Advanced Algorithm Analysis

Department of Computer Science
Swat College of Science & Technology

CS Course : Advanced Algorithm Analysis
Course Instructor : Muzammil Khan

Assignment (Due date: by the Next Lecture)

- Select a current research paper (discussing advance algorithm) in domain of your interest
- Summarize the paper
  - About One & Half page
- Be ready for presenting it in the class
  - Presentation duration upto 7-10 minutes
- Submit
  - Printed copy
  - Soft copy
- Best summary will be award with 2% marks

Chapter 6

Recurrence
Or
Recurrence Relation

Recurrence

- Also called recurrence relation
- Recurrence is
  - An equation or inequality that describes a function in terms of its values on smaller inputs
  - Characteristics
    - The function is defined over a set of ***natural number***
    - The definition include a ***base value*** for function
      - Also call *boundary condition*
- Example "Merge Sort"

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Recurrence (Cont...)

- When an algorithm contain recursive calls
  - Its running time often described by recurrence equation
  - Which describes running time for problem of size ***n***
- Example "Merge Sort"

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- It can be solved by using mathematical loops
- Algorithms, recursive in nature
  - Usually follow divide and conquer strategy

Examples

- The factorial function ***f(n) = n!*** can be expressed as
  - By recurrence

$$f(n) = n.f(n-1)$$
$$f(0)=1 \qquad \text{(boundary condition)}$$

- The Fibonacci Sequence ***f(n)*** can be define as
  - By recurrence

$$f(n) = f(n\text{-}1) + f(n\text{-}2)$$
$$f(0) = 0, \quad f(1)=1 \quad \text{(boundary condition)}$$

Examples (Cont...)

- Recurrence for the running time of common algorithms
  - ***T(n)*** is the running time of a problem of size ***n***
- Recurrence relation for ***decrease-and-conquer*** problem

$$T(n) = T(\underbrace{n-1}_{\text{Subproblem size}}) + \underbrace{cn}_{\text{Cost of decreasing}}$$

- Recurrence relation for ***divide-and-conquer*** problem

$$T(n) = \underbrace{8}_{\text{Number of subproblems}} T(n/\underbrace{4}_{\text{Subproblem size}}) + \underbrace{c\,n^2}_{\text{Cost of dividing and combining}}$$

How to Solve Recurrence

- There are four techniques to solve recurrence relation
  - Iteration method
  - Substitution method
  - Recursion Tree method &
  - Master Theorem method

Iteration Method

- In iteration method
  - The recurrence is solved by Top-Down Approach
- Involve the following steps
  1. Using definition
     - Equations are set up for arguments *n, n-1, n-2, …*
  2. On reaching the bottom level the ***boundary condition*** is applied
  3. The equations are ***summed up***
  4. Finally, the solution is obtain by
     - Canceling out the ***identical terms*** on both sides of the iterated equation

Iteration Method (Cont...)

- The method is particularly useful in
  - ***Decrease and Conquer*** problems
- In other cases
  - Additional efforts are required for same terms cancelation
- Example
  - Next slide

Iteration Method Example 1

- Here is linear search recurrence, which is based on decrease-&-conquer algorithms

$$T(0)=0$$
$$T(n)=T(n-1)+c$$

Iterating the recurrence:.

$$\begin{aligned}
T(n) &= T(n-1) + c\\
T(n-1) &= T(n-2) + c\\
T(n-2) &= T(n-3) + c\\
&\dots\dots\dots\dots\dots\dots\dots\dots\\
T(3) &= T(2) + c\\
T(2) &= T(1) + c\\
T(1) &= T(0) + c
\end{aligned}$$

Iteration Method Example 1 (Cont...)

- Adding both sides of the equations, and canceling equal terms

$T(n) = c + c + \dots\dots\dots + c$

Or, $T(n)=n.c$

It follows that $T(n)=\theta(n)$

Iteration Method Example 2

- In selection sort the largest element is searched and placed at last position
  - This procedure is repeatedly applied to sub-arrays
- The recurrence of the selection sort algorithm is

$$T(0)=0$$
$$\underbrace{T(n)}_{\text{Sorting } n \text{ elements}} = \underbrace{T(n-1)}_{\text{Sorting } n-1 \text{ elements}} + \underbrace{c.n}_{\text{Finding maximun and exchanging}}$$

- Iterating the recurrence:

$$T(n) = T(n-1) + c.n$$
$$T(n-1) = T(n-2) + c.(n-1)$$
$$T(n-2) = T(n-3) + c.(c-2)$$
$$\dots \dots \dots \dots \dots \dots \dots \dots$$

Iteration Method Example 2 (Cont...)

$$T(3) = T(2) + c.3$$
$$T(2) = T(1) + c.2$$
$$T(1) = T(0) + c.1$$

- Adding both sides of the equations, and canceling equal terms:

