Introduction to Complexity Analysis of Recursive Algorithms

Session No.: 7
Course Name: Design and analysis of algorithm
Course Code: R1UC407B
Instructor Name: Mili Dhar
Duration: 50 Min.
Date of Conduction of Class:

GALGOTIAS UNIVERSITY

Review of the key concepts

1. Analyzing Non-Recursive Algorithms

Q: What happens when a problem is divided into subproblems in recursive algorithms?

Learning Outcome

Solve recurrence relations using the **Master Theorem.**

Session Outline

1 Complexity Analysis of Recursive Algorithms usin Master Theorem

2 Reflection learning activity

3 Conclusion and post-session activity

Master Theorem:

The **Master Theorem** is used to analyze the time complexity of divide-and-conquer recurrences of the form:

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

where:

- a is the number of subproblems,
- b is the factor by which the problem size is reduced in each recursive call,
- f(n) is the additional work done outside the recursive calls.

Master Theorem:

Not all recurrence relations can be solved with the use of the master theorem i.e. if

- Subproblems have uneven sizes
- f(n) is not a polynomial, ex: $T(n) = 2T(n/2) + 2^n$
- The value of $a < 1$

Master Theorem:

This theorem is an advance version of master theorem that can be used to determine running time of divide and conquer algorithms if the recurrence is of the following form:

$$T(n) = aT(n/b) + \theta(n^k \log^p n)$$

where n = size of the problem
a = number of subproblems in the recursion and a >= 1
n/b = size of each subproblem
b > 1, k >= 0 and p is a real number.

Master Theorem

It has three cases:

1. if $a > b^k$, then $T(n) = \theta(n^{\log_b a})$
2. if $a = b^k$, then
   (a) if $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
   (b) if $p = -1$, then $T(n) = \theta(n^{\log_b a} \log\log n)$
   (c) if $p < -1$, then $T(n) = \theta(n^{\log_b a})$

3. if $a < b^k$, then
   (a) if $p >= 0$, then $T(n) = \theta(n^k \log^p n)$
   (b) if $p < 0$, then $T(n) = \theta(n^k)$

Activity 1

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

Activity 1

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

Activity 1

3. $T(n) = 8T(n/2) + n^2$

Activity 1

4. T(n) = 9T(n/3) + 1

Activity 1

5. T(n) = 8T(n/2) + nlogn

Activity 2

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

Activity 2

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

Activity 2

2. $T(n) = 4T(n/2) + n^2 \log n$

Activity 2

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

Activity 2

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

Activity 2

4. T(n) = 2T(n/2) + n/logn

Activity 2

5. $T(n) = 2T(n/2) + n\log^{-2}n$

Activity 2

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

Activity 2

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

Activity 2

8. $T(n) = 2T(n/2) + n^2 \log n$

Activity 2

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

Post session activities

1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
MCHELQ

In the next session, Analysis of Recursive Algorithms using substitution method will be discussed in detail.