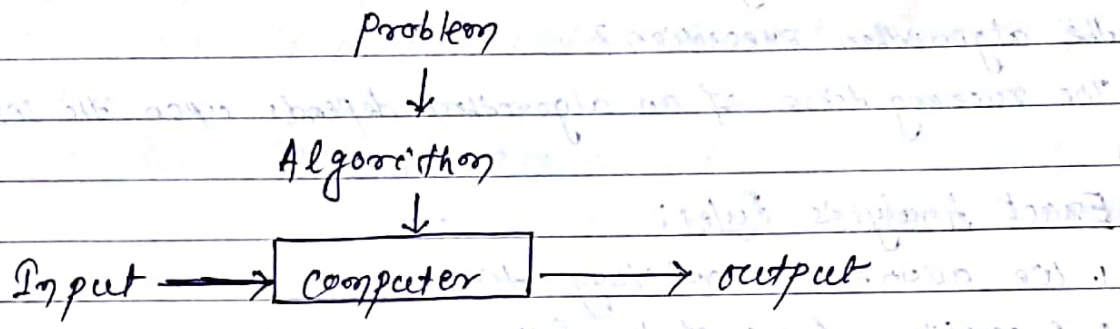
ALGORITHMS:

An algorithm is any well-defined computational procedure that takes some value or set of values as input and produces some value or set of values as output.

→ An algorithm is a sequence of unambiguous instructions for solving a problem i.e. for obtaining a required output for any legitimate input in a finite amount of time.

→ An algorithm is a step-by-step procedure for solving a problem in a finite amount of time.



→ An algorithm is a tool for solving a well specified computational problem.

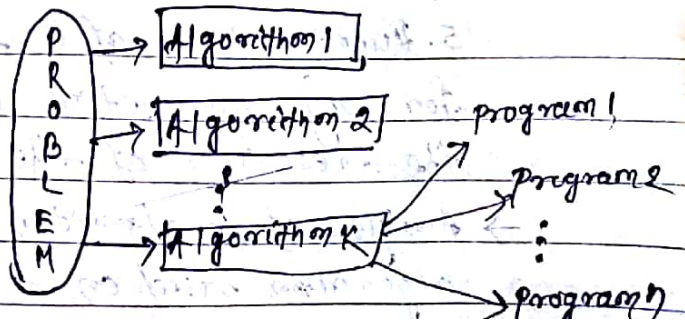
CHARACTERISTICS / FEATURES OF ALGORITHM

1. Input: Zero or more quantities are externally supplied. OR valid inputs are clearly specified.
2. Output: At least one quantity is produced. OR It can be proved to produce the correct output given a valid input.
3. Definiteness: Each instruction is clear and unambiguous.
4. Finiteness: The algorithm terminates after a finite number of steps.
5. Effectiveness: Each steps are sufficiently simple and basic.

DESIGN TECHNIQUES

There are many ways to design or create algorithms.

(1) Devide and conquer.
(2) Greedy Approch.
(3) Dynamic programming.
(4) Back Tracking.
(5) Branch and Bound.



Analysis Of An Algorithm:

Analysis of an algorithm means predicting the resources that the algorithm requires. Example of resources are memory, communication bandwidth or computer hardware and computational time.

→ Analysis of an algorithm focuses on time complexity and space complexity.

* Space Complexity: How much memory is required to execute the program?
* Time Complexity: How much time is taken to complete the algorithm execution?

The running time of an algorithm depends upon the size of I/P.

Exact Analysis Rules:

1. We assume an arbitary time unit.
2. Execution of one of the following operations takes time 1 (one):
   (a) Assignment operations.
   (b) Single I/O operations.
   (c) Single boolean operations, numeric comparisons.
   (d) Single arithmetic operations.
   (e) function return.
   (f) Array index operations, pointer dereferences.
3. Running time of a selection statement (if, switch) is the time for the condition evaluation + the maximum of the running times for the individual clauses in the selection.
4. Loop execution time is the sum, over the number of the loop is executed, of the body time + time for the loop checks and update operations, + time for the loop setup. (Always assume that the loop executes the maximum number of iterations possible).
5. Running time of a function call is 1 for setup + the time for any parameter calculations + the time required for the execution of the function body.

