22.04.2024

Module 1

- Algorithm – ~~step~~ sequence of unambiguous steps to solve a particular problem.

- Notion of an algorithm

Problem
↓
Algorithm
↓
Input → [Computer] → Output

Consideration of an algorithm :-

→ Never compromise ~~with~~ on the steps.
→ Range of input values.
→ Have to choose an algorithm that is most suitable for the problem.
→ Speed at which the algorithm works.

Eg:- Euclid's algorithm for computing gcd(m, n).

Step 1:- If $n=0$, return the value of m as the answer & stop, otherwise proceed to step-2.

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

Step 3:- Assign the ~~remainder~~ value of n to m and r to n. Go to step-1.

Algorithm

// Computes gcd (m, n) by Euclid's algorithm.
// Input : two non-negative, non-zero number.
// Output :- greatest divisor. of m & n.

while $n \neq 0$ do

$r \leftarrow m \mod n$

$m \leftarrow n$

$n \leftarrow r$

return m

23.04.2024

- the 2^{nd} integer of the pair gets smaller with each iteration and it cannot become negative.
- The new value of n on the next iteration is $\boxed{m \mod n}$ which is always $< n$.
- Hence, the value of the 2^{nd} integer eventually becomes zero and the algorithm stops.

$$\boxed{\gcd(m, n) = \gcd(n, m \mod n)}$$

Considerations-

(1) Ascertaining the capabilities of the Computational Device.
- ~~Capacity~~ capability/capacity of machine
- algorithm for serial processing or parallel processing
  - → some machines do not support parallel processing.
  - → same algorithms can follow both processing types.

(2). Understanding the problem.
- input, output → note down.
- note down special test cases.
- function of algorithm

(3). Choosing between Exact and approximate problem solving

- Not all algorithms give exact ~~so~~ answer for all inputs

Eg:- finding square roots → not all numbers have exact solution

approximate algorithm.

(4). Algorithm Design Techniques.

- clarity on whether the algorithm applies to all problems or only one problem.
- data structures involved.
  - ↳ cost-effectiveness.
  - ↳ easier handling.

(5). Method of specifying an Algorithm

- natural language / flowchart / high-level language.
  - ↳ combination of natural language + signs.
- algorithm → also called pseudocode or blueprint of a program.

(6). Proving an Algorithm's Correctness.

- testing → by giving various input.

Eg:- gcd of m & n.

→ check for a stop condition

→ check if proper output is obtained for a given input.

→ constraint to accept only +ve integer values

(7). Analyzing an Algorithm.

- efficiency of algorithm → interms of time & space.
- simplicity of the algorithm
- generic or specific.

• time taken to produce an output.

(8). Coding an Algorithm
• implementing the algorithm.
• 90% of algorithms are converted to code.

Space v/s time

• time over space
• algorithm must ~~be~~ taken less time → speed
• since space / capacity of the system can be increased.
• to calculate time.
→ consider a unit — speed / clock / external factors.
↓ as
unit of measurement

→ identity the basic step which is crucial for processing
→ calculate time taken to execute that step.
→ if external factor considered → applicable only to that step.
→ no. of inputs also matters.
→ size of input matters.
[giving huge input data requires more time to generate output].

Eg: (i) Consider a quadratic eqn. say x^2+2x+3.
The algo for this problem can be analyzed by varying diff. degrees of x and not by giving diff. values to x.
(ii). algorithm

Run-time measurement of an algorithm

$\cancel{T(n)} \approx Cop \frac{(C_n)}{}$ $\quad$ ~~$T(n) \approx C_{op}\, C(n)$~~

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

$T(n) \rightarrow$ run-time ~~of the~~

$Cop \rightarrow$ cost of most basic operation

$Cn \rightarrow$ no. of times the particular operation executes

Orders of Growth

- to know the efficiency of algo.

Order	Description
1	constant (the no. of times the step executes does not depend on the input size) → executes only once.
$\log_2 n$	Logarithmic (output changes with input) Eg:- $n = 8 \Rightarrow \log_2 8 = 3$ → no. of times the Basic step executes changes for diff. values of n.
$n \log_2 n$	linear logarithmic (no. of times the Basic step executes increases with the value of n)
n^2	Quadratic
n^3	Cubic
2^n	Exponential
$n!$	
n	Linear

Eg - Sequential search algorithm

- → best case
- → worst case
- → average case

} the position at which match is found.

Eg - Finding a topic in a book

- * best case → finding in the beginning of a book.
- * worst case → finding ~~in~~ the topic at the end of the Book.
- * average case → finding the topic in the middle of the Book.

