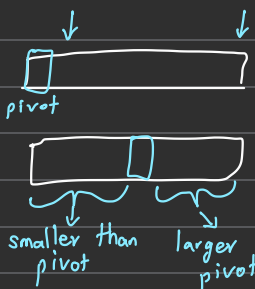
Introduction

22nd Aug -

Problem solving → How to solve it (book ⭐)
- G. Polya

① Check sorts
- bubble/selection
- quick sort
- recursive to add 10 nos.

QUICK SORT :

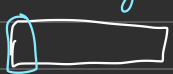


BUBBLE :

CHECK

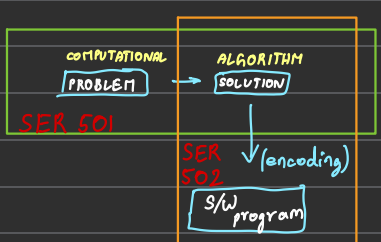
- element is compared with each ele
- then swapped each time if greater

SELECTION :



- element compared with each ele
- then only one ele is swapped

CHECK



check out ACM

Big O :

O(...) → set

→ set of algos. that can run in a time

→ complexity of quick sort belongs to $O(n^2)$

→ big O & worst case are diff.

→ can say big O in avg. case too.

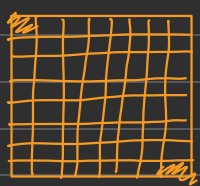
27th Aug:

→ some problems have a sol^n., but no algorithm.

→ software is like a black box,
data enters & data manipulation takes place.

① Find slides, go through

8 x 8 problem:



→ think of it as chess board
→ removing corners (diag) means 1 black & 1 white
→ domino always covers 1 black & 1 white
→ must only remove same

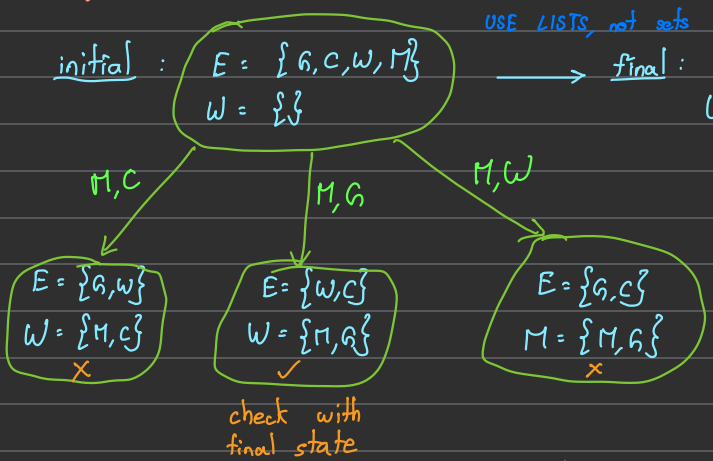
2 things in computational problem solving-
→ representation (capturing all relevant aspects)
→ algorithm

problem → ABS → rept. → algo → program

ABSTRACTION-

removing unnecessary info.
→ Ex- data types & sys. libraries

Goat, cabbage & wolf problem-



→ avoid infinite loops by checking if a state has been visited.

Check slides

using Lists:

1 LIST only

	INITIAL					FINAL			
	M	G	C	W		M	G	C	W
[	E,	E,	E,	E]	→	[W,	W,	W,	W]

if you think ① 2 sides → then you end up with 2 lists

② where item is → then 1 list

29th Aug:

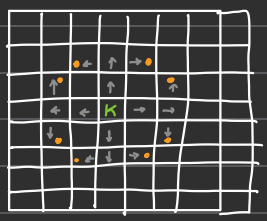
Data Structures - way to store & organize data

↳ ⓐ When there is RELATION b/w data

① Knight store problem
⇒ chess.

LEARN:

Knight store problem



Primality Algo-
(check)

Date problem:

① Use array of structures,
each date is a struct, looks like : struct date { int day int mo int yr }

② Do not use 2-d arrays,
cuz does not show that '3' parts of date belongs to a single date.

P & NP -

CHECK

P - Polynomial

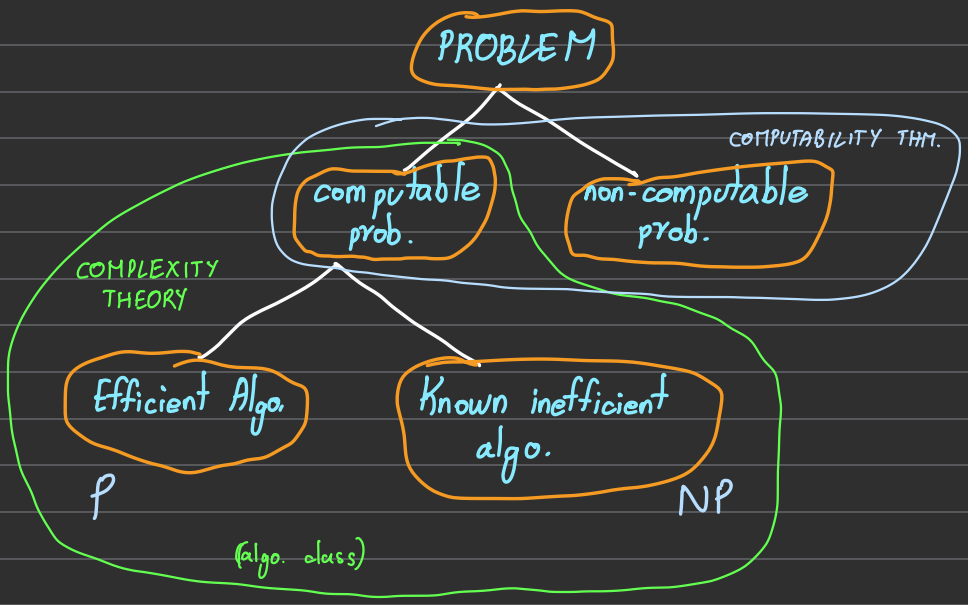
NP - Non-deterministic polynomial (approximation algorithms)

P is a subset of NP

P ⊂ NP

P = NP | P ⊂ NP

Limits of computations :



For every prog. in the world, for 1 input the prog. will halt.

