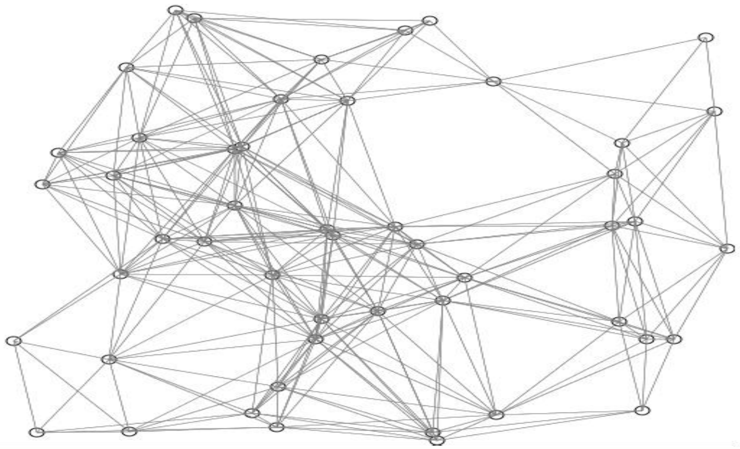
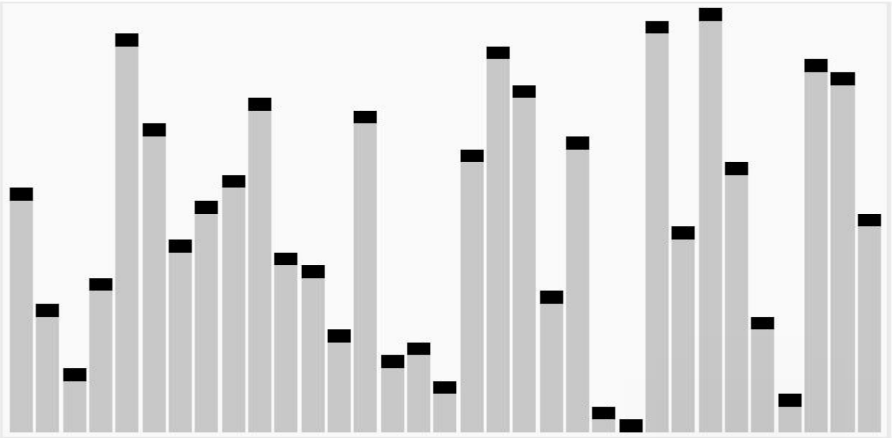
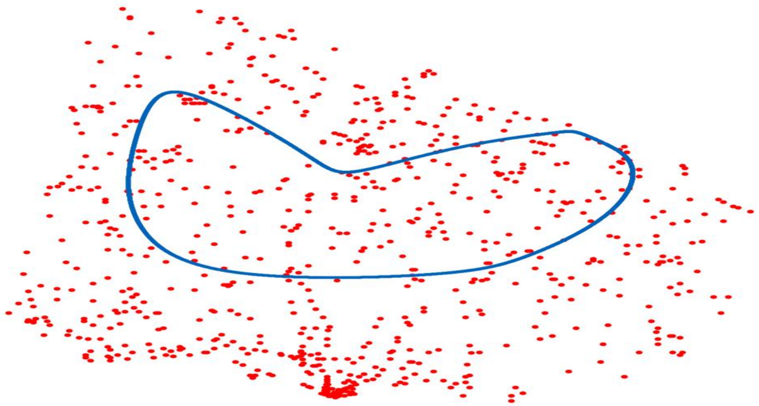
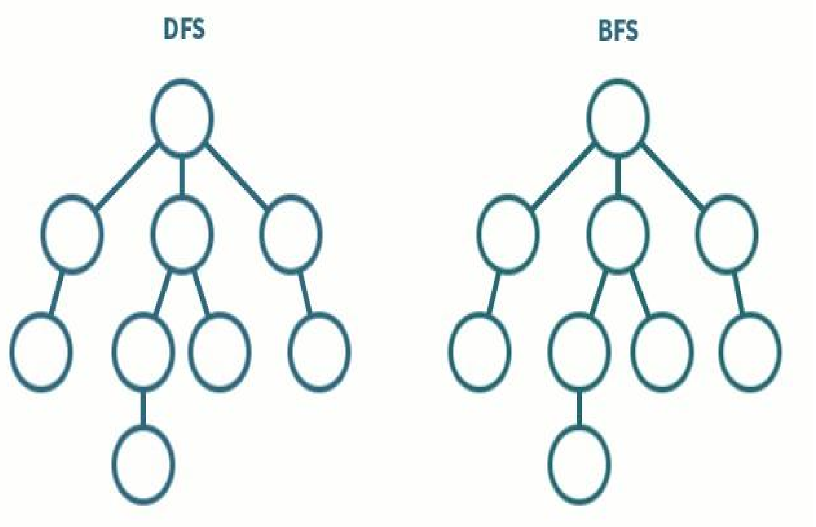