Algorithm for Sequential | Linear search (A [0, n-1], K).

list of elements

// searches for a given value in a given array by sequential search

total no. of elements in array

// Input: An array A [0, n-1] and a search key K.

// Output: The index of the first element in A that matches K or -1 if there are no matching elements.

```
i ← 0  // assigning 0 to i
while i<n and A[i] ≠ K do
    i = i + 1
if i<n return i  // executed only when 'while' fails.
else return -1.
```

→ i>n

→ A[i] = K

Eg:- $A = \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \end{array}$
$A = [2, 1, 3, 0, 8, 6, 5, 9]$

Stopping conditions
→ when element is found. ⇒ $A[i] = K$
→ when element is not found ⇒ $A[i] \neq K$.

$n = 8$

$i = 0$

$K = 6$

$(i<n)$ $A[i] \neq K$.

I. $i=0 \Rightarrow 0<8$ AND $2 \neq 6 \Rightarrow$ True
& $i = i+1 = 0+1 = 1$.

II. $i=1 \Rightarrow 1<8$ AND $1 \neq 6 \Rightarrow$ True
$i = i+1 = 1+1 = 2$

III. $i=2 \Rightarrow 2<8$ AND $3 \neq 6 \Rightarrow$ True
$i = i+1 \Rightarrow 2+1 = 3$

IV. $i=3 \Rightarrow 3<8$ AND $0 \neq 6 \Rightarrow$ True
$i = i+1 \Rightarrow 3+1 = 4$

V. $i=4 \Rightarrow 4<8$ AND $8 \neq 6 \Rightarrow$ True
$i = i+1 \Rightarrow 4+1 = 5$.

VI. $i=5 \Rightarrow 5<8$ AND $6 \neq 6 \Rightarrow$ False.

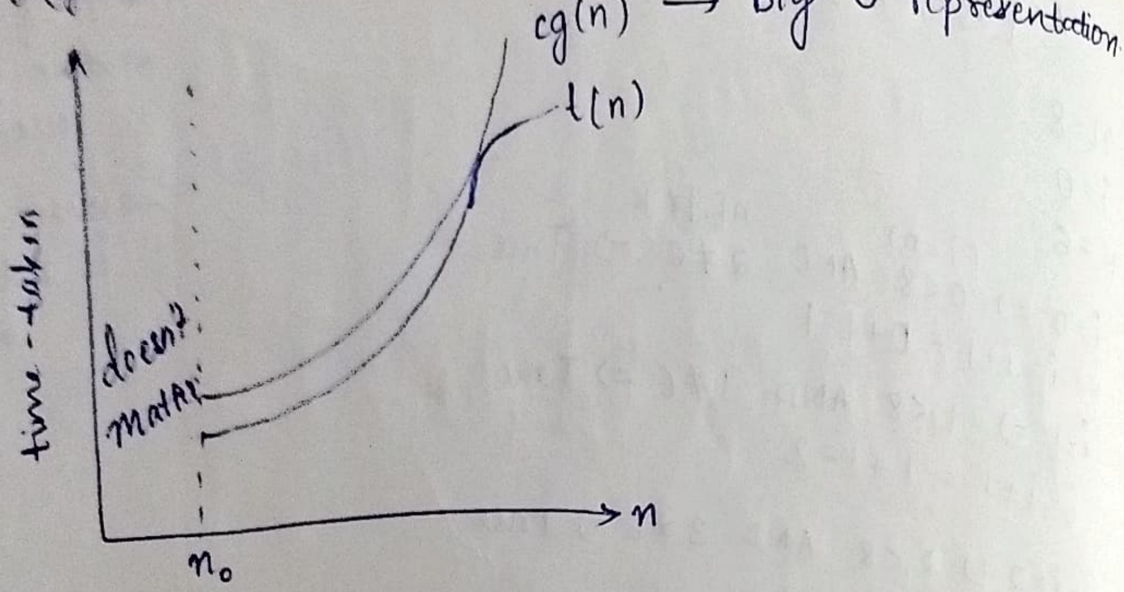
if $5<8 \Rightarrow$ True
return 5.

Asymptotic Notations.

Big O → 90%
Big Ω } → rarely 10%
Big Θ

→ Mathematical representations for analysis of efficiency algorithms.
→ only time efficiency.
→ based on the no. of executions.

(1.) Big O
→ gives max. usage of an algo.



$n \rightarrow$ no. of input values.

$t(n) \rightarrow$ function.

say, $t(n) = 2n^2 + n$ → constant → value such that LHS ≤ RHS

$t(n) = c \cdot g(n)$ → $g(n) = n^2$.

