CS & IT ENGINEERING

Algorithms

Analysis of Algorithm

Lecture No.- 04

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

- **Topic** — Apriori Analysis ✓
- **Topic** — Types of Analysis : $\boxed{W.C;\ B.C;\ Av.C}$
- **Topic** — Worst-Case and Best-Case Behaviour
- **Topic**
- **Topic**

Topics to be Covered

Topics

Asymptotic Notations

Big – Oh, Big – Omega, Theta Notations

Small Notations

Topic: Asymptotic Notations

Time-Complexity ~ T(n)

Step-count

Order of Magnitude

$T(n) = 1 + n + n^2$

$= 4n^2 + 8n + 6$

(A.S.N) → repr. with a suitable Notation

Charact

Polynomial (n^x)

Log

Const

Quad

Cubic

Exponential

a^n $(a > 1)$

Topic: Asymptotic Notations

Asymplotatic Notations (ASN)

↓

Math Tool to obtain / repr. Bounds of functions

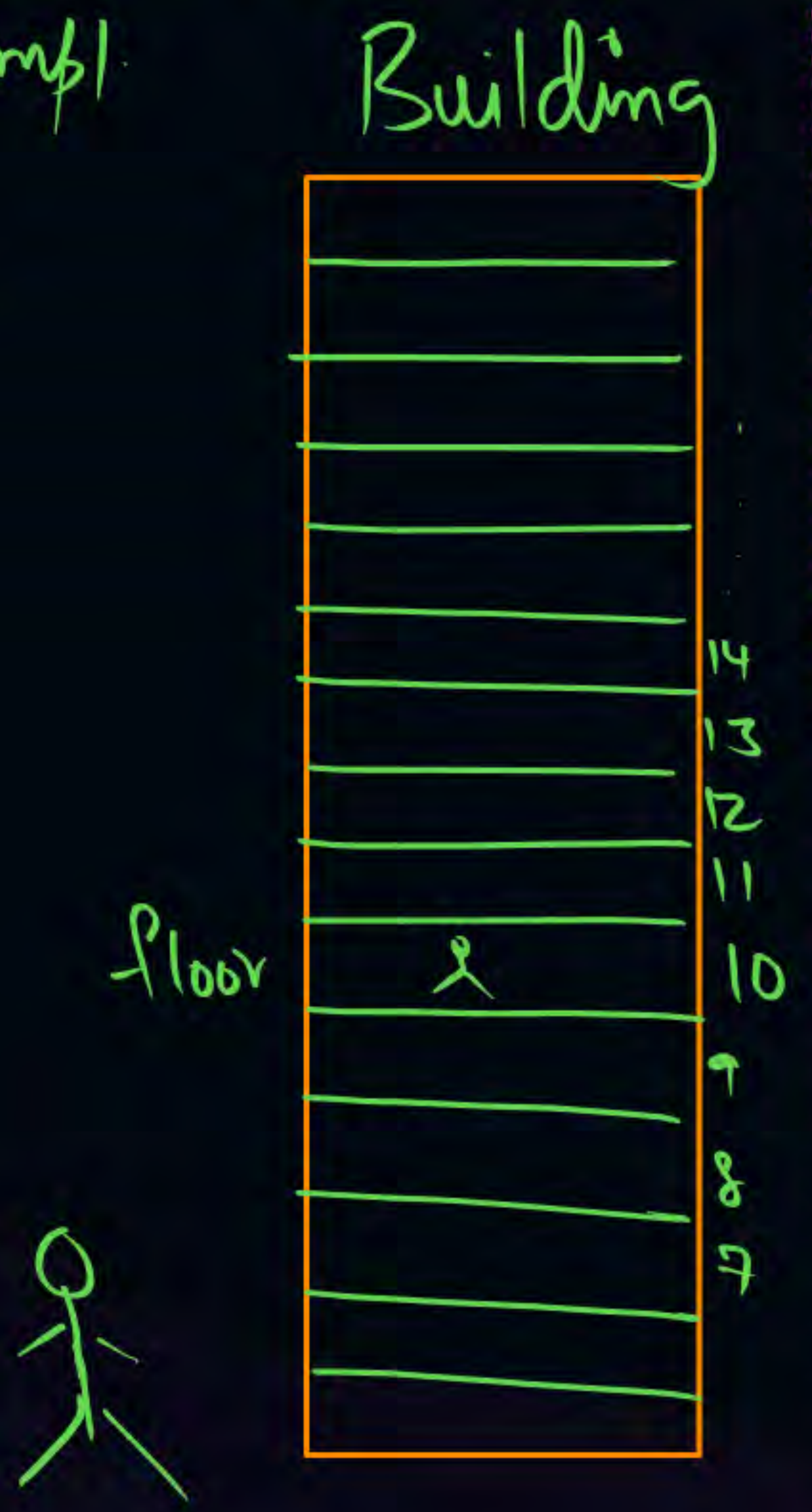
- Upper Bound ⟨Max⟩
- Lower Bound ⟨Min⟩
- Tight Bound