$H(P, z)$

YES NO

Limits of computational theory

Look at GCD, Blast
→ go through PPT.

3rd Sept :

Algorithm: Finite steps of unambiguous steps to solve a problem.

Learn maybe 10 basic DS, but CREATE YOUR OWN based on need.

5 big steps :

1. Analysis of Algorithms
2. Algorithmic techniques
3. Data structures - Trees, mostly graphs
5. Complexity theory

Algorithm Analysis

ANALYSIS of ALGOS:

① Asymptotic analysis:

$$1 + 8 + 27 = 36$$
$$\Rightarrow 1^3 + 2^2 + 3^2 = 6^2$$

$$1^3 + 2^2 + 3^2 + 4^3 = 64 + 6^2$$
$$\Rightarrow 1^3 + 2^3 + 3^3 + 4^3 = 10^2$$

$$1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 15^2$$

Trying to proove- $1^3 + 2^3 + 3^2 + \ldots + (n-1)^3 = p^2$

$$\Rightarrow 1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 = \left[\frac{n(n-1)}{2}\right]^2 \quad -①$$

if ① is true, then:

$$1^3 + 2^3 + 3^3 + \ldots + (n-1)^3 + n^3 = \left[\frac{n(n+1)}{2}\right]^2 \quad -②$$

subtract ① from ②:

$$n^3 = \left[\frac{n(n+1)}{2}\right]^2 - \left[\frac{n(n-1)}{2}\right]^2$$

no need to prove ①.
cuz if ① is TRUE, then ② is TRUE.

$P(n) \longrightarrow P(n+1)$
✓ ✓

INDUCTION:

IF you are trying to prove $(n-1)$ is true, then show its TRUE for 'n'.

Instead of proving for 'n' in general, then ASSUME it is true for $(n-1)$. then prove for 'n' in general.

① Math induction
② Recursively add 10
③ Proof by contr-adiction
→ assume opposite of what is true.

CHECK SLIDES

Example 1: Prove $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$ is true for all $n \geq 0$

① For $n = 0$, $\sum_{k=0}^{0} k = 0$

CANNOT ASSUME WHAT YOU ARE TRYING TO PROVE.

② Assume that it is true for 1 single value of n, NOT all values of n.

∴ For $n = p$, $\sum_{k=0}^{p} k = \frac{p(p+1)}{2}$ → if you use n, $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$

③ so, show: → for $(n = n+1)$ basically

$$\sum_{k=0}^{n+1} k = (n+1) + \sum_{k=0}^{n} k$$

→ replace with ↗

(after this is MATH)

At home -

① Prove $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$ for $n \geq 0$

ⓐ for $n=0$, $\sum_{k=0}^{0} k = \frac{0(0+1)}{2}$

$\therefore \boxed{\sum_{k=0}^{0} k = 0}$ — (1)

ⓑ For arbitrary p (let's say p), assume statement is TRUE

$$\sum_{k=0}^{p} k = \frac{p(p+1)}{2} \quad - (2)$$

Now, this p can be any value of $n \geq 0$.
So, say 'p' is 'n' (greatest value n can take)

From (2), $\boxed{\sum_{k=0}^{n} k = \frac{n(n+1)}{2}}$ — (3)

ⓒ Statement was assumed to be true for $n = 'n'$, so, let's see if statement is true for $n = 'n+1'$.

$$\therefore \sum_{k=0}^{n+1} k = (n+1) + \sum_{k=0}^{n} k$$

↳ It is (3)

① so here, first term on RHS will be the 'k' of LHS, replaced with upperbound of LHS
② and second term of RHS will be the LHS of assumption.

From (3),

$$\sum_{k=0}^{n+1} k = (n+1) + \frac{n(n+1)}{2}$$

This replacement is cuz → if you split the LHS: all terms ADD upto $\sum_{k=0}^{n} k$ & then last term is $(n+1)$.

$$= \frac{2(n+1) + n(n+1)}{2}$$

$$\therefore \boxed{\sum_{k=0}^{n+1} k = \frac{(n+1)(n+2)}{2}}$$

↳ Which is basically eq. (3) if you replace 'n' by '$n+1$'.

Hence, proved. ∎

② Prove $\sum_{k=1}^{n} 2k-1 = n^2$ for $n \geq 1$

ⓐ For $n=1$: $\boxed{\sum_{k=1}^{1} 2k-1 = 1}$ — ①

ⓑ For $n = 'n'$: $\boxed{\sum_{k=1}^{n} 2k-1 = n^2}$ — ②

ⓒ For $n = 'n+1'$: $\sum_{k=1}^{n+1} 2k-1 = 2(n+1)-1 + \sum_{k=1}^{n} 2k-1$

↳ replace with eq. ②

③ Prove $\sum_{k=0}^{n} 2^k = 2^{n+1}-1$, $n \geq 0$

ⓐ For $n=0$, $\sum_{k=0}^{0} 2^k = 2^1 - 1 \Rightarrow \boxed{\sum_{k=0}^{0} 2^k = 1}$ — ①

