Algorithms - 1

INTRODUCTION

Data Structures & Algorithms

Algorithms

(1) Introduction

(2) Asymptotic Notation

(3) Time complexity of loops

(4) Time & Space complexity of Recursive algorithms

(5) Methods to solve recurrence relations

(6) Divide & Conquer Algorithms

(7) Greedy Algorithms

(8) Dynamic Programming

(9) Graph Representation & Traversal

(10) Sorting Algorithms

Data Structures

1. Arrays
2. Linked Lists - Single LL, Double LL
   Circular LL
3. Stacks
4. Queues
5. Binary Trees
6. Binary Search Trees
7. Maps, Hash Tables, Sets
8. Graphs

Book :- Data Structures & Algorithms in Python
Goodrich, Tamassia, Goldwasser

P1

Algorithm vs Program

Algorithm	Program
Step by step procedure to solve a problem	Same
Abstract concept	Concrete implementation of algorithm
Language independent	must be written in a programming language C, C++, ... etc.
Non Implementable - you can't provide an algorithm to a computer & expect it to produce an O/P	Must be written in a programming language . Syntax, semantics has to be adhered

Syntax → how you write the code

print("Hello world!")

Semantics → Behaviour

O/P ⇒ Hello world!

Algorithm example

$n! = n \times (n-1) \times (n-2) \times \cdots 1$

① n = Take an I/P
- should be an Integer ≥ 0

$n \in N$

② if $n == 0$
    Return 1
else
    result = 1

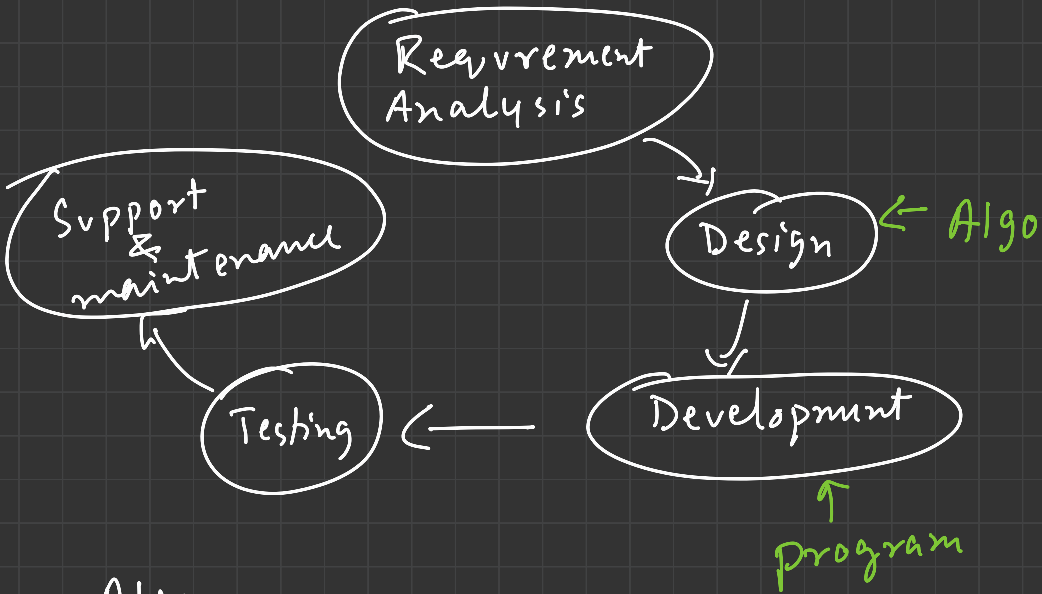
for i = 1 to n
{
    result = result * i
}

③ Return result

- Developed during Design phase

- Program developed during implementation phase

SDLC Cycle



Algo : ?

