CMR
INSTITUTE OF
TECHNOLOGY

USN ☐☐☐☐☐☐☐☐☐☐

First Internal Test

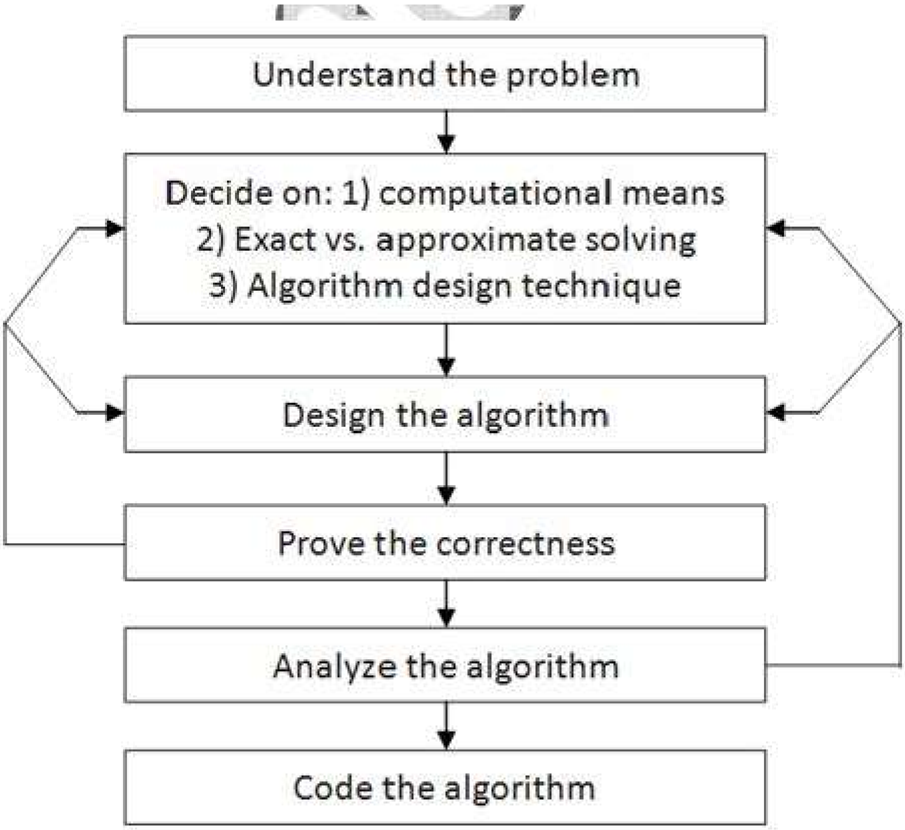
Sub:	**DESIGN AND ANALYSIS OF ALGORITHMS**							Code:	17CS43
Date:	06/03/2019	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	**ISE**
Answer Any **FIVE FULL** Questions									

	Marks	OBE CO	OBE RBT
1 (a) What is Algorithm? What are the characteristics of algorithm? Explain the algorithm design process with flow chart?	[6]	CO1	L2
(b)	[4]	CO2	L4

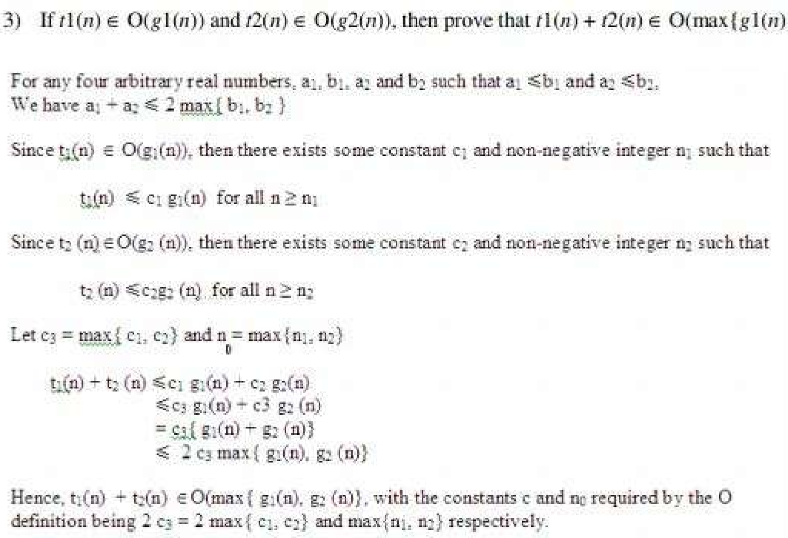
An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks. An algorithm is an efficient method that can be expressed within finite amount of time and space.

A sequence of steps one typically goes through in designing and analyzing an algorithm

(b)



$T1(n)=O(g1(n))$ And $T2(n)=O(g2(n))$ Prove that $T1(n)+T2(n)=O(Max(g1(n),g2(n)))$



3) If $t1(n) \in O(g1(n))$ and $t2(n) \in O(g2(n))$, then prove that $t1(n) + t2(n) \in O(\max\{g1(n), g2(n)\})$

For any four arbitrary real numbers, a_1, b_1, a_2 and b_2 such that $a_1 \leqslant b_1$ and $a_2 \leqslant b_2$.
We have $a_1 + a_2 \leqslant 2 \max\{ b_1, b_2 \}$

Since $t_1(n) \in O(g_1(n))$, then there exists some constant c_1 and non-negative integer n_1 such that

$$t_1(n) \leqslant c_1 g_1(n) \text{ for all } n \geq n_1$$

Since $t_2(n) \in O(g_2(n))$, then there exists some constant c_2 and non-negative integer n_2 such that

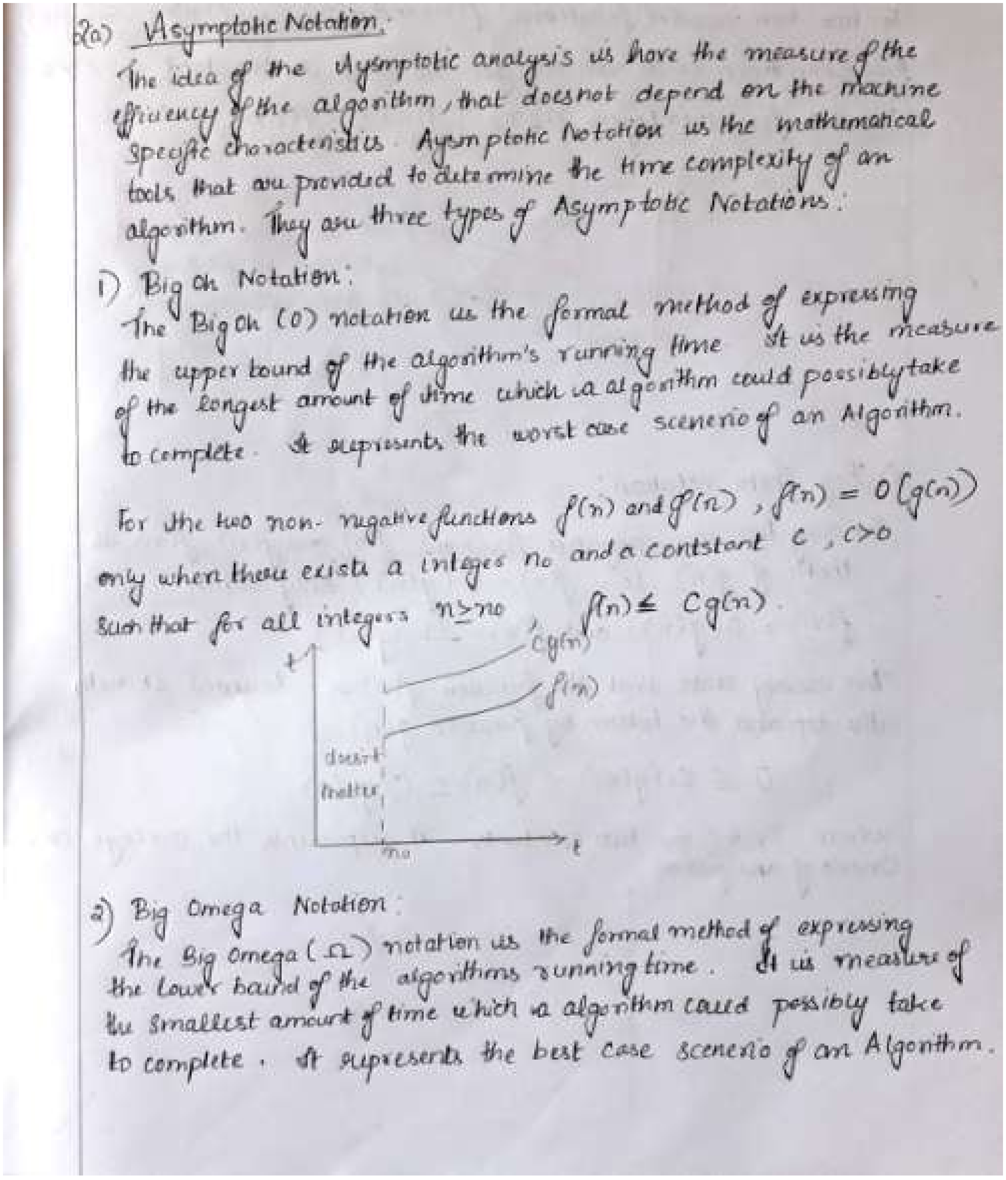
$$t_2(n) \leqslant c_2 g_2(n) \text{ for all } n \geq n_2$$

