Assignment 4

Chandan Roy

================================

Question 1. $T(n) = 2T(n/2) + n$

<u>recursion tree method</u>

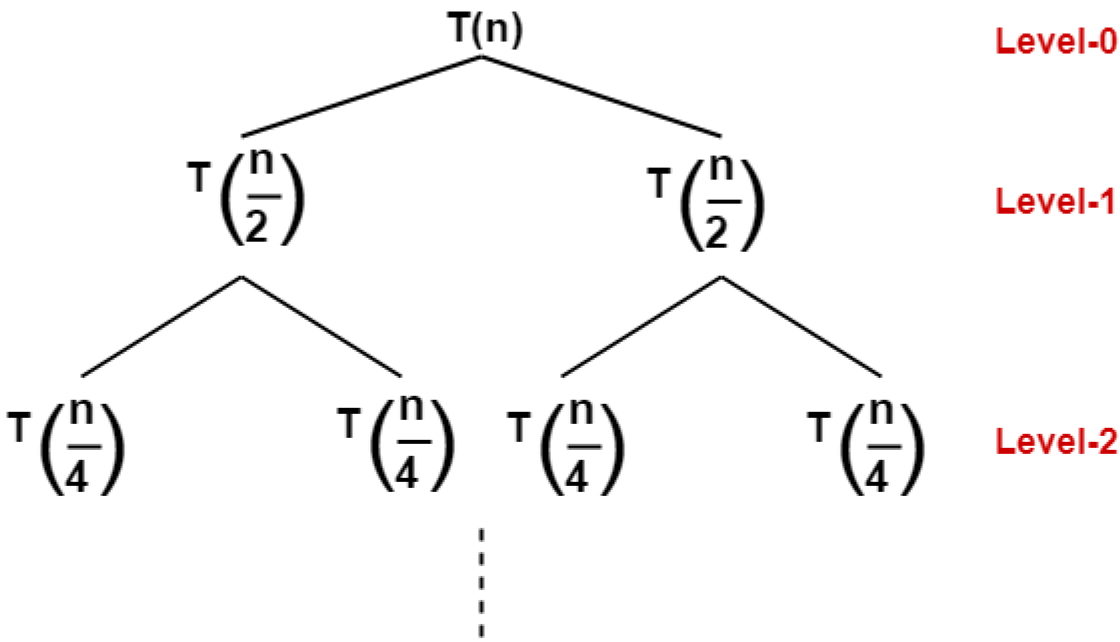
<u>Step-01:</u>

Draw a recursion tree based on the given recurrence relation.

The given recurrence relation shows-

- A problem of size n will get divided into 2 sub-problems of size n/2.
- Then, each sub-problem of size n/2 will get divided into 2 sub-problems of size n/4 and so on.
- At the bottom most layer, the size of sub-problems will reduce to 1.

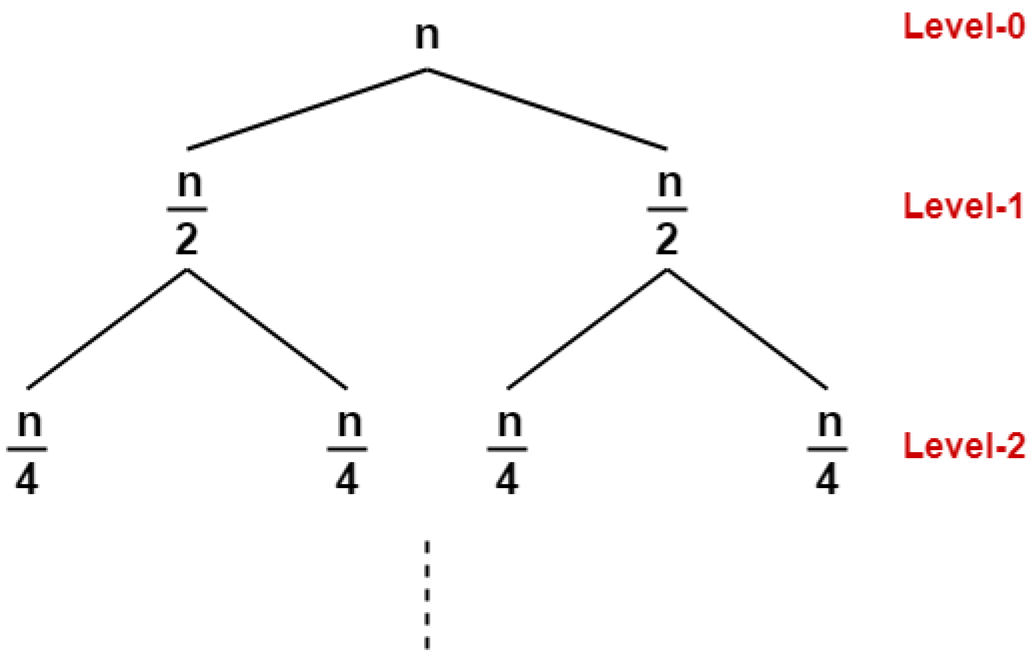
This is illustrated through following recursion tree-



