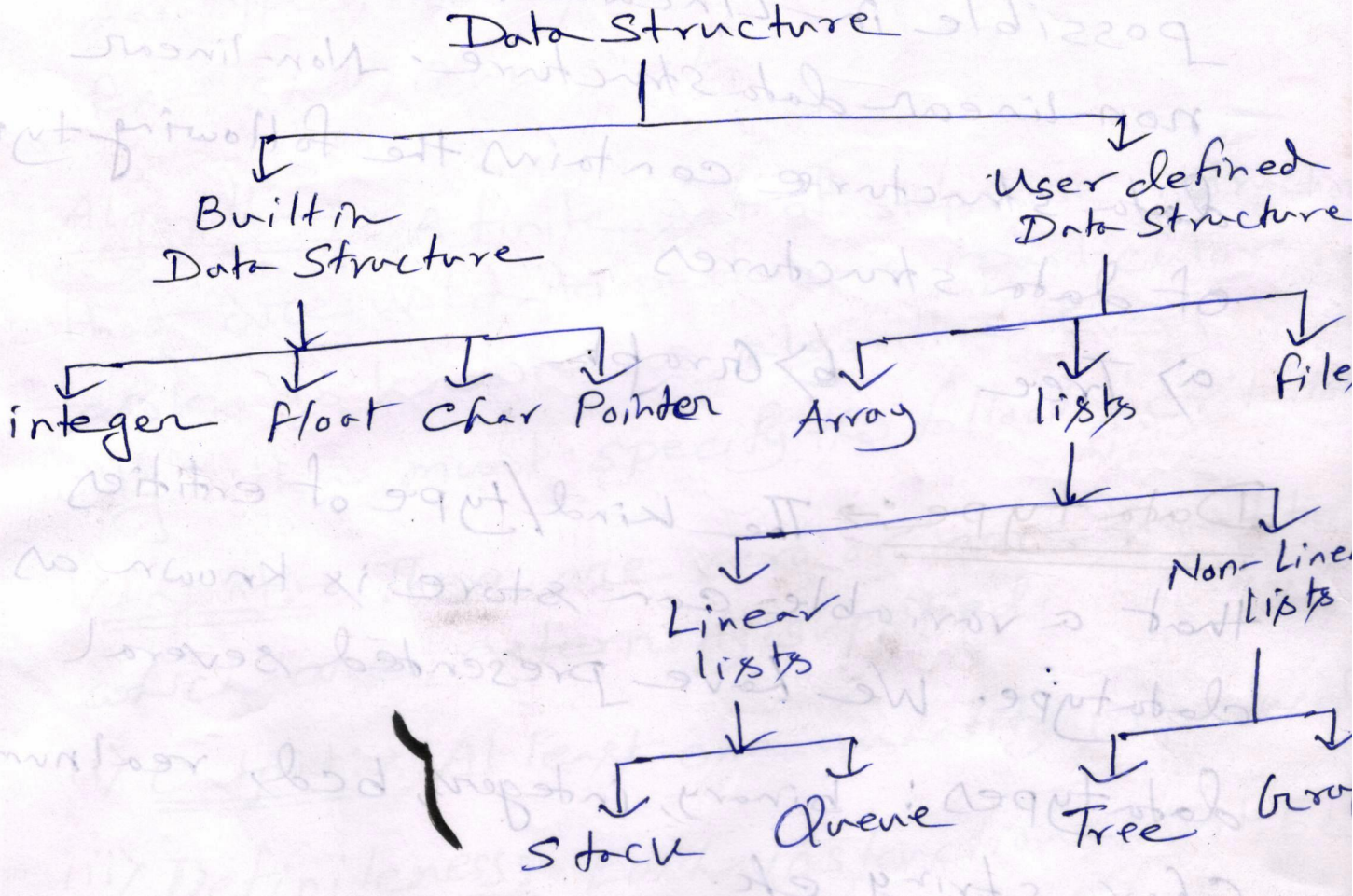
Data Structure

Def:- Data Structure is a particular way of organizing data in a computer so that it can be used efficiently.

or

The logical & mathematical model of a particular organization of data is called data Structure.