Let $c_3 = \max\{ c_1, c_2\}$ and $n_0 = \max\{n_1, n_2\}$

$$\begin{aligned} t_1(n) + t_2(n) &\leqslant c_1 g_1(n) + c_2 g_2(n) \\ &\leqslant c_3 g_1(n) + c3\ g_2(n) \\ &= c_3\{ g_1(n) + g_2(n)\} \\ &\leqslant 2 c_3 \max\{ g_1(n), g_2(n)\} \end{aligned}$$

Hence, $t_1(n) + t_2(n) \in O(\max\{ g_1(n), g_2(n)\}$, with the constants c and n_0 required by the O definition being $2 c_3 = 2 \max\{ c_1, c_2\}$ and $\max\{n_1, n_2\}$ respectively.

2 (a) Explain the three asymptotic notations, Big Oh, Big Omega and Big Theta notations with example	[10]	CO1	L3



2(a) Asymptotic Notation:

The idea of the Aysmptotic analysis is have the measure of the efficiency of the algorithm, that doesnot depend on the machine specific characteristics. Aysmptotic Notation is the mathematical tools that are provided to determine the time complexity of an algorithm. They are three types of Asymptotic Notations:

1) Big Oh Notation:

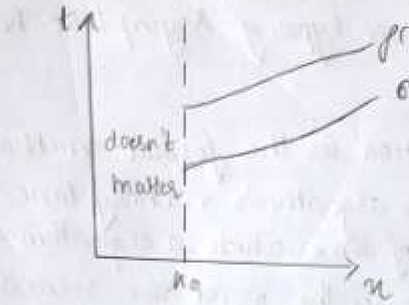
The Big Oh (O) notation is the formal method of expressing the upper bound of the algorithm's running time. It is the measure of the longest amount of time which a algorithm could possibly take to complete. It represents the worst case scenerio of an Algorithm.

For the two non-negative functions $f(n)$ and $g(n)$, $f(n) = O(g(n))$ only when there exists a integer n_0 and a contstant c, $c>0$ such that for all integers $n \geq n_0$, $f(n) \leq cg(n)$.

2) Big Omega Notation:

The Big Omega (Ω) notation is the formal method of expressing the lower bound of the algorithms running time. It is measure of the smallest amount of time which a algorithm could possibly take to complete. It represents the best case scenerio of an Algorithm.

For two non-negative functions $f(n)$ and $g(n)$, $f(n) = \Omega(g(n))$ only when there exists an integer n_0 and a constant c, $c>0$ suchthat for all integers $n \geq n_0$, $f(n) \geq cg(n)$.



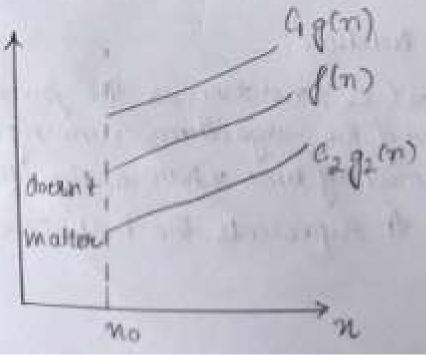
3) Big Theta notation;

For any two non-negative functions $f(n)$ and $g(n)$, $f(n)$ is theta of $g(n)$ ie, $f(n) = \Theta(g(n))$ only when $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

This clearly states that the function $f(n)$ is bounded at both the top and the bottom by function $g(n)$.

$$0 \leq c_1(g(n)) \leq f(n) \leq c_2 g_2(n)$$

where c_1 & c_2 are two constants. It represents the average ca Senerio of analgorithm



3 (a) Discuss the 'Tower of Hanoi' problem and write a recursive algorithm to solve it. Mathematically analyze the Tower of Hanoi problem and find its complexity. [10] CO2 L3

3(a) Tower of Honoi

It is a very famous Mathematical puzzle, wherein there will be 'n' no. of disks which vary in size and 3 pegs. The problem is that 'n' no. of disks should be transferred from 1st peg to 3rd peg by making use of the 2nd peg, such that

1> No two disks can be moved together, ie, can move only one disk at a time.

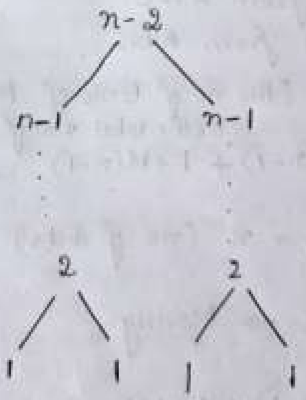
2> The smaller disk should always be above the larger disk

Recursive Algorithm:

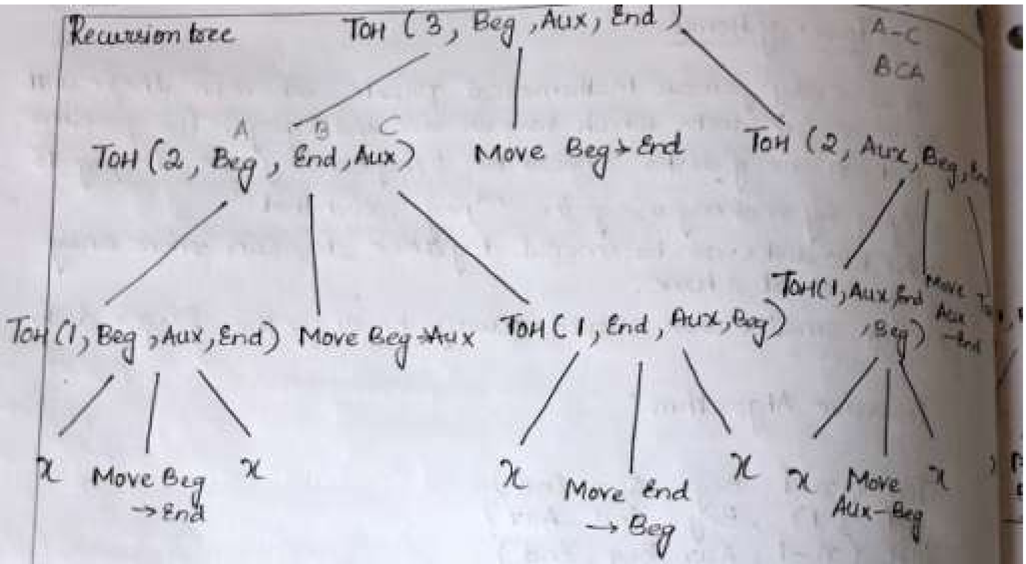
```
TOH (n-1, Beg, Aux, End)
TOH  (1) , Beg, End, Aux)
TOH (n-1, Aux, Beg, End)
```

The above functions are recursively called, till the disks reach 1.

