Parul® University

Design and analysis of Algorithms

Pushpak Raval

Assistant Professor CSE Department

Unit-1

Recureence tree method

Introduction

- It is Graphical Representation based method for finding the time complexity of recurrence equation.
- It is divide & conquer based approach.
- Each node of Tree represents the cost of single sub problem.
- And Each level cost will be sum up as a pre level cost.
- Now finally all pre level cost will be sum up to find Total cost of Problem.

- **How To Derive the Tree?**
- The root is indicated by second term of recurrence equation.

- Example: $T(n) = 2T(n/2) + n^2$
- The root will be - n^2
- Now the problem will be divided in to sub problems.
- So $2T(n/2) = T(n/2) + T(n/2)$

Introduction

- **Other Example :T(n)=T(n/3)+T(2n/3)+n**
- Here Root Will be: n
- And sub problem will be: T(n/3) & T(2n/3)

- **Now Guess Example :T(n)=T(n/4) + T(n/2) + cn^2**
- Here Root Will Be: ?
- And Sub problems will be: ?

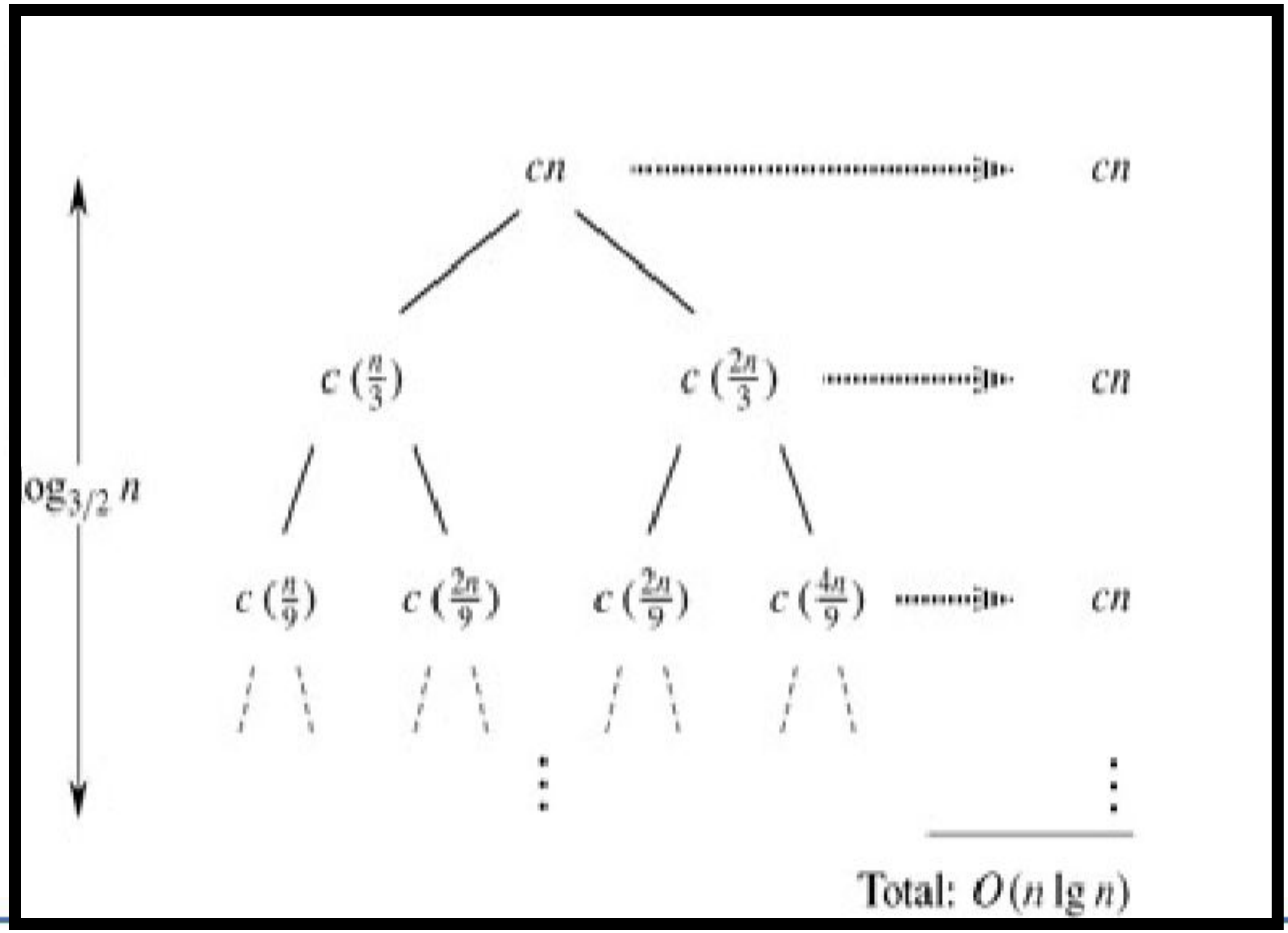
Deriving Recurrence Tree and Finding Cost

