Analysis of Algorithms

Lecture No. 4 & 5

Complexity and asymptotic Notations

Dr. Shamila Nasreen

What is Complexity?

- The level in difficulty in solving mathematically posed problems as measured by
  - The time
    (time complexity)
  - number of steps or arithmetic operations
    (computational complexity)
  - memory space required
    (space complexity)

Agenda

- What is complexity
- Asymptotic analysis
- Types of Analysis
- Big O Notation (Worst Case Analysis)
- Big Omega (Best Case analysis)
- Examples

Complexity of Algorithms

- What is Complexity of Algorithms

- Storage Requirement
- Execution Time
- Implementation Details
- Hardware Requirement etc

To measure/compute all these resources is basically the complexity of Algorithms

Why ???

- Why we are measuring the complexity of Algorithms ???.

- Reason : Not only to design algorithms but to design efficient type of algorithms which can finish in finite amount of time.

What is Asymptotic Analysis ???

Asymptotic analysis describes the behavior of an algorithm as the input size grows towards infinity. It focuses on:

- Growth rate of the runtime
- Ignoring constants and lower-order terms

Analogy: Think of driving: Speed limit signs (asymptotic notation) tell you how fast you can expect to go under certain road conditions, not the exact speed every time.

Why asymptotic Notations

- Objective: To measure efficiency these notations are used.
- Design efficient algorithm terminate in finite amount of Time/ acceptable amount of time.

Making use of Asymptotic notations for measuring running time of algorithm for large input size and measure its growth rate .

Input Size

Input size (number of elements in the input)

- size of an array
- polynomial degree
- # of elements in a matrix
- # of bits in the binary representation of the input
- vertices and edges in a graph

Efficiency Measurement

- Efficiency is relative term
- Single algorithm : Polynomial Time Period (Efficient)

An algorithm is said to run in **polynomial time** if its **execution time** is upper bounded by a **polynomial expression** in the size of the input.

That is: $T(n)=O(n^k)$

- Comparing Algorithms for finding their efficiencies

Types of Analysis

- BIG O
- Omega *Ω*
- Theta *Θ*

For Large inputs these analysis are performed.

Complexity Analysis

- Algorithm analysis means predicting resources such as
  - computational time
  - memory
  - computer hardware etc
- Worst case analysis
  - Provides an upper bound on running time
  - An absolute guarantee
- Average case analysis
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?
    - Random (equally likely) inputs
    - Real-life inputs

Types of Analysis

- **Worst case (at most BIG O)**
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are

Let us suppose that

- D_n = set of inputs of size n for the problem
- I = an element of D_n.
- t(I) = number of basic operations performed on I
- Define a function W by

  $$W(n) = \max\{t(I) \mid I \in D_n\}$$

  called the worst-case complexity of the algorithm
- W(n) is the maximum number of basic operations performed by the algorithm on any input of size n.

Types of Analysis

- **Average case (Theta *Θ*)**
  - Provides a prediction about the running time
  - Assumes that the input is random
- Average cost =

  A(n) = Pr(succ)/succ(n) + Pr(fail)/fail(n)
- Worst Analysis computing average cost

  Take all possible inputs, compute their cost, take average

Types of Analysis

- Best case (at least Omega *Ω)*
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

How do we compare algorithms?

- We need to define a number of objective measures.

(1) Compare execution times?

Not good: times are specific to a particular computer !!

(2) Count the number of statements executed?

Not good: number of statements vary with the programming language as well as the style of the individual programmer.

Ideal Solution

- Express running time as a growth function of the input size *n* (i.e., *f(n)*).
- Compare different functions corresponding to running times.
- Such an analysis is independent of machine time, programming style, etc.

Asympototic Notations Properties

- Categorize algorithms based on asymptotic growth rate e.g. **linear, quadratic, polynomial, exponential(in next lecture)**
- Ignore small constant and small inputs
- Estimate upper bound and lower bound on growth rate of time complexity function
- Describe running time of algorithm as n grows to ∞.
- Describes behavior of function within the limit.