Recursive tree



Recursion tree



Mathematical Analysis of TOH

Move (n-1) disks from A to B
Move 1 disk from A to C
Move (n-1) disks from B to C.

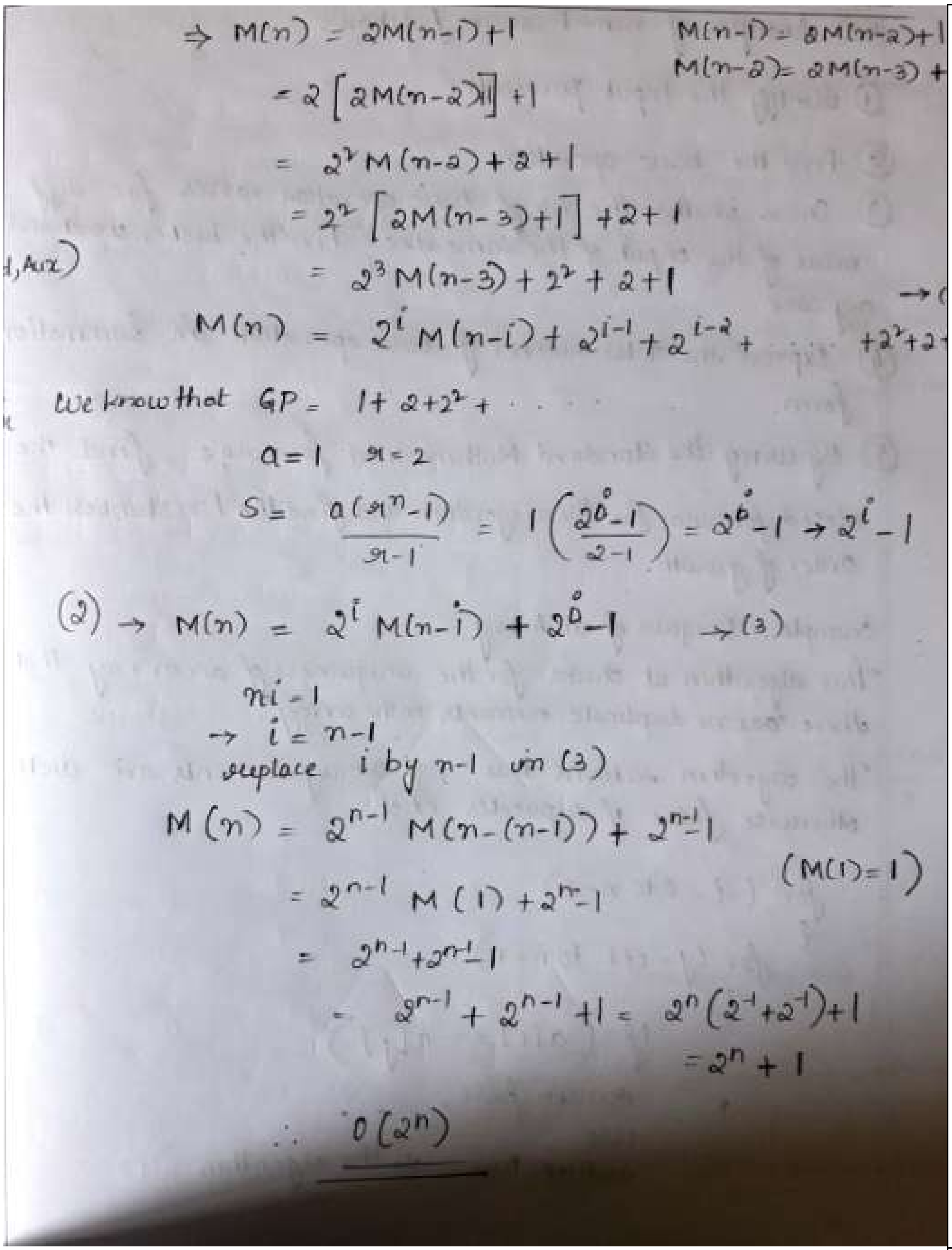
We have M(n) → (the no. of times of transferring the disks) or the total no. of Moves in moving n disks.

$$M(n) = M(n-1) + 1 + M(n-1). \rightarrow (1)$$

① Input parameter → 'n' (no. of disks)

② Basic operation → Moving.

③ $M(n) = M(n-1) + 1 + M(n-1).$ from (1)



$\Rightarrow M(n) = 2M(n-1)+1$ $\qquad M(n-1) = 2M(n-2)+1$

$M(n-2) = 2M(n-3) +$

$= 2[2M(n-2)+1]+1$

$= 2^2 M(n-2)+2+1$

$= 2^2[2M(n-3)+1]+2+1$

d,Aux)

$= 2^3 M(n-3)+2^2+2+1$ $\rightarrow$ (

$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2^2 + 2 +$

We know that GP = $1 + 2 + 2^2 + \cdots$

$a = 1 \quad r = 2$

$S = \frac{a(r^n - 1)}{r-1} = 1\left(\frac{2^i - 1}{2-1}\right) = 2^i - 1 \rightarrow 2^i - 1$

(2) $\rightarrow M(n) = 2^i M(n-i) + 2^i - 1$ $\rightarrow$ (3)

$ni = 1$

$\rightarrow i = n-1$

replace i by n-1 in (3)

