Course Name-Design and Analysis of Algorithms Course
Code-COM-301
Lecture No-4

Topic – Use of order notations and related results
Date- 06-09-2023

MIET

Model Institute of Engineering & Technology

UNIT 1-Delivery Plan

Course Outcomes	Topics	Blooms Taxonomy
CO1	Introduction to Algorithms : Mathematical preliminaries	Understanding, Remembering
CO2	Time complexity and Space complexity	Understanding, Remembering, Applying
CO2	Worst-case and Average-case analysis	Understanding, Remembering, Applying
CO2	Use of order notations and related results	Understanding, Remembering, Applying
CO3	Divide and conquer recurrences	Understanding, Remembering
CO2	Recurrence relations: substitution method	Understanding, Remembering, Applying
CO2	Recurrence trees method, Master’s theorem and its applications	Understanding, Remembering, Applying

Lecture : Learning Outcomes

Sr. No.	At the end of the Lecture, the student shall be able to :
1	Understand the Asymptotic Analysis of algorithms (Growth of function)
2	Why is Asymptotic Notations are Important?
3	Solve examples based on Asymptotic Notations
4	Elaborate the Complexity chart for understanding notations

Asymptotic Analysis of algorithms (Growth of function)

- The Resources for an algorithm are usually expressed as a function regarding input. Often this function is messy and complicated to work.
- To study Function growth efficiently, we reduce the function down to the important part.

  Let f (n) = an^2+bn+c

- In this function, the n^2 term dominates the function that is when n gets sufficiently large.
- Dominate terms are what we are interested in reducing a function, in this; we ignore all constants and coefficient and look at the highest order term concerning n.

Asymptotic Analysis of algorithms (Growth of function)

- The word Asymptotic means approaching a value or curve arbitrarily closely (i.e., as some sort of limit is taken).

Asymptotic analysis

- It is a technique of representing limiting behavior. The methodology has the applications across science. It can be used to analyze the performance of an algorithm for some large data set.
- In computer science in the analysis of algorithms, considering the performance of algorithms when applied to very large input datasets

Asymptotic Analysis of algorithms (Growth of function)

- The simplest example is a function $f(n) = n^2+3n$, the term 3n becomes insignificant compared to n^2 when n is very large. The function "$f(n)$ is said to be **asymptotically equivalent** to n^2 as $n \to \infty$", and here is written symbolically as $f(n) \sim n^2$.
- **Asymptotic notations** are used to write *fastest* and *slowest* possible running time for an algorithm. These are also referred to as **'best case'** and **'worst case'** scenarios respectively.
- In asymptotic notations, we derive the complexity concerning the **size of the input**. (Example in terms of n).
- These notations are important because without expanding the cost of running the algorithm, we can estimate the complexity of the algorithms.

Why is Asymptotic Notations are Important?

They give simple characteristics of an algorithm's efficiency.

They allow the comparisons of the performances of various algorithms.

Asymptotic Notations

- The Asymptotic Notation is a way of comparing function that ignores constant factors and small input sizes. Three notations are used to calculate the running time complexity of an algorithm:

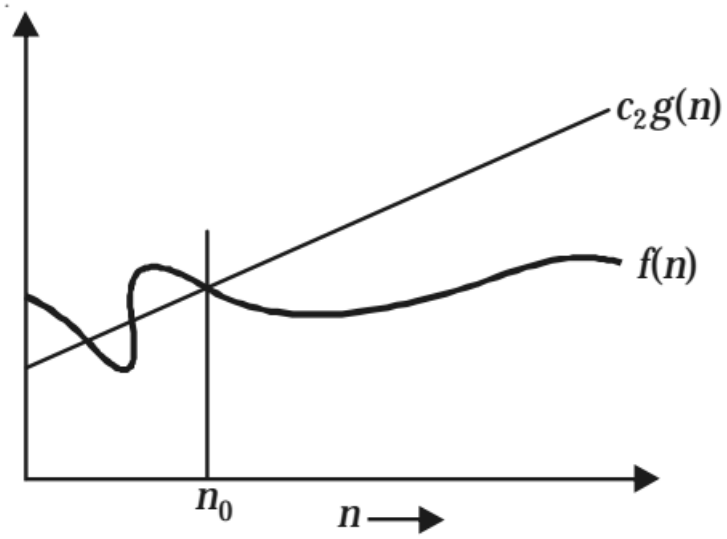
1. **Big-oh notation (O)**
2. **Omega Notation (Ω)**
3. **Theta Notation (θ)**

- The above notations will define function classes and they will do exactly what we discussed so far.