Repr. Time & Space Compl of Algo by functions

↓

(ASN)

- U.B (Max)
- L.B (Min)



Topic: Asymptotic Notations

ASN

Big

Upper Bound

→ Big-oh : O

Lower Bound

→ Big-omega : Ω

→ Theta : Θ

Tight Bound

Small | Little

→ Little oh : o → Proper U.B

→ Little omega : ω → Proper L.B

Topic: Asymptotic Notations

$\Longrightarrow$ Let 'f' & 'g' be functions from the set of integers/Reals to Real numbers;

① Big-oh (O): Upper Bound

$f(n)$ in $O(g(n))$ iff there exists some constants $c > 0$ and $n_0 > 0$

such that $\boxed{f(n) \le c \cdot g(n), \text{ whenever } n \ge n_0}$

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

$f(n) \in \underbrace{O(g(n))}_{\text{Set}}$

Topic: Asymptotic Notations

$f(n) = 1 + n + n^2 \quad O(n^2)$

$1 + n + n^2 \leq n^2 + n^2 + n^2$

$f(n) \leq 3 \cdot n^2, \quad n \geq 1$

$c \quad g \quad n_0$

$\boxed{f(n) \text{ is } O(n^2)}$

$1 + n + n^2 \leq 5 \cdot n^3, \quad n > 1$

$1 + 1 + 1 \leq 5 \cdot 1$

$1 + 2 + 4 \leq 5 \cdot 8$

$f(n)$ is $O(n^2)$ ✓ ∴ Closest | Highest U.B

$f(n)$ is $O(n^3)$ ✓

$f(n)$ is $O(n^4)$ ✓

Whenever we determine the U.B & L.B, we should find that function 'g' which is closest to the given function;

Construction:

(L.B) → 50L

B_2:

~~4~~ 45L

> 5000 rs

50L

B_1: U.B

< 100L

< 50L

200 Sq.yd.

(G+1) 3R 1H

House

A → B

5km

Abids

> 5s

(20m) ✓

1 Hr

1 Day

Ambani's

Business_Man :

- walk :
- Cycle : 15K
- Bike : 80K
- Car : 20L
- Jet : 100 cr

Investment

O(crs)

2) Big-omega (Ω): Lower Bound

$f(n)$ is $\Omega(g(n))$ iff there exists Const's 'c' & 'n_0' such that $\boxed{f(n) \boxed{\geq} c \cdot g(n), \text{ whenever } n \geq n_0;}$

1) $f(n) = 1+n+n^2$

- $\Omega(1)$ ✓
- $\Omega(n)$ ✓
- $\boxed{\Omega(n^2)}$ ✓

$1+n+n^2 > 1 \cdot 1, \quad n \geq 1$

$1+n+n^2 > 1 \cdot n, \quad n > 1$

$\boxed{1+n+n^2 \geq \underset{c}{1} \cdot \underset{g}{n^2}, \quad n > 1}$

$O(n^2)$

$1+n+n^2 \leq 3 \cdot n^2$

3) Theta (Θ) : Tight Bound :

$f(n)$ is $\Theta(g(n))$ iff $f(n)$ is $O(\underline{g(n)})$

& $f(n)$ is $\Omega(\underline{g(n)})$

$$\underbrace{1+n+n^2}_{f} \begin{cases} O(\overset{g}{n^2}) \\ \Omega(\underset{g}{n^2}) \end{cases} \therefore \Theta(n^2)$$

$$\boxed{c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)}$$

Topic : Analysis of Algorithms

Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity.

Big O is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann-Landau notation or asymptotic notation. The letter O was chosen by Bachmann to stand for Ordnung, meaning the order of approximation.

In computer science, big O notation is used to classify algorithms according to how their run time or space requirements grow as the input size grows.

In analytic number theory, big O notation is used to express a bound on the difference between an arithmetical function and a better understood approximation; a famous example of such a difference is the remainder term in the prime number theorem.

Topic : Analysis of Algorithms

Big O notation is also used in many other fields to provides similar estimates.

Big O notation characterizes functions according to their growth rates: different functions with the same asymptotic growth rate may be represented using the same O notation. The letter O is used because the growth rate of a function is also referred to as the order of the function. A description of a function in terms of big O notation usually only provides an upper bound on the growth rate of the function.

Associated with big O notation are several related notations, using the symbols O, Ω, ω and Θ, to describe other kinds of bounds on asymptotic growth rates.

Definition: A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation f(n) = O(g(n)) means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n".

Formal Definition: f(n) = O(g(n)) means there are positive constants c and k, such that $0 < f(n) \leq cg(n)$ for all $n \geq K$. The values of c and k must be fixed for the function f and must not depend on n.

Big-Omega Notation (Ω):

Similar to big O notation, big Omega (Ω) function is used in computer science to describe the performance or complexity of an algorithm. If a running time is $\Omega(f(n))$, then for large enough n, the running time is at least k.f(n) for some constant k.

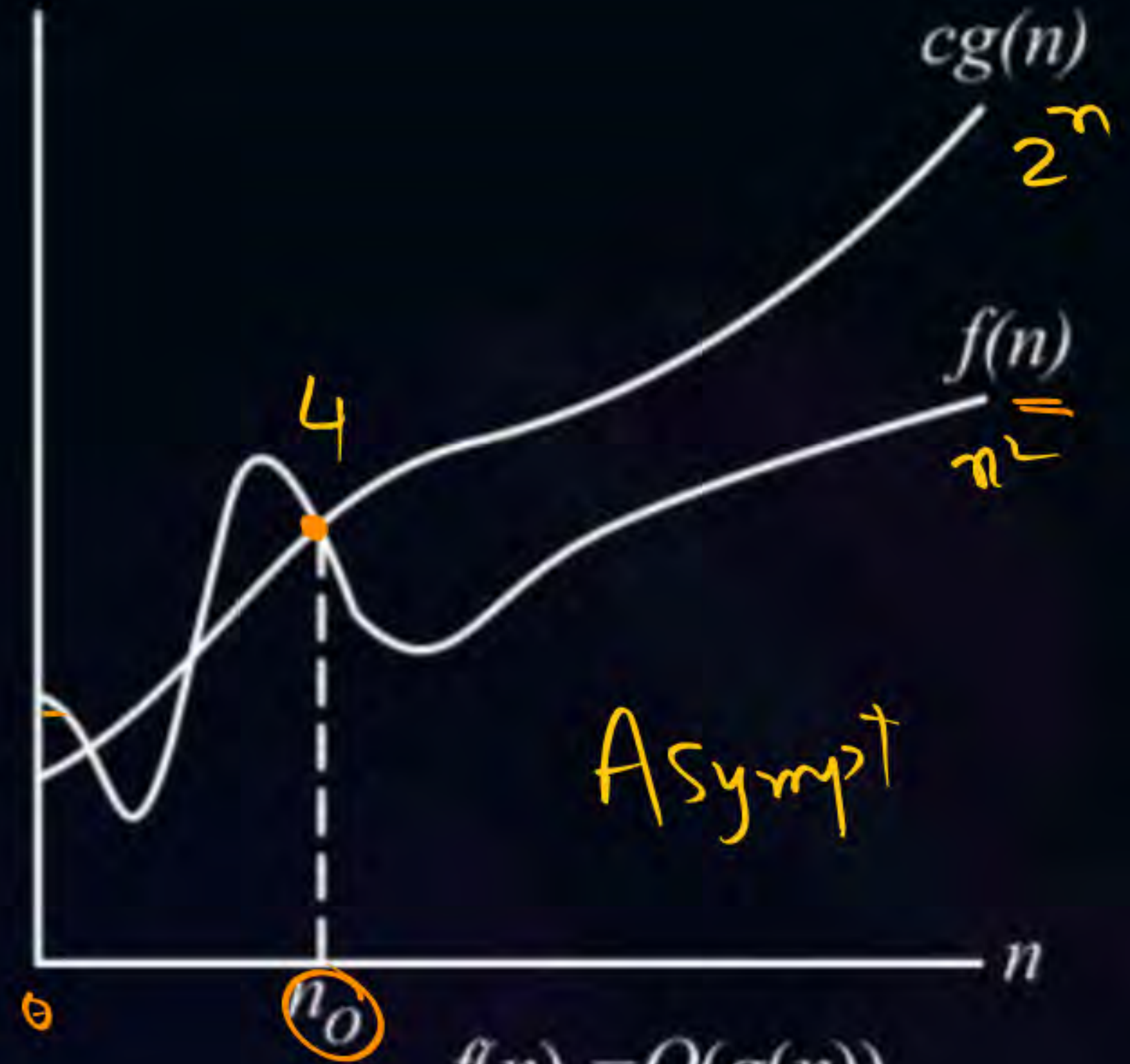
The formal definitions associated with the Big Notation are as follows:

- $f(n) = O(g(n))$ means $c \, . \, g(n)$ is an upper bound on $f(n)$. Thus there exists some constant c such that $f(n)$ is always $\leq c.g(n)$, for large enough n (i.e., $n \geq$ no for some constant n_0).
- $f(n) = \Omega(g(n))$ means $c.g(n)$ is a lower bound on $f(n)$. Thus there exists some constant c such that $f(n)$ is always $\geq c. \, g(n)$, for all $n \geq$ no.

Topic: Analysis of Algorithms

$f(n) = n^2$; $g(n) = 2^n$

n	f	g
1	1	2
2	4	4
3	9	8
4	16	16
5	25	32
6	36	64
7	49	128



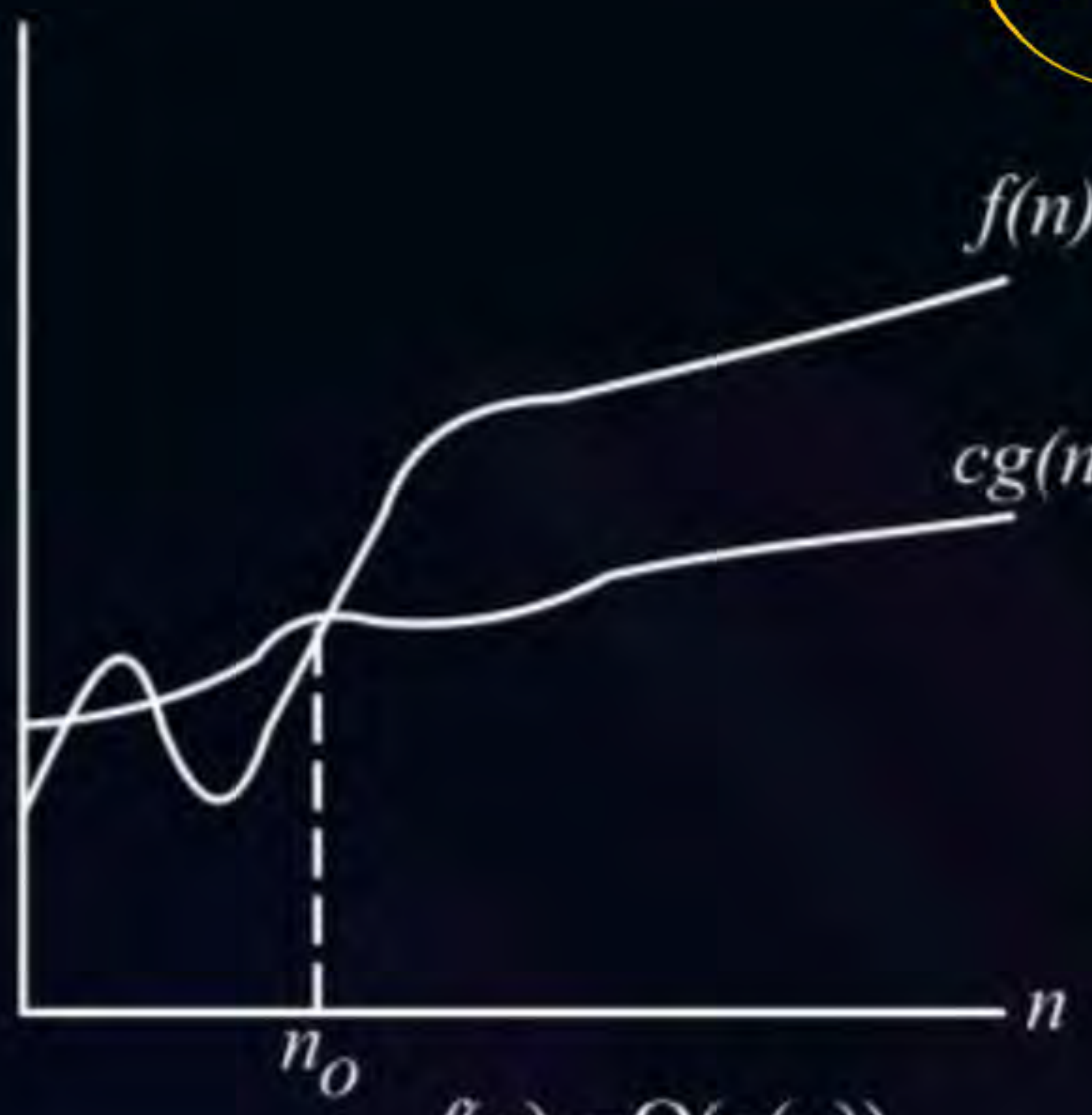
$f(n) = O(g(n))$

Set

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

We write $f(n) = O(g(n))$ to indicate that a function $f(n)$ is a member of the set $O(g(n))$. Note that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$, since Θ-notation is a stronger notion than O-notation. Written set-theoretically, we have

(Big-omega)



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

$O(n^3)$

c	$n \cdot \log n$
n	$n^2 \cdot \log n$
n^2	$\sqrt{n}$
$\log n$	

$c \le a \cdot n^3$

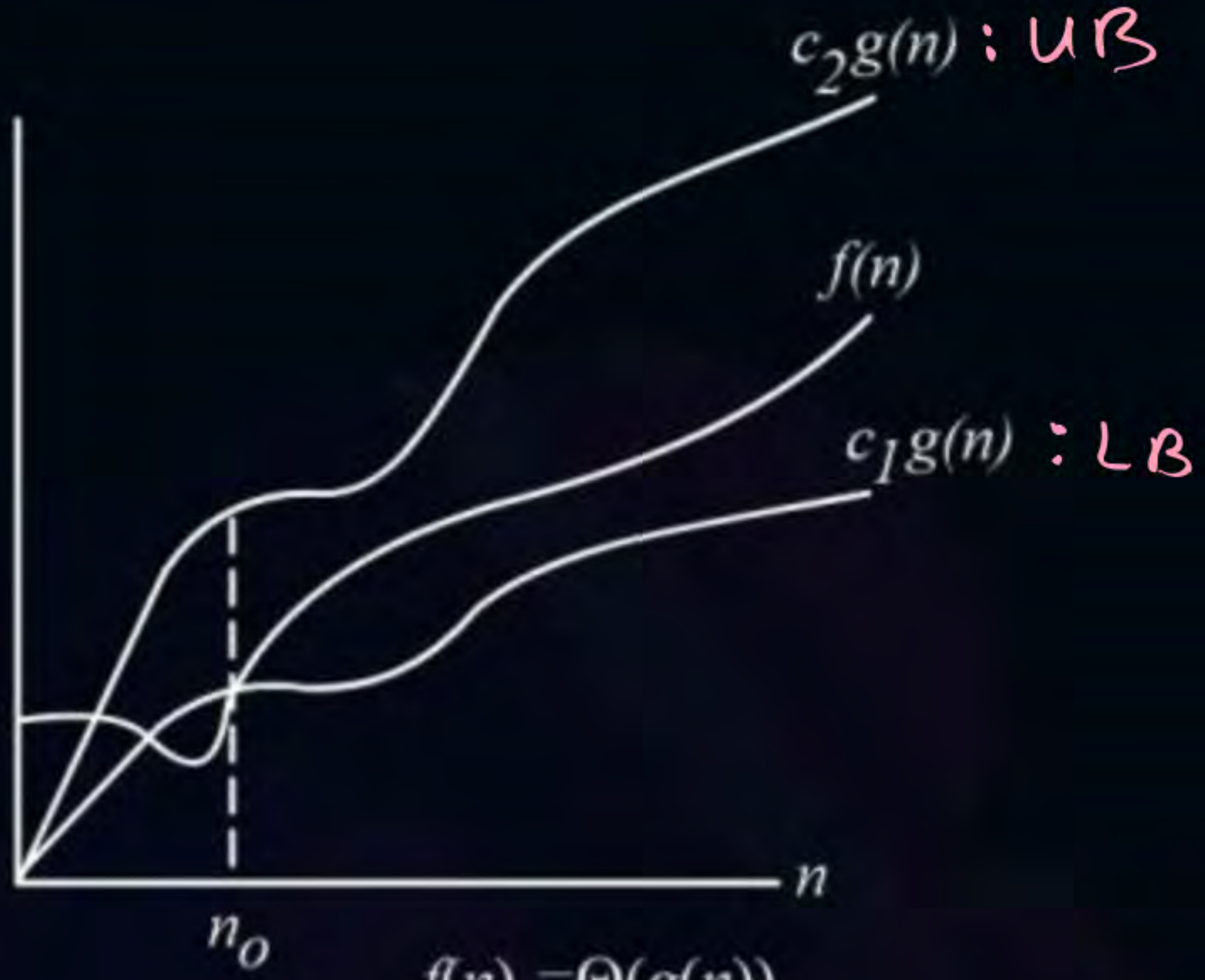
$n \le a \cdot n^3$

$f \quad n^2 \le a \cdot n^3$

Topic: Analysis of Algorithms

- $f(n) = \Theta(g(n))$ means $c_1.\, g(n)$ is an upper bound on $f(n)$ and $c_2.g(n)$ is a lower bound on $f(n)$, for all $n \geq no$. Thus there exist constants c_1 and c_2 such that $f(n) \leq c_1.g(n)$ and $f(n) \geq c_2.g(n)$. This means that $g(n)$ provides a nice, tight bound on $f(n)$.

Topic: Analysis of Algorithms



$f(n) = \Theta(g(n))$

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. ∎

1) $f(n) = 1 + n + n^2 \begin{cases} O(n^2) \\ \Omega(n^2) \end{cases} \therefore \Theta(n^2)$

2) $f(n) = n \begin{cases} O(n) \\ \Omega(n) \end{cases}$

$\boxed{n \leq c \cdot n}$ $n > 1$

$n \leq 2 \cdot n \; , \; n > 1$

$n \geq d \cdot n \; , \; n > 1$

$\therefore \Theta(n)$

$\geq \frac{1}{2} \cdot n$

3) $f(n) = 2^{100} \begin{cases} O(1) \\ \Omega(1) \end{cases} \therefore \Theta(1)$

$\boxed{2^{100} \leq 3^{100} \cdot 1}$

$f(n) = c \begin{cases} O(1) \\ \Omega(1) \end{cases} \therefore \Theta(1)$

4) $f(n) = n + \log n$ $\begin{cases} O(n) \\ \Omega(n) \end{cases}$ $\therefore \Theta(n)$

5) $f(n) = \sqrt{n} + \log n$ $\begin{cases} O(\sqrt{n}) \\ \Omega(\sqrt{n}) \end{cases}$

$\therefore \Theta(\sqrt{n})$ ✓

$n + \log n \le n + n$

$\le 2 \cdot n$, $n > 1$

c

$n + \log n \ge 1 \cdot n$, $n > 1$

c

$\log \sqrt{n}$

$\log (n)^{1/2}$

$\frac{1}{2} \cdot \log n$ $>$

$(\sqrt{n})$

$\sqrt{64}$

8

$\log_2 n$

$\log_2 64$

6

$\log \log n$

$$f(n) = n^{1/\log n} = O(\quad)$$

THANK - YOU