UNIT I ROLE OF ALGORITHMS IN COMPUTING & COMPLEXITY ANALYSIS

Algorithms – Algorithms as a Technology – Time and Space complexity of algorithms – Asymptotic analysis – Average and worst-case analysis – Asymptotic notation – Importance of efficient algorithms – Program performance measurement – Recurrences: The Substitution Method – The Recursion – Tree Method – Data structures and algorithms.

RECURRENCES: THE SUBSTITUTION METHOD

The substitution method comprises two steps:

1. Guess the form of the solution using symbolic constants.
2. Use mathematical induction to show that the solution works, and ûnd the constants.

We substitute the guessed solution for the function on smaller values hence the name is the substitution method. This method is powerful, but we must guess the form of the answer. We can use the substitution method to establish either an upper or a lower bound on a recurrence. It's usually best not to try to do both at the same time. That is, rather than trying to prove a, Θ bound directly, first prove an O-bound, and then prove an Ω-bound. As an example of the substitution method, let's determine an asymptotic upper bound on the recurrence:

$$□(□) = 2□(⌊□/2⌋) + □(□)$$

This recurrence is similar to recurrence for merge sort, except for the floor function, which ensures that T(n) is defined over the integers. Let's guess that the asymptotic upper bound is the same - T(n) D = O(n lg n) and use the substitution method to prove it.

RECURRENCES: THE RECURSION-TREE METHOD

1) Substitution Method: We make a guess for the solution and then we use mathematical induction to prove the guess is correct or incorrect.

For example consider the recurrence T(n) = 2T(n/2) + n

We guess the solution as T(n) = O(nLogn). Now we use induction to prove our guess. We need to prove that T(n) <= cnLogn. We can assume that it is true for values smaller than n.

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

<= 2cn/2Log(n/2) + n
= cnLogn - cnLog2 + n
= cnLogn - cn + n
<= cnLogn

Master Method: Master Method is a direct way to get the solution. The master method works only for following type of recurrences or for recurrences that can be transformed to following type.

T(n) = aT(n/b) + f(n) where a >= 1 and b > 1

There are following three cases:

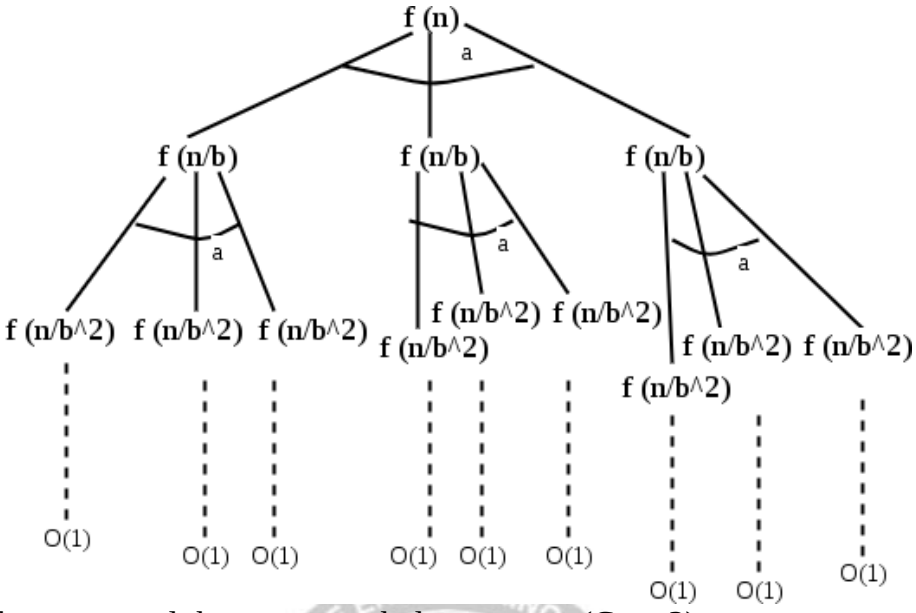
If f(n) = O(nc) where c < Logba then T(n) = Θ(nLogba)

If f(n) = Θ(nc) where c = Logba then T(n) = Θ(ncLog n)

How does this work?

Master method is mainly derived from recurrence tree method. If we draw recurrence tree of T(n) = aT(n/b) + f(n), we can see that the work done at root is f(n) and work done at all leaves is Θ(nc) where c is Logba. And the height of recurrence tree is Logbn

In recurrence tree method, we calculate total work done. If the work done at leaves is polynomially more, then leaves are the dominant part, and our result becomes the work done at leaves (Case 1). If work done at leaves and root is asymptotically same, then our result becomes height multiplied by work done at any level (Case 2). If work done at root is asymptotically



more, then our result becomes work done at root (Case 3).

