USN | | | | | | | | | | |

Internal Assessment Test 1 – December 2023

Sub:	Analysis & Design of Algorithms Answer Key					Sub Code:	BCS401	Branch:	AIDS & CSE(AIDS)
Date:	**5/6/2024**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**IV -A, B & C**		**OBE**

		Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
1	a	What is an algorithm? What are the properties of an algorithm? **Answer:** **Difinition:** Step by step procedure for solving a computational problem.A finite set of statements that guarantee an optimal solution in finite interval of time. **(1 Mark)** **Properties:** Input- should have 0 or more input values. Output – Must generate some result. Function must do something, maybe just return void. Definiteness – Every statement should be clear and unambiguous. Finiteness – Should have limited steps. Must terminate at some point. Example: web server has to stop service at some point. Can't just keep providing service endlessly. Effectiveness – It should reach a solution.	4	1	L1
	b	Define asymptotic notations for worst case, best case and average case time complexities with example. **Answer:** **Big-Oh(O):** Definition: f(n) is in O(g(n)), denoted f(n) $\in$ O(g(n)), if order of growth of f(n) $\leq$ order of growth of g(n) (within constant multiple), i.e., there exist positive constant c and non-negative integer n0 such that f(n) $\leq$ c g(n) for every n $\geq$ n0	6	1	L2

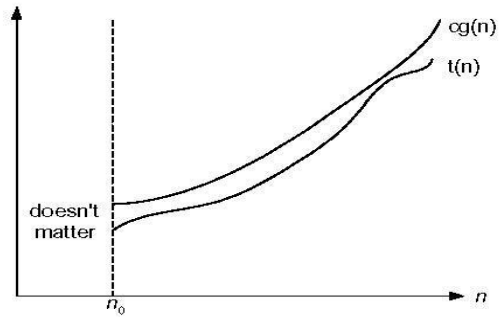


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$ **(2 Mark)**

Big-Omega(Ω):

A function t(n) is said to be in Ω(g(n)), denoted t(n) ∈ Ω(g(n)), if t(n) is bounded below by some constant multiple of g(n) for all large n, i.e., if there exist some positive constant c and some non-negative integer n0 such that

t(n) ≥ cg(n) for all n ≥ n0

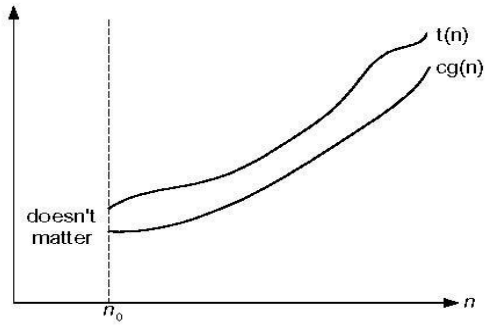


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$ **(2 Marks)**

Theta(θ):

A function t(n) is said to be in Θ(g(n)), denoted t(n) ∈ Θ(g(n)), if t(n) is bounded both above and below by some positive constant multiples of g(n) for all large n, i.e., if there exist some positive constant c1 and c2 and some nonnegative integer n0 such that

