Algorithms - 2.1

Asymptotic Notations

P1 Importance of Algorithm Analysis

Priori → not accurate but is the fastest & practical way to analyze an algorithm

Posteriori → accurate but dependent on
- program impl.

Topics

- ☐ Importance of Algorithm analysis
- ☐ Example - Finding the Key

Importance of Algorithm Analysis

- There are many algorithms to solve a problem
  - Out of that → we need to select the best candidate to solve the problem.

choose the most efficient ?

→ measured in terms of
- time
- memory space consumption

should be the least

- Example

1	23	34	46	65	78	90	101

Requirement

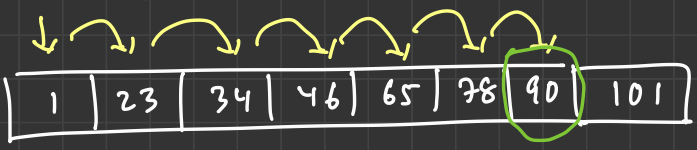
- Find Key = (90)

Assumption

- The key exists the the list
- The items are in ASC order

Approach - 1

p* = pointer



(1) p = 1

Is p == 90?

↓ No

(2) p = 23

Is p == 90?

↓ No

⋮

(7) p = 90

Is p == 90?

↓ YES

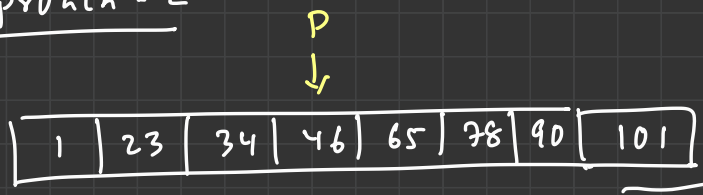
FOUND

Total comparison = 7

Is this the only approach?

NO!

Approach - 2



n = Total items = 8

(1) Find the middle of the list

$$\frac{n}{2} = \frac{8}{2} = 4$$

• Note: The list is in ASC order

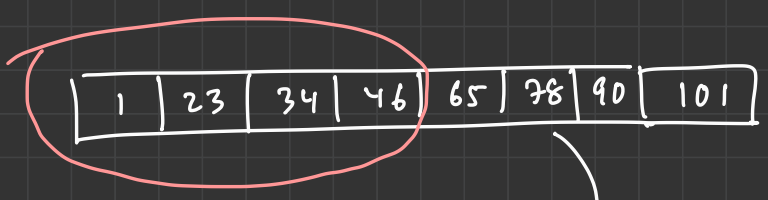
Comparison 1

(2) P = 46

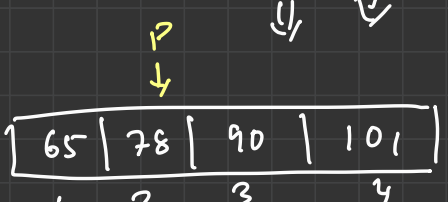
Is Key(90) > P(46)?

↓

YES



Eliminate this's

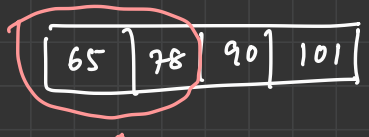


(2)

$n = 4 \Rightarrow P = \frac{n}{2} = 2$

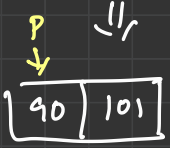
Is Key(90) > P(78)? => YES

comparison count: 2



↑ can be eliminated

(3)



$n = 2$ P => $\frac{n}{2} = 1$

Is Key(90) > P(90) ——> 3

↓

NO

We found the Key

[Binary Search]

Approach 1

7

[Linear search]

Approach 2

3. faster

- Although the improvement is just of 4 for $n = 8$
  - what if $n = 100$ or $n = 10^{10}$?

Linear search is slower than Binary search.

$n = 100$

∴ Linear search = 100 comparison.

∴ Binary search ⇒ $\log_2 100$

$$= \log_2 2^7 = 7 \text{ comparison.}$$

Note:-

- Binary search requires sorted data

P2

Importance of Growth rates

Topics

- [ ] What is growth rate? Why it is important?

Linear Search

- 100 items
  - worst case → 100 comparisons (100ms)

- 1 billion
  - → 1 billion comparisons (11 days)

Binary Search

- 100 items
  - → 7 comparisons (worst case) (7ms)
- For 100 items

Linear Search = 14 x Binary Search

Will it always be 14 times?