- no dependence on H/W
- Algo : Analyzed
  Efficiency
  - Time complexity
  - Space "

- Prog : Depends on H/W
- Prog : Tested

P2

Characteristics of Algorithms

(1) I/P — 0 or more i/p

e.g :- 0 input

- generate a random number & o/p the result

e.g :- 2 inputs

i/p => num1, num2

sum = num1 + num2

return sum

(2) O/P → algorithm must produce at least 1 output

- more than 1 o/p possible

e.g :-

(i) num = i/p

(ii) Output : num^2 (print)

(iii) Output : num^3 (print)

(3) Finiteness - An algo must terminate in finite time.

e.g :-

```
i = 1
result = 0
while (i > 0)
        do something.
```

infinite loop!
No Termination

ex:-

```
i = 1
while (i < 10)
        do something
        i++
```

(4) Definiteness

- must be clear / unambigious
- each & every step : must be precise

ex:- a, b $\in$ I/P

? $\longrightarrow$ perform some operation on a & b

o/p the result

(5) Effectiveness

should take
- less time
- less memory

] to execute

ex:-

Not Effective

Sum of N natural numbers

```
def func (n):
        sum = 0
        for i in range (1, n+1):
                sum += i          --- O(n)
        return sum
```

loop

Effective

```
def func (n):
        return n*(n+1)/2          --- O(1)
```

$O(n) > O(1)$

* We will define Time complexity shortly.

Summary

(1) I/P - 0 or more
(2) O/P - at least 1
(3) Finiteness - must terminate in finite time
(4) Definiteness - clear & unambiguous
(5) Effectiveness - less time, less memory & space

Ex - For you

1. Start
2. I/P = a, b
3. check a, b are numbers
4. Perform some operation of your choice
5. stop

Identify missing Characteristics

o

P3

Significance of Algorithms

* Bad Algo
  - → takes longer time
  - → consumes more memory

Bad Algo $\xrightarrow{\text{based}}$ Prog → it will take long time + may consume more space

ex:- calculate the sum of the first 10^{12} natural numbers.

```
def sum_nos(n):
    sum = 0                        --- 1
    for i in range(1, 10^12+1):    10^12 x2
        sum += i
    return sum
```

BAD ALGO

How many number of instructions?

$$1 + 10^{12} \times 2 + 1 \simeq 10^{12} \times 2 = 2000 \text{ Billion}$$

Let 100 million instructions = 1 sec

2K Billion " $= \dfrac{2K \times 1000}{100}$

$\simeq$ 20K secs

$\simeq$ 5.56 hours

How to improve the Algorithm?

```
def sum_nos:
    n = 10^12                  # 1 op
    return n x (n+1) / 2
```

$n = 10^{12}$ — 1 instructions

$n+1$ = 1 op. → add

$n \times (n+1)$ = 1 op → multiply

$\frac{n(n+1)}{2}$ = 1 op → division

Total = 4 instructions

GOOD ALGO

Time taken = $\frac{4}{100 \text{ million}}$ = 4 nano secs.

5.56 hrs → 4 nano seconds

That's the improvement

Note:-
This is why analysis of Algorithm is needed

P4 # Decidable vs Undecidable Problem

Topics

- Decidable problems
- Undecidable problem
- Polynomial & Exponential time

What is a decidable problem?

- The problem for which an efficient algorithm exists is a decidable problem.

Bad Algo	Good Algo
5.56 hrs	4 ns

context of decidable problem: This is efficient

Both of them are decidable

* An efficient algorithm takes $\leq$ Polynomial time to execute

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \quad n \in N$$

Let say

$$f(n) = 2n^2 + n + 1$$

n = size of the i/p.

Finite I/P $\longrightarrow$ $\boxed{\text{Algo (decidable)}}$ $\longrightarrow$ Terminates in polynomial time or less

Undecidable

- No Efficient Algo exists
- Takes exponential time $\rightarrow$ Can't decide outcome in a specific time frame.

$$f(x) = a^x$$

$a > 1$ $\qquad\qquad$ $0 < a < 1$


Domain = $(-\infty, \infty)$

Range = $(0, \infty)$

Size of i/P $\rightarrow$

n	2^n
1	2
2	4
3	8
4	16
⋮	
10	1024
15	32768

n	2^n
20	1048576
⋮	
100	incomprehensible no.