$M(n) = 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1$

$(M(1) = 1)$

$= 2^{n-1} M(1) + 2^{n-1} - 1$

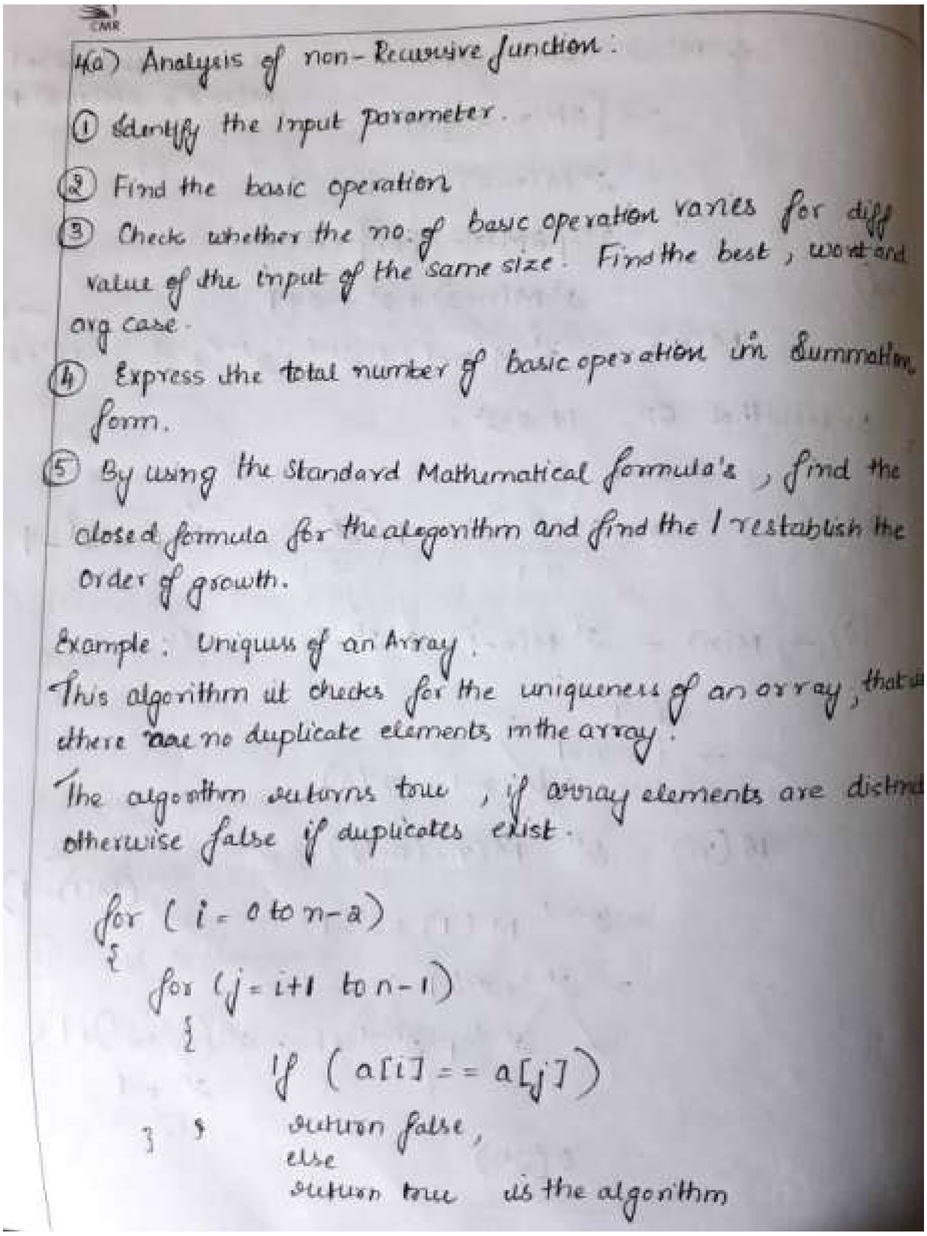
$= 2^{n-1} + 2^{n-1} - 1$

$= 2^{n-1} + 2^{n-1} + 1 = 2^n(2^{-1} + 2^{-1}) + 1$

$= 2^n + 1$

$\therefore O(2^n)$

4 (a) Explain the general plan for mathematical analysis of non-recursive function with example?	[10]	C01	L2



4(a) Analysis of non-Recursive function :

① Identify the input parameter.

② Find the basic operation

③ Check whether the no. of basic operation varies for diff value of the input of the same size. Find the best, worst and avg case.

④ Express the total number of basic operation in Summation form.

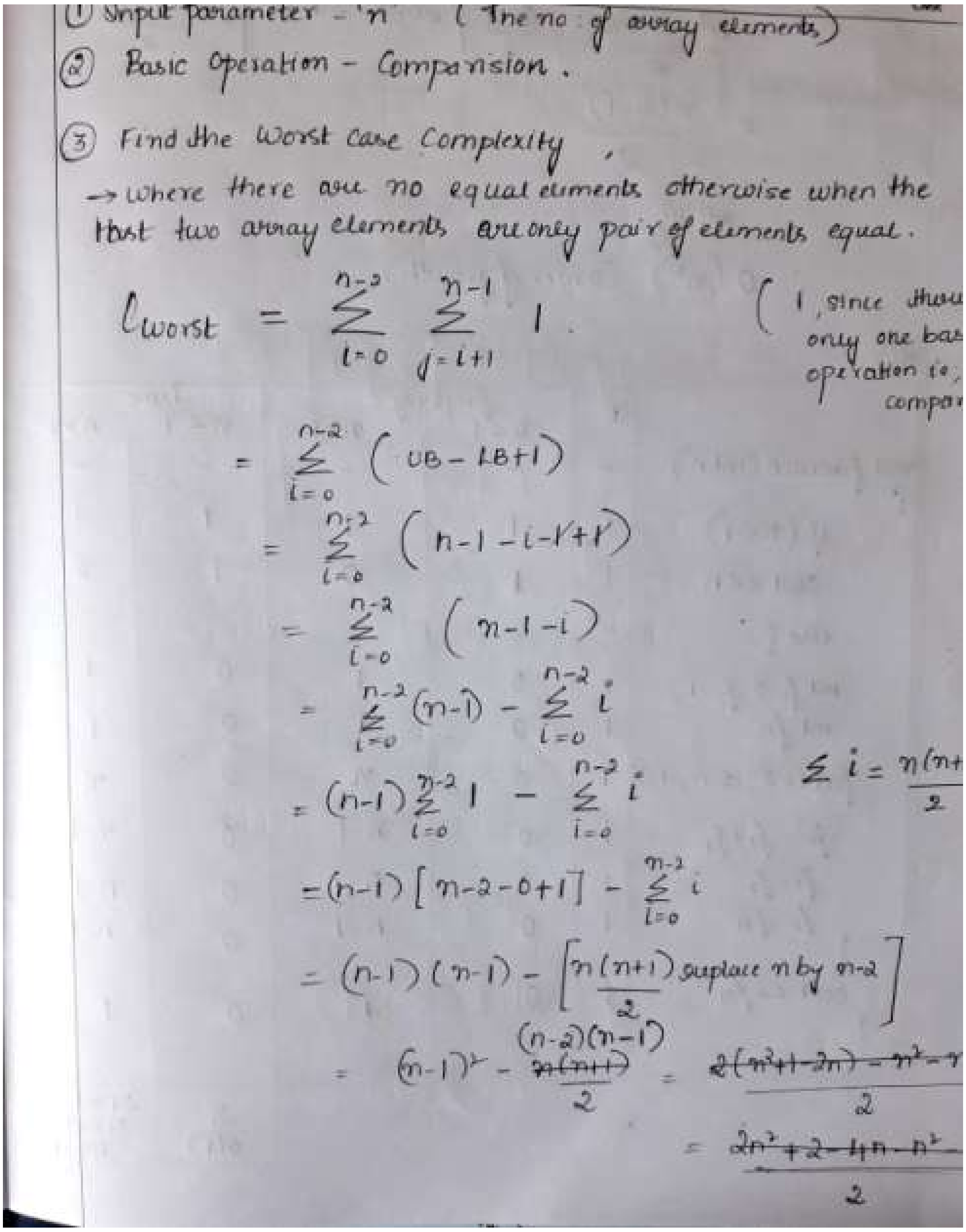
⑤ By using the Standard Mathematical formula's, find the closed formula for the alogorithm and find the / establish the order of growth.

Example : Uniquess of an Array.

This algorithm it checks for the uniqueness of an array, that there are no duplicate elements in the array.

The algorithm returns true, if array elements are dist otherwise false if duplicates exist.

```
for ( i = 0 to n-2)
{
    for (j = i+1 to n-1)
    {
            if ( a[i] == a[j] )
    }  }        return false,
                else
                return true    is the algorithm
```



① Input parameter - 'n' (The no of array elements)

② Basic Operation - Comparision.