ⓑ For $n = 'n'$, $\boxed{\sum_{k=0}^{n} 2^k = 2^{n+1}-1}$ — ②

ⓒ For $n = 'n+1'$, $\sum_{k=0}^{n+1} 2^k = 2^{(n+1)} + \sum_{k=0}^{n} 2^k$

↳ replace with eq. ②

④ Prove $\sum_{k=0}^{n} r^k = \frac{1-r^{(n+1)}}{1-r}$

ⓐ For $n=0$, $\sum_{k=0}^{0} r^k = \frac{1-r^{(1)}}{1-r}$

ⓑ For $n = 'n'$, $\sum_{k=0}^{n} r^k = \frac{1-r^{(n+1)}}{1-r}$

ⓒ For $n = 'n+1'$, $\sum_{k=0}^{n+1} r^k = r^{(n+1)} + \sum_{k=0}^{n} r^k$

(These 2 proofs will not be ASKED in exam, but understand PROOF of CONTRADICTION)

5th Sept-

Proof by contradiction.

(a) There is no largest prime no.

→ Assume p_k is largest prime no.

$p_1, p_2, p_3 \ldots\ldots p_k$; P is prime no.
p_k is largest prime no.

$p_1 \times p_2 \times \ldots \times p_k$ → NOT prime no.

but, $(p_1 \times p_2 \times \ldots \times p_k) + 1$; IS A PRIME no.
①

Relative to all $[p_1, p_2 \ldots p_k]$, the no. $[(p_1, p_2 \ldots p_k) + 1]$ is a PRIME no.
②

∴ ASSUMPTION DISPROVED.

There is a big gap b/w ① & ②.
② is NOT the next biggest prime after ①.

(b) $\sqrt{2}$ is an irrational no.

→ Assume $\sqrt{2}$ is rational.

$\sqrt{2} = \frac{p}{q}$; p & q are integers

$\Rightarrow 2q^2 = p^2$ → p^2 will be even, p also even

say $(p = 2m)$,

$\Rightarrow 2q^2 = 2m^2$

$\Rightarrow q^2 = 2m^2$ → q^2 also even, q also even

∴ 2 is common factor b/w p & q.

so, assumption that $\sqrt{2} = \frac{p}{q}$ is DISPROVED.

Sum of first 'n' odd nos. $= n^2$, $\forall n > 0$

↳ works only cuz of constraint

(A) 'n' is no. of odd nos. being considered, NOT the value.

Ex- $\forall n \geq 0$ $P(n)$ is true.

1. Show $n=0$, $P(0)$ is TRUE.
2. Assume for some 'n', $P(n)$ is TRUE
3. Prove $P(n+1)$ is TRUE.

$$P(0) \wedge \big(P(n) \rightarrow P(n+1)\big) \longrightarrow P(n) \ \forall n \geq 0$$

'0' is True; 'n' is TRUE; 'n+1' is TRUE;

for all $n \geq 0$, $P(n)$ is TRUE

if $P(n)$ is TRUE, $P(n+1)$ is TRUE.

At Home :

① Prove for $n>0$, n^3+2n is divisible by 3

ⓐ For $n=0$: $(0)^3 + 2(0)$

$\Rightarrow \underline{0}$ → is divible by 3

ⓑ → Prove that the truth of the formula for 'n' implies the truth for 'n-1'

→ Prove that the truth of the formula for 'n' implies the truth for 'n+1'

∴ Assume that statement is true for a given n: n^3+2n is divisible by 3.

Now,

For $(n-1)$:

$\Rightarrow (n-1)^3 + 2(n-1)$

$\Rightarrow n^3 - 3n^2 + 3n - 1 + 2n - 2$

$\Rightarrow n^3 - 3n^2 + 5n - 3$

$\Rightarrow n^3 + 2n + 3n - 3n^2 - 3$

$\Rightarrow n^3 + 2n + 3(n - n^2 - 3)$

↓ div. by 3, assumed ↓ div. by 3, clearly

For $(n+1)$:

$\Rightarrow (n+1)^3 + 2(n+1)$

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

$\Rightarrow n^3 + 3n^2 + 5n + 3$

$\Rightarrow n^3 + 2n + 3n + 3n^2 + 3$

$\Rightarrow n^3 + 2n + 3(n + n^2 + 3)$

② Prove for all $n>0$, (3^n-1) is a multiple of 2.

① For $n=1$, $3^{(1)}-1$

$\Rightarrow 2 \rightarrow$ is a multiple of 2

②→ Prove that the truth for the formula for 'n' implies the truth for 'n-1'.

→ Prove that the truth for the formula for 'n' implies the truth for 'n+1'.

∴ Assume truth for formula for some value 'n'; 3^n-1 is multiple of 2.

So now:

For (n-1):

$\Rightarrow 3^{(n-1)}-1$

$\Rightarrow \frac{3^n}{3}-1$

For (n+1):

$\Rightarrow 3^{(n+1)}-1$

$\Rightarrow (3^n \times 3)-1$

Ⓐ Complete

③ Prove : $1.2 + 2.3 + 3.4 + \ldots + n(n+1) = \left(\frac{1}{3}\right)[n(n+1)(n+2)]$

ⓐ For $n = 1$:

$$1.2 = \left(\frac{1}{3}\right)[1(1+1)(1+2)]$$

$$\Rightarrow \quad 2 = \frac{6}{3} \quad \Rightarrow \quad \boxed{2 = 2} \quad - (1)$$

ⓑ Assume st. TRUE for some $n = 'k'$

$$\boxed{1.2 + 2.3 + 3.4 + \ldots + k(k+1) = \frac{1}{3}[k(k+1)(k+2)]} \quad - (2)$$

