Master Method

It is a general method for solving recurrence relations of the form

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n), \]

where \( aT\left(\frac{n}{b}\right) \) is the time for solving a subproblems, and \( f(n) \) is the time for dividing the problem into a subproblems and combining solutions of the subproblems.

This recurrence relation covers the forms involving \( T([n/b]) \) and \( T([n/b]) \). Replacing \( T\left(\frac{n}{b}\right) \) by either \( T([n/b]) \) or \( [n/b] \) does not affect the asymptotic behavior of the recurrence relation.

**Theorem 1 (Master Theorem)** Let \( a \geq 1, b > 1 \) be constants, \( f(n) \) be a function, and \( T(n) \) be defined on non-negative integers by the recurrence

\[
\begin{align*}
T(1) &= \Theta(1) \\
T(n) &= aT\left(\frac{n}{b}\right) + f(n)
\end{align*}
\]

Then,

**Case 1:** If \( f(n) = O(n^{\log_b a - \epsilon}) \) for some constant \( \epsilon > 0 \), then

\[ T(n) = \Theta(n^{\log_b a}). \]

**Case 2:** If \( f(n) = \Theta(n^{\log_b a}) \), then

\[ T(n) = \Theta(n^{\log_b a \log n}). \]
Case 3: If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some constant \( \epsilon > 0 \), and \( af\left(\frac{n}{b}\right) \leq cf(n) \) for some constant \( c < 1 \) for all sufficiently large \( n \), then

\[
T(n) = \Theta(f(n)).
\]

Remark 1 All 3 cases involve comparing \( f(n) \) with \( n^{\log_b a} \):

- If \( f(n) \) is polynomially smaller than \( n^{\log_b a} \) by a factor of \( n^\epsilon \), then Case 1 holds.
- If \( f(n) \) and \( n^{\log_b a} \) are the same (under \( \Theta \)), then Case 2 holds.
- If \( f(n) \) is polynomially larger than \( n^{\log_b a} \) by a factor of \( n^\epsilon \) and \( f(n) \) also satisfies \( af\left(\frac{n}{b}\right) \leq cf(n) \), then Case 3 holds.

Using Master Method

In all the following examples, \( T(1) = \Theta(1) \).

Example 1

\[
T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).
\]

Since \( a = b = 2, \log_b a = \log 2 = 1, f(n) = \Theta(n) = n^{\log_b a} \), this is an instance of Case 2. Therefore, \( T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n) \).

Example 2

\[
T(n) = 3T\left(\frac{n}{2}\right) + cn, c > 0.
\]

Since \( a = 3, b = 2, \log_b a = \lg 3 > 1, f(n) = \Theta(n) = O(n^{\log_b a - \epsilon}) \) for any \( \epsilon \) s.t. \( 0 < \epsilon \leq \lg 3 - 1 \), this is an instance of Case 1. Therefore, \( T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\lg 3}) \).

Example 3

\[
T(n) = 9T\left(\frac{n}{3}\right) + n.
\]

Since \( a = 9, b = 3, \log_b a = \log_3 9 = 2, f(n) = \Theta(n) = O(n^{\log_b a - \epsilon}) \) for any \( \epsilon \) s.t. \( 0 < \epsilon \leq 1 \), this is an instance of Case 1. Therefore, \( T(n) = \Theta(n^{\log_b a}) = \Theta(n^2) \).
Example 4

\[ T(n) = T \left( \frac{2}{3} n \right) + 1. \]

Since \( a = 1, \ b = \frac{2}{3} \), \( \log_b a = \log_{\frac{2}{3}} 1 = 0 \), \( f(n) = \Theta(1) = O(n^{\log_{\frac{2}{3}} 1}) = O(n^{\log_{\frac{2}{3}} a}) \), this is an instance of Case 2. Therefore, \( T(n) = \Theta(\lg n) \).

