Registration Number : 22BPS1032

VIT
Vellore Institute of Technology
CHENNAI

Continuous Assessment Test 2 - October 2024

Programme	B.Tech.(CSE)	Semester	Fall 2024-25
Course	Design and Analysis of Algorithms	Code	BCSE204L
Faculty	Dr B Indira Dr G Kavipriya Dr N Sivaramakrishnan Dr D Selvam Dr J Omana Dr M Raja	Slot/Class No.	A2+TA2 CH2024250100959 CH2024250100963 CH2024250101368 CH2024250101371 CH2024250101375 CH2024250102307
Time	90 Minutes	Max. Marks	50

Instructions:

- Answer all the FIVE questions.
- If any assumptions are required, assume the same and mention those assumptions in the answer script.
- Use of intelligence is highly appreciated.
- Your answer for all the questions should have both the 'design' component and the 'analysis component'
- The 'Design' component should consist: understanding of the problem, logic to develop the pseudocode, illustration, pseudocode.
- The 'Analysis' component should consist: Proof-of-Correctness, Computation of T(n), Time-complexity.

1. Consider a positive integer n representing the size of an $n \times n$ matrix denoted as L, where each entry in L_{ij} belongs to the set $\{1, 2, \ldots, n\}$. You are given a partially filled matrix L as follows:

$$L = \begin{bmatrix} 1 & - & - & - \\ - & 2 & - & - \\ - & - & 3 & - \\ - & - & - & 4 \end{bmatrix}$$

Your task is to complete the partially filled matrix by ensuring the following constraints: Each number appears exactly once in every row. Formally, for any row $i \in \{1, 2, \ldots, n\}$, the set of entries in that row, $\{L_{i1}, L_{i2}, \ldots, L_{in}\}$, must contain no repeated elements and must be equal to $\{1, 2, \ldots, n\}$. Each number appears exactly once in every column, for any column $j \in \{1, 2, \ldots, n\}$, the set of entries in that column, $\{L_{1j}, L_{2j}, \ldots, L_{nj}\}$, must contain no repeated elements and must also be equal to $\{1, 2, \ldots, n\}$. Apply a suitable algorithm to explore all possible ways to fill the incomplete matrix by satisfying the above constraints. [10 marks]

[**Rubrics**: Logic: 2 mark, Illustration: 3 marks, Pseudocode : 3 marks, Time-complexity : 2 mark]

2. Let x and y be two odd integers. A string $S[1 \ldots x]$ is said to be a First-Half Match if the first half of S appears somewhere in the second half of a text $U[1 \ldots y]$. Similarly, S is said to be a Second-Half Match, if the second half of S appears in the first half of U. For example, if $\mathbf{S} = \mathbf{ABCDE}$, then the first half of S is **AB**, and the second half is **CDE**. If $\mathbf{U} = \mathbf{XYCDEFGAB}$, **AB** is found in the second half of

U at position 7 (First-Half Match), and **CDE** is found in the first half of U at position 2 (Second-Half Match). Design a pseudocode that determines whether S is a First-Half Match, a Second-Half Match, or both, and calculate the corresponding shift (starting index) of the match in U. If no match is found, the output should be none.

[10 marks]

[**Rubrics**: Logic: 2 mark, Illustration: 3 marks, Pseudocode : 3 marks, Time-complexity : 2 mark]

3. Given a set of symbols $S = \{A, B, C, D, E, F, G, H\}$ and their corresponding frequencies $F = \{7, 3, 8, 2, 6, 5, 12, 10\}$. You are tasked with creating an efficient Huffman encoding scheme. However, instead of a traditional binary tree, you are asked to design a tree with a variable branching factor b (where $b \geq 3$). Compute the total cost of encoding and compare the efficiency of the proposed algorithm with traditional binary approach for the same set of symbols.

[10 marks]

4. Consider the given algorithmS

Algorithm 1 F1(S_1, S_2)

1: **Input:** S_1, S_2
$\{S_1, S_2$ are strings of the same length, and $\_$ indicates an empty string$\}$
2: $n = S_1$.length()
3: $d\_table[1, .., n] = \_$
4: $d\_table[1] = S_1$
5: $d\_table[2] = S_2$
6: **return** $(F2(n, d\_table)$

Algorithm 2 F2(n, d_table)

1: **if** $d\_table[n] \neq \_$ **then**
2:     **return** $d\_table[n]$
3: **end if**
4: $S_x =$ F3(F2($n-2, d\_table$), F2($n-1, d\_table$))
5: $d\_table[n] = S_x$
6: **return** S_x

Algorithm 3 F3(S_1, S_2)

1: $S_x = \_$
2: $m = S_1$.length()
3: **for** $i = 1$ to m **do**
4:     **if** $i \% 2 \neq 0$ **then**
5:         $S_x = S_x + S_1[i]$
    else
        $S_x = S_x + S_2[i]$
    end if
end for
return S_x

(a) For the input ABCD, DEFG, compute the output of the algorithm. [3 marks]

(b) Describe clearly the functionality of the algorithm. [3 marks]

(c) The above algorithm is modified by deleting line 5 in **Algorithm 2**. Compare the complexity of the modified algorithm with the original algorithm. [4 marks]

5. There are a row of n coins of values $\{v_1, v_2, v_3, \ldots, v_n\}$; the objective is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the above list can be picked up. Develop a dynamic programming solution for this optimization problem. For e.g. if the values of $n = 6$ coins given are $\{5, 1, 2, 10, 6, 2\}$, then the coins that need to be picked up to obtain the maximum sum of 17 are $C1 = 5, C4 = 10, C6 = 2$

[10 marks]

[**Rubrics**: Logic: 2 mark, Illustration: 3 marks, Pseudocode : 3 marks, Time-complexity : 2 mark]

$\Longleftrightarrow \Longleftrightarrow \Longleftrightarrow \Longleftrightarrow$