ⓒ For $n = k+1$,

$$1.2 + 2.3 + 3.4 + \ldots + (k+1)(k+2) = \frac{1}{3} \times [(k+1)(k+2)(k+3)]$$

(term to the left of $(k+1)(k+2)$)

$$\Rightarrow \quad \underbrace{1.2 + 2.3 + 3.4 + \ldots + k(k+1)}_{\text{FROM eq. (2), replace this.}} + (k+1)(k+2) = \frac{1}{3} \times [(k+1)(k+2)(k+3)]$$

$$\Rightarrow \quad \frac{1}{3}[k(k+1)(k+2)] + (k+1)(k+2) =$$

Factor out $(k+1)(k+2)$

$$\Rightarrow \quad (k+1)(k+2)\left[\frac{1}{3}k + 1\right] = \frac{1}{3} \times [(k+1)(k+2)(k+3)]$$

$$\Rightarrow \quad \therefore \frac{1}{3} \times [(k+1)(k+2)(k+3)] = \frac{1}{3} \times [(k+1)(k+2)(k+3)]$$

Hence, proved.

④ Prove : $1.3 + 3.5 + 5.7 + \dots + (2n-1)(2n+1) = \frac{1}{3} \times [n(4n^2 + 6n - 1)]$

ⓐ For $n=1$: $1.3 = \frac{1}{3} \times 1(4 + 6 - 1)$

$\Rightarrow \quad 3 = \frac{1}{3} \times 9 \quad \Rightarrow \quad \boxed{3 = 3}$ - ①

ⓑ Assume st. is true for some $n = 'k'$,

$1.3 + 3.5 + 5.7 + \dots + (2k-1)(2k+1) = \frac{1}{3} \times [k(4k^2 + 6k - 1)]$ - ②

ⓒ For $n = k+1$:

$1.3 + 3.5 + \dots + (2(k+1)-1)(2(k+1)+1) = \frac{1}{3} \times [(k+1)(4(k+1)^2 + 6(k+1) - 1)]$

$\Rightarrow \quad 1.3 + 3.5 + \dots + (2k-1)(2k+1) + (2(k+1)-1)(2(k+1)+1) = \quad "$

$\Rightarrow \quad \frac{1}{3} \times [k(4(k)^2 + 6k + 1)] + [2(k+1)-1][2(k+1)+1] = \quad "$

$\Rightarrow \quad \frac{1}{3} \times [k(4k^2 + 6k + 1)] + [2k+1][2k+3]$

$" \quad + \quad 4k^2 + 6k + 2k + 3 = \quad "$

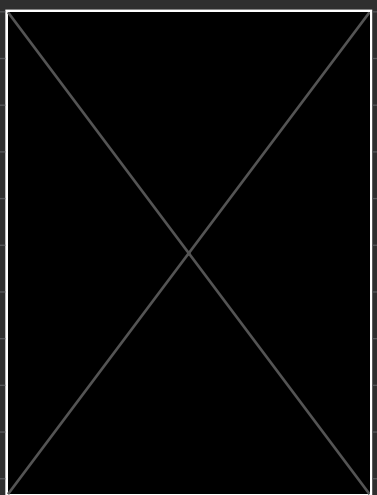
$\Rightarrow \quad \frac{1}{3} \times [k(4k^2 + 6k + 1)] + (4k^2 + 6k + 1) + 2(k+1) = \quad "$

(*) complete

10th Sept.

Logarithms : $\log_b x = y$

How many times do you divide x by 'b' to get 1's



Logs come into the picture when there is repeated division of data in algo.
That is why QUICKSORT complexity is $n \log(n)$

$b^m = n$ $\quad$ $\log_b n = m$

$\Longrightarrow$ Repeated division.

Logarithmic complexity will ALWAYS be SUBSTANTIALLY better than polynomial.
Logs grow slowly.

All logs grow the same, but are shifted.

lg means $\log_2$
ln means $\log_e$

12th Sept:

No. of Natural nos. : 1, 2, 3, 4 ..... 'N' natural nos.

(multiply by 2)

No. of even nos. = 2, 4, 6, 8 ..... 'N' even nos.

∴ no. of Natural nos = no. of Even nos.

= no. of Odd nos.

= no. of Rational nos.

dealing with ∞ is hard.

Any enumerate seq. that can be written has same nos. as no. of Natural nos.

'REAL nos' have a bigger ∞ than natural nos.

$|R| > |N|$, cardinality of Real nos. GREATER that Natural nos.

Continuum hypothesis.

2 diff. functions -

$\lim_{x \to \infty} f(x) = \infty$ ⋮ $\lim_{x \to \infty} g(x) = \infty$

take RATIO:

$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$ → then $f(x)$ grows SLOWER than $g(x)$

$\lim_{x \to \infty} \frac{f(x)}{g(x)} = \infty$ → then $f(x)$ grows FASTER than $g(x)$

→ then SAME pace

Recurrance Relation : * check audio 5

→ 'n' will always be positive, 'n' → no. of elements (input size)

Asymptotic analysis : * pdf start → check audio

17th Sept- Recording Available

'n' → no. of operations

$f(n) \rightarrow g(n)$

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

Big O: ⇒ $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$

↳ g(n) will be considered upper bound

$2n^2 \leq 3n^2$

(beyond) At some point, f(n) will be behind g(n).

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

↳ no. of operations f(n) will take will alwaysss be less than no. of operations g(n) will take.

we are approximating.
but relative error is LESS, so approximation is allowed.

2 algorithms - → A_1 & A_2

→ compared by finding O of each then comparing.