Limitations

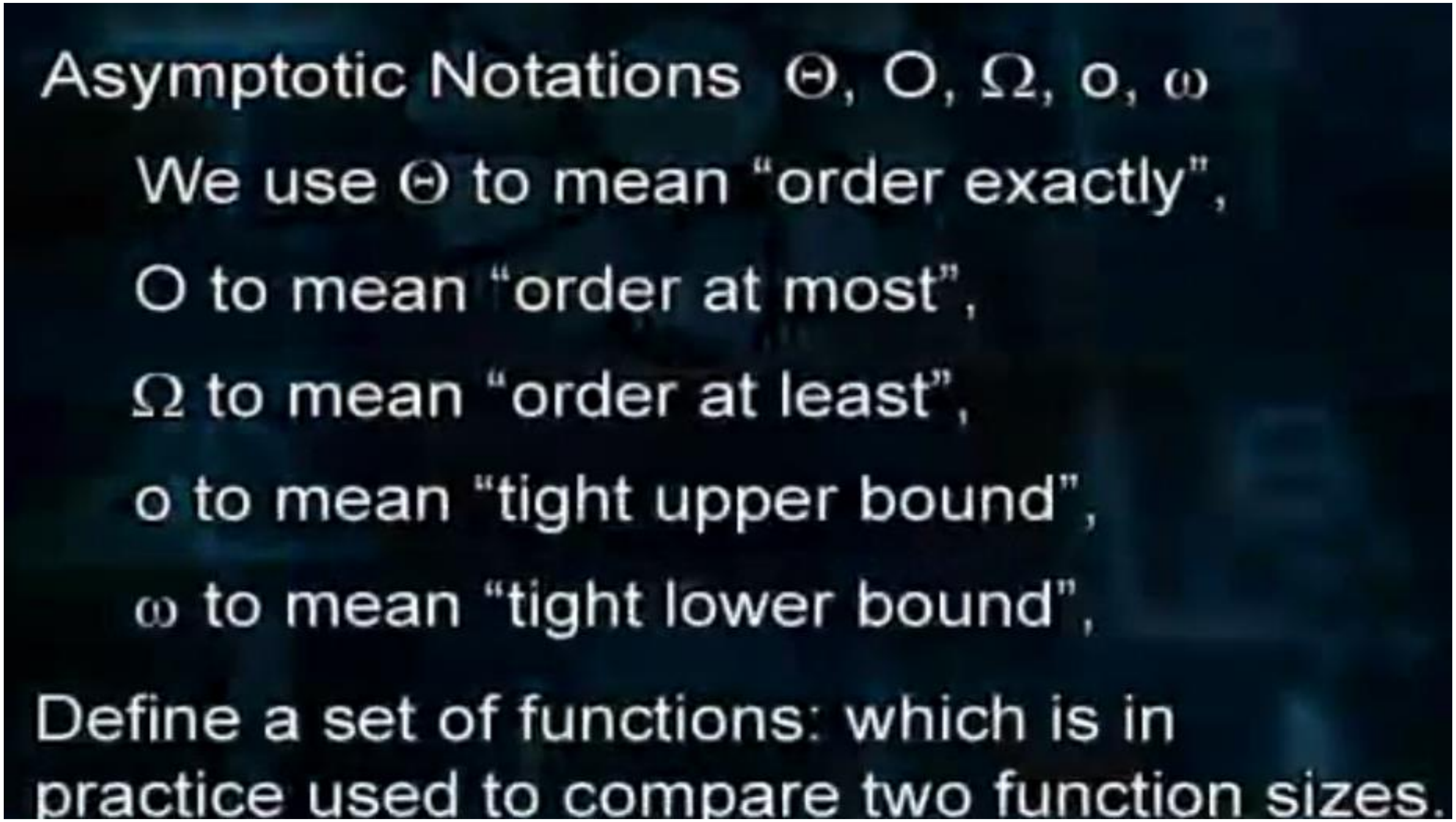
- not always useful for analysis on fixed-size inputs.
- All results are for *sufficiently large* inputs

Asymptotic Analysis

- To compare two algorithms with running times *f(n)* and *g(n),* we need a **rough measure** that characterizes **how fast each function grows.**
- *Hint:* use *rate of growth*
- Compare functions in the limit, that is, **asymptotically!**

  (i.e., for large values of *n*)

Asymptotic Notations



Asymptotic Notations Θ, O, Ω, o, ω

We use Θ to mean "order exactly",

O to mean "order at most",

Ω to mean "order at least",

o to mean "tight upper bound",

ω to mean "tight lower bound",

Define a set of functions: which is in practice used to compare two function sizes.