→ Analysis of an algorithm is to evaluate the performance of the algorithm based on

   (1) Input size.
   (2) Running time (worst-case and average case)
   (3) Order of growth.

How to Calculate Running Time of an Algorithm

We can estimate an algorithm's performance by counting the number of basic operations required by the algorithm to process an input of a certain size.

* Basic Operation: The time to complete a basic operation does not depend on the particular values of its operands. So, it takes a constant amount of time.

Examples: Arithmetic operation (addition, subtraction, multiplication, division), Boolean operation (AND, OR, NOT), comparison operation, Module operation, Branch operation etc.

* Input size: It is the number of input processed by the algorithm.

* Growth Rate: The growth rate of an algorithm is the rate at which the running time or cost of the algorithm grows as the size of the input grows.

Algorithms Vs Programs

1. program does not have to satisfy the finiteness condition but algorithm have finiteness condition.
2. Algorithm is the design phase of a problem but the program is the implementation phase of a problem.

Pseudo Code Conventions

For writing any algorithm following pseudo codes have been used:

1. proper identification should be followed.
2. Looping constructs while, for, repeat-until, if-then-else.
3. The Symbol ▷ and // indicates comment.
4. Left arrow ← is assignment operator.
5. Length[A] is length of Array.
6. A[i..j] indicates array from i to j.
7. Arithmetic operators +, −, *, / are used.
8. ↔ for swapping.

Expressing Algorithm: There are some way to express the algorithm 3

1. English 2. pseudo code 3. Real programming language.

pseudo code: It consists of keywords and english-like phrase which specify the flow of control. The pseudo code hides the implementation details. But it focus on the computational aspects of an algorithm.

GROWTH OF FUNCTIONS

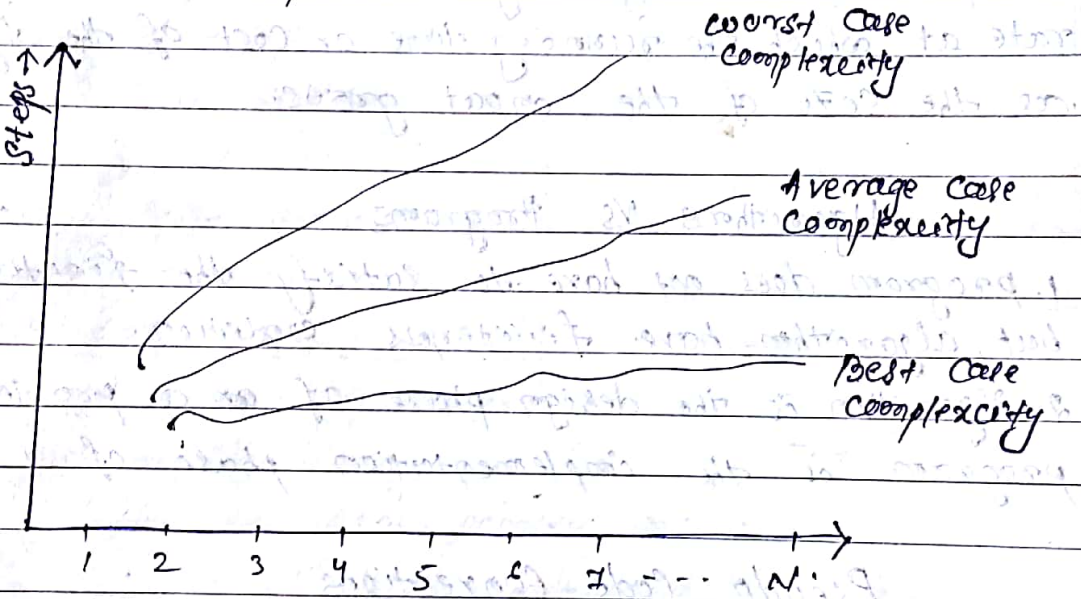
Best, worst and Average case complexity:

→ The best, worst and average cases of an algorithm express what the resource usage is at least, at most and on average respectively.

→ The worst-case complexity of the algorithm is the function defined by the maximum number of steps taken on any instance of size n.