  $T(n) = c(1 + 2 + 3 + \dots \dots \dots + n)$
- Summing the arithmetic series:

  $T(n)=c.n(n+1)/2$
- It follows that

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

Substitution Method

- It is a symmetric procedure for solving recursive equation
  - It follows Top-down approach (as iteration method)
- Involve the following steps
  1. In the recurrence
     - Values are plugged in repeatedly on the right hand side of the equation
  2. The procedure is repeated until the ***base case*** is reached
  3. The iteration step guarantee some kind of ***pattern or a series***
  4. The summation for the series is analyzed to determine the asymptotic behavior
- Is useful method for both
  - Decrease-and-conquer and divide-and-conquer approaches

Substitution Method Example 1

- The recurrence of binary search algorithm is

$$T(1) = c$$
$$T(n)=T(n/2) + c, \quad n > 1$$

1^{st} substitution, $T(n) = T(n/2) + c = T(n/2^1) + c$

2^{nd} substitution, $T(n) = T(n/4) + 2c = T(n/2^2) + 2.c$

3^{rd} substitution, $T(n) = T(n/8) + 3c = T(n/2^3) + 3.c$

....

k^{th} substitution, $T(n) = T(n/2^k) + k.c$

It will be seen that, on continuing, the base case $T(1)$ is reached when $\boldsymbol{n / 2^k = 1}$, or $\boldsymbol{n=2^k}$

i.e. $\boldsymbol{k = lg\ n}$

Substituting for k, we get

$$T(n) = T(1) + lg\ n.\ c$$
$$= c + lg\ n.\ c$$

Thus, $T(n) = \theta(lg\ n)$

Substitution Method Example 2

- The recurrence of *finding largest element* in the array

$T(1) = c$

$T(n)=2T(n/2) + c, \quad n > 1$

Initially:

$T(n) = 2.T(n/2) + c \quad = 2.T(n/2^1) + 2^0 c$

Substituting for $T(n/2)$:

$T(n) = 2.[2.T(n/4)+c] + c$

$\quad = 4.T(n/4) + 3c \quad = 2^2T(n/2^2) + (2^0 + 2^1).c$

Substituting for $T(n/4)$:

$T(n) = 4.[2\ T(n/8)+c] + 3c$

$\quad = 8.T(n/8) + 7.c \quad = 2^3T(n/2^3) + (2^0+2^1+2^2).c$

After k^{th} substitution,

$T(n) = 2^kT(n/2^k) + (2^0+2^1+2^2+\dots\dots 2^{k-1}).c$

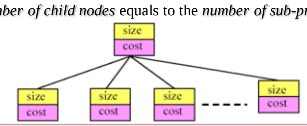
$\quad = 2^kT(n/2^k) + (2^k - 1)c$

Substitution Method Example 2 (Cont...)

- Continuing,
- it will be seen that the base case $T(1)$ is reached when
  - $\boldsymbol{n / 2^k = 1}$, or $\boldsymbol{n=2^k}$,
- Substituting for 2^k, we get
  - $T(n) = n.T(1) + (n-1).c = n.c + n.c - c$
  - $T(n) = 2n.c - c$
  - $T(n) = \theta(n)$

Recursion Tree

- Recursion tree provide visual tool for solving recursive equation
  - Involve 4 steps
- **Step # 1** : The recurrence is expressed in a *hierarchical way*
  - Using a *tree structure*, such that
    - Each node contained two fields
      - The *size field* and *cost field*
    - The *number of child nodes* equals to the *number of sub-problems*



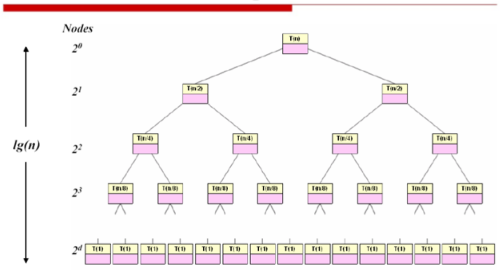
Recursion Tree (Cont...)

- **Step # 2**
  - The *size field* of a node is set by plugging
    - The size of the parent into the relation
- **Step # 3**
  - The *cost field* is set by substituting node into *cost function of the relation*
- **Step # 4**
  - The solution is found by *summing the cost over all* nodes of the tree

Recursion Tree Example 1

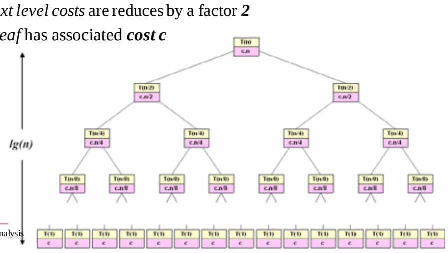
- ***T(n) = 2 T(n/2) + cn*** , n > 1 and ***T(1) = c*** , n = 1
- *Constructing tree structure*
  - Fully extended tree has 2^d nodes, called ***leaves*** and ***d*** is *tree depth*
  - At bottom level $T(n/2^d) = T(1)$,
  - It follows $2^d = 1$, or $2^d = n$, i.e. $d = lgn$
  - Thus
    - ***Tree depth = lg n*** and
    - *Number of leaves* $2^d = n$
- As shown in the figure

Recursion Tree Example 1 (Cont...)



