Introduction

An algorithm → word given by man Al-khorezmi is a sequence of unambiguous instructions for solving a problem, i.e. for obtaining a required o/p for any legitimate (valid) input in a finite amount of time

problem
↓
algorithm
↓
i/p → [computer] → o/p

Properties / characteristics of algo

- Non ambiguity
- range of i/p
- same algo → diff representation
- problem → diff algo

1. Finiteness :- terminates after finite steps
2. Definiteness :- rigorously and unambiguously specified
3. Clearly specified i/p :- valid i/p are clearly specified
4. Clearly specified o/p :- Expected o/p :-
   can be proved to produce the correct o/p given a valid i/p
5. Effectiveness :- steps are sufficiently simple & basic.

Note :-

1. Algorithm can be represented in various forms
2. Several algorithm for solving the same probl may exist (search < linear, binary, hash)

3. Different design techniques exists for designing the algorithms (linear search can be done in many ways)

4. Data of same problem can be represented in different ways.

Problem of finding GCD (Euclid's algorithm) → eventually stops because gcd no always gets reduced and does not become negative

step 1 :- If n=0 return m as answer and stop
else proceed to step 2

step 2 :- Divide m by n assign the value of remainder to r.

step 3 :- Assign value of n to m and r to n
go to step - 1

Pseudocode v/s Algorithm

↓ ↓

If knowledge of programming is there then this is used.

It can be easily converted to code

Anybody can write this in form of steps step by step

Pseudocode for Euclid's algorithm

// compute gcd (m, n) by Euclid's algorithm
// i/p :- Two non negative not both zero int m & n

// o/p :- gcd of m & n

while $n \neq 0$ do

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m

① $(\overset{m}{6}, \overset{n}{24})$

$24 \neq 0$

$r \leftarrow 6 \bmod 24 = 6$

$m \leftarrow 24$

$n \leftarrow 6$

$6 \neq 0$

$r \leftarrow 0$

$m \leftarrow 6$

$n \leftarrow 0$

$0 == 0$

return m i.e. 6.

② $(\overset{m}{24}, \overset{n}{6})$

$6 \neq 0$

$r \leftarrow 0$

$m \leftarrow 6$

$n \leftarrow 0$

$0 == 0$

return 6

Worst case :- gt becomes half for every iteration

Best case :- gt becomes zero in first iteration
i.e. reduced by its own value

Rest it can get reduced by more than half too
Everytime n is reduced by modulo operation.

Time complexity :- $\log n$
i.e. by 1/2 (in worst case)

grows very slowly

So worst case is for relative prime nos

Eg 13, 7

4, 9

7 13

r ← 7	7 != 0	c != 0	6 != 0
m ← 13	r ← 6	r ← 1	r ← 0
n ← 7	m ← 7	m ← 6	m ← 1
	n ← 6	n ← 1	n = 0

4 times → depends on value taken but grows slowly

GCD = 1

∴ This is efficient algo due to use of modulo operator.

Problem of GCD (by consecutive integer checking method)

i/p type is both non zero +ve nos
i.e. if one of o/p is zero it does not work

Step 1:- Assign the value of min{m, n} to t

Step 2:- Divide m by t.
If remainder is 0 go to step 3.
else go to step 4

Step 3:- Divide n by t.
If remainder is 0 return t as ans & stop
else goto step 4

Step 4:- Decrease t by 1 go to step 2.

Eg:- gcd (6, 24)

$t \leftarrow \min(6, 24)$

$t \leftarrow 6$

$m \% t = 0$

$n \% t = 0$

return 6

Eg:- gcd (7, 13)

$t \leftarrow \min(7, 13)$

$t \leftarrow 7$

$m \% t = 0$

$n \% t \neq 0$

$t --$ ⟹ (6)

$m \% t \neq 0$

$t --$ ⟹ (5)

$m \% t \neq 0$

$t --$ ⟹ (4)

⟹ (3)

⟹ 2

⟹ 1

$m \% 1 == 0$

$n \% 1 == 0$

return 1

(7) times → depends on value of m & n

Step 1:- Find the prime factors of m

Step 2:- n

Step 3 Identify common factors ~~and return it as gcd~~

Step 4: xy all of them & return product as gcd

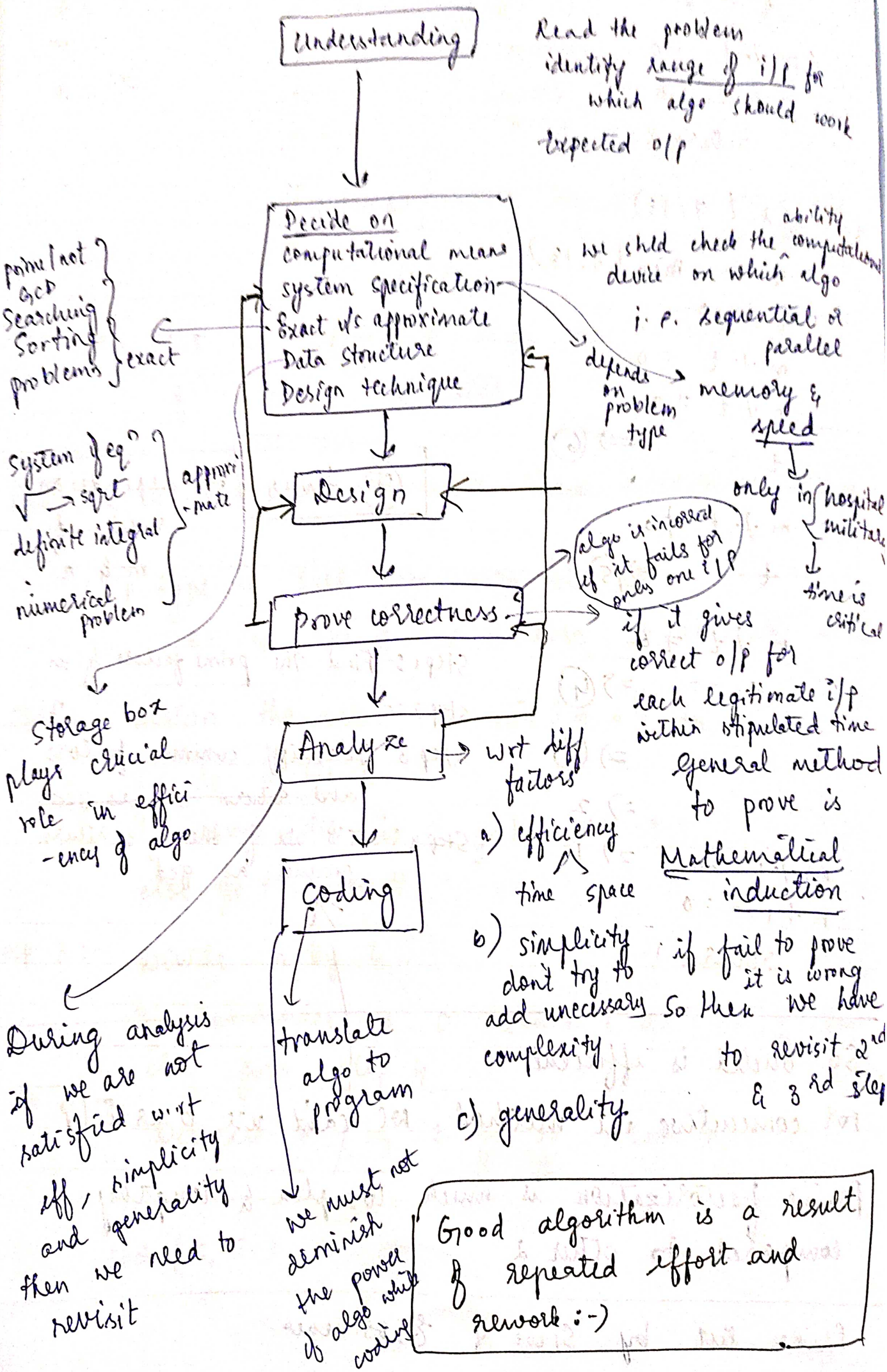
So Euclid is efficient

For consecutive int method we can't use 0 as i/p

Prime factorization is much complex & lengthy compared to other 2

Prime list by Sieve of Eratosthenes

Fundamental steps in designing algorithm



→ sequential algorithm → one operation at a time
→ one by one
→ von Neuman systems.

→ thread = execution path
→ modern system = multicore architecture → executes operations concurrently
→ parallel operation

Parallel processing → Algorithms which are designed to work on these are called parallel algorithm

parallel algorithm → We can execute the algorithm to program parallelly

OpenMp → application program interface
↓
gcc -
openmp.h
↓
used in Divide & Conquer algorithms.

Design technique :- general model for solving problem → general approach to solve problems algorithmically that is applicable to various problems from diff areas of comp

↓ Eg :- Brute force
Transform & conquer.
many design tech for algo to ↑ eff
Divide & conquer...

Methods to represent algo / express algo

→ pseudocode = mixture of natural language & programming language
→ flowchart
→ Natural language (step by step)

time efficiency → time required to complete the program
space efficiency → amount of space occupied by program of algo while its executing.

* space
- data space
- instruction space
- environment space
  - needed to resume the suspended function / program

* Generality :- → problem it solves & i/p it accepts
  - generality of problem the algo solves
    i.e. gcd of two nos can be even used to solve the checking of relative primes

* Analysis
  - Theoritical
  - Empirical.
    - through Analysis framework.
  - Practical

* while coding we should not minimize the power of algo designed
  - Done by code optimization

Algorithm optimality is imp
i.e. what is the minimum amount of work / effort algo does to solve problem

Types of problems
- Diff algo exists for same problem
- Diff design techniques for algo.