→ The best-case complexity of the algorithm is the function defined by the minimum number of steps taken on any instance of size n.

→ The average-case complexity of the algorithm is the function defined by the average number of steps taken on any instance of size n.



Worst-Case Running Time:

The behaviour of the algorithm with respect to the worst possible case of the input instance. The worst-case running time of an algorithm is an upper bound on the running time for any input. It gives us a guarantee that the algorithm will never take any longer time.

Average-Case Running Time:

The average-case running time of an algorithm is an estimate of the running time for an average input.

Asymptotic Analysis

→ Asymptotic means a line that tends to converge to a curve, which may or may not touch the curve.

→ It is a line that stays within bounds.

Asymptotic Notations

→ Asymptotic notations is used to describe the running time of an algorithm. This shows the order of growth of function. We can map the time taken by an algorithm in terms of mathematical function.

→ Example: An algorithm taking n^2 microseconds can be thought of "the order of n^2" as unit is constant.

Why are Asymptotic Notations Important

1. It determines the algorithm's efficiency.
2. It allow the comparison of the performances of various algorithms.

→ Growth Rate of functions can be represented by asymptotic notations. There are 5 types of notation:

(1) Big-oh Notations (O)
(2) Big-omega Notations (Ω)
(3) Theta Notations (Θ)
(4) Little-oh Notation (o)
(5) Little-Omega Notation (ω)

(1) Big-Oh Notation (O):

→ It is the method of expressing the upper bound of an algorithm's running time. It is the measure of the longest amount of time it could possibly take for the algorithm to complete.

→ 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 $n > n_0$.

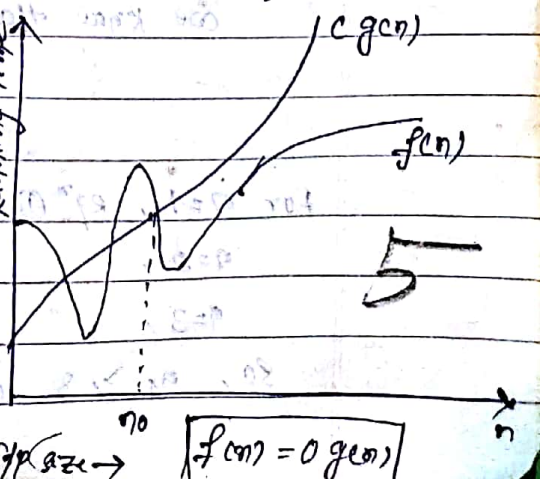
$$f(n) \leq c \cdot g(n)$$

Then $f(n)$ is Big-oh of $g(n)$.

This is denoted as, $f(n) = O(g(n))$

or

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0\}$



Example 1: prove that $5n-2$ is $O(n)$

proof: $f(n) = 5n-2$

$g(n) = n$

so, $f(n) \leq c \cdot g(n)$

$5n-2 \leq c \cdot n$

$5n-2 \leq 5n$ — ①

for $n=1$, eqn ① becomes, $3 \leq 5$ (True)

so, $n_0 \geq 1$ and $c=5$

$5n-2 \leq 5n$

$f(n) \leq c \cdot g(n)$

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

so, $5n-2$ is $O(n)$ for $n_0 \geq 1$ and $c=5$

Example 2: prove that $10n^3+7n+5$ is $O(n^3)$

proof: $f(n) = 10n^3+7n+5$

$g(n) = n^3$

we know, $f(n) \leq c \cdot g(n)$

$10n^3+7n+5 \leq c \cdot n^3$

$10n^3+7n+5 \leq 11n^3$ — ①

for $n=1$, eqn ① becomes, $22 \leq 11$ (False)

for $n=2$, eqn ① becomes, $99 \leq 88$ (False)

for $n=3$, eqn ① becomes, $296 \leq 297$ (True)

so, $n_0 \geq 3$ and $c=11$

$10n^3+7n+5 \leq 11n^3$

$f(n) \leq c \cdot g(n)$

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

Hence, $10n^3+7n+5$ is $O(n^3)$ for $n_0 \geq 3$ and $c=11$

Example 3: prove that n^2+5 is $O(n^2)$

proof: $f(n) = n^2+5$

$g(n) = n^2$

we know that $f(n) \leq c \cdot g(n)$

$n^2+5 \leq c \cdot n^2$

$n^2+5 \leq 2n^2$ — ①

for $n=1$, eqn ① becomes $6 \leq 2$ (False)

