Design and Analysis of Algorithms

Nirdosh Bhatnagar

Introduction

1. What is an Algorithm?

Definition: An algorithm is a well defined computational procedure that converts input into output.

$$\text{INPUT} \dashrightarrow \text{ ALGORITHM } \dashrightarrow \text{OUTPUT}$$

□

Note that we have not defined precisely computational procedure.

Example:
Input: A sequence of n real numbers $(a_1, a_2, \ldots, a_n)$.
Output: A reordering of the input sequence $(a'_1, a'_2, \ldots, a'_n)$, where $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.
For instance, given the input sequence, $(31, 41, 59, 26, 41, 58)$, a sorting algorithm outputs $(26, 31, 41, 41, 58, 59)$. The input sequence is called an instance of the problem. □

2. What is Analysis of Algorithms?

The analysis of algorithms is the theoretical study of computer-program performance and resource usage. Before, an algorithm is implemented its performance requirements determines its feasibility or infeasibility. That is, performance requirements determine, if the algorithm-implementation is possible or impossible.

Why Study Algorithms?

Algorithms address issues related to: feasibility, efficiency and performance, and scalability.

- Study of algorithms enable us to determine, if a computer program is *feasible* or *infeasible.*
- *Efficient* algorithms lead to efficient computer programs, and efficient use of hardware resources.
- Algorithms help us to understand issues related to *scalability.*
- Analysis of algorithms provides a *language* for talking about program behavior.

However, we should understand that computer program efficiency is only a certain facet of overall computer resource usage.

3. Efficient Computer Programs

Several factors influence program efficiency. These are:

- Problem being addressed or solved.
- Programming language.
- Compiler
- Computer hardware
- Programmer ability and effectiveness
- Algorithm

Other Important Issues.

In addition to the analysis of an algorithm, there are equally more important requirements of a computer program. In no particular order, these are:

- Correctness of implementation
- Extensibility
- Functionality
- Maintainability
- Modularity
- Programmer-time
- Reliability
- Robustness
- Simplicity
- User-friendliness