UNIT-01

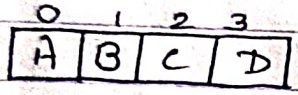
DATA STRUCTURE

01

The data may be assigned in several ways. The logical and mathematical model of a particular Data organisation is called Data Structure.

Data Structure Operations

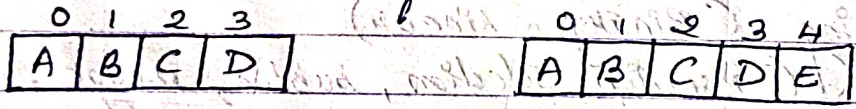
① Traversing: Accessing element record exactly once. This is called visiting (print) or we can count the number of node/element during traversing.



Output → prints A B C D.

② Searching: To find the location of the record which satisfy one or more condition.

③ Insertion: Adding a new record in existing structure. This increases the size of database.



④ Deletion: To remove existing record from Database. This decreases the size of the database.

If element from centre is removed or front, all terms shift forward.

⑤ **Sorting:** Process of arranging records in desired order (By default ascending). It may be alphabetical, increasing or decreasing order.

⑥ **Merging:** Combining the sorted record into single record.

TYPES OF DATA STRUCTURE

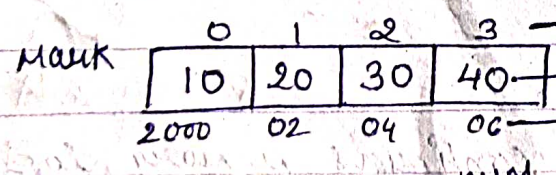
1. Array
2. Link list
3. Stack
4. Queue
5. Tree
6. Graph.

① **Array:** It is the simplest type of Data Structure. Collection of homogeneous data element stored in contiguous memory location. A linear array means list of finite no. of data elements.

It is denoted by [] in C language.

example:

1D `int Mark[4];`

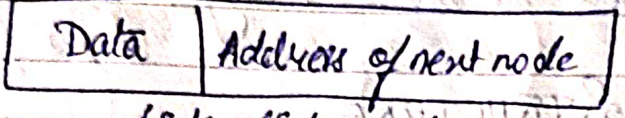


2D `int NUM[2][2];`

NUM	0	1
0	10	20
1	30	40

② **Link List**: The Data structure in which element is determined by the contain of next address is called link list.

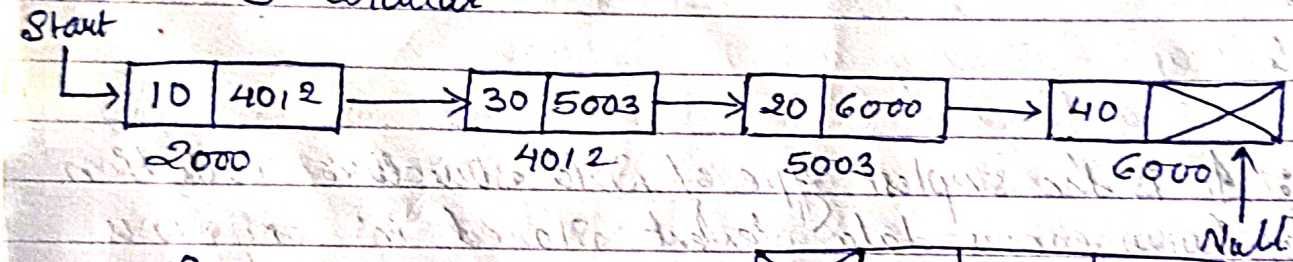
Every node of link list contains data and address of next node.



Link list node

There are 3 types of link lists:

1. Singly
2. Doubly
3. Circular



int ar[4];

	0	1	2	3
ar	10	30	20	40

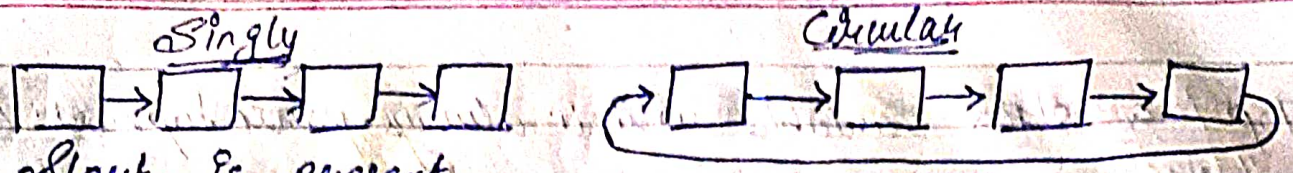
ar	50	60	80	70

Stored using linklist

ar	10	20	30	50	40	60

Can't be stored, coz no continuous memory location.

