Unit 2: Algorithm

a. Concept and Definition
b. Design of algorithm
c. Characteristic of algorithm
d. Big O notation

Concept and definition

- "An algorithm is a finite set of precise instructions executed in finite time for performing a computation or for solving a problem."
- To represent an algorithm we use English language however for the simplicity we can use ***pseudocode*** to represent the algorithm.
- ***Pseudocode*** represents an algorithm in a clear understandable manner as like English language and gives the implementation view as like programming language.

Example: write an algorithm for finding the factorial of a given number.

```
fact(n)
        {
        fact=1;
        for(i=1;i<=n; i++)
                fact=fact*i;
        return fact;
        }
```

This algorithm is for getting the factorial of n. The algorithm first assigns the value 1 to the variable *fact* and then until the n is reached, the *fact* is assigned a value *fact*i* where value of *i* is from *1* to *n*. At last the value *fact* is returned. The pseudocode used here is loosely based on the programming language C.

Characteristics (properties) of algorithm

- **Input:** An algorithm has input values from a specified set.
- **Output:** From each set of input values, an algorithm produces output values from a specified set. The output values are the solution to the problem.
- **Definiteness:** The steps of an algorithm must be defined precisely.
- **Correctness:** An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
- **Effectiveness:** It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
- **Generality:** The devised algorithm should be capable of solving the problem of similar kind for all possible inputs.

- The above algorithm for factorial satisfies all the properties of algorithm as follows:
  - The input for the algorithm is any positive integer and output is the factorial of the given number.
  - Each step is clear since assignments, finite loop, the arithmetic operations and jump statements are defined precisely.
  - Each input gives its corresponding factorial value after a finite number of steps → Correctness.
  - When the return statement is reached, the algorithm terminates and each step in the algorithm terminates in finite time. All other steps, other than the loop are simple and require no explanation. In case of loop, when the value of *i* reaches *n+1*, then the loop terminates.
  - The algorithm is general since it can work out for every positive integer.

Model question (2008)

- What are the major characteristics of algorithms?

Complexity of algorithms

- When an algorithm is designed, it must be analyzed for its **efficiency**.
- The efficiency of an algorithm is measured in terms of **complexity**.
- The **complexity** of algorithms is mentioned in terms of **computational resource** needed by the algorithm.
- We generally consider two kinds of resources used by an algorithm: **TIME** (related to computer's processing power) and **SPACE** (related to computer's primary memory).

Complexity of algorithms…

- The measure of time required by an algorithm to run is given by **time complexity** and the measure of space (computer memory) required by an algorithm is given by **space complexity**.
- We focus mostly on **time complexity** of an algorithm which is given by the number of operations needed by an algorithm for a given set of inputs.
- Since actual time required may vary from computers to computers (e.g. on a supercomputer it may be million times faster than on a PC), we are using the number of operations required to measure the **time complexity**.

- **Example (Time complexity):**
  - We can find the time complexity of the above algorithm by enumerating the number of operations required for the algorithm to obtain the factorial of *n*.
  - We can clearly see that the assignment of the variable *fact* is done once, the *for* loop executes n+1 times, the statement inside the *for* loop executes n times and the last operation that is the return statement executes single time.
  - Thus the total time in terms of number of operations required for finding the factorial of *n* can be written as:

    $f(n) = 1 + (n+1) + n + 1 = 2n+3.$

Big-Oh (**O**) notation

- The complexity analysis (**TIME**) of an algorithm is very hard if we try to analyze exact.
- So, the complexity of an algorithm is analyzed in terms of a mathematical function of the size of the input.
- The algorithm is analyzed in terms of bound (upper and lower), by understanding the growth of the function.
- For this purpose we need the concept of asymptotic notations.
- Big-O notation is one of the asymptotic notation used for complexity analysis of an algorithm.

Big-O notation…

- When we need only asymptotic upper bound, then we use Big-O notation.
- *Note: When we need asymptotic lower bound, then we use Big-Omega (**Ω**) notation.*
- *Note: When we need asymptotically tight bound, then we use Big-Theta (**θ**) notation.*
- Definition: Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that *f(n)* is O(g(n)) (read as *f(n)* is big-oh of *g(n)*) if there are constants *C* and *k* such that

$$|f(n)| \leq C^{*}|g(n)|$$

whenever $n > k$.

- For example: In the above algorithm

  $f(n) = 2n+3 \leq 6.n$

  so that we can infer that $f(n) = O(g(n))$ with $g(n) = n$, $C=6$ and $n > 1$.

  Hence the time complexity of the factorial algorithm is $O(n)$ i.e. linear complexity.

Categories of algorithms

- Based on Big-O notation, the algorithms can be categorized as follows:
  - **Constant time algorithms → $O(1)$**
  - **Logarithmic time algorithms → $O(\log n)$**
  - **Linear time algorithms → $O(n)$**
  - **Linearithmetic time algorithms → $O(n \log n)$**
  - **Polynomial time algorithms → $O(n^k)$ for $k>1$**
  - **Exponential time algorithms → $O(k^n)$ for $k>1$**

Increasing order of complexity

Worst, Best and Average Case Complexity

- The *worst-case complexity* of the algorithm is defined by the maximum number of steps taken on any instance of size *n*.
- The *best-case complexity* of the algorithm is defined by the minimum number of steps taken on any instance of size *n*.
- The *average-case complexity* of the algorithm is defined by the average number of steps taken on any random instance of size *n*.

- Note: In the case of running time, the *worst-case time complexity* indicates the longest running time performed by an algorithm for any input of size *n*, and thus this guarantees that the algorithm finishes on time.
- Note: Algorithms are analyzed for their efficiency mostly in terms of its *worst-case complexity*.