ANSWER KEY

REG.NO:

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CONTINUOUS ASSESSMENT TEST - I
WINTER SEMESTER 2024-2025

SLOT: A2 +TA2

Programme Name & Branch	:	**B.Tech (Computer Science & Engineering)**
Course Code and Course Name	:	**Design and Analysis of Algorithms & BCSE204L**
Faculty Name(s)	:	**Dr.P.Iyappan**
Class Number(s)	:	**Common to all**
Date of Examination	:	**27.01.2025**
Exam Duration	:	**90 minutes** **Maximum Marks: 50**

General instruction(s):

- Answer All Questions
- M - Max mark; CO – Course Outcome; BL – Blooms Taxonomy Level (1 – Remember, 2 – Understand, 3 – Apply, 4 – Analyze, 5 – Evaluate, 6 – Create)
- Course Outcomes:
  CO-1: Apply the mathematical tools to analyse and derive the running time of the algorithms.
  CO-2: Demonstrate the major algorithm design paradigms.

Q. No	Question	M	CO	BL

1.	a. Design an algorithm by finding the required design paradigm to find the highest and lowest marks scored by students in the exam by fulfilling the below parameters and derive the respective recurrence relation for the same.
	i. Base case 1(If the input is only one element) give solution i.e what is highest and lowest?
	ii. Base case 2(If the input is two elements) give solution.
	iii. If the input is more than two elements, divide the inputs into sub-problems and use the recursive function for solving the sub-problems and write the procedure for combining the sub-solutions.
	Solution: Divide and Conquer Strategy
	Algorithm MaxMin(i, j, max, min)
	// $a[1:n]$ is a global array. Parameters i and j are integers,
	// $1 \le i \le j \le n$. The effect is to set max and min to the
	// largest and smallest values in $a[i:j]$, respectively.
	{
	if $(i = j)$ **then** $max := min := a[i]$; // Small(P)
	else if $(i = j - 1)$ **then** // Another case of Small(P)
	{
	if $(a[i] < a[j])$ **then**
	{
	$max := a[j]$; $min := a[i]$;
	}
	else
	{
	$max := a[i]$; $min := a[j]$;
	}
	}
	else
	{ // If P is not small, divide P into subproblems.
	// Find where to split the set.
	$mid := \lfloor (i + j)/2 \rfloor$;
	// Solve the subproblems.
	MaxMin(i, mid, max, min);
	MaxMin($mid + 1, j, max1, min1$);
	// Combine the solutions.
	if $(max < max1)$ **then** $max := max1$;
	if $(min > min1)$ **then** $min := min1$;
	}
	}
		5	C01	BL4

	Recurrence Relation: $$T(n) = \begin{cases} T(\lceil n/2 \rceil) + T(\lceil n/2 \rceil) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$ b. Demonstrate the Recursive tree method to find the asymptotic complexity of the following recurrence equation (Note: Recursive tree with level wise time complexity should be given). **T(n)= 2T(n/2)+cn** **Solution : nlogn**	5	C01	
2.	Demonstrate how the Huffman code is used to make data smaller by Compressing it? Consider the following string "BCAADDDCCACACAC" compress it and find out how many minimum numbers of bits are required for transmitting the above said string. Write down the needed process and the respective algorithm and analyse the time complexity.(5M) Algorithm: HUFFMAN(C) 1 $n = \|C\|$ 2 $Q = C$ 3 for $i = 1$ to $n - 1$ 4 allocate a new node z 5 $z.left = x =$ EXTRACT-MIN(Q) 6 $z.right = y =$ EXTRACT-MIN(Q) 7 $z.freq = x.freq + y.freq$ 8 INSERT(Q, z) 9 return EXTRACT-MIN(Q) // return the root of the tree Time Complexity : **O(n.log n)** i. Calculate the frequency of each character(1M) A: 5 times B: 1 time C: 6 times D: 3 times ii. Build a priority Queue(or min heap) (1M) Priority Queue : B: 1 D: 3 A: 5 C: 6 iii. Build the Huffman Tree(1M)	10	C01	BL3