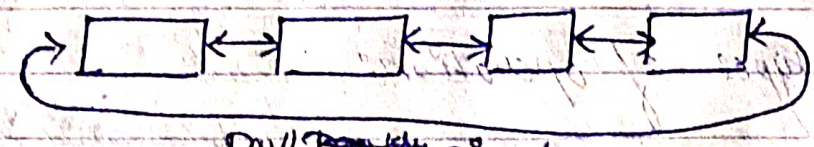
So, link list is used to store data.



Start is present
Null is present

No start, No null.

In doubly, both previous and next node address
In singly, only next node address.



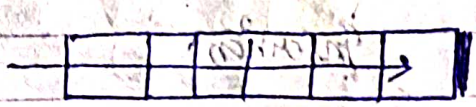
Doubly Circular

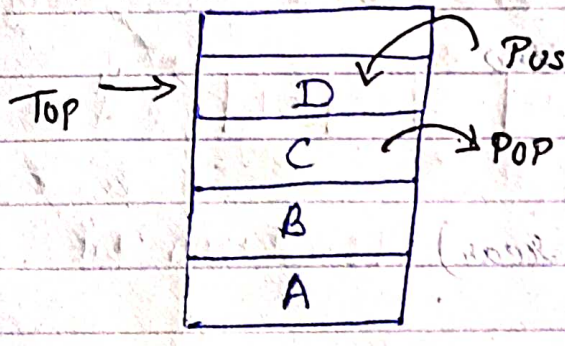
Doubly L.L.

Singly L.L.

(3) **Stack**: Stack is a type of array open from one side and close from another side.

It is also known as LIFO / FILO.





char st[4];

Top is a variable pointing the top most element.

Top is a stack variable which always points topmost element of stack.

element ↑ Top ↑ , element ↓ top ↓

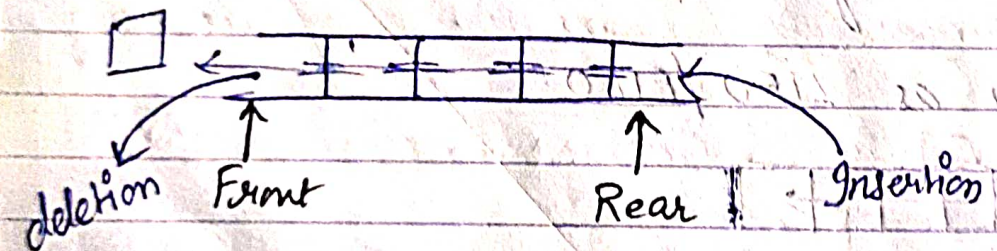
example : water drum.

There are 2 types of operation:

1. Push (Insertion)
2. Pop. (deletion)

(4) Queue: It is also called FIFO.

It is a linear list, in which insertion takes place from one end, i.e. front and deletion takes place from other end.



example: queue of ticket window

stack → 1 pointer (Top)

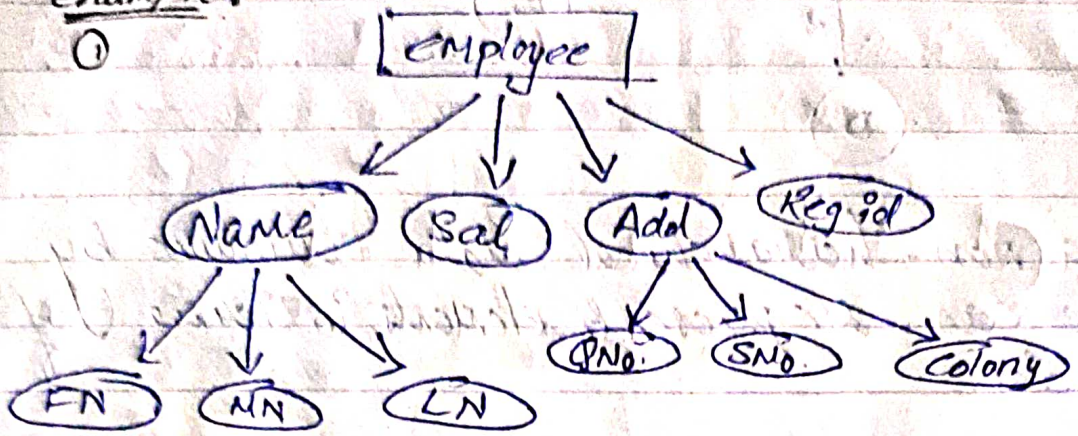
Queue → 2 pointers (front, rear)