$n=2$, " $9 \leq 8$ (False)

$n=3$, " $14 \leq 18$ (True)

so, $n_0 \geq 3$ and $c=2$

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

$f(n) \leq c \cdot g(n)$

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

Hence, n^2+5 is $O(n^2)$

Example 4: prove that $2^{n+1} = O(2^n)$

proof: $f(n) = 2^{n+1}$

$g(n) = 2^n$

we know that, $f(n) \leq c \cdot g(n)$

$2^{n+1} \leq c \cdot 2^n$

$\Rightarrow 2^n \cdot 2^1 \leq c \cdot 2^n$

$\Rightarrow 2^1 \cdot 2^n \leq c \cdot 2^n$

$\Rightarrow 2 \cdot 2^n \leq 2 \cdot 2^n$ — ①

For $n=1$, eqn ① becomes $4 \leq 4$ (True)

so, $n_0 \geq 1$ and $c = 2$

$2 \cdot 2^n \leq 2 \cdot 2^n$

$2^{n+1} \leq 2 \cdot 2^n$

$f(n) \leq c \cdot g(n)$

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

Hence, $2^{n+1} = O(2^n)$

Example 5: prove that $n! = O(n^n)$

proof: $f(n) = n! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n$

$g(n) = n^n = n \cdot n \cdot n \cdot \cdots \cdot n \cdot n$

we know that $f(n) \leq c \cdot g(n)$

$\Rightarrow 1 \cdot 2 \cdot 3 \cdot \cdots \cdot (n-1) \cdot n \leq n \cdot n \cdot n \cdot \cdots \cdot n \cdot n$

$1 \leq n$

$2 \leq n$

$\vdots$

$n-1 \leq n$

$n \leq n$

$\Rightarrow n! \leq n^n$

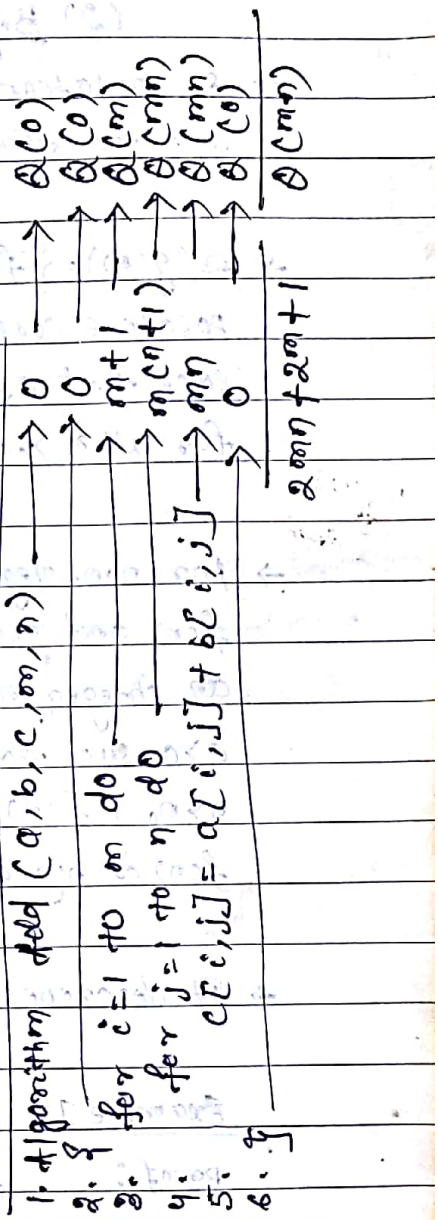
so, $n_0 = 1$ and $c = 1$

$1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n \leq 1 (n \cdot n \cdots n)$

$f(n) \leq c \cdot g(n)$

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

$n! = O(n^n)$ (proved)



1. sum(a, n) → 0 → θ(0)
2. { → 0 → θ(0)
3. s=0; → 1 → θ(1)
4. for i=1 to n do → n+1 → θ(n)
5. s = s + a[i] → n → θ(n)
6. return s → 1 → θ(1)

