• Pseudo Code

a) Expressions : Standard mathematical symbol to express numeric & boolean expressions.
   i) we use '←' sign as the '=' operator in c/c++/java.
   ii) we use '=' sign as the '==' operation in c/c++/java

b) Method Declarations : name (parameter1, parameter2, ...) declares a new method "name" & its parameters.
   → we can call & define the method in the pseudocode as we do in c/c++/java

c) Decision structure :
```
if (<Condition>) then
  { action }
else
  { action }.
```
] Just like c/c++/java

d) while loops :
```
while (<condition>) then
  do { action }.
```

e) Repeat loops :
```
repeat
  { action }
until (<condition>)
```

f) for loops :
```
for ( variable initialization, condition, increment ).
  { action }.
```

g) Array Indexing : A[i] represents the 'i'th cell in the array 'A'.

h) Method Calls : object.name ("xyz", 'pqr'). [object is optional].

i) Method Returns : return value. Just like c/c++/java.

RAM MODEL

(Random Acess Machine Model).

Program → P → A_1, A_2, A_3 } Algorithm.

↓

Some/one is efficient than others.

↓

executes on Computers.

Slower ← $i_3 < i_5 < i_7$ → faster

i_3 is slower than i_5 & i_5 is faster than i_3.

Different processor (gives different output)/takes different time to execute some Algorithm/program.

↓

Random Access Machine (RAM) Model.

↓

Hypothetical Machine

→ not concerned about the processor.

⇒ Only considered how many number of steps an algorithm will take.

A_1 → 100 Steps
A_2 → 20 "
A_3 → 5 "

} A_3 is faster than A_1 & A_2.

Primitive Operations: It corresponds to a low level instruction with an execution time that depends on the H/w & S/w environment but is nevertheless constant. The following are considered as primitive, —.

→ assigning a value to a variable
→ calling a method
→ performing an arithmatic operation
→ Comparing two numbers.
→ Indexing into an array.
→ Following an object reference
→ Returning from a method.

Algorithm arrayMax (A, n):

Input: An array A storing $n \geq 1$ integers.

Output: The maximum element in A.

①	currentMax ← A[0]	2
②	for i ← 1 to n-1 do	2+n
③	if currentMax < A[i] then	2(n-1)
④	currentMax ← A[i]	2(n-1)
4.1	{increment}	2(n-1)
⑤	return currentMax.	1

Line ① corresponds to two primitive operations.
- → indexing into an array. ⇒ A[0]
- → assigning to a variable. ⇒ currentMax ← A[0]

Thus it contributes 2 units of time.

Line ② at the beginning the counter initialized to '1'
- → assigning value to a variable.

Thus it contribute an unit amount of time

③ before entering the body of the loop a condition i<n is verified.
this action corresponds to executing one primitive instruction
→ (comparing two numbers).
it will iterate for n+1 time.

③, 4, 4.1

The body of the for loop is executed 'n-i' time.
at each iteration

<3> ⇒ A[i] is compared to currentMax.
- → Indexing into the array
- → comparing with currentMax.

Thus it involves two operations, & takes 2(n-1) time.

Statement <4> ⇒ A[i] is possibly assigned to currentMax.
- → Indexing into the array.
- → ~~assigning A[i] currentMax.~~
- → Assigning A[i] to current max.

Thus it takes two operations & 2(n-1) time.

<4.1> Increment. $i = i+1$.

→ addition operation
→ assignment operation.

But it also involves two primitive operations.
& takes $2(n-1)$ times.

(5). Returning the value of variable currentMax corresponds to one primitive operations & executes only once.

⟶ One unit of time.