c2 g(n) ≤ t(n) ≤ c1 g(n) for all n ≥ n0

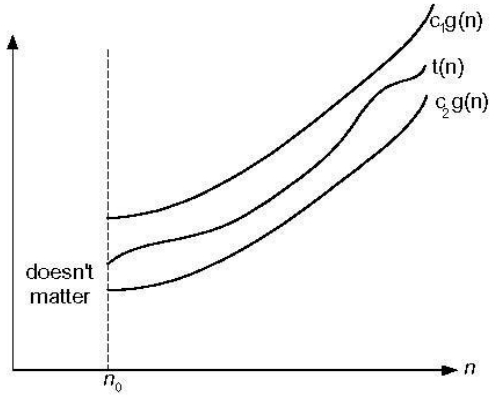
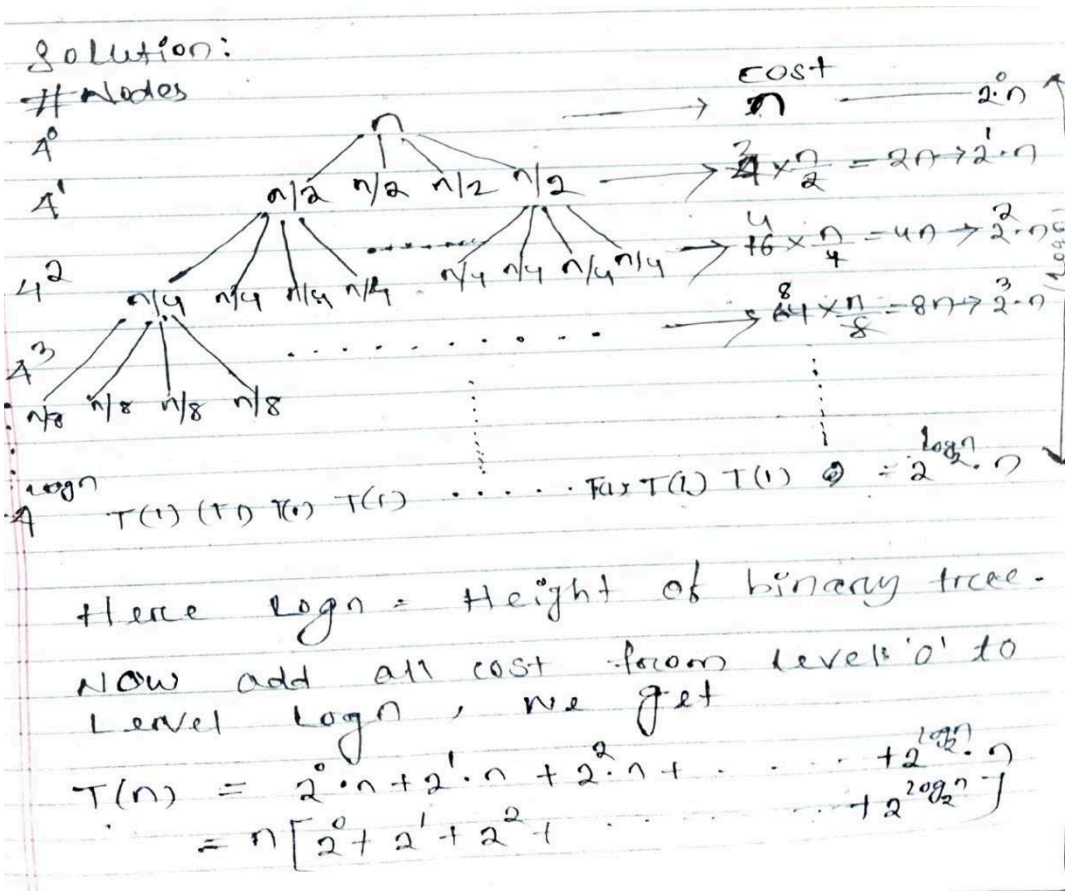


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$ **(2 Marks)**

2	a	Use a recursion tree method to determine a good asymptotic upper bound on the following recurrence:	5	1	L3

$T(n) = 4T(n/2)+n$ for $n>1$, $T(1) =1$ for $n=1$.

Answer:



(3 Marks)

$$= n\left[2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{\log_2 n - 1} + 2^{\log_2 n}\right]$$

$$= n\left[\sum_{i=0}^{\log_2 n - 1} 2^i\right]$$

$$= n \sum_{i=0}^{\log_2 n - 1} 2^i = n\left[2^{\log_2 n - 1 + 1} - 1\right]$$

$$\left[\because \sum_{i=0}^{n} 2^i = 2^{n+1} - 1\right]$$

$$= n\left[2^{\log_2 n - 1 + 1} - 1\right]$$

$$= n\left[2^{\log_2 n} - 1\right]$$

$$= n[n - 1] \qquad (\because 2^{\log_2 n} = n)$$

$$= n^2 - n$$

$$= O(n^2)$$

$$\therefore T(n) = O(n^2)$$

Note: How to prove $2^{\log_2 n} = n$

Let $y = 2^{\log_2 n}$

Put 'log' on both side, with base 2, we get

$\log_2 y = \log_2 2^{\log_2 n}$

$= \log_2 y = \log_2 n \cdot \log_2 2$ $(\because \log m^n = n \log m)$

$= \log_2 y = \log_2 n \cdot 1$ $(\because \log_a a = 1)$

$= \log_2 y = \log_2 n$

$\therefore y = n$

previously, $y = 2^{\log_2 n}$

so, $\boxed{2^{\log_2 n} = n}$ (proved)

—— end ——

(2 Marks)

b

Write a recursive algorithm to search for a key element in an array of size n. Derive an equation for the best-case and worst-case complexity of your algorithm.