Total = 2n+3 → θ(n)

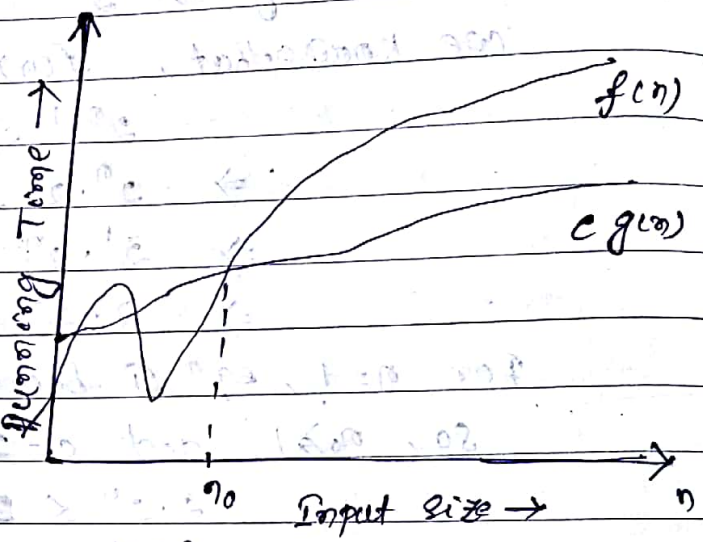
Assignment:

1. prove that $3n+1$ is $O(n)$
2. prove that n^2+2n+5 is $O(n^2)$

(2) Big-Omega Notation (Ω)

→ Ω notation provides the lower bound of an algorithm's running time.

→ $\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq c\,g(n) \leq f(n)$ for all $n \geq n_0\}$



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

→ 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 $n > n_0$, $f(n) \geq c \cdot g(n)$ then $f(n)$ is big omega of $g(n)$. This is denoted as

$$\boxed{f(n) = \Omega(g(n))}$$

→ It describes the best that can happen for a given data size.

Example 1: prove that $3n+2$ is $\Omega(n)$

proof: $f(n) = 3n+2$

$g(n) = n$

we know that $f(n) \geq c \cdot g(n)$

$3n+2 \geq c \cdot n$

$3n+2 \geq 3 \cdot n$ — (1)

for $n=1$, eqn (1) becomes $5 \geq 3$ (True)

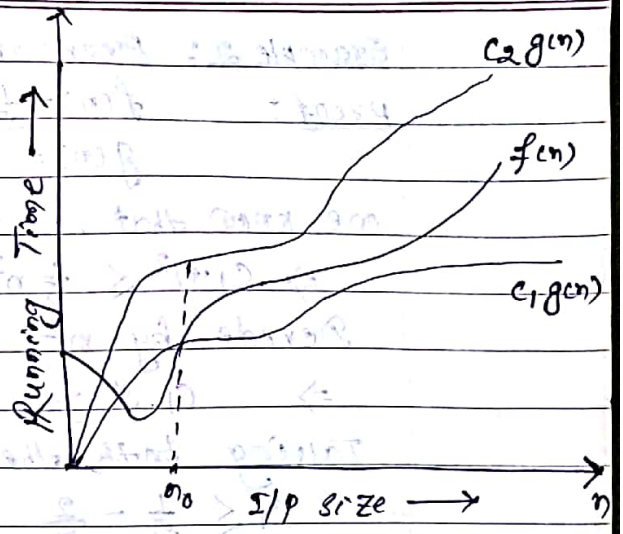
Hence, $n_0 \geq 1$ and $c=3$

$f(n) \geq c \cdot g(n)$

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

Hence $3n+2 = \Omega(n)$

(3) Theta Notation (Θ):



→ The lower and upper bound for the function 'f' is provided by the theta notation (Θ).

→ For non-negative functions $f(n)$ and $g(n)$ if there exists an integer n_0 and positive constants c_1 and c_2 i.e. $c_1 > 0$ and $c_2 > 0$ such that for all integers $n \geq n_0$

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

Then $f(n) = \Theta(g(n))$. It means "f is order g".

