CS-2001
DATA STRUCTURES
FALL2021

Dr. Hashim Yasin

Mr. Muhammad Husnain

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

Algorithm

- An algorithm is a definite procedure for solving a problem in a finite number of steps.
- Algorithm is a well-defined computational procedure that takes some value (s) as input and produces some value (s) as output.
- Algorithm is finite number of computational statements that transform input into the output

Algorithm

- Finite sequence of instructions.
- Each instruction having a clear meaning.
- Each instruction requires finite amount of effort.
- Each instruction requires finite time to complete.

Evaluation of Algorithm

- **Completeness:** is the strategy guaranteed to find a solution when there is one?
- **Optimality:** does the strategy find the highest-quality (least-cost) solution when there are several different solutions?
- **Time complexity:** how long does it take to find a solution?
- **Space complexity:** how much memory is needed to perform the search?

Analysis of an Algorithm

- What is the goal of analysis of algorithms?
  - To compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirements, programmer's effort etc.)

- What do we mean by running time analysis?
  - **Determine how running time increases as the size of the problem increases.**

Analysis of an Algorithm

- Ways of measuring efficiency:
  - Run the program and see how long it takes
  - Run the program and see how much memory it uses

- Lots of variables to control:
  - What is the input data?
  - What is the hardware platform?
  - What is the programming language/compiler?

Types of Analysis

- **Worst case**
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are.
- **Best case**
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

Types of Analysis

- **Average case**
  - Provides a prediction about the running time
  - Assumes that the input is random.

$$\boxed{Lower\ Bound \le Running\ Time \le Upper\ Bound}$$

Asymptotic Notation

- Asymptotic notations – Asymptotic notations are the notations used to describe the behavior of the time or space complexity.
- Let us represent the time complexity and the space complexity using the common function $f(n)$.
  - O (*Big Oh notation*)
  - *Ω* (*Omega notation*)
  - θ (*Theta notation*)
  - o (*Little Oh notation*)

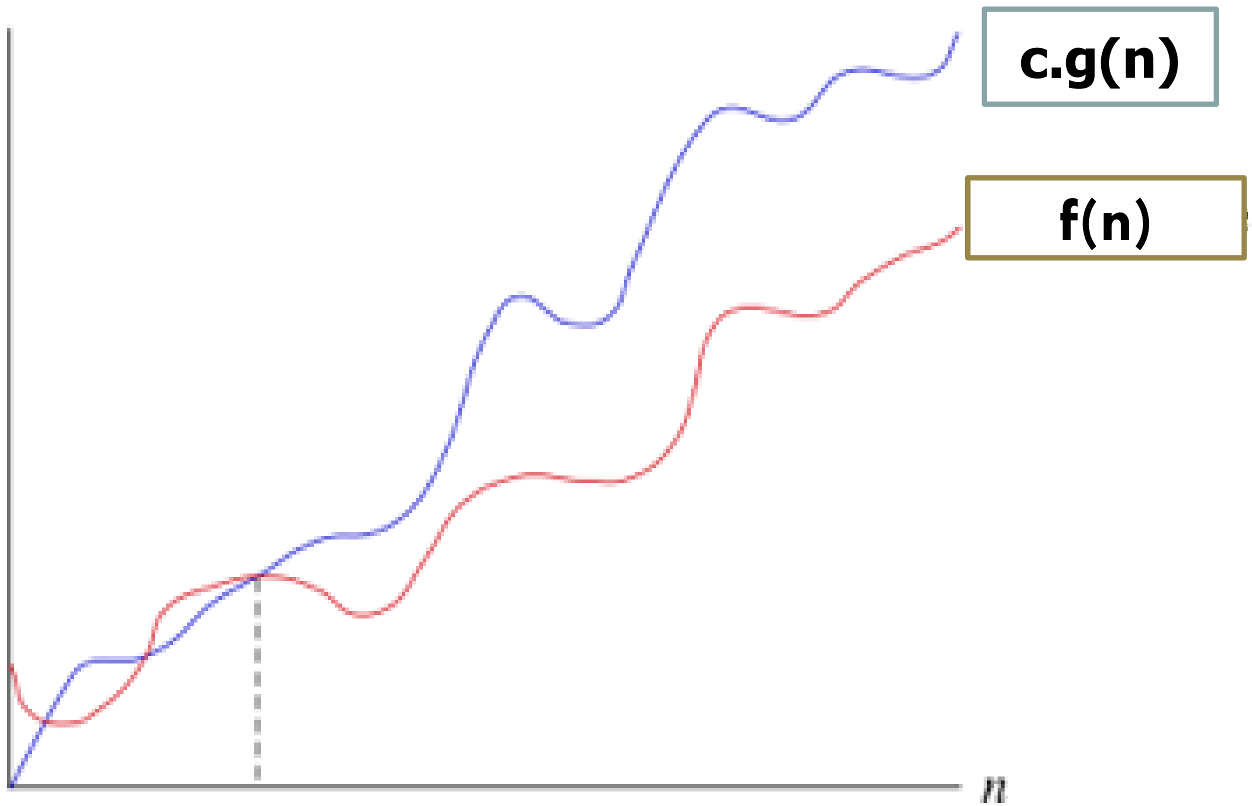
O – Big Oh notation

- The big Oh notation provide s an **upper bound** for the function $f(n)$.