$O(n^2)$ → set of all functions that can be upperbounded asympotically by (n^2).

$\therefore$ f(n) belongs to $O(n^2)$.

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ means $f(n) < g(n)$
↳ strictly.

Omega: $f(n) \geq c \cdot g(n)$ → $f(n)$ will always take more op. than $g(n)$

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

obv., upperbound $(O())$ is more useful than lowerbound $(\Omega())$.

Lowerbound for a comparison based sorting is '$n \cdot \log(n)$', only place Ω is useful.

Theta:

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

then, $f(n) = \Theta \cdot g(n)$

if $f(n) = O(n^2)$, then $f(n) = O(n^3)$ & $f(n) = O(n^3)$
if $f(n) = \Omega(n^2)$,

↳ this helps us define Θ (theta)

✱ Don't use BEST, WORST & AVERAGE case, cuz that depends on input.
→ each case can have upper & lower bound.
→ O : upperbound
Ω : lowerbound

$f(n) = O(g(n))$
→ $f(n)$ is UPPERBOUND by $g(n)$

$f(n) = \Omega(g(n))$
→ $f(n)$ is LOWERBOUND by $g(n)$

Big O :

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

→ $f(n)$ is **UPPERBOUND** by $g(n)$

→ $f(n)$ has Rate of Growth **NO GREATER** than $g(n)$

→ $0 \leq f(n) \leq c.g(n)$

→ Example : $f(n) = O(n^2)$

→ no. of operations $f(n)$ will take is always going to be lesser than 'n^2'.

Omega :

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

→ $f(n)$ is **LOWERBOUND** by $g(n)$

→ $f(n)$ has Rate of Growth **NO LESSER** than $g(n)$

→ $f(n) \geq c.g(n)$

Big-O Notation	Comparison Notation	Limit Definition
$f \in o(g)$	$f \lessdot g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$
$f \in O(g)$	$f \preceq g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} < \infty$
$f \in \Theta(g)$	$f \eqsim g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} \in \mathbb{R}_{>0}$
$f \in \Omega(g)$	$f \succeq g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} > 0$
$f \in \omega(g)$	$f \gtrdot g$	$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty$

✡ The small 'o' is a stronger condition than big 'O'.

✡ First 4 slides of ppt2 from module3

19th Sept-

Recurrance Relation-

merge sort - dividing has NO effort, merging has
quick sort - dividing has effort, merging has NO effort

Analysis of Merge Sort

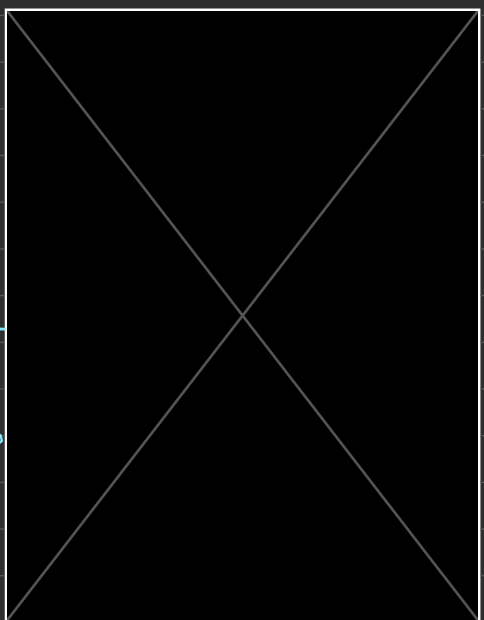
Statement	Effort
`MergeSort(A, left, right) {`	`T(n)`
`  if (left < right) {`	`Θ(1)`
`  mid = floor((left + right) / 2);`	`Θ(1)`
`  MergeSort(A, left, mid);`	`T(n/2)`
`  MergeSort(A, mid+1, right);`	`T(n/2)`
`  Merge(A, left, mid, right);`	`Θ(n)`
`  }`	
`}`	

- So T(n) = Θ(1) when n = 1, and
  2T(n/2) + Θ(n) when n > 1
- So what (more succinctly) is T(n)?

} Add these

} get this, ignore 2*Θ(1) cuz small

Recurrance Examples: * recording



c - constant effort at each level

if 'i' was 'n', then Bogo Sort

8rd: starting at 'n', repeatedly dividing by '8' to reach 1, so $\log_8(n)$.

4th: starting at 'n', repeatedly dividing by 'b' to reach 1, so $\log_b(n)$.

8 types we'll learn.

① Substitution · guess & check
② Iteration - expand recurrence { sometimes add, practice.
③ Master thm. · have a shortcut based on observation.

(check slides, good examples)

Polynomial - variable ^(constant)
Exponential - constant ^(variable)

At Home :

① $s(n) = \begin{cases} 0, & n=0 \\ c+s(n-1), & n>0 \end{cases}$

$s(n) = c + s(n-1)$

$\Rightarrow s(n) = c + c + s(n-2) \Rightarrow s(n) = 2c + s(n-2)$

$\Rightarrow s(n) = 2c + c + s(n-3) \Rightarrow s(n) = 3c + s(n-3)$

$\Rightarrow s(n) = 3c + c + s(n-4) \Rightarrow s(n) = 4c + s(n-4)$

$\Rightarrow$ ⋮

$\Rightarrow s(n) = kc + s(n-k)$

So far, for $n \geq k$, we have

$s(n) = ck + s(n-k)$

→ trying to make $s(n-k) \dashrightarrow s(0)$. so, choose $\boxed{n=k}$

if $n=k$, $s(n) = cn + s(0) \Rightarrow \boxed{s(n) = c.n}$

∴ In general, instead of $s(n) = \begin{cases} 0, & n=0 \\ c+s(n-1), & n>0, \end{cases}$

