WORK ॐ

8 Jan, 2023
Monday

Lec-1 :-

d/w apostriory analysis → writing & executing program

(apriory analysis)

- One way to calculate TC is no. of operations.

Algorithm → very general way of describing any problem
- Pseudo code
- Program

(gives theoretical TC)

give input → also like algo, but more specific wrt program

but not specific to any programming language

Input a, b ;

step by step procedure to solve any problem

usually

- Properties Algorithm must posses :-
  1. must have input / output
  2. should terminate

for eg. square root of no. → can go ∞, but u have to stop it somewhere.
→ some ending criteria must be there, as will use ∞ resources d/w, bad ∞ toh hai nahi.

- why want algorithm?
  ↳ To solve some problems.
- always precheck input.
  ↳ agar pehle hi sorted, fir 'algorithm kyu lgaayeinge?

Binary search → totally useless, Jab tak data sorted na ho!
∴ so Data structure is important.

⊙ TC can get by: ① time clock
time end − time start
something....

but this gives time on your system only, not a generic method.

T = T + 1 etc.
↳ is generic TC.

Analysis → Design → Analysis → Redesign .. etc.. jab tak optimized solution na mile!

Profiling → Apostiory analysis
(yeh hai) ↳ (Implementation wise checking)

Input known → [System known] → Output Unknown?

Analysis

design should come first → as don't know algo.

Complexity of system to produce output :-

① time it takes to produce output
② physical req. of system " " ",
↳ i) computer cycles
ii) memory

Input known → [System Unknown?] → Expected output known

DESIGN

↳ How to get that output & how many ways in?

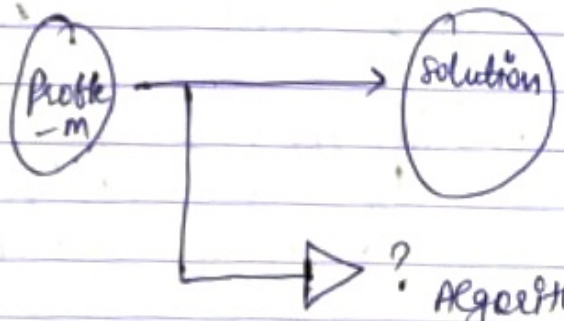
Realizing the System :- (1) DESIGN
(2) SYNTHESIS problem.

Algo → is transfer / mapping function
: method to solve a problem.

Input : Problem
Output : Solution
System : prblm solver algorithm.

Problem → | Algo | → Solution

Technically,
algo is precise method to be implemented on a machine to solve problem.



Algorithm → transformation
(mapping a problem to solution)

sorting → also one type of transformation,
as mapping numbers to diff - 2s places!

similarly, ascending, descending → also type of transformations.

Problem : Input specifications . . . [I] → possible set of inputs.
Solution : Output (results) . . . [O]

$[O] \propto [I]$

$[O] = f([I])$

Transfer function.

WORK

finite executable
When f is sequence of unambiguous steps, then it is called Algorithm.

Q. Solve :- $x^2 - 7x + 12 = 0$

$ax^2 + bx + c = 0$

↳ a, b, c are input, not x!

Input : $a = 1, b = -7, c = 12$

Output : roots $= x_1, x_2$

$x_1 = f_1(a, b, c)$

$x_2 = f_2(a, b, c)$

$f = (f_1, f_2)$: Algorithm

f_1, f_2 → may depend on nature of a, b, c.

i.e.

$f = (f_n, f_1, f_2)$

f_n = nature (a, b, c)

= discriminant fn (d)

$= b^2 - 4ac$

Case1 : $d = 0$

$f1 = f2$

$x1 = x2 = -b/a$

Case2 : $d > 0$

$f1, f2$ different

$x1 = \frac{-b + \sqrt{d}}{2a}$

$x_2 = \frac{-b - \sqrt{d}}{2a}$

Case3 : $d < 0$

$f1, f2$ → complex conjugate roots.

