UNIT-I
INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

Definition: - An algorithm is a ***Step By Step*** process to solve a problem, where each step indicates an intermediate task. Algorithm contains finite number of steps that leads to the solution of the problem.

Properties /Characteristics of an Algorithm:-

Algorithm has the following basic properties

- Input-Output:- Algorithm takes '0' or more input and produces the required output. This is the basic characteristic of an algorithm.
- Finiteness:- An algorithm must terminate in countable number of steps.
- Definiteness: Each step of an algorithm must be stated clearly and unambiguously.
- Effectiveness: Each and every step in an algorithm can be converted in to programming language statement.
- Generality: Algorithm is generalized one. It works on all set of inputs and provides the required output. In other words it is not restricted to a single input value.

Categories of Algorithm:

Based on the different types of steps in an Algorithm, it can be divided into three categories, namely

- Sequence
- Selection and
- Iteration

Sequence: The steps described in an algorithm are performed successively one by one without skipping any step. The sequence of steps defined in an algorithm should be simple and easy to understand. Each instruction of such an algorithm is executed, because no selection procedure or conditional branching exists in a sequence algorithm.

Example:

// adding two numbers
Step 1: start
Step 2: read a,b
Step 3: Sum=a+b
Step 4: write Sum
Step 5: stop

Selection: The sequence type of algorithms are not sufficient to solve the problems, which involves decision and conditions. In order to solve the problem which involve decision making or option selection, we go for Selection type of algorithm. The general format of Selection type of statement is as shown below:

```
if(condition)
  Statement-1;
else
  Statement-2;
```

The above syntax specifies that if the condition is true, statement-1 will be executed otherwise statement-2 will be executed. In case the operation is unsuccessful. Then sequence of algorithm should be changed/ corrected in such a way that the system will re-execute until the operation is successful.

Example1:

```
// Person eligibility for vote
Step 1 : start
Step 2 : read age
Step 3 :  if age > = 18 then step_4 else step_5
Step 4 : write "person is eligible for vote"
Step 5 : write " person is not eligible for vote"
Step 6 : stop
```

Example2:

```
// biggest among two numbers
Step 1 : start
Step 2 : read a,b
Step 3 :  if a > b then
Step 4 : write "a is greater than b"
Step 5 : else
Step 6 : write "b is greater than a"
Step 7 : stop
```

Iteration: Iteration type algorithms are used in solving the problems which involves repetition of statement. In this type of algorithms, a particular number of statements are repeated 'n' no. of times.

Example1:

Step 1 : start
Step 2 : read n
Step 3 : repeat step 4 until n>0
Step 4 : (a) r=n mod 10
(b) s=s+r
(c) n=n/10
Step 5 : write s
Step 6 : stop

Performance Analysis an Algorithm:

The Efficiency of an Algorithm can be measured by the following metrics.

i. Time Complexity and
ii. Space Complexity.

i.Time Complexity:

The amount of time required for an algorithm to complete its execution is its time complexity. An algorithm is said to be efficient if it takes the minimum (reasonable) amount of time to complete its execution.

ii. Space Complexity:

The amount of space occupied by an algorithm is known as Space Complexity. An algorithm is said to be efficient if it occupies less space and required the minimum amount of time to complete its execution.

1. Write an algorithm for roots of a Quadratic Equation?

// Roots of a quadratic Equation
Step 1 : start
Step 2 : read a,b,c
Step 3 : if (a= 0) then step 4 else step 5
Step 4 : Write " Given equation is a linear equation "
Step 5 : d=(b * b) _ (4 *a *c)
Step 6 : if (d>0) then step 7 else step8
Step 7 : Write " Roots are real and Distinct"
Step 8: if(d=0) then step 9 else step 10
Step 9: Write "Roots are real and equal"
Step 10: Write " Roots are Imaginary"
Step 11: stop

