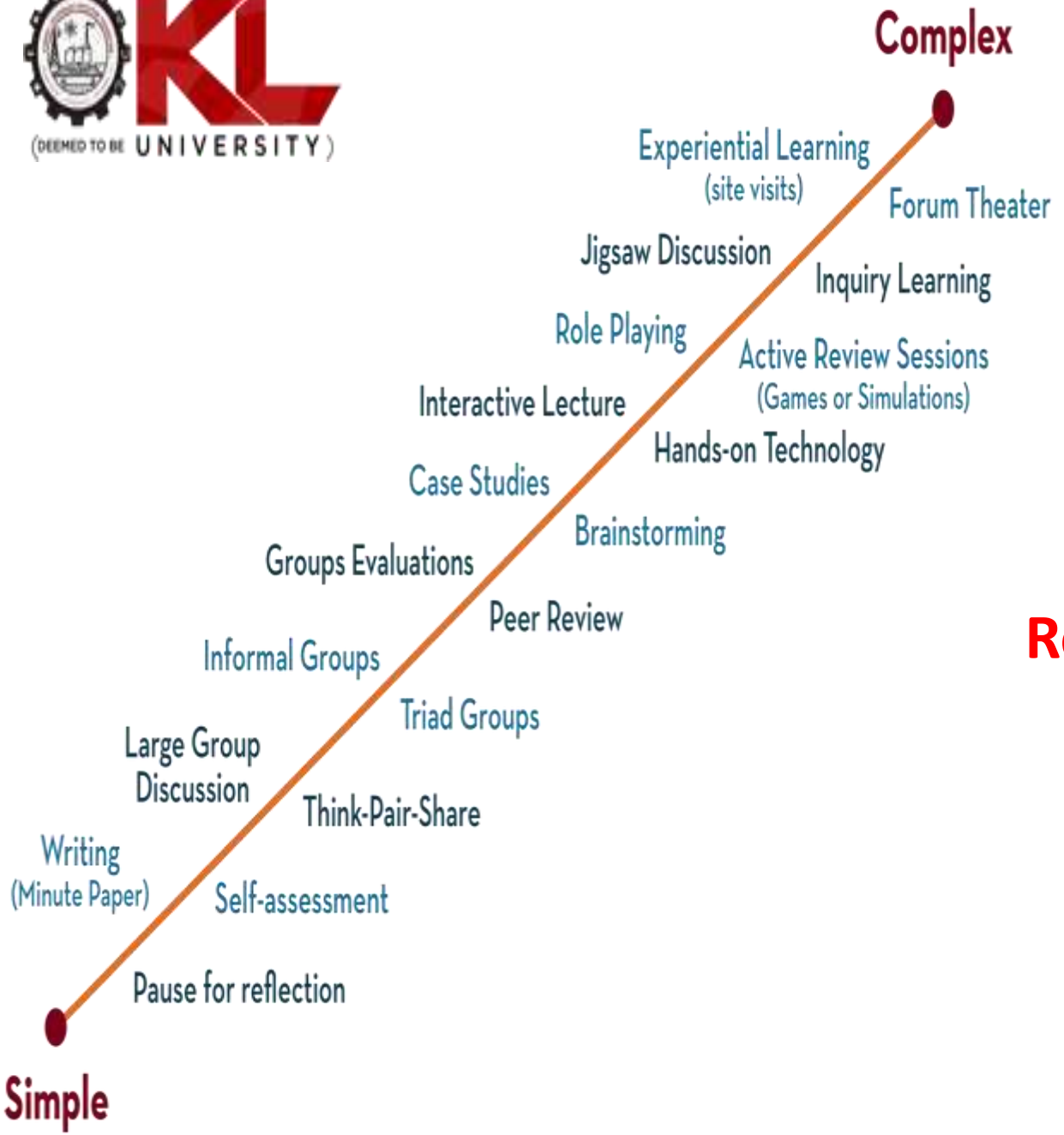

Department of CSE(H)

THEORY OF COMPUTATION
23MT2014

Topic:

Reducibility in Computational Complexity

AIM OF THE SESSION

To understand the concept of reducibility in computational complexity, including its types (many-one reducibility and Turing reducibility), and to apply these concepts in analyzing the relationships between various computational problems.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. define many-one reducibility and Turing reducibility and differentiate between the two types, providing examples for each.
2. perform many-one reductions between given computational problems, explaining the process of reduction and its significance in complexity theory.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. articulate how reducibility establishes relationships between problems, particularly in the context of NP-completeness and problem classification..
2. analyze and categorize various computational problems based on their reducibility, identifying which problems can be reduced to others and drawing conclusions about their relative complexity.

