Agenda of today's session

Recurrence Relation Solving Method

Recurrence Relation Meaning : The procedure for finding the terms of a sequence in a recursive manner is called recurrence relation.

for example :

1. Binary Search Algorithm :

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

where T(n) is the time required for binary search in an array of size n

2. Merge Sort Algorithm :

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

3. Strassens Matrix Multiplication :

$$T(n) = 7T(n/2) + n^2$$

Now the problem is to evaluate the time complexity with the help of a given Reccurence Relation.

This can be done by 3 methods :

- Master's Theorem
- Substitution Method
- Recursive Tree Method

Master's Theorem : basically take care of who is greater

$T(n) = aT(n/b) + f(n)$; a and b are positive constants

$a,b > 1$

Simplest way to evaluate this is to compare two values :

- $n^{(\log_b a)}$
- $f(n)$

Now if one of them is larger, then that's the solution of your recurrence relation.

If both are equal the solution is $T(n) = O(f(n)\log n)$

Problem 1 :

$T(n) = 8T(n/2) + n^2$

1. $n^{(\log_2 8)} \Rightarrow n^3$

2. $f(n) = n^2$

$T(n) = O(n^3)$

Problem 2 :

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

1. $n^{(\log_2 2)} = n$

2. $f(n) = n^2$

$T(n) = O(n^2)$

Problem 3 :

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

1. $n^{(\log_2 2)} = n$

2. n

1 and 2 are equal, no one is greater

$T(n) = O(f(n)\log n) = O(n\log n)$

Problem 4 :

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

1. $n^{(\log_2 1)} = n^0 = 1$

2. c

1 and 2 both are equal, no one is greater

$T(n) = O(f(n)\log n) = O(c.\log n) = O(\log n)$