Answer:

Pseudocode:

```
BinarySearch(A[0…n-1], key, start, end)
// Input: An integer sorted array A[0…n-1].
// Output: returns mid (if key found) or -1 (key not found)
if   start >end
      return -1
else
mid ——   (start+end)/2
if  key = A[mid]
     return mid
if  key < A[mid]
     return BinarySearch (A, n, key, start, mid-1)
else
     return BinarySearch (A, n, key, mid+1, end)
```

(2.5 Marks)

5 | 1 | L3

Analysis:

Best-Case: When the array is divided into half if key happens to be A[mid], then no recursive calls are needed. Hence, the running time is,

$$T(n) = O(1)$$

Worst-Case:

After first iteration, the length of array = n

After second iteration, the length of array = n/2

After third iteration, the length of array = n/4

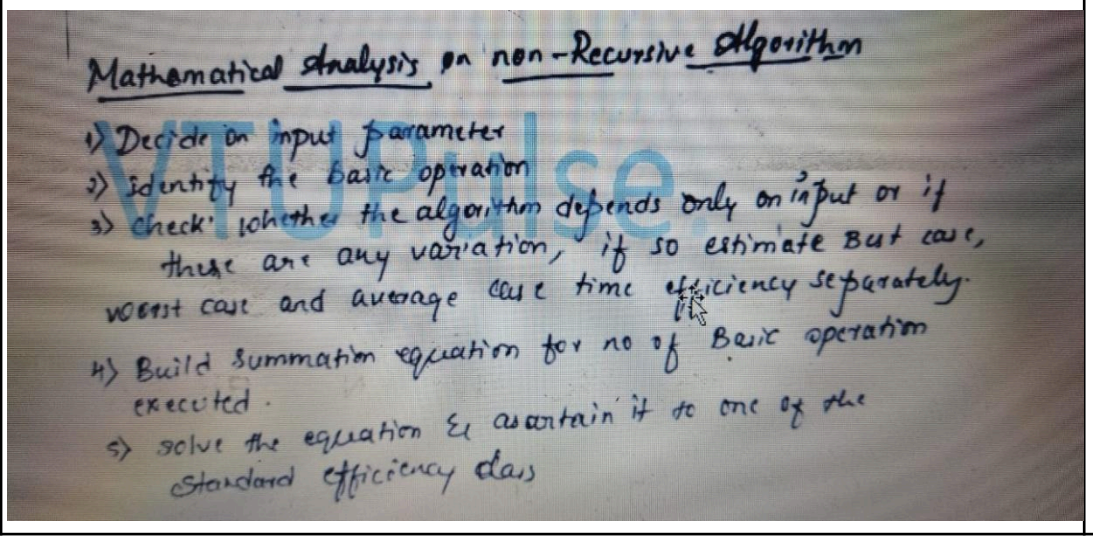
.

.

.