③ Find the Worst case Complexity.

→ Where there are no equal elements otherwise when the last two array elements are only pair of elements equal.

$$C_{worst} = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1.$$

(1, since thou
only one bas
operation i.e,
compar

$$= \sum_{i=0}^{n-2} (UB - LB + 1)$$

$$= \sum_{i=0}^{n-2} (n-1-i-1+1)$$

$$= \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= (n-1)\sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

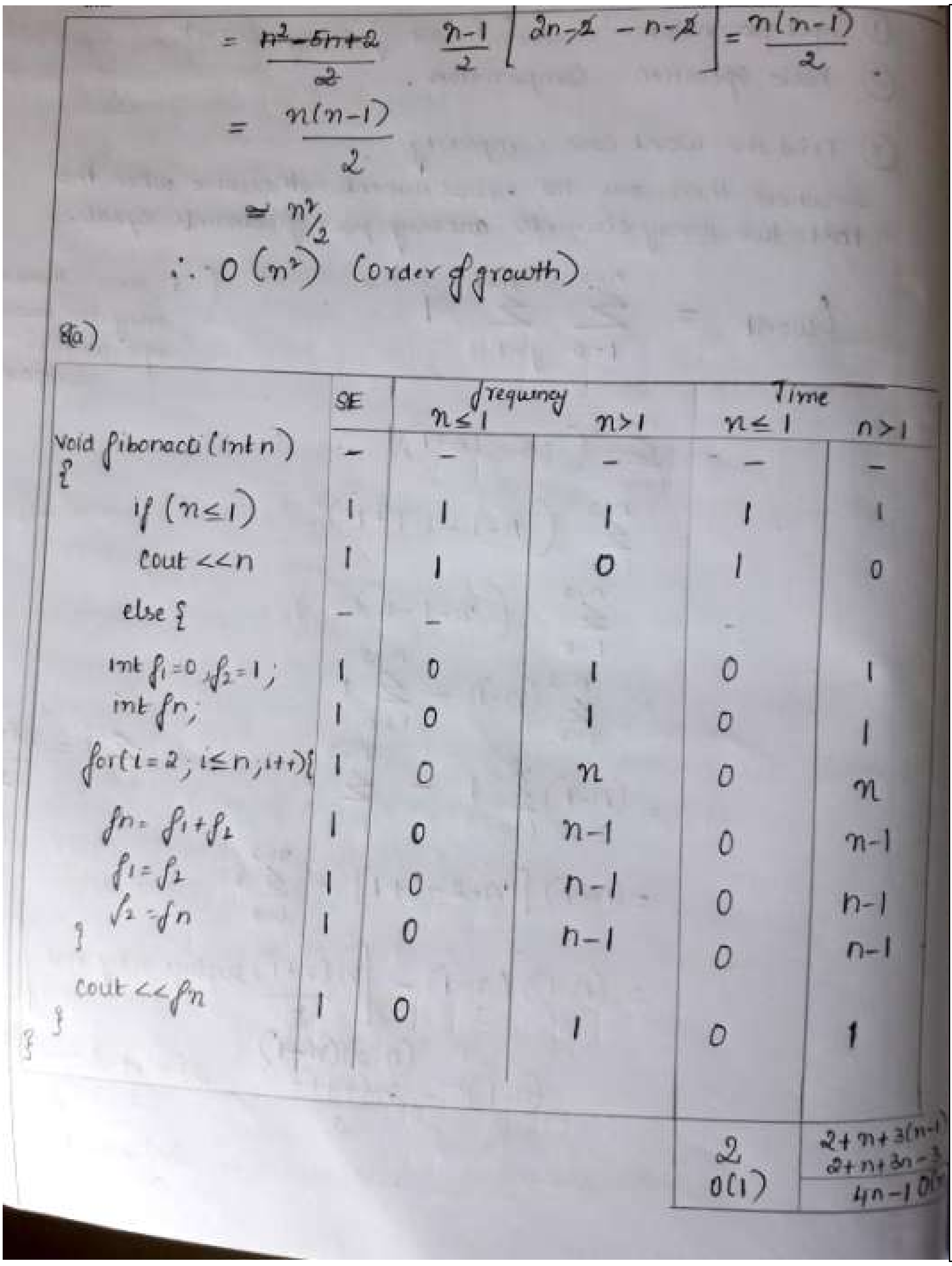
$$\sum i = \frac{n(n+}{2}$$

$$= (n-1)[n-2-0+1] - \sum_{i=0}^{n-2} i$$

$$= (n-1)(n-1) - \left[\frac{n(n+1)}{2} \text{ replace } n \text{ by } n-2\right]$$

$$= (n-1)^2 - \frac{\overset{(n-2)(n-1)}{\cancel{n(n+1)}}}{2} = \frac{\cancel{2(n^2+1-2n) - n^2 - n}}{2}$$

$$= \frac{\cancel{2n^2+2-4n-n^2-}}{2}$$



$$= \frac{\cancel{n^2-5n+2}}{2} \quad \frac{n-1}{2}\left[2n-\cancel{2} - n-\cancel{2}\right] = \frac{n(n-1)}{2}$$

$$= \frac{n(n-1)}{2}$$

$$\simeq n^2/2$$

$\therefore O(n^2)$ (order of growth)

8(a)

	SE	Frequency $n \le 1$	$n > 1$	Time $n \le 1$	$n > 1$
void fibonacci(int n)	–	–	–	–	–
{					
if (n≤1)	1	1	1	1	1
cout<<n	1	1	0	1	0
else {	–	–		–	
int f1=0, f2=1;	1	0	1	0	1
int fn;	1	0	1	0	1
for(i=2; i≤n; i++){	1	0	n	0	n
fn= f1+f2	1	0	n-1	0	n-1
f1=f2	1	0	n-1	0	n-1
f2=fn	1	0	n-1	0	n-1
}					
cout<<fn	1	0	1	0	1
} }					
				2 O(1)	2+n+3(n-1 2+n+3n-3 4n-1 O

		Marks	CO	Level
5 (a)	Explain Masters theorem. Prove that i) T (n) = 16T (n/4) + n= θ(n^2) ii) T (n) = 3T (n/2) + n^2 = θ(n^2) iii) T (n) = 3T (n/3) + √ n= θ(n) iv) T (n) = 4T (n/2) + n^2= θ(n^{2}logn)	[2+4]	CO2	L3
(b)	Write the algorithm for sequential search, obtain the time complexity of this algorithm for Successful and unsuccessful search in the worst case and best case.	[4]	CO3	L2

6 (a) Prove if following equalities are correct or incorrect [6] CO2 L4

i) $6n^2-5=\theta(n^2)$ c1g(n)<f(n)<c2g(n)
ii) $2n^2 + 8=O(n)$fail
iii) $120n + 5 =\Omega(n^{3)\text{ fail}}$

(b) Explain the concept of divide and conquer strategy along with its general recurrence equation. [4] CO4 L3