Introduction

- Reducibility is a fundamental concept in computational complexity theory. It allows us to compare the complexities of different problems and understand how the solution of one problem can be transformed into the solution of another.
- **NP-Complete Problem Reduction**
- A problem is NP-Complete if:
  1. It is in NP (nondeterministic polynomial time).
  2. Every problem in NP can be reduced to it in polynomial time.

Definition of Reducibility

- A problem A is reducible to a problem B (denoted as $A \leq B$) if there is an algorithm that transforms instances of A into instances of B in polynomial time, such that the solution of B provides a solution for A.

Reducibility: An Example

- **Co-NP and Complement Problems**
- **Definition:** A problem is in co-NP if its complement (the set of instances where the answer is "no") is in NP.
- **Example: Non-Primality**
- **Problem A**: Primality Test (in NP)
- **Problem B**: Non-Primality Test (in co-NP)
- **Reduction Process**
  - To show that non-primality can be reduced to primality:
  - Use the primality test as a subroutine.
  - If a number is found to be prime using the primality test, the answer for non-primality is "no."
  - If it’s not prime, the answer is "yes."
- **Conclusion**: This reduction shows the relationship between NP and co-NP.

Types of Reducibility

- **Many-One Reducibility**: A problem A is many-one reducible to problem B $(A \leq_m B)$ if there exists a function f that transforms instances of A to instances of B such that:
- - f is computable in polynomial time
- - $x \in A$ if and only if $f(x) \in B$

- **Turing Reducibility:** A problem A is Turing reducible to problem B $(A \leq_T B)$ if A can be solved using a Turing machine that has access to an oracle for B.

Example of Many-One Reducibility

- Consider the problems:
- **SAT (Satisfiability Problem)**: Determine if a boolean formula is satisfiable.
- **3-SAT**: A specific case of SAT where each clause has exactly three literals.

- We can reduce SAT to 3-SAT by transforming any boolean formula into an equivalent 3-CNF formula in polynomial time.

Example of Many-One Reducibility

Reduction Process: 3-SAT to SAT

1. **Instance of 3-SAT**:
   1. Suppose we have a 3-SAT instance represented as a Boolean formula:

$$C_1 \wedge C_2 \wedge C_3$$

   where each clause C_i has exactly three literals. For example:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee x_5)$$

   Function f:
   1. The function f transforms the 3-SAT formula into a SAT formula.
   2. In this case, the function f can be the identity function, where we simply output the same formula as the SAT instance.
2. For instance, the formula:

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee x_5)$$

   can be used directly as an input for SAT since it is already in CNF.

Correctness of the Reduction:

1. If the original 3-SAT formula is satisfiable (i.e., there exists an assignment of truth values that makes the formula true), then the transformed SAT formula is also satisfiable.
2. Conversely, if the transformed SAT formula is satisfiable, then there exists an assignment that also satisfies the original 3-SAT formula.

- **Implication:** Since we can transform any instance of 3-SAT into an instance of SAT in polynomial time (using the identity function in this case), we conclude: $3 - SAT \leq_m SAT$

Example of Turing Reducibility

- Let A be the problem of deciding if a given Turing machine halts on an input, and let B be the problem of deciding if a number is prime.

- A can be solved using a Turing machine that has access to an oracle for B, allowing it to check the primality of numbers involved in its computations.

Importance of Reducibility

- **Classification of Problems**: Helps in classifying problems into complexity classes.
- **Solving Hard Problems:** If a known hard problem can be reduced to a new problem, the new problem is at least as hard as the known problem.
- **Understanding Relationships**: Provides insight into the relationships between different problems.

THANK YOU

Team – TOC