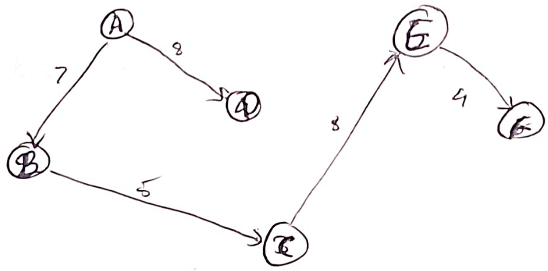
DsA

Complexity theory & time Running analysis

- Running time Analysis. (Complexity theory.)
    - there are 2 type of complexity theory.
        - memory (space) complexity (amount of space it needs)
        - Time complexity (amount of time it needs to run
    - if you want any one of them better than you have to trade off other one to have.

      ex:- if you want to execute in less time it needs more memory or vice-versa
    - ~~the~~ now, the problem is how do we consider it is fast enough or not. (algorithem).
    - ex:-



- if you want to find the shortest path if we want to execute it on smartphone the it's going to be fast but if we want execute it on PC's then it's gonna be faster. so then we cannot say the time is the only measurement.

- we also have to consider the devices as well. (it's not right)
- instead - with respect to the input size & the no. of steps the algorithem requires.
- because it's generic & machine independent.
- ex: we have an array of 1 D which elements are.

12	4	-2	1	20	0	8	3

to analyze algorithem we have to consider no. of items. we are dealing with (input size

10 items	- 100ms
100 items	- 1000 ms

- assumption calculation. (no. of i/p items) - taks certain (ms) to execute based on that we consider ~~the~~ the result
- in 1D array we have 8 items & we have 80ms to sort if the time taken to execute sorting based on that we consider best, average, worst
- algorithems running time with respect to the no. of itemes (input)
- this is the order of growth - how the algorithem scales and behaves with the N input sizes.

10 items	—	100 ms	
100 items	-	1000 ms	← Good
100 items	-	100000 ms	← bad.

- we prefer algorithems when the running time scales linearly with the inputsize

input size			running time
10 items	x 10	=	100 ms
100 items	x 10	=	1000 ms
1000 items	x 10	=	10000 ms

- what's the probleme in other approach.
  it doesn't scale well & we want to make sure it doesnt freeze during the sorting
- and we like deterministic algorithems where the running times are approximatly linear or ~~non~~ sublinear.

complexity theory illustration

#1 Algorithem

sorting 10 items : 1ms

sorting 20 items : 2ms

sorting 100 items : 100 ms.

- it's called O(N) linar running time
- bcoz it's scaling linearly with the input size

2 Algorithem

sorting 10 items : 1 ms
sorting 20 items : 4 ms
sorting 100 items : 100 ms.

- this is called $O(N^2)$ quadratic running time
- as the running time scales quadratically with input size.

- we always choose $O(N)$ because it's increasing linearly
- usually we are interested in large i/p sizes (Asymptotic Analysis.)

- we will drop the terms that grow slowly & only keep the ones that grow fast as N (so the input size) becomes larger.