UNIT- I

Fundamentals of Data structures

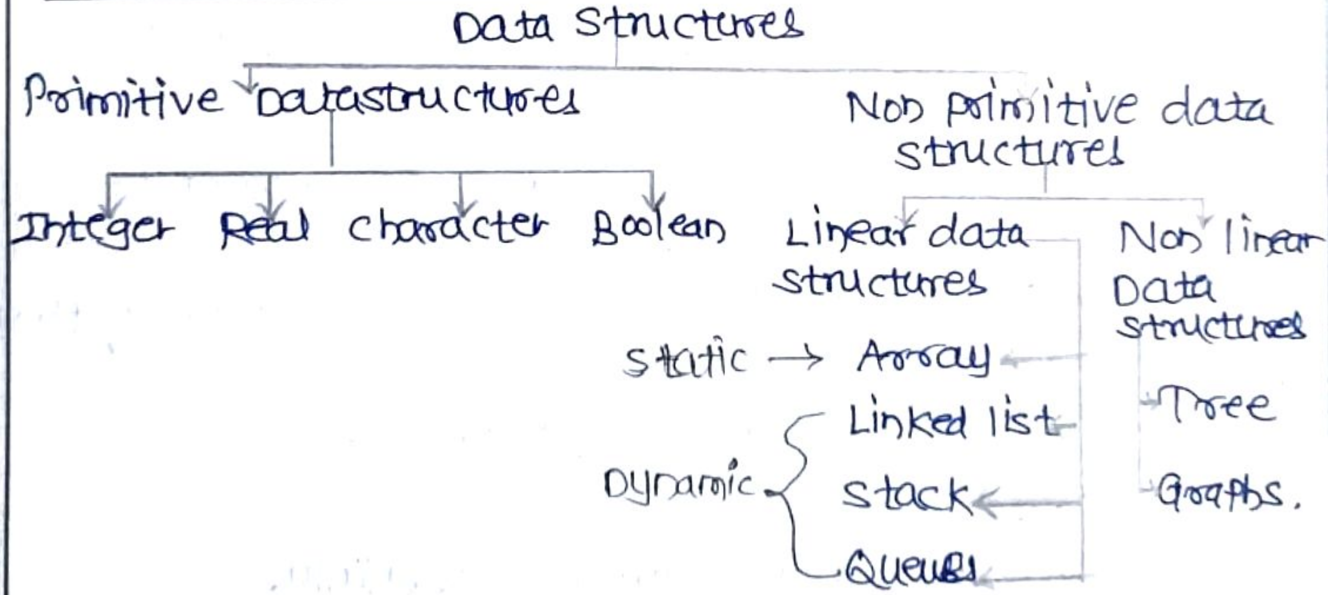
1.1 Data Structures :-

- Data structures is a way of storing and organizing data so that it can be used effeciently.
- This data is organized in the memory.
- Array is one of the data structure in C language. Array is collection of memory elements in which data is stored sequentially.
- A data structure should be seen as a logical concept that must address two fundamental concerns:

1) How the data will be stored

2) What operations will be performed on it.

1.1.1 Types of Data structures :-



A] Primptive data structures :~

- Primptive data structure is a kind of data structure that stores data of only one type.
- Primptive data structure will contain some value i.e it can't be NULL.
- Its size is depends on type of data structure.
- Example: Integer, character, float

B] Non-primitive data structure :~

- Non-primitive data structure is a type of data structure that can stores data of more than one type.
- Non-primitive data structure can consists of NULL value.
- Size is not fixed.
- Examples: Array, linked list, stack, queus.

a) Linear data structure :~

- In linear data structure, arrangement of data is in sequential manner.
- Since elements are arranged sequntially, they are easy to implement.

e.g. Arrays, linked list, stack, Queus.

b) Non-linear data structure :~

- They are not arranged sequentially.

- Instead they are arranged in hierarchical maner.
e.g. Trees, Graph

1.1.2 Major operations that can be performed on DS :-

1) Searching : searching of any element in a data structu

2) Sorting: sorting of element in an ascending or decending order.

3) Insertion:- Insert a new element in data structure.

4) Updation:- Replace the element with another element

5) Deletion:- Remove element from the data structure.