Example 5

\[ T(n) = 3T \left( \frac{n}{4} \right) + n \lg n. \]

We know that \( a = 3, \ b = 4, \ \log_b a = \log_4 3 = 0.793 < 1 \), \( f(n) = \Theta(n \lg n) = \Omega(n^{0.793+\varepsilon}) \) for any \( \varepsilon \) s.t. \( 0 < \varepsilon \leq 1 - 0.793 \). Also, \( a(f(n^\frac{1}{4})) = 3(f(n^\frac{1}{4})) = 3 \frac{n}{4} \lg \frac{n}{4} < \frac{3}{4} n \lg n \leq cn \lg n \) for any \( c \) s.t. \( \frac{3}{4} \geq c < 1 \). Clearly, we are in Case 3. Therefore, \( T(n) = \Theta(f(n)) = \Theta(n \lg n) \).

Remark 2 Important: The Master Theorem does not cover all recurrence relations of the form \( T(n) = aT(\frac{n}{b}) + f(n) \). The following is an example.

Example 6

\[ T(n) = 2T \left( \frac{n}{2} \right) + n \lg n. \]

\( a = b = 2, \ \log_b a = 1, \ O(n^{\log_{b} a}) = n, \ f(n) = \Theta(n \lg n) \).

First, this fails to be covered by Case 1, since \( n \lg n \neq O(n^{1-\varepsilon}) \) for any \( \varepsilon \) s.t. \( 0 < \varepsilon \).

Second, this fails to be covered by Case 2, since \( n \lg n \neq \Theta(n) \).

Finally, this fails to be covered by Case 3 since \( n \lg n \neq \Omega(n^{1+\varepsilon}) \) for any \( \varepsilon \) s.t. \( 0 < \varepsilon \).

Thus, we have to use other techniques other than the Master Method to solve this recurrence relation.

Divide and Conquer Algorithm Design Method

In Lecture Notes #1, we mentioned a couple of examples. We introduce several new examples.

Finding maximum and minimum

Given an array \( A[1..n] \) of \( n \) numbers, find the max and min of these numbers using pairwise comparisons.
Algorithm #1

procedure MinMax1(A)
{
    min := A[1]; max := A[1];
    for i = 2 to n do
        if A[i] > max then max := A[i]
        else if A[i] < min then min := A[i]
    print(min, max)
}

Total number of comparisons: \(2(n - 1) = 2n - 2\).

Algorithm #2

function MinMax2(A, p, q)
{
    if p = q then return (A[p], A[p]) else
            else return (A[q], A[p])
        }

        (min1, max1) := MinMax2(A, p + \(\left\lfloor \frac{q-p}{2} \right\rfloor \));
        (min2, max2) := MinMax2(A, p + \(\left\lfloor \frac{q-p}{2} \right\rfloor + 1, q\));
        if min1 ≤ min2 then min := min1 else min := min2;
        if max1 ≥ max2 then max := max1 else max := max2;
        return(min, max)
}

The main program:

\(\left(\text{min}, \text{max}\right) := \text{MinMax2}(A, 1, n)\);
print(min, max)

Total number of comparisons in the worst case (assuming \(n = 2^k\)):
\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
1 & \text{if } n = 2 \\
T([n/2]) + T([n/4]) + 2 & \text{if } n \geq 3 
\end{cases}
\]

\[
T(n) = 2T\left(\frac{n}{2}\right) + 2 \\
= 2(2T\left(\frac{n}{4}\right) + 2) + 2 \\
= 4T\left(\frac{n}{4}\right) + (2 + 4) \\
= \ldots \\
= 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i \\
= 2^{k-1}T(2) + 2 \sum_{i=0}^{k-2} 2^i \\
= 2^{k-1} + 2(2^{k-1} - 1) \text{ since } \sum_{i=0}^{k-2} 2^i = 2^{k-1} - 1 \text{ and } T(2) = 1 \\
= 2^{k-1} + 2^k - 2 \\
= \frac{3}{2}n - 2
\]

Clearly, Algorithm #2, which uses divide-and-conquer method, is more efficient than Algorithm #1.