Big-oh Notation (O)

- If f(n) and g(n) be two non-negative functions of non-negative arguments, then **O(g(n))** is defined as

- **O(g(n)) : {f(n) | f(n) is a non-negative function such that ∃ constants c2 and n0 such that f(n) ≤ c2g(n) for n ≥ n0}**



So O-notation gives the **asymptotic upper bound** for a function to within a constant factor.

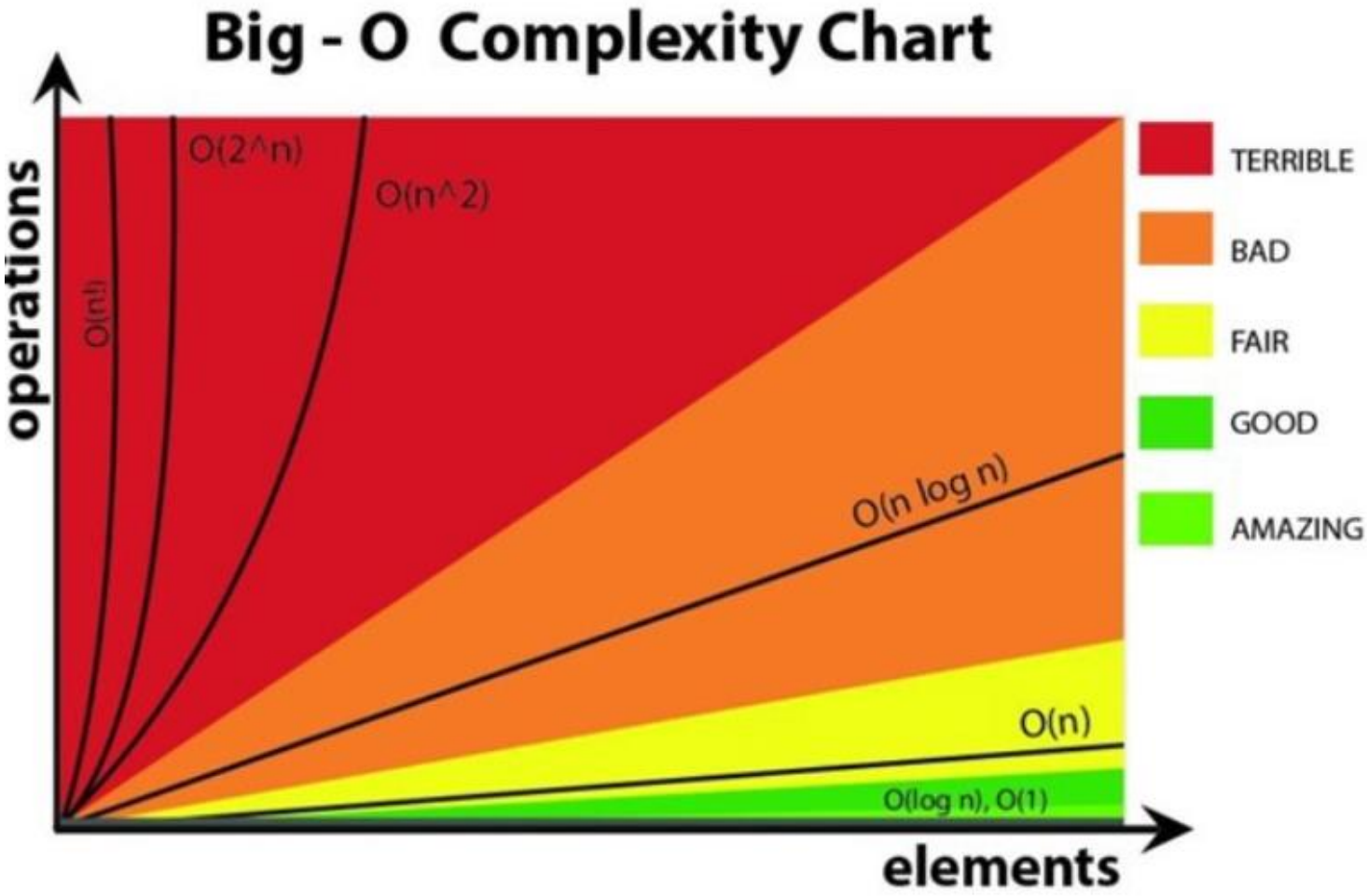
Big-oh Notation (O)

