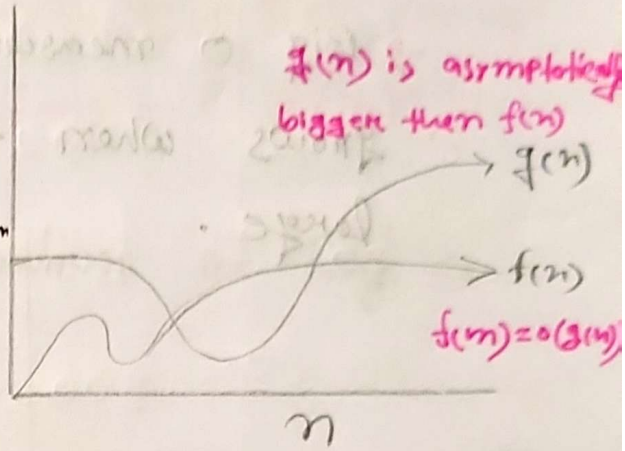
Growth Rate comparison of function

* Asymptotic Analysis

→ It is a mathematical way of describing how an algorithm performs as the input size (n) grows large.



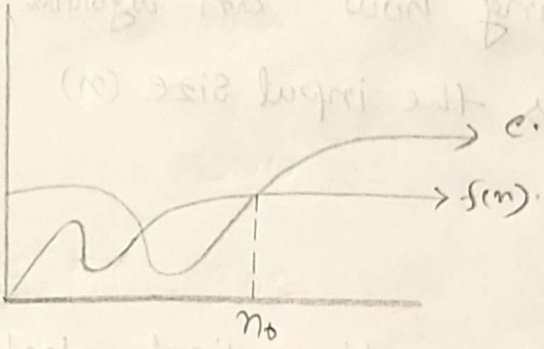
→ It is a mathematical tool to compare algorithms independent of Software, compilers, ~~and~~ or Programming language.

Main Asymptotic Notations:

Notation	Meaning	Bounds
Big O (O)	Upper Bound	Worst-case time. Algorithm won't be slower than this.
Big Ω (omega)	Lower Bound	Best-case time. Algo won't be faster than this.
Big Θ (Theta)	Tight Bound	Exact growth (both upper and lower match)

Big(O) Notation

→ Big O measures how fast a function grows when the input n becomes very large.



Assuming f(n) and g(n) are non-negative function.

Here,

f(n) = real ~~time~~ amount of time or steps that algorithm needs.

Non-Negative functions

g(n) = simple math function use to describe how fast f(n) grows.

■ $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constant.

$f(n) = O(g(n))$ → g(n) is tight upper bound of f(n).

c → Constant multiplier

n_0 → min value of n.

→ After certain point of n_0, the function f(n) will always be less than or equal to c times g(n).

Problem 1: Assume that $f(n) = 5n + 50$ and $g(n) = n$. Is $f(n) = O(g(n))$?

→ According to the defination of Big O Notation:

$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constants.

n	5n	50
5	25	50
10	50	50
20	100	50
40	200	50

(5n: dominating function)

n	2n	3n	n^2	$2n^2$
1	2	3	1	2
2	4	6	4	8
3	6	9	9	18
4	8	12	16	32
5	10	15	25	50

(2n: tight upper bound; 3n: upper bound)

- 5n এর সাথে অপেক্ষা বলছে Cn. 5n এর upper bound Cn

$5n \leq Cn \mid n^2 / cn^2$

Assuming, $C = 6$

n	$5n+50$	$6n$
10	100	60
20	150	120
50	300	300
51	305	306

$5n + 50 = O(n)$

for $C = 6$ and $n \geq 50$

Problem-2: Assume that $f(n) = 5n + 50$ and $g(n) = \log_{10} n$. Is $g(n)$ is upper bound of $f(n)$?

According to the ~~function of~~ defination of Big O notation $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n. where $n > n_0$ and c and n_0 are constant.

Assuming, $C = 1000$

	$5n + 50$	$1000 \log_{10} n$
$n = 10$	100	1000
$n = 10^2$	550	2000
$n = 10^3$	5050	3000
$n = 10^4$	50050	4000
$n = 10^5$	500050	5000

$g(n)$ is not upper bound of $f(n)$.