Design and Analysis of Algorithm

Dr. Supriyo Mandal

- Order of Growth and Approximation

- Asymptotic Notations:
  - Big Oh
  - Big Omega
  - Theta

Time of execution $\propto$ **Input Size**

- $T_1 = 5n + 10$
- $T_2 = 100n + 20$

  $T_3 = 19n$

Linear growth

Quadratic Growth

- $T_4 = n^2 + 5n + 3$
- $T_5 = 6n^2 + 7n + 10$
- $T_6 = 5n^2$

Logarithmic Growth

- $T_1 = \log n$
- $T_2 = 2\log n + 5$

Exponential Growth

- $T_1 = 2^n + 3$
- $T_2 = 3^n$
- $T_3 = 5^n + 6$

Constant Growth

- $T_1 = 5$
- $T_2 = 100$
- $T_3 = 1000$
- $T_4 = 100000000000$
- $T_5 = 100000000000000000000000000000$

Comparison of the Functions

$$10 < \log\log n < \log n < \sqrt{n} < n < n\log n < n^{\frac{3}{2}}$$

$$< n^2 < n^3 < n^{\log n} < n^{\sqrt{n}} < 2^n < n! < n^n$$

Comparison of the Functions

Arrange the following function in increasing order of their growth.

$n^{\sqrt{n}}$, $2^{\sqrt{n}\log_2 n}$, n!

Considers the following functions:

f (n) = 2^n

g(n) = n!

h(n) = n^{logn}

Asymptotic Notations

Asymptotic Notation is used to describe the running time of an algorithm - how much time an algorithm takes with a given input n.

- ☐ **Time complexity**
- ☐ **Space complexity**

$f(n)$ *is representing the running time of an algorithm*

- ❑ **Big O**
- ❑ **Big Ω**
- ❑ θ

ASSUMPTIONS:

1. $f(n)$ *is an increasing function*
2. $f(n)$ *is positive*
3. n *is a large number*

Big O Notation

- Gives the tight **upper bound** on the running time of an algorithm
- Represents the **worst** case behaviour of algorithm