- → sorting
- → searching
- → String processing
- → Graph problems
- → combinatorial problems
- → Geometric problems
- → Numerical problems

① Sorting : Arranging list w.r.t ascending / descending order

↓ why required ?

→ operations can be easily performed on sorted list

list of nos
list of strings
list of char
list of records → using key values.

Eg: ~~Bubble sort search~~
Binary search.

Sorting

stability ; Implacebility → sorting algo is inplace if it does not require extra space then it is inplace.

stability

↓

sorting algo is stable if the relative position of 2 equal elements is maintained in ~~sorted list~~

Say we have 2 equal elements at position i, j where $i < j$

Eg: 10 5 8 2 15 18 2
3 6
i j

then after sorting in sorted list 2 equal elements appear in i' & j''

then if $i' < j''$

then algo is stable.

② Searching

a) Linear search → Best case 1st pos
Worst case last pos

b) Binary search → best case middle
Worst case 2 extremes

↓ list must be sorted.

There is no stability & inplacebility issues in search.

Problem is :- The underlying data changes i.e no of elements keep changing

↓ As in insertion & deletion

It is solved using choosing appropriate data structure

Eg:- Binary search using Binary search tree

Binary search
↓
efficient using only BST ~~only~~ in linked repr^n

Binary search tree
↓
changes dynamically
it can become skewed one
So depth ↑
hence time ↑ for search
So we must make sure it is balanced BST.

→ organizing large volume of data for search is tedious

---

③ String processing :- Searching of pattern in text

Eg:- Search engine.
Plagiarism check
Bio informatics → DNA seq matching
forensic appl^n.

④ Graph problems :-
Transformation problems
communication problems

⑤ Combinatorial problems
in combinatorial objects
Involves permutation
Combination
power sets

Eg:- job scheduling problems
job assignment problems

⑥ Geometric problems
Involves points
line
polygons
} geometric objects

Eg:- convex and shortest path

⑦ Numerical problems
Involves mathematical objects
↓
mostly solved approximately.

n! assignments

24 ↑ by 1
120 24 to 120
size grows fast

$\begin{cases} P_1 & P_2 & P_3 & P_4 \\ J_1 & J_2 & J_3 & J_4 \end{cases}$

We can arrange in 4! ways i.e. one person only one job with less cost, time

Fundamentals of Analysis of Algorithms.

1) Efficiency
↓
if it takes less time to execute and consumes less memory space.

performance is measured based on
1) Time complexity
2) space complexity

2 Approaches ← Theoritical Analysis → before we have code in hand we shld analyze
Empirical Analysis → we have program with us by checking time, space, i/p complexity...
← in time.h start time and end time.

Analysis framework

In early days both time and space were important
In modern days space is not of much concern
but time constraint is critical even now.

so we only check time efficiency
But same spacework is suitable for both time & space

Analysis framework

1) Measuring input size
→ Most of programs run longer for larger i/p
Eg:- more elements in search set
more time to search.

So we should investigate efficiency as a function of parameter called input size.
input size & time taken.

Measuring size of i/p depends on type of problem.

→ Eg:- Straightforward
- Input size for sorting is no of elements
- —"— searching is no of elements
- finding min element —"— —"—
- evaluating polynomial = degree of polynomial

→ For some problems there is a choice for i/p size

Matrix multiplication
- no of elements
- order of matrix

} for square matrix any one is ok

But for rest order mxn is needed.

But efficiency for each is diff quantitatively -erent for same problem.

→ For some problems i/p size depends on type of operation the algo performs

Eg: Spell check algo.

if we do char by char, i/p size = no of char
word by word = no of word

→ For some problems i/p size is determined by ~~one~~ more than one parameter

Eg:- BFS & DFS graph traversals

↓ Depends on no of
- vertices
- edges

} Both

→ For some problems i/p size depends on value of N (inpu

Eg:- Factorial → depends on value of n
Prime → —"— —"— —
gcd → m & n.

i.e it depends on Number characteristics

→ For number characteristics algo the i/p size is determined using no of bits required in binary repre of that number. It is called magnitude.

$$b = \log_2 n + 1$$

b → magnitude. n → No of bits in the binary representation.

Eg:- $n = 10$

$b = (\log_2 10) + 1$

$= 3.321 + 1$

$b = 4$

$10 = 1010$ → 4 bits

2|10
2|5 - 0
2|2 - 1
1 - 0

Note:- All algo where it is reduced

Problem	Input size measure
Search key in n items	n
x^y 2	
a^n	n
BFS & dfs in graph	edge & vertices
Sum of N nos	N
N!	log N + 1
Largest elem	No of elements N
gcd	mag^n i.e by among larger among m & n
prime	log N + 1

---

2) Units for measuring Running time → mc, s not ok because depends of system speed & compiler speed

So, Time efficiency is analyzed by repitition of basic operation

i.e. ~~Repition~~ Repitition of basic operation as a function of input size.

Basic operation : Most important operation in the algo
↳ It is the one which is contributing the most towards running time of the algorithm.

Eg: ① if we have $+^n$ $\times^n$ $-^n$ in a loop then $\times^n$ is considered as basic operation because it takes ↑ time (repeated $+^n$)

② In nested loop the operation in the inner most loop is the Basic op
∵ It is executed most time

outer loop $\times^n$ ∴ Basic op = Division.
inner loop $\div^n$

$$T(n) \approx C_{op} C(n)$$

- $T(n)$: Running time
- C_{op} : execution time for Basic op for one particular system
- $C(n)$: No of times basic operation is executed.

Diff methods (Recursion / iteration

n = input size.

Note:-

$C(n)$:- Does not provide any info other than basic op

C_{op} :- approximately taken

We Derive a function $T(n)$ based on basic operation

if $T(n) = x$ then if speed of system ↑ by 10 times
then $T(n) = \frac{1}{10}x$. 10 times faster

Eg:- $T(n) = \frac{1}{2} n(n-1)$ → repeated

$= \left(\frac{1}{2}\right)n^2 - \left(\frac{1}{2}\right)n \approx \frac{1}{2}n^2$

$\boxed{T(n) \approx \frac{1}{2}n^2}$

Q:- How much slower algo runs if size ↑ by 2 times

i.e $\frac{T(2n)}{T(n)} = \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4$

Eg: $\sqrt{n}$ 4 times $\frac{\sqrt{4n}}{\sqrt{n}} = 2$ times slower

N^2 4 times $\frac{(4N)^2}{N^2} = 16$ times slower

2^N 4 times $\frac{2^{4n}}{2^n} = 2^{3n}$ times ⟶ very slow

(input size)

3) orders of growth of function

The performance of algo can't be defined for smaller i/p. They are almost same for smaller i/p.

Eg:

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	3.3×10^1	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{36}	9.3×10^{157}
10^3	10	10^3	10×10^4	10^6	10^9		
10^4	13	10^4	1.3×10^5	10^8	10^{12}		

Now we have 3 algo for same problem a_1 a_2 a_3 with the functions $T_1(n)$ $T_2(n)$ $T_3(n)$

Now among 3 which grows slowly is most efficient.

$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2}$$

Now if $T_1(n) = \log_2 n$

$T_2(n^B) = n^3$

then a_1 is more efficie

Basic efficiency classes

1 – constant

$\log n$ logarithmic

n linear

↓ order of growth increases

$n \log n$	$n \log n$
n^2	quadratic
n^3	cubic
e^n	exponential
$n!$	factorial

↓

Binary search tree

Different cases of efficiency

For some algo, efficiency depends on type of i/p

→ Best case eff = easiest

→ Worst case eff

→ Avg case = hardest

→ Amortized eff → eff of algo for sequence of n operations
eg: deleting from BST
↓
depends on depth of tree
depth = $\log n$ for

In some algo eff is also determined by type of i/p like

1 3 7 9 11 13 } they have diff efficiency even
15 11 9 7 3 1 } if i/p size is same in

Bubble sort → i.e. depends on type of i/p

Worst case eff : It is the eff of algo for worst case i/p.

Worst case i/p : i/p of some size for which algo runs the longest when compared to all other possible i/p of the same size

Best case effic : It is the eff of algo for the best case i/p

Best case i/p :- i/p of some size for which algo runs the shortest when compared to all other possible i/p of same size

Average case eff : eff of algo for random i/p

→ In sequential search eg, key comparision is basic operation

Eg: 3 1 5 7 9 18

if key = 3 → C(n) = 1 → constant (best)

if key = 18 → C(n) = n (here 6) → linear (worst)

$$C_{avg}(n) = \frac{n+1}{2} = \frac{\text{worst} + \text{best}}{2}$$

→ The avg no of basic op for random key we are searching. $0 \le p \le 1$

If p = probability of successful search then unsuccessful = 1-p

Let probability of an element present at i be same for all i then $C_{avg}(n)$ = successful case + unsuccessful case

$$= \left[1\frac{p}{n} + 2\cdot\frac{p}{n} + 3\frac{p}{n} + \dots + n\,p/n\right]$$

$\frac{p}{n}$ = probability of single element in n element

Now $\left(1\times\frac{p}{n} + 2\times\frac{p}{n} + \dots + n\times\frac{p}{n}\right) + (1-p)n$ (successful | unsuccessful)

$$\frac{p}{n}\left[1+2+\dots+n\right]+(1-p)n$$

∵ if element is in 1st pos we need 1 comparision

if in 2nd pos we need 2 compar

∴ so on

if p = 1 i.e. successful search

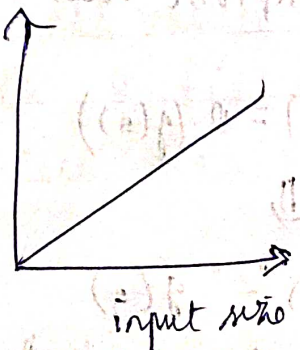
then $\frac{1}{n}\left[\frac{n(n+1)}{2}\right] + 0 = \frac{n+1}{2}$ → average case

simplified as $\frac{p}{n}\left[\frac{n(n+1)}{2}\right] + (1-p)n = \left(\frac{p}{2}(n+1) + (1-p)n\right)$

Graph plotting for order of growth

x → C(n)

n	worst case	Best case
10	10	1
100	100	1
1000	1000	1



Asymptotic Analysis

The function representing the algo's basic operation count → The growth of this function must be checked.