we say $s(n) = cn$

② $s(n) = \begin{cases} 0 & , n = 0 \\ n + s(n-1) & , n > 0 \end{cases}$

$s(n) = n + s(n-1)$

$\Rightarrow s(n) = n + n-1 + s(n-2)$

$\Rightarrow s(n) = n + n-1 + n-2 + s(n-3)$

$\Rightarrow s(n) = n + n-1 + n-2 + n-3 + s(n-4)$

$\Rightarrow$ ⋮

$\Rightarrow s(n) = n + n-1 + n-2 + \dots + n-(k-1) + s(n-k)$

$\Rightarrow s(n) = \sum_{i=n-k+1}^{n} (i) + s(n-k)$

so far, for $n \geq k$, we have:

$$s(n) = \sum_{i=n-k+1}^{n} (i) + s(n-k)$$

Reason for choosing '$n \geq k$':

① we are trying to replace $s(n-k)$ term with first value in the question.

② '$n-k$' should be equal to 0 for pt. 1 to happen.
so, $n-k=0 \Rightarrow \boxed{n=k}$

for $n = k$,

$$s(n) = \sum_{i=1}^{n} (i) + s(0)$$

$$\Rightarrow \boxed{s(n) = \frac{n(n+1)}{2}}$$

③ $$T(n) = \begin{cases} c & , n=1 \\ 2T\left(\frac{n}{2}\right) + c & , n>1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + c$$

$$\Rightarrow T(n) = 2\left[2T\left(\frac{n}{2\times 2}\right) + c\right] + c \Rightarrow T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2c + c$$

$$\Rightarrow T(n) = 2^2\left[2T\left(\frac{n}{2\times 2^2}\right) + c\right] + 3c \Rightarrow T(n) = 2^3 T\left(\frac{n}{2^3}\right) + 4c + 3c$$

$$\Rightarrow T(n) = 2^3\left[2T\left(\frac{n}{2\times 2^3}\right) + c\right] + 7c \Rightarrow T(n) = 2^4 . T\left(\frac{n}{2^4}\right) + 15c$$

$$\vdots$$

$$\Rightarrow T(n) = 2^k . T\left(\frac{n}{2^k}\right) + (2^k - 1)c$$

so far, for $n \geq 2^k$, we have:

$$T(n) = 2^k . T\left(\frac{n}{2^k}\right) + (2^k - 1).c$$

for $k = \lg(n) \rightarrow \log_2(n)$

$$T(n) = 2^{\lg(n)} . T\left(\frac{n}{2^{\lg(n)}}\right) + (2^{\lg(n)} - 1).c$$

trying to make $\left(\frac{n}{2^k}\right)$ equal to 1,

cuz $s(1)$ is given.

$\therefore \frac{n}{2^k} = 1$, so $\boxed{n = 2^k}$

$$\Rightarrow T(n) = n . T\left(\frac{n}{n}\right) + (n-1).c$$

$$\Rightarrow T(n) = n . T(1) + nc - c$$

$$\Rightarrow T(n) = n.c + nc - c$$

$$\Rightarrow \boxed{T(n) = (2n-1).c}$$

① $\log_2 8 = 3$ means that $2^3 = 8$.

② $2^{[\log_2 8]} \Rightarrow 2^3 = 8$

$\boxed{b^{(\log_b a)} = a}$

③ $$T(n) = \begin{cases} c & , n = 1 \\ 2T\left(\frac{n}{2}\right) + c & , n > 1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + c$$

$$\Rightarrow T(n) = 2\left[2T\left(\frac{n}{2\times 2}\right) + c\right] + c \Rightarrow T(n) = 2^2 . T\left(\frac{n}{2^2}\right) + 2c + c$$

$$\Rightarrow T(n) = 2^2\left[2T\left(\frac{n}{2\times 2^2}\right) + c\right] + 3c \Rightarrow T(n) = 2^3 . T\left(\frac{n}{2^3}\right) + 7c$$

$$\Rightarrow T(n) = 2^3 . \left[2T\left(\frac{n}{2\times 2^3}\right) + c\right] + 7c \Rightarrow T(n) = 2^4 . T\left(\frac{n}{2^4}\right) + 15c$$

$$\vdots$$

$$\Rightarrow T(n) = 2^k . T\left(\frac{n}{2^k}\right) + (2^k - 1)c$$

so far, for $n \geq 2^k$, we have:

$$T(n) = 2^k . T\left(\frac{n}{2^k}\right) + (2^k - 1) . c$$

④ $$T(n) = \begin{cases} c & , n = 1 \\ a.T\left(\frac{n}{b}\right) + cn & , n > 1 \end{cases}$$

$$T(n) = a.T\left(\frac{n}{b}\right) + cn$$

$$\Rightarrow T(n) = a.\left[a.T\left(\frac{n}{b.b}\right) + c\left(\frac{n}{b}\right)\right] + cn \quad \Rightarrow T(n) = a^2.T\left(\frac{n}{b^2}\right) + \frac{a}{b}.cn + cn$$

$$\Rightarrow T(n) = a^2.\left[a.T\left(\frac{n}{b^2.b}\right) + c\left(\frac{n}{b^2}\right)\right] + \frac{a}{b}.cn + cn$$

$$\Rightarrow T(n) = a^3.T\left(\frac{n}{b^3}\right) + \frac{a^2}{b^2}.cn + \frac{a}{b}.cn + cn$$

$$\vdots$$

$$T(n) = a^k.T\left(\frac{n}{b^k}\right) + \frac{a^{(k-1)}}{b^{(k-1)}}.cn + \frac{a^{(k-2)}}{b^{(k-2)}}.cn + \ldots\ldots + \frac{a^0}{b^0}.cn$$

$$\Rightarrow T(n) = a^k.T\left(\frac{n}{b^k}\right) + cn\left[\frac{a^{(k-1)}}{b^{(k-1)}} + \frac{a^{(k-2)}}{b^{(k-2)}} + \ldots + \frac{a^2}{b^2} + \frac{a}{b} + 1\right]$$

for $n \geq b^k$, we have the above.