$x1 = x + iy$

$x_2 = x - iy$

where, $x = -b/2a$

$y = \sqrt{-d}/2a$

WORK
30
MM.

In all 3 cases, computational time is different.

Structured representation
↳ Use pseudocode

Input : a, b, c
Output : x1, x2

```
{  d = b^2 - 4ac;
if d=0 then x1 = x2 = -b/2a;
else if d>0 then
    {  x1 = (-b + √d)/2a;
       x2 = (-b - √d)/2a;
    };
else
    {
    x = -b/2a ; y = √-d/2a;
        x1 = x + iy;
        x2 = x - iy
    }
```

What is no. of operation for :- (4/10)
i) Case 1
ii) case 2
iii) case 3

algorithm :
starts with
follows sequence of
produces
and

definiteness → clarity regarding what should be done
unambiguous
INPUT
FINITE EXECUTABLE STEPS
OUTPUT
TERMINATES.
↳ effectiveness & executability indicating finite amount of resources
Time
Space

WJMK 3
= 3M.

⊙ 5 characteristic properties of an Algo :-

1) Input
2) Output
   ⇒ It should have complete procedure, that :-
3) Termination
4) Definiteness
5) Effectiveness (should run within available resources)

* executing with piece of pencil & paper (imp.)

WJMK :)
:)mm.

11 Jan 2023
Thursday

Lec-2 :-

- In terms of no. of operations, steps u take by writing code should be analysed called apriory analysis.

Apostriory analysis → analysis taken after writing code.

- clock cycle • (Time = clock ends − clock starts)
  ↳ System dependent (^_^)

- Division more expensive than multiplication,
  " expensive than $+^n$ and $-^n$.
  ↓
  but we assume all take same ~~unit~~ of time = 1 unit of time.

- Computation time & Computation Space both are imp..
  (as ↓ must consume less resources).
  ↑ more → ~~compute~~ algorithm capability reduces.

- If computation is definite (finite) step. (as able to compute it)
  ↳ is it effective step?
  ↳ as if $d=0 \Rightarrow \sqrt{d}$ not computed
  $d>0 \Rightarrow \sqrt{d}$ is defined
  $d<0 \Rightarrow$ modifies as $\sqrt{-d}$
  ↳ Even after this explicit computation, it is not effective step (as Ramanujan hoga where to terminate ;)
  as $\sqrt{2} = 1.333333\ldots$ (whole resources will get exhaust)

(Prime no. → ① $i = 1$ to n
② $i = 1$ to $n/2$
③ $i = 1$ to $\sqrt{n}$)

PIPELINING :

WJMK

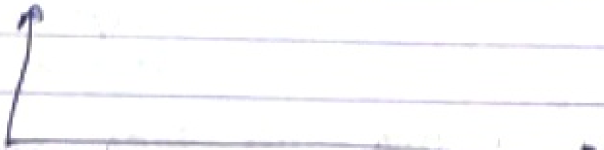
To compute x_1, x_2, → compute d.
↳ can compute concurrently by PARALLEL ALGORITHM.

⊙ Factors for Analysis :-

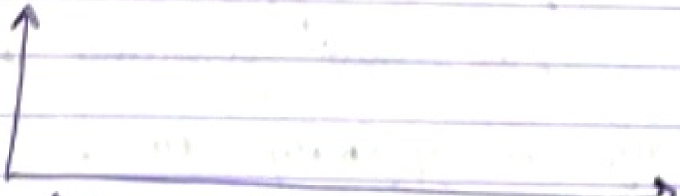
(Individual analysis) ① Quality of roots – Accuracy in values of roots

(analysis in combination) ② Overflow, underflow detection & prevention
③ computation time
④ Space req. for computation.