12/01/16
06

⑤ **Tree:** The tree contain hierarchical structure between element and the data structure that reflects this relationship is called tree.

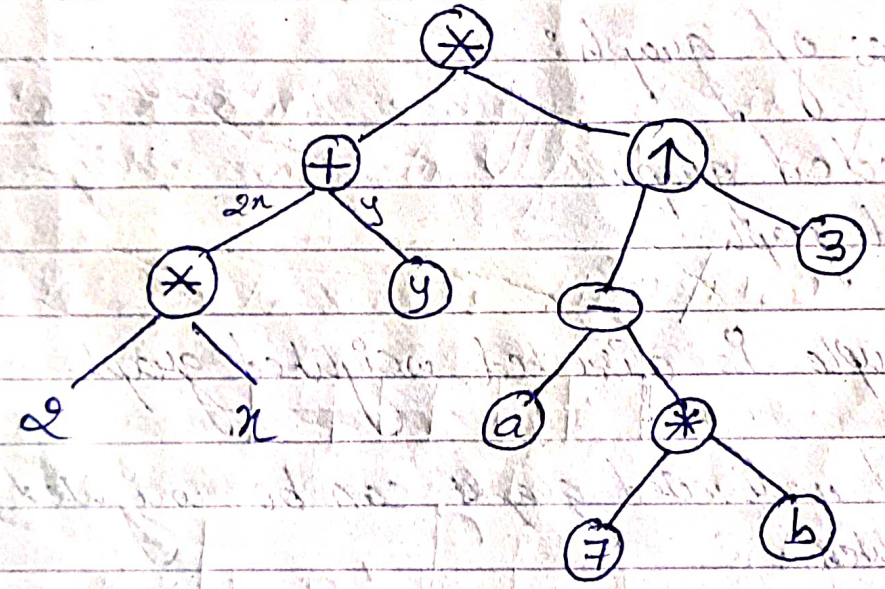
example:

①



② $(2x+y) * (a-7b)^3$ ↑ ⇒ for power.

↳ Root node.

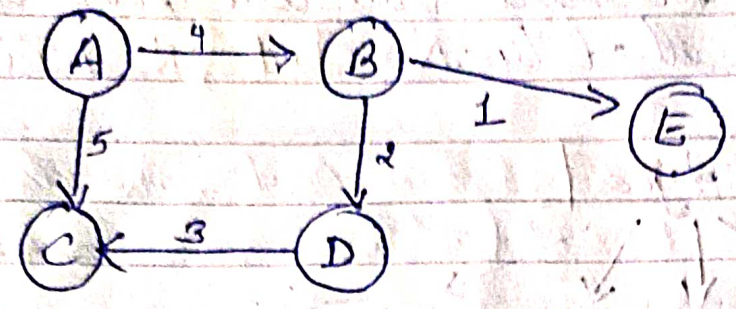


* Tree is hierarchical and single root node.
Graph is Non-hierarchical and no root node required.

21/10/21

01

⑥ Graph:



A graph is a non-heirarchical data structure by which we can represent elements in terms of connected nodes.

For example, the graph of air line connecting 5 cities A, B, C, D, E in above example figure.

There are 2 types of graph:

1. Undirected graph
2. Directed graph

Here, above example is directed weighted graph.