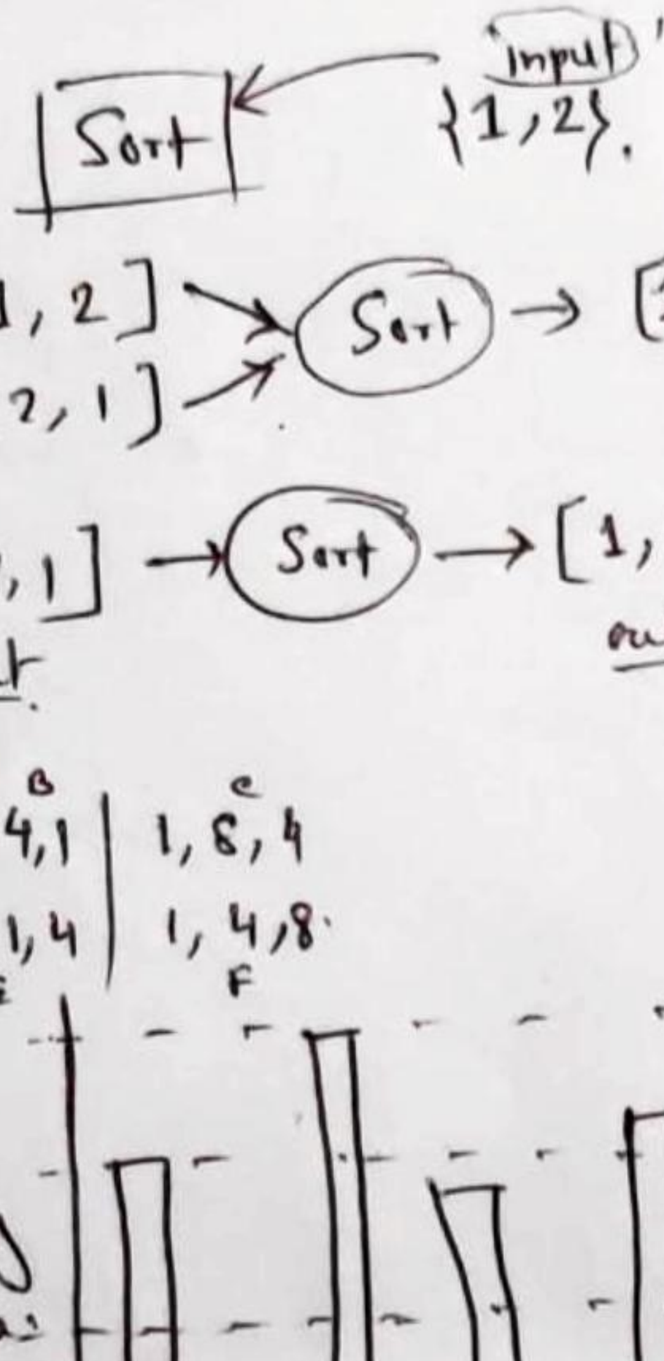
Thus summing up all steps.

$$t(n) = 2 + 2+n + 2(n-1) + 2(n-1) + 2(n-1) + 1$$

$$t(n) = 7n-2.$$

Best - Average - Worst

⇒ Read Again the RAM model in page-2.



