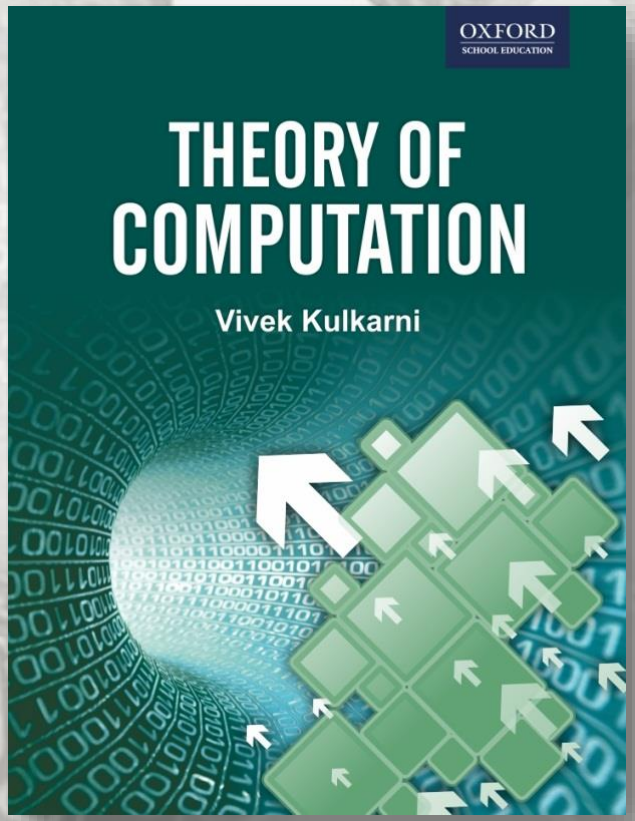
OXFORD
HIGHER EDUCATION

THEORY OF COMPUTATION

Vivek Kulkarni

Slides for Faculty Assistance



Chapter 10

Complexity and Classification of Problems

Author: Vivek Kulkarni

vivek_Kulkarni@yahoo.com

Outline

- Following topics are covered in the slides:
  - The complexity of a problem
  - Mathematical notations for complexity measure
  - Time and space complexity
  - Non-deterministic algorithm
  - Problem classification: P-type vs NP-type
  - Satisfiability problem
  - NP-hard and NP-complete problems
  - Cook's theorem

Complexity of a Problem

- The complexity of a problem (or a program) is measured in terms of the computing time required and/or the storage (or space) requirement of the problem. The former is called the **time complexity** of the problem and the later is termed as the **space complexity**.

- Two parameters are required for computing the time complexity of a problem (or a program): the frequency count of each operation (or statement), i.e., the number of times each operation is performed (or statement is executed), and the time taken for one computation. The total time required to solve the problem is the product of these two parameters.

Mathematical Notations for Time Complexity

- **Big-O Notation**
  - Big-O notation is used to express an upper bound of the computing time.
- **Little-o Notation**
  - The little-o notation is defined the same way as the big-O notation, except that the computing time of a problem is less than what is expressed.
- **Omega (Ω) Notation**
  - omega (Ω) notation is used to express a lower bound of the computing time.
- **Theta (θ) Notation**
  - It represents both lower and upper bounds on the computing time.

Time and Space Complexity of a TM

- the time complexity of a TM T is denoted as:

  $\tau_T(n)$ = maximum number of moves T can make on any input string of length n

  If for a given input string, T loops forever, then the time complexity is undefined.

- A space complexity of a TM T is denoted as:

  $S_T(n)$ = maximum number of tape cells used by T while working with any input string of length n

  If on some input, T loops forever and causes an infinite number of tape cells to be used, then $S_T(n)$ is undefined.

Classification of Problems

- Many of the problems that we know are broadly classified into two main categories: polynomial time (**P**) problems and non-polynomial time (**NP**) problems. Actually, NP is the term used for 'non-deterministically polynomial'.