- 1 billion items
  - → 30 comparisons (30ms)

Linear Search = 31 million x Binary Search

* One example can't tell us → how fast an algo is / how efficient an algorithm is

- As the size of the i/p ↑ => Speed of binary search grows drastically.

- What is important to know

  ↳> Time complexity of an algorithm w.r.t i/p size

GROWTH RATE

* Rate at which the running time increases as a function of i/p is the growth rate.

* But how do we measure the growth rate?

  * Big O Notation

P3

Big O Notation

- way to describe the performance of an algo-rithm w.r.t i/p size

Performance of an algorithm

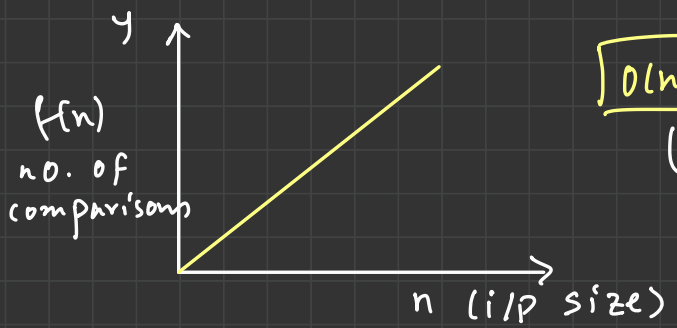
=> How fast (time)

=> How much memory (memory space)

Linear Search

* The no. of comparisons $\propto$ i/p size

n =	1	2	10	100	
f(n) =	1	2	10	100	—> no. of comparisons



O(n) —> Running Time.

↳ growth rate of Linear search

Running Time = O(n)

—> We have not mentioned it in seconds.

(Why is that so?)

Big O —> helps in comparing the number of operations w.r.t i/p size

* Recall:

. Algorithm is not h/w dependent

O(n)
↳ i/p

Growth Rate (no. of operations reqd by the algorithm w.r.t n)

P3

Big O Notation & Binary Search

Topics

- Big O Notation and binary search
- Growth rate of Binary Search
- Growth rate comparison → Linear Search & Binary Search

Big O Notation & Binary Search

- In binary search, every time half of the list is eliminated
  - → middle of the list
  - → compare that with the key.

Number of items (n)	Number of comparisons O(n)
10	$\log_2 10 \simeq 3.32 \simeq 4$
100	$\log_2 100 \simeq 6.643 \simeq 7$
1000	$\log_2 1000 \simeq 9.96 \simeq 10$
1000000	$\log_2 1000000 \simeq 20$
1 Billion	$\log_2 1B \simeq 30$
n	$\log_2 n$

(* refer to the next page)

items

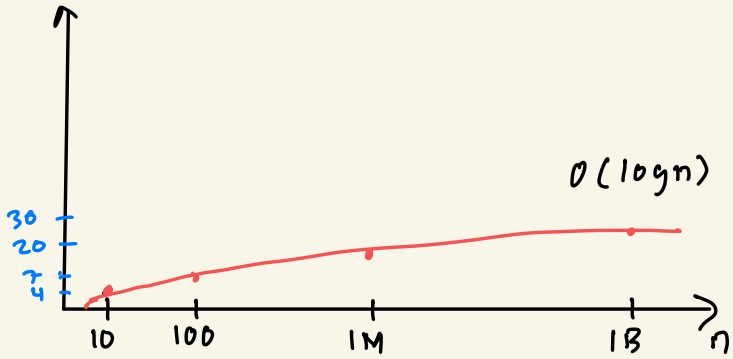
$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{2^2} \rightarrow \cdots \frac{n}{2^k}$

Total no. of boxes = $k+1$ = Total no. of comparisons.

$\frac{n}{2^k} = 1$

$\Rightarrow 2^k = n \quad \Rightarrow \boxed{k = \log_2 n}$ → we are dropping the 1 for our simplification.

Running time = $O(\log n)$

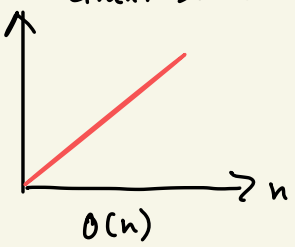


* As the size of the i/p ↑ => the running time does not increase drastically.

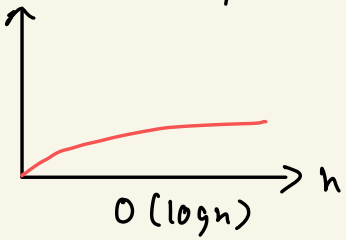
∴ Binary Search => Efficient.

