Recursion Tree Method

This involves representing the recurrence relation as a tree, where each node represents the cost of single problem, and the children of each node represent the costs of the sub problem generated by node. By summing the costs at each level of the tree and then summing across all levels, we can determine the total cost.

Consider the recurrence relation:-

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

1. Expand the Recurrence

-> At root (level 0) cost is n.

-> At level 1, there are 3 subproblems, each size $n/2$. The cost at this level is $3 \cdot n/2 = 3n/2$

-> At level 2, ~~th~~ each of 3 subproblems from the previous level again splits into 3 sub problems of size $n/4$. The cost at this level is $3^2 \cdot n/4 = 9n/4$

2. Label Costs:

* level 0 : n
* level 1 : $3n/2$
* level 2 : $9n/2$
* level k : $3^k \cdot n/2^k = n \cdot (3/2)^k$.

3. Calculate costs at Each level.

* ~~At~~ level k, the cost is $n \cdot (3/2)^k$.

4. Sum Over All Levels:

- Determine the depth of the tree ends when the Problem Size reaches 1. Solving

  $n/2^k = 1$ gives $k = \log_2 n$.

- Sum the costs across all levels.

$$T(n) = n + \frac{3n}{2} + \frac{9n}{4} + \cdots + n\cdot\left(\frac{3}{2}\right)^{\log_2 n}.$$

- This is a geometric series with ratio 3/2. The sum of geometric Series $a + ar + ar^2 + \cdots + ar^m$ is $a\,\dfrac{r^{m+1}-1}{r-1}$.

- Here, $a = n$, $r = 3/2$ and the no. of terms ie $\log_2 n + 1$

$$T(n) = \frac{\left(\frac{3}{2}\right)^{\log_2 n + 1} - 1}{\frac{3}{2} - 1}$$

- Simplifying, $\left(\frac{3}{2}\right)^{\log_2 n} = 3^{\log_2 n} = n^{\log_2 3}$

$$T(n) = n \cdot \frac{n^{\log_2 3} \cdot \frac{3}{2} - 1}{\frac{1}{2}} = 2n\left(n^{\log_2 3 - 1} \cdot \frac{3}{2} - \frac{1}{2}\right)$$

$$= 2n^{\log_2 3}$$

- Thus, $T(n) = O(n^{\log_2 3})$.

—— d ——

Master Theorem:-

1. identify a,b and f(n) in the recurrence

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

2. Compute $\log_b a$.

3. Compare f(n) with $n^{\log_b a}$.

→ case 1: if $f(n) = O(n^c)$ where $c < \log_b a$ 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~~ if $f(n) = \Omega(n^c)$ where $c > \log_b a$, and

if $af(n/b) \leq k f(n)$ for some $k < 1$, then

$T(n) = \Theta(f(n))$.

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

where, $a \geq 1$ is the no. of recursive calls per level.

$b > 1$ is the factor by which the problem size is divided.

f(n) is the cost outside the recursive calls.

Example :-

case 1:-

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

* Here, $a = 2$, $b = 2$ and $f(n) = n$.

* $\log_b a = \log_2 2 = 1$

* Since $f(n) = O(n^1)$ and $c = 1$.

case2: $f(n) = \Theta(n^{\log_b a})$

if $f(n)$ grows Polynomially at the same rate as $n^{\log_b a}$, then the overall complexity is dominated by the combined work done at all levels of the recursion tree. Thus, soultion is,

$T(n) = \Theta(n^{\log_b a} \log n)$

Example:-

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

* Here, $a = 2$, $b = 2$ and $f(n) = n$.

$\log_b a = \log_2 2 = 1$

$\because$ $T(n) = \Theta(n \log n)$.

case 3:- $f(n) = \Omega(n^c)$ where $c > \log_b a$

if $f(n)$ grows polynomially faster than $n^{\log_b a}$, then overall complexity is dominated by the cost of the work done outside the recursive calls. However, for this case to apply, an additional regularity condition must be satisfied :

$af(n/b) \le k f(n)$ for some constant $k<1$ and sufficiently large n. If these conditions are met, the solution is:

$T(n) = \Theta(f(n))$

Example :-

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

Here,

$a=2,\ b=2,\ f(n)=n^2$

$\log_b a = \log_2 2 = 1.$

Since, $f(n) = \Omega(n^2)$ and $c=2>1$ ~~and~~ $= \log_b a$, we check the regularity condition:

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

Since $n^2/2 \le kn^2$ for $k = 1/2 < 1$, the regularity condition is satisfied. Therefore, we use Cas 3:

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