Algorithms and Problem Solving (15B11CI411)
EVEN 2022

Module 1: Lecture 4

Jaypee Institute of Information Technology (JIIT)
A-10, Sector 62, Noida

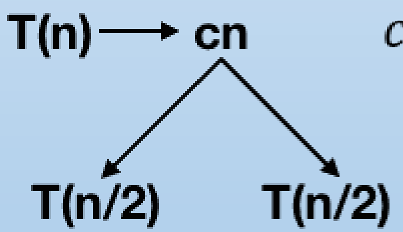
Solving Recurrence relation using Recursion Tree method

Recursion Tree - A visual representation of recursive call hierarchy where <u>each node represents the cost of a single subproblem</u>

Example :

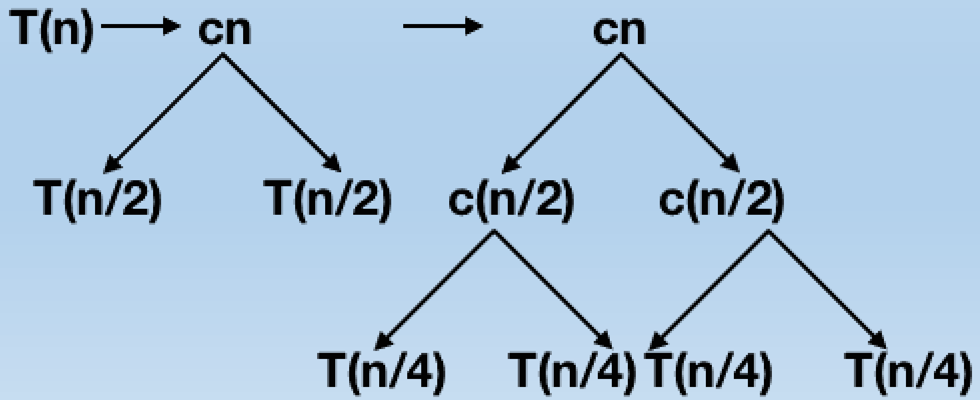
$$T(n) = cn + 2T\left(\frac{n}{2}\right)$$

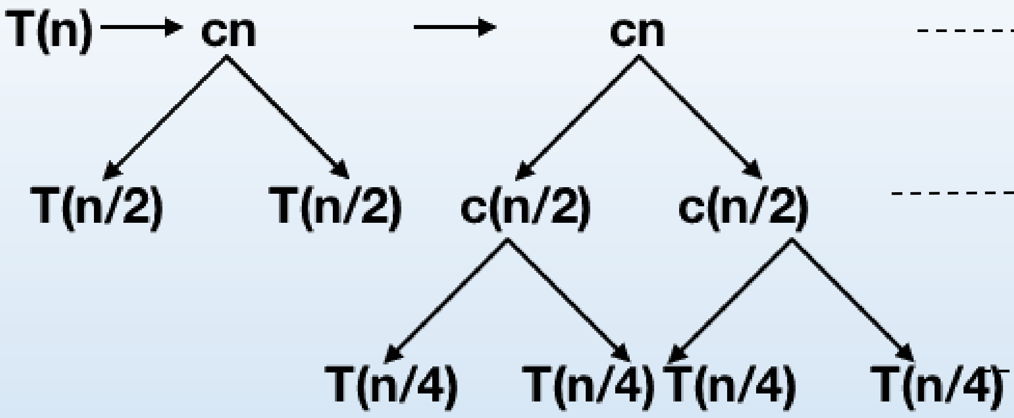
$T(n)$ is the running time of the entire algorithm and this running time is broken into two terms - cn and $2T\left(\frac{n}{2}\right)$