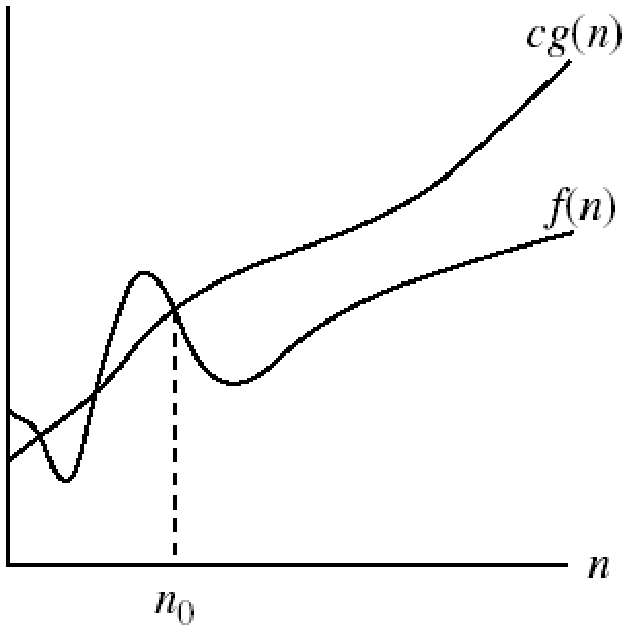
Asymptotic Notation

- O notation :Big-O is the formal method of expressing the upper bound of an algorithm's running time.
- It's a measure of the longest amount of time it could possibly take for the algorithm to complete.
- Formally, for non-negative functions, *f(n)* and *g(n)*, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$, $f(n) \leq cg(n)$, then *f(n)* is Big O of *g(n).*

Asymptotic notations

- *O-notation*

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}\ .$$



$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

Asymptotic notations

- Examples

Example 1: Prove that $2n^2 \in O(n^3)$

Proof:

Assume that $f(n) = 2n^2$, and $g(n) = n^3$

$f(n) \in O(g(n))$?

Now we have to find the existence of c and n_0

$f(n) \leq c.g(n)$ ⍰ $2n^2 \leq c.n^3$ ⍰ $2 \leq c.n$

if we take, $c = 1$ and $n_0 = 2$ OR

$c = 2$ and $n_0 = 1$ then

$2n^2 \leq c.n^3$

Hence $f(n) \in O(g(n))$, $c = 1$ and $n_0 = 2$

Asymptotic notations

Example 2: Prove that $n^2 \in O(n^2)$

Proof:

Assume that $f(n) = n^2$, and $g(n) = n^2$

Now we have to show that $f(n) \in O(g(n))$

Since

$f(n) \leq c.g(n)$ ⍰ $n^2 \leq c.n^2$ ⍰ $1 \leq c$, take, $c = 1$, $n_0 = 1$

Then

$n^2 \leq c.n^2$ for $c = 1$ and $n \geq 1$

Hence, $2n^2 \in O(n^2)$, where $c = 1$ and $n_0 = 1$

Asymptotic notation

Example 3: Prove that $n^3 \in O(n^2)$

Proof:

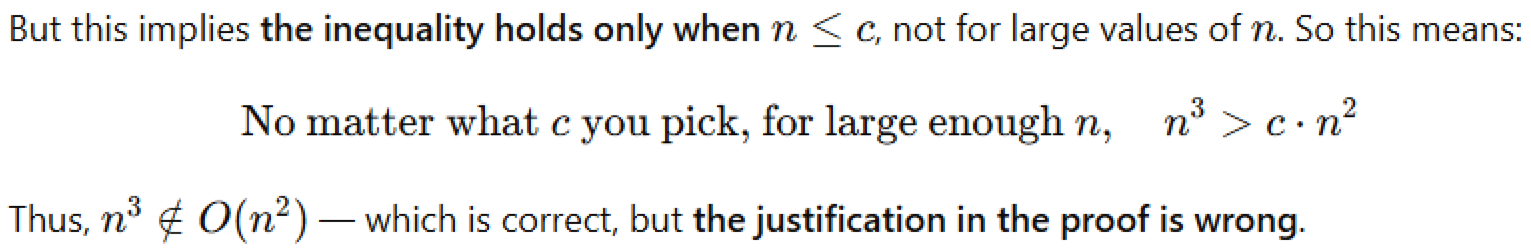
On contrary we assume that there exist some positive constants c and n_0 such that