→ For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$

Example 1: prove that $\frac{1}{2}n^2 - 3n$ is $\Theta(n^2)$

Proof: $f(n) = \frac{1}{2}n^2 - 3n$

$g(n) = n^2$

we know that $c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$

$\Rightarrow c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

Devide by n^2, we get

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

Taking both the inequalities

$c_1 \leq \frac{1}{2} - \frac{3}{n}$

c_1 is a +ve constant i.e. $c_1 > 0$

If $n = 6$, then $c_1 = 0$ which is false

so, $n \geq 7$

Then $c_1 \leq \frac{1}{2} - \frac{3}{7} = \frac{7-6}{14} = \frac{1}{14}$

$\boxed{c_1 \leq \frac{1}{14}}$ $(\because n_0 \geq 7)$

$\frac{1}{2} - \frac{3}{n} \leq c_2$

c_2 is a +ve constant.

c_2 can have maximum value $1/2$ as -ve term is subtracted.

$\boxed{c_2 \geq 1/2}$ $(\because n_0 \geq 1)$

Hence, lowest value of n for which $f(n)$ lies between $c_1(g(n))$ and $c_2(g(n))$ is $n_0 = 7$

so, $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Example 2: Prove that $\frac{1}{5}n^2 - 2n = \Theta(n^2)$

Proof: $f(n) = \frac{1}{5}n^2 - 2n$

$g(n) = n^2$

We know that, $c_1 g(n) \leq f(n) \leq c_2 g(n)$

$\Rightarrow c_1 \cdot n^2 \leq \frac{1}{5}n^2 - 2n \leq c_2 \cdot n^2$

Devide by n^2

$\Rightarrow c_1 \leq \frac{1}{5} - \frac{2}{n} \leq c_2$

Taking both the inequalities

$c_1 \leq \frac{1}{5} - \frac{2}{n}$

c_1 is a +ve constant i.e. $c_1 > 0$

If $n = 10$, then $c_1 = 0$ (False)

So, $n = 11$

$c_1 \leq \frac{1}{5} - \frac{2}{11} = \frac{11-10}{55} = \frac{1}{55}$

$\boxed{c_1 \leq \frac{1}{55}} \quad (\because n_0 \geq 11)$

$\frac{1}{5} - \frac{2}{n} \leq c_2$

c_2 is a +ve constant i.e. $c_2 > 0$

c_2 can have maximum value $1/5$ as a -ve term is subtracted

$\boxed{c_2 \geq 1/5} \quad (\because n_0 \geq 1)$

Hence, $1/5 n^2 - 2n = \Theta(n^2)$

(4) Little-oh Notation (o):

→ Asymptotic upper bound provided by O-notation may or may not asymptotically tight. So, o-notation is used to denote an upper bound that isn't asymptotically tight.

$o(g(n)) = \{ f(n) :$ for any +ve constant $c > 0$, if a constant $n_0 > 0$ such that $0 \leq f(n) < c \cdot g(n)$ for all $n \geq n_0 \}$

→ The function $f(n)$ becomes insignificant relative to $g(n)$ as n approches infinity

i.e. $\boxed{\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0}$

→ The main difference between O-notation and o-notation is that in $f(n) = O(g(n))$, the bound $0 \leq f(n) \leq c \cdot g(n)$ holds for some constant $c > 0$, but in $f(n) = o(g(n))$, the bound $0 \leq f(n) < c \cdot g(n)$ holds for all constants $c > 0$

Example 1: prove that $3n+2$ is $o(n^2)$

proof: $f(n) = 3n+2$

$g(n) = n^2$

$$\lim_{n\to\infty} \frac{3n+2}{n^2} = \lim_{n\to\infty} \frac{3}{n} + \frac{2}{n^2} = 0$$

$\therefore f(n) = o(n^2)$ satisfy

Example 2: prove that $3n+2$ is not $o(n)$

proof: $f(n) = 3n+2$

$g(n) = n$

$$\lim_{n\to\infty} \frac{3n+2}{n} = \lim_{n\to\infty} 3 + \frac{2}{n} = 3$$

$\therefore f(n) \neq o(g(n))$ (Not satisfy)

(5) Little-Omega Notation (ω):

→ Little-Omega (ω) is used to denote an lower bound that is not asymptotically tight.