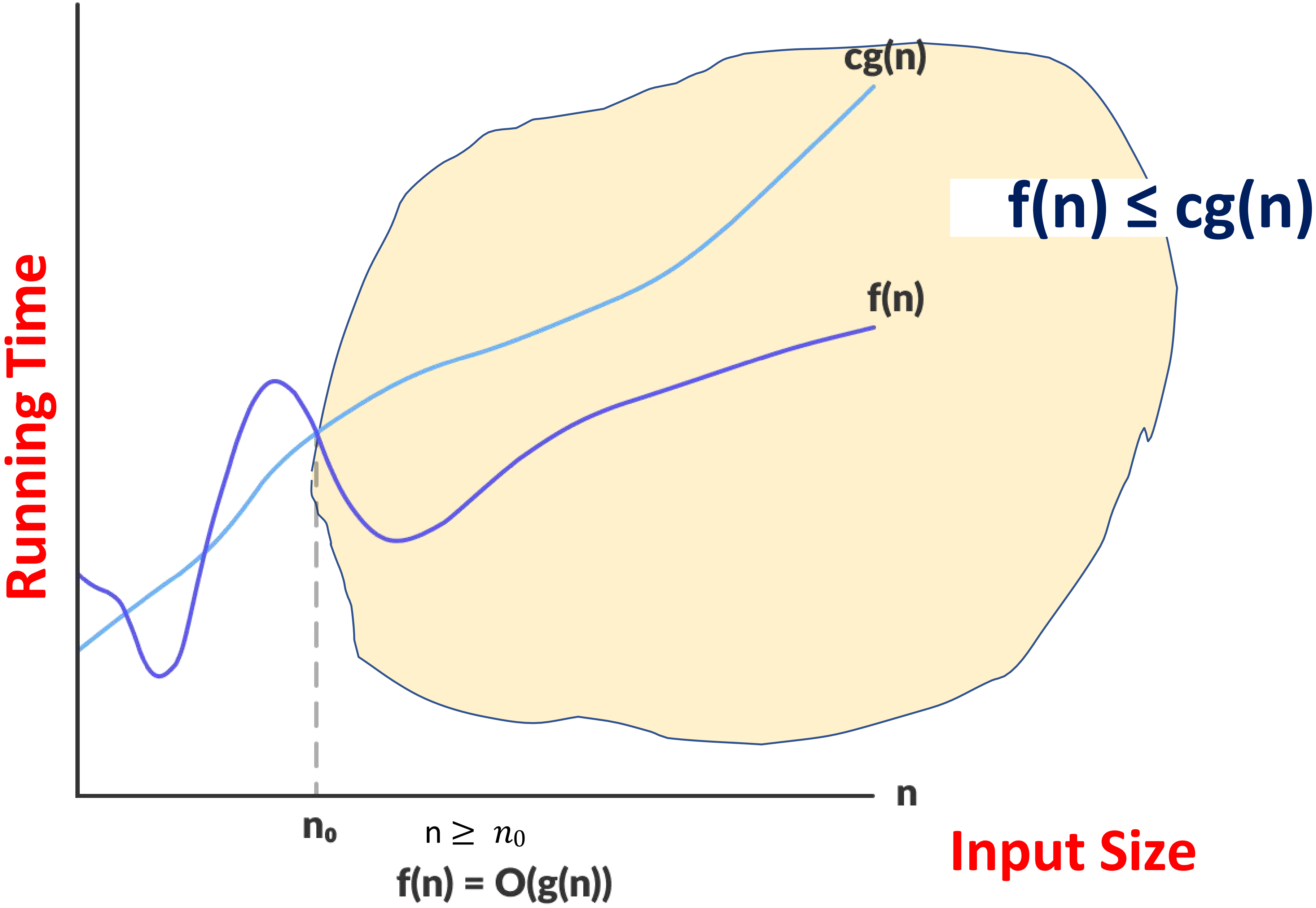
f(n) = O(g(n)) **=** **g(n) is asymptotically larger or equal than f(n)**

Big O Notation

If $f(n) = O(g(n))$ then there exists 2 positive constants **c** and n_0 such that

$$f(n) \le cg(n) \quad \forall\, n \ge n_0$$

UPPER Bound



Running Time
cg(n)
f(n) ≤ cg(n)
f(n)
n
n_0
$n \geq n_0$
f(n) = O(g(n))
Input Size

Big O Notation

$f(n) = 3n^2 + n+4$

$g(n) = n^2$

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

$f(n)=3n+2$

$g(n)=n$

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

Big Ω Notation

- Gives the tight lower bound on the running time of an algorithm
- Represents the BEST case behaviour of algorithm

$f(n) = \Omega(g(n))$ = g(n) is asymptotically smaller or equal than f(n)