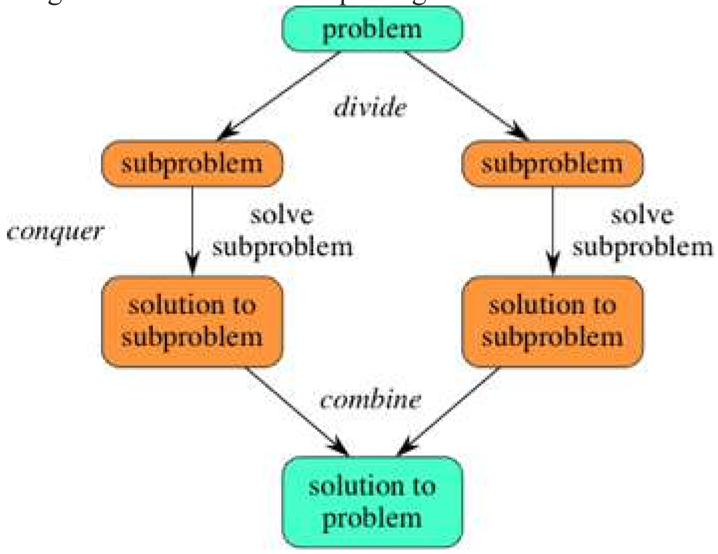
This paradigm, **divide-and-conquer**, breaks a problem into subproblems that are similar to the original problem, recursively solves the subproblems, and finally combines the solutions to the subproblems to solve the original problem. Because divide-and-conquer solves subproblems recursively, each subproblem must be smaller than the original problem, and there must be a base case for subproblems. You should think of a divide-and-conquer algorithm as having three parts:

Divide the problem into a number of subproblems that are smaller instances of the same problem.

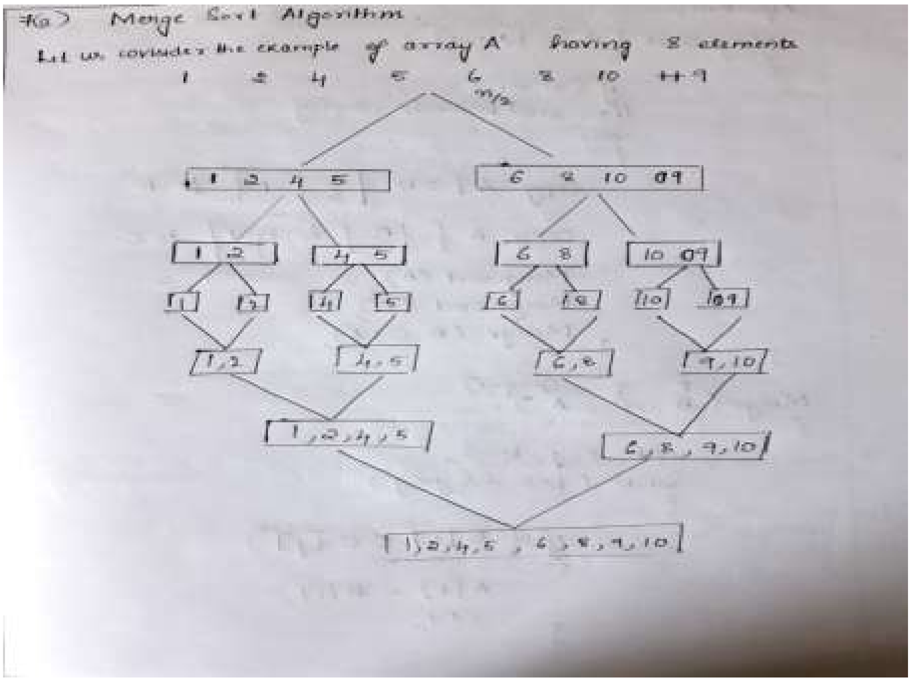
Conquer the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.

Combine the solutions to the subproblems into the solution for the original problem.

You can easily remember the steps of a divide-and-conquer algorithm as *divide, conquer, combine*. Here's how to view one step, assuming that each divide step creates two subproblems (though some divide-and-conquer algorithms create more than two):



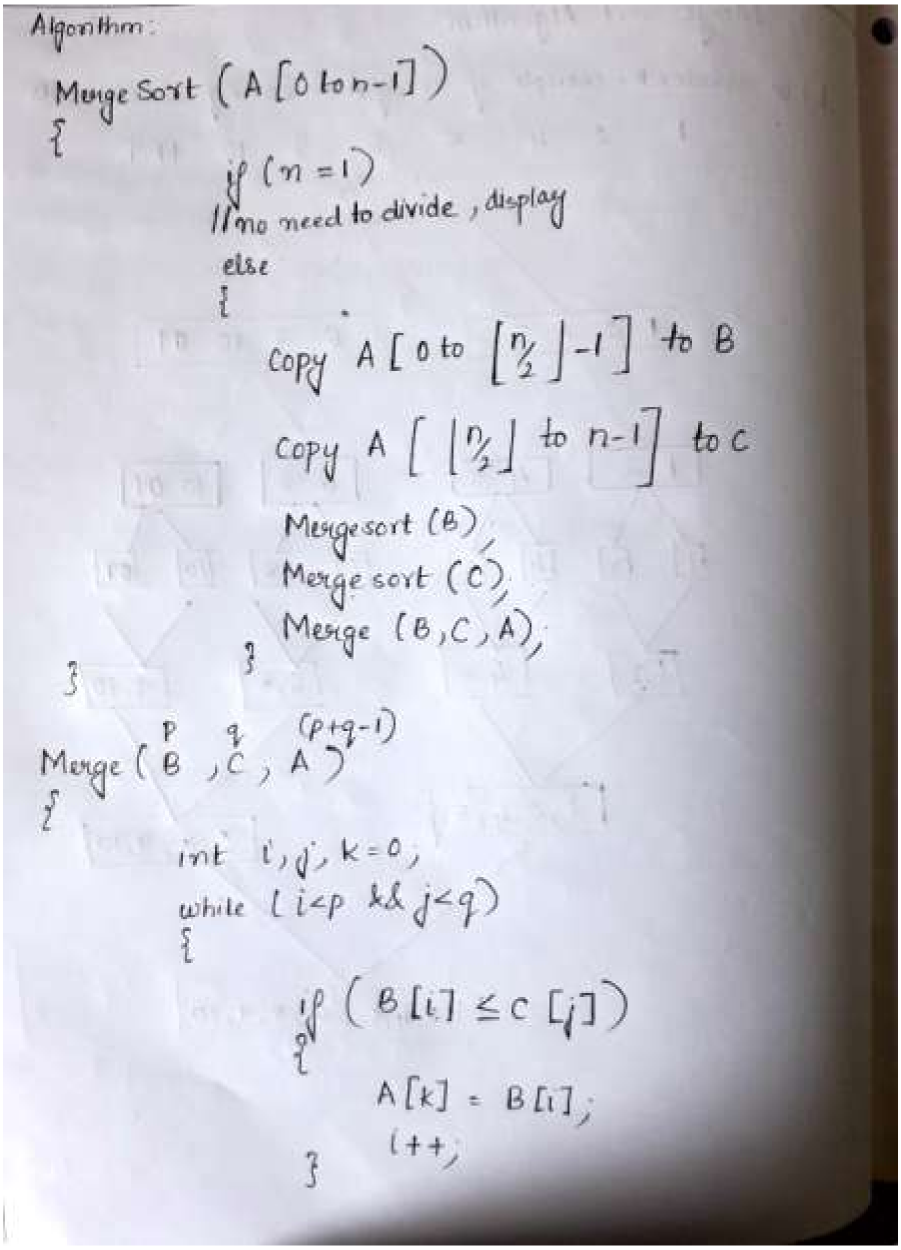
7 (a) Write the merge sort algorithm and explain with example. Also give the recurrence relation of merge sort algorithm and find its time complexity. [10] CO4 L3



