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

Internal Assessment Test 3 – Feb 2023

Sub:	Design and Analysis of Algorithms			Sub Code:	21CS42	Branch:	CSE
Date:		Duration:	90 mins	Max Marks: 50	Sem / Sec:	IV(A, B & C)	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	Consider the following algorithm and derive the time complexity of the following algorithm using analytical framework:	6	CO1 & CO2	L3

```
ALGORITHM  MatrixMultiplication(A[0..n − 1, 0..n − 1], B[0..n − 1, 0..n − 1])
    //Multiplies two square matrices of order n by the definition-based algorithm
    //Input: Two n × n matrices A and B
    //Output: Matrix C = AB
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            C[i, j] ← 0.0
            for k ← 0 to n − 1 do
                C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
    return C
```

SOLUTION:

Obviously, there is just one multiplication executed on each repetition of the algorithm's innermost loop, which is governed by the variable k ranging from the lower bound 0 to the upper bound $n-1$. Therefore, the number of multiplications made for every pair of specific values of variables i and j is

$$\sum_{k=0}^{n-1} 1,$$

and the total number of multiplications $M(n)$ is expressed by the following triple sum:

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1.$$

Now, we can compute this sum by using formula (S1) and rule (R1) given above. Starting with the innermost sum $\sum_{k=0}^{n-1} 1$, which is equal to n (why?), we get

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

1 (b)		4	CO1 & CO2	L3

```
ALGORITHM Enigma (A [0… n-1, 0…. n-1])
    for i ← 0 to n - 2 do
        for j ← i + 1 to n – 1 do
            if A [i, j] ≠ A [j, i]
                return false
            end for
        end for
    return true
```

i. What does this algorithm compute?
ii. Identify the basic operation.
iii. Calculate, how many times the basic operation is executed.
iv. Derive the efficiency class of this algorithm.

SOLUTION

i. Algorithm checks whether a given matrix is its transpose or not.
ii. The basic operation is the comparison
iii. The basic operation is calculated n* (n-1)/2 times as j starts from i + 1.
iv. The efficiency class of this algorithm is O(n^2)

2 (a)	The Factorial function n! has a value 1 when n<=1 and a value of n*(n-1) when n>1. For the above description design the following:	6	CO3	L2

i. Recursive algorithm (pseudocode) to compute n!
ii. Iterative algorithm (pseudocode) to compute n!

SOLUTION

Recursive Algorithm

```
ALGORITHM F(n)
    //Computes n! recursively
    //Input: A nonnegative integer n
    //Output: The value of n!
    if n = 0 return 1
    else return F(n − 1) ∗ n
```

Iterative Algorithm

```
INPUT number
SET factorial := 1, i := 1
WHILE i <= number DO
    COMPUTE factorial := factorial * i
    INCREASE i by 1
END LOOP
```

2 (b)	Explain Big- O, Big – Ω, Big -Θ notations with examples.	4	CO2	L2

SOLUTION

O-notation

DEFINITION A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \le cg(n) \quad \text{for all } n \ge n_0.$$

$$100n + 5 \le 100n + 5n \text{ (for all } n \ge 1) = 105n$$

Ω-notation

DEFINITION A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \ge cg(n) \quad \text{for all } n \ge n_0.$$

Here is an example of the formal proof that $n^3 \in \Omega(n^2)$:

$$n^3 \ge n^2 \quad \text{for all } n \ge 0,$$

Θ-notation

DEFINITION A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, 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 constants c_1 and c_2 and some nonnegative integer n_0 such that

$$c_2 g(n) \le t(n) \le c_1 g(n) \quad \text{for all } n \ge n_0.$$

The definition is illustrated in Figure 2.3.

