

EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: <http://www.eecs.yorku.ca/course/3101M>
Also on Moodle

Recurrences from Divide and Conquer algorithms

$$T(n) = \begin{cases} \text{time to solve trivial problem} & \text{if } n = 1 \\ aT(n/b) + \text{time to divide} + \text{combine} & \text{if } n > 1 \end{cases}$$

E.g. Merge sort: $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(n/b) + \Theta(n) & \text{if } n > 1 \end{cases}$

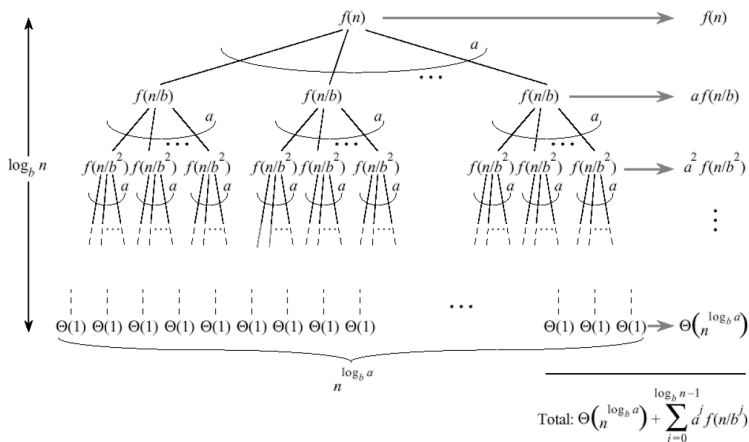
The Master Theorem

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

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

- $a \geq 1$ 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 $a = b = 2$, $f(n) = \Theta(n)$

The Master Theorem – 2



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

The Master Theorem – 3

$$\begin{aligned}T(n) &= f(n) + aT\left(\frac{n}{b}\right) \\&= f(n) + af\left(\frac{n}{b}\right) + a^2T\left(\frac{n}{b^2}\right) \\&= f(n) + af\left(\frac{n}{b}\right) + a^2f\left(\frac{n}{b^2}\right) + a^3T\left(\frac{n}{b^3}\right) \\&= \dots \\&= f(n) + af\left(\frac{n}{b}\right) + \dots + a^{\log_b n - 1}f\left(\frac{n}{b^{\log_b n - 1}}\right) \\&\quad + a^{\log_b n}T(1)\end{aligned}$$

Thus

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

The Master Theorem: Intuition

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

- 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)
- Three common cases:
 - 1 Running time dominated by cost at leaves
 - 2 Running time evenly distributed throughout the tree
 - 3 Running time dominated by cost at root

The Master Theorem: Intuition - 2

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

- Consequently, to solve the recurrence, we need only to characterize the dominant term
- In each case compare $f(n)$ with $O(n^{\log_b a})$

The Master Theorem: Case 1

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

- $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$: $f(n)$ 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 is $T(n) = \Theta(n^{\log_b a})$

The Master Theorem: Case 2

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

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

The Master Theorem: Case 3

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

- $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$: $f(n)$ grows polynomially (by factor n^ϵ) faster than $n^{\log_b a}$
- The work the root dominates
- Inverse of Case 1
- Also need a regularity condition:
 $\exists c \in (0, 1), \exists n_0 > 0, \forall n > n_0, af(n/b) \leq cf(n)$
- Thus, the overall cost is $T(n) = \Theta(f(n))$

The Master Theorem: Summary

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

- $f(n) = O(n^{\log_b a - \epsilon})$, $\epsilon > 0$: $T(n) = \Theta(n^{\log_b a})$
- $f(n) = \Theta(n^{\log_b a})$: $T(n) = \Theta(n^{\log_b a} \lg n)$
- $f(n) = \Omega(n^{\log_b a + \epsilon})$, $\epsilon > 0$: $T(n) = \Theta(f(n))$

Caveat: 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

The Master Theorem: Examples

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

- Mergesort ($a = 2, b = 2, f(n) = \Theta(n)$). Case 2:
 $T(n) = \Theta(n \lg n)$
- Binary Search (recursive): $T(n) = T(n/2) + 1$.
 $a = 1, b = 2, n^{\log_2 1} = 1, f(n) = 1 \in \Theta(1)$.
Case 2: $T(n) = \Theta(\lg n)$
- Artificial example 1: $T(n) = 9T(n/3) + n$.
 $a = 9, b = 3, f(n) = n \in \Theta(n), n^{\log_3 9} = n^2$,
 $f(n) = O(n^{2-\epsilon})$ with $\epsilon = 1$
Case 1: $T(n) = \Theta(n^2)$

