Algorithm

```
Program analysis tool:
1.Flow Chart
2.Algorithm
```

An algorithm is a sequence of unambiguous instructions for solving a problem. It takes some valid input and produces the desired output in a finite amount of time.

Two widely used approaches used to specify an algorithm are:

- Natural Language representation
- Pseudo code.

Natural language representation causes ambiguity and difficulty to code in any programming language because of unclear definition.

Pseudo code is a mixture of natural language and programming language constructs. It is precise and gives clear description of the algorithm.

Criterion to write an algorithm:

1. Definiteness: Each instruction of an algorithm should be clear and written unambiguously.
2. Finiteness: Any repetition of statements must be terminated after a finite number of steps.
3. Effectiveness: Every instruction must have some effect towards the solution of the problem.

Example1 : Write an algorithm to add all the elements of an array with n elements.

```
ADDELEMENTS(A,n):
```

//This algorithm returns the sum of all elements of an array A[1. . . n] of n elements

```
1.  sum = 0
2.  for  k = 1 to n STEP 1
3.     sum = sum + A[ k ]
4. return sum.
```

Example 2 : Write an algorithm to search number in an n-element array.

```
LSEARCH(A, n, num):
```

//This algorithm returns the location of num in an array A of n elements and return 0 otherwise.

```
1.  loc = 0
2.  while  loc <= n  AND number ≠ A [ loc ]
3.     loc = loc + 1
4. return loc.
```

Analysis of Algorithm:

For one problem, there exist many algorithms. To find the efficient one, analysis is required. Algorithm analysis includes the time, space requirement by an algorithm in order to solve a particular problem.

Time complexity is the amount of time needed by an algorithm to solve a problem. Running time here means not the clock time, but number of execution steps involved in the algorithm.

space complexity is the amount of memory spaces used by an algorithm.

Measuring the running time of the algorithm

ADDELEMENTS(A,n):

This algorithm computes the sum of all elements of an array A[1….n] of n elements and returns the sum.

```
1. sum = 0                          - - - - -       1
2. FOR  k = 1 to n STEP 1           - - - - -       n+1
3.      sum = sum + A[ k ]          - -  - -        n
4. RETURN sum.                      - -  -  -       1
                                        ----------------
                                    T(n) =    2n + 3
```

Best- case, worst-case and average- case efficiency:

There are some kinds of problem, where same algorithm behaves differently. This different behavior is due to the input instances. For some input instance, the algorithm computes the problem quickly and for some other input instance algorithm runs slow.

Best- case of the algorithm occurs when an algorithm, for an input instance, takes minimum amount of time to compute the problem.

Worst- case of the algorithm occurs when an algorithm, for an input instance, takes maximum amount of time to compute the problem

Average- case of the algorithm occurs when an algorithm, for an input instance, takes average amount of time to compute the problem. It is difficult to calculate average running time of an algorithm.

Asymptotic Notation

Asymptotic notations are used to describe the running time of an algorithm and compare the order growth of two functions. Following are the notations used to specify the running time.

Notations	Symbols	Gives
Big Oh notation	O	Upper bound (tight)
Big Omega notation	Ω	Lower Bound (Tight)
Theta notation	θ	Tight bound
Little Oh notation	o	Upper bound
Little Omega notation	ω	Lower Bound

Big Oh Notation: This notation gives an upper bound to the asymptotic growth of the function.

Definition**:**

If f(n) and g(n) are two functions defined for non-negative integers, then f(n)=O(g(n)), if and only if there exists two positive constants c and n0 such that $0 \le f(n) \le c \times g(n)$ for all n >= n0.

Problem: The running time of an algorithm is described by a function f(n) = 3n + 4. Find the Big Oh notation and find the value of the constant c and n_0.

Solution:

$$f(n) = 3n + 4$$
$$\Rightarrow f(n) \le 3n + n$$
$$\Rightarrow f(n) \le 4n$$
$$\Rightarrow f(n) = O(n), \quad and\ c = 4$$

n	f(n) 3n+4	Cxg(n) 4xn	$f(n) \le c \times g(n)$
0	4	0	False
1	7	4	False
2	10	8	False
3	13	12	False
4	16	16	True
5	19	20	True

The relation f() ≤ c* g(n) is true at n = 4 and continued to be true for any value of n ≥ 4. That means the growth rate of f(n) is less than equal to growth rate of c*g(n) for all value of n that is greater than or equal to 4. So value of n0 = 4.

So the running time of the algorithm which is expressed by linear function f(n) = 3n + 4 is O(n).

Exercise: Find asymptotic growth of following functions by using Big Oh notation.

1. $f(n) = 2n^2 + 3n + 5$
2. $f(n) = 2^n + 5n$
3. $f(n) = 5000$

Big Omega Notation: This notation gives a lower bound to the asymptotic growth of the function.

Definition:

If f(n) and g(n) are two functions defined for non-negative integers, then f(n) = Ω(g(n)) if and only if there exists two positive constants c and n_0 such that $f(n) \geq c \times g(n)$ for all n >= n_0.

