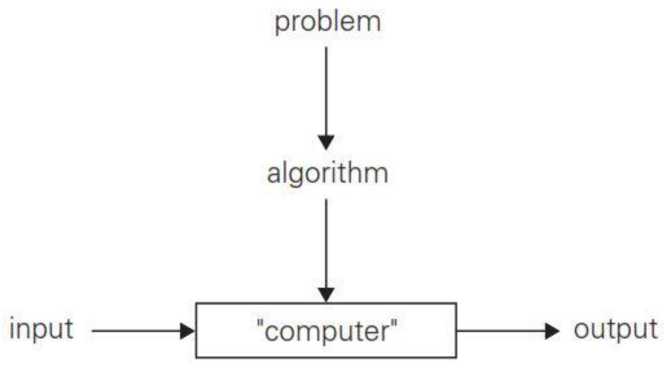
UNIT – I

PART - A

1. **Define notation of an algorithm.**



2. **Define Algorithm and list out its characteristics.**

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

3. **List out various steps in algorithmic problem solving.**

The fundamental steps are

- Understanding the problem
- Ascertain the capabilities of computational device
- Choose between exact and approximate problem solving
- Decide on appropriate data structures
- Algorithm design techniques
- Methods for specifying the algorithm
- Proving an algorithms correctness
- Analyzing an algorithm
- Coding an algorithm

4. **State the Euclid's algorithm for finding GCD of two given numbers.**

```
ALGORITHM Euclid (m, n)
//Computes gcd(m,n) by Euclid's algorithm
//Input   : Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while $n \neq 0$ do
        $r \leftarrow m \bmod n$
        $m \leftarrow n$
        $n \leftarrow r$
return m.
```

5. **Write an algorithm to find number of binary digits in the binary representation of a positive decimal integer? (APRIL/MAY 2015)**

```
Algorithm : Binary(n)
//input:     A positive decimal integer
//output:    The number of binary digits in n's binary representation

Count ← 1;
while n > 1 do
        Count  ← Count + 1
        n ← [n/2]
return Count;
```

6. **Write the recursive Fibonacci algorithm and its recursive relation? (NOV/DEC 2015)**

Algorithm

```
 int fib (int n)
{
if(n==0) return 0;
elseif (n==1) return 1;
else
return fib (n-1) + fib (n-2);
```

Recurrence relation

$T(n) = T(n-1) + T(n-2) +\theta(1)$

7. **What is Algorithm Design Technique? Nov/Dec 2005**

An algorithm design technique is a general approach to solve problems algorithmically, which is applicable to a variety of problems from different areas of computing.

Algorithm's Correctness

To prove that the algorithm yields a required result for every legitimate input in a finite amount of time.

8. **What is Efficiency of algorithm?**

Efficiency of an algorithm can be precisely defined and investigated with mathematical rigor. There are two kinds of algorithm efficiency.

I. Time Efficiency – Indicates how fast the algorithm runs

II. Space Efficiency – Indicates how much extra memory the algorithm needs.

9. **How to measure an algorithm's running time?**

Let C_{op} be the time of execution of an algorithm's basic iteration on a particular computer and let C (n) be the number of times this operation needs to be executed for this algorithm. Then we can estimate the running time T(n) of a program implementing this algorithm on that computer by the formula

$$T(n) \approx C_{op} C(n)$$

10. Find the basic operation and its order of growth for the following algorithm

```
algorithm sum(a,n)
{     s=0.0;
for i=1 to n do
s=s+a[i];
}
```

Basic Operation: Addition

Order of Growth: $\sum_{i=1}^{n} 1 = n - 1 + 1 = n$

11. Design an algorithm for finding the distance between the two closest elements in an array of numbers.

```
ALGORITHM  MinDistance($A[0..n-1]$)
    //Input: Array $A[0..n-1]$ of numbers
    //Output: Minimum distance between two of its elements
    $dmin \leftarrow \infty$
        for $i \leftarrow 0$ to $n-1$ do
            for $j \leftarrow 0$ to $n-1$ do
                if $i \neq j$ and $|A[i] - A[j]| < dmin$
                $dmin \leftarrow |A[i] - A[j]|$
    return $dmin$
```