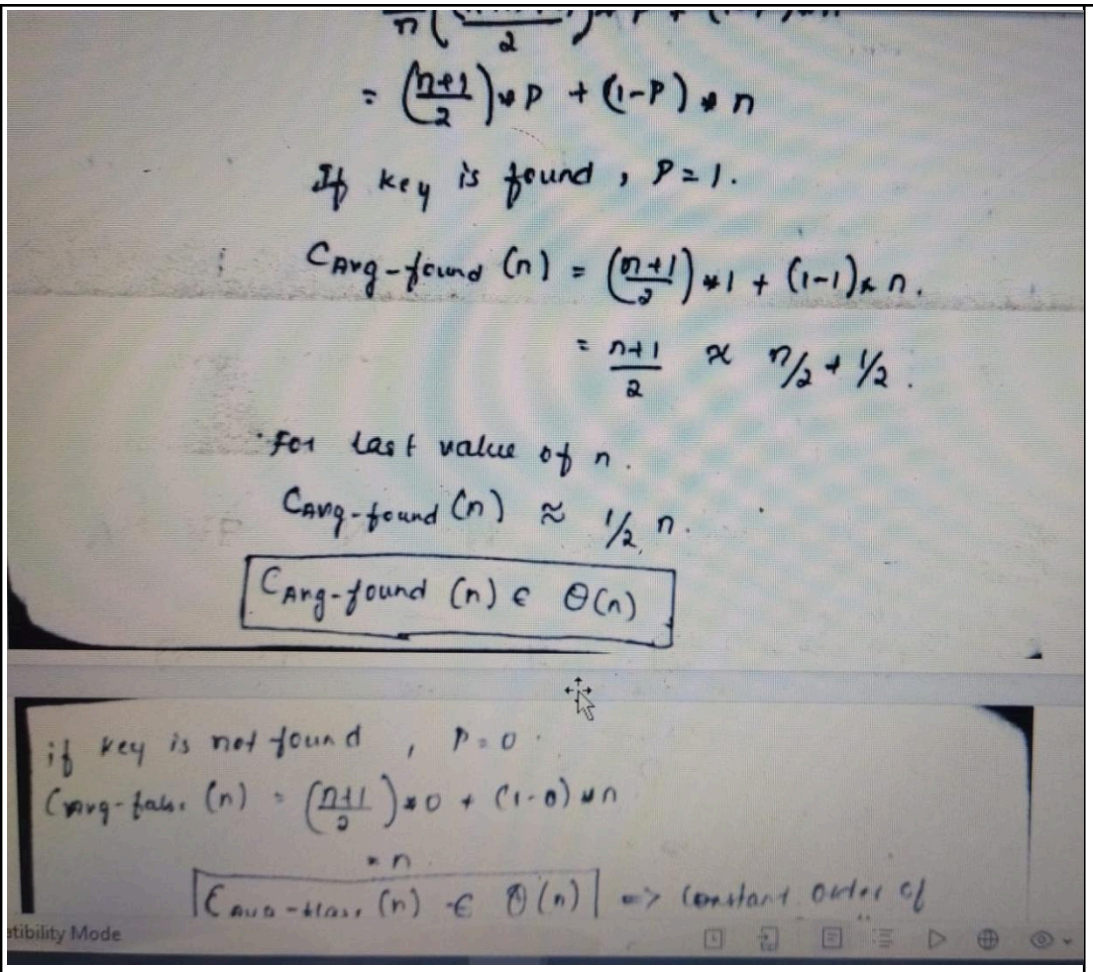
.

.

		After k iterations, the length of array = $n/(2^k)$ Let the length of array become 1 after k iterations So, $n/(2^k) = 1$ $n = 2^k$ put log on both side of equation, we get $\log n = \log 2^k$ $\log n = k\log 2$ $\log n = k.1$ as log2 base 2=1 $\log n = k$ due k iterations happened, the time complexity is $T(K) = O(k)$ Put k= logn $T(n) = O(\log n)$ **Average-Case:** **$T(n) = O(\log n)$** **(2.5 Marks)**			
3	a	Explain the general plan for analysing the efficiency of a non-recursive algorithm with example. **Answer:**  	5	1	L2

		Ex:-
		Algorithm linear_search (A[1....n], key).
		//Input : An array of Integers A[] with n elements and key
		//output : returns true if found else false.
		for i ← 1 to n do.
		if (A[i] == key) then
		return True
		end if
		end for
		