	iv. Generate the Huffman Codes(1M) C: 1 A: 01 B: 000 D: 001 v. Encode the Original String, write the encoded string and find how many bits are in the encoded string. (1M) **Encoded String:** 000101010010010011101101011011 **No of Bits required:** 30			
3.	Given an array consisting of number of following elements {-2, -5, 6, -2, -3, 1, 5, -6}. Find a contiguous non-empty subarray within the array that has the maximum sum, and return the sum as well as the subarray elements. Implement using appropriate design paradigm and give the optimized algorithm and derive the time complexity. Max Subarray Sum: 7 Subarray Elements are: {6,-2,-3,1,5} **Solution: Divide and Conquer Strategy** FIND-MAXIMUM-SUBARRAY($A, low, high$) 1 **if** $high == low$ 2 **return** $(low, high, A[low])$ // base case: only one element 3 **else** $mid = \lfloor (low + high)/2 \rfloor$ 4 $(left\text{-}low, left\text{-}high, left\text{-}sum) =$ FIND-MAXIMUM-SUBARRAY(A, low, mid) 5 $(right\text{-}low, right\text{-}high, right\text{-}sum) =$ FIND-MAXIMUM-SUBARRAY$(A, mid + 1, high)$ 6 $(cross\text{-}low, cross\text{-}high, cross\text{-}sum) =$ FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$ 7 **if** $left\text{-}sum \geq right\text{-}sum$ and $left\text{-}sum \geq cross\text{-}sum$ 8 **return** $(left\text{-}low, left\text{-}high, left\text{-}sum)$ 9 **elseif** $right\text{-}sum \geq left\text{-}sum$ and $right\text{-}sum \geq cross\text{-}sum$ 10 **return** $(right\text{-}low, right\text{-}high, right\text{-}sum)$ 11 **else return** $(cross\text{-}low, cross\text{-}high, cross\text{-}sum)$	10	C01	BL4

	FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$ 1 $left\text{-}sum = -\infty$ 2 $sum = 0$ 3 **for** $i = mid$ **downto** low 4 $sum = sum + A[i]$ 5 **if** $sum > left\text{-}sum$ 6 $left\text{-}sum = sum$ 7 $max\text{-}left = i$ 8 $right\text{-}sum = -\infty$ 9 $sum = 0$ 10 **for** $j = mid + 1$ **to** $high$ 11 $sum = sum + A[j]$ 12 **if** $sum > right\text{-}sum$ 13 $right\text{-}sum = sum$ 14 $max\text{-}right = j$ 15 **return** $(max\text{-}left, max\text{-}right, left\text{-}sum + right\text{-}sum)$ Time Complexity : **O(nlog n)**			
4.	Consider a knapsack bag where n = 4, and the values (profits) are given by {10, 12, 12, 18} and the weights given by {2, 4, 6, 9}. The maximum weight is given by $W = 15$. The solution is to find subset of objects such that the total value is maximized, and the sum of weights of the objects does not exceed a given capacity W. Implement and write the algorithm using Dynamic Programming strategy and derive time complexity. **Solution: Dynamic Programming** Max Profit: 40 Weights Considered: {1,1,0,1} Formula for Filling the Table: **V[i,j] = Max[v[i-1,j], v[i-1,j-w_i] + v_i** Time Complexity: • Time for constructing the table: O(nW) • Time taken for Composition of Optimal load: O(n+W)	10	C02	BL1
5.	i. In real time application, many problems which deals with searching for set of solutions or which ask for an optimal solution satisfying some constraints can be solved using which design paradigm?. Define the Design paradigm and Differentiate implicit and explicit Constraints with example. **Solution: Backtracking** **Definition :**	10	C02	BL2

	It is a systematic approach that helps find solutions by incrementally making choices, and when necessary, undoing those choices to explore alternative paths. This method is particularly useful for problems where there are multiple possible solutions, but not all of them meet certain constraints or criteria. Example: 8-Queens Problem- Place 8 queens on the 8x8 chess board so that no two queens attack each other, i.e. no two of them are on the same row, column, or diagonal. **Explicit Constraints:** The explicit constraints using this formulation are: Si = {1,2,...,8}, 1≤ i ≤ n The solution space consist of 8 tuples. **Implicit Constraints:** The implicit constraints for this problem are: 1. No two xi's can be same (i.e., all queens must be placed on different columns). 2. No two queens can be on the same diagonal. ii. Demonstrate N queen (where n is 5), the problem is to place n chess Queens on N*N chess board, So that no queens should attack each other. Show the steps that by involving the appropriate design paradigm to attain the list of solutions where no queens should attack each other. Discuss the algorithm and analyse its time Complexity.			

```
Algorithm NQueens(k, n)
// Using backtracking, this procedure prints all
// possible placements of n queens on an n × n
// chessboard so that they are nonattacking.
{
    for i := 1 to n do
    {
        if Place(k, i) then
        {
            x[k] := i;
            if (k = n) then write (x[1 : n]);
            else NQueens(k + 1, n);
        }
    }
}
```

```
Algorithm Place(k, i)
// Returns true if a queen can be placed in kth row and
// ith column. Otherwise it returns false. x[ ] is a
// global array whose first (k – 1) values have been set.
// Abs(r) returns the absolute value of r.
{
    for j := 1 to k – 1 do
        if ((x[j] = i) // Two in the same column
            or (Abs(x[j] – i) = Abs(j – k)))
                // or in the same diagonal
            then return false;
    return true;
}
```