Analyze for larger parameter value for the function

Find the growth of function for larger parameter value

i.e analyze asymptotic

∴ The way of comparing functions that ignores constant factors and small input sizes.

∴ To compare two algo with running times $t(n)$ & $g(n)$ we need rough measure that characterizes how fast each function grows

we use rate of growth → compare functions in limit

To analyze the growth of functions asymptotically we use 3 asymptotic notations

To compare and rank the orders of growth frame work uses 3 notations

O notation ("big oh")	Ω notation ("big omega")	Θ notation ("big theta")
↓	↓	↓
asymptotic less than	asymptotic greater than	asymptotic "equality"
$t(n) = O(g(n))$	$t(n) = \Omega(g(n))$	$t(n) = \Theta(g(n))$
⇓	⇓	⇓
$t(n)$ "$\leq$" $g(n)$	$t(n)$ "$\geq$" $g(n)$	$t(n)$ "$=$" $g(n)$
for larger value of n	for larger n	for larger n

Note :- $g(n)$ is any function used for comparision
$t(n)$ is running time found using Basic op count.

Big oh (O) notation

$O(g(n))$ → is the set of all functions with same or smaller order of growth as $g(n)$ to within a constant multiple as n goes to infinity.

Eg:- $n \in O(n^2)$ → growth of n is always less than n^2
Here $t(n) = n$, $g(n) = n^2$

$100n + 5 \in O(n^2)$

$\frac{n(n-1)}{2} \in O(n^2)$

A function $t(n)$ is said to be in $O(g(n))$, denoted by $t(n) \in O(g(n))$

if $t(n)$ is bounded above by some constant multiple of $g(n)$ for "large" value of n.

i.e. if there exists a positive constant c & a non negative integer n_0 such that

~~$t(n) \leq c\,g(n)$~~ $t(n) \leq c\,g(n)$ for all $n \geq n_0$

n_0 → i/p size

Eg:- $100n + 5 \in O(n^2)$

We can fix c & initial value i.e. input size i.e. n_0

i.e. after n_0 for all values of n the growth of $t(n)$ is always less than $g(n)$

If $n_0 = 5$ LHS = 505

$c = ?$

Now $cn^2 = 505$
$c \times 25 = 505$
$c = 21$

i.e initial value should be equal after n_0 it should be le than

Next 505 = $t(n)$
where $= g(n)$

9) $n_0 = 1 \quad c = 105$

$100(1) + 5 \geq 105$

$c\,n^2 \geq 105$

$c(1) \geq 105$

$c \geq 105$

Now $100n + 5 \in O(n^2)$

consider $n_0 = 5$

$RHS = 505$

$c(n^2) = 505$

$c = 21$

Let $n = 6$

$LHS = 605$

$RHS = cn^2$

$= 21(6^2)$

$= 756$

True

later check for 6, 7, 8..

consider $n_0 = 1$

$LHS = 105$

Let $n = 2$ $\quad c(n^2) = 105$

$LHS = 205$

$RHS = cn^2$ $\quad c = 105$

$= 105(2^2)$ later check for

$= 420$ 2, 3, 4 ....

True

Now $100n^2 + 5 \in O(n^2)$

(a) consider $n_0 = 1$

$100(1) + 5$

$RHS = 105$ $\quad \boxed{c = 105}$

(b) consider $n_0 = 2$.

$LHS = 405$

Now 105×2^2

$105 \times 4 = 420 \rightarrow$ True

(c) $n_0 = 3$

$LHS = 905$

Now 105×3^2

$105 \times 9 = 945 \rightarrow$ True

Now $\frac{1}{2} n(n-1) \in O(n^2)$

$\frac{n^2}{2} - \frac{n}{2}$

Consider $n_0 = 1$

$LHS = \frac{1}{2} - \frac{1}{2} = 0$

Now we can $cn^2 = 0$

$n = 0$

Consider $n_0 = 0$

$LHS = 0$

$RHS = cn^2 = 0$

let $c = 1/2$

Now $n = 1$

$LHS = 0$

$RHS = \frac{1}{2}$ true

$\therefore$ we can take $c = 1/2$ or any

Now $n = 2$

$LHS = \frac{n^2}{2} - \frac{n}{2} = \frac{4}{2} - \frac{2}{2} = 1 \Rightarrow$

$RHS = cn^2$

$= \frac{1}{2}(2^2) = 2 \Rightarrow$

$\therefore$ True.

Now $0.0001 n^3 \in O(n^2)$

As $n \to \infty$ it will not be true

So its false

Big omega (Ω) notation

$\Omega(g(n))$ is the set of all functions with same/larger order of growth as $g(n)$ to within a constant multiple as n goes to ∞

A function $t(n)$ is said to be in $\Omega(g(n))$ denoted by $t(n) \in \Omega(g(n))$ if $t(n)$ is bounded below by some constant multiple of $g(n)$ for all large n

$t(n) \geq c\, g(n)$ for all $n \geq n_0$

Eg:- $n \notin \Omega(n^2)$

$n^2 \in \Omega(n)$

$n^3 \notin \Omega(n^4)$

$n^3 \in \Omega(n^2)$

$n^3 \in \Omega(n^2)$

$\frac{n(n-1)}{2} \in \Omega(n^2)$

$100n + 5 \notin \Omega(n^2)$

Q:- $\log(n) \in ?(n)$

$\log n$ grows less than n

$\therefore \log(n) \in O(n)$

also $n \in \Omega(\log n)$

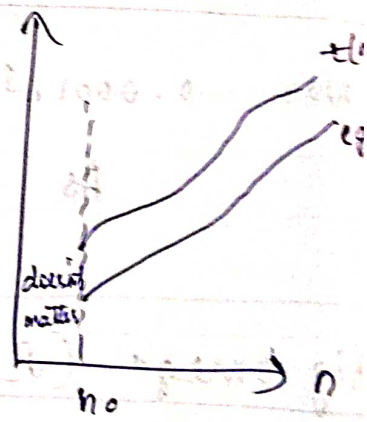
Q:- $2^n \in ?(n!)$

2^n grows less than $n!$

$2^n \in O(n!)$

$n! \in \Omega(2^n)$

$n \log n \in \Omega(\log n)$



Big thetha (Θ) notation

$\Theta(g(n))$ is the set of all functions within a constant multiples as n goes to infinity

Eg:- $n^2 + \log n \in \Theta(n^2)$

$\frac{n(n-1)}{2} \in \Theta(n^2)$

Asymptotic growth = ignoring constants and smaller values

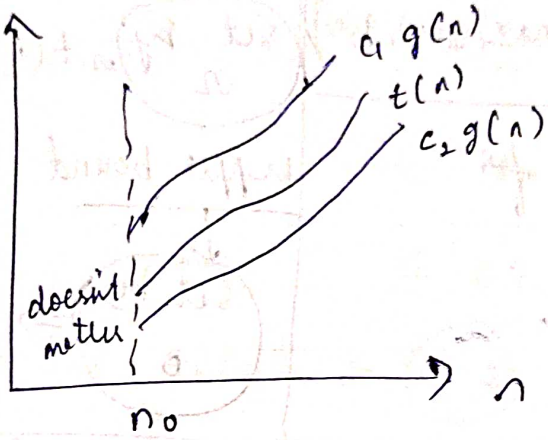
A function $t(n)$ is said to be in $O(g(n))$

denoted by $t(n) \in O(g(n))$

if $c_2 g(n) \leq t(n) \leq c_1 g(n)$

i.e.

$t(n) \in O(g(n))$ if $t(n)$ is bounded both above & below by some constant multiples of $g(n)$ for all large n



Eg:- $\frac{n(n-1)}{2} \in O(n^2)$

$c_1 = 1/2 \quad c_2 = 1/4$

$n_0 = 2$

Eg:- $\frac{1}{2}(n)(n-1) \in O(n^2)$

$c_2(n^2) \leq \frac{1}{2} n(n-1) \leq c_1(n^2)$

$\frac{n^2}{2} - \frac{n}{2} \geq \frac{n^2}{2} - \frac{n}{2} \cdot \frac{n}{2}$

$\geq \frac{1}{4}[2n^2 - n^2]$

$\geq \frac{1}{4}[n^2]$

Let $n_0 = 8$

$c_1 = \frac{1}{2}$

$c_1 = 1/2 \quad n_0 = 0$

$c_2 = 1/4 \quad n_0 = 2$

$n_0 = 8$

Consider upper bound $\frac{1}{2} n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \quad \forall n \geq 0$

Lower bound $\frac{1}{2} n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2} n \frac{1}{2} n$

$= \frac{1}{4} n^2$

$\therefore c_2 = \frac{1}{4} \quad c_1 = \frac{1}{2} \quad n_0 = 2$

Useful property involving asymptotic Notations

Property :-

Property useful in analyzing the algorithms consisting of two consecutively executed parts.

if $t_1(n) \in O(g_1(n))$

$t_2(n) \in O(g_2(n))$

inf

$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Assertions are true for Ω & O.

lower bound → minimum amount of comparision (work/operation) required by any algo to solve the given problem.