$\log_2 8 = 3$
$2^3 = 8$

$\therefore$ for $k = \log_b n$,
$n = b^k$,
$T(n) = a^k.T(1) + cn[$

3 answers for $a = b$, $a < b$ & $a > b$

Recording -
- definition of rec.
- diff b/w subst. & iteration
- def^n.
- Master thm.
- Recursion - def.
- $T(n) = T(n/4) + T(n/2) + n^2$ slide
- Again Master Thm. MAIN
- mid term
- BACK to Master THM.

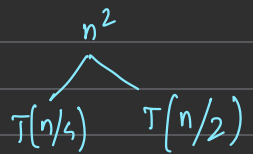
24th Sept -

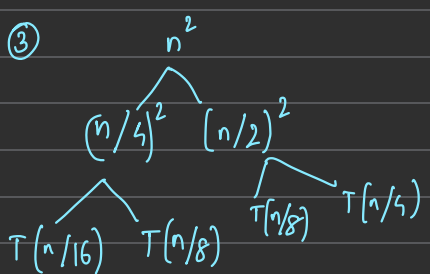
$T(n)$ - no. of operations algo will take to solve a problem size 'n'

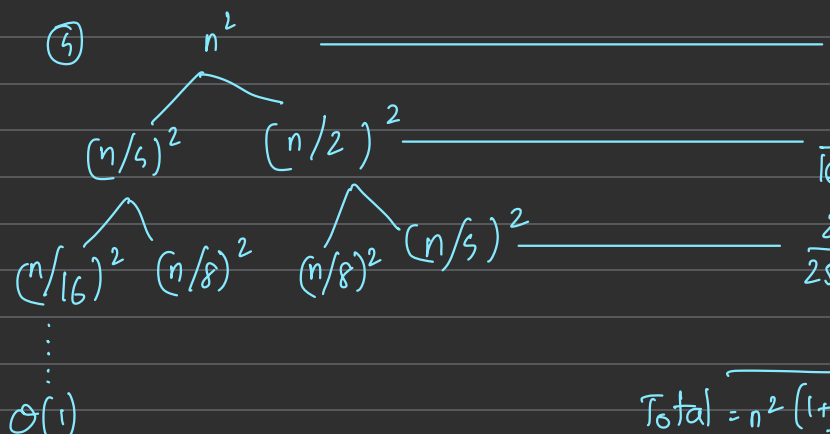
RECURSION TREE -

$$T(n) = T(n/4) + T(n/2) + n^2$$

① $T(n)$

②


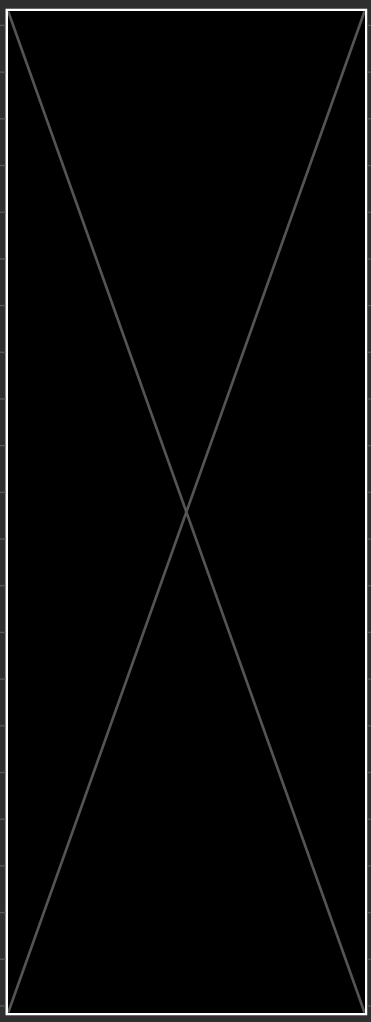
③


⑤


$$\text{Total} = n^2 \left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \dots\right)$$

$$\Rightarrow \text{Total} = O(n^2)$$

Nasler Theorem :



Effort incr
so max at bottom

Effort decr.
so max at Top

26th Sept:

$$T(n) = \underbrace{aT(n/b)}_{\text{bottom level effort depends on this}} + \underbrace{f(n)}_{\text{top level effort}}$$

process to div. & conq. ↓ ; ← effort to div. & conq.