2. Write an algorithm to find the largest among three different numbers entered by user

Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4: If a>b
    If a>c
      Display a is the largest number.
    Else
      Display c is the largest number.
  Else
    If b>c
      Display b is the largest number.
    Else
      Display c is the greatest number.
Step 5: Stop

3. Write an algorithm to find the factorial of a number entered by user.

Step 1: Start
Step 2: Declare variables n,factorial and i.
Step 3: Initialize variables
    factorial←1
    i←1
Step 4: Read value of n
Step 5: Repeat the steps until i=n
  5.1: factorial←factorial*i
  5.2: i←i+1
Step 6: Display factorial
Step 7: Stop

4. Write an algorithm to find the Simple Interest for given Time and Rate of Interest .

Step 1: Start
Step 2: Read P,R,S,T.
Step 3: Calculate S=(PTR)/100
Step 4: Print S
Step 5: Stop

ASYMPTOTIC NOTATIONS

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as $f(n)$ and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases. Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

The time required by an algorithm falls under three types −

- **Best Case** − Minimum time required for program execution.
- **Average Case** − Average time required for program execution.
- **Worst Case** − Maximum time required for program execution.

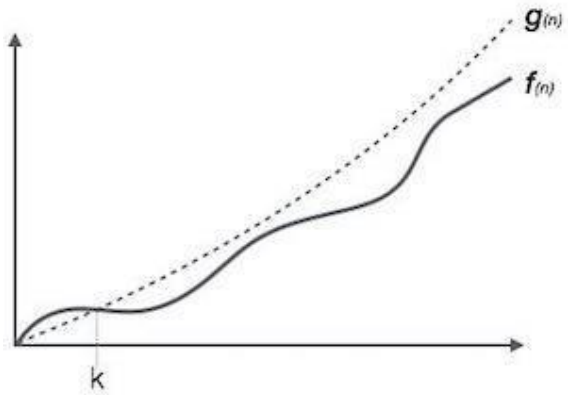
Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- θ Notation

Big Oh Notation, O

The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

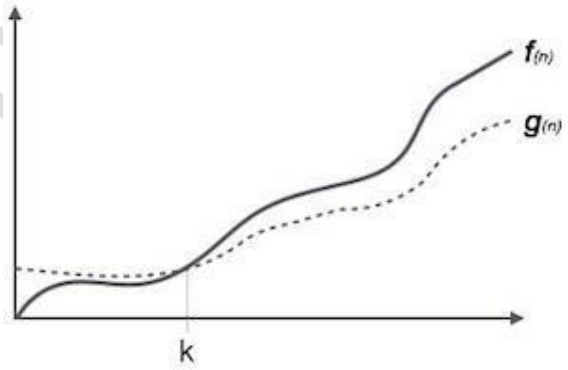


For example, for a function ***f*(n)**

O(*f*(n)) = { *g*(n) : there exists c > 0 and n_0 such that *f*(n) ≤ c.*g*(n) for all n > n_0. }

Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

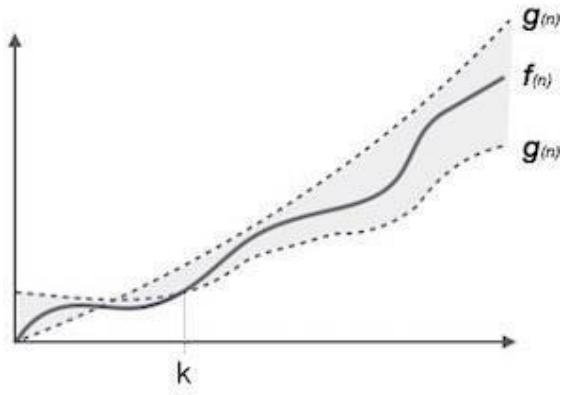


For example, for a function ***f*(n)**

Ω(*f*(n)) ≥ { *g*(n) : there exists c > 0 and n_0 such that *g*(n) ≤ c.*f*(n) for all n > n_0. }

Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −



θ(*f*(n)) = { *g*(n) if and only if *g*(n) = O(*f*(n)) and *g*(n) = Ω(*f*(n)) for all n > n_0. }

DATA STRUCTURES

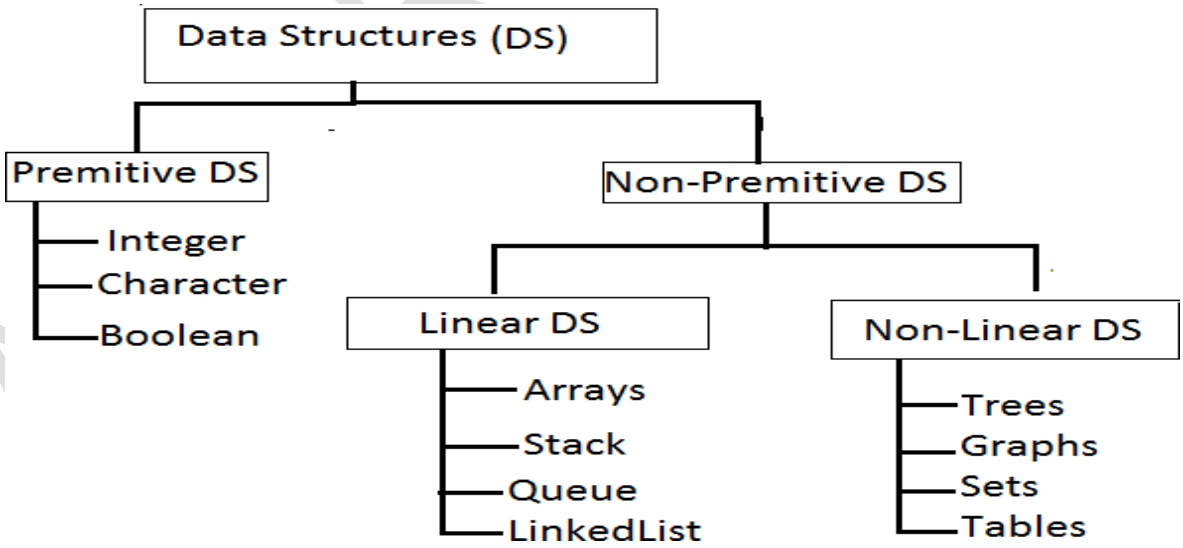
Data may be organized in many different ways logical or mathematical model of a program particularly organization of data. This organized data is called "Data Structure".

Or

The organized collection of data is called a 'Data Structure'.

Data Structure=Organized data +Allowed operations

Data Structure involves two complementary goals. The first goal is to identify and develop useful, mathematical entities and operations and to determine what class of problems can be solved by using these entities and operations. The second goal is to determine representation for those abstract entities to implement abstract operations on this concrete representation.



Primitive Data structures are directly supported by the language ie; any operation is directly performed in these data items.

Ex: integer, Character, Real numbers etc.

Non-primitive data types are not defined by the programming language, but are instead created by the programmer.

Linear data structures organize their data elements in a linear fashion, where data elements are attached one after the other. Linear data structures are very easy to implement, since the memory of the computer is also organized in a linear fashion. Some commonly used linear data structures are arrays, linked lists, stacks and queues.

In nonlinear data structures, data elements are not organized in a sequential fashion. Data structures like multidimensional arrays, trees, graphs, tables and sets are some examples of widely used nonlinear data structures.

Operations on the Data Structures:

Following operations can be performed on the data structures:

1. Traversing
2. Searching
3. Inserting
4. Deleting
5. Sorting
6. Merging

1. Traversing- It is used to access each data item exactly once so that it can be processed.

2. Searching- It is used to find out the location of the data item if it exists in the given collection of data items.

3. Inserting- It is used to add a new data item in the given collection of data items.

4. Deleting- It is used to delete an existing data item from the given collection of data items.

5. Sorting- It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

6. Merging- It is used to combine the data items of two sorted files into single file in the sorted form.