		Analysis
		1. Input parameter - 'n' size of array A
		2. Basic operation - Search /comparision - A[i] = key
		3. Apart from input size 'n' the algorithm produces varying order of growth, therefore estimate time analysis for best case, worst case and average cas seperately			

b		⇒ Best case : if the key is found at first position. comparision is one $C_{Best}(n) = 1$ $\therefore i]$ $C_{Best}(n) \in a(1)$ ⇒ constant order of growth Worst case : if the key is found in last position. $C_{worst}(n) = \sum_{i=1}^{n} 1$ $\left\{ \sum_{i}^{u} \text{constant} = \text{constant} * (u-l+1) \right.$ $= 1.(n-1+1)$ $= n$// $\therefore ii]$ $C_{worst}(n) \in O(n)$ ⇒ linear order of growth Average case : $C_{Avg}(n) = \frac{(1+2+3+4+\dots+n)}{n} * p + (1-p) * n$ $= \frac{1}{n}\left(\frac{(n+1)*n}{2}\right) * p + (1-p) * n$ $= \frac{(n+1)}{2} * p + (1-p) * n$	5 **(2+3)Marks**	1	L2



I) Prove that $100n+5 \neq \Omega(n^2)$

Answer:

$100n + 5 \neq \Omega(n)$

Condition for Big-omega (Ω)

if $f(n) \geq c \cdot g(n)$

then $f(n) \in \Omega(g(n))$ or $f(n) = \Omega(g(n))$

For given data:

$f(n) = 100n + 5$

$g(n) = n$

$f(n) \geq c \cdot g(n)$

So $\therefore$ $100n + 5 \geq c \cdot n$, where $c > 0$, $n \geq n_0$

case-1 if $c_1 = 1$, $n = 1$

$100 \times 1 + 5$ 1·1

$100 + 5$ 1

$105 \not\geq 1$

$100n + 5 \not\geq c \cdot n$

case-2 if $c = 100$, $n = 2$

~~$100 \times 100 + 5$~~

$100n + 5$ $c \cdot n$

$100 \times 2 + 5$ 100×2

$205 \not\geq 200$

$100n + 5 \not\geq c \cdot n$

$c = 200$, $n = 1$

$100n + 5$ $c \cdot n$

$100 \times 1 + 5$ 200×1

$105 < 200$

But $100n + 5 \not\geq c \cdot n$.

for $c = 200$, $n = 1$

It has been proved that $100n + 5 \neq \Omega(n)$

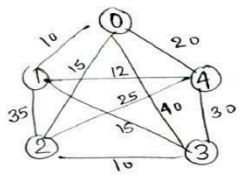
(proved)

(2.5 Marks)

i. (ii)Consider the following algorithm

		Algorithm Guess (A[][]) For i<-0 to n-1 for j<-0 to i a[i][j]<-0 • What does the algorithm compute? • What is basic operation? • What is the efficiency of the algorithm? **Answer:** This algorithm computes the Lower Triangular Matrix. Here bsic operation are multiplication and addition. The time complexity is $O(n^2)$. **(2.5 Marks)**			
4	a	What is a "Brute force "method? Under what condition does the method become desirable? **Answer:** **Brute Force Method:** A brute force method is an approach to problem-solving that involves straightforward, exhaustive search through all possible solutions and picking the best one. It typically involves checking all possibilities without any optimizations or heuristics, relying instead on sheer computational power and thoroughness. **Conditions When Brute Force is Desirable:** 1. **Small Problem Size**: When the problem size (such as the number of elements to consider or the range of values) is small enough that checking all possible solutions is feasible within a reasonable time frame. For example, checking all permutations of a set of 8 elements (8!) is manageable. 2. **No Efficient Algorithm Known**: Sometimes, for certain types of problems, no efficient algorithm with better time complexity than brute force is known or feasible. In such cases, using brute force might be the only practical option. 3. **Verification or Testing:** Brute force is often used as a baseline or verification method to test more complex algorithms. It ensures correctness by comparing results and can be invaluable in validating the outputs of more optimized approaches. 4.**Educational Purposes**: Brute force methods are also valuable in educational contexts to illustrate basic principles of algorithm design, complexity analysis, and problem-solving strategies.	5**(1+4 Marks)**	1	L1
	b	Find the optimal tour of the following given graph using travelling salesman problem(using exhaustive search method): **Answer:**	5 **(5Marks)**	1	L2

Question – 4(b)



There are 5 vertex,
So, all possible routes will be
$\Rightarrow (n-1)!$
$= (5-1)! = 4!$
$\Rightarrow 4 \times 3 \times 2 \times 1 \Rightarrow \boxed{24}$