$0 \le n^3 \le c.n^2$ where $n \ge n_0$

$0 \le n^3 \le c.n^2 \rightarrow n \le c$

Since c is any fixed number and n is any arbitrary constant, therefore $n \le c$ is not possible in general.

Hence our supposition is wrong and $n^3 \le c.n^2$, for $n \ge n_0$ is not true for any combination of c and n_0.



But this implies **the inequality holds only when $n \le c$**, not for large values of n. So this means:

$$\text{No matter what } c \text{ you pick, for large enough } n, \quad n^3 > c \cdot n^2$$

Thus, $n^3 \notin O(n^2)$ — which is correct, but **the justification in the proof is wrong**.

Asymptotic notation

Example 4: $5n+20 \in O(n)$

Example 5: Prove that $6n^4+3n^3+n^2 \in O(n^4)$

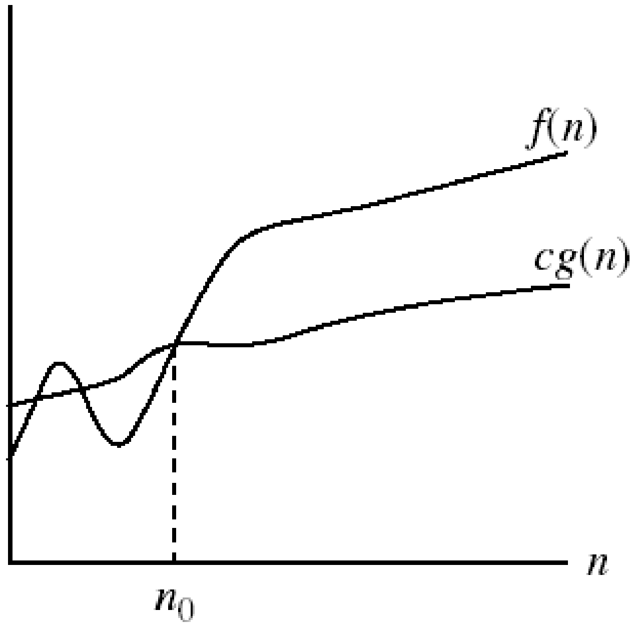
Asymptotic Notation

- ***Big-Omega Notation*** Ω
- This is almost the same definition as Big Oh, except that "$f(n) \geq cg(n)$"
- This makes $g(n)$ a lower bound function, instead of an upper bound function.
- It describes the **best that can happen** for a given data size.

- For non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$, $f(n) \geq cg(n)$, then $f(n)$ is omega of $g(n)$. This is denoted as "$f(n) = \Omega(g(n))$".

Asymptotic notations (cont.)

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that
$0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.



Ω(*g*(n)) is the set of functions with larger or same order of growth as *g*(n)

$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Asymptotic Notation

Example 1: Prove that $5.n^2 \in \Omega(n)$

Proof:

Assume that $f(n) = 5.n^2$, and $g(n) = n$

$f(n) \in \Omega(g(n))$?

We have to find the existence of c and n_0 s.t.

$c.g(n) \leq f(n)$ where $n \geq n_0$

$c.n \leq 5.n^2 \rightarrow c \leq 5.n$

if we take, $c = 5$ and $n_0 = 1$ then

$c.n \leq 5.n^2$ for $n \geq n_0$

And hence $f(n) \in \Omega(g(n))$, for $c = 5$ and $n_0 = 1$

Asymptotic Notation

Example 2: Prove that $5.n + 10 \in \Omega(n)$

Proof:

Assume that $f(n) = 5.n + 10$, and $g(n) = n$

$f(n) \in \Omega(g(n))$?