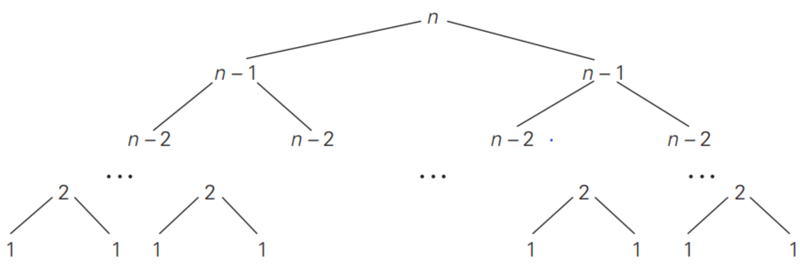
For example, let us prove that $\frac{1}{2}n(n-1) \in \Theta(n^2)$. First, we prove the right inequality (the upper bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \quad \text{for all } n \ge 0.$$

Second, we prove the left inequality (the lower bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \ge \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \text{ (for all } n \ge 2) = \frac{1}{4}n^2.$$

Hence, we can select $c_2 = \frac{1}{4}$, $c_1 = \frac{1}{2}$, and $n_0 = 2$.

3(a)	 The above diagram represents a recursive tree with the recursive calls. From the diagram, identify the recurrence relations and solve the same using analytical framework to find the time efficiency. **SOLUTION** The recurrence relations from the above tree are written as follows: $M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1.$ $M(n) = 2M(n-1) + 1 \quad \text{for } n > 1,$ $M(1) = 1.$ $M(n) = 2M(n-1) + 1 \qquad \text{sub. } M(n-1) = 2M(n-2) + 1$ $= 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1 \quad \text{sub. } M(n-2) = 2M(n-3) + 1$ $= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3M(n-3) + 2^2 + 2 + 1.$ $M(n) = 2^iM(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^iM(n-i) + 2^i - 1.$ $M(n) = 2^{n-1}M(n-(n-1)) + 2^{n-1} - 1$ $= 2^{n-1}M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1.$ The time efficiency is O (2^n)	6	CO1 & CO2	L3
3(b)	With the help of a simple example, give the general plan of analyzing a recursive function using analytical framework. **SOLUTION** **General Plan for Analyzing the Time Efficiency of Recursive Algorithms** **1.** Decide on a parameter (or parameters) indicating an input's size. **2.** Identify the algorithm's basic operation. **3.** 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. **4.** Set up a recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed. **5.** Solve the recurrence or, at least, ascertain the order of growth of its solution. **EXAMPLE (Can be any recursive function or algorithm)**	4	CO1 & CO2	L3

	ALGORITHM $F(n)$
	//Computes $n!$ recursively
	//Input: A nonnegative integer n
	//Output: The value of $n!$
	if $n = 0$ **return** 1
	else return $F(n-1) * n$
	
	$M(n) = \underbrace{M(n-1)}_{\text{to compute } F(n-1)} + \underbrace{1}_{\text{to multiply } F(n-1) \text{ by } n}$ for $n > 0$.
	
	$M(n) = M(n-1) + 1$ substitute $M(n-1) = M(n-2) + 1$
	$= [M(n-2) + 1] + 1 = M(n-2) + 2$ substitute $M(n-2) = M(n-3) + 1$
	$= [M(n-3) + 1] + 2 = M(n-3) + 3.$
	
	$M(n) = M(n-1) + 1 = \cdots = M(n-i) + i = \cdots = M(n-n) + n = n.$			

4(a)	With the help of an example prove that $t_1(n)$ € $O(g_1(n))$ and $t_2(n)$ € $O(g_2(n))$ then $t_1(n)+t_2(n)$ €$O(\max\{g_1(n),g_2(n)\})$ **SOLUTION** If f(n) is O(g(n)), and d(n) is O(e(n)) then, f(n) + e(n) is O(max{g(n), e(n)}) $f(n) = n = O(n)$ $d(n) = n^2 = O(n^2)$ $f(n) + d(n) = n + n^2$ $= O(n^2)$ $= e(n)$	2	CO2	L2
4(b)	Design an algorithm to search an element in an array using sequential search. Discuss worst case, best case and average case efficiency of this algorithm. **SOLUTION**	4	CO2 & CO3	L2

```
ALGORITHM SequentialSearch2($A[0..n]$, $K$)
    //Implements sequential search with a search key as a sentinel
    //Input: An array $A$ of $n$ elements and a search key $K$
    //Output: The index of the first element in $A[0..n-1]$ whose value is
    //        equal to $K$ or $-1$ if no such element is found
    $A[n] \leftarrow K$
    $i \leftarrow 0$
    while $A[i] \neq K$ do
        $i \leftarrow i + 1$
    if $i < n$ return $i$
    else return $-1$
```

