums fact]

$If \quad a < b^d \quad [RSP < RWS]$

$$= O(n^d)$$

[total work dominated by top level]

Case 3

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

:= r > 1

<= constant * largest term

$If \quad a > b^d \quad [RSP > RWS]$

$(*) = O(n^d \cdot (\frac{a}{b^d})^{\log_b n})$

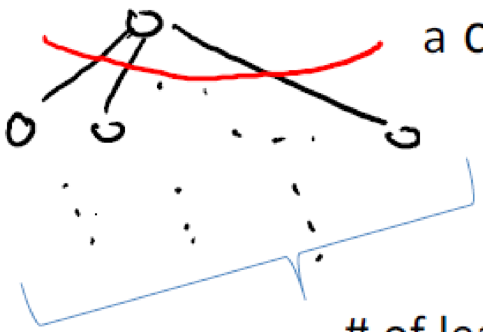
$Note \; : \quad b^{-d \log_b n} = (b^{log_b n})^{-d} = n^{-d}$

$So \; : \quad (*) = O(a^{\log_b n})$

Level 0

Level 1

Level $\log_b n$



a children

# of leaves = $a^{\log_b n}$

Which of the following quantities is equal to $a^{\log_b n}$?

- ○ The number of levels of the recursion tree.
- ○ The number of nodes of the recursion tree.
- ○ The number of edges of the recursion tree.
- ○ The number of leaves of the recursion tree.

Case 3 continued

Total work: $\leq cn^d \times \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$ (*)

$$So: \quad (*) = O(a^{\log_b n}) = O(\#\ leaves)$$

$$Note: \quad a^{\log_b n} = n^{\log_b a}$$

More intuitive

Simpler to apply

$$[Since \quad (\log_b n)(\log_b a) = (\log_b a)(\log_b n)]$$

[End Case 3]

The Master Method

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

then

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