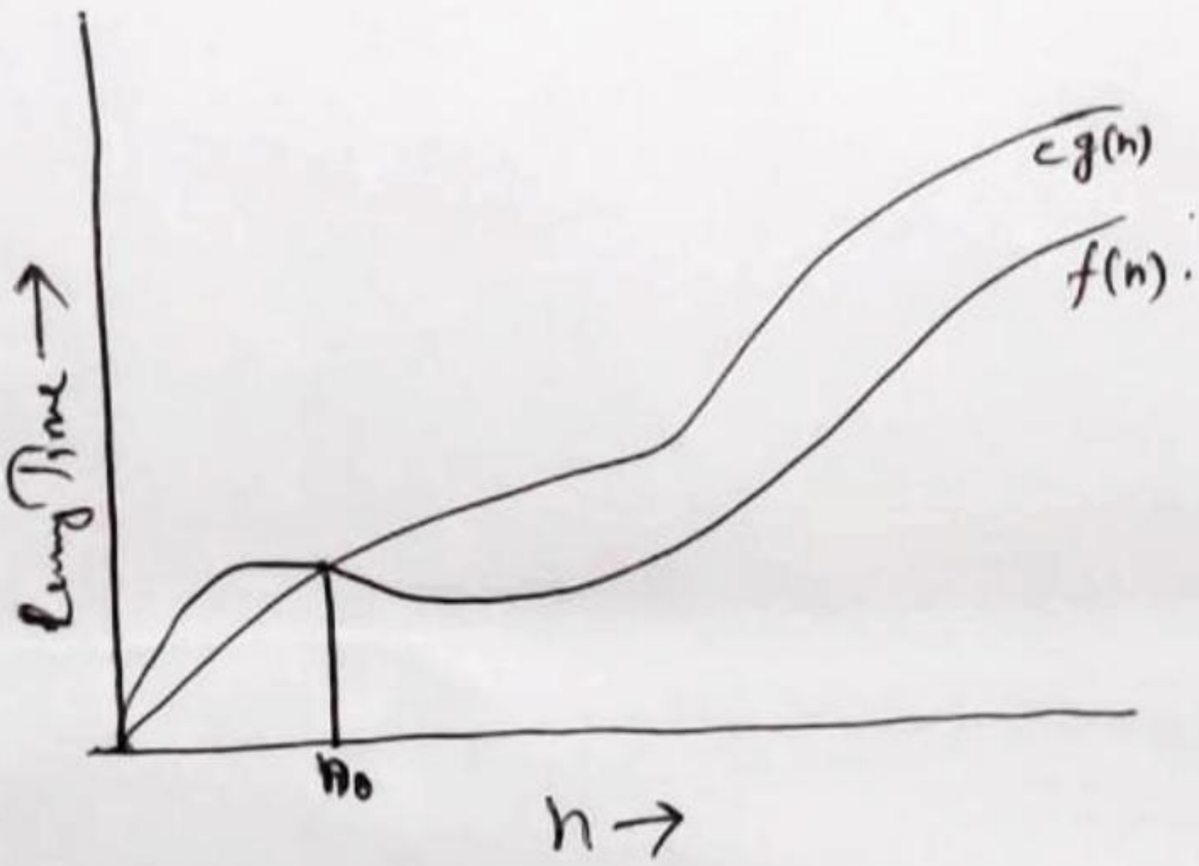
The Big-oh notation

Let $f(n)$ & $g(n)$ are functions mapping non-negative integers to real numbers.

we say that $f(n)$ is $O(g(n))$ if there in a real constant $c > 0$ & an integer constant $n_0 \geq 1$

Such that $f(n) \leq c\,g(n)$ for every integer $n \geq n_0$.

This condition is refer as big-oh or we can say $f(n)$ is of the order of $g(n)$.



Example:- $7n-2$ is $O(n)$.

Proof:- by the definition of big-oh we have to find a constant ~~c & n₀~~ $c\,(>0)$ & $n_0\,(\geq 1)$ such that

$7n-2 \leq cn$. for every $n \geq n_0$.

it's easy to see that $c=7$ & $n_0=1$ However, This is One of the many possible solution.

Let $d(n)$, $e(n)$, $f(n)$ & $g(n)$ are functions mapping non-negative integers to non negative reals then, —

1) If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$ for any constant $a>0$.
2) if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)+e(n)$ is $O(f(n)+g(n))$.
3) if $d(n)$ is $O(f(n))$ & $e(n)$ is $O(g(n))$ then $d(n)e(n)$ is $O(f(n)g(n))$.
4) If $d(n)$ is $O(f(n))$ & $f(n)$ is $O(g(n))$ then $d(n)$ is $O(g(n))$.
5) if $f(n)$ is a polynomial of degree d (i.e. $f(n)=a_0+a_1n+\cdots+a_dn^d$), then $f(n)$ is $O(n^d)$.
6) n^x is $O(a^n)$ for any fixed $x>0$ & $a>1$.
7) $\log n^x$ is $O(\log n)$ for any fixed $x>0$
8) $\log^x n$ is $O(n^y)$ for any fixed constants $x>0$ & $y>0$.

⇒ It is considered poor test to include constant factors & lower order terms in the big-oh notation.

Example: $2n^3+4n^2\log n$.

by applying theorem as follows: -

→ $\log n$ is $O(n)$ (rule 6)

→ $4n^2\log n$ is $O(n^3)$ (Rule 3)

→ $2n^3+4n^2\log n$ is $O(2n^3+4n^3)$ (Rule 2).

→ $2n^3+4n^3$ is $O(n^3)$ (Rule 5 or Rule 1)

→ $2n^3+4n^2\log n$ is $O(n^3)$.

⇒ Commonly used functions which appear often in the algorithms, —.

logarithmic.	linear	quadratic	Polynomial.	exponential
$O(\log n)$.	$O(n)$	$O(n^2)$	$O(n^k)$ $k>1$	$O(a^n)$ $a>1$.