Set by Ω ⇒ $t(n) \in \Omega(g(n))$

upper bound

Set by O ⇒ $t(n) \in O(g(n))$

Eg:- Search → Binary search — 1st sort → $t_1(n)$ — then search → $t_2(n)$ → running time } 2 consecutively executing parts

Then overall time is analyzed by $\max(t_1(n), t_2(n))$

→ Its efficiency is $\log n$.

Bubble sort $= n^2$

Binary search $= \log_2 n$ } So max among 2 is n^2

$\therefore \max(t_1, t_2) = n^2$

Proof : Consider 4 real nos $a_1\ a_2\ b_1\ b_2$

Let $a_1 \le b_1$

$a_2 \le b_2$

then $a_1 + a_2 \le 2 \max\{b_1, b_2\}$

Now

$t_1(n) \in O(g_1(n)) \Rightarrow t_1(n) \leq c_1 \cdot g_1(n) \quad \forall n \geq n_1$

$t_2(n) \in O(g_2(n)) \Rightarrow t_2(n) \leq c_2 g_2(n) \quad \forall n \geq n_2$

Let us denote : $c_3 = \max\{c_1, c_2\}$

Consider $n \geq \max\{n_1, n_2\}$

Adding $t_1(n) + t_2(n)$

$t_1(n_1) + t_2(n_2) \leq c_1 g_1(n_1) + c_2 g_2(n_2)$

$t_1(n) + t_2(n) \leq c_3 g_1(n) + c_3 g_2(n)$

$\leq c_3 [g_1(n) + g_2(n))$

$\leq c_3 \cdot 2 \cdot \max\{g_1(n), g_2(n)\}$

$t_1(n) + t_2(n) \leq (c_3\, 2) \max\{g_1(n), g_2(n)\}$

$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\}$

with $c = 2 c_3$

$c_3 = \max\{c_1, c_2\}$

$n = \max\{n_1, n_2\}$

If an algo comprises of 2 consecutively executed parts the overall efficiency of algo is determined by the part with the larger order of growth

---

Using limits for comparing the order of growth.

$$\lim_{n \to \infty} t(n)/g(n) = \begin{cases} 0 & \text{order of growth of } t(n) < \text{order of growth of } g(n) \\ c > 0 & t(n) = g(n) \\ \infty & t(n) \geq g(n) \end{cases}$$

$\lim_{n \to \infty} \frac{t(n)}{g(n)}$ — $t(n)$ and $g(n)$ are non negative fn

So only +ve no <0 . - -

Eg: ① $t(n) = \log_2 n$

$g(n) = \sqrt{n}$

Now $\lim_{n \to \infty} \frac{\log(n)}{\sqrt{n}} = 0$ (log → very small; $\sqrt{n}$ → larger growth as compared to log)

So at ∞ v.v. small / large $= 0$

$\therefore \log_2(n) < \sqrt{n}$

$\Rightarrow \boxed{\log_2(n) \in O(\sqrt{n})}$

② $t(n) = \sqrt{n}$

$g(n) = \log n$

Now $\lim_{n \to \infty} \frac{\sqrt{n}}{\log n} = \infty$

$\therefore \boxed{\sqrt{n} \in \Omega \log n}$

③ $t(n) = \frac{1}{2} n(n-1)$

$g(n) = n^2$

$$\lim_{n \to \infty} \frac{\frac{1}{2} n(n-1)}{n^2}$$

$$\lim_{n \to \infty} \frac{1}{2} \frac{n}{n} - \frac{1}{2} \frac{n}{n}$$

$$\lim_{n \to \infty} \frac{1}{2}\left(1 - \frac{1}{n}\right) \to 0$$

$$= \frac{1}{2} > 0$$

$\therefore \boxed{\frac{1}{2} n(n-1) \in \Theta(n^2)}$ → equal

(4) $f(n) = n!$

$g(n) = 2^n$

$$\lim_{n\to\infty} \frac{n!}{2^n} = \infty$$

$$\boxed{n! \in \Omega(2^n)}$$

Indicate which of the following equalities are true/false

(1) $6n^2 - 8n \in \Theta(n^2)$

$f(n) = 6n^2 - 8n$

$g(n) = n^2$

$$\lim_{n\to\infty} \frac{6n^2 - 8n}{n^2}$$

$$6 - \frac{8}{n} \to 0$$

$= 6 > 0$

$\therefore$ True

By asymptotic notation growth of Both $g(n)$ & $f(n)$ are almost same

(2) $\frac{n(n+1)}{2} \in O(n^3)$

$$\lim_{n\to\infty} \frac{n(n+1)}{2n^3}$$

$$\frac{1}{2} \lim_{n\to\infty} \frac{n^2+n}{n^3}$$

$$\frac{1}{2}\left[\frac{1}{n} + \frac{1}{n^2}\right] \to 0$$

$= 0$

$\therefore$ True

(or)

growth of $n^3 > n^2$

so true

---

(3) $\frac{n(n+1)}{2} \in O(n^2) \to$ true

Exactly it shld be

$\frac{n(n+1)}{2} \in \Theta(n^2)$

by def^n But its less than/ equal

$\therefore$ True

But more exact is O.

(4) $\frac{n(n+1)}{2} \in \Omega(n)$

True

(5) $\frac{n(n+1)}{2} \in \Omega(n^3)$

False

n^2 (n^3)

For the following functions indicate the class $\Theta(g(n))$ the functions belong to.. (use the simplest $g(n)$)

① $(n^2+1)^{10}$ ~~$\in \Theta(?)$~~

Sol^n $(n^2+1)^{10} \in \Theta(?)$

$\Theta(g(n))$

$(n^2+1)^{10} \in \Theta(n^{20})$

$\therefore g(n) = n^{20}$

② $\sqrt{10n^2+7n+3}$

$\in \Theta(n)$

$g(n) = n$

③ $2^{n+1} + 3^{n-1} \in \Theta(?)$

$2^n \cdot 2 + 3^n \cdot \frac{1}{3}$

2 & 3 $\Rightarrow$ constant

$2^n + 3^n$

growth of 3^n is ↑ among 2^n & 3^n

$\therefore$ By theorem

$2^{n+1} + 3^{n-1} \in \Theta(3^n)$

$g(n) = 3^n$

Arrange the following functions in decreasing order of growth

$(n+1)!$, 2^{3n} , $2n^4 + 2n^3 + 4$, $n \log n$, $6n$, $8n^2$, 3^n

Soln :- 1) $(n+1)!$

2) 3^n

3) 2^{3n}

4) $2n^4 + 2n^3 + 4$

5) $8n^2$

6) $n \log n$.

7) $6n$

~~8) 6~~

Time efficiency of non recursive algorithms.

Steps in mathematical analysis of non recursive algo

1) Decide on parameter n indicating i/p size
2) Identify algo's basic operation
3) Determine worst, average, best case for i/p of size n
4) set up summation for n reflecting loop structure of algo
5) Simplify summation using standard formulas to find order of growth

Summation rules

1) $\sum_{i=l}^{u} ca_i = c \sum_{i=1}^{u} a_i$

upper limit

lower limit

(2) $\sum_{i=1}^{u} a_i \pm b_i = \sum_{i=1}^{u} a_i \pm \sum_{i=1}^{u} b_i$

(3) $\sum_{i=1}^{u} a_i = \sum_{i=1}^{m} a_i + \sum_{i=m+1}^{u} a_i$

Formula

(1) $\sum_{i=1}^{u} 1 = u - l + 1$

(2) $\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1 + 2 \cdots + n$

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

(3) $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3}$

(4) $\sum_{i=1}^{n} i^k = \frac{1}{k+1} n^{k+1}$

$\sum_{i=1}^{n} i^5 \approx \frac{n^6}{6}$

(5) $\sum_{i=1}^{k} a^i = a + a^2 + a^3 \cdots$

$= \frac{a^{n+1} - 1}{a - 1}$ for $a \neq 1$

(6) $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

Write an algo to find largest of n elements in an array and analyze its time efficiency

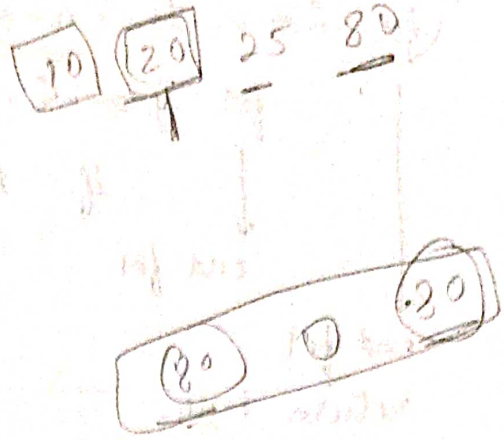
→ Algo maxElement(A[0 . . . . n-1])

// finds largest element in array →

// i/p :- array A [0 . . . n-1] →

// o/p :- largest element

```
max ← a[0]
for i ← 1 to n-1 do
    if a[i] > max
        max ← a[i]
end for
return max
```



Analysis

i/p size - No of elements 'N'

Basic operation - comparision

Basic operation count is not different for different inputs of the same size

$$c(n) = \sum_{i=1}^{n-1} 1 = u - l + 1$$

i/p size $= (n-1) - 1 + 1$

Basic op count $= \sum_{i=1}^{n-1} 1$

$= n - 1 - 1 + 1$

$= n - 1$

$= n - 1$

$c(n) = n - 1$

$c(n) \in \Theta(n)$

$c(n) \in O(n^2)$

$c(n) \in \Omega(\log n)$

$\Theta(n)$

$O\ n^2$

$\Omega\ \log n$

} order of growth

Efficiency class = linear

Write an algo to check if all elements of an array are distinct and analyze its efficiency.