The given recurrence relation shows-

- The cost of dividing a problem of size n into its 2 sub-problems and then combining its solution is n.
- The cost of dividing a problem of size n/2 into its 2 sub-problems and then combining its solution is n/2 and so on.

This is illustrated through following recursion tree where each node represents the cost of the corresponding sub-problem-



Step-02:

Determine cost of each level-

- Cost of level-0 = n
- Cost of level-1 = n/2 + n/2 = n
- Cost of level-2 = n/4 + n/4 + n/4 + n/4 = n and so on.

Step-03:

Determine total number of levels in the recursion tree-

- Size of sub-problem at level-0 = $n/2^0$
- Size of sub-problem at level-1 = $n/2^1$
- Size of sub-problem at level-2 = $n/2^2$

Continuing in similar manner, we have-

Size of sub-problem at level-i = $n/2^i$

Suppose at level-x (last level), size of sub-problem becomes 1. Then-

$$n / 2^x = 1$$

$$2^x = n$$

Taking log on both sides, we get-

$$x\log 2 = \log n$$

$$x = \log_2 n$$

∴ Total number of levels in the recursion tree = $\log_2 n + 1$

Step-04:

Determine number of nodes in the last level-

- Level-0 has 2^0 nodes i.e. 1 node
- Level-1 has 2^1 nodes i.e. 2 nodes
- Level-2 has 2^2 nodes i.e. 4 nodes

Continuing in similar manner, we have-

$$\text{Level-}\log_2 n \text{ has } 2^{\log_2 n} \text{ nodes i.e. } n \text{ nodes}$$

Step-05:

Determine cost of last level-

$$\text{Cost of last level} = n \times T(1) = \theta(n)$$

Step-06:

Add costs of all the levels of the recursion tree and simplify the expression so obtained in terms of asymptotic notation-

$$T(n) = \underbrace{\{\, n + n + n + \ldots\ldots. \,\}}_{\text{For } \log_2 n \text{ levels}} + \theta\,(n)$$

$= n \times \log_2 n + \theta(n)$

$= n\log_2 n + \theta(n)$

$= \boldsymbol{\theta(n\log_2 n)}$

Using Master Theorem

$F(n)=af(n/b)+cn^d$ -------------general equation

Comparing above equation with general equation, we get

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

$b^d = 2$

so, $a= b^d$

From Master theorem we know that time complexity of f(n)

$= O(n^d * \log n)$ if $a= b^d$.

So, the time complexity of the function is O(n *logn).

Recurrence Relation

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

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

$= 2(2T(n/4) + n/2) + n = 4T(n/4) + n + n = 4T(n/4) + 2n$

$= 4(2T(n/8) + n/4) + 2n = 8T(n/8) + n + 2n = 8T(n/8) + 3n$

$= 8(2T(n/16) + n/8) + 3n = 8T(n/16) + n + 3n = 16T(n/16) + 4n$

$= 32T(n/32) + 5n$ …………….....

$= n*T(1) + \log2(n)*n$

$= O(n*\log2(n))$

Substitution Method

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

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

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

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

|

| K times

|

$2^kT(n/ 2^k) + kn$

$T(1)=1$

$\Rightarrow$ $n/ 2^k =1$

$\Rightarrow$ $K=\log 2^n$

2^log 2^n T(n/ 2^k) + log 2^nn

T(1)=1

= n*T(1) + log2(n)*n

=n + log2(n)*n

= O(n*log2(n))