- The function $f(n) = O(g(n))$ if and only if there exists positive constants c and n_0 such that

$$f(n) \leq cg(n) \text{ for all } n \geq n_0$$

O – Big Oh notation



O – Big Oh notation

$f(n) = 3n + 2$

$f(n) \leq cg(n)$ for all $n \geq n_0$

Let us take $g(n) = n$

$c = 4$

$n_0 = 2$

Let us check the above condition

$3n + 2 \leq 4n$ for all $n \geq 2$

The condition is satisfied. Hence $f(n) = O(n)$.

O – Big Oh notation

$f(n) = 10n^2 + 4n + 2$

$f(n) \le cg(n)$ for all $n \ge n_0$

Let us take $g(n) = n^2$

$c = 11$

$n_0 = 6$

Let us check the above condition

$10n^2 + 4n + 2 \le 11n$ for all $n \ge 6$

The condition is satisfied. Hence $f(n) = O(n^2)$.

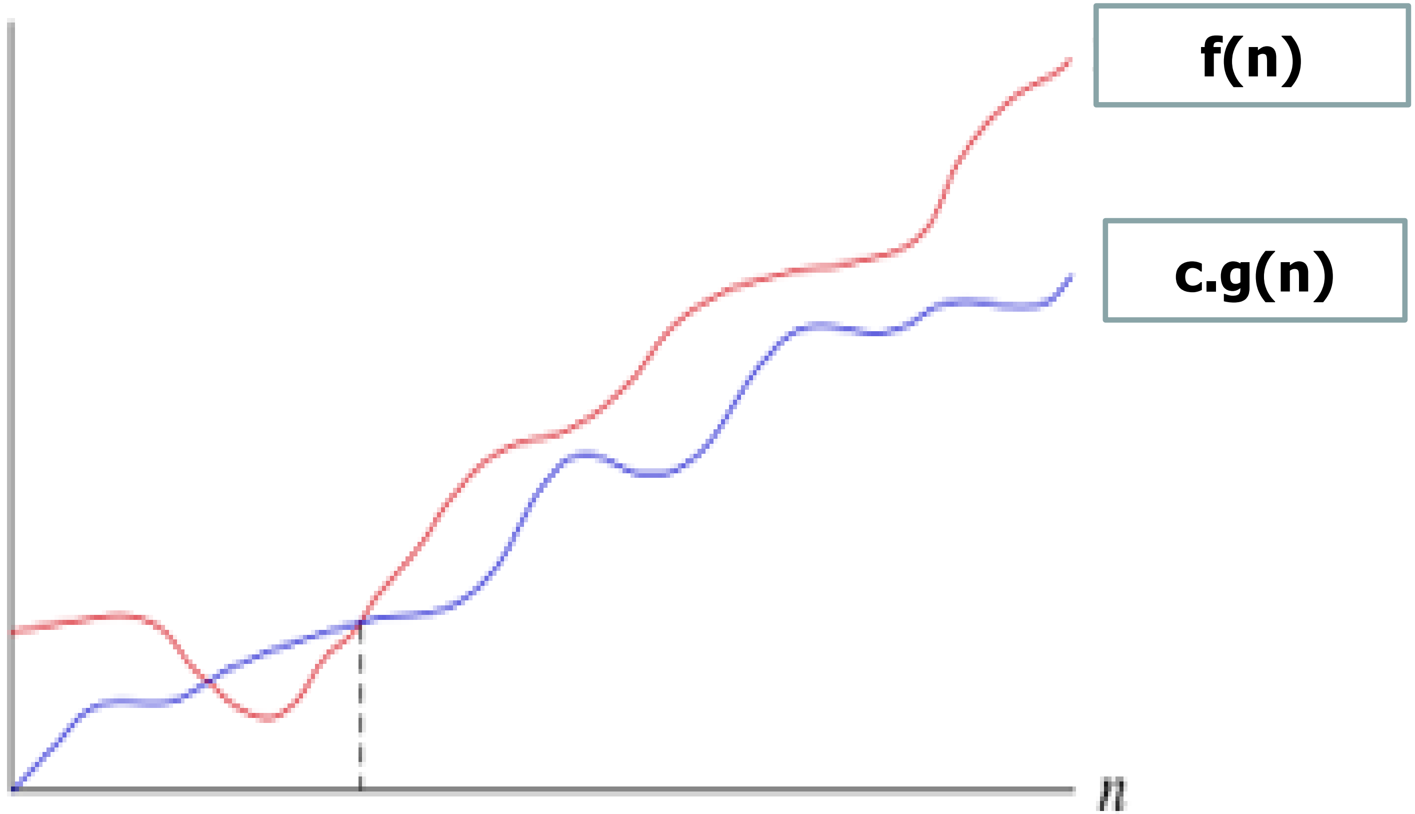
Ω - Omega notation

- The omega notation (Ω) provides a **lower bound** for the function $f(n)$.