1 2 3 4 ✓

1 2 1 4 ✗

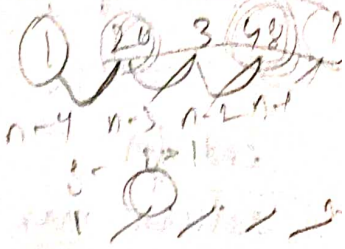
→ Algo elementUniqueness (A[0 .... n-1])

// Determine if all elements are distinct

// i/p: Array A[0 .... n-1]

// o/p :- Returns true if distinct, otherwise false

```
for i ← 0 to (n-2) do
    for j ← i+1 to n-1 do
        if A[i] = A[j] return false
    end for
end for
return true.
```



Analysis

→ I/P size : No of elements ('N')

→ Basic operation - comparision (equality)

→ Basic operation count is diff for diff i/p of same size

→ Best case → if 1st 2 elements are same

→ worst case → if all elements are distinct
or if last two elements are same

$\therefore C_{best}(n) = 1$ → const

$C_{best}(n) \in \Theta(1)$

efficiency class = constant

$$C_{worst}(n) = \sum_{i=0}^{n-2} \left(\sum_{j=i+1}^{n-1} 1 \right)$$

1st loop inner loop

$$= \sum_{i=0}^{n-2} \left[n-1 - (i+1) + 1 \right]$$

$$= \sum_{i=0}^{n-2} \left[n-1-i-1+1 \right]$$

$$= \sum_{i=0}^{n-2} \left[n-1-i \right]$$

$$= \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= (n-1) \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$= n-1 \left[n-2-0+1 \right] - \frac{(n-2)^2}{2}$$

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

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} \quad n-1$$

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

$$= \frac{(n-1)(n)}{2} \approx \frac{n^2}{2} - \frac{n}{2}$$

order & growth

efficiency class = quadratic

If $\frac{n(n-1)}{2}$ is compared with n^2

$$\lim_{n \to \infty} \frac{\frac{n(n-1)}{2}}{n^2} = \frac{1}{2} \lim_{n \to \infty} \frac{n^2-n}{n^2}$$

$$= \frac{1}{2} \lim_{n \to \infty} 1 - \frac{1}{n} \to 0$$

$$= \frac{1}{2} > 0 \to \Theta$$

$\therefore \frac{n(n-1)}{2} \in \Theta(n^2)$ class = Quadratic

---

Definition based algo to multiply 2 matrix and analyse its efficiency — efficiency is n^3

ALGORITHM Matrix Multiplication $(A[0..n-1], B[0..n-1])$

// Multiplies two square matrices of order n by def^n based algo

// i/p :- 2 $n \times n$ matrices A & B

// o/p :- matrix $C = A \times B$

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        c[i,j] ← 0.0
        for k ← 0 to n-1 do
            c[i,j] ← c[i,j] + a[i,k] * b[k,j]
return c
```

input size → order of matrix N

Basic o/p → Multiplication

Basic o/p count is same for diff inputs of same size

$$C(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [n-1-0+1]$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} \left[n \sum_{j=0}^{n-1} 1 \right] = \sum_{i=0}^{n-1} [n[n-1-0+1]]$$

$$= \sum_{i=0}^{n-1} n^2$$

$$= n^2 \sum_{i=0}^{n-1} 1 = n^2 [n-1-0+1] = n^3$$

$$C(n) \in \Theta(n^3)$$

Now $C(n) = M(n) = C_m n^3$ ✓ → ×n

$C(n) = A(n) = C_a n^3$ ✓ → +

$M(n) + A(n) = C_m n^3 + C_a n^3$

$= n^3 [C_m + C_a]$

C_m → time required to perform ×1

C_a → +1

Lab 1 → linear search → for 10, 100, 1k i/p

random generator

Euclid's

command → gnuplot

gedit plot.gnu

gedit linear_B.txt

gnuplot → plot.gnu

Design algo to find no of binary digits in binary representation of a positive decimal integer

ALGORITHM binary(n)

// i/p integer (positive) n

// o/p No of digits in binary form of n

count ← 1

while n > 1 do

count ← count + 1

n ← ⌊n/2⌋

return count.

input size → integer N

Basic o/p → Division

Basic o/p count depends on value of N

everytime it gets reduced by half

∴ Overall efficiency = $\log_2 n$

Algorithm Euclid (M, n)

// ... on beginning
while (n > 0)
    R ← m mod n
    M ← N
    N ← R
return m

log — worst
const — Best

Algorithm consecutive Intger (M, N)

I/P → Two non zero, non -ve integers.
O/P - GCD

L1: t ← min {m, n}
    if M % t == 0
        if N % t == 0 return t
    t ← t - 1 goto L1
    return t

worst case is if t is 1

(12) 24 16
t = 12 -
12 % 12 = 0
16 % 12 ≠ 0
11
12 % 11 ≠ 0

Best case
if on is multiple of n or vice versa
12 % 10 ≠ 0

Worst case
Relative primes
i.e, gcd = 1

9
8
7
6
5
4

Algorithm Modified Euclid (M, N)

:
while (N ≥ 0)
if (M < N) swap (M, N)
    M ← M - N
return M

12 % 2 = 0
16 % 2 = 0
GCD = 1

12 (24)
(24) 12

24 - 12 = 12
M N
12 12
M = 0 N = 12

N = 0
M

Eg gcd (6, 9) → Euclid → 3 division
consecutive int → 5 division
modified Euclid → 4 subtractions

(12) 0
M N

Find GCD of (31415, 14142), by applying i) Euclid's algo
Find Basic op count in each case
ii) Consecutive int checking
iii) Modified euclid algo

i) Euclid's algo

m = 31415 n = 14142

r = m % n
= 31415 % 14142
= 3131

n ← 3131
m ← 14142
⋮

GCD(31415, 14142)
GCD(14142, 3131)
GCD(3131, 1618)
GCD(1618, 1513)
GCD(1513, 105)
GCD(105, 93)
GCD(93, 19)
GCD(93, 19, 5)
GCD(4, 1)
GCD(1, 0)

10 divisions

ii) Consecutive int check

requires 14142 x 2 division if 1st condition is true
14142 → if both

Euclid's algo is faster by

$\frac{14142}{10}$ and $\frac{14142 \times 2}{10}$ i.e. 1414 & 2828

Smallest no of division made by Euclid's algo among all its inputs (m, n), $1 \le (m, n) \le 10$

Smallest no of divisions is 1
i.e. if m is multiple of n & $m \ge n$

Largest no of div is $m < n$ to get max div

(1,2), (1,3) ... (1,10) → not needed because n is a multiple of 1

(2,3), (2,4) (2,5), (2,6) ... (2,9), (2,10)
(3,4), (3,5), (3,7), (3,8), (3,10)
(4,5) ... (4,9) ... (5,6) (5,7) (5,8) (5,9) → 5 divisions

GCD (5, 8)
GCD (8, 5)
GCD (5, 3)
GCD (3, 2)
GCD (2, 1)
GCD (1, 0)

} 5 division

How do u say Euclid algo is log of [illegible]

10
20
50
100

→ No of divisions = $\log_{\phi}(3-\phi)N$

$\phi = \frac{1+\sqrt{5}}{2}$

Eg (M, N) ≤ (10) → N

No of division = $\log_{\phi}(3-\phi)$

$\phi = \frac{(1+\sqrt{5})}{2} = 1.62$

$= \log_{1.62}(3-1.62)10$

$= \log_{1.62}(13.8) = \frac{2.6242}{0.4824}$

= 5.44

= 5 divisions

1 → 10
5
20
6
30
6

Algorithm XYZ (N)

// I/P :- +ve integer N

// O/P :- ?

0 + 1 + 2 + ... N

S ← 0

for i ← 1 to N do

S ← S + i

end for

return S

i) What is the o/p of the algo
ii) What is the basic operation
iii) Establish the order of growth of basic op count
iv) Indicate the efficiency class
v) can you suggest a better algo for this?

Sol^n
i) Sum of first N natural nos
ii) Basic op = Addition
iii) $C(n) = \sum_{i=1}^{N} 1$

$= N - 1 + 1 = N$

$C(n) \in O(N)$ equal $N = N$
$C(n) \in \Omega(\log N)$ greater $N \not> \log N$
$C(n) \in O(N^2)$ lesser $N < N^2$

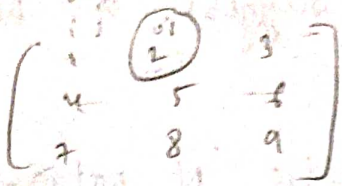
iv) linear
v) using formula $S \leftarrow \frac{N(N+1)}{2}$ // no loop is requi

Division by 2 is neglected
Bcoz of internal Binary division
$N+1$ → also neglected
Just 1 multiplication
efficiency class = constant

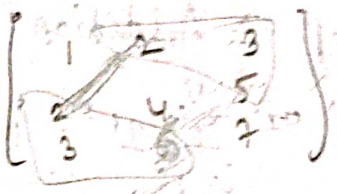
$\frac{n(n+1)}{2}$

Algorithm xyz (A[0.... n-1, 0.... n-1])
// i/p : n x n matrix
// o/p : ?

