1. **Classify two forms of Analysis of Time Complexity.**

Analysis of Time Complexity cane be given by two forms,

1.Mathematical Analysis: calculates basic operation and find recurrence relation. Then it uses any of the 2 methods to find its Time Complexity. These methods calculate efficiency (Complexity) of Recursive Algorithms only.

- Master Theorem
- Substitution Method

2.Empirical Analysis: Empirical Analysis and Summation (Mathematical Analysis) can be used to find efficiency(complexity) of Non-Recursive algorithm only. It is solved by Frequency Count Method by,

- Counting the number of loops
- Iterations made by all the loops and statements within the loops.

2. **Explain about Time Complexity in Analysis.**

Time complexity of an algorithm refers to the amount of time required by the algorithm to execute and get the result. It is determined by 2 parts.

$$f(n) = C + T(I)$$

• **Constant time part(C):** Any instruction that is executed just once comes in this part. Eg:Arithmetic Operations

• **Variable Time Part(T(I)):** Any instruction that is executed more than once, say n times, comes in this part. Eg:Loops,Recursion.

3. **Explain about Space Complexity in Analysis.**

Space Complexity of an algorithm refers to the amount of memory required by the algorithm to store the variables and get the result. It is determined by 2 parts.

$$S(n) = C + S(I)$$

Fixed Part (C): This refers to the space required by the algorithm. For example, input variables

Variable PartS(I): This refers to the space that can be different based on the implementation of the algorithm. For example:recursion stack.

4. What are the Fundamentals of Algorithmic Problem Solving?

- Understanding the Problem
- Choose an appropriate algorithmic approach and select the right data structures for efficiency.
- Choose Methods of Specifying an Algorithm; E.g.: Flow Chart,
- Proving an Algorithm's Correctness
- Analysis of Algorithm Complexity based on Time and Space.
- Coding an Algorithm:

5. What are the methods to specify an algorithm.

There are three ways to specify an algorithm.

1. **Natural Language:** We express the Algorithm in the natural English language.
2. **Flowchart:** We express the Algorithm by making a graphical/pictorial representation containing descriptions of the algorithm's steps.
3. **Pseudo Code:** Pseudocode is a mixture of a natural language and programming language constructs like if condition, while and for loops. But it has no syntax.

6. Classify Asymptotic Notations and its functions.

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis. There are mainly three asymptotic notations.

Big-O Notation (O-notation): represents the upper bound of the running time of an algorithm. Thus, it provides the worst-case complexity of an algorithm.

$$f(n) \leq c * g(n) \text{ for all } n \geq n0$$

Omega Notation (Ω-Notation): Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm.

$$f(n) \geq c*g(n) \text{ for all } n \geq n0$$

Theta Notation (Θ-Notation):

Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the **average-case**

complexity of an algorithm.

$$\mathbf{c1^* g(n) \leq f(n) \leq c2 * g(n)} \text{ for all } n \geq n0$$

7. Show an algorithm to subtract the product and sum of an integer with input 'n'=234.

Logic:

Input= n= 234;

Product of digits=2*3*4=24

Sum of digits=2+3+4=9

Result= Product-Sum=24-9=15

Algorithm subprodsum(n)

{

sum=0; prod=1;

while (n > 0)

{

sum = sum + (n % 10)

prod= prod*(n%10)

n=n/10

}

return (prod – sum);

}

- **Time Complexity**: **$O(\log_{10}(n))$**, where n is the input number. This is because the loop runs a number of times equal to the number of digits in n, which is proportional to $\log_{10}(n)$.
- **Space Complexity**: **O(1)**, since only a constant amount of space is used.

8.Show an algorithm to add two integers in array nums= [3,2,4]; target=6.

Algorithm sum(nums[], target)

{

//Input: nums=[3,2,4],

//Output: Add any two number to get target and return it's indices.

numslen= sizeof(nums)

for (int i = 0; i < numslen; i++)

{

for (int j = i + 1; j < numslen; j++)

{

if (nums[i] + nums[j] == target)

{

result[0]=i;//Store first index

result[1] = j; // Store the second index

return result;

}

The **time complexity** of this algorithm is **O(n²)**,

Total iterations=(n−1)+(n−2)+⋯+1

=((n^2)+n)/2

=n^2(consider higher degree of polynomial)

The space complexity of this algorithm is O(1) (constant space)

9.Show an algorithm to check any number is even or odd.

Algorithm evenodd(n)

{

//Input: any integer

//Output: check whether is number is even or odd and display it

if(num%2==0)

Display("Number is Even ")

Else

Display("Number is Odd ")

}

Time complexity : O(1) because all operations (modulus and conditional check) take constant time.

Space complexity: O(1), since it uses only a fixed amount of space.

10.List the Basic Asymptotic Efficiency Class and its Order.

Input	Classes of Function	Order represented
1	:Constant.	O (1)
log n..	Logarithmic	O (log n)
n log n	linearithmic	O (n log n)
n^2	Quadratic	O(n^2)
n^3	Cubic	O (n^3).
2^n	Exponential Form	O(2^n)

ORDER OF GROWTH

Measuring the performance of Algorithm in relation with Input Size 'n' is called Order of Growth.

Example 1: 4n2 + 3n + 100

After ignoring lower order terms, we get 4n2.

After ignoring constants, we get n2.

Hence **order of growth = O(n^2)**