- The function $f(n) = \Omega(g(n))$ if and only if there exists positive constants c and n_0 such that

$$f(n) \geq cg(n) \text{ for all } n \geq n_0$$

Ω - Omega notation



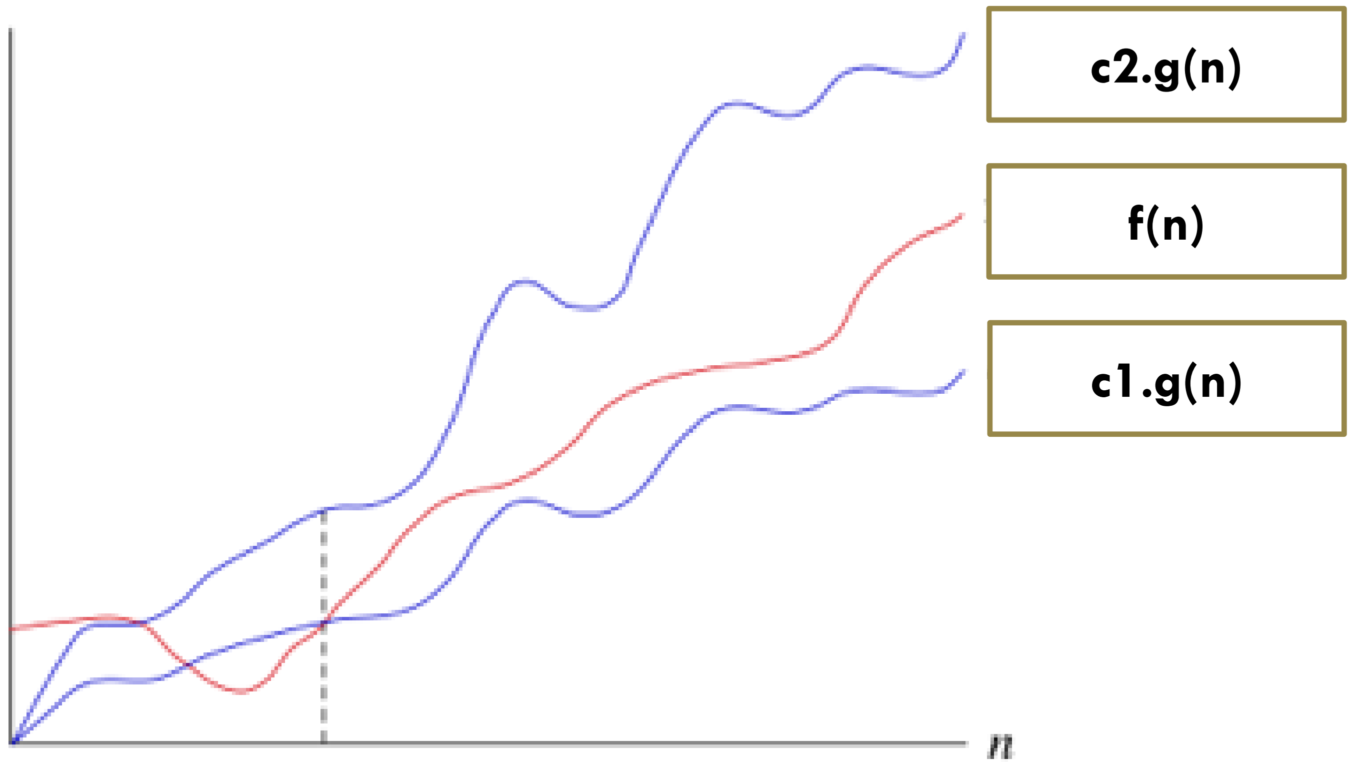
θ – Theta notation (Tight Bound)

- The theta notation (θ) is used when the function $f(n)$ can be bounded by both from above and below the same function $g(n)$.

- The function $f(n) = \theta(g(n))$ if and only if there exists positive constants c_1, c_2 and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0$$

θ – Theta notation (Tight Bound)



o - Little Oh notation

- The little o notation (o) is,

$$f(n) = o(g(n))$$ if and only if

$$f(n) = O(g(n))$$ and

$$f(n) \neq \Omega(g(n))$$

(OR)

- *(f(n) / g(n)) becomes zero as 'n' approaches to infinity*
  - *i.e., f(n) is strictly less than g(n) in order of growth*

Reading Materials

- Mark Allen Weiss – Chapter#2
- Nell Dale: Chapter # 3 (Section 3.4)
- D. S. Malik – Chapter#1