12. Write algorithm for finding uniqueness of array element.

```
ALGORITHM  UniqueElements($A[0..n-1]$)
    //Determines whether all the elements in a given array are distinct
    //Input: An array $A[0..n-1]$
    //Output: Returns "true" if all the elements in $A$ are distinct
    //          and "false" otherwise
    for $i \leftarrow 0$ to $n-2$ do
        for $j \leftarrow i+1$ to $n-1$ do
            if $A[i] = A[j]$ return false
    return true
```

13. Define recurrence relation.

A **recurrence relation** is an **equation** that recursively defines a sequence, once one or more initial terms are given: each further term of the sequence is defined as a function of the preceding terms.

14. Define order of growth.

Order of growth in algorithm means how the time for computation increases when you increase the input size. It really matters when your input size is very large.

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

15. Differentiate big oh from Omega Notation.

1.	the ***worst*** case running time for an algorithm.	the ***best*** case running time for a given algorithm.
2.	the lowest valued function that will always have a higher value than the actual running of the algorithm.	the highest valued function that will always have a lower value than the actual running of the algorithm
3.	asymptotic upper bound	asymptotic lower bound

16. Say the following equalities are true or false and prove that $10n^2+5n+2 \in O(n^2)$.

$$\lim_{n\to\infty} \frac{10n^2 + 5n + 2}{n^2} = \lim_{n\to\infty} \frac{10n^2}{n^2} + \frac{5n}{n^2} + \frac{2}{n^2}$$

$$= \lim_{n\to\infty} 10 + \frac{5}{n} + \frac{2}{n^2} = 10 > 0$$ case 2;

False.

17. Find the complexity of the recurrence relation T(n)= = 8T(n/2) + n 2 with T(1) = 1

$$\begin{aligned} T(n) &= n^2 + 8T(n/2) \\ &= n^2 + 8(8T(\frac{n}{2^2}) + (\frac{n}{2})^2) \\ &= n^2 + 8^2T(\frac{n}{2^2}) + 8(\frac{n^2}{4})) \\ &= n^2 + 2n^2 + 8^2T(\frac{n}{2^2}) \end{aligned}$$

$$
\begin{aligned}
&= n^2 + 2n^2 + 8^2(8T(\frac{n}{2^3}) + (\frac{n}{2^2})^2) \\
&= n^2 + 2n^2 + 8^3T(\frac{n}{2^3}) + 8^2(\frac{n^2}{4^2})) \\
&= n^2 + 2n^2 + 2^2n^2 + 8^3T(\frac{n}{2^3}) \\
&= \dots \\
&= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \dots
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \dots + 2^{\log n - 1}n^2 + 8^{\log n} \\
&= \sum_{k=0}^{\log n - 1} 2^k n^2 + 8^{\log n} \\
&= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}
\end{aligned}
$$

Now $\sum_{k=0}^{\log n-1} 2^k$ is a geometric sum so we have $\sum_{k=0}^{\log n-1} 2^k = \Theta(2^{\log n-1}) = \Theta(n)$
$(2^3)^{\log n} = (2^{\log n})^3 = n^3$

$$
\begin{aligned}
T(n) &= n^2 \cdot \Theta(n) + n^3 \\
&= \Theta(n^3)
\end{aligned}
$$

18. Give the general plan for analyzing time efficiency of non-recursive algorithm.

- ✓ Decide on a parameter (or parameters) indicating an input’s size.
- ✓ Identify the algorithm’s basic operation. (As a rule, it is located in the innermost loop.)
- ✓ Check whether the number of times the basic operation is executed depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately.
- ✓ Set up a sum expressing the number of times the algorithm’s basic operation is executed.
- ✓ Using standard formulas and rules of sum manipulation, either find a closed form formula for the count or, at the very least, establish its order of growth.