1.1.3 Advantages of data structure :-

i) Data structure helps in efficient storage of data in storage device.

ii) Helpfull for design of efficient algorithms.

iii) Allows easier processing of data.

iv) Data structure provides effective and efficient processing of small as well as large ammount of data.

v) Data structure usage can simply encourage reusability in long run as well.

vi) we can access data anytime and anywhere.

vii) Graphs models real life problems.

1.2 Abstract Data Types:-

i) An abstract data type also abbreviated as ADT is a logical description of how we view data and the operations that are allowed without regard to how they will be implemented.

ii) This means that we are concerned only with what data is representing and not with how its eventually constructed.

1.2.1 Abstraction :-

- Abstraction means displaying only essential information and hiding the details.
- It hides background details or implementation.

e.g. Consider a real life example of man driving a car. The man only knows how to drive a car. He don't know about the inner mechanism of a car.

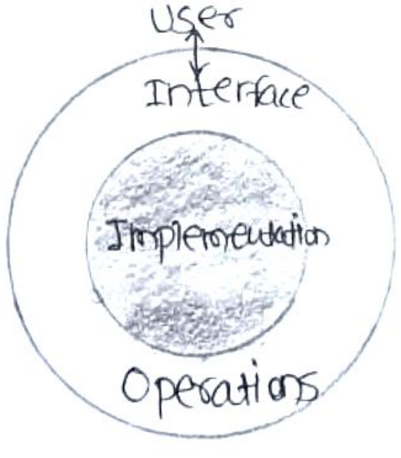


Fig. ADT

1.2.2 Data Type :-

Data type is a way to classify various types of data such as integer, string etc.

A] Built in data type

B] Derived data type

A] Built in data type :-

Those data type for which a language has buit in support are known as built in data types.

e.g. Integer, Boolean, Floating, character, strings.

B] Derived data type :-

Those data type which are implementation independant as they can be implemented in one or other way are known as derived data types.

e.g. List, Array, stack, Queue.

1.3 performance Analysis of an Algorithm :-

i) An algorithm is a step by step set of rules that leads to the final solution.

ii) Algorithm is independent from any programming languages i.e its same for any programming language.

Note:- Refer the Unit - I of CFP for more details of an algorithm

iii) An algorithm is said to be efficient and fast if it takes less time to execute and consumes less memory.

iv) The performance of an algorithm is measured on the basis of following properties:

v) The complexity of an algorithm computes the ammount of time and spaces required by an algorithm for an input of size (n). The complexity of an algorithm can be divided into two types:

1) Time complexity

2) space complexity

1) Time complexity:~

i) Every algorithm requires some ammount of computer time to execute its instructions to perform specific task.

ii) This computer time is called as time complexity.

iii) Time complexity is defined as "The total time required by an algorithm to complete its execution."

iv) Generally, the running time of an algorithm depends upon following:

① Whether it is running on single processor m/c or multiprocessor m/c.

② Whether it is a 32 bit m/c or 64 bit m/c.

③ Read and write speed of machine.

④ The ammount of time required by an algorithm to perform arithmetic operations, logical operatios, return value and assignment operations etc.

⑤ Input data.

v) Time complexity considers how many times each statements executes.

⇒ Is time complexity of an algorithm/code same as running time/execution time of code?

Time complexity of algorithm/code is not equal to the actual time required to execute a particular code, but no. of times a statement executes.

e.g.

Write a C/C++ code to find maximum betn N nos. where N varies from 10, 100, 1000, 10000. In linux if we run the program with the following commands

```
gcc program.c -o program
time ./program
```

Results will be as follows:

for N=10 0.5ms time

for N=10,000 0.2 ms time may require.

This example shows that actual time required to execute code is machine dependent.

consider following 2 scenarios to more understand time complexity.

Suppose you are having one problem and you have 3 algorithms for same problem. Now among these 3 algorithms you want to choose best one. How to choose it?

Solution 1: Run all 3 algorithms on 3 different computer provide same input and find time taken by all three algorithms and choose the one who took less time. But in this solution there is possibility that these systems might be using different processors. So processing speed might vary. so this soln is not efficient.

Solution 2: Run all 3 algorithms on same computer & find which algorithm is taking least time. But here also we might get wrong results because at the time of execution of a program, other things are also running simultaniously. so this soln also not efficient.

so there should be some standard notation to analyze the algorithm. These notations are called as "Asymptotic Notations".

- In asymptotic notation system configuration is not considered. Rather order of growth of i/p will be considered.
- will try to find out how time or space taken by algorithm will increase/decrease after increasing/decreasing input size.
- There are 3 asymptotic notations that are used to represent time complexity of an algorithm.

