LECTURE 1

Main metrics to characterize / judge an algo :

1. Running times
2. Space usage ; will be largely dependent on the kind of data structures used.

→ Industry is more interested in optimizing Run times.

→ It's expected that given 2 algos running on similar machines
- ↳ faster run-times are more desirable
- ↳ space complexity is only considered if it affects the run-time in some way.

→ Measuring an Algo's performance
- ↳ function (input size) ⇒ $f(n)$
  - ↳ Basically a function which defines rate of growth of time w.r.t time

→ Now, we can run a prog. for different input size & note the (end time - start time)
- but this may not be the best way to determine actual $f(n)$
- as we would prefer to measure the performance independent of machine specs.

→ Hence we have come up with RAM Model of algo performance func. determination

Primitive Ops : A low level instruction with execution time dependent on hardware or software but will largely remain same.

e.g. ① Assigning value to a variable ② Comparing two numbers.

We do not go into the nitty gritty of these instruction run times.

Instead, we count no. of Primitive ops = t

RANDOM ACCESS MACHINE (RAM) MODEL

Assumptions

- Computer is a simple CPU connected to bank of memory cells
- CPU in RAM model can perform any primitive ops in constant no. of steps irrespective of input size.

E.g. Find max element in an Array

Algo arrayMax (A, n)
I/p : Array A of length $n \geq 1$ (integers)
o/p : Max entry in A

2(po) assigning i; initialize Array A ← currentMax = A[0] → since we started indexing from 0.
for i=1 to n-1 do → n(po)
1(po) if currentMax < A[i] then → 4(n-1) (po) OR 6(n-1) (po)
currentMax = A[i]
return currentMax → 1(po)

∴ at least

$$2 + 1 + n + 4(n-1) + 1 = 5n$$

or at most

$$2 + 1 + n + 6(n-1) + 1 = 7n - 2$$

Here, best case occurs If A[0] is max i.e. $5n$ (no reassignment of A[0] needed)
worst case if array is sorted in ascending & currentMax is reassigned in every loop.

★ Recursion not same as Iteration

Recursion : Procedure P is allowed to make calls to itself as a subroutine.
Required Base Case that can be solved w/o recursion.

Algo recursiveMax (A, n) :
I/O = array A of length n
O/P = Max element of A
if n = 1 then
return A[0]
return max {recursiveMax (A, n-1), A[n-1]}

Statement	# of PO	# of repitions
if n=1 then	1 , 1	1 → assign n=1 ; n-1 → no. of times if condition is run
return A[0]	2	1 → will be executed only once.

return max {recursiveMax (A, n-1), A[n-1]}

A[n-1]	1	n-1
A, n-1	1, 1	n-1
recursiveMax (A, n-1)	1	n-1, n-1
max { " , A[n-1] }	1	n-1
return	1	n-1

$$\text{Total} = 1 + (n-1) + 2 + 6(n-1)$$
$$= 7n - 4$$

Counting primitive operations

innovate achieve lead

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size

Algorithm ArraySum(A, n)	#Operations	Remarks
(1) Sum = A [0]	2	Indexing , Assignment
(2) i = 1	1	Assignment
(3) while (i<n)	n	Comparison
(a) Sum = Sum + A[i]	3 (n-1)	(n-1) times indexing, addition and assignment
(b) i = i + 1	2 (n-1)	(n-1) times addition and assignment
(4) return Sum	1	1 times returning

Total Primitive Operations = 2 + 1 + n + 5 (n-1) + 1 = 4 + n + 5n - 5 = 6n -1

Q. Solve the f(n) for following code

```
① j=0;
② while(j<N)do
        ③k=0
        ④while(k<N)do
                ⑤a[k] = a[j] + a[k]
                ⑥k=k+1;
        ⑦endwhile
⑧b[j] = a[j]+j
⑨j=j+1
⑩endwhile
```

Steps	# of PO.	# repetition
①	1	1
②	1	N+1 ; while runs 1 extra time.
③	1	N+1
④	1	$(N+1)^2$ → outer loop of j, inner loop of k
⑤	1,1,1,1,1 = 5	all of them $(N+1)^2$
⑥	1,1 = 2	$(N+1)^2$
⑧	1,1,1,1,1 = 5	N+1 – we are now in outer loop
⑨	1,1 = 2	N+1

Total = $1 + (N+1) + (N+1) + (N+1)^2 + 5(N+1)^2 + 2(N+1)^2 + 5(N+1) + 2(N+1)$

$= 1 + 9(N+1) + 7(N+1)^2$

$= 1 + 9N + 9 + 7N^2 + 7 + 14N$

$= 7N^2 + 23N + 17$

or $f(n) = O(n^2)$ ⟶ Big Oh notation.

'Big-Oh' Notation:- Textbook 1; 1.2

→ To get $O(n)$ for time complexity of a problem, say $f(n) = 7n - 2$

- first drop any constant (here, -2)
- second drop any lower form of n (like if $f(n)$ had n^2 & n, we could drop n ∵ n^2 is of a higher order).
- drop any constant attached to the highest order of n.