Problem-3: Find the upper bound of ~~f(n)~~

$f(n) = 3n + 8$

→ $3n$ is the dominant term

Assuming, $c = 4$

dominating function

n	$3n$	8
10	30	8
20	300	8
50	3000	8

n	$3n+8$	$4n$
7	29	28
8	32	32
9	35	36
100	308	400

n_0 min value of n. এই point এর পর $f(n)$ এর value বড়।

value সমান না হলেও কাছাকাছি যেকোনো value নিলেও চলবে

$3n + 8 \leq 4n$ is true.

$\therefore 3n + 8 = O(n)$ for $c = 4$ and $n_0 = 8$

Problem -4: Find the upper bound of $f(n) = n^2 + 10$.

→ n^2 is the dominant term.

$g(n) = n^2$ (because it's the dominant term)

n	n^2 ← dominant term	10
10	100	10
100	10000	10
1000	1000000	10

Assuming, $c = 2$

n	n^2+10	$2n^2$
1	11	2
2	14	8
3	19	18
4	26	32
5	35	50

n_0

$n^2 + 10 \leq 2n^2$ True

$\therefore n^2 + 10 = O(n^2)$ for $c = 2$ and $n_0 = 4$

Problem - 5: Find the upper bound of

$$f(n) = 2n^3 - 2n^2$$

$2n^3$ is the dominant term.

$g(n) = n^3$

Assuming, $c = 2$

n	n^3	n^2
1	1	1
2	8	4
3	27	9

(এখানে dominating function হচ্ছে $2n^2$ বিহীনা হচ্ছে।)

n	$2n^3 - 2n^2$	$2n^3$
1	0	2
2	8	16
3	36	54
4	96	128

n_0

$\therefore 2n^3 - 2n^2 = O(n^3)$ for $c = 2$

and $n_0 = 1$.

$2n^3 - 2n^2 \leq 2n^3$ is true.

Problem 6: find the upper bound of

$$f(n) = n^4 + 100n^2 + 35$$

n^4 is the dominant term

$g(n) = n^4$

n	n^4	n^2
3	81	9
4	256	16

Assuming,

$c = 2$

n	$n^4 + 100n^2 + 35$	$2n^4$
8	10531	8192
9	14696	13122
10	20035	20000
11 (n_0)	26776	29282
12	35171	41472

$\therefore n^4 + 100n^2 + 35 = O(n^4)$ for $c = 2$ and $n_0 = 11$

$n^4 + 100n^2 + 35 \leq 2n^4$ is true.

Problem 7: Find the upper bound of

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

→ 2^n is the dominant term.

$g(n) = 2^n$

n	2^n	n^3
1	2	1
2	4	8
3	8	27

Assuming,

$c = 2$

n	$2^n + 3n^3$	2^{n+1}
10	4024	2048
11	6041	4096
12	9280	8192
13 ← n_0	14783	16384
14	24616	32768

$\therefore 2^n + 3n^3 = O(2^n)$ for $c = 2$

and $n_0 = 13$

Problem 8: Find upper bound of $f(n) = 300$

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

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

$$300 \leq c \cdot g(n)$$

300 is the dominant term.

$g(n) = 1$

Assuming,

$c = 300$

$$300 \leq 300 \cdot 1$$

$$300 \leq 300$$

$\therefore 300 = O(1)$ for $c = 300$ and $n_0 = 1$.

Common Big O Runtimes

① O(log n) – Logarithmic Time

The runtime grows logarithmically with the input size. This mean that as the input size increases, the time taken increases very slowly.

→ It divide the Problem in half at each step.

Example: Binary Search.

② O(n) – Linear Time

The runtime grows linearly with the input size. If the input size is double then the time taken roughly doubles. This is typical for algorithms that need to look at each element of the input once.

→ Linear search. In linear search we check each element in a list one by one, untill we find the target or reach the end.

③ $O(n \log n)$

The runtime grows proportionally to n multiplied by the logarithm of n. This is a common complexity for many efficient sorting algorithms. It's better than quadratic time ($O(n^2)$) but not as good as linear time ($O(n)$).

Ex: Quicksort divides the array and recursively sorts sub-arrays.