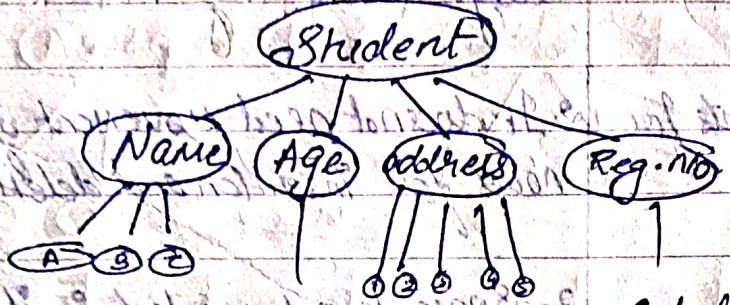
Both directed and undirected graphs can be weighted as well as non-weighted.

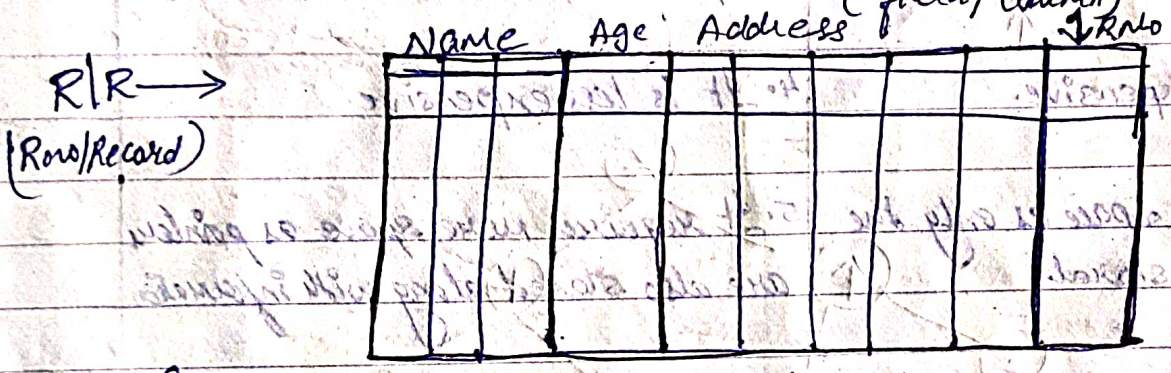
★ To print link list in reverse order, stack is used, i.e. if 1, 2, 3, 4 is stored in link list, if stack is used, the output will be 4, 3, 2, 1. as in stack, LIFO is followed. L.L. is stored in stack & then stack is printed

A → B → C

ELEMENTARY DATA ELEMENT {From EDRP}

File is collection of Record.
Record is collection of Fields.
Field contains data, for ex. name, age, etc





Structure of the data. It requires 10 different columns to store the data.
Here, Name, age, address, ~~Reg. No~~ are Non elementary or primary data. They can be similar for more than one data.
But, one column or field in which Not duplicate value could be present, is the elementary field. It must be unique and not null. Here, Reg. no will be primary, as its value is unique.

ASSIGNMENT

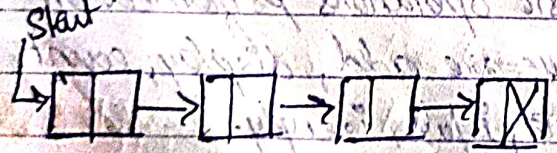
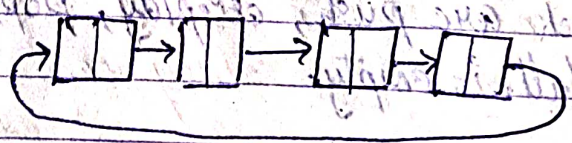
1) Array and Link list.

ARRAY	LINK LIST
1. Insertions and deletions are difficult (shifting Req.)	1. Insertions and deletions are easy. (shifting not Req.)
2. It needs movements of elements for insertion and deletion.	2. It doesnot need movement of the nodes for insertion and deletion.
3. Space is wasted in it.	3. Space is not wasted in it.
4. It is more expensive.	4. It is less expensive.
5. It requires less space as only the information is stored.	5. It requires more space as pointers are also stored along with information.
6. Its size is fixed.	6. Its size is not fixed.
7. It cannot be extended or reduced according to requirement.	7. It can be extended or reduced according to requirement.
8. Elements are stored in consequtive memory.	8. Elements may or may or may not be stored in consecutive memory.