Performance Analysis



Accuracy of soln Vs Accuracy in $\sqrt{d}$



Accuracy Vs number of significant digits in a, b, c

⊙ Causes of Computation Pitfalls :-
① multiplication & division
② Addition & subtraction
③ sq root computation

(Imp) ⊙ Same algo can behave differently on different no. of operations. (like best case, worst case, avg case)

WJMK c/ 3v =bMo

(yeh bhi bnaana code k saath) (then multiply both loops.) → freq of opr's | multiply both → no. of add → apriory time complexity

Q. Swap 2 numbers.

no of operations

①
```
{ read a, b;
    t ← a;
    a ← b;
    b ← t;
}
```
→ more variable, less computation!

②
```
{
  read a, b
    a ← a+b
    b ← a-b
    a ← a-b
}
```

→ This shows trade off b/w space (no. of variables) & time (no. of operations).

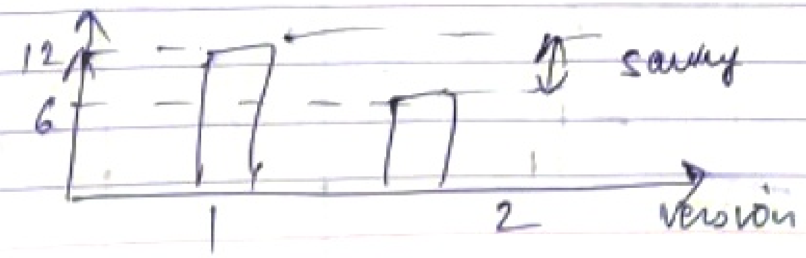
→ Prove that both algo works.
and which suffers more computation pitfalls?

	# Assignment	no. of arithmetic operations	# Variables
①	3	0	3
②	3	3	2

multiplication → n × m.

Version 1: p ← 0
i = 1 to n
{ p = p + m; }

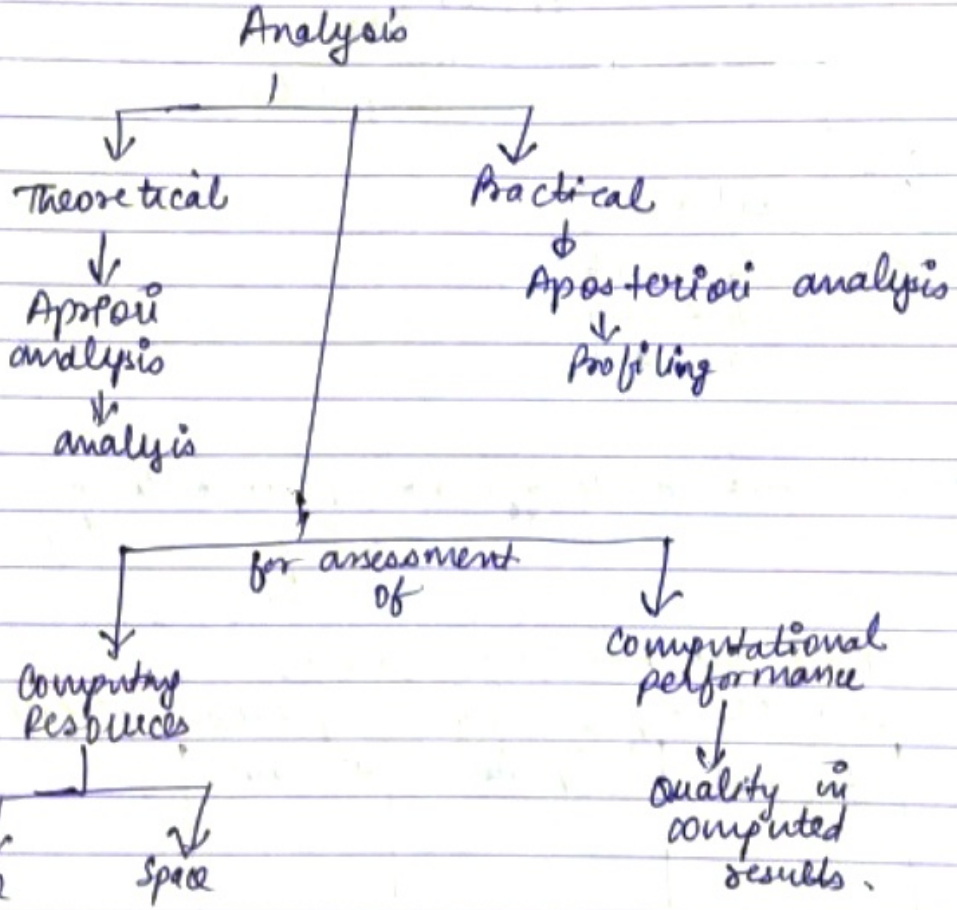
version 2:- p ← 0
i = 1 to m.
{ p = p + n; }