19. **Solve the recurrence relation X(n)=X(n-1)+5 for n>1, X(1)=0**

$x(n) = x(n-1) + 5 \quad \text{for } n > 1, \quad x(1) = 0$

$$\begin{aligned} x(n) &= x(n-1) + 5 \\ &= [x(n-2) + 5] + 5 = x(n-2) + 5 \cdot 2 \\ &= [x(n-3) + 5] + 5 \cdot 2 = x(n-3) + 5 \cdot 3 \\ &= \ldots \\ &= x(n-i) + 5 \cdot i \\ &= \ldots \\ &= x(1) + 5 \cdot (n-1) = 5(n-1). \end{aligned}$$

20. **Solve the recurrence relation X(n)=X(n-1)+n for n>0, X(0)=0**

$x(n) = x(n-1) + n \quad \text{for } n > 0, \quad x(0) = 0$

$$\begin{aligned} x(n) &= x(n-1) + n \\ &= [x(n-2) + (n-1)] + n = x(n-2) + (n-1) + n \\ &= [x(n-3) + (n-2)] + (n-1) + n = x(n-3) + (n-2) + (n-1) + n \\ &= \ldots \\ &= x(n-i) + (n-i+1) + (n-i+2) + \ldots + n \\ &= \ldots \\ &= x(0) + 1 + 2 + \ldots + n = \frac{n(n+1)}{2}. \end{aligned}$$

21. **Solve the recurrence relation X(n)=3X(n-1) for n>1 X(1)=4**

$x(n) = 3x(n-1) \quad \text{for } n > 1, \quad x(1) = 4$

$$\begin{aligned} x(n) &= 3x(n-1) \\ &= 3[3x(n-2)] = 3^2 x(n-2) \\ &= 3^2[3x(n-3)] = 3^3 x(n-3) \\ &= \ldots \\ &= 3^i x(n-i) \\ &= \ldots \\ &= 3^{n-1} x(1) = 4 \cdot 3^{n-1}. \end{aligned}$$

22. **Solve the recurrence relation $T(n)=8T(n/2)+n^2$ using Master's Theorem.**

Mapping with general EQN. a = 8, b = 2, d = 2

$b^d = 2^2 = 4$ So, $a>b^d$ then $\Theta(n^{\log_b a}) = \Theta(n^{\log_2 8}) = \Theta(n^3)$

23. **Find the complexity of the recurrence relation**

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

Mapping with general EQN.

a = 2, b = 2, d = 2

$b^d = 2^2 = 4$ So, $a < b^d$ then $\Theta(n^{\log_b a}) = \Theta(n^d) = \Theta(n^2)$

24. **Solve the recurrence relation x(n)=x(n/3)+1**

$x(n) = x(n/3) + 1 \quad \text{for } n > 1, \quad x(1) = 1 \quad (\text{solve for } n = 3^k)$

$$\begin{aligned}
x(3^k) &= x(3^{k-1}) + 1 \\
&= [x(3^{k-2}) + 1] + 1 = x(3^{k-2}) + 2 \\
&= [x(3^{k-3}) + 1] + 2 = x(3^{k-3}) + 3 \\
&= \dots \\
&= x(3^{k-i}) + i \\
&= \dots \\
&= x(3^{k-k}) + k = x(1) + k = 1 + \log_3 n.
\end{aligned}$$

25. Give the general plan for analyzing time efficiency of recursive algorithm.

- ✓ Decide on a parameter (or parameters) indicating an input's size.
- ✓ Identify the algorithm's basic operation. (As a rule, it is located in the innermost loop.)
- ✓ Check whether the number of times the basic operation is executed can vary on different inputs of the same size; if it can, the worst-case, average-case, and best-case efficiencies must be investigated separately.
- ✓ Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed.
- ✓ Solve the recurrence or, at least, ascertain the order of growth of its solution.

26. Consider the following recursive algorithm and setup a recurrence relation

```
Algorithm Q(n)
//input: a positive integer n
if  n=1 then
return 1;
else
return Q(n-1)+2n-1
```

$$Q(n) = Q(n-1) + 2n - 1 \quad \text{for } n > 1, \quad Q(1) = 1.$$

PART – B

1. Explain fundamentals of problem solving in details.
2. Elaborate Asymptotic Notation with examples. Write down the properties of Asymptotic notations
3. Solve the puzzle tower of Hanoi and give its complexity
4. Write a recursive algorithm for sum of N numbers and compute its complexity.
5. Write the recursive algorithm for Fibonacci and compute the time Efficiency
6. Give the general plan for analyzing non recursive algorithm and write down the algorithm to find largest element in an array and compute its efficiency