The data structure are classified in the following categories:

i) Linear Data Structure

ii) Non-Linear Data Structure

(2)

i) Linear Data Structure:→ In linear data structure data can be processed one by one sequentially. Linear data structure contains the following types of data structure

a) Array b) Linked List c) Stack & queue

ii) Non-Linear Data Structure:→ A data structure in which insertion & deletion is not possible in Linear fashion is called non-linear data structure. Non-linear data structure contains the following types of data structures

a) Tree b) Graph

Data type:→ The kind/type of entities that a variable can store is known as datatype. We have presented several datatypes: binary, integers, bcd, real number, char, string etc.

Abstract Data Type (ADT):→ The abstract datatype is special kind of datatype, whose behaviour is defined by a set of values & set of operation The keyword "Abstract" is used as we can use these datatypes, we can perform different operations. But how these operations are working that is totally hidden from the user. The ADT is made of with primitive datatypes, but operation logics are hidden. Ex- Stack, Queue etc.

Algorithm:→ A finite set of step-wise instruction that are used to perform a particular task is known as algorithm. Every algorithm must specify the following criteria

i) **Input:→** There are zero or more quantities which are externally supplied.

ii) **Output:→** At least one quantity is produced

iii) **Definiteness:→** Each instruction must be clear & unambiguous.

iv) **Finiteness:-** All algorithm will terminat after a finite number of steps.

v) **Effectiveness:→** Each algorithm must be effective.

(4)

Ex:- An algorithm for making a cup of Tea.

S1:→ Put the teabag in a cup.

S2:→ Fill the kettle with water.

S3:→ Boil the water in the kettle.

S4:→ Pour some of the boiled water into the cup.

S5:→ Add milk to the cup.

S6:→ A sugar to the cup.

S7:→ Stir the tea.

S8:→ Drink the tea.

Ex:→ The following are steps required to add 2 numbers entered by the user:

S1:→ Start

S2:→ Declare 3 variables a, b & sum

S3:→ Enter the values of a & b.

S4:→ Add the values of a & b and store the result in the sum variable, ie,

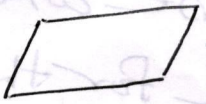
Sum = a+b

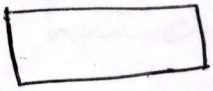
S5:→ Print sum.

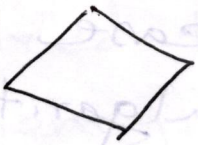
S6:→ Stop.

Flow Chart:→ The symbolic or graphical representation of stepwise instructions which are used to develop a program is known as a flow chart. The following symbols are used in drawing a flow chart.

i)  → Start/stop

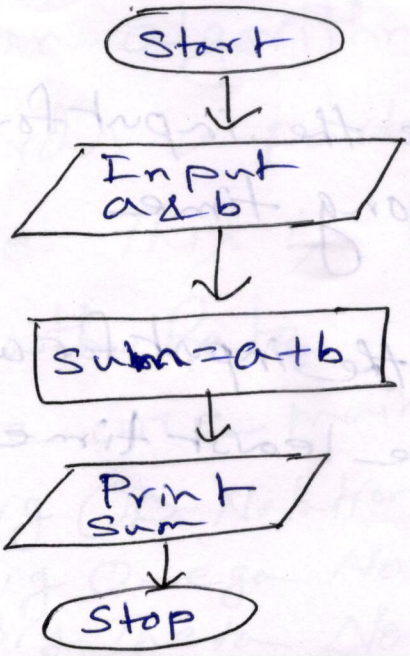
ii)  → Input/output

iii)  → Processing/Operations/Assignments

iv)  → Decission

v)  → Flow of instructions.

Ex- Sum = a+b



Type of Analysis :→ To analyze the given algorithm, we need to know with which inputs the algorithm takes less time & with which inputs the algorithm takes a long time. We know that an algorithm can be represented in the form of an expression. That means we represent the algorithm with multiple expression: One for the case where it takes less time & another for the case where it takes more time.

In general, the 1st case is called the best case & the 2nd case is called the worst case for the algorithm.