Here we use Hamiltonian cycle to solve the problem –

1) 0 – 1 – 2 – 3 – 4 – 0 ⇒ 10 + 35 + 10 + 30 + 20 ⇒ 105
2) 0 – 1 – 2 – 4 – 3 – 0 ⇒ 10 + 35 + 25 + 30 + 40 ⇒ 140
3) 0 – 1 – 4 – 2 – 3 – 0 ⇒ 10 + 12 + 25 + 10 + 40 ⇒ 92
4) 0 – 1 – 4 – 3 – 2 – 0 ⇒ 10 + 12 + 30 + 10 + 15 ⇒ 77
5) 0 – 1 – 3 – 4 – 2 – 0 ⇒ 10 + 15 + 30 + 25 + 15 ⇒ 95
6) 0 – 1 – 3 – 2 – 4 – 0 ⇒ 10 + 15 + 10 + 25 + 20 ⇒ 80
7) 0 – 2 – 3 – 1 – 4 – 0 ⇒ 15 + 10 + 15 + 12 + 20 ⇒ (72)
8) 0 – 2 – 3 – 4 – 1 – 0 ⇒ 15 + 10 + 30 + 12 + 10 ⇒ 77
9) 0 – 2 – 1 – 4 – 3 – 0 ⇒ 15 + 35 + 12 + 30 + 40 ⇒ 132
10) 0 – 2 – 1 – 3 – 4 – 0 ⇒ 15 + 35 + 15 + 30 + 20 ⇒ 115
11) 0 – 2 – 4 – 3 – 1 – 0 ⇒ 95
12) 0 – 2 – 4 – 1 – 3 – 0 ⇒ 107
13) 0 – 3 – 2 – 4 – 1 – 0 ⇒ 97
14) 0 – 3 – 2 – 1 – 4 – 0 ⇒ 117
15) 0 – 3 – 1 – 4 – 2 – 0 ⇒ 107
16) 0 – 3 – 1 – 2 – 4 – 0 ⇒ 135
17) 0 – 3 – 4 – 1 – 2 – 0 ⇒ 132
18) 0 – 3 – 4 – 2 – 1 – 0 ⇒ 140
19) 0 – 4 – 3 – 2 – 1 – 0 ⇒ 105
20) 0 – 4 – 3 – 1 – 2 – 0 ⇒ 115
21) 0 – 4 – 1 – 2 – 3 – 0 ⇒ 117
22) 0 – 4 – 1 – 3 – 2 – 0 ⇒ (72)
23) 0 – 4 – 2 – 3 – 1 – 0 ⇒ 80
24) 0 – 4 – 2 – 1 – 3 – 0 ⇒ 135

So, the shortest path are –

0 – 2 – 3 – 1 – 4 – 0 ⇒ 72
0 – 4 – 1 – 3 – 2 – 0 ⇒ 72

Ans

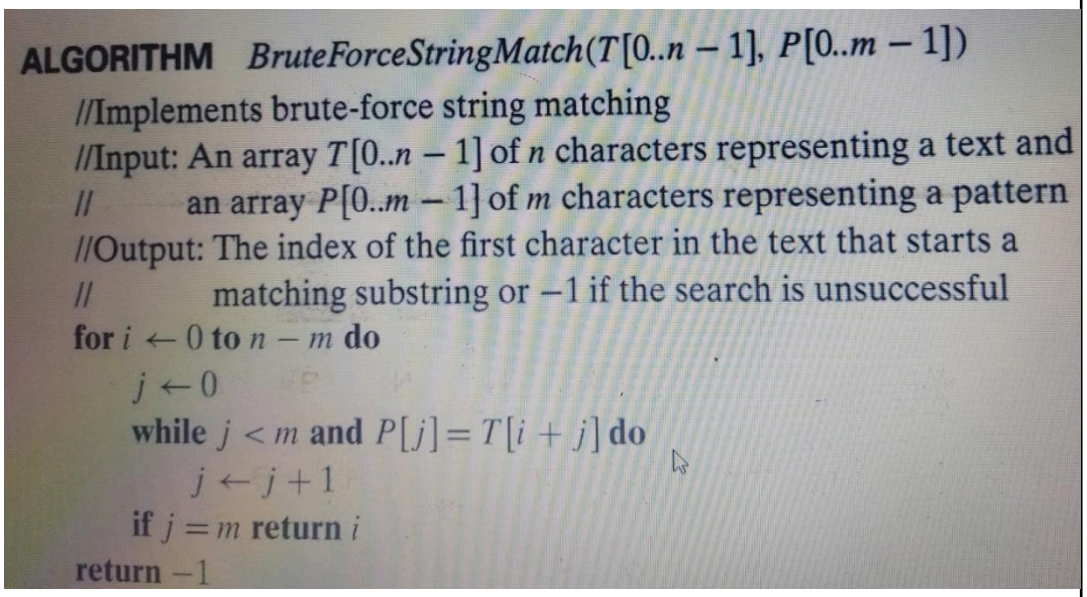
5	a	Write a C/C++ code for implementing Insertion Sort With time complexity. **Answer:**	5	1	L2