Finite i/p $\rightarrow$ $\boxed{\text{Algo (undecidable)}}$ $\rightarrow$ Terminates in exponential time but it is unbearable

Difference

Polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

(a) Linear Polynomial

$$f(x) = 3x + 7$$

(b) Quadratic Polynomial

$$f(x) = x^2 - 4x + 3$$

(c) Cubic Polynomial

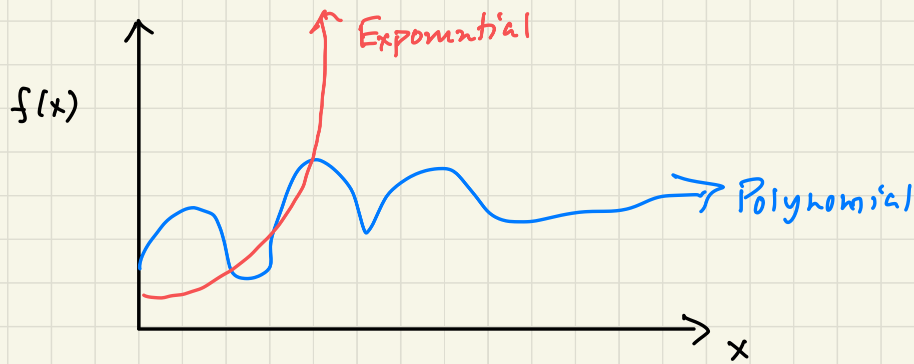
$$f(x) = 2x^3 - 5x^2 + 4x + 1$$

(d) Constant

$$f(x) = 5$$

Exponential

$$f(x) = a^x$$
$$a > 1$$



P5

The Nature of undecidable Problems

Finite I/P $n = 100$ $\longrightarrow$ $\boxed{2^n}$ Algorithm $\longrightarrow$ How much time it will take to execute?

no. of instructions $= 2^{100}$

Fastest computer

1 sec $\rightarrow$ 2^{20} instructions

In 1 yr. : $2^{20} \times 60 \times 60 \times 24 \times 365$

$2^{20} \times 2^{6} \times 2^{6} \times 2^{5} \times 2^{8} \simeq 2^{45}$ instructions per yr.

2^{45} instructions $=$ 1 yr.

$\Rightarrow 2^{100}$ " $= \dfrac{2^{100}}{2^{45}}$ yrs $= 2^{55}$ yrs

- Age of universe = 14 billion.
- 2^{55} yrs = 36 K billion yrs.

Conclusion

- For $n = 100$ $\rightarrow$ comprehensible size of i/p.
- Algo takes $= 2^{55}$ yrs

* Inefficient

* Undecidable problem.

P6

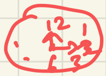
Posteriori & Priori Analysis

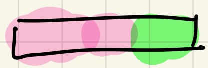
Topics

- Introduction to analysis of Algorithms
- Priori vs Posteriori Analysis

Intro - Analysis of Algorithms

- Study of performance of Algorithms

Based on → Time 

Based on → memory space consumption 

- Algorithm takes least - time, memory --> That algo is preferred

Two ways to analyze an algorithm

1. Priori analysis
2. Posteriori "

DIFFERENCE

Priori

- Estimate the
  - Time
  - memory

before executing it on system.

" rough estimation " computation

Posteriori

- Calculating the time & memory space required by an algorithm after executing it on the system

Priori

- Independent of the programming language
- No H/W dependence

↑
we will follow this

Posteriori

- Dependent on programming language

prog → Java → may run faster
prog → python

* Choice of programming language ⇒ important

- H/W dependent

Pentium Processor → slow

i7 → fast