To analyze an algorithm, we need some kind of syntax, and that forms the base for asymptotic notation. There are 3 types of analysis:

Worst Case :→ Define the input for which the algorithm takes a long time

Best Case :→ Defines the input for which the algorithm takes the least time.

Average Case :- Run the algorithm many times, using many different inputs that come from some distribution that generates these inputs, compute the total running time (by adding the individual times), and divide by the number of ~~to~~ trials.

lower bound <= Average time <= upper bound

Asymptotic Notation :- The efficiency of an algorithm depends on the amount of time, storage & other resources required to execute the algorithm. The efficiency is measured with the help of asymptotic notations.

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value. This is known as an algorithm's Growth Rate.

There are mainly 3 asymptotic notation:

i) Big Oh Notation
ii) Big Omega Notation
iii) Big Theta Notation

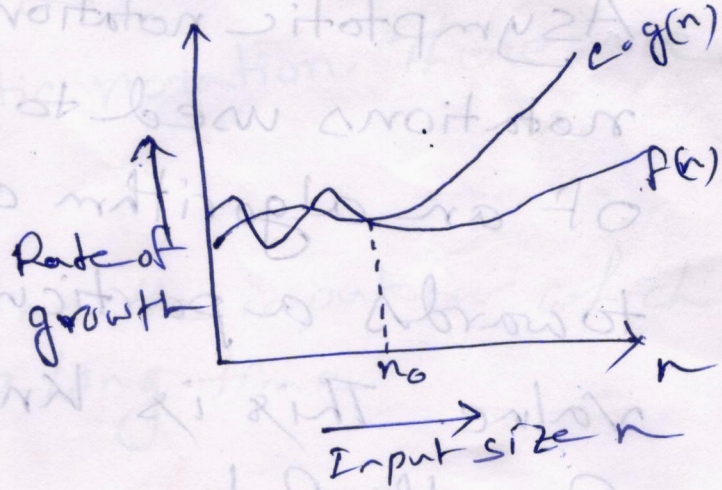
(8)

Big-O notation: ⇒ Big-O notation represe the tight upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

We can write $f(n) = O(g(n))$ iff there exists a +ve integer n_0 & +ve constant c, such that

$$f(n) \leq c \times g(n) \quad \forall \quad n \geq n_0$$

Here, n maybe number of inputs & $O(n)$ gives the maximum amount of time algorithm needs. At larger values of n, the upper bound of $f(n)$ is $g(n)$.



Ex- $f(n) = 16$

$f(n) \leq 16 * 1$

where $c = 16$ & $n_0 = 0$

So, $g(n) = 1$

$\therefore f(n) = O(1)$.

Ex→ $f(n) = 3n+5$

$f(n) = 3n+5$ where $n \geq 5$ $[n_0 = 5]$

$\leq 3n+n$

$\leq 4 \cdot n$

$\because g(n) = n$

$\therefore f(n) = O(n)$

Ex→ $f(n) = 27n^2+16n$

$f(n) = 27n^2+16n$ for $n^2 \geq 16n$

$\leq 27n^2+n^2$ $\Rightarrow n \geq 16$ where $n_0 = 16$.

$\leq 28n^2$

$\therefore f(n) = O(n^2)$

Ex→ $f(n) = 2^n + 6n^2 + 3n$

For $n^2 \geq 3n$ when $n \geq 3$

$\therefore f(n) = 2^n + 6n^2 + 3n$

$\leq 2^n + 6n^2 + n^2$

$\leq 2^n + 7n^2$

$\leq 2^n + 7 \cdot 2^n$

$\leq 8 \cdot 2^n$

$\therefore f(n) = O(2^n)$

For $2^n \geq n^2$ [For $n \geq 4$]