We have to find the existence of c and n_0 s.t.

$c.g(n) \leq f(n)$ for $n \geq n_0$

$c.n \leq 5.n + 10 \rightarrow c.n \leq 5.n$

if we take, $c = 5$ and $n_0 = 1$ then

$c.n \leq 5.n + 10$ for $n \geq n_0$

And hence $f(n) \in \Omega(g(n))$, for $c = 5$ and $n_0 = 1$

Asymptotic Notation

Example 3: Prove that $100.n + 5 \notin \Omega(n^2)$

Proof:

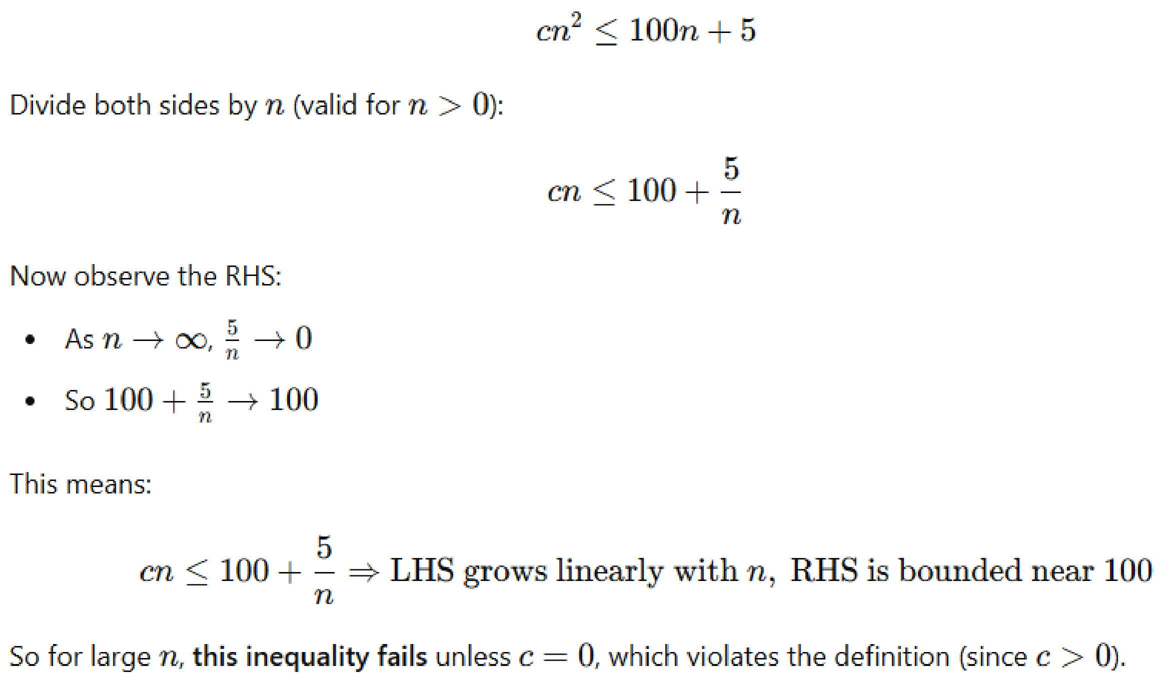
Let $f(n) = 100.n + 5$, and $g(n) = n^2$

Assume that $f(n) \in \Omega(g(n))$?

Now if $f(n) \in \Omega(g(n))$ then there exist c and n_0 s.t.

$c.g(n) \leq f(n)$ for $n \geq n_0$

$c.n^2 \leq 100.n + 5$



$$cn^2 \leq 100n + 5$$

Divide both sides by n (valid for $n > 0$):

$$cn \leq 100 + \frac{5}{n}$$

Now observe the RHS:

- As $n \to \infty$, $\frac{5}{n} \to 0$
- So $100 + \frac{5}{n} \to 100$

This means:

$$cn \leq 100 + \frac{5}{n} \Rightarrow \text{LHS grows linearly with } n, \text{ RHS is bounded near 100}$$

So for large n, **this inequality fails** unless $c = 0$, which violates the definition (since $c > 0$).

Summery

- Asymptotic Notation