```
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        if A[i,j] ≠ A[j,i] return false
    end for
end for
return true
```



returns false



returns true

n = 3

i ← 0 to 2

j ← i+1 to 2

Now i = 0

j ← 1 to 2

A[01] = A[10] j = 1

A[02] = A[20] j = 2

Now i = 1

Now j ← 2 to 2

A[12] = A[21]

i) Check if matrix is symmetric or not

ii) Basic op = comparision

Worst case = Symmetric & only last comparision fail

Best case = 1st comparision fails, unsuccessful.

Algo works diff for diff i/p value & same size N.

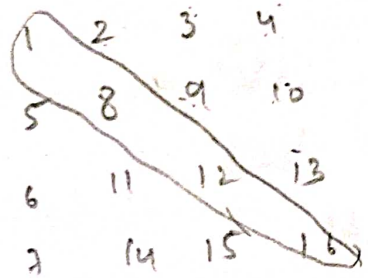
Best case :- unguarded
$C(n) = 1 \in \Theta(1)$ → constant
best

Worst case :- Successful
$C(n)$ = $\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$
worst

$C(n) \in \Theta(n^2)$ → refer distinct elements problem

Efficiency = Quadratic

Highest efficiency class → logarithmic (after count)
(least order of growth)



No of comparisions = $C(n)$

$= \frac{n^2 - n}{2}$

Any algorithm to check matrix symmetric required to compare all upper diagonal elem with lower diagonal elem

∴ Total no of comparisions needed = $\frac{n^2 - n}{2}$

$\frac{n^2}{2} - \frac{n}{2} \in O(n^2)$

We can't improve efficiency class to linear or log ← Efficiency class is quadratic
bcoz these many comparisions are necessary.

Algorithm XYZ $(A[0 \ldots n-1])$

// I/P : Array of size n

// O/P : ?

```
x ← A[0]
y ← A[0]
for i ← 1 to n-1 do
    if A[i] > x
        x ← A[i]
    if A[i] < y
        y = A[i]
end for
return x - y
```

Sol^n

Let n be 5 $\quad A = \{1, 5, -9, 8, 6\}$

$x = 1$

$y = 1$

for (1 to 4)

if $5 > 1$

$x = 5$

if $5 \not< 1$

⋮

o/p :- Largest - Smallest element of Array

$$c(n) = \sum_{i=1}^{n-1} 2$$

$$= 2 \sum_{i=1}^{n-1} 1$$

$$= 2[n-1]$$

$$= 2n \in O(n)$$

Note:-
Minimum no of Basic op for Euclid = 1
i.e. $m < n$
and m is multiplicative const of n.

For range $1 \le m, n \le N$

$\log_{\phi}(\sqrt{5}\, N)$ } Maximum no of division i.e. basic op.

$\phi = \frac{1+\sqrt{5}}{2}$

So first we check for

	Best	Worst
1 to 10	(1)	(5)
1 to 20	(1)	(6)
1 to 200	(1)	(11)

} growth is slow so its log

Note:- system ("pause") → to stop the screen

Basic op ~~count~~ in Euclid is division
$\therefore$ its growth follows this pattern
So growth is logarithmic

Write an algorithm to find number of digits required for binary representation of integer decimal no & perform its Analysis

10
↓
1010

Algorithm Binary(n)
// I/P :- A +ve integer n
// O/P :- No of digits required in binary representation

```
cnt ← 1
while (n > 1)
    cnt ← cnt + 1
    n ← n/2
return cnt
```

Tracing :-

cnt	n
1	10
10 > 1	
2	5
5 > 1	
3	2
2 > 1	
4	1
1 ≯ 1	
return 4	

Analysis :-

Frequently performed operation is comparision $n > 1$

$n/2$ happens iff $n > 1$ is satisfied

Here count of $n = +1$ of the no of times loop is executed

everytime n is reduced to half so $\log_2 n$

$\therefore$ no of comparision = $\log_2 n$

Basic operation :- comparision → Here basic operation is the part of comparision of loop not inside loop

Here i/p is based on no characteristics

$\therefore$ $b = \log_2 n + 1$

(magnitude)

$n = 2^b$

$\therefore$ o/p size :- 2^b

The basic op is not inside loop. It is a part of loop condition

But the choice does not matter as no of time condition gets executed is just one more than the no of times loop is executed

Since the loop variable is getting reduced exactly by half in each iteration, no of times loop gets executed is $\log_2 n$

$C(n) = \log_2 n + 1$ → equal to b
i.e. No of digits in binary rep :-)

Improve the algo of def^n based matrix x^n by reducing No of +^n. What effect this change will have on algo efficiency.

Eg:- $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$

Here c[i,j] ← 0
1 entry = 3 +^n
9 entries = 27 +^n

∴ No of +^n = n^3
where n = order of matrix

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        c[i,j] ← A[i,0] * B[0,j]
        for k ← 1 to n-1 do
            c[i,j] ← c[i,j] + A[i,k] * B[k,j]
```

So c[0,0] is initialized to 2 before only

Now c[0 0] = 2 + (2 × 1)
2 + (2 × 1)

∴ 1 entry = 2
9 entries = 18

18 = 3

∴ from 27 it is reduced to 18

∴ 27 − 9 = 18

$n^3 \rightarrow n^3 - n^2$

We are only optimizing the algo But same eff cls → But efficiency cls is same ∵ x^n can't be reduced

Algorithm xyz (A[0...n-1, 0...n])

// I/P :- Matrix of order n x (n+1)

```
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
        t = A[j,i] / A[i,i]
        for k ← i to n do
            A[j,k] ← A[j,k] - A[i,k] x t
        end for
    end for
end for
```

replace this by + to ↑ se efficiency

What inefficiency does this algorithm contains and how eliminate it to speed up the efficiency

Sol^n The inefficiency is due to $\frac{A[j,i]}{A[i,i]}$ which is independent of k.

Division is a costly operation and instead of repeating it by (n-i) times inside for loop of k we can do that only once by keeping it outside k loop using temp variable

Basic op = x^ or ÷^

so 1

order of growth

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=i}^{n} 1$$

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} n-i+1$$

$$= \sum_{i=0}^{n-2} \left(\sum_{j=i+1}^{n-1} n-i + \sum_{j=i+1}^{n-1} 1 \right)$$

$$= \sum_{i=0}^{n-2} \left\{ (n-i) \left[n-1-i-1+1 \right) + \left(n-1-i-1+1 \right) \right\}$$

$$= \sum_{i=0}^{n-2} \left\{ (n-i)(n-i-1) + (n-i+1) \right\}$$

$$= \sum_{i=0}^{n-2} \left\{ (n-i+1) \sum_{j=i+1}^{n-1} 1 \right\}$$

$$= \sum_{i=0}^{n-2} \left\{ (n-i+1) \left[n-1-i-1+1 \right] \right]$$

$$= \sum_{i=0}^{n-2} \left\{ (n-i+1)(n-i-1) \right]$$

$$= \sum_{i=0}^{n-2} (n-i)^2 - 1$$

$$= \sum_{i=0}^{n-2} \left[n^2 + i^2 - 2ni - 1 \right]$$

$$= \sum_{i=0}^{n-2} n^2 + \sum_{i=0}^{n-2} i^2 - \sum_{i=0}^{n-2} 2ni - \sum_{i=0}^{n-2} 1$$

$$= n^2 \left[n-2-0+1 \right] + \frac{(n-2)^3}{3} - 2n \frac{(n-2)^2}{2} - (n-2-0+1)$$

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

neglected

$$= n^3 + n^2 + n^3 - n^3 - n$$

$$= n^3 - n$$

$\therefore c(n) \in \Theta(n^3)$

Sample variance of n measurements $x_1, x_2 \ldots x_n$ computed

i) $$\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1} \qquad \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

(or)

ii) $$\frac{\sum_{i=1}^{n-1} x_i^2 - \left(\sum_{i=1}^{n-1} x_i\right)^2 / n}{n-1}$$

Calculate the no of divisions, multiplication, addition / subtraction which are used to compute variance using formula

Sol^n: (i) $D(n) = 2$ → one @ $\bar{x}$ (n)
→ other ($n-1$)

$M(n) = n$ → Eg: $x_1\ x_2\ x_3\ x_4$

$(x_1 - \bar{x})$ $(x_1 - \bar{x})$
$(x_2 - \bar{x})$ $(x_2 - \bar{x})$
$(x_3 - \bar{x})$ $(x_3 - \bar{x})$
$(x_4 - \bar{x})$ $(x_4 - \bar{x})$

$\therefore 4 = n$

If we initialize
sum = x_1
then no of iteration
for sum of all elem = $n-1$

Now
For $\bar{x}$ → $n-1$ + 1
For var → $n-1$ + 4
$(x_i - \bar{x})^2$ → n → if t is not taken
n → if $t = x_i - \bar{x}$ ✓
$n-1$ → 1

i.e. $(x_i - \bar{x})(x_i - \bar{x})$
then only 1 subtraction

$\therefore \quad A(n) \mid s(n) = (n-1) + (n-1) + n + 1$

$A(n) \mid s(n) = 3n - 1$

(ii) $M(n) = n + 1$

$A(n) \mid s(n) = n-1 + n-1 + 1 + 1$

$= 2n$

$D(n) = 2$

$\Sigma x_i^2 \rightarrow n$

$(\Sigma x)^2 \rightarrow 1$

$\therefore$ More efficient

$\therefore$ Division is independent of n.