- **Example: T(n) = T (n/3) + T (2n/3) + cn.**

Steps For deriving Recurrence tree:

- **Example: Recursion tree T(n) = T (n/3) + T (2n/3) + cn.**
- **Step-1**
- Root Will be: ?
- Two sub problem will be : ?
- **Step-2**
- Let’s Start to divide the problem and find out pre-level cost (cost at each level)

- **Tree with sub problem.**
- Two sub problem per each given node.(n/3),(2n/3)



- **Step-3**
- **Lets sum up all the pre-level cost and find the final cost for the given equation.**
- **You can find that in given Tree:**

  - **At Level-0**
  - Root will be n –(Pre-level cost at level-0 :Cn)
  - Each node represent cost of sub problem it can be seen as(n/3),(2n/3)

  - **At level-1**
  - At first level Pre- level cost will be cn= (n/3)+(2n/3)=(3n/3)=n(where c is a Constant)

- **At level-2**

- At level-2 again (n/3) will be divided in to n/9 and 2n/9
- And (2n/3) will be divided in to 2n/9 and 4n/9
- Total cost at level 2(pre-level cost at level -2)= c(n/9+2n/9+2n/9+4n/9)=c9n/9=cn

- Find pre-level cost for all level (Which will eventually be: cn).

- For final cost lets sum up all pre-level cost. And Find upper bound.

Lets Analyse & Sum up

- Omit floor and ceiling functions for simplicity. As before, we let c represent the constant factor in the O(n) term.
- When we add the values across the levels of the recursion tree, we get a value of cn for every level.
- The longest path from the root to a leaf is $n \rightarrow (2/3)n \rightarrow (2/3)^2 n \rightarrow \cdots \rightarrow 1$. Since $(2/3)^k n = 1$ when $k = \log_{3/2} n$, the height of the tree is $\log_{3/2} n$.
- The solution to the recurrence to be at most the number of levels times the cost of each level, or $O(cn \log_{3/2} n) = O(n \lg n)$.
- The total cost is evenly distributed throughout the levels of the recursion tree. There is a complication here: we have yet to consider the cost of the leaves. If this recursion tree were a complete binary tree of height $\log_{3/2} n$, there would be leaves.

- Since the cost of each leaf is a constant, the total cost of all leaves would then be , which is ω(n lg n).
- This recursion tree is not a complete binary tree, however, and so it has fewer than leaves. Moreover, as we go down from the root, more and more internal nodes are absent.
- Consequently, not all levels contribute a cost of exactly cn; levels toward the bottom contribute less.

- This is just guess so let’s verify.

Let's verify with substitute method

- **T(n)<=T(n/3)+T(2n/3)+cn**
- T(n) <=d(n/3)lg(n/3)+ d(2n/3)lg(2n/3)+cn
  = (d(n/3)lg n-d(n/3)lg 3) +(d(2n/3) lg n –d(2n/3) lg(3/2)) +cn
  = dn lg n –d((n/3) lg 3 + (2n/3) lg (3/2) + cn
  = dn lg n –d((n/3) lg 3 + (2n/3) lg (3/2) –(2n/3) lg 2) +cn
  =dn lg n-dn(lg 3 – 2/3) + cn
  <=dn lg n

Class Exercise

- **Draw the recursion tree for**

  $T(n) = 3T(n/4) + cn^2$.

  where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

Another Example

- **Draw the recursion tree for**

  T (n) = 4T (⌊n/2⌋)+cn, where c is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

Unit-1

Master Theorem

Terminology

- Direct way to find the Solution of given Recurrence equation.
- It can apply only where the recurrence equation, can be transferred in following form or is of this type.