WORK

Profiling :- process of executing a correct program on data sets & measuring the time & space it takes to compute the results.

(testing)



step	operations	Time	frequency
for (i ← 1 to n)	i = i+1 ← 1 +1 i > n ? >1	03	n+1
do p ← p+m	← 1 +1	02	n
p ← 0	←	01	1

WJMP & Emm.

Computing time

$t \propto 1 \cdot 1 + 3 \cdot (n+1) + 2 \cdot n$

$t \propto 5n + 4 \rightarrow$ theoretical / apriory TC

↑ if no input, still 4 operations honge! (means 4 se kam nhi ho skte kafhi thi, so it is bounded by 4).

$t \geq 4$.

w/o even executing code

TC shows nature of algo. (more inp. than time) → linear in nature

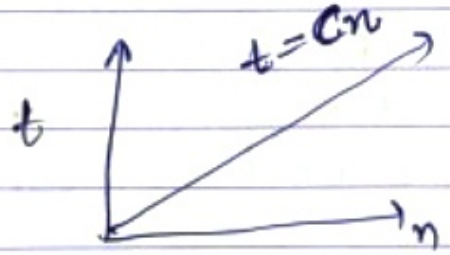
if n is very large, then

$t \propto 5n$

$t \propto n$

$t = O(n)$.

$t \leq Cn$



int n;

n = 11;

for (int i = 0; i < n; i++)

{ arr[i] = n; → 1 → n

}

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

$t =$

WOMK ☺ WOMK ☺ WOMK ☺

15 Jan 2023
Monday

lec-3 :-

* $\log n < n < n^2 < n^3 \ldots$
don't prefer algo with TC > $O(n^3)$.

① TC is used to compare different -2 algo.

② Nature of TC depends on ~~no~~ of lines in algo? X.
size of input & output,
mostly on size of input ✓
no. of operations X
machine X

After writing code :-

```
Time ← 0;
p ← 0;
Time ← Time + 1;
for i ← 1 to n do
{
    Time ← Time + 3;
    p ← p + m;
    Time ← Time + 2;
}
```

nature of this Time should ∝ (or ~~not~~ match)
with nature of approx time complexity,
o/w problem.
↳ How to match it ??

WORK ಓಂ ನಮಃ.

for eg: Binary search,

Theoretically, TC = O(log n)

↓ → Practical

n	log(n)	Time (Do print karaayenge screen pe)
10	.	.
100	.	.
:	.	.
.	.	.

↳ both should match for diff - 2 (n).

→ Then sahi hai algorithm.

Or multiplication ← add bigger no. ; smaller no. of time.

↓

for that if-else statement lgaani hogi

↓

TC will ↑.

:

TC = O(min(m, n))

↑

linear

↳ so algo is linear algo.

multiplication Using lookup Table :-

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2		4	6	8	10	12	14	16	18
3			9	12	15	18	21	24	27
4				16	20	24	28	32	36
5					25	30	35	40	45
6						36	42	48	54
7							49	56	63
8								64	72
9									81

WOMK :
3 mm.

How to Implement it ?. DS ?