7(a) Merge Sort Algorithm

Let us consider the example of array A having 8 elements

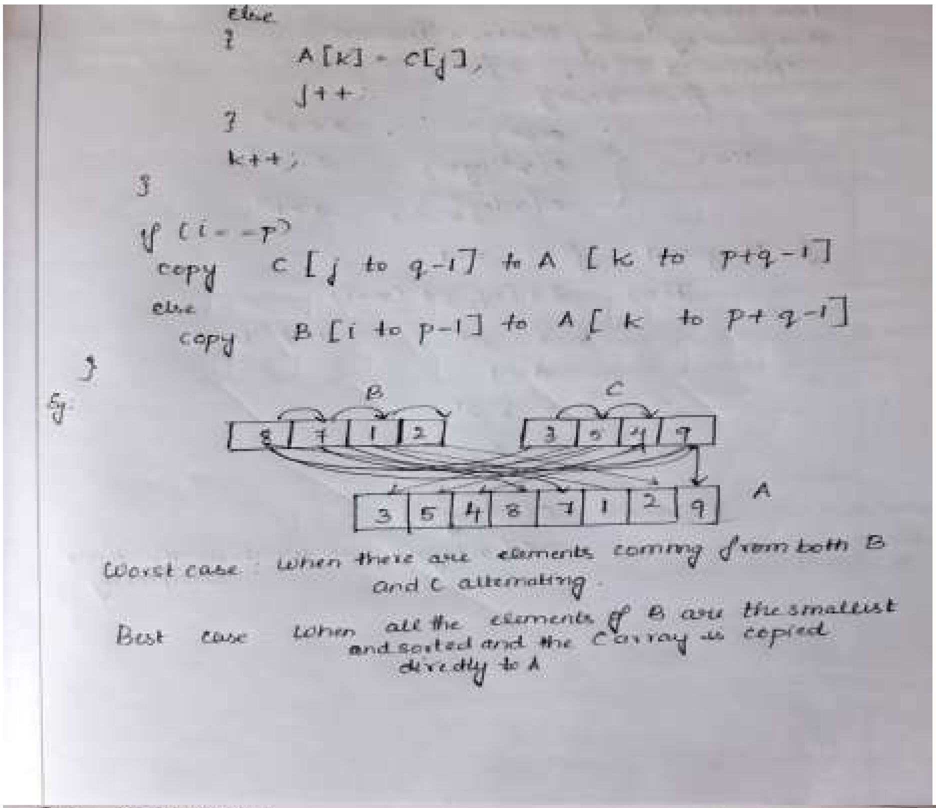
1 2 4 5 6 8 10 ~~11~~9



Algorithm:

```
Merge Sort ( A [0 to n-1] )
{
        if (n = 1)
        //no need to divide, display
        else
        {
            copy A [0 to $\lceil n/2 \rceil$ -1] to B
            copy A [ $\lfloor n/2 \rfloor$ to n-1] to C
            Mergesort (B);
            Merge sort (C);
            Merge (B, C, A);
        }
}

Merge ( $\overset{p}{B}$ , $\overset{q}{C}$ , $\overset{(p+q-1)}{A}$ )
{
      int i, j, k=0;
      while (i<p && j<q)
      {
            if (B[i] $\leq$ C [j])
            {
                  A[k] = B[i];
                  i++;
            }
```

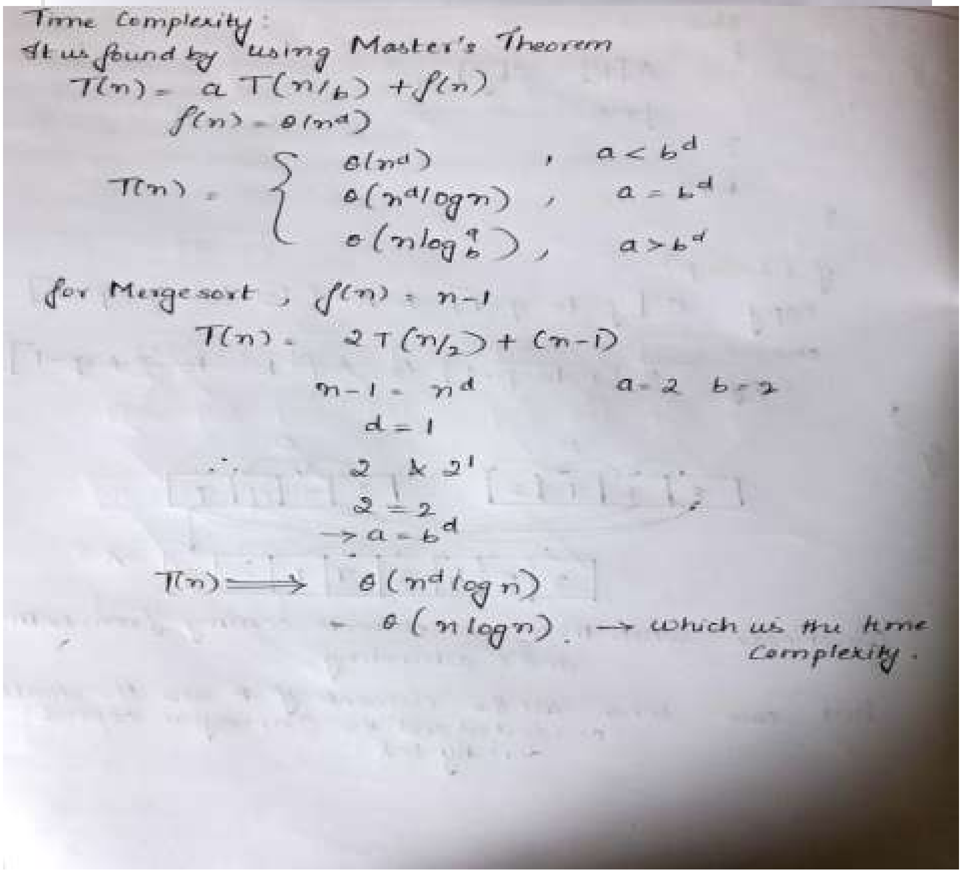


```
        else
        {
            A[k] = C[j];
            j++;
        }
        k++;
    }
    if (i == p)
     copy   C[j to q-1] to A [k to p+q-1]
    else
     copy   B[i to p-1] to A[k to p+q-1]
}
```

Eg.

Worst case : When there are elements comming from both B and C alternating.

Best case   When all the elements of B are the smallest and sorted and the C array is copied directly to A



Time Complexity:

It is found by using Master's Theorem

$T(n) = a\,T(n/b) + f(n)$

$f(n) = \theta(n^d)$

$$T(n) = \begin{cases} \theta(n^d) & , \ a < b^d \\ \theta(n^d \log n) & , \ a = b^d \\ \theta(n \log_b^a) & , \ a > b^d \end{cases}$$