↓

$2n^2 + n$.

least upper ~~hold~~ bound → least value in upper threshold

$n \rightarrow$ lower bound

out of $n^2, n^3, n^4, \ldots$ → n^2 as upper bound.

$2n^2 + n \leq c \cdot n^2$ ~~$g(n^2)$~~.

~~Substitute~~ Substituting diff. values for n & c

Eg:- $n=1$, $c=1$

$3 \nleq 1$

$n=1$, $c=2$

$3 \nleq 2$

$n=2$, $c=3$

⋮

(i) Consider C as constant & vary only n.

$\Rightarrow LHS \leq RHS \Rightarrow$ always true

✓

(ii) $\boxed{2n^2+n = c\cdot n^2}$

constant

→ Pick a value for $c > 2$.

$\Rightarrow LHS \leq RHS \Rightarrow$ always true.

<u>Big Ω</u>

→ represents highest lower bound, & in a lower bound range.

$t(n) = \Omega(\ldots)$ $\Rightarrow$ & $t(n) = 2n^2+n$

$\Omega(n) = c\cdot g(n)$.

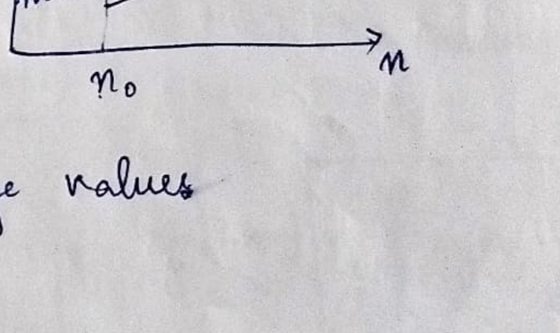
$g(n) = n$ → since n is lower bound.

$$t(n) \geq c\cdot g(n)$$

$$\downarrow$$

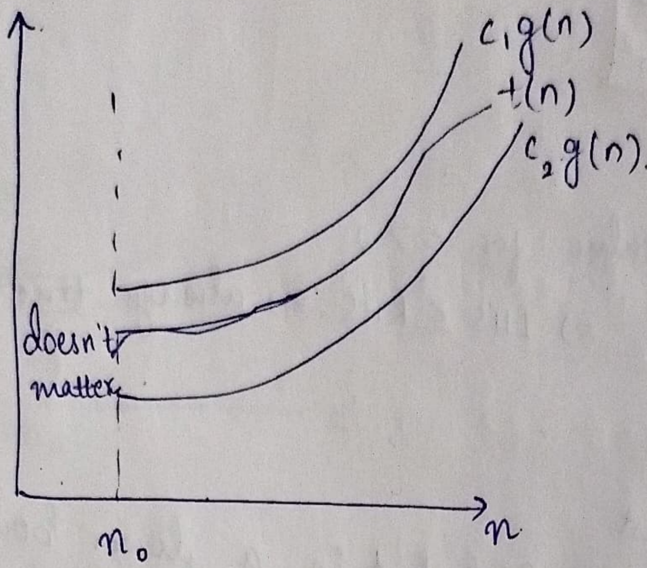
$$2n^2+n$$

$$\boxed{2n^2+n \geq c\cdot n}$$



<u>Big Θ</u>

→ represents average values



$$\boxed{\begin{array}{l} 2n^2 \le 2n^2+n \le 3n^2 \\ 2 \qquad\qquad\qquad 3 \end{array}}$$

Best case → Big O

Average case → Big Θ

Worst case → Big Ω

29.04.2024

General Plan for Analyzing the Time efficiency of Non-recursive Algorithms.

(1). Decide on a parameter (s) indicating an input's size.

(2). Identify the algorithm's basic operation. (As a rule, it is located in the inner most loop).

(3). check whether the no. of times the basic operation is executed depends only on the input size.
Also depends on some additional property, the worst, the average case, & if necessary, best-case efficiencies have to be investigated separately.

(4). Set up a sum expressing the no. of times the algorithm's basic operation is executed.

(5). Using standard formula & rules of sum manipulation, either find a closed-form approach for the count or, at the very least, establish its order of growth.

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

(ii). $\sum_{i=l}^{u}(a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$

(iii). $\sum_{i=l}^{u} i = \frac{n(n+1)}{2}$

(iv). $\sum_{i=l}^{u} i^2 = \frac{n(n+1)(2n+1)}{6}$

where
$l \rightarrow$ lower limit
$u \rightarrow$ upper limit
$n \rightarrow$ no. of times basic step executes.

Condition: $l \leq u$

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