Growth Rate comparison

Linear Search



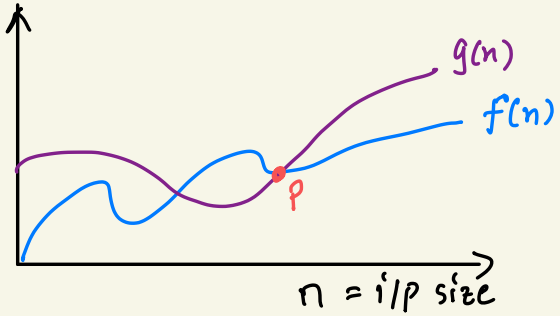
Binary search.



P4 :- Formal Definition of Big O

Topics

- Growth rate comparison of functions → f1, f2
- Definition -> Big O



Q:- Is $f(n) > g(n)$ or $g(n) > f(n)$??

After point P, $g(n) > f(n)$ → g(n) is asymptotically bigger than f(n).

But before point P, there is no guarantee

- At times $g(n) < f(n)$
- Other times $g(n) \geq f(n)$

- As $n \to \infty$, $g(n) > f(n)$

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

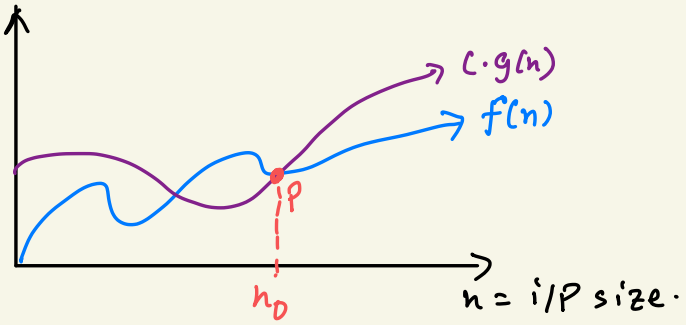
Because f(n) will never be able to grow faster than g(n) after point P.

Formal Definition of Big O

- Assume $f(n)$ & $g(n)$ are non-negative functions

  Why? $f(n)$ or $g(n)$ will represent
  - Time
  - Memory Space.

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



Note

. $g(n)$ = Tight upper bound of $f(n)$

. $f(n)$ can't grow bigger than $g(n)$ after n_0.

Big O Notation - Problem Set 1

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

$c \cdot g(n) = c \cdot n \rightarrow g(n) = c \cdot n$ (let's take $c = 6$)

	$f(n) = 5n+50$	$6n$	Is $6n \geq 5n+50$?
$n = 10$	$5 \cdot 10 + 50 = 100$	60	No.
$n = 20$	$5 \cdot 20 + 50 = 150$	120	No
$n = 50$	$5 \cdot 50 + 50 = 300$	300	yes
$n = 51$	$5 \cdot 51 + 50 = 305$	306	yes
	⋮		⋮

$n_0 = 50$ beyond which $c \cdot g(n) \geq f(n)$.

- $\begin{bmatrix} n_0 \geq 50 \\ c = 6 \end{bmatrix}$ $g(n)$ is asymptotically bigger than $f(n)$. $c \cdot g(n) \geq f(n)$.

- $g(n)$ = tight upper bound of $f(n)$. Why?

  ↑

  Least = closest to $5n+50$ / $f(n)$

  - There can be many upper bound of $f(n)$
    - we could take $g(n^2)$ or $g(2^n)$ or so on.
    - But we took $\boxed{c \cdot g(n)}$

$c = 6$ & $n \geq n_0$ where $n_0 = 50$

- So for $c = 6$, $n_0 = 50$, $5n + 50$ cannot grow more than "n".

Problem 2: Assume that $f(n) = 5n + 50$ and $g(n) = \log_{10} n$

Is $g(n)$ upper bound of $f(n)$?

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

For the above to be true

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

Assume $c = 1000$

n	$5n + 50$	$1000 \log_{10} n$	$c \cdot g(n) \geq f(n)$?
10	100	$1000 \times \log_{10} 10 = 1000$	yes
10^2	550	$1000 \times \log_{10} 10^2 = 2000$	yes
10^3	5050	$1000 \times \log_{10} 10^3 = 3000$	No.
10^4	50050	$1000 \times 4 = 4000$	No.

∴ Conclusion

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

Problem 3

Find the upper bound for $f(n) = 3n + 8$.

* Here $g(n)$ is not available to us
* How do we find $g(n)$?

Steps

(1) Find the dominant term.

$3n$.

As the value of n increases, 8 will be irrelevant.