=> other levels depend on both

Ex- $T(n) = 4T(n/2) + n^3$

bottom level : $n^{\log_b a}$ $\quad n^{\log_b a} = n^2$

top level : n^3 $\quad f(n) = n^3$

$\therefore f(n) = \omega(n^2)$ $\quad [f(n) > n^2]$

instead of using ω (small omega), to use big omega Ω,

we say there's a constant ε → epsilon

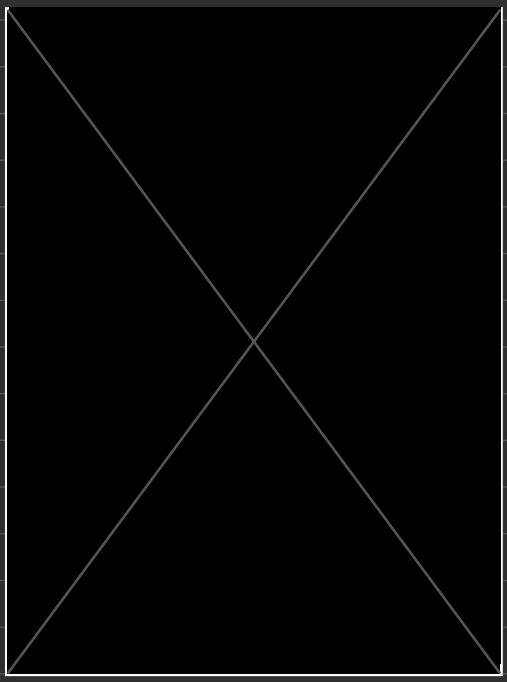
$f(n) = \Omega(n^3)$

$\therefore f(n) > \Omega(n^{2+\varepsilon})$, where $\varepsilon = 1$

regularity condition

easy way to think is, look at $n^{\log_b a}$, that has to be term in Ω or O.
add or remove an epsilon based on $f(n)$ value.

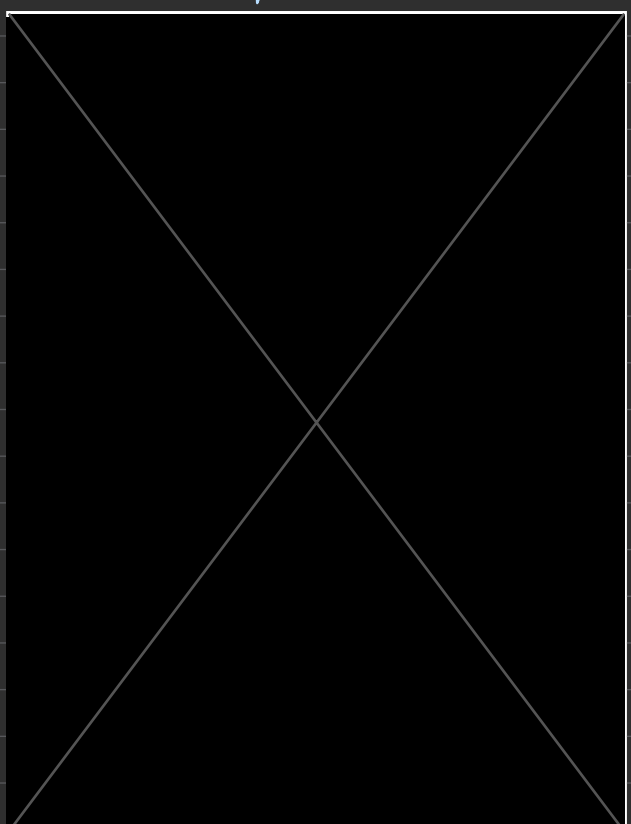
Substition Method:



1st Oct-

Substitution method

Divide & Conquer:



Maximum subarray

3rd October:

Max sub-array

- $O(n^2)$
- → keep track of max with running max.

3 8 -1 4 -5 7 -6 9

[3] [38] [38-1] [38-14] ....

[8] [8-1] [8-14] [8-145] ....

⋮

Can we do better in Brute Force ? NOPE

DIVIDE & CONQUER: ↑ same prob.

- split into 2
- find max subarray
- cant just STOP there by comparing 2.

3 8 -1 4 -5 7 -6 9

↓ M.SA1 ↓ M.SA2

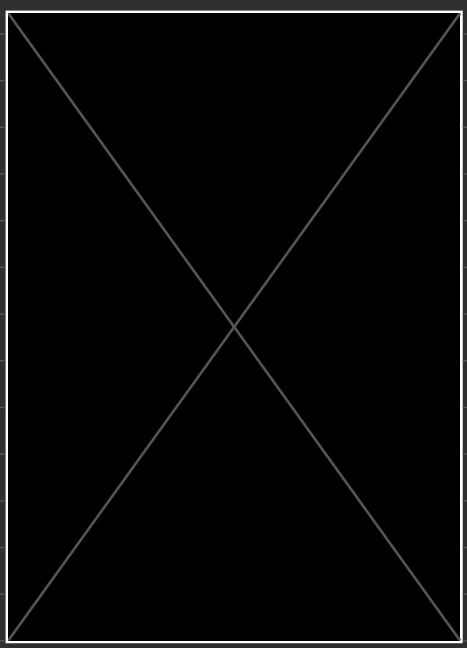
cannot just compare, cuz cross border subarrays not considered.

· need B → split ARRAY into 2, 2 1 crossing subarray

→ byte code

→ observation (1-1minute)

$O(n) \Rightarrow T(n) = 2T(n/2) + n$



N_1, N_0, N_2

8th Oct-

Cadence Algo. → Dynamic Prog. (later in detail)

3 5 -6 1 8 -9 7 4

start from right

curr max S-
curr arry -
↙
max sub. arry. touching boundary

4 by itself is max subarray
→ if max subarray so far ≤ 3

4 is part of max subarray
→ if 7 is part of max subary

THINK!

if curr max = 9, could be anywhere

M → R
M → R+C

@ -3 5 -6 1 8 -9 7 4

i) max S- -3, 5, 5, 5, 9, 9, 9, 11
curr arry - -3, 5, -1, 1, 9, 0, 7, 11

keep!
cuz no. by itself

→ if curr arr is negative, throw it away & use no. by itself

Matrix Multiplication
(Div & Conq.)

Assignment-2:

100, 113, 110, 85, 105, 102, 86, 63, 81, 101, 94, 106, 101, 79, 94, 90, 97

29th October: