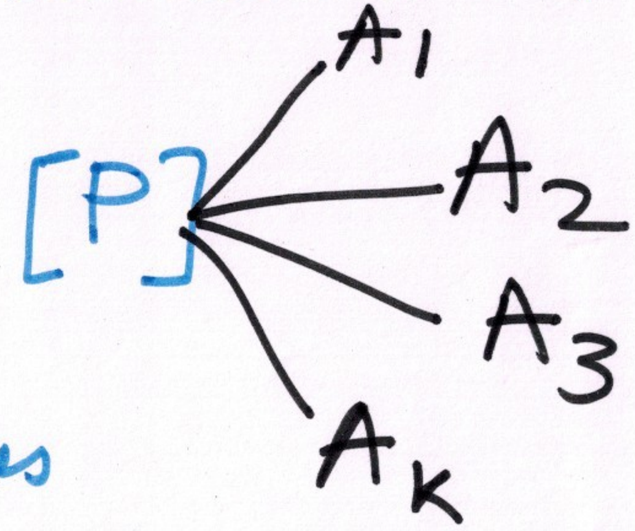
Given an problem [P], one can write/ design multiple algorithms.



Each one will run over some machine and consume resources that are computed in terms of

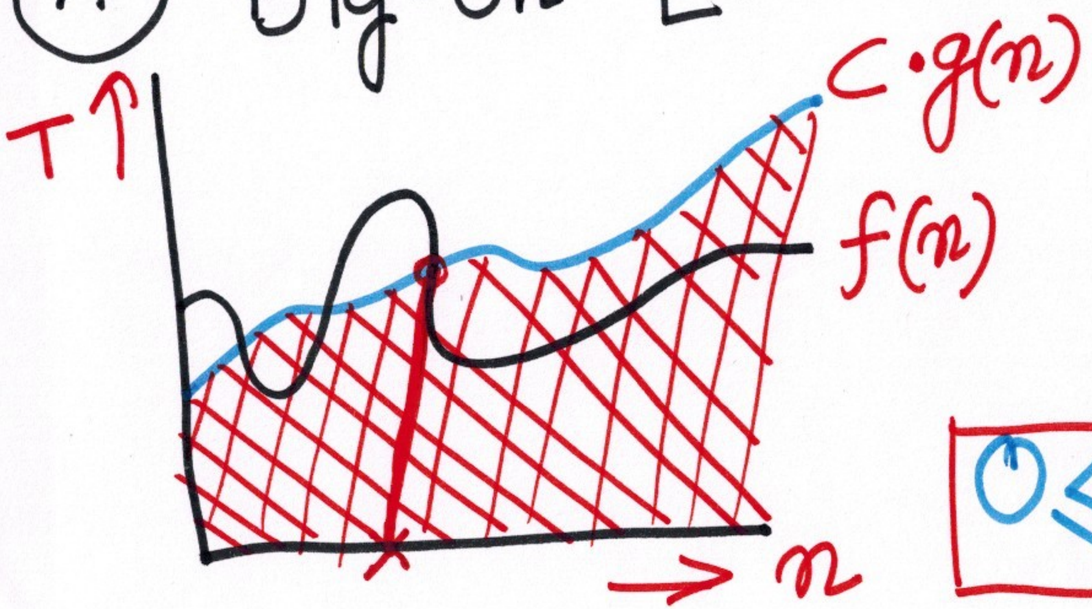
a) Time b) Space

Most of time we are interested in asymptotic efficiency of algorithms. (Using Asymptotic Notations)

* We are interested to analyse how running time of an algorithm increases with input size.

* Asymptotically efficient algorithms are the best choice for all but very small i/pts.

(A) Big-Oh [O- Notation] $\Rightarrow f(n) = O(g(n))$



$O(g(n)) = \{ f(n)$: there exist positive Constants c and n_0 s.t

$0 \leq f(n) \leq c\, g(n) \quad \forall \quad n \geq n_0$

Most of the time Big oh notation is used for worst Case analysis. (As it is upper bound.)

Many time upper bound and lower bound are easily Computed by observing the algorithm.

Insertion Sort → Best Case
→ Worst Case

$O(g(n))$ ⇒ Actually a class of functions that are upper bounded by $g(n)$ for some Computable c & n_0.

Given any $f(n)$ → we have to find such $g(n)$

Example : $f(n) = 3n+2 \qquad g(n) = n$

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

then $3n+2 \leq cn \Longrightarrow$ Anything greater than 4

$c = 4$

Hence $3n+2 \leq 4n \Longrightarrow n \geq 2$

Hence $\boxed{c = 4 \; ; \; n_0 = 2}$

If (n) is bounding this $f(n)$ definitely

$g(n) = n^2, n^3 \text{---}$ will also bound it.

Or — 2^n.

Similarly:

$100n + 6 = O(n)$

↳ $100n + 6 \leq 101n \quad \forall \quad n \geq 10$

$10n^2 + 4n + 2 = O(n^2)$

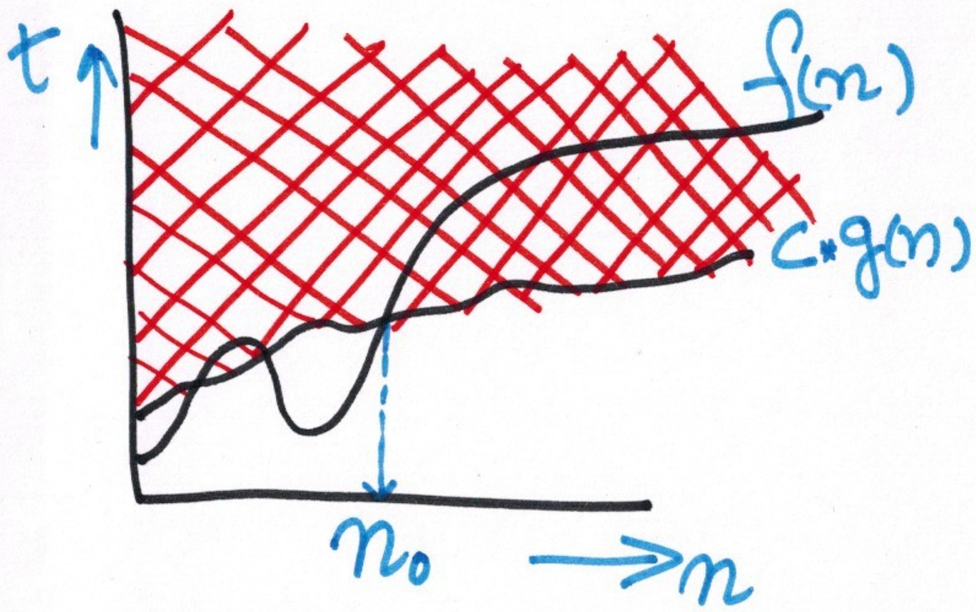
↳ $10n^2 + 4n + 2 \leq 11n^2 \quad \forall \quad n \geq 5.$

$10n^2 + 4n + 2 \neq O(n)$

↳ we cannot find suitable c & n_0

(B) Ω notation: It provides asymptotic lower bound.

Generally used for best case analysis.



$\Omega(g(n)) = \{ f(n) :$ there exist +ve Constants c and n_0 s.t $0 \leq f(n) \leq c\, g(n) \}$ $\forall \cdot n \geq n_0$

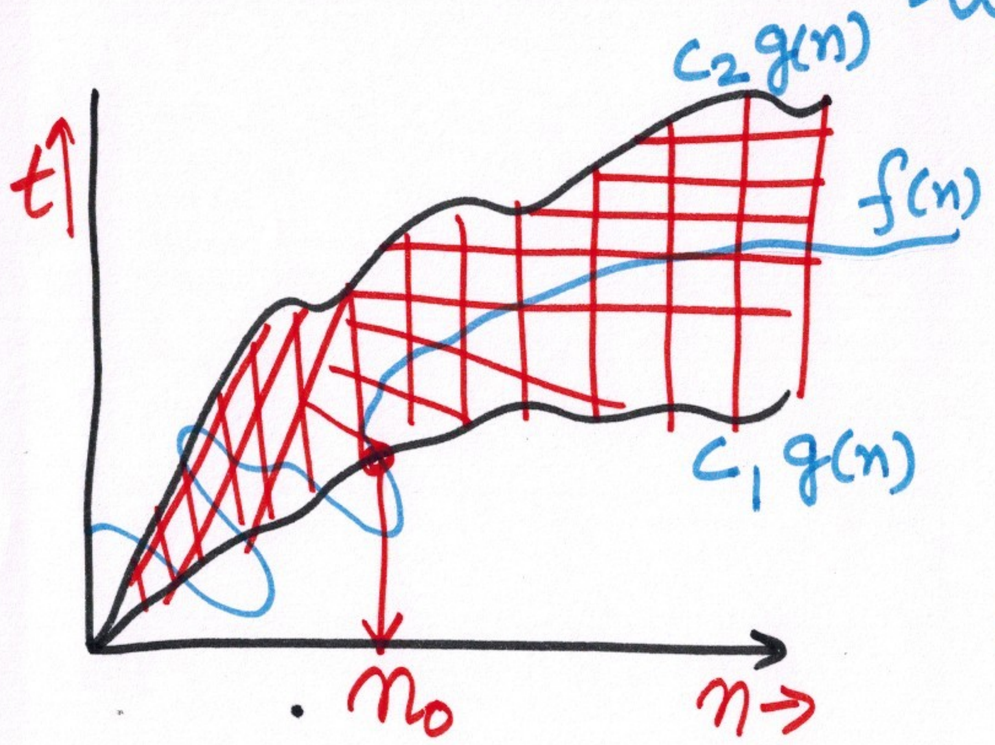
ex: $3n+2 = \Omega(n)$

$\hookrightarrow 3n+2 \geq 3n \quad \forall\ n \geq 1$

$c > 0$
$n_0 > 0$

This is even true for $\underline{n=0}$

(c) Theta notation [θ]: It defines the exact and tight asymptotic behaviour.



$\theta(g(n)) = \{ f(n)$: there exist +ve constants c_1 and c_2 and n_0 s.t

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall \; n \geq n_0$$

ex: $3n+2 = \theta(n)$

$\hookrightarrow 3n+2 \geq 3n \; \forall \; n \geq 2$ & $3n+2 \leq 4n \; \forall \; n \geq 2$

$c_1 = 3$, $c_2 = 4$, and $n_0 = 2$

$6 \cdot 2^n + n^2 = \theta(2^n)$ & $10 * \log(n) + 4 = \theta(\log(n))$

Prove: $f(n) = a_m n^m + a_{m-1} n^{m-1} + -- \; a_1 n + a_0$

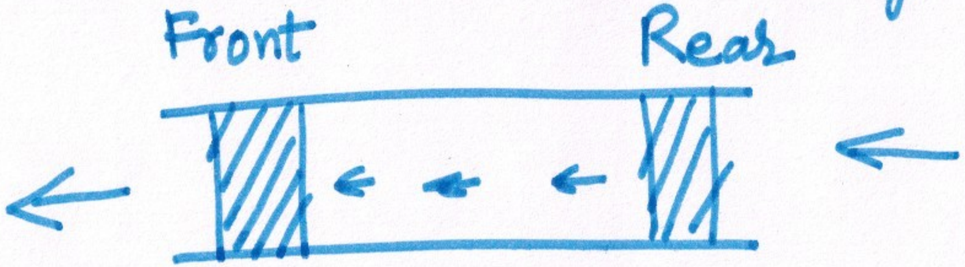
then $f(n) = O(n^m)$

$f(n) = a_m n^m + \text{——} \; a_0$ and $a_m > 0$

then $f(n) = \theta(n^m)$.

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32,768	4,294,967,296

QUEUES : Linear list of elements in which deletions can take place only at front and insertions can take place only at rear. [FIFO]



Front = Front + 1

Rear = Rear + 1

⇓

This implies that after [N] insertions the rear element of queue will occupy QUEUE[N] i.e, queue occupy last element, even though queue is Empty.

F 0
R 0

1 2 3 4 5

Empty

F 1
R 3

A B C

A, B, C
(In)

F 2
R 3

B C

A
(Del)

F 2
R 5

B C D E

D, E
(In)

⇒ (R=N)

F 4
R 5

D E

B, C
(Del)

F 4
R 1

F D E

F
(In)

F 5
R 1

F E

D
(Del)

10

(F=N) ⇐

F 5
R 3

F G H E

G, H
(In)

F 1
R 3

F G H

E
(del)

F 2
R 3

G H

F
(Del)

F 2
R 4

G H K

K
(~~Del~~)
In

F 4
R 4

K

G, H
(Del)

Empty ⇐

F 0
R 0

K
(Del)

(A) Front = NULL ⇒ Our queue is empty

(B) Front = Rear = NULL ⇒ First element is going to be inserted, Queue is empty.

Special handle after (N), In/del

(C) If Rear = N ⇒ Queue may be full. Assuming its a Circular queue and Rear ++ → Rear = 1 (reset)

(D) If Front = N ⇒ Reset Front to 1 ⇒ Front = 1

(E) Front = Rear ≠ NULL ⇒ Queue have only one element.

↳ And if deleted then both have to be reset

Front = Rear = Null.

No. of element in Queue: [Rear - Front + 1]

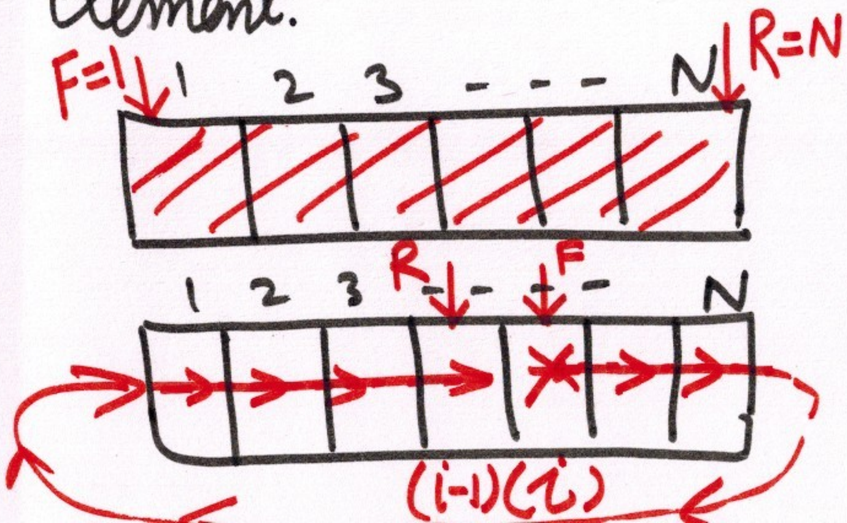
Rear - Front + 1 + N

Overflow and Underflow.

ⓐ Front = 1 and Rear = N

ⓑ Front = Rear + 1

Queue is full With (N) elements.



Underflow. → Front = Null

Front ++ never make it to point Null.

It is handled specially

→ If may be first element. inserted

→ Last element deleted (F = R ≠ Null) only one element.

QINSERT(Q, N, Front, Rear, Item)

1. Queue is full:
   If Front=1 & Rear=N or Front=Rear+1, then:
   Write - Overflow and Return.
2. Find new value of Rear:
   If Front = NULL then [Queue is empty]
   Set Front = 1 and Rear = 1.
   Else if Rear = N, then [After N insertion for circular]
   Set Rear = 1
   Else
   Set Rear = Rear + 1
   End IF.
3. Set Queue[Rear] = Item [Inserting new element]
4. Return

Q Delete (Q , N, Front Rear, Item) Return deleted Item.

1. Queue already empty

   If Front = NULL, then Write : Underflow and Return

2. Set Item = Q [Front]

3. Find new value of Front.

   If Front = Rear, then [Queue has only one element

   Set Front = NULL and Rear = Null

   Else if Front = N, then

   Set Front = 1

   Else

   Set front = Front + 1

   End of If.

4. Return Item.