(2) Choose $g(n)$ according to the dominant term.

Options

- n -> nearest
- $n \log n$
- n^2
- n^3

$\therefore g(n) = n$

Step 3: Apply the Big O Definition.

$c = 4$

Is $3n + 8 \leq 4n$

n	$3n+8$	$4n$	Is $3n+8 \leq 4n$
1	$3\times1+8=11$	4	NO

$4n \geq 3n+8 \Rightarrow n \geq 8$

7	$3\times7+8=29$	28	NO
8	$3\times8+8=32$	32	yes
9	$3\times9+8=35$	36	yes
:	:	:	:

$\therefore f(n) \leq O(g(n))$ for $c=4$ $n_0=8$ for all $n \geq n_0$

$\therefore$ The upper bound $= 4n$

Problem 4

Find the upper bound for $f(n)=n^2+10$

Step 1 :- Find the dominant term $= n^2$

Step 2 :- options for $g(n)$

- n^2 ---> nearest
- $n^2 \log n$
- n^3

..

$\therefore g(n)=n^2$

Step3 :- $c = 2$

$\therefore c\, g(n) = 2n^2$

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

$\Rightarrow 2n^2 \geq n^2 + 10$

$\Rightarrow n^2 \geq 10$

$\Rightarrow n \geq \sqrt{10} = 3.16 \ldots 4$

n	$f(n) = n^2 + 10$	$c \cdot g(n) = 2n^2$	$c \cdot g(n) \geq f(n)$?
1	11	2	No
4	26	32	yes

Upper bound : $2n^2$

$f(n) \leq c \cdot g(n)$ or $f(n) = O(g(n))$

$c = 2 \quad n_0 = 4$

Problem 5

Find the upper bound for $f(n) = 2n^3 - 2n^2$

step1 :- The dominant term = $2n^3$.

step2 :- Options for $g(n)$
- n^3 → nearest
- $n^3 \log n$
- n^4
- :

Let $c = 2$ $\quad c \cdot g(n) = 2n^3$ $[c = 2]$

n	$f(n) = 2n^3 - 2n^2$	$c \cdot g(n) = 2n^3$	Is $c \cdot g(n) \geq f(n)$?
1	0	2	yes
2	$2 \times 8 - 2 \times 4 = 8$	16	yes

$\therefore$ $2n^3 - 2n^2 = O(n^3)$ for $c = 2$ and $n_0 = 1$

Problem 6

Find the upper bound for $f(n) = n^4 + 100n^2 + 35$

$g(n) = n^4 \quad c = 2$.

$\therefore$ $c \cdot g(n) = 2n^4$

n	n^4+100n^2+35	$2n^4$	Is $c \cdot g(n) \geq f(n)$?
1	136	2	NO
2	451	32	NO
11	26776	29282	yes

Upper bound for $n^4+100n^2+35 = O(n^4)$
where $c=2$ $n_0 = 11$

Problem-7

Find the upper bound of $f(n) = 2^n + 3n^3$

Step 1 :- Find the dominant term.
2^n , n^3 $\longrightarrow$ 2^n

n	2^n	n^3
0	1	0
3	8	27
:		
10	1024	1000
:	:	:
20	1048576	8000

Step 2 :- Assume $g(n) = 2^n$.
$c \cdot g(n) = c \cdot 2^n$ may surpass $2^n + 3n^3$

Let $c = 2$

n	$f(n) = 2^n + 3n^3$	$c \cdot g(n) = 2 \cdot 2^n$
1	5	4
⋮		
10	4024	2048
⋮		
13	14783	16384

$g(n) = 2^n$

$f(n) = 2^n + 3n^3 = O(2^n)$ where $c = 2$ $n_0 = 13$

Note: We could have checked $n = 14$, and still conclude

$2^n + 3n^3 = O(2^n)$ for $c = 2$ and $n_0 = 14$

Problem - 8

Find the upper bound for $f(n) = 300$

Step1 : Identify the dominant term $= 300$

Step2 : $g(n) = 1$ $c = 300$

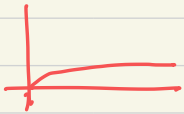
$f(n) \leq c \cdot g(n)$

$c = 300$ $n_0 = 1$ for which

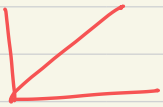
$f(n) = O(1)$ for $c = 300$, $n_0 = 1$

Common Runtimes - Big O

- **O (log n)** → also known as log time.
  Ex:- Binary Search.
  This is one of the fastest algorithm known.