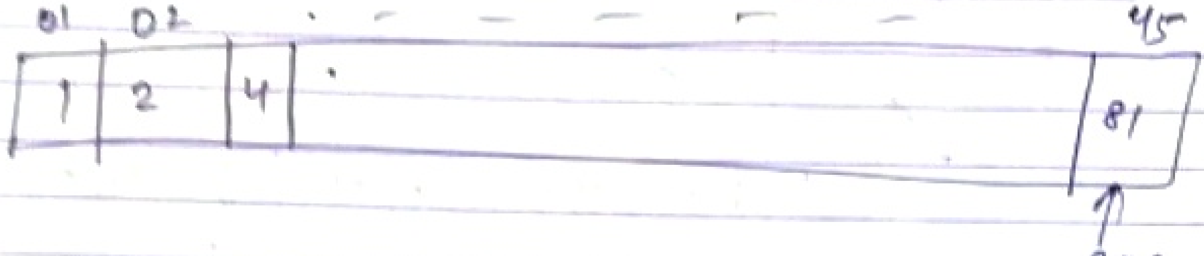
→ access & pointers

① arr [10][10] → bad Idea ! not effective !

↑ many space (:-/) = $O(n^2)$

Don't go 2D, 1D mein socho !

⇒ Do like :-



Study Instances :-

$4 * 3 = 12$ → position :- 9

$6 * 2 = 12$ → position :- 17

$5 * 4 = 20$ → " :- 14

$m * n = mn$ → position = ?.

m^{th} table commences from position $\frac{m(m-1)}{2} + 1$

as 1st table :- " " 1.

3rd ——— 4

m.1 → found on position = $\frac{m(m-1)}{2} + 1$

m.2 → ——— = $\frac{m(m-1)}{2} + 2$

⋮

m.n → ——— = $\frac{m(m-1)}{2} + n$

WORK $\frac{3}{2}$ TOM.

here SC = $O\left(\frac{n^2}{2}\right)$
access → one pointer

→ ∃ Time - space tradeoff!

a*b Case1 :- a, b ∈ IR
(i) fractional form
(ii) exponent "

Case2 :-
(i) look up table
(ii) ??

981 * 1234 = ? (using look up Table)
↳ 2 ways ???
① →
② ←

① can parallel implementation be done? Yes!
② can fill individual rows concurrently? Yes!
③ all rows ans are independent?
↳ Yes!
↳ so do → 981 * 1
→ 981 * 2
→ 981 * 3
→ 981 * 4
→ and add them!
↳ so give 4 things to 4 processors.

④ can we add each column concurrently?
↳ NO!
→ as dependent!
→ carry (:')

Work is worship.

→ Always think of parallel implementation, as we've multicores, multiple processes.

↓

to utilize them !

3rd way of multiplication :-

i) Write multiplicand and multiplier side by side.

ii) Create 2 columns one under each operand :-

① Divide (truncate) the no. in left column by 2. (Halve the no.)

② Double no. in right hand column by adding it to itself

iii) Cross out each row where no in left column is even

iv) Add up the numbers.

Hand Simulation :-

981	1234	odd	1234
490	2468	even	discard XXXX
245	4936	odd	4936
122			
61			
30			
⋮			
1	631808	odd	631808

→ add ans.

WORK in progress.

Here parallelism is not possible.

4th way :- (Divide & Conquer approach).

→ Both multiplicand & multiplier should be ≥ 1.

→ Both should have same no. of digits.

→ no. of digits should be power of 2.

$981 * 1234 = 0981 * 1234$

Instance	multiplication	shift	Result
1	09 * 12	4	1080...
2	09 * 34	2	306..
3	81 * 12	2	972..
4	81 * 34	0	2754
			+ 1210054

↓

agar 4 digit multiplication nahi aati, toh 2 digit kro, nahi aati toh 1 digit kro (wo toh aati hai!)

↳ Recursively calling function.

→ Can we show trade-off b/w these 4 algo!