20) Stack and Queue

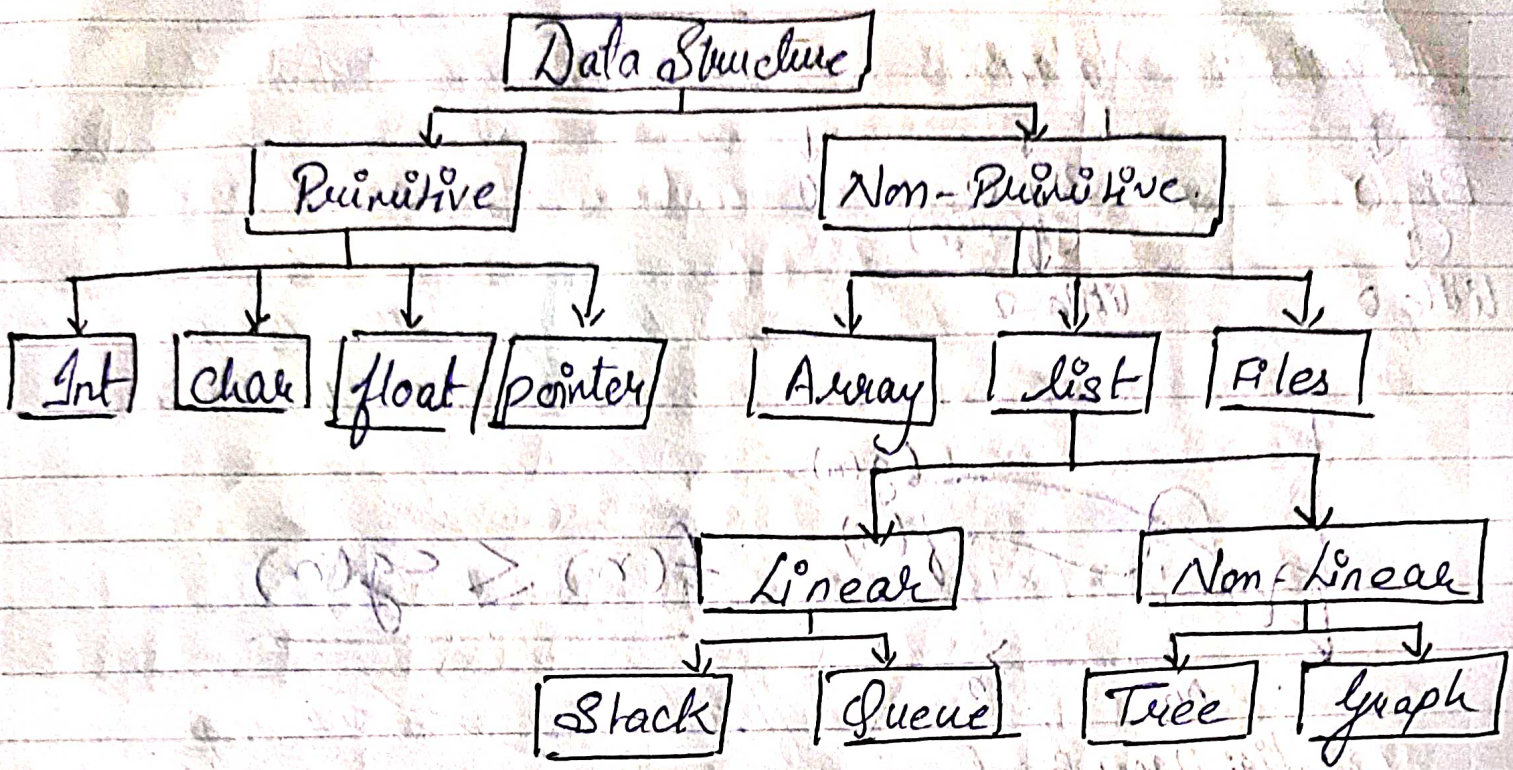
STACK	QUEUE
1. In stack, insertion and deletion operations are performed at same end.	1. In queue, insertion and deletion operations are performed at different ends.
2. In stack, an element which is inserted last is deleted first.	2. In queue, an element which is inserted first is deleted first.
3. It is called First in Last Out or Last in First Out [LIFO/FILO].	3. It is called First In First Out. [FIFO].
4. In stack, only one pointer is used called TOP.	4. In queue, two pointers are used, called FRONT and REAR.
5. In stack, there is no wastage of memory space.	5. In queue, there is wastage of memory space.
6. Basic operations performed on stack are push, display, pop, isfull, is empty.	6. Basic operations performed on queue are add, display, count, delete, isfull, isempty.
7. There is no concept of circular stack and priority stack.	7. The concept of circular ~~stack~~ queue and priority ~~stack~~ queue exists in queue.