Examples of some standard algorithms whose time complexity can be evaluated using Master Method

Merge Sort: T(n) = 2T(n/2) + Θ(n). It falls in case 2 as c is 1 and Logba] is also 1. So the solution is

Θ(n Logn)

Binary Search: T(n) = T(n/2) + Θ(1). It also falls in case 2 as c is 0 and Logba is also 0. So the

solution is Θ(Logn)

Notes:

It is not necessary that a recurrence of the form T(n) = aT(n/b) + f(n) can be solved using Master Theorem. The given three cases have some gaps between them. For example, the recurrence T(n) = 2T(n/2) + n/Logn cannot be solved using master method.

Case 2 can be extended for f(n) = Θ(ncLogkn)

If f(n) = Θ(ncLogkn) for some constant k >= 0 and c = Logba, then T(n) = Θ(ncLogk+1n)

The Recursion- Tree Method

Recursion Tree Method is a pictorial representation of an iteration method which is in the form of a tree where at each level nodes are expanded.

In general, we consider the second term in recurrence as root.It is

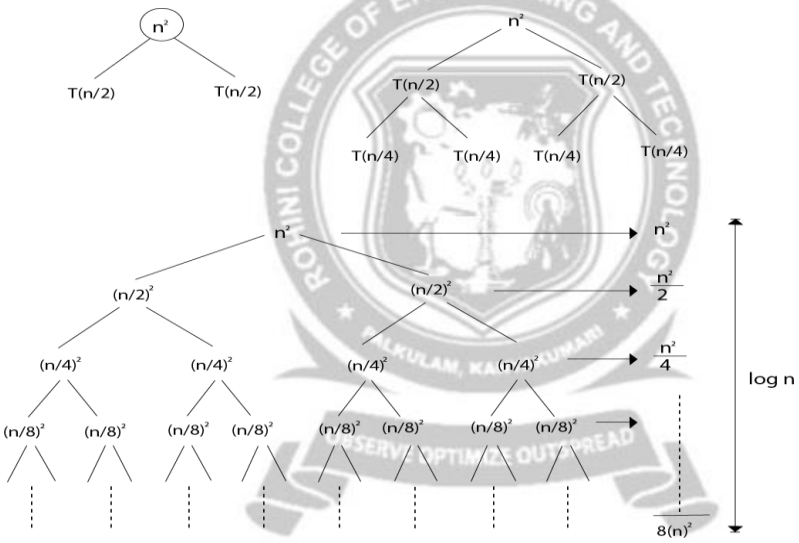
useful when the divide & Conquer algorithm is used. It is sometimes difficult to come up with a good guess. In the Recursion tree, each root and child represents the cost of a single subproblem. We sum the costs within each of the levels of the tree to obtain a set of pre-level costs and then sum all pre-level costs to determine the total cost of all levels of the recursion. A Recursion Tree is best used to generate a good guess, which can be verified by the Substitution Method.

Example 1

Consider $T(n) = 2T\left(\frac{n}{2}\right) + n2$

We have to obtain the asymptotic bound using recursion tree method.

Solution: The Recursion tree for the above recurrence is



$$T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \ldots\ldots \log n \text{ times.}$$

$$\leq n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2^i}\right)$$

$$\leq n^2 \left(\frac{1}{1-\frac{1}{2}}\right) \leq 2n^2$$

$$\mathbf{T(n) = \theta n^2}$$

Recurrence Tree Method: In this method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum the work

done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

For example consider the recurrence relation T(n) = T(n/4) + T(n/2) + cn2

cn2
/ \
T(n/4) T(n/2)

If we further break down the expression T(n/4) and T(n/2), we get following recursion tree.

cn2
/ \
c(n2)/16 c(n2)/4
/ \ / \
T(n/16) T(n/8) T(n/8) T(n/4)

Breaking down further gives us following cn2
/ \
c(n2)/16 c(n2)/4
/ \ / \
c(n2)/256 c(n2)/64 c(n2)/64 c(n2)/16
/ \ / \ / \ / \

To know the value of T(n), we need to calculate the sum of tree nodes level by level. If we sum the above tree level by level, we get the following series

T(n) = c(n^2 + 5(n^2)/16 + 25(n^2)/256) + ...

The above series is geometric progression with ratio 5/16.

To get an upper bound, we can sum the infinite series. We get the sum as (n2)/(1 - 5/16) which is O(n2)