$f(n) = 3n + 4$
$\Rightarrow f(n) \geq 3n$
$\Rightarrow f(n) = \Omega(n),$
$Where \quad c = 3$

Theta Notation: This notation gives both tighter upper bound and tighter lower bound to the asymptotic growth of the function.

Definition:

If f(n) and g(n) are two functions defined for non-negative integers, then f(n) = θ(g(n)) if and only if there exists three positive constants c1,c2 and n_0 such that $c1 \times g(n) \leq f(n) \leq c2 \times g(n)$ *for all n >= n_0.*

To find time complexity of a function in terms of theta notation, we need to find a lower bound i, e $c1 \times g(n) \leq f(n)$ and an upper bound i.e. $f(n) \leq c2 \times g(n)$

Lets the function given is f(n) =3n+4

To find a tighter lower bound,

$f(n) = 3n + 4$

$\Rightarrow f(n) \geq 3n$ …… ……..(eq. 1)

So $f(n) = \Omega(n),$ and the constant c1 =3.

To find a tighter upper bound,

$f(n) = 3n + 4$
$\Rightarrow f(n) \leq 3n + n$
$\Rightarrow f(n) \leq 4n$ ……(eq. 2)

So f(n) = O(n) and the constant c2 = 4

Form eq. 1 and eq.2 we get $3 \times n \leq f(n) \leq 4 \times n$

So we can write f(n) = θ(g(n)) and the constant c1 = 3 and c2 = 4.

Little Oh Notation:

Like Big Oh, this notation also gives an upper bound to the asymptotic growth of the function. But this notation does not give the tighter bound.

Definition:

If f(n) and g(n) are two functions defined for non-negative integers, then f(n) =o(g(n)) if and only if there exists two positive constants c and n_0 such that $0 \leq f(n) < c \times g(n)$ for all n >= n_0.

Little Omega Notation:

Like Big Omega, this notation also gives a lower bound to the asymptotic growth of the function. But this does not give a tighter lower bound.

Definition:

If f(n) and g(n) are two functions defined for non-negative integers, then f(n) = ω (g(n)) if and only if there exists two positive constants c and n_0 such that $f(n) > c \times g(n)$ *for all n >= n_0.*

Introduction to Data Structure

Data Type:

Data type of a variable or literals specifies the **type of data** it contains and the **operation** that can be performed on it.

More technically, a data type says about the storage format, memory size, range of values and the applicable operations.

Example: int data type in C can take whole numbers with 4 byte memory space and the applicable operations are addition, subtractions, multiplication, division, modulo division

There are two types of Data type:

1. Scalar/simple: It holds a simple piece of data value (atomic). Example: int, char, float, Boolean etc

2. Structured: Holds a collection of data values .Example –Array, record (struct), string, class etc

Data Structure:

Logical and mathematical model of organization of data is called Data structure.

Data structure consists of two things: A **collection of simple or structure data type** and a **set of rules** to organize and accessing the collection.

Data structures are classified into linear and non-linear data structures based on arrangement of data.

Linear data structure:

The data items in linear data structure form a sequence and maintain a linear ordering. Data are arranged in linear fashion though their memory locations may not be sequential.

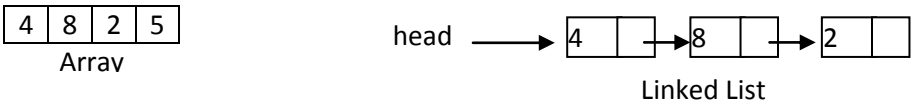
In this data structure, the data/elements are traversed sequentially one after another and only one element can be directly accessible at each step. Example: Array, Linked list, stack, queue

Non-Linear Data structure:

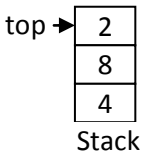
In Non-linear data structure, data are not arranged in a sequence. It is constructed by attaching a data element to several other data element in such way that at one-step multiple data element can be accessible. Example Tree, Graph.

Arrays : Array contains multiple numbers of homogenous data in adjacent memory location. It supports direct access to its elements.

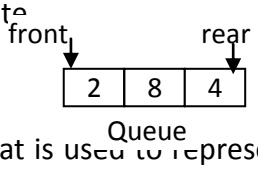
Linked list: Linked list data structure consists of nodes, each containing data and link pointing to the next node in the sequence. It does not allow direct access to a node.



Stack : Stack is a restricted linear data structure in which insertion and deletion of data take place at one end. Stack is called a LIFO List (Last In First Out) as the items inserted last will be processed first.



Queue: Queue is a restricted linear data structure in which insertion of data takes place at one end and deletion of data takes place in other end. Queue is a FIFO List (First In First Out) list as the items inserted first is the first item to delete.



Tree : Tree is a non linear data structure that is used to represents the data in a hierarchical relationship. A node may have multiple successors (children nodes) but only one predecessor (parent node)

Graph: Graph is a nonlinear data structure, consists of a set of nodes and a set of links. A node is called vertex. A link is called the edge that connects two vertices. In Graph, relationship among the nodes is less restricted. It means a node can have multiple predecessors.