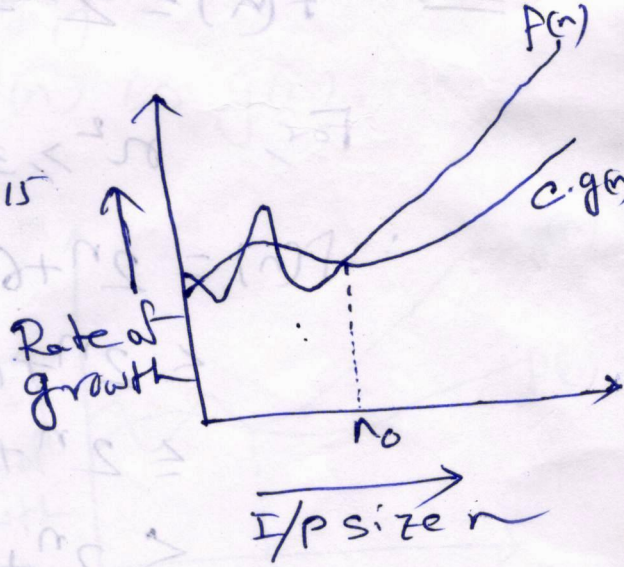
$n_0 = 4$ $\forall n \geq n_0$

(10) Big-Ω notation:→ Big-Ω notation represe
the tight lower bound of the running time
of an algorithm. Thus, it gives the best-ca
complexity of an algorithm.

We can write $f(n) = \Omega(g(n))$ iff

there exists a +ve integer n_0 & +ve constan
c, such that $f(n) \geq c * g(n)$ $\forall$ $n \geq n_0$

Here, n may be number of inputs & Ω(n)
gives the minimum amount of time
algorithm needs. At any values of n, the
lower bound of f(n) is g(n).



Ex:→ $f(n) = 16$

$\geq 15 * 1$ where $C = 15$
$n_0 = 0$

$\therefore f(n) = \Omega(1)$

$f(n) = 3n + 5$

$> 3n$ $\forall$ n & $C = 3$

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

$f(n) = 27n^2 + 16n$

$\geq 27n^2$ $\forall n$

$\therefore f(n) = \Omega(n^2)$

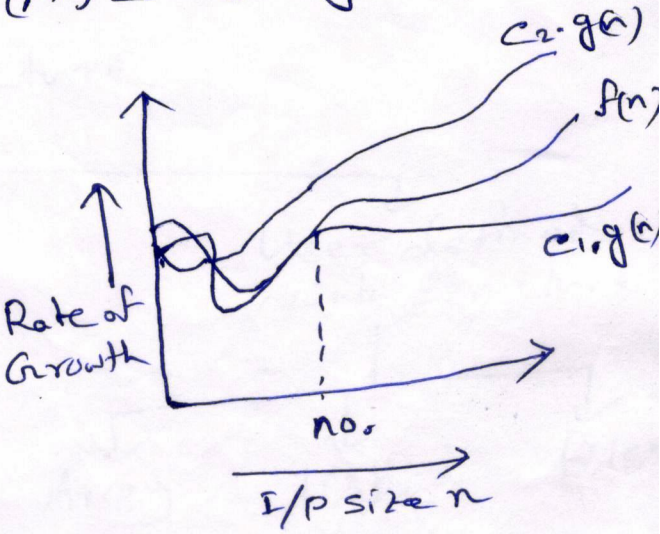
$f(n) = 2^n + 6n^2 + 3n$

$> 2^n$

$\therefore f(n) = \Omega(2^n)$

Big-O notation → Big-O notation bounds the function from above & below. Since it represents the upper & the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

We can write $f(n) = O(g(n))$ iff there exists a +ve integer n_0 & 2 +ve constants c_1 & c_2 s.t. $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ $\forall n \geq n_0$



Ex- Suppose $P(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_m n^m$ i.e. degree $P(n) = m$, prove that $P(n) = O(n^m)$

Let $b_0 = |a_0|$, $b_1 = |a_1|$, $\ldots$ $b_m = |a_m|$.

Then for $n \geq 1$

$$P(n) \leq b_0 + b_1 n + b_2 n^2 + \ldots + b_m n^m$$

$$= \left(\frac{b_0}{n^m} + \frac{b_1}{n^{m-1}} + \ldots + b_m\right) n^m$$

$$\leq (b_0 + b_1 + \ldots b_m) n^m = M n^m$$

where $M = |a_0| + |a_1| + \ldots + |a_m|$

Hence $P(n) = O(n^m)$