Insertion sort:

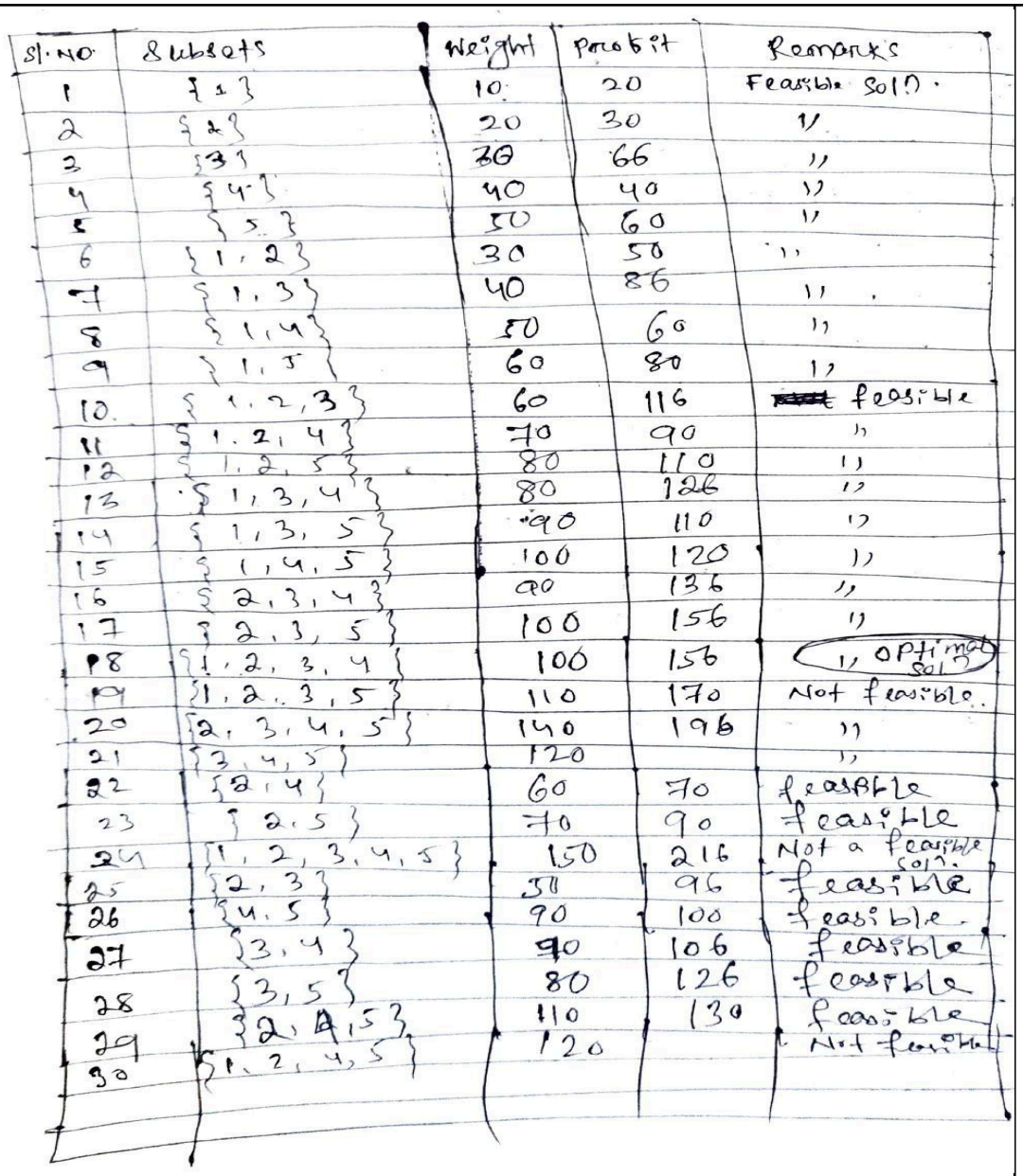
Algorithm InsertionSort(A[0..n-1], n)
//Input: An array A[0…n-1]
//Output: Sorted array A
For j←1 to n-1 do
  k←A[j]
  i←j-1
  While i≥0 and k<A[i] do
      A[i+1] ←A[i]
      i←i-1
 A[i+1] ←k
end InsertionSort