$+^n$ & $\times^n$ is dependent of n

$\therefore$ among $3n-1$ & $2n$

$3n-1$ is larger growth

$\therefore$ (ii) is more efficient.

ANALYSIS of RECURSIVE ALGORITHMS

1st 3 steps are same as iterative algo analysis

4) Set up a recurrance relation (instead of summation) to express the basic op count with appropriate initial condition.

5) Solve the recurrance relation to find the order of growth of basic op count & indicate the efficiency class.

Write a recursive algo to find factorial of a no and perform its analysis

Algo Fact (n)
// i/p :- +ve integer n
// o/p :- n!
if n=0 return 1
return n * Fact(n-1)

→ i/p size n = 2^b b = $\log_2 n + 1$
→ Basic op = Multiplication
→ M(n) = M(n-1) + 1
M(0) = 0 // initial condition setting
Methods to solve
1) forward substitution → works for limited cases
↓ time consuming usually
↳ 1st value
M(0)
M(1)
:
M(k)
:
M(n)
} then predict pattern.

2) backward substitution
Express n as n-1
n-1 as n-2
;
;
so on
} and solve it using initial value

3) linear 2nd order recurrance with constant
Eg:- Length of Fibonacci

iterative
Fact ← 1
for i ← 2 to n do
Fact ← Fact * i
end
return fact

size of i/p = b(msg)
$C(n) = \sum_{i=2}^{n} 1$
$= n - 2 + 1$
$= n - 1$
$C(n) \in \Theta(n)$
→ eff class = linear
→ order of growth = n
→ Basic op = x
→ size of i/p = msg = b

Backward subs:

$$M(n) = M(n-1) + 1$$
$$= M(n-2) + 1 + 1$$
$$= M(n-2) + 2$$
$$= M(n-3) + 1 + 2$$
$$= M(n-3) + 3$$
$$\vdots$$

$$M(n) = M(n-i) + i \quad \text{---(1)}$$

4!
4 x 3!
3 x 2!
2 x 1!
1 x 0!

Now initial cond^n $\Rightarrow M(0) = 0$

Now in RHS of (1) we need to get $M(n-i)$ as $M(0)$ i.e base case

$\therefore i = n$

$$M(n) = M(0) + n$$
$$= 0 + n$$
$$M(n) \in \Theta(n)$$

efficiency class $\rightarrow$ linear

Note:- To solve recurrance rel^n & problems we use

1) Backward substi
2) Forward substi
3) Smoothness theorem
4) Second order linear recurrance with constant coefficients

$$ax(n) + bx(n-1) + cx(n-2) = f(n)$$

$a, b, c \rightarrow$ real constants $\quad a \neq 0$

Algorithm xyz (A, n)

// i/p :- Array A of size N

// o/p :- ?

```
if n=1 return A[0]
    t ← xyz (A, n-1)
    if t ≤ A[n-1] return t
    else
        return A[n-1]
```

Input size :- No of elements

o/p :- smallest element in array

Basic op :- comparision

Doesn't depend on type only size

$C(n) = C(n-1) + 1$ → $t \leq A[n-1]$

for $n > 1$ $C(1) = 0$

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

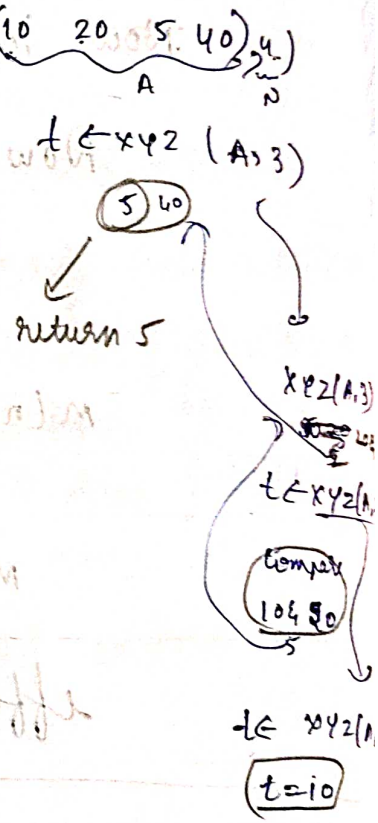
$C(n-i) + i$ $\quad n-i = 1$, $n-1$

$= 0 + n-1$

$C(n) = n-1$

$C(n) \in \Theta(n)$

Efficiency class = linear



Write a recursive algo to find number of digits in binary representation of a decimal and perform its analysis

Algo Binary (n)
if n → 1 return 1
return Binary (n/2) + 1

// i/p :- +ve integer n
// o/p :- no of digits in binary

Input size :- magnitude
o/p :- No of digits in binary
Basic op :- + (Not division bcoz if is a parameter)

n/2 is a param

Doesn't depend on type only on size

$A(n) = A(n/2) + 1$ for $n > 1$
$A(1) = 0$

$= (A(n/4) + 1) + 1$

$= (A(n/8) + 1) + 1 + 1$

$= \vdots$

$= A\left(\frac{n}{2^i}\right) + i$

For base case

$\frac{n}{2^i} = 1$

$n = 2^i$

$\log n = i \log 2$

$\frac{\log n}{\log 2} = \log_2 n$

$A\left(\frac{n}{2^{\log_2 n}}\right)$
$A(1) = 0$

$= A(\log_2 n) + \log_2 n$
$= 0 + \log_2 n$
$= \log_2 n$

OR

1) For these problems where ÷ is there as parameter instead of subtraction, solve the recurrance for $n = 2^k$

2) use "smoothness theorem" → The recurrance pattern obtained for 2^k as parameter holds good for all other values i.e. for any arbitrary value of n.

Solve for $A(2^k) = A(2^k/2) + 1$
$= A(2^{k-1}) + 1$
$= A(2^{k-2}) + 1 + 1$
$= A(2^{k-i}) + i$

Base case
$2^{k-i} = 1$
$k - i = 0$
$i = k$

$= 0 + k$

$= k$

WKT $n = 2^k$

$k = \log_2 n$

$\therefore A(n) = \log_2 n$

$A(n) \in \theta(\log_2 n)$

efficiency class = log

$x(n) = x(n/2) + n$ for $n > 1$ $\quad x(1) = 1$

Let $n = 2^k$

$x(2^k) = x\left(\frac{2^k}{2}\right) + 2^k$

$= x(2^{k-1}) + 2^k$

$= \left(x(2^{k-2}) + 2^{k-1}\right) + 2^k$

$= \left(x(2^{k-3}) + 2^{k-2}\right) + 2^{k-1} + 2^k$

$\vdots$

$= x(2^{k-i}) + 2^k\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots\right]$

$= x(2^{k-i}) + 2^{k-i+1} + 2^{k-i+2} + \cdots + 2^k$

$2^{k-i} = 1$

$k - i = 0$

$k = i$

$= x(1) + 2^1 + 2^2 + \cdots + 2^k$

$= 1 + 2^1 + 2^2 + \cdots 2^k$

$= 2^{k+1} - 1$

formula is $\frac{a(r^n - 1)}{r - 1}$

$x(n) = 2 \cdot (2^k) - 1$

$= 2 \cdot n - 1$

$x(n) = 2n - 1$

$x(n) = x(\sqrt{3}) + 1$ for $n > 1$ $x(1) = 1$

Let $n = 3^k$

$x(3^k) = x\left(\frac{3^k}{3}\right) + 1$

$= x(3^{k-1}) + 1$

$= \left(x(3^{k-2}) + 1\right) + 1$

$= x(3^{k-i}) + i$

$k - i = 0 \quad k = i$

$= 1 + k$

$= k + 1$

$n = 3^k$

$\log n = k \log 3$

$k = \log_3 n$

$\log n = \log 8^k$

$\log n = k \log 3$

$k = \frac{\log 3}{\log 3}$

$= \log_6 n$

$\boxed{x(n) = \log_3 n + 1}$

Algorithm xyz(A[l ... r])

// I/P :- Array with lower index l & highest index r

// O/P :- ?