- **O(n)** → Linear time
  Ex → Linear Search
  As the size of the i/p increases, the growth rate increases linearly



- **O (n log n)** → Combine the above two to get this
  Ex → Quick sort

- **O(n^2)** → Known as Polynomial time.
  Ex → Selection Sort

- **O(n!)** → exponential time
  Ex → Traveling Salesman (Undecidable)
  It is a problem. Not an algorithm.

Visualize Runtimes

These are no. of operations

n	log n	n	n log n	n^2	n !
10	1	10	10	100	362800
20	1.3	20	26	400	2.43×10^{18}
100	2	100	200	10000	9.33×10^{157}

Functions in Asymptotic Notations

(1) Decrement Functions

- If $x_2 > x_1$ and $f(x_2) < f(x_1)$ then $f(x)$ is a decrement function.
- e.g:- $f(x) = -x$

x	1	2	3
$f(x)$	-1	-2	-3

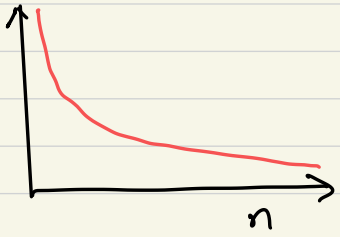
- e.g:- $\frac{c}{n}$

Here the numerator < denominator.

c = constant n = i/p.

As $n \to \infty$, c = constant has no relevance

- eg:- $\frac{n}{n^2}$ → This is also decrement function
- eg:- $\frac{n}{2^n}$ → Decrement function
- $\frac{n^2}{3^n}$ → " "



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

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

Step 1 : Arrange the fractions

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

Step 2 : Compare the denominators & arrange.

- function that has the least denominator
  ↳ slowest decreasing function
- function that has the greatest denominator
  ↳ fastest decreasing function

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

↑ slowest decreasing function

↑ fastest decreasing function

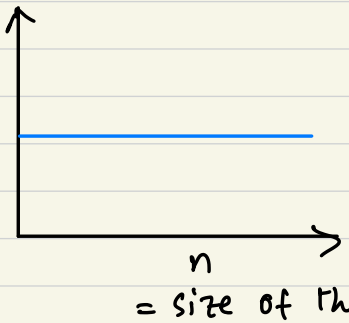
$$n < 2^n < 3^n$$

(2) Constant Functions

- Constant Functions
- Compare with Decrement Functions.

Constant Function => A function parallel to x-axis's

$f(x) = c$



$f(n) = c$

* The function does not depend on the size of the i/p.

- A constant function is asymptotically bigger than decrement function.

ex:- $f(n) = \frac{100}{n}$ $g(n) = 10000$ Is $f(n) = O(g(n))$.

$f(n) \leq c \cdot g(n)$ for $n \geq n_0$

Let's try to make $g(n)$ as small as possible

$c = \frac{1}{10000}$ $\therefore$ $c \cdot g(n) = \frac{1}{10000} \times 10000 = 1$

n	$f(n) = \frac{100}{n}$	$c \cdot g(n)$	$c \cdot g(n) \geq f(n)$
1	100	1	No
100	1	1	yes
1000	0.1	1	yes

for $c = \frac{1}{10^4}$ $\quad g(n) = 10000$

$$f(n) = \frac{100}{n} = O(10000)$$

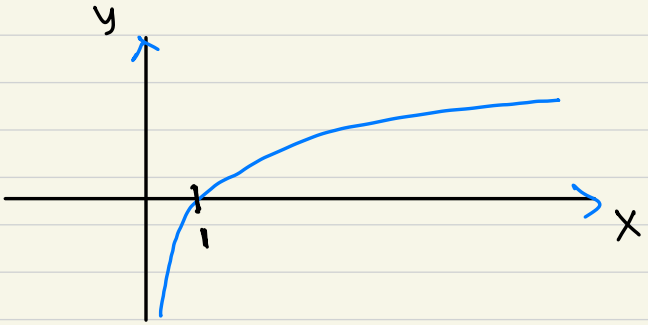
where $n_0 = 100$

∴ Constant function is asymptotically bigger than decrement function.