Code:

```
#include<iostream>
using namespace std;
const long MAX = 20;
void InsertionSort(int [], int);
int main()
{
   int a[MAX];
   int n;
   cout <<"Enter array size:";
   cin>>n;
   cout <<"Enter the array: ";
   for(int i=0; i<n; i++)
   cin>>a[i];
   InsertionSort(a, n);
   cout <<"The sorted array is:"<<endl;
   for(int i=0;i<n;i++)
     cout << a[i] <<"";
     cout <<endl;
}
void InsertionSort(int a[], int n)
{
   int j, i, k;
   for(j=1; j<n;j++)
   {
k=a[j];
      for(i=j-1;(i>=0 && k<a[i]);i--)
      a[i+1] = a[i];
      a[i+1] = k;
   }
}
```

	b	Write a Algorithm for Pattern Matching by using Brute force technique?
		Answer:
		pattern: a string of m characters to search for
		text: a (longer) string of n characters to search in
		problem: find a substring in the text that matches the pattern
		Brute-force algorithm
		Step 1 Align pattern at beginning of text
		Step 2 Moving from left to right, compare each character of pattern to the corresponding character in text until
		• all characters are found to match (successful search);
		or • a mismatch is detected
		Step 3 While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2
		(2.5 Marks)
		Example:
		Pattern: 001011
		Text: 10010101101001100101111010
		(2.5 Marks)
		Pseudocode:
		 
		ALGORITHM *BruteForceStringMatch*($T[0..n-1]$, $P[0..m-1]$)
		//Implements brute-force string matching
		//Input: An array $T[0..n-1]$ of n characters representing a text and
		// an array $P[0..m-1]$ of m characters representing a pattern
		//Output: The index of the first character in the text that starts a
		// matching substring or -1 if the search is unsuccessful
		for $i \leftarrow 0$ **to** $n-m$ **do**
		$j \leftarrow 0$
		while $j < m$ **and** $P[j] = T[i+j]$ **do**
		$j \leftarrow j+1$
		if $j = m$ **return** i
		return -1
		Time complexity: O(mn)
			5	1	L2

6	a	For the below given Knapsack problem instance, find the solution using exhaustive search method: Where, n=5, [w1,w2,w3,w4,w5] = [10,20,30,40,50] [p1,p2,p3,p4,p5] = [20,30,66,40,60] M=100 **Answer:**	5	1	L2

					
b	Write a recurrence relation for Fibonacci series and solve it. **Answer:** The Fibonacci algorithm (recursive) Fib(n) { If n<=1 return n Else Return F(n-1) + F(n-2) 65 Department of ISE BMS Institute of Technology and Mgmt BMS Institute of Technology and Mgmt • The recurrence equation for this problem is: T(n) = T(n-1) + T(n-2) for n>1 and the initial conditions are T(0) =0, T(1) = 1 Solution to recurrence relation: T(n) = T(n-1) + T(n-2) T(n) –T(n-1) –T(n-2) = 0 This is of the form ax(n) +bx(n-1) +cx(n-2) =0 Which is a homogeneous second order linear relation with constant co-efficients	5	1	L2

CI **CCI** **HoD**