**NOTES**

**1.Algorithm:**

“An algorithm is “a step-by-step procedure for accomplishing some task'' An algorithm can be given in many ways. For example, it can be written down in English (or French, or any other ''natural'' language). However, we are interested in algorithms which have been precisely specified using an appropriate mathematical formalism--such as a programming language.”

**2.Characteristics of a good algorithm:**

Every algorithm should have the following five characteristics:

1.**Input**: The algorithm should take zero or more input.

2. **Output**: The algorithm should produce one or more outputs.

3. **Definiteness**: Each and every step of algorithm should be defined unambiguously.

4. **Effectiveness**: A human should be able to calculate the values involved in the procedure of the algorithm using paper and pencil.

5. **Termination**: An algorithm must terminate after a finite number of steps.

**3.Complexity of an algorithm** :

The complexity of an algorithm M is the function f(n) which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data. Frequently, the storage space required by an algorithm is simply a multiple of the data size n. Therefore, the term “complexity” shall refer to the running time of the algorithm. We find the complexity function f(n) for certain cases. The two cases one usually investigates in complexity theory are :- i. Worst case:- the maximum value of f(n) for any possible input ii. Best case:- the minimum possible value of f(n). Complexity of an Algorithm An algorithm is a sequence of steps to solve a problem; there may be more than one algorithm to solve a problem. The choice of a particular algorithm depends upon following consideration:- 1) Time Complexity 2) Space Complexity Time Complexity:- The time complexity of an algorithm is the amount of time it needs to run to completion. Some of the reasons for studying time complexity are:- We may be interested to know in advance whether the program will provide a satisfactory real time response. There may be several possible solutions with different time requirement. Space Complexity:- The space complexity of an algorithm is the amount of memory it needs to run to completion. Some of the reasons to study space complexity are: - There may be several possible solutions with in different space requirement. To estimate the size of the largest problem that a program can solve.

**4.Find the complexity of an algorithm &**

**relation between the time and space complexities of an algorithm:**

Complexity of an algorithm is the measure of analysis of algorithm. Analyzing an algorithm means predicting the resources that the algorithm requires such as memory, communication bandwidth, logic gates and time. Most often it is computational time that is measured for finding a more suitable algorithm. This is known as time complexity of the algorithm. The running time of a program is described as a function of the size of its input. On a particular input, it is traditionally measured as the number of primitive operations or steps executed. The analysis of algorithm focuses on time complexity and space complexity. As compared to time analysis, the analysis of space requirement for an algorithm is generally easier, but wherever necessary, both the techniques are used. The space is referred to as storage required in addition to the space required storing the input data. The amount of memory needed by program to run to completion is referred to as space complexity. For an algorithm, time complexity depends upon the size of the input, thus, it is a function of input size ‘n’. So the amount of time needed by an algorithm to run to its completion is referred as time complexity. The best algorithm to solve a given problem is one that requires less memory and takes less time to complete its execution. But in practice it is not always possible to achieve both of these objectives. There may be more than one approach to solve a same problem. One such approach may require more space but takes less time to complete its execution while the other approach requires less space but more time to complete its execution. Thus we may have to compromise one to improve the other. That is, we may be able to reduce space requirement by increasing running time or reducing running time by allocating more memory space. This situation where we compromise one to better the other is known as **Time-space tradeoff.** The best algorithm to solve a given problem is one that requires less space in memory and takes less time to complete its execution. But in practice it is not always possible to achieve both of these objectives. There may be more than one approach to solve a problem. One approach may require more space but less time to complete its execution. The 2nd approach may require less space but takes more time to complete execution. We choose 1st approach if time is a constraint and 2nd approach if space is a constraint. Thus we may have to sacrifice one at cost of the other. That is what we can say that there exists a time space trade among algorithm. Q

**5.Use a symptotic notation in the study of algorithm ,commonly used asymptotic notations and their significance.**

The running time of an algorithm depends upon various characteristics and slight variation in the characteristics varies the running time. The algorithm efficiency and performance in comparison to alternate algorithm is best described by the order of growth of the running time of an algorithm. Suppose one algorithm for a problem has time complexity as c3n 2 and another algorithm has c1n 3 +c2n 2 then it can be easily observed that the algorithm with complexity c3n 2 will be faster than the one with complexity c1n 3 +c2n 2 for sufficiently larger values of n. Whatever be the value of c1, c2 and c3 there will be an ‘n’ beyond which the algorithm with complexity c3n 2 is faster than algorithm with complexity c1n 3 +c2n 2 , we refer this n as breakeven point. It is difficult to measure the correct breakeven point analytically, so Asymptotic notation are introduced that describe the algorithm efficiency and performance in a meaningful way.

These notations describe the behavior of time or space complexity for large instance characteristics. Some commonly used asymptotic notations are:

**Big oh notation (O):** The upper bound for the function ‘f’ is provided by the big oh notation (O). Considering ‘g’ to be a function from the non-negative integers to the positive real numbers, we define O(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n0 , f(n)≤cg(n) for all n≥n0. Mathematically, O(g(n))={f(n): there exists positive constants such that 0≤f f(n)≤cg(n) for all n, n≥n0} , we say “f is oh of g”.

**Big Omega notation (Ω):** The lower bound for the function ‘f’ is provided by the big omega notation (Ω). Considering ‘g’ to be a function from the non-negative integers to the positive real numbers, we define Ω(g) as the set of function f such that for some real constant c>0 and some non negative integers constant n0 , f(n)≥cg(n) for all n≥n0. Mathematically, Ω(g(n))={f(n): there exists positive constants such that 0≤cg(n) ≤f(n) for all n, n≥n0}.

**Big Theta notation (θ):** The upper and lower bound for the function ‘f’ is provided by the big oh notation (θ). Considering ‘g’ to be a function from the non-negative integers to the positive real numbers, we define θ(g) as the set of function f such that for some real constant c1 and c2 >0 and some non negative integers constant n0 , c1g(n)≤f f(n)≤c2g(n) for all n≥n0. Mathematically, θ(g(n))={f(n): there exists positive constants c1 and c2 such that c1g(n)≤f f(n)≤c2g(n) for all n, n≥n0} , we say “f is oh of g