3) Singly and circular Link list

SINGLY LINK LIST	CIRCULAR LINK LIST
1. Singly link list has the node inserted only at the end and corresponds to next pointer.	1. In circular link list, next pointer of last node points to first node
2. It has two nodes:	2. We can traverse only in left to right direction. It can have 2 or 3 nodes.
3. It is one way directional.	3. It is one directional but in circular motion and can be bidirectional
4. Start node is present, is first node.	4. Start node ~~is~~ can be any node. ~~absent.~~
5. Null node is present is last node.	5. Null node is ~~absent~~ absent.
6. Requires small space for each element.	6. Execute to the end can be quickly.
7. 	7. 

13/01/15

CLASSIFICATION OF DATA STRUCTURE



Abstract Data type (ADT): It is a type for objects whose behaviour is defined by a set of value & a set of operations. It define what operation to be performed & not How it will be implemented. Abstract mean implementation independent view.

The process of providing only necessary data & hiding the details is known as abstraction. The ADT is made of with primitive datatypes but operation logics are hidden.

Ex:- Stack, queue, Linked list

These ADT use some function to know their states by if isfull() , isempty() , size() etc

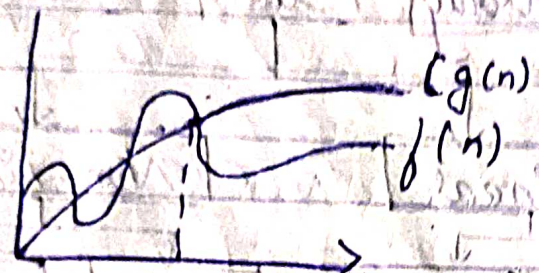
ALGORITHM:

Asymtotic Notations:-

Big O
↓
Little O

Big Ω (Omega)
↓
Little Ω

Θ



$f(n) \leq cg(n)$

fig. Upper Bond

Algorithm: It is set of rules for carrying out calculation either by hand or by machine.
It is sequence of computational steps that transforms the input into output.

Performance of algorithm is obtained by totalling the no. of occurences of each operation when running the algorithm. Performance evaluated as a function of i/p size n

Time and Space Complexity-

Space- Space complexity of an algorithm is the amount of memory needs to run the program completion.

Space needed by an algorithm is sum of following two components

1) Fixed part - That is independant of feature of input and output which include instruction Space, Space for Constant.

2) Variable Part - That consist of space needed by component variable whose size is depends on the particular problem instance being solved.

The Space required by the problem X can be computed as

$$S(x) = c + S_x$$

(Where) S_x is the variable part
c is Constant
$S(x)$ is space required

The reasons for Space Complexity are

1. If the program run into multi-user System and may be required to specify the amount of memory to be allowed to the program.

2. If interested to know in advance whether sufficient memory is available to run the program.

Time Complexity-

It is the amount of completion time it needs to run to completion. This is the Sum of Compile time and run time.

The Compile time doesn't depends on instance feature and the program Compiled once can be executed many times without recompiling it.

For runtime calculation we have to Calculate time taken in arithmatic operation, load, store, etc. we count the no. of operation.

We can count the no. of program steps and no. of steps of any program depends on the kind of statement.

Reasons for studying time complexity are-

1. We are interested to know in advance that whether an algo or program will provide a satisfactory real time response

2. There may be several possible solution with different time requirement.

Time space Tradeoff:- It is a situation where one factor increase & other will decrese. It is a way to solve problem in.

1) Either in less time & by using more space or
2) In very little space by spending a long time.

The best algo is that which helps to solve a problem that requires less space in less time to produce output. but in general it is difficult to achive. Therefore more time efficient algo you have, that would be less space efficient. Ex. ① stored image or Re-rendering ② lookup table & Recalculation ③ Compressed & Uncompressed data

Asymptotic Notation - (Approaching a value or curve)

To choose between best algorithm, we need to check efficiency of each algorithm which is measured by time complexity. they are imp. because of :-

1. Asymptotic notation gives the simple feature of algorithm efficiency.
2. They allow comparison of performance among algorithm.