(3) Logarithmic Functions

$f(x) = \log_{10} x$

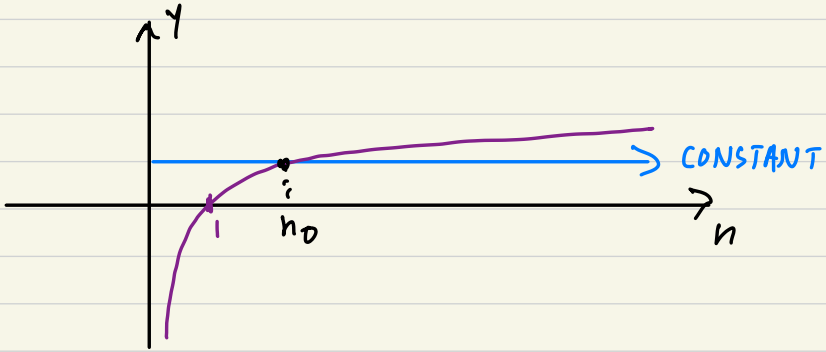
↳ grows positively but with slower ($b > 1$) growth rate



Ex:- $\log n$, $(\log n)^{10}$, $\log \log n$,

$f(n) = \log n$

compare $\log n$ with constant functions



↳ Since $\log(n)$ is increasing, beyond n_0 it will grow asymptotically bigger than $f(n) = c$

e.g:- $f(n) = 100$ $g(n) = \log_{10} n$ Is $f(n) = O(g(n))$?

Let $c = 100$

n	$f(n)$	$c \cdot g(n)$	Is $c \cdot g(n) \geq f(n)$?
1	100	0	NO
10	100	100	yes

So if $c = 100$ for $n_0 = 10$, $c \cdot g(n) \geq f(n)$

or $f(n) = 100 = O(g(n))$ where $c = 100$, $n_0 = 10$

$\therefore$ $\boxed{f(n) = O(g(n))}$

If we had $c = 1$

n	$f(n)$	$c \cdot g(n)$	$c \cdot g(n) \geq f(n)$?
10	100	1	No
$n_0 \rightarrow 10^{100}$	100	100	yes
10^{1000}	100	1000	yes

$\therefore f(n) = O(g(n))$ for $c = 1$ $n_0 = 10^{100}$

$\log n > f(n) = c >$ Decreasing fn ()

---

(4) Polynomial function

- Polynomial function
- Compare Polynomial function with logarithmic functions.

Def :- A function whose growth rate increases polynomially.

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Ex :- $n^{0.1}$, $\sqrt{n}$, n^2, $n \log n$, n^K etc.

↳ Here $K > 0$

- Growth rate => increases polynomially.
- polynomial function is asymptotically bigger than logarithmic function.

ex:- $f(n) = \log_{10} n$ and $g(n) = \sqrt{n}$ Is $f(n) = O(g(n))$

Assume $c = 1$

	n	$\log_{10} n$	$\sqrt{n}$	$c \cdot g(n) \geq f(n)$
$n_0 \rightarrow$	2	0.30	1.414	yes

$f(n) = O(g(n))$ for $c = 1$ $n_0 = 2$

O comparison of polynomial vs logarithmic

- Polynomial functions are asymptotically bigger than logarithmic function.

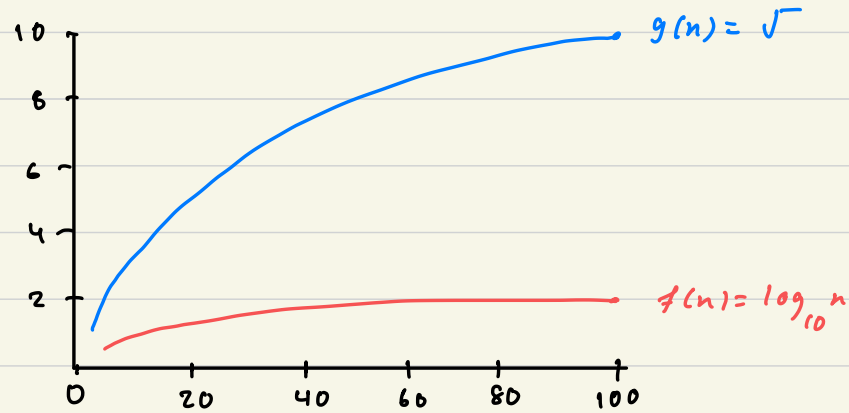
ex:- $f(n) = \log_{10} n$ and $g(n) = \sqrt{n}$. Is $f(n) = O(g(n))$?