→ $f(n) \in \omega(g(n))$ iff $g(n) \in o(f(n))$

$\omega(g(n)) = \{ f(n)$: for any positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq c\,g(n) < f(n)$ $\forall\ n \geq n_0 \}$

→ The relation $f(n) = \omega(g(n))$ implies that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \quad \text{or} \quad \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

i.e. $f(n)$ becomes arbitrarily large relative to $g(n)$ as n approches infinity.

→ Example: prove $n! = \omega(2^n)$

sol^n we have $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ for $f(n) = \omega(g(n))$

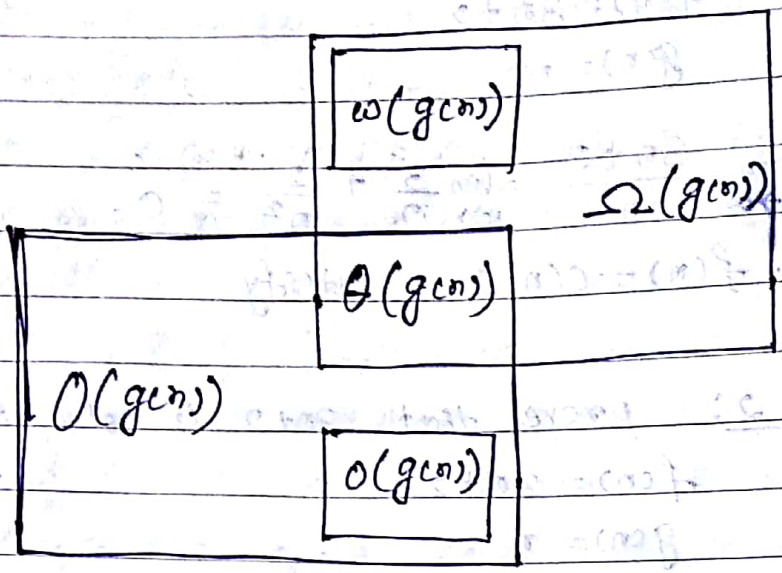
we can write $n! \approx n^n$

Applying $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^n}{2^n} \approx \left(\frac{n}{2}\right)^n \approx n^n$

Using L' Hospital's Rule $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^n}{1} = \frac{n\,n^{n-1}}{0} = \infty$

Thus $n! = \omega(2^n)$ (proved)

Relationships Between $O, o, \Theta, \Omega, \omega$ Notations



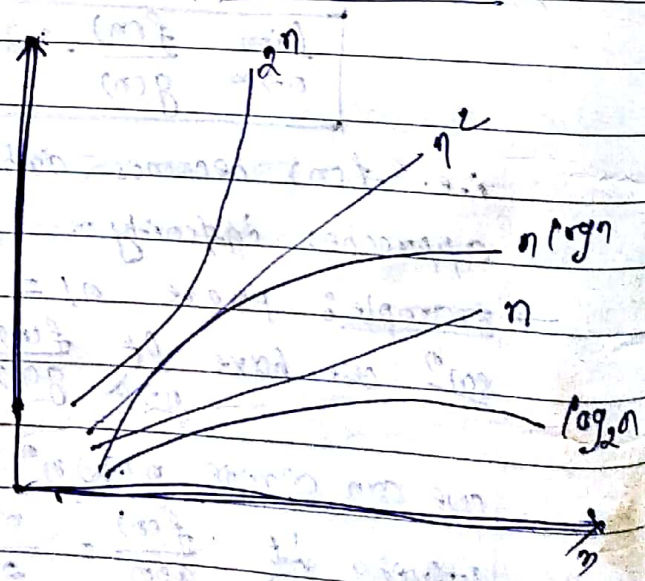
→ Running time of an algorithm can be following order.

$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^2) < O(n^3) < O(2^n)$

Function values:

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4,294,967,296

NOTE:
2^n grows very rapidly.



→ Asymptotic order of growth is:

$\log\log n < \log n < n^\epsilon < n < n \log n < n^2 < n^3 < \cdots < 2^n < n! < n^n$