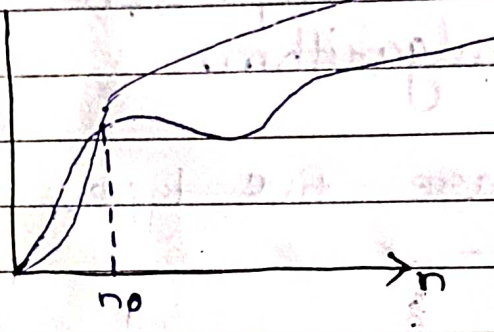
There are five types of asymptotic notation -

1. Big Oh (O)
2. Big Omega (Ω)
3. Theta (Θ)
4. Little oh (o)
5. Little omega (ω)

1. Big Oh (O) - (Tight upper bond) (For worst Case)

It is used to calculate upper bond of the function. Let $f(n)$ and $g(n)$ are two non-negative function then the function $f(n)$ is big oh of $g(n)$ $f(n) = O(g(n))$ should be written as

$$f(n) \leq c \cdot g(n) \quad n_0 < n, \; c > 0$$



Here $g(n)$ is upper bond of $f(n)$ so $g(n)$ grow faster than $f(n)$.

It is actual order of growth.

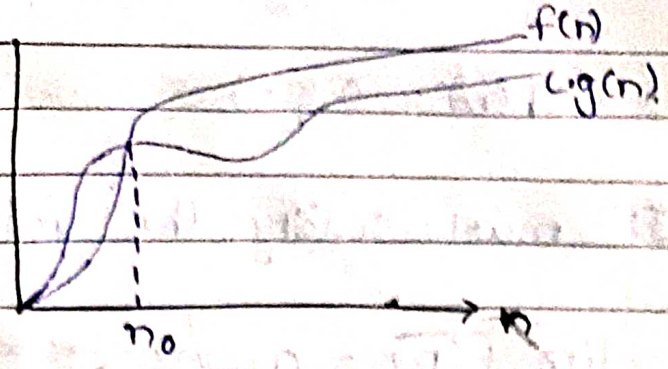
2. Big Omega (Ω) - (for Best Case)

This notation is used to calculate lower bond of a given function.

The function $f(n)$ is big omega of $g(n)$, should be written as

$$f(n) \in \Omega(g(n))$$

It is actual order of lower growth.

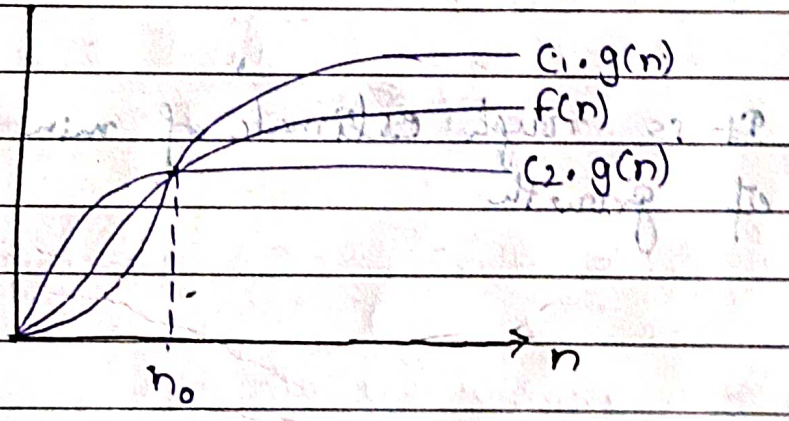


3. Theta (Θ)-

It is used to calculate middle bound of a given function. The f(n) is Θ of g(n), should be written as

$$f(n) \in \Theta(g(n))$$

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \qquad c_1 > 0,\ c_2 > 0$$



4. Little oh (o)- (loose upper bond)

The function f(n) is little oh of g(n), should be written as

$$f(n) = o\,g(n)$$

It must satisfy the following eqn.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

It is rough estimate of the max order of growth.

5. Little omega (ω)-

The function $f(n)$ is little omega of $g(n)$, should be written as

$$f(n) = \omega\, g(n)$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

It is rough estimate of min order of growth.