∴ $O(n)$ of given $f(n) = O(n)$

Ex 1 $20n^3 + 10n \log n + 5$
$= O(n^3)$

Ex 2 $3\log n + \log \log n$
$= O(\log n)$ ∵ $\log \log n < \log n$

Ex 3 2^{100}
$= O(1)$ ∵ 2^{100} is a constant.

Ex 4 $5/n$
$= O(1/n)$

Theorem 1.7: *Let $d(n)$, $e(n)$, $f(n)$, and $g(n)$ be functions mapping nonnegative integers to nonnegative 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 $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.*
4. *If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.*
5. *If $f(n)$ is a polynomial of degree d (that is, $f(n) = a_0 + a_1 n + \cdots + a_d n^d$), then $f(n)$ is $O(n^d)$.*
6. *n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$.*
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 $x > 0$ and $y > 0$.*

Common terminology in Big oh notation!

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

Table 1.6: Terminology for classes of functions.

polynomial runtime is mostly better than an exponential runtime

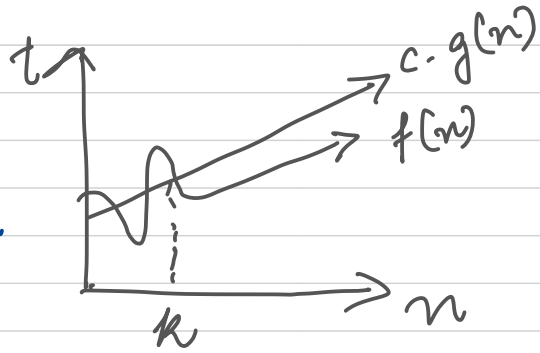
we say $f(n) \in O(g(n))$; $f(n)$ is O of $g(n)$ (→ order)

Big-Oh (O)

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

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

such that $c > 0$, $n \geq k$ & $k \geq 0$



WORST CASE
upper bound
(At most)

Ex: Say $f(n) = 2n^2 + n$

∵ $O(g(n))$ talks about worst case scenario, the worst case for $n \geq 1 = O(n^2)$ for given $f(n)$

So here, $g(n) = n^2$

Big-Omega (Ω)

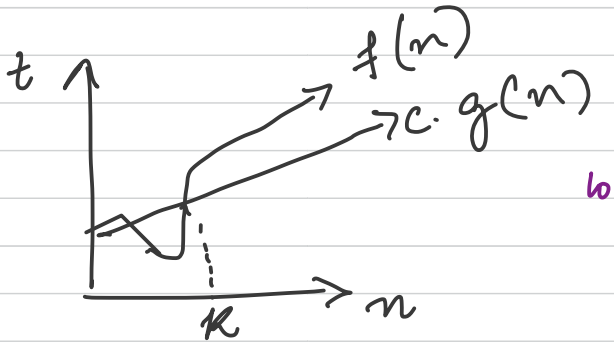
$f(n) = \Omega\, g(n)$ so for $n \geq 0$

$f(n) \geq c \cdot g(n)$ $\Omega\, g(n) = n^2$

Ex. $2n^2 + n \geq c * n^2$

$c=2$ $2n^2 + n \geq 2n^2$

$n \geq 0$ * One function is $\geq$ to another, upto a constant factor (c)



BEST CASE
lower bound
(At least)

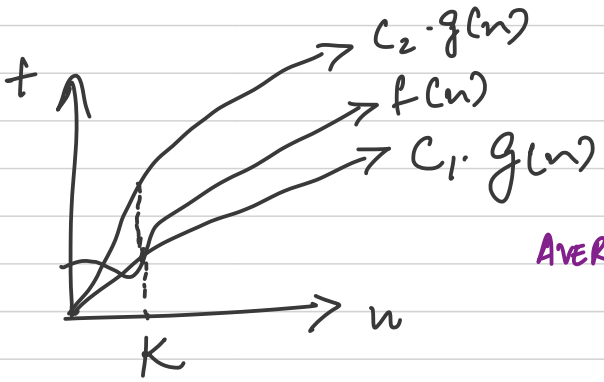
Big-Theta (Θ)

Denoted as

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

$c_1 g(n) \leq f(n) \leq c_2 g(n)$

$1 n^2 \leq 2n^2 + n \leq 3n^2$



AVERAGE CASE

Little-O $f(n) < cg(n)$

Little ω (omega) $f(n) > cg(n)$

→ YT video that explained it : https://youtu.be/7dz8laf_weM

CLASS QUESTIONS :-

Questions

- Is $T(n) = 9n^4 + 876n = O(n^4)$? ✓
- Is $T(n) = 9n^4 + 876n = O(n^3)$? x $O(n^4)$
- Is $T(n) = 9n^4 + 876n = O(n^{27})$? x $O(n^4)$

- $T(n) = n^2 + 100n = O(?)$ $O(n^2)$
- $T(n) = 3n + 32n^3 + 767249999n^2 = O(?)$ $O(n^3)$

— END