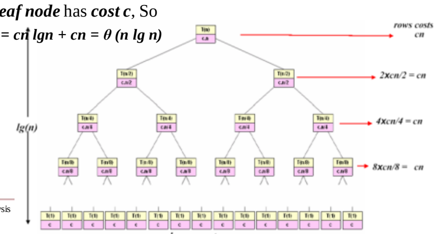
Recursion Tree Example 1 (Cont...)

- The *root* associated with ***size n*** and ***cost cn***
- Each *child of root* has ***size n/2*** and ***cost cn/2***
  - *Next level costs* are reduces by a factor **2**
- *Each leaf* has associated ***cost c***



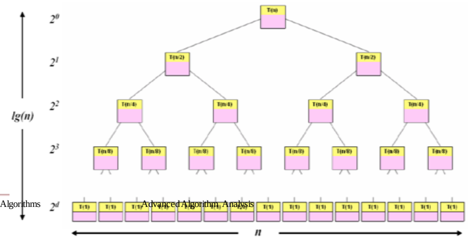
Recursion Tree Example 1 (Cont...)

- Each row has ***cost cn***
- There are ***lg (n-1) internal nodes*** and ***one root node***
- Total cost is ***cn (lg n-1) + cn = cn lgn***
- Each ***n leaf node*** has ***cost c***, So
  - $T(n) = cn\ lgn + cn = \theta(n\ lg\ n)$



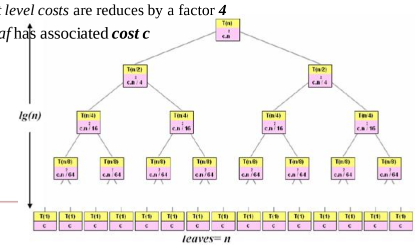
Recursion Tree Example 2

- $T(n) = 2\,T(n/2) + cn^2$, n > 1 and $T(1) = c$, n = 1
- *Step # 1* : *Constructing tree structure*
  - Fully extended tree has 2^d leaves, and ***Tree depth = lg n***



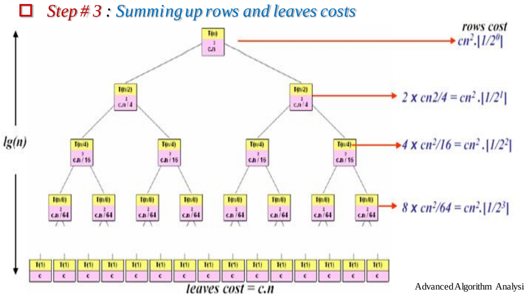
Recursion Tree Example 2 (Cont...)

- *Step # 2* : *Insertion Cost*
  - The *root* associated with ***size n*** and ***cost cn^2***
  - Each *child of root* has ***size n/2*** and ***cost $cn^2/4$***
    - *Next level costs* are reduces by a factor ***4***
  - *Each leaf* has associated ***cost c***



Recursion Tree Example 2 (Cont...)

- *Step # 3* : *Summing up rows and leaves costs*



Recursion Tree Example 2 (Cont...)

- *Step # 3* : *Summing up rows and leaves costs*
  - $T(n) = cn^2 \cdot [1/2^0 + 1/2^1 + 1/2^3 + \ldots\ldots 1/2^{\lg n-1}] + cn$
  - Asymptotic behavior of the series is determine by
    - The largest term
    - Which is ***1***
  - $1/2^0 + 1/2^1 + 1/2^3 + \ldots\ldots 1/2^{\lg n-1} = \theta(1)$.
  - Therefore
    - $T(n)=cn^2.\theta(1)+cn= \theta(n^2)$
    - n^2 is the dominant term of the sum

Recursion Tree Example 3

- ***Homework***
  - $T(n) = 3T(n/4) + cn^2$, n > 1 and $T(1) = c$, n = 1

Master Theorem

- Let $a >= 1$ and $b > 1$ be constants, then the recurrence
  - $T(n) = a\,T(n/b) + f(n)$
  - Has solutions like