81 * 12

⇒

	shift	Result
8 * 1	2	8..
8 * 2	1	16.
1 * 1	1	1.
1 * 2	0	+ 2
		972

WJMK ☺

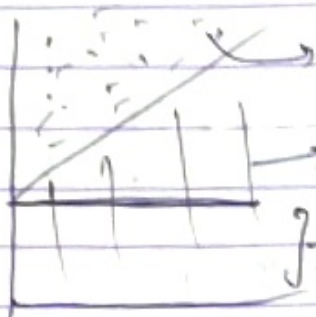
Lec-4 :-

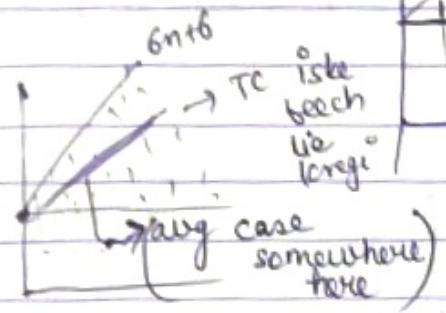
18 Jan 2023 | Thursday

scanf → 2 operations → read & fetch

arr[i] == key → 2 operations.

Linear Search





O(1) → best ~~worst~~ case

O(n) → worst case

What is average case ???

↑ consider every case,

theoretically ← { jab key=1 pe ho, jab key=2 pe ho. . . .
sbko add kro, 'n', divide by

Practically
↳ kuch trials kro, har trial mein random key
dekho, jo khin bhi ho
skti hai → TC dekho,
sbko add kro, ÷ by no. of trials →
get practical avg case TC.

(must do in exams)

↳ obtain this theoretically & practically, in both way

and plot graphs for both.

t = t + (no. of operations take, (har line ke baad))

WORK ϵ = 5 ♡♡.

Merge sort
↳ Divide & Conquer
&
(conquer back again)

& 3rd thing ⇒ use "clock_time"

n			
ct			

when element not found or found at last position, $TC = O(n)$.
↑
both are Worst case!

when $n = 1$
↑ Best case & Worst case is same!

BINARY SEARCH:- (Approach :- Divide and Discard) (convergence very fast)
→ half of array
↳ pre-requisite : array should be sorted
↳ $O(N \log_2 N)$

assuming 0-based indexing :→

```
low = 0, high = N-1;
while (low <= high)
{
    mid = (low + high)/2;          → (low + high) << 1;
                                     ↑ saves computation
    if (arr[mid] == x) {
        done }
    if (arr[mid] > x)
        high = mid-1;
    else
        low = mid+1;
}
```

~~(no. of elements that can be searched in 1 operation)~~

In 1st operation → $(N/2)^{th}$ element can be searched only)
In 2nd operation → $N/4 + 3N/4$
In 3rd operation → $\frac{N}{8}, \frac{3N}{8}, \frac{5N}{8}, \frac{7N}{8}$
In 4th operation → 4^{n-1}

(yeh 2 elements mil jayenge in 2 operations)

$T(n) = 2T(n/2) + (\ \ )$
- 2T → divide (as har baar half mein tod the hain)
- then put n ko n/2 & so on
- ( ) ↳ conquer (here discard)

WOMK $= \frac{?}{3}$ mm.

as n ko divide karte jaa rahe hain & can do log n times only!!

(log n)

in 'N' operations $\rightarrow N * 2^{\log N - 1}$ → no. of elements can be searched!

T = Total operation

$= 1*1 + \dots + N * 2^{\log N - 1}$

$2T = 2 + \dots + N * 2^{\log N}$

subtract both →

$T = 1 + 2 + 4 + 8 + \dots + N * 2^{(\log N)}$

$= 1 * \frac{(2^{\log N} - 1)}{(2-1)} + N * 2^{\log N}$

so,

$$\text{avg } T = \frac{2^{\log N} - 1 + N \log N}{N}$$

$= 1 - \frac{1}{N} + \log N.$

so, $T = O(\log N)$

&

sort bhi karoge pehle ⇒ $T = O(N \log N)$

↑ is done once, but searching has to be done many times!!

↓ so hum karte hain, chahe costly!!

7/10

Bubble sort,

Insertion sort!

→ also prove correctness of algo

↳ apriory analysis

↳ graph

↳ TC

↳ profile matching