Big Ω Notation

1. $n^3 = \Omega(n^2)$
2. $\log n = \Omega(\log\log n)$

TRUE

1. $n = \Omega(n^2)$
2. $n\log n = \Omega(n^2)$

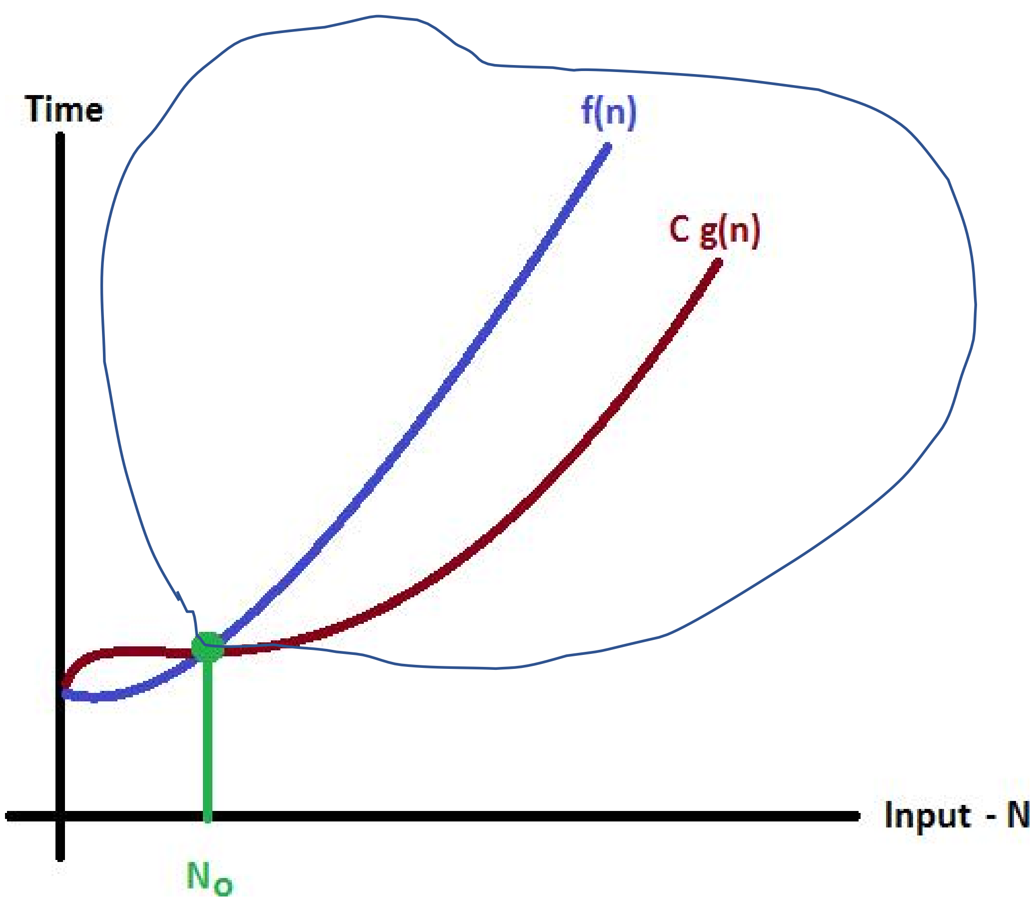
FALSE

Big Ω Notation

If $f(n) = \Omega(g(n))$ then there exists 2 positive constants c and n_0 such that

$$f(n) \geq c\,g(n) \quad \forall\, n \geq n_0$$

LOWER Bound



$$f(n) \geq \textcolor{red}{c}g(n)$$

Big Ω Notation

$f(n) = 3n^2 + n+4$
$g(n) = n^2$

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

$f(n)=3n+2$
$g(n)=n$
$f(n) = \Omega\ (g(n))$??

θ Notation

- Gives the tight **bound** (**Lower Bound & Upper Bound**) on the running time of an algorithm

f(n) = θ(g(n))

=

g(n) is asymptotically equal to f(n)