cn is the constant time involved in the algorithm except the two subproblems

$T\left(\frac{n}{2}\right)$ *is further broken into*



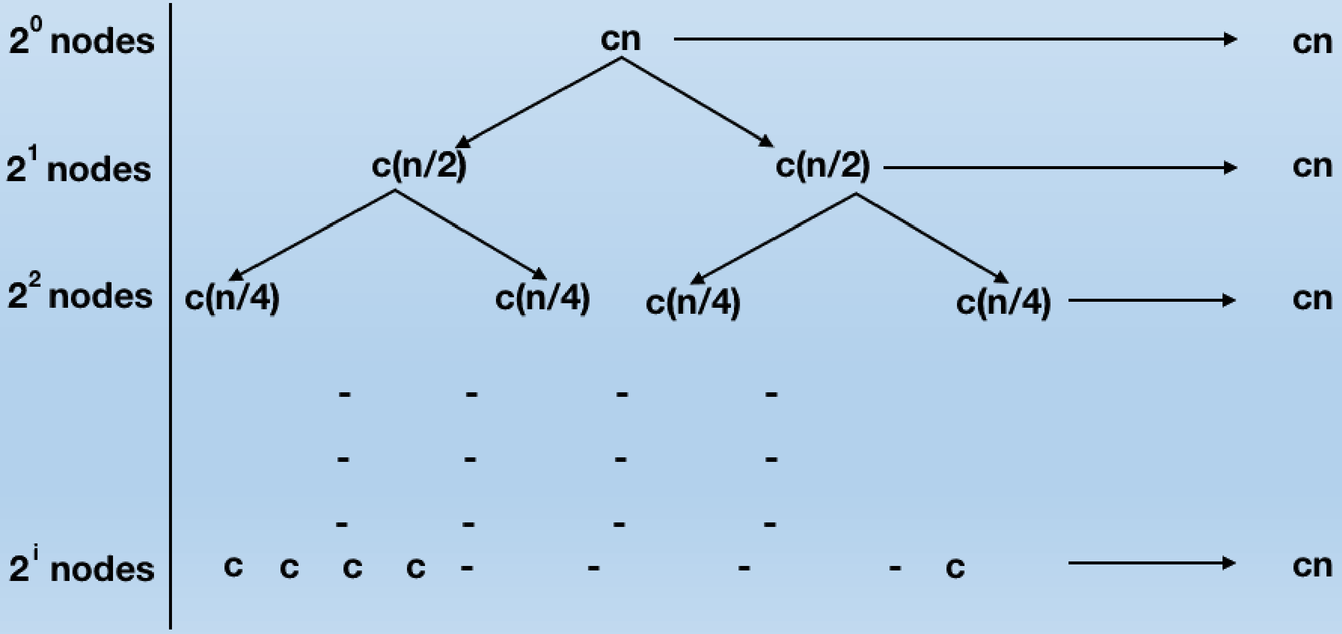


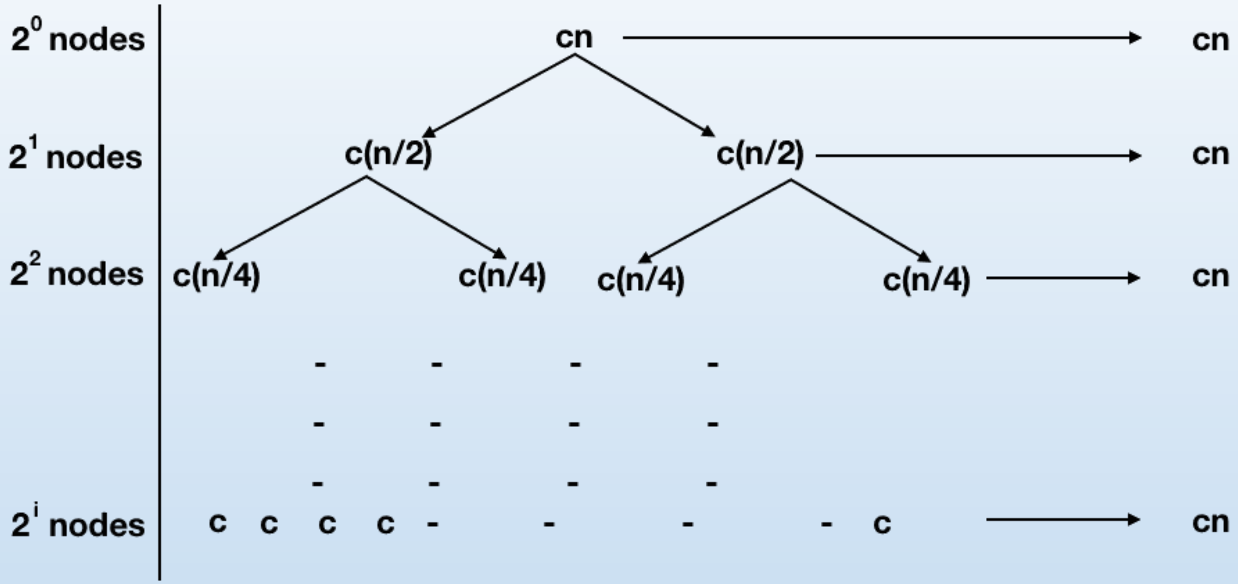
$Cost\ - cn$

$Cost\ - c\left(\frac{n}{2}\right) + c\left(\frac{n}{2}\right) = cn$

$Cost\ - c\left(\frac{n}{4}\right) + c\left(\frac{n}{4}\right) + c\left(\frac{n}{4}\right) + c\left(\frac{n}{4}\right) = cn$