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

Assume $c = 1$

n	$f(n) = \log_{10} n$	$c \cdot g(n) = n^{\frac{1}{2}}$	$f(n) \leq c \cdot g(n)$?
10	1	$\sqrt{10} = 3.16$	yes
100	2	10	yes
1000	3	31.62	yes

$\therefore f(n) = \log_{10} n = O(g(n))$ where $c = 1$ and $n_0 = 10$



Polynomial fns > logarithmic fns > Constant fns > Decrement fns

(5) Exponential functions

Def: A function whose growth rate increases rapidly.

ex:- 2^n, 3^n, $n!$, n^n, a^n etc

$n!$ = super exponential

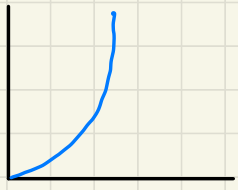
exponential function = a^n

$3! = 3 \times 2 \times 1 = 6$

At $n = 5$ $\quad 2^5 = 32 \quad 5! = 120$

(fixed base)

- Factorial growth rate >> fixed base exponential function.



$f(n) = a^n$

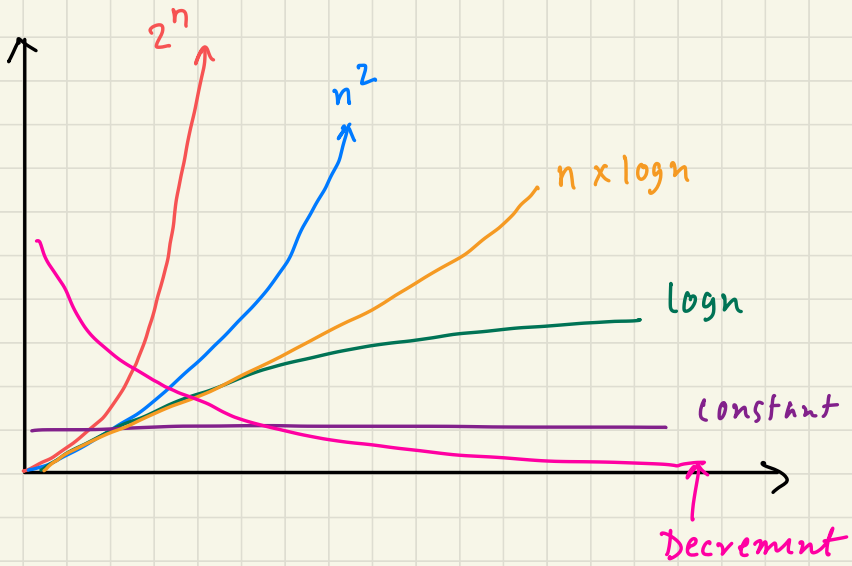
ex:- $f(n) = n^2$ and $g(n) = 2^n$ Is $f(n) = O(g(n))$?

Assume $c = 1$

n	$f(n) = n^2$	$c \cdot g(n) = 2^n$	$c \cdot g(n) \geq f(n)$
1	1	2	yes
10	100	1024	yes

Summary

Exponential fns > Polynomial fns > logarithmic fns > Constant fns > Decrement fns



- Asymptotic : $n \to \infty$

VI6 **Problems**

1) Write the following functions in asymptotically increasing order

n^3, $n^{3.1}$, $n^2 \log n$, $\log n$, $(\log n)^{10}$, 100, 1 billion, $(0.3)^n$, $(1.5)^n$

Categories

1. Exponential = $(1.5)^n$
2. a. Polynomial = $n^{3.1}$, n^3
   b. Poly-Logarithmic = $n^2 \log n$ → Dominating term
      ↳ That makes it Polynomial
3. Logarithmic = $(\log n)^{10}$, $\log n$
4. Constant = 1 billion, 100
5. Decrement = $(0.3)^n$ $\left[\left(\frac{3}{10}\right)^n\right]$

- $(1.5)^n > n^{3.1} > n^3 > n^2 \log n > (\log n)^{10} > \log n > 1 \text{ billion} > 100 > (0.3)^n$.

(2) Which of the following function is asymptotically bigger?

$f(n) = \log_{10}(n)$ $\quad g(n) = \log_{10}(\log_{10} n)$

n	$\log_{10} n$	$\log_{10} \log_{10} n$
10^{10}	10	1
10^{100}	100	2
$10^{10^{10}}$	10^{10}	10

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

(3) Which of the following function is asymptotically bigger

$f(n) = \log_{10} n$ & $g(n) = (\log_{10} \log_{10} n)^{10}$

n	$f(n) = \log_{10} n$	$g(n) = (\log_{10} \log_{10} n)^{10}$
10	1	0
10^{10}	10	1

$g(n) = O(f(n))$ $\quad f(n)$ is bigger.

Big Omega Notation (Ω)

- Lower bound of an algorithm.