  T(n) = aT(n/b) + f(n) where a >= 1 and b > 1 and f(n) is a asymptotic function .

  where we interpret n/b to mean either ⌊n/b⌋ or ⌈n/b⌉ is a size of problem. Then T (n) can be bounded asymptotically as follows.
- **There are following three cases:**

  1. If $f(n) = \Theta(n^c)$ where $c < \text{Log}_b a$ then $T(n) = \Theta(n^{\text{Log}_b a})$
  2. If $f(n) = \Theta(n^c)$ where $c = \text{Log}_b a$ then $T(n) = \Theta(n^c \text{Log } n)$
  3. If $f(n) = \Theta(n^c)$ where $c > \text{Log}_b a$ then $T(n) = \Theta(f(n))$

Cont...

- Examples of some standard algorithms whose time complexity can be evaluated using Master Method.

- Merge Sort: $T(n) = 2T(n/2) + \Theta(n)$. It falls in case 2 as c is 1 and $Log_b a$ is also 1. So the solution is $\Theta(n \log n)$
- Binary Search: $T(n) = T(n/2) + \Theta(1)$. It also falls in case 2 as c is 0 and $Log_b a$ is also 0. So the solution is $\Theta(\log n)$

Cont...

- Master method is mainly derived from recurrence tree method. If we draw recurrence tree of T(n) = aT(n/b) + f(n), we can see that the work done at root is f(n) and work done at all leaves is $\Theta(n^c)$ where c is $Log_b a$. And the height of recurrence tree is $Log_b n$.

- In recurrence tree method, we calculate total work done. If the work done at leaves is polynomially more, then leaves are the dominant part, and our result becomes the work done at leaves (Case 1).

- If work done at leaves and root(f(n)) is asymptotically same, then our result becomes height multiplied by work done at any level (Case 2).

- If work done at root(f(n)) is asymptotically more, then our result becomes work done at root (Case 3).

Case 1

- **Lets take. T (n) = 9T(n/3) + n.**

  In given recurrence,

  we have find a = 9, b = 3, f (n) = n.

  so $\Theta(n^{Log_b a})=\Theta(n^2)$.

  Since f(n)=n , where c= 1

  we can apply case 1 of the master theorem and conclude that the solution is T (n) = $\Theta(n^2)$.

Case 2

- **Lets take. T (n) = T (2n/3) + 1.**

  In given recurrence,

  we have find a = 1, b = 3/2, f (n) = 1.

  so $\Theta(n^{Log_b a})=\Theta(n^{log_{3/2} 1}) = n^0 = 1$

  Since f(n)=1.

  we can apply case 2 of the master theorem and conclude that the solution is T (n) = Θ(log n).

Case 3

- **T(n) = 3T(n/4) + n lg n.**

we have a = 3, b = 4, f (n) = n lg n, and $\Theta(n^{Log_b a}) = \Theta(n^{0.793})$. Since $f(n) = \Theta(n^{0.793+c})$

Where c=0.217 so case 3 can be apply so

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

For sufficiently large n, a $f(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$. Consequently, by case 3, the solution to the recurrence is $T(n) = \Theta(n \lg n)$.

Where master method can not apply?

- **T(n/2)=2T(n/2)+n lg n**

- Even though it has the proper form: a=2,b=2,f(n)=n lg n, and $n^{\log_b a}$ =n
- Now f(n)=n lg n which is greater than $n^{\log_b a}$ =n
- This is actually not a polynomially larger. That is the ration is (n lg n)/n=lg n is asymptotically less that n^c for any c
- This case fall between case 2 and case 3

Class exercise

1. T (n) = 4T(n/2) + n.
2. T (n) = 4T(n/2) + n^2.
3. T (n) = 4T(n/2) + n^3.
4. Can the master method be applied to the recurrence T (n) = 4T(n/2) + n2 lg n? Why or why not? Give an asymptotic upper bound for this recurrence.