The Master Theorem: More Examples

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

- Artificial example 2: $T(n) = 4T(n/2) + n^3$.

$$a = 4, b = 2, f(n) \in \Theta(n^3), n^{\log_2 4} = n^2,$$

$$f(n) = \Omega(n^{2+\epsilon}) \text{ with } \epsilon = 1$$

Case 3: $T(n) = \Theta(n^3)$ provided the regularity condition holds

Check: $4f(n/2) \leq cf(n)$ for some $c < 1$

$$\begin{aligned} 4f(n/2) &= 4(n/2)^3 \\ &= n^3/2 \\ &\leq cn^3 \text{ for } c \leq 1/2 \end{aligned}$$

The Master Theorem: Last Example

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

- Artificial example 3: $T(n) = 2T(n/2) + n \lg n$.
 $n^{\log_b a} = n^{\log_2 2} = n$, $f(n) = n \lg n$

Neither Case 2 nor Case 3.

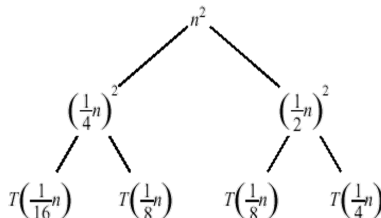
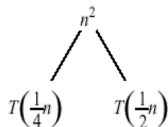
If the Master Theorem fails...

- Recursion tree approach
- Induction

Recursion Tree Method

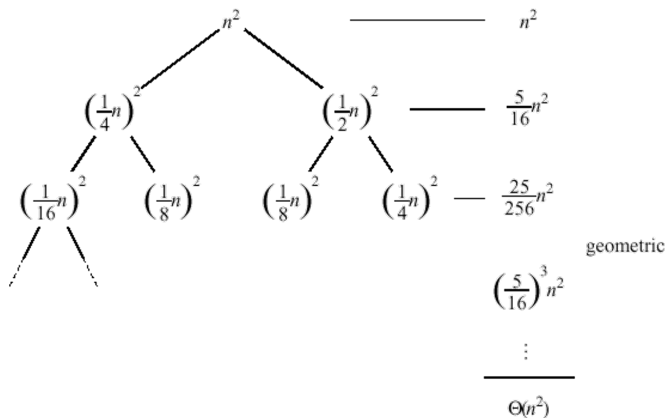
Example: $T(n) = T(n/4) + T(n/2) + n^2$

Rule: recursive term creates children, other term attached to node



Recursion Tree Method

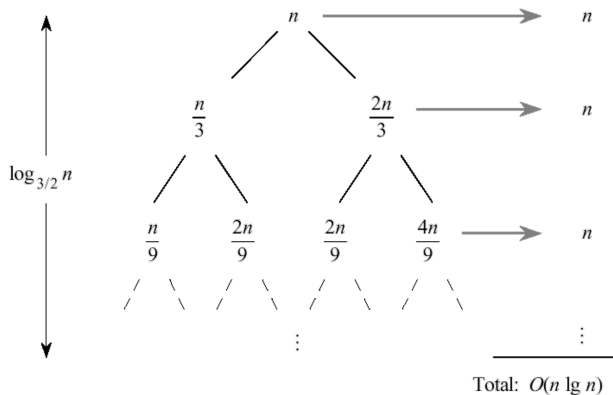
Example: $T(n) = T(n/4) + T(n/2) + n^2$



Recursion Tree Method – Another Example

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

Rule: recursive term creates children, other term attached to node



Induction

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

Attempt 1: $T(n) = O(n^3)$. Assume $T(k) \leq ck^3$ for $k \leq n/2$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^3 + n \\ &= cn^3/2 + n \\ &\leq cn^3 \text{ if } n \leq cn^3/2, n \geq n_0 \end{aligned}$$

True if $c = 2, n_0 = 1$

Induction – Tighter Bound

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

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

Attempt 1: Assume $T(k) \leq ck^2$ for $k \leq n/2$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^2 + n \\ &= cn^2 + n \\ &\not\leq cn^2 \text{ for any } c > 0 \end{aligned}$$

try to strengthen the hypothesis:

$T(n) \leq (\text{answer you want}) - (\text{something positive})$

Induction – Tighter Bound

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

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

Attempt 2: Assume $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4(c_1(n/2)^2 - c_2(n/2)) + n \\ &= c_1 n^2 - 2c_2 n + n \\ &\leq c_1 n^2 - c_2 n - c_2 n + n \\ &= c_1 n^2 - c_2 n - (c_2 - 1)n \\ &\leq c_1 n^2 - c_2 n \text{ for } c_2 \geq 1 \end{aligned}$$

Note: c_1 must be chosen to be large enough so that $T(1) \leq c_1 - c_2$.