Best Case: O(1)
Average Case : O(n)
Worst Case : O(n)

4(c)	Consider the elements of array {E, X, A, M, P, L, E}. Perform bubble sort and selection sort on these elements and list out the number of comparisons taken for each sort. Identify which algorithm performed better.	4	CO3	L3

SOLUTION
Bubble Sort:

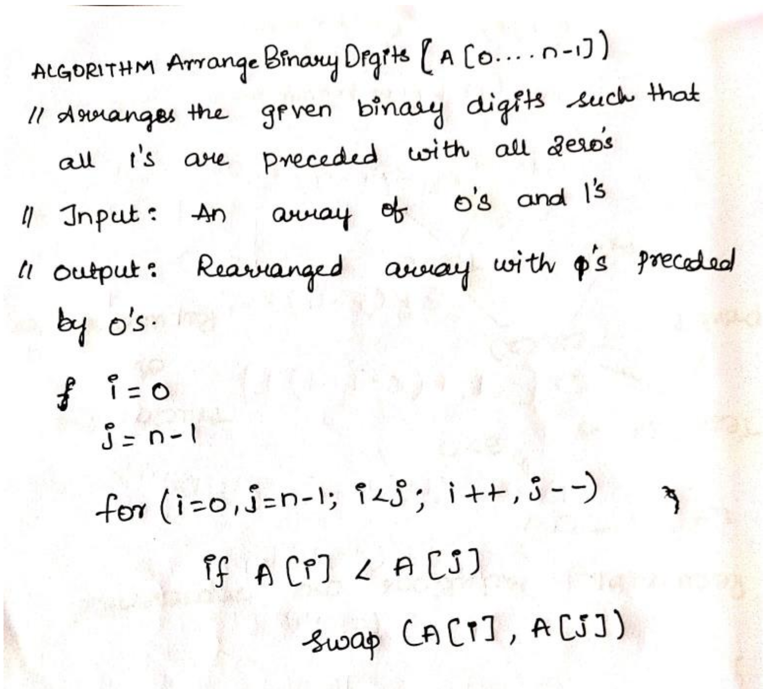
$$\begin{array}{lllllllllllll}
E & \overset{?}{\leftrightarrow} & X & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E \\
E & & A & & X & \overset{?}{\leftrightarrow} & M & & P & & L & & E \\
E & & A & & M & & X & \overset{?}{\leftrightarrow} & P & & L & & E \\
E & & A & & M & & P & & X & \overset{?}{\leftrightarrow} & L & & E \\
E & & A & & M & & P & & L & & X & \overset{?}{\leftrightarrow} & E \\
E & & A & & M & & P & & L & & E & & |X \\
E & \overset{?}{\leftrightarrow} & A & & M & & P & & L & & E & & \\
A & & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & P & \overset{?}{\leftrightarrow} & L & & E & & \\
A & & E & & M & & L & & P & \overset{?}{\leftrightarrow} & E & & \\
A & & E & & M & & L & & E & & |P & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & M & \overset{?}{\leftrightarrow} & L & & E & & & & \\
A & & E & & L & & M & \overset{?}{\leftrightarrow} & E & & & & \\
A & & E & & L & & E & & |M & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & \overset{?}{\leftrightarrow} & E & & & & & & \\
A & & E & & E & & |L & & & & & & \\
A & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & E & \overset{?}{\leftrightarrow} & L & & & & & &
\end{array}$$

