• TIME COMPLEXITY

RECURRENCE RELATIONSHIP

Lets see it

A recurrence relation is an equation that recursively defines a sequence.

Fibonacci series:

$$F(n) = f(n-1) + f(n-2)$$

MASTER THEOREM

Gives the time complexity for the recurrence relation:

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

For recurrence: $T(n) = aT(n/b) + O(n^c)$

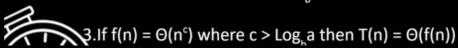
Master Theorem

For the Recurrence: $T(n) = aT(n/b) + \Theta(n^c)$, a >= 1, b > 1

There are following three cases:

1. If
$$f(n) = \Theta(n^c)$$
 where $c < Log_b a$ then $T(n) = \Theta(n^{Log_b a})$

2. If
$$f(n) = \Theta(n^c)$$
 where $c = Log_b a$ then $T(n) = \Theta(n^c Log n)$



Problems:

1.
$$T(n) = 2 T(n/2) + \Theta(n)$$

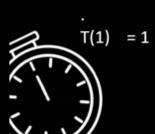
$$a = 2, b = 2, c = 1$$

 $\Rightarrow c = \log_{h} a$

Time Complexity: Θ(n Log₂n)



Recurrence Tree Method:



Recurrence Tree Method:

Adding all the terms, we get



Recurrence Tree Method:

$$T(n) = n + (n-1) + (n-2) + (n-3) + + 1$$

 $T(n) = (n * (n+1))/2$

$$T(n) = \Theta(n^2)$$



```
/*
recurrence: in best case O(nlogn)
                             hence time complexity = O(n^2)
       in worst case
      T(n) = T(n-1) + n
      T(n-1) = T(n-2) + n-1
*/
/*
recurrence:
                                level: n/2^k =1
T(n) = 2T(n/2) + n
                                      n = 2^k
T(n/2) = 2T(n/4) + n/2
                                      k =log n
T(n/4) = 2T(n/8) + n/4
T(n/8) = 2T(n/16) + n/8
                            T(n) = n+n+n+n+n+\dots \log n
                            T(n) = nlogn
T(1) = 1
*/
```