Example: Go through the following example where I have taken a sample function `f(n) = 4n + 3` and `g(n) = n` . Now we need to see whether we can say `f(n) = O(g(n))` . How we can check that? We have to try and see whether we can find values for `c` and `n0` where it will fulfil the condition `if f(n) ≤ cg(n) for any n ≥ n0 where c, n are real numbers and c > 0 and n0 ≥ 1 then we say f(n) = O(g(n)).` If we are successful then we can say `f(n) = O(g(n)) when f(n) = 4n + 3 and g(n) = n` . See the proof below in Figure_01.

Let's see whether $f(n) \leq cg(n)$
For that $4n + 3 \leq cn$ *for some* $c > 0$ *and some* $n_0 \geq 1$
when $c = 5$ *and* $n_0 = 3$ *for any* $n \geq 3$ *this* $4n + 3 \leq 5n$ *is true*
Thus, we say if $f(n) = 4n + 3$ *and* $g(n) = n$
$f(n) = O(g(n))$

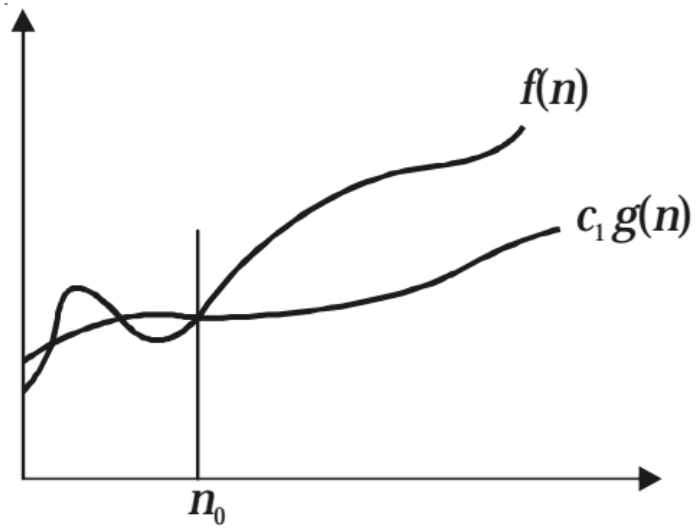
Figure_01

Big-oh Notation (O)



Omega Notation (Ω)

- If f(n) and g(n) be two non-negative functions of non-negative arguments, then **Ω(g(n))** is defined as
- **Ω(g(n)) :{f(n) | f(n) is a non-negative function such that ∃ constants c1 and n0 such that c1g(n) ≤ f(n) for n ≥ n0**



The Ω-notation provides us **asymptotic lower bound** for a function within a constant factor.