Selection Sort

$$\begin{array}{lllllll}
|\,E & X & \mathbf{A} & M & P & L & E \\
A\,| & X & \mathbf{E} & M & P & L & E \\
A & E\,| & X & M & P & L & \mathbf{E} \\
A & E & E\,| & M & P & \mathbf{L} & X \\
A & E & E & L\,| & P & \mathbf{M} & X \\
A & E & E & L & M\,| & \mathbf{P} & X \\
A & E & E & L & M & P\,| & X
\end{array}$$

Number of Comparisons in Bubble sort and Selection sort are: 21

5(a)	Design an algorithm for checking whether all elements in a given array are distinct.
	
	SOLUTION
	
	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
		4	CO2	L2

5 (b)	You have a row of binary digits arranged randomly. Arrange them in such an order that all 0's precede all 1's or vice-versa. The only constraint in arranging them is that you are allowed to interchange the positions of binary digits if they are not similar.
	Ex: 1 1 0 1 0 1 0 0 → 0 0 0 0 1 1 1 1
	Design an algorithm for solving this problem
	
	SOLUTION
	
	
	ALGORITHM ArrangeBinaryDigits [A [0....n-1])
	// Arranges the given binary digits such that
	  all 1's are preceded with all zeros
	// Input: An array of 0's and 1's
	// output: Rearranged array with φ's preceded
	  by 0's.
	  { i = 0
	    j = n-1
	    for (i=0, j=n-1; i<j; i++, j--) }
	        if A[i] < A[j]
	            swap (A[i], A[j])
		6	CO2
			&
			CO3
				L3

6 (a)	Consider the problem of counting, in a given text, the number of substrings that start with A and end with B. (For example, there are four such strings in CABAAXBYA). Design a brute force algorithm for this problem and determine its efficiency class **SOLUTION**	7	CO2 & CO3	L3

Bruteforce Algorithm

```
Algorithm CountSubString1(T[0…n-1])
// Implements a bruteforce algorithm to count substrings starting with 'A' and ending with 'B'
// Input:  String T consisting of n characters
// Output: Number of substrings in T starting with 'A' and ending with 'B'
count ← 0
for i in 0 to n-2 do
        if T[i] = 'A'
                for j in i+1 to n-1 do
                        if T[j] = 'B'  count ← count +1
return count
```

Efficient Algorithm (using extra space)