for Mergesort , $f(n) = n-1$

$T(n) = 2T(n/2) + (n-1)$

$n - 1 = n^d \qquad a = 2 \quad b = 2$

$d = 1$

$\therefore \ 2 \ k \ 2^1$

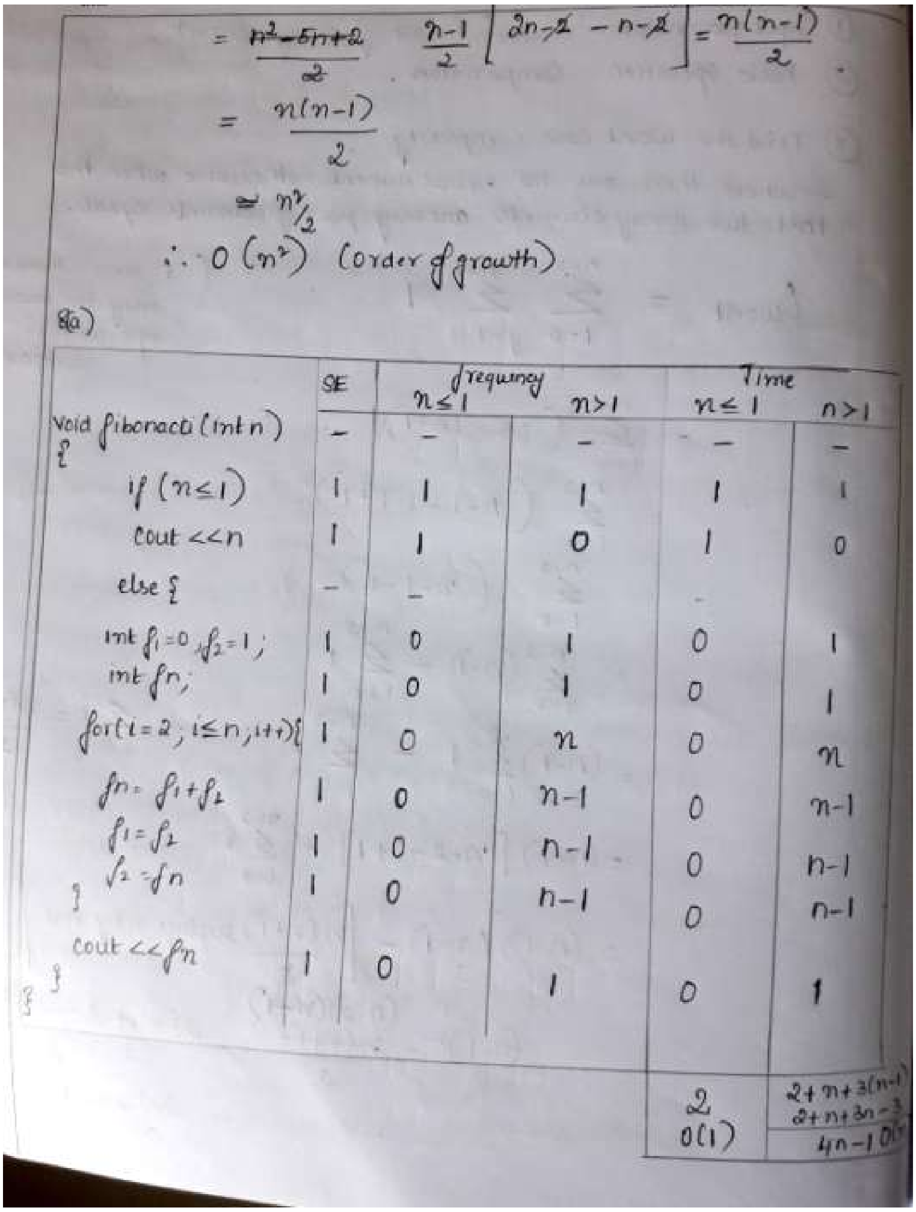
$2 = 2$

$\rightarrow a = b^d$

$T(n) \Longrightarrow \theta(n^d \log n)$

$= \theta(n \log n)$. → Which is the time Complexity.

8 (a) How can you measure time complexity of algorithm for Fibonacci series using tabular method? Write algorithm and explain?	[5]	CO1	L3



$= \frac{n^2-5n+2}{2}$ $\quad \frac{n-1}{2}\left[2n-\cancel{2} - n-\cancel{2}\right] = \frac{n(n-1)}{2}$

$= \frac{n(n-1)}{2}$

$\simeq n^2/2$

$\therefore O(n^2)$ (order of growth)

8(a)

	SE	Frequency $n\le 1$	Frequency $n>1$	Time $n\le 1$	Time $n>1$
void fibonacci (int n)	–	–	–	–	–
{					
if (n≤1)	1	1	1	1	1
cout<<n	1	1	0	1	0
else {	–	–		–	
int f1=0, f2=1;	1	0	1	0	1
int fn;	1	0	1	0	1
for(i=2; i≤n; i++){	1	0	n	0	n
fn= f1+f2	1	0	n-1	0	n-1
f1=f2	1	0	n-1	0	n-1
f2=fn	1	0	n-1	0	n-1
}					
cout<<fn	1	0	1	0	1
}					
}					
				2 O(1)	2+n+3(n-1) 2+n+3n-3 4n-1 O(

(b) Explain important problem types	[5]	CO1	L2

8(b) Important Problem Types.

1> Sorting:

Sorting problem, refers to arrange the elements in a array in a non-decreasing order. Also there are may Sorting Algorithms, but there is no best solution, as few algorithms may be simpl but takes longtime, other's may be faster but would be complex. And also all algorithms cannot be used to sort large storage files on disk.

2> Searching:

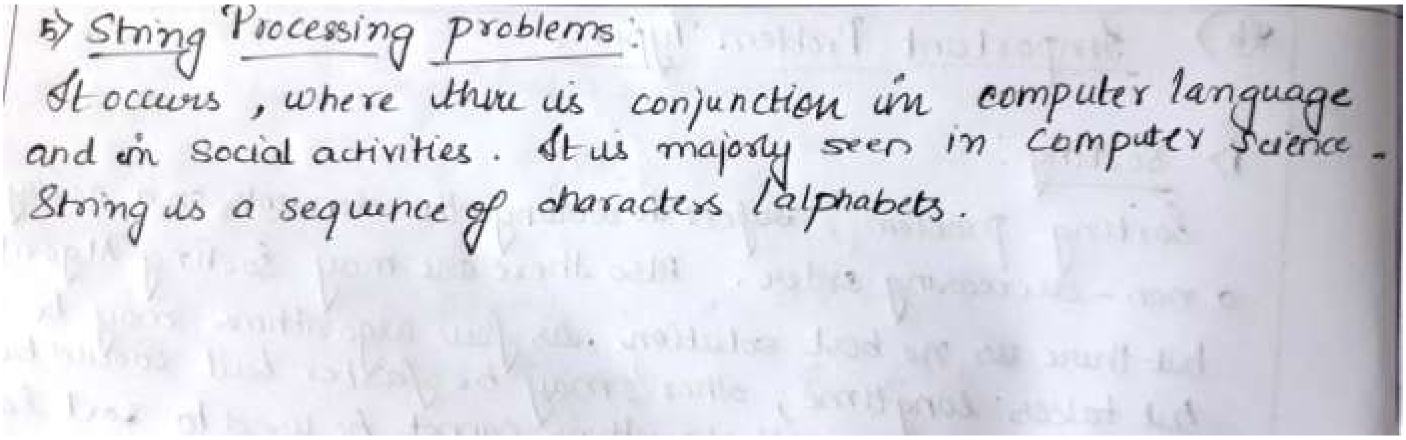
Searching Problem refers to of finding a particular elem (Searchkey) from a group of elements. Also the worst case would be, that the key is at the end of the array or at n at all there. There are many Searching techniques such as ~~Selection~~ Linear search and the most important Binary Search. But there is no algorithm that could take the least time.

3> Graph problems:

It is one of the oldest and interesting topic in Algorithmics, wherein several points are considered as nodes/ vertices and th are edges b/w them. Graphs can be used for modelling vario applications such as communication, transportation. Also is current active research topic.

4> Combinatorial problems.

This is one of the major problem, as there is no solutio for this in a ~~am~~ for a small amount of time. The problem arises because, the Combinatorial objects grow very fa in size with increasing size of the problem, to an extent that cannot be imagined of. The Solution cannot be found limited amount of time.



5) String Processing problems:

It occurs, where there is conjunction in computer language and in social activities. It is majorly seen in Computer Science. String is a sequence of characters /alphabets.