④ $O(n^2)$ - Polynomial Time

→ The runtime grows quadratically with the input size. If you double the input size, the time taken increases by a factor of four.

→ If an algorithm uses loops where one loop is inside another (nested loop), and both loops look at the input items, the work often adds up like $O(n^2)$.

Example - Selection sort

$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = 0$

a_n → Co-efficient

Co-efficient শুধুমাত্র 0 হলেও (+, -) হবে।

x → variable, $a_0, a_1, \dots, a_n$ are efficient

n → is a non-negative integer.

$a_n \neq 0$

⑤ $O(n!)$ - Exponential time

→ The runtime grows factorially with the input size. This is a very rapid growth rate. Algorithms with this complexity are typically only practical for very small input size.

Example: Traveling Salesman.

* Linear Polynomial

$2x + 3 = 0$ (degree 1)

* Quadratic Polynomial

$x^2 - 5x + 6 = 0$

(degree-2) → x এর 2টি value
২টের বেশি নয়।

Visualization of Runtimes

n	$\log n$	n	$n \log n$	n^2	$n!$
10	1 1 ms	10 10 ms	10 10 ms	100 100 ms	3628800 6min
20	1.3 1.3 ms	20 20ms	26 26 ms	400 400ms	2.43290×10^{18} 7.7×10^{7} year
100	2 2 ms	100 100 ms	200 200 ms	1000 100ms	9.332622×10^{157} 2.958994×10^{147} Year

Decrement functions

Defination: A function in which the denominator is bigger than the numerator. (হর > লব)

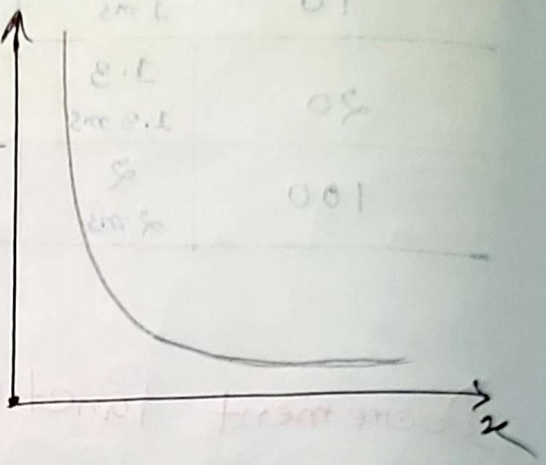
→ হর লাবের চেয়ে বড় হলে ভেগ্নাংশের মান ০ এর চেয়ে ছোট হয়।

When 'n' gets larger and larger

- The denominator (হর) get much, much bigger then numerator (লব).
- This makes the value of function get smaller
- The value of function approaches zero.

Example:

$\frac{c}{n}$ (c = Constant), $\frac{n}{n^2}$, $\frac{n}{2^n}$, $\frac{n^2}{3^n}$ ....



Graph of decrement function

যে Input (n) এর value বাড়ার সাথে সাথে ফলাফল ছোট হতে থাকে, এরকম ফাংশনের গ্রাফ বলতে বলতে শূন্যের কাছে চলে আসে।

Problem - 1 Arrange the following functions in the order of decrease from slowest to fastest:

$\frac{100}{n}$, $\frac{n}{2^n}$, $\frac{n}{n^2}$, $\frac{n^2}{3^n}$, $\frac{n^3}{3^n}$

Simplify the functions:

$\frac{100}{n}$, $\frac{n}{2^n}$, $\frac{1}{n}$, $\frac{n^2}{3^n}$, $\frac{n^3}{8^n}$

Compare the denominators and arrange

$$\frac{100}{n}, \frac{n}{2^n}, \frac{1}{n}, \frac{n^2}{3^n}, \frac{n^3}{3^n}$$

$n < 2^n < 3^n$

$$\frac{100}{n}, \frac{1}{n}, \frac{n}{2^n}, \frac{n^2}{3^n}, \frac{n^3}{3^n}$$

$$\frac{100}{n}, \frac{1}{n}, \frac{n}{2^n}, \frac{n^3}{3^n}, \frac{n^2}{3^n}$$

Slowest ⟶ fastest