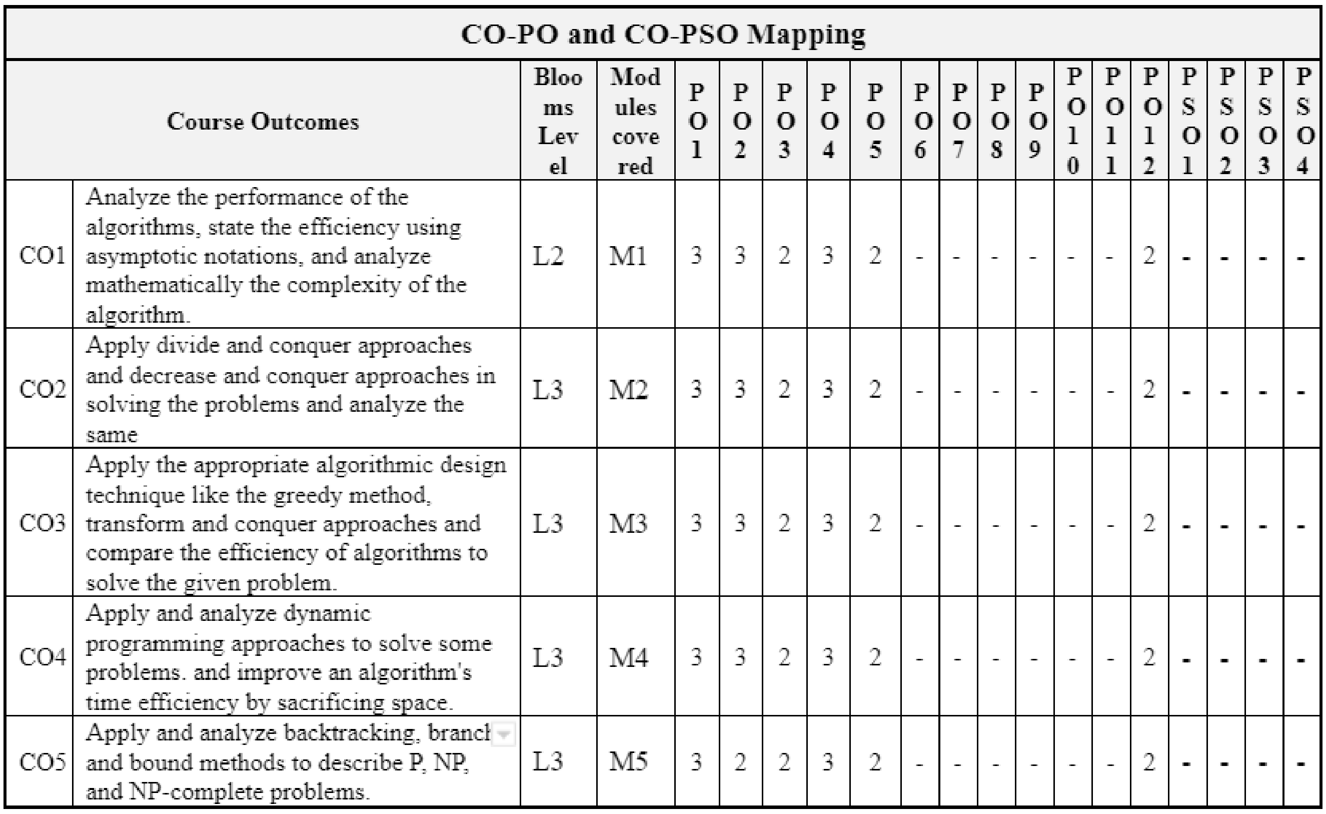
```
Algorithm CountSubString2(T[0…n-1])
// Linear time algorithm to count substrings starting with 'A' and ending with 'B'
// Input:  String T consisting of n characters
// Output: Number of substrings in T starting with 'A' and ending with 'B'

Let C[0…n-1] be a new array of size n
if T[0] = 'A'    C[0] = 1
else  C[0] = 0
for i in 1 to n-2 do
        if T[i] = 'A'   C[i] ← C[i-1] + 1  // Number of A's in prefix string T[0…i]
        else C[i] ← C[i-1]
count ← 0
for i in 1 to n-1 do
        if T[i] = 'B'   count = count + C[i]
return count
```

6(b)	Give an example of a text of length n and a pattern of length m that constitutes a worst-case input for the brute-force string matching algorithm. Exactly how many character comparisons will be made for such input? **SOLUTION** The text composed of n zeros and the pattern $\underbrace{0\ldots0}_{m-1}1$ is an example of the worst-case input. The algorithm will make $m(n-m+1)$ character comparisons on such input.	3	CO3	L3

CI **CCI** **HOD**

PO Mapping



CO-PO and CO-PSO Mapping																				
Course Outcomes		Blooms Level	Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Analyze the performance of the algorithms, state the efficiency using asymptotic notations, and analyze mathematically the complexity of the algorithm.	L2	M1	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO2	Apply divide and conquer approaches and decrease and conquer approaches in solving the problems and analyze the same	L3	M2	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO3	Apply the appropriate algorithmic design technique like the greedy method, transform and conquer approaches and compare the efficiency of algorithms to solve the given problem.	L3	M3	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO4	Apply and analyze dynamic programming approaches to solve some problems. and improve an algorithm's time efficiency by sacrificing space.	L3	M4	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO5	Apply and analyze backtracking, branc and bound methods to describe P, NP, and NP-complete problems.	L3	M5	3	2	2	3	2	-	-	-	-	-	-	2	-	-	-	-

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.

L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				