In general , for a level i there are 2^i nodes and cost of each level is computed as $2^i.\left(\frac{cn}{2^i}\right)$





<u>Steps to compute the total time taken:</u>

1) Compute time taken by each level
2) Calculate the total number of levels
3) Time taken = total number of levels and time taken by each level

If the last level of the tree is level i, then

$$2^i = n$$
$$=> \log(n) = i$$

The total number of levels are $1 + \log(n)$ (counting of the levels is starting from 0).

The total time taken = $cn + cn + \ldots + (1+\log(n))cn = (1+\log(n))cn = (cn + c(n\log(n))$

By ignoring the lower order terms and the constant, the order of the growth to be of $O(n\log n)$

Master theorem

- General formula that works if recurrence has the form $T(n) = aT(n/b) + f(n)$
  - a is number of subproblems
  - n/b is size of each subproblem
  - $f(n)$ is cost of non-recursive part

Master Theorem

- Consider a recurrence of the form

  $T(n) = a\,T(n/b) + f(n)$

with $a>=1$, $b>1$, and $f(n)$ eventually positive.

Case 1 : If $f(n) = O(n^{\log_b(a)-\varepsilon})$ for some $\varepsilon > 0$ then

Solution : $T(n)=\Theta(n^{\log_b(a)})$.

Case 2: If $f(n) = \Theta(n^{\log_b(a)})$ then

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

Case 3: If $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$; $\varepsilon > 0$

and

$af(n/b) <= cf(n)$ for some constant $c<1$ $\forall n$ then

Example : Solve T(n)=$T(n) = 9T\left(\frac{n}{3}\right)$ +n

Solution :

Step 1 : Check a>=1 and b >1, in this case a=9 ,b =3

Step 2 : Compare $n^{\log_b a}$ and f(n)

(i) if they are in the same θ class then apply case 2

(ii) if f(n) is polynomialy smaller than nlog$_b\, a$ (by a factor of n^{ε}) then apply case 1

(iii) if f(n) is polynomialy larger than nlog$_b\, a$ (by a factor of $\frac{1}{n^{\varepsilon}}$) then apply case 3

$n^{\log_b a}$ =$n^{\log_3 9}$ =$n^{\log_3 3^2}$ =n^2

f(n) is polynomial smaller than $n^{\log_b a}$, hence case 1 applies

f(n)=O($n^{\log_b a-\varepsilon}$) for some $\varepsilon > 0$

=> O($n^{2-\varepsilon}$) , assume ε=1, then

=> O(n)=f(n),

hence, $T(n)$=**T(n)=Θ(n$^{\log b(a)}$)=θ(n^2).**

Example : Solve T(n)=$T(n) = 3T\left(\frac{n}{4}\right)$ +nlog n

Solution :

Step 1 : Check a>=1 and b >1, in this case a=3 ,b =4 , f(n)= nlog n

Step 2 : Compare $n^{\log_b a}$ and f(n)

$n^{\log_b a}$ =$n^{\log_4 3}$ $\approx n^{0.79}$

f(n) is polynomial larger than $n^{\log_b a}$, hence case 3 applies

Case 3: If f(n)=$\boldsymbol{\Omega(n^{\log_b a+\varepsilon})}$ for some $\boldsymbol{\varepsilon > 0}$ and

$af\left(\frac{n}{b}\right) \leq c.f(n)$;c<1,$\forall n$ $then\ T(n) = \theta(f(n))$

$nlogn$ grows faster than $\boldsymbol{\Omega(n^{\log_b a+\varepsilon}}$, the first condition case (3) holds applying second condition

$3 * \left(\frac{n}{4}\right)\log\frac{n}{4} \leq c.n\log n$;$\forall n, c < 1$

Assume c=$\frac{3}{4}$

$\frac{3}{4} * n\log\frac{n}{4} \leq \frac{3}{4}.n\log n$ => satisfies the second condition also.

hence, $T(n)$=$\theta\big(f(n)\big) = \theta(n\log n)$