A] Θ Notation

B] Big O Notation

C] Ω Notation.

Usually time required by an algorithm falls under three types:

① Best case: Minimum time required for program execution.

② Average case: Average time required for program execution.

③ Worst case: Maximum time required for program execution.

Example:-

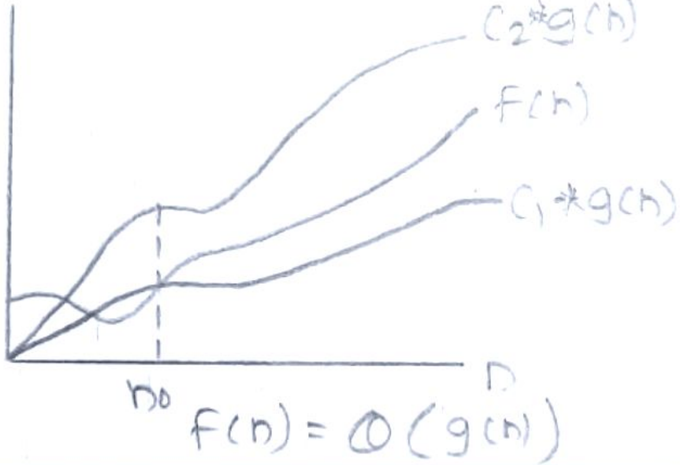
consider an array of size N. If we want to find out one particular element from say 1,2,3,4,5 i.e consider we want to find 1, then we found it

in this scenario. so this will be the best case.
Now suppose array elements are [2,3,4,5,1] and we want to find again 1. It is present but ~~not~~ at last location. so this can be considered as average case.

Again if the array is [2,4,5,3,1] and we are trying to find 6 then this is worst case scenario as 6 is missing in the array.

A] Θ Notation :- Θ is use to find out average bound of an algorithm. i.e it defines upper bound and lower bound and your algorithm will lie betn these levels. so if g(n) is a function then Θ(g(n)) is

$$\Theta(g(n)) = \{ F(n) : \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \leq c_1{*}g(n) \leq F(n) \leq c_2{*}g(n) \text{ for all } n \geq 0 \}$$

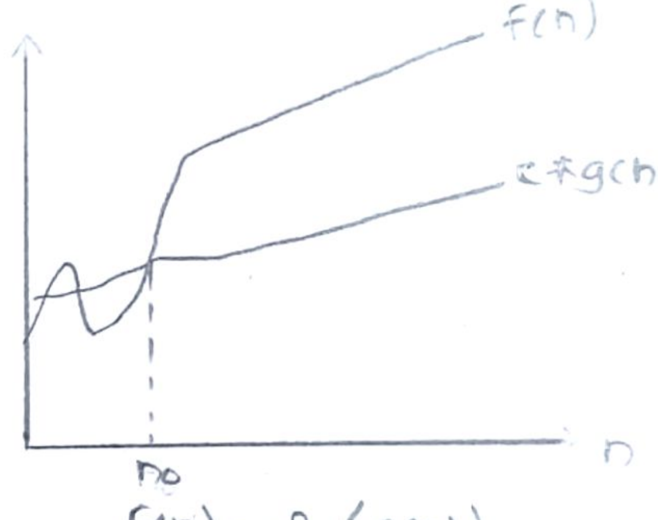


F(n) = Θ(g(n))

B] Ω Notation:-

- The Ω notation denotes the lower bound of an algorithm i.e the time taken by the algorithm can't be lower than this.
- In other words this is the fastest time taken by the algorithm when provided with best case i/P.
- If g(n) is a function then $\Omega(g(n))$ given as:

$$\Omega(g(n)) = \{ f(n): \text{there exists positive constants } c \,\&\, n_0 \text{ such that } 0 < c*g(n) \leq f(n) \text{ for all } n \geq n_0 \}$$