1. $T(n)= \theta(n^{\log_b a})$
   - When $f(n)=O(n^{\log_b a-\varepsilon})$ for some $\varepsilon>0$
2. $T(n)= \theta(n^{\log_b a} \lg n)$
   - When $f(n)=\theta(n^{\log_b a})$
3. $T(n)= \theta(f(n))$
   - When $f(n)= \Omega(n^{\log_b a+\varepsilon})$ for some $\varepsilon>0$

  - Provide also that $af(n/b) \leq c.f(n)$ for some $c<1$ and large n
- MT provide generalized solution to ***divide-and-Conquer*** Algos

Master Theorem

- In Master Theorem
  - The cost function ***f(n)*** is *compared with the function* $n^{\log_b a}$
    - Depends on outcome
    - The *larger* of the two functions provides the solution, subject to some additional constraints
  - The constraint is that the function ***f(n)*** and $n^{\log_b a}$ should not be *simply larger or smaller asymptotically*, but
    - Should grow *faster or slower* by *polynomial factor* n^{ε}
      - Where ε is some *arbitrary small positive* constant
  - Having the following 3 cases

Master Theorem (Cont...)

- ***Case 1 :*** If $f(n)= O(n^{\log_b a-\varepsilon})$ *f(n) grows slower than*
  - $n^{\log_b a}$ *by a factor of* n^{ε}, then
  - The solution of recurrence
    - $T(n) = \theta(n^{\log_b a})$
- ***Case 2 :*** If $f(n)= \theta(n^{\log_b a})$, i. e. *f(n) grows as fast as* $n^{\log_b a}$.
  - Then, the solution of recurrence
    - $T(n) = \theta(n^{\log_b a} \lg n)$
- ***Case 3 :*** If $f(n)=\Omega(n^{\log_b a+\varepsilon})$, i. e. *f(n) grows faster than*
  - $n^{\log_b a}$ *by a factor of* n^{ε}, and $f(n/b) \le c.f(n)$ for some c < 1
  - The solution of recurrence
    - $T(n)= \theta(f(n))$

Master Theorem Examples

- *Example 1* : **T(n) = 4 T(n/2) + n**
  - Here a = 4, b = 2, f(n) = n
  - Consider $n^{\log_b a-\varepsilon} = n^{\log_2 4-\varepsilon} = n^{2-\varepsilon}$ take $\varepsilon = 0.5$
  - *f(n) = n* grows slower then $= n^{\log_b a-\varepsilon} = n^{1.5}$, it follows that $f(n)= O(n^{\log_b a-\varepsilon})$
  - Thus (Case 1)
    - $T(n)=\theta(n^{\log_b a})=\theta(n^{\log_2 4}) = \theta(n^2)$
- *Example 2* : **T(n) = T(n/2) + 1**
  - Here a = 1, b = 2, f(n) = 1
  - Consider $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$ take $\varepsilon = 1$
  - *f(n) = 1* grows as fast as $n^{\log_b a} = 1$ it follows that $f(n)= \theta(n^{\log_b a})$
  - Thus (Case 2) $T(n)=\theta(n^{\log_b a}.\lg n)=\theta(n^{\log_2 1}.\lg n) = \theta(\lg n)$

Master Theorem Examples (Cont...)

- **Example(3): *T(n)=T(n/3) +n***
  Here $a=1$, $b=3$, $f(n)=1$
  Consider $n^{\log_b a+\varepsilon} = n^{\log_3 1+\varepsilon} = n^{\varepsilon}$. *Take* $\varepsilon=0.5$
  Since $f(n) = n$ ***grows faster than*** $n^{\log_b a+\varepsilon} = n^{0.5}$, it follows that $f(n)= \Omega(n^{\log_b a+\varepsilon})$
  Further , *af(n/b)<c. f(n) if 1/(n/3)<c.n i.e, n/3 <c.n for some c.* This is true if $c=1/4$
  This is case 3 of Master Theorem. Therefore , $T(n)=\theta(f(n))=\theta(n)$

- **Example(4): *T(n)=3T(n/4) +n lg n***
  Here $a=3$, $b=4$, $f(n)=n \lg n$
  Consider $n^{\log_b a+\varepsilon} = n^{\log_3 4+\varepsilon} = n^{0.793+\varepsilon}$. Take $\varepsilon=0.207$
  Since $f(n) = n \log n$ ***grows faster than*** $n^{\log_b a+\varepsilon} = n$, it follows that $f(n)= \Omega(n^{\log_b a+\varepsilon})$
  Further , *af(n/b)<c. f(n) if 3(n/4)lg (n/4) <c.n lg n some c=3/4.*
  This is case 3 of Master Theorem. Therefore , $T(n)=\theta(f(n))=\theta(n \lg n)$

End of the Chapter

- Solve the given (uploaded document) examples
  - Using
    - Iteration Method
    - Substitution Method
    - Recursive Tree Method
    - Master Theorem
- You may have quiz next week