θ Notation

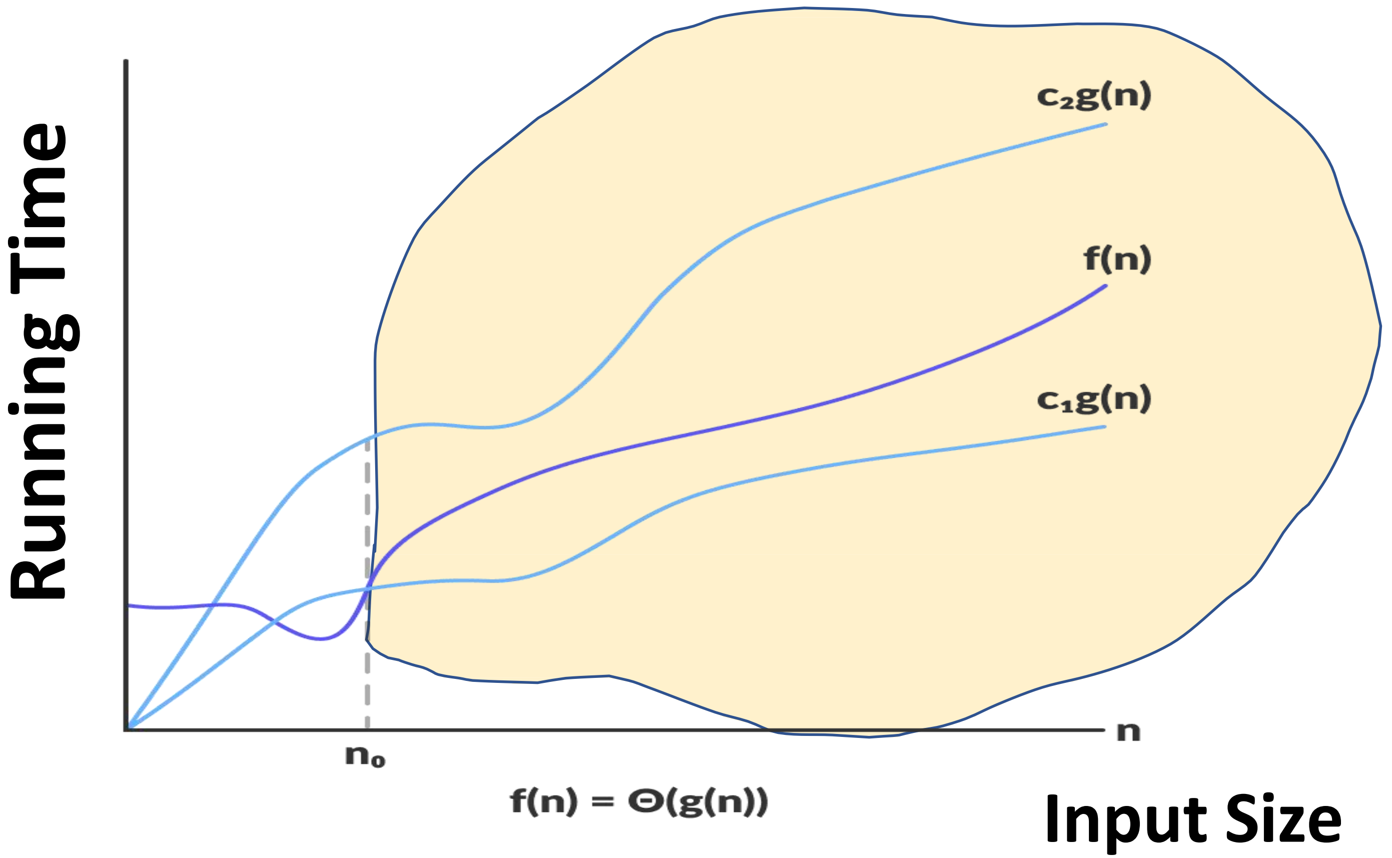
If $f(n) = \theta(g(n))$ then there exists 3 positive constants c_1 and c_2 and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall\, n \geq n_0$$

Lower Bound

Upper Bound

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \qquad \forall\, n \geq n_0$$



θ Notation

$$\underbrace{c_1 g(n) \le \overbrace{f(n) \le c_2 g(n)}^{f(n) = O(g(n))}}_{f(n) = \Omega(g(n))} \quad \forall\, n \ge n_0$$

θ Notation

$f(n) = \theta(g(n))$ $\equiv$ $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Big θ Notation

1.$f(n) = 3n^2 + n+4$

$g(n) = n^2$

$f(n) = \boldsymbol{\theta}(g(n))$??

$f(n)=3n+2$

$g(n)=n$

$f(n) = \boldsymbol{\theta}(g(n))$??

$f(n) = n^2$

$g(n) = n$

$f(n) = \boldsymbol{\theta}(g(n))$??