- A problem, whose solution can be given by a polynomial time algorithm, which means that the function $f(n)$ representing the computing time is a polynomial, then the problem is called a ***P-type* problem**.
  - Computing time is a polynomial of a small degree. For example, computing time of a problem is of the order $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, and so on.
  - Computing time is a polynomial of a greater degree. Computing time for these types of problems is $O(n^2 2^n)$, $O(2^{n/2})$, and so on.

Non-deterministic Algorithm

- For a *deterministic* solution, the result of every operation is uniquely defined and predictable.
- If we allow an algorithm to contain operations, whose result is not uniquely determined but is limited to a finite set of possibilities, then such an algorithm is said to be *non-deterministic*.
- A non-deterministic algorithm for searching for an element x from a given set of elements, can be written as follows:

```
i = choose (n);
if (A[i] = = x)
        printf ('%d', i);   // success
printf('0') ;              // failure
```

- Here, 'choose (n)' is a procedure assumed, which randomly chooses one element of the given n elements. A zero is printed if x does not belong to the set of elements.
- The above algorithm is of non-deterministic complexity, O(1). This is because, there is only one way to achieve success, and the algorithm is possibilistic. The complexity of a deterministic algorithm for the same problem is $\Omega(n)$.

Satisfiability Problem

- A *formula* in propositional calculus is an expression that can be constructed using variables, their negations, and the operators, $\wedge$ (and) and $\vee$ (or). For example, $(x_1 \vee x_2)$.
- A formula is said to be in *conjunctive normal form* (CNF), if it can be represented as: $e_1 \wedge e_2 \wedge \ldots \wedge e_n$, where $e_1, e_2, \ldots, e_n$ are sub-formulae.
- A formula is said to be in *disjunctive normal form* (DNF), if it can be represented as: $e_1 \vee e_2 \vee \ldots\ldots \vee e_n$, where $e_1, e_2, \ldots, e_n$ are sub-formulae.

- **Satisfiability Problem**: The *satisfiability problem* (often written in all capitals, or abbreviated as SAT) is to determine if a given propositional formula is true for some assignment of truth values to the variables in the formula.
  - The worst-case complexity of a deterministic algorithm to find the satisfiability of a given formula of n variables is $O(2^n)$.

NP-hard and NP-complete Problems

- *P* is the set of all decision problems solvable by a deterministic algorithm in polynomial time, and *NP* is the set of all decision problems solvable by a non-deterministic algorithm in polynomial time.
- P-type problems are also known as efficiently solvable or *tractable* problems, while NP-type problems are also known as *intractable* problems.
- Let A_1 and A_2 be problems such that A_1 *reduces* to A_2 (written as $A_1 \propto A_2$) if and only if it is possible to solve A_1 using a deterministic polynomial time algorithm that uses a deterministic algorithm to solve A_2 in polynomial time.
- A problem *A* is said to be *NP-hard* if and only if the satisfiability problem reduces to *A*.
- A problem *A* is said to be *NP-complete* if and only if *A* is NP-hard and $A \in$ [NP].
- The halting problem is an NP-hard problem.

Cook's Theorem

- **Statement of Cook's theorem:**
  The Boolean satisfiability problem is NP-complete.

- The deterministic algorithm for the satisfiability problem requires exponential time. The problem is very important in complexity theory and is considered as the boundary between P-type and NP-type problems.

- It is easy to obtain a polynomial time non-deterministic algorithm that terminates successfully if and only if a given propositional formula of n variables is satisfiable. Such an algorithm, thus, chooses non-deterministically one of the 2^n possible assignments of truth values to n variables and can verify that the formula is true for that assignment. The time required by the non-deterministic algorithm to choose the value of n variables and the time required to deterministically evaluate the formula for that set of values is $O(n)$. This time is thus proportional to the length of the formula.

- An important consequence of the theorem is that if there is a deterministic polynomial time algorithm for solving Boolean satisfiability, then there exists a deterministic polynomial time algorithm for solving all problems in [NP].