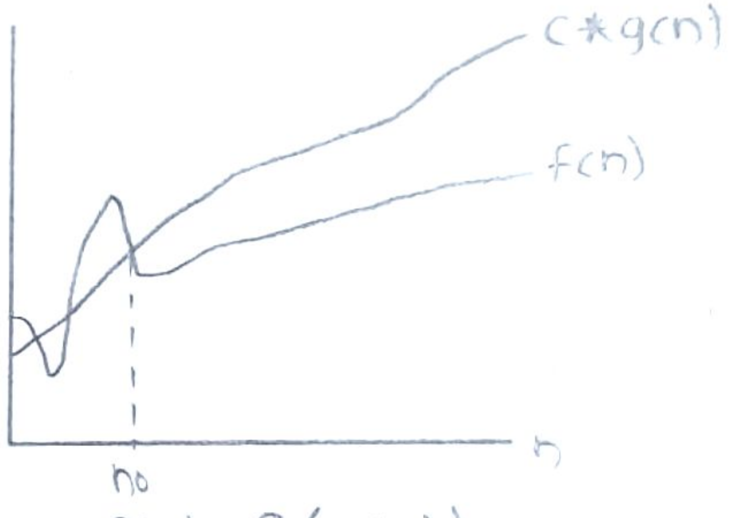


$f(n) = \Omega(g(n))$

C] **Big O Notation:-** The Big O Notation defines the upper bound of any algorithm i.e your algorithm can't take more time than this time.

- In other words we can say that big O denotes maxm time taken by an algorithm.
- The big O notation is the most used notation for time complexity of an algorithm.
- If g(n) is the function then O(g(n)) given as:

$$O(g(n)) = \{f(n) : \text{there exists positive constants } c \text{ \& } n_0 \text{ such that } 0 \leq f(n) \leq c*g(n) \text{ for all } n \geq n_0\}$$



f(n) = O(g(n))

1.4 Common Asymptotic Notations:-

Following is the list of some common asymptotic notation.

Name	Notations	Examples
constant	$O(1)$	odd or even no.
logarithmic	$O(\log n)$	Finding element on sorted array with binary search.
linear	$O(n)$	Find max element in unsorted array.
$n \log n$	$O(n \log n)$	sorting elements in ~~unsorted~~ array with merge sort.
quadratic	$O(n^2)$	sort array with bubble sort.
cubic	$O(n^3)$	
polynomial	$n^{O(1)}$	

Topic .... Unit No. ....

* How to calculate time complexity?

Time taken by simple statement is constant like:

```
let i=0;
i=i+1;
```

This constant time is considered as Big O of 1 i.e O(1).

Example 1:-

```
for(i=0; i<N; i++)
  {
    statement;
  }
```

The time complexity for above algorithm will be linear. The running time of loop is directly proportional to N.

Example 2:-

```
for(i=0; i<N; i++)
  {
    for(j=0; j<N; j++)
      {
       statement;
      }
  }
```

Time complexity for above code is Quadratic. The running time of two loops is proportional to the square of N. When N doubles running time increases by N*N

2) Space complexity:-

- Space complexity is nothing but amount of memory space that an algorithm or a problem takes during execution of that particular problem.
- The space complexity is not only calculated by space used by variables in the problem but it also includes & considers the space for input values in it.

e.g. sum of N natural number

```
int sum(int n)
  {
    int i, sum=0;
    for (i=n; i>1; i--)
     sum=sum+i;
    return sum;
  }
```

In above example, i/p value is 'n' that is constant which will take the space of O(1). So total space complexity is O(1).

Topic .... Unit No. ....

* How to calculate space complexity of an algorithm?

Example 1: Addition of Numbers

```
{
  int a = x+y+z;
  return(a);
}
```

In above example, there are 4 integer variables those are a, x, y, z. So they will take 4 bytes space for each variable. Also extra 4 byte space will also be added to total space complexity for return value i.e a.

∴ Total space complexity = 4*4+4 = 20 bytes.

But for this example, this is fixed complexity and because of same variable inputs such space complexities are considered as "constant space complexity". i.e O(1).

Example 2: sum of all elements in array

```
func_sumofnumbers(arr[], N)
  {
    sum=0;
  for(i=0; i≤n; i++){
     sum= sum+arr[i]
   }
  Printf("sumis", sum);
}
```

Topic .... Unit No. ....

Here,

1. In array (arr) the size of array is "N" and each element will take "4 bytes" so space taken by arr is N*4 bytes.

2. Sum variable takes "4 bytes".

3. i variable is used to iterate over all the elements in the array. So it will take "4 bytes".

4. for loop & printf function will combinely take "4 bytes".

∴ Total space complexity = (4*N+12) bytes.

But these 12 bytes are constant so we will not consider it and after removing all constants (4 from 4*N) we can finally say that this algorithm have complexity of O(N).