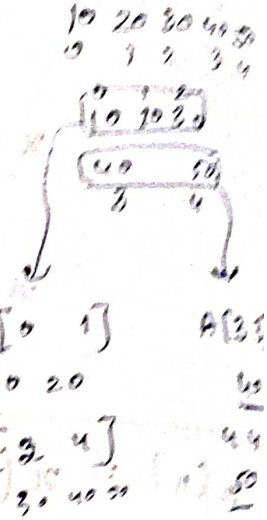
if (l = r) return A[l]

t1 ← xyz(A[l ..... (l+r)/2])

t2 ← xyz(A[(l+r)/2 + 1 ..... r])

if t1 < t2 return t1

else return t2



Finding minimum by using divide and conquer

i.e. min in first half } min among two.
min in 2nd half }

input size

O/P

Basic op

Doesn't depend on +

$$c(n) = c(n/2) + c(n/2) + 1 \qquad c(1) = 0, \; n > 1$$

$$= 2c(n/2) + 1$$

$$n = 2^k$$

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

$$= 2c(2^{k-1}) + 1$$

$$= 2(2c(2^{k-2}) + 1) + 1$$

$$= 2^2 c(2^{k-2}) + 2 + 1$$

$= 2^3 c(2^{k-3}) + 2^2 + 2 + 1$

$\vdots$

$= 2^i c(2^{k-i}) + 2^i - 1$

$k - i = 0$

$k = i$

$= 2^k (1) + 2^k - 1$

$= 2^k + 2^k - 1$

$= 2^{k+1} - 1$

$= 2n - 1$

Note:-

Recurrance rel^n :- Explicit formula to find nth value of fun^n

Recurrance for no of basic op :- To find order of growth & eff cls for recurrance rel^n.

Algorithm Q(n)

// i/p :- +ve int

if n=1 return 2

else

return Q(n-1) + 2 * n-1

Sol^n :- Q(n) = 2

Let n = 3 (7)

Q(2) + 2(2)

3 + 4

Q(1) + 2(1)

1 + 2 = 3

using Forward substitution

Set up recurrance relation

1, 3, 7

Q(1) Q(2) Q(3)

Q(4) = 7 + 2(3)

= 7 +

Q(2) = Q(1) + 2 {1}

= 1 + 2 * 2 - 1

1 + 4 - 1

= 4

Q(3) = Q(2) + 2 * n - 1

= Q(2) + 2 * 3 - 1

= 4 + 5

= 9

Solve same thing in backward substitution

Q(3) + 5 ... = 9

Q(2) + 3

Q(1)

1

$= n^2$

n^2

Set up the recurrance for no of multiplications and solve the order of growth

$$M(n) = M(n-1) + 1 \qquad n > 1$$
$$M(1) = 0$$

$$= [M(n-2)+1] + 1$$

$$= [M(n-3)+1] + 1 + 1$$

$$= M(n-i) + i \qquad n - i = 1$$
$$i = n-1$$

$$= 0 + n - 1$$

$M(n) = n-1$ order of growth $\in \Theta(n)$ iff class: linear

$A(n)$ ~~$\in \Theta(n)$~~ $= A(n-1) + 1 = n-1$

Write a recursive algo to find nth Fiboracci no

Algorithm Fib (n)

// I/P :- int n (+ve)

// O/P :- nth Fib no

if n = 0 return 0

if n = 1 return 1

else return Fib (n-2) + Fib (n-1)

I/P size :- Magnitude of n $n = 2^b$

Basic op :- +

The Basic op is independent of input type

Set up recurrance rel^n for this problem

$$F(n) = F(n-1) + F(n-2) \qquad F(0)=0,\ F(1)=1$$

Note:

Any recurrance of type

$$ax(n) + bx(n-1) + cx(n-2) = f(n)$$

if $f(n)$ is equated to zero for all values of n then it is homogeneous (if $f(n)=0$)

If not, non-homogeneous

2nd order linear recurrance with constant coefficients

degree of var is 1

becoz $n, n-1, n-2$ is the parameter of linear var x

a, b, c are real constants

Consider a homogeneous case

$$ax(n) + bx(n-1) + cx(n-2) = 0$$

Sol^n is obtained using 3 formula

which formula to be used is based on the roots of Quadratic eq^n which is of form $ar^2 + br + c = 0$ (characteristic eq^n)

Case 1:- If roots are real & distinct

i.e. $b^2 - 4ac > 0$

$$\boxed{x(n) = \alpha r_1^n + \beta r_2^n}$$

α & β → real constants

r_1, r_2 → root of QE

Case 2:- If roots are real and equal $b^2 - 4ac = 0$

$$\boxed{x(n) = \alpha r^n + \beta n r^n}$$

α & β → real constant

r → root

Case 3:- If roots are complex $b^2 - 4ac < 0$

with roots $(u \pm iv)$

$$\boxed{x(n) = \gamma^n [\alpha \cos\theta + \beta \sin\theta)}$$

α, β → real const

$\gamma = \sqrt{u^2 + v^2}$

$\theta = a\tan(v/u)$

Eg:- ~~$x(n) - 6x(n-1) + 9 = 0$~~

$x(n) - 6x(n-1) + 9x(n-2) = 0$; $x(0) = 0$, $x(1) = 3$

$a = 1$
$b = -6$
$c = 9$

characteristic eq^n : $r^2 - 6r + 9 = 0$

$r = 3$ real & equal

$x(n) = \alpha 3^n + \beta n 3^n$

$x(0) = \alpha 3^0 + \beta(0) 3^0$

$x(0) = \alpha$ $\quad (\because x(0) = 0)$

$\alpha = 0$

$x(1) = \alpha 3^0 + \beta 3$

$x(1) = 1$

$3 = \beta 3$

$\beta = \frac{3}{3}$ $\quad \beta = 1$

$x(n) = n 3^n$

Now solving for Fibonacci recurrance rel^n

↓

i.e. explicit formula to find n^{th} fib no.

$F(n) = F(n-1) + F(n-2)$ $\quad F(0) = 0$, $F(1) = 1$

$F(n) - F(n-1) - F(n-2) = 0$

$a = 1$
$b = -1$
$c = -1$

$r^2 - r - 1 = 0$

$r_1 = 1.61803$
$r_2 = -0.61803$

$b^2 - 4ac \quad 1 - 4(1)(-1) = 5$

$r_1 = \frac{-b + \sqrt{5}}{2a}, \qquad r_2 = \frac{-b - \sqrt{5}}{2a}$

$= \frac{-b \pm \sqrt{5}}{2a}$ Now $b = -1 \quad a = 1$

$\frac{1 + \sqrt{5}}{2}, \quad \frac{1 - \sqrt{5}}{2}$

$F(n) = \alpha \left[\frac{1+\sqrt{5}}{2}\right]^n + \beta \left[\frac{1-\sqrt{5}}{2}\right]^n$

Let $n = 0$

$0 = \alpha \left[\frac{1+\sqrt{5}}{2}\right]^0 + \beta \left[\frac{1-\sqrt{5}}{2}\right]^0$

$0 = \alpha + \beta$ $\qquad \alpha = -\beta$

Let $n = 1$

$1 = \alpha \left[\frac{1+\sqrt{5}}{2}\right] + \beta \left[\frac{1-\sqrt{5}}{2}\right]$

$= \alpha \left[\frac{1+\sqrt{5}}{2}\right] - \alpha \left[\frac{1-\sqrt{5}}{2}\right]$

$= \frac{\alpha + \alpha\sqrt{5}}{2} - \frac{\alpha - \alpha\sqrt{5}}{2}$

$1 = \frac{2\alpha\sqrt{5}}{2}$

$\alpha = \frac{1}{\sqrt{5}}$ $\qquad \beta = -\frac{1}{\sqrt{5}}$

$F(n) = \alpha r_1^n + \beta r_2^n$

$F(n) = \frac{1}{\sqrt{5}} \left[\frac{1+\sqrt{5}}{2}\right]^n - \frac{1}{\sqrt{5}} \left[\frac{1-\sqrt{5}}{2}\right]^n$

$$F(n) = \frac{1}{\sqrt{5}}\left[\phi^{n} - \hat{\phi}^{n}\right]$$

where $\phi = \frac{1+\sqrt{5}}{2}$

$\hat{\phi} = \frac{1-\sqrt{5}}{2}$

$\phi = 1.618$

$\hat{\phi} = -0.618$

As $n \to \infty$, $\hat{\phi}$ tends to 0

$\therefore \frac{1}{\sqrt{5}}[\phi^{n}]$

$\phi = 1.618 \longleftrightarrow$ called golden ratio

$\therefore F(n) = \frac{1}{\sqrt{5}}[\phi^{n}]$

Golden ratio

1) Fibonacci seq has role in plant study

Flower
Leaf pattern } follow fib sequence. 1 1 2 3 5
Seeds

2) Entire body structure is based on Golden ratio
So its called Divine Ratio.

3) $a/b = \frac{a+b}{a}$ when $a > b$ then a & b are in golden ratio.

4) In cryptography we use fib

No of +1 in fibonacci

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

In Iterative → growth is linear

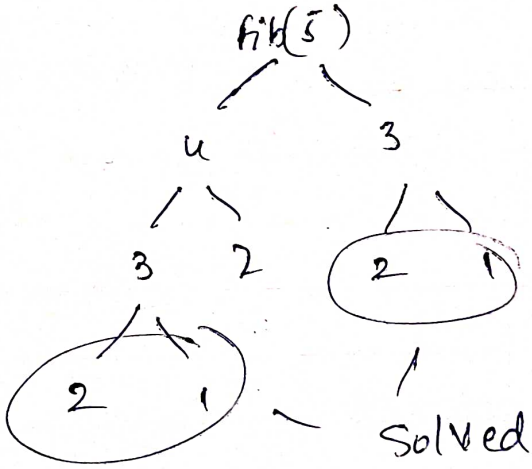
In recursion → it is exponential ($c\phi^n$)

So nature of recurrance

, recursion calls 2 more

tree of recursive calls is made

Here smaller subproblem gets computed again and again



Solved again & again.