Big O → tight upper Bound

Big ω → lower bound

Let $f(n)$, $g(n)$ are two functions. If

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

$\Rightarrow$ $g(n)$ is asymptotically bigger than $f(n)$.

$f(n)$ → upper bound : $g(n)$ = worst case run time

$f(n)$ → lower bound : $\Omega(n)$ = Best case run time.

$\Rightarrow$ $f(n)$ cannot grow lesser than $\Omega(n)$

Def :- Assume $f(n)$ and $g(n)$ are non-negative fns.

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

iff $f(n) \geq c \cdot g(n)$ for some $c > 0$

and for all values of $n \geq n_0$.

c & n_0 are constants.

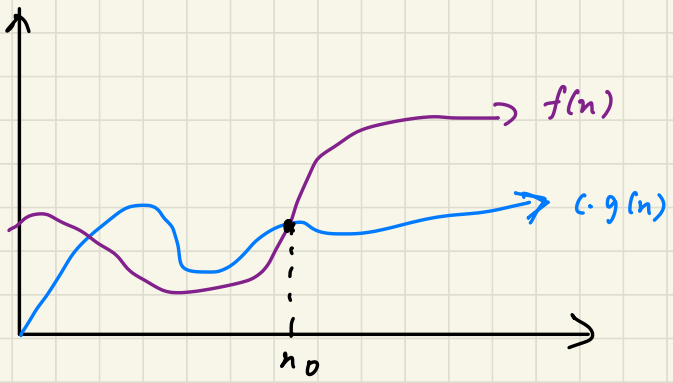
Note $\Omega(n)$

- can be used for run time
- memory consumption

but we are not focussed on memory consumption

tight upper bound = closest upper bound ($O(n)$)

tight lower bound ($\Omega(n)$) => closest lower bound



After some n_0, $f(n)$ will grow more than $c \cdot g(n)$

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

NOTES

WILL COME HERE FROM SCANS

Asymptotic Notation problems

(3) Let $f(n)$, $g(n)$, $h(n)$ be functions defined for +ve integers. such that

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

from these 2 conditions, we can say $g(n) > f(n)$

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

$g(n) = f(n)$ asymptotically

$g(n) = O(h(n))$
$h(n) = O(g(n))$

→ for both to satisfy $g(n) = f(n)$

Which one of the following statements is false?

(A) $f(n) + g(n) = O(h(n) + h(n))$

(B) $f(n) = O(h(n))$

(C) $h(n) \neq O(f(n))$ [GATE IT 2004]

(D) $f(n) \cdot h(n) \neq O(g(n)\, h(n))$

Let $f(n) = n$ $g(n) = n^2$ $h(n) = n^2$

has to be same asymptotically in order to satisfy

$g(n) = O(h(n))$
$h(n) = O(g(n))$

Now let's take (A) : Option

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

This statement is not false

$\Rightarrow n + n^2 = O(n^2 + n^2)$ → Can we say this? ✔ TRUE

(B) $f(n) = O(h(n))$ => TRUE

$n = O(n^2)$ -> This is also true ✓

(C) $h(n) \neq O(f(n))$

$n^2 \neq O(n)$ => TRUE

(D) $f(n) \cdot h(n) \neq O(g(n) \cdot h(n))$

$n \cdot n^2 \neq O(n^2 \cdot n^2)$ => FALSE

∴ Option D is False -> And this is the correct option.

---

(4) Consider the following functions

$f(n) = 2^n$ $g(n) = n!$ $h(n) = n^{\log n}$

Which one of the below is true?

(A) $f(n) = O(g(n))$; $g(n) = O(h(n))$

(B) $f(n) = \Omega(g(n))$; $g(n) = O(h(n))$

(C) $g(n) = O(f(n))$; $h(n) = O(f(n))$

(D) $h(n) = O(f(n))$; $g(n) = \Omega(f(n))$ <GATE 2011>

Let's take option (A)

2^n = Exponential

$n!$ = Exponential

$n^{\log n}$ = Exponential

Let's compare $f(n)$ and $h(n)$

$f(n) = 2^n$ and $h(n) = n^{\log n}$

If we compare between 2^n & 3^n => we know which one is bigger.

. But how to compare between 2^n and $n^{\log n}$

↓

conflicting functions

Let's apply logarithm

$f(n)$

$\log(2^n)$

$= n \log_2(2)$

$\Rightarrow \underbrace{n}_{\text{Poly}}$

$h(n)$

$\log(n^{\log n})$

$\Rightarrow (\log_2 n)^2$

$\underbrace{(\log_2 n)^2}_{\text{Log}}$

$\therefore$