Omega Notation (Ω)

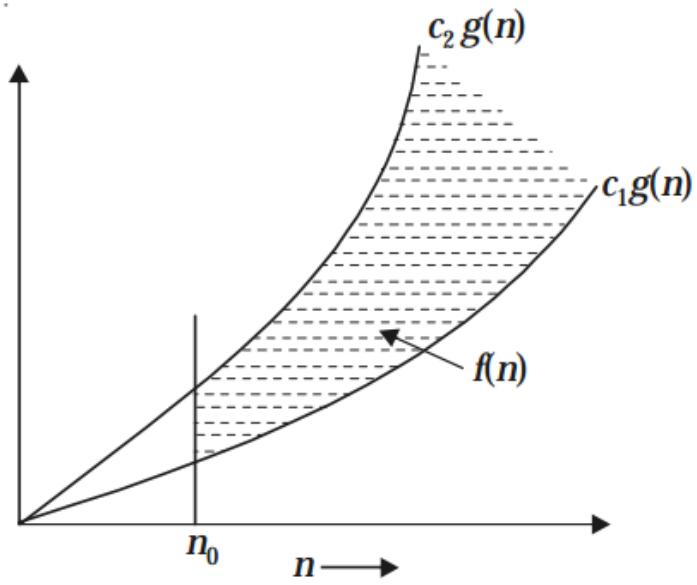
Example: Let's take the same function `f(n) = 4n + 3` and `g(n) = n` . Now we need to see whether we can say `f(n) = Ω(g(n))` . For that, we have to try and see whether we can find values for `c` and `n0` where it will fulfil the condition `if f(n) ≥ cg(n) for any n ≥ n0 where c, n are real numbers and c > 0 and n0 ≥ 1 then f(n) = Ω(g(n)).` If we are successful then we can say `f(n) = Ω(g(n)) when f(n) = 4n + 3 and g(n) = n` . See the proof below in Figure_02.

Let's see whether $f(n) \geq cg(n)$
For that $4n + 3 \geq cn$ *for some* $c > 0$ *and some* $n_0 \geq 1$
when $c = 1$ *and* $n_0 = 1$ *for any* $n \geq 1$ *this* $4n + 3 \geq n$ *is true*
Thus, we say if $f(n) = 4n + 3$ *and* $g(n) = n$
$f(n) = \Omega(g(n))$ *or simply* $4n + 3 = \Omega(n)$

Figure_02

Theta (θ) Notation

- Let f(n) g(n) be two **non-negative** functions of non-negative arguments, then θ(g(n)) is defined as:
- θ(g(n)) : {f(n) | f(n) is a non-negative function such that ∃ constants c1, c2, n0 such that c1g(n) ≤ f(n) ≤ c2 g(n) this is true not necessarily for all values of n but certainly for n ≥ n0}



This means that g(n) provides a nice, tight bound on f(n).

Theta (θ) Notation

- Recall the properties that we have previously discussed. Whenever what the class structure we define, we should give importance to $\mathbf{n \rightarrow \infty}$.
- Here in the definition of θ(g(n)), "for $n \geq n0$" is the part of our requirement.
- We are only bothered about what happens when $n \geq n0$, we are not at all worry about smaller values of n, as well as not worry about constant multipliers.
- In the relation $c1g(n) \leq f(x) \leq c2g(n)$, we want f(n) to be **sandwiched** between c1g(n) and g(c2n).

Theta (θ) Notation

Example: Let's take the same function `f(n) = 4n + 3` and `g(n) = n` . Now we need to see whether we can say `f(n) = Θ(g(n))` . For that, we have to try and see whether we can find values for `c1, c2` and `n0` where it will fulfil the condition `if c1g(n) ≤ f(n) ≤ c2g(n) for any n ≥ n0 where c1, c2, n are real numbers and c1,c2 > 0 and n0 ≥ 1 then f(n) = Θ(g(n))` . See the proof below in Figure_03.

Let's see whether $c_1 g(n) \leq f(n) \leq c_2 g(n)$
when $c_1 = 1$ *and* $n_0 = 3$ *for any* $n \geq 3$ *this* $4n + 3 \geq n$ *is true*
when $c_2 = 5$ *and* $n_0 = 3$ *for any* $n \geq 3$ *this* $4n + 3 \leq n$ *is true*
Thus, we can say if $f(n) = 4n + 3$ *and* $g(n) = n$
$f(n) = \Theta(g(n))$

Figure_03


Thank You