Academic Year 2022- 2023 (Even Sem)

Department of Computer Science and Engineering

Date	10/8/2023	Maximum Marks	50
Course Code	21CS43	Duration	90 Min

Design and Analysis of Algorithms

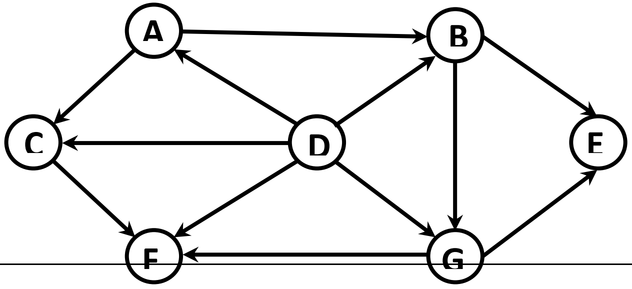
CIE 2 Scheme and Solution

SL	PART A	M	BT	CO
1 a	You are given n coins (where n = 2k, for k is a Natural number) which look exactly alike but one is fake (Lighter than the rest). Write an efficient algorithm to determine the fake coin using a balance (Only balance is given and no weights). Find the complexity of the algorithm considering weighing as the basic operation. Comment on the design technique used. Algorithm = 3 Marks Time complexity = 1 Marks($\log_2 n$) Design Technique = 1 Marks(Decrease and Conquer)	5	3	2
1b	Write an algorithm to perform insertion sort, and sort the given elements using same 89, 45, 68, 90, 29, 34, 17 Algorithm 2 marks	5	2	2

```
ALGORITHM InsertionSort($A[0..n-1]$)
//Sorts a given array by insertion sort
//Input: An array $A[0..n-1]$ of $n$ orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 1$ to $n-1$ do
    $v \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \geq 0$ and $A[j] > v$ do
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$
```

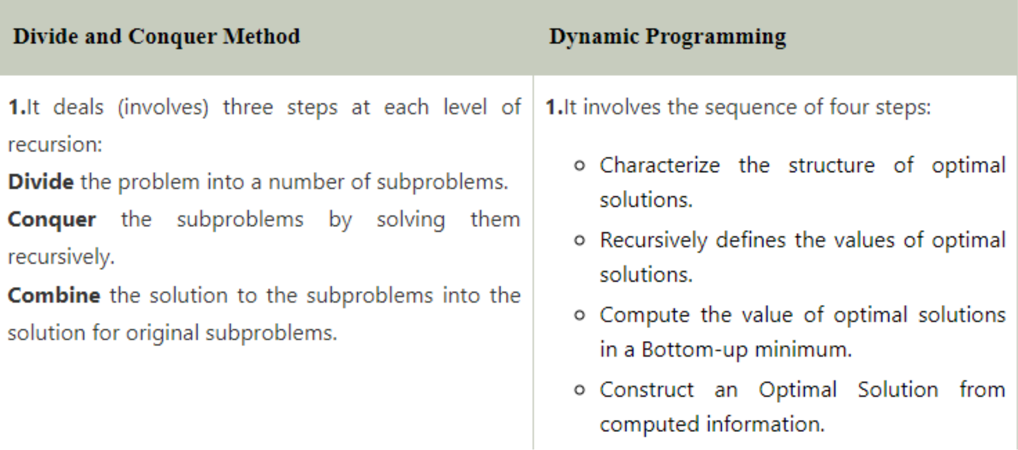
Tracing

```
89 | 45   68   90   29   34   17
45   89 | 68   90   29   34   17
45   68   89 | 90   29   34   17
45   68   89   90 | 29   34   17
29   45   68   89   90 | 34   17
29   34   45   68   89   90 | 17
17   29   34   45   68   89   90
```

SL	PART A	M	BT	CO
2a	Along with DFS algorithm write the topological order of the given graph 	5	3	3

Topological Order : D, A , C, B, G, F, E

2b	Illustrate with example how Dynamic programming is differ from Divide and Conquer	5	2	4



Divide and Conquer Method	Dynamic Programming
1.It deals (involves) three steps at each level of recursion: **Divide** the problem into a number of subproblems. **Conquer** the subproblems by solving them recursively. **Combine** the solution to the subproblems into the solution for original subproblems.	**1.**It involves the sequence of four steps: ○ Characterize the structure of optimal solutions. ○ Recursively defines the values of optimal solutions. ○ Compute the value of optimal solutions in a Bottom-up minimum. ○ Construct an Optimal Solution from computed information.

Example 2 marks

3a	Design Sort by counting algorithm and Sort the elements using same 20,25,21,23,67,22,23,28,26,21	5	3	3

```
ALGORITHM ComparisonCountingSort(A[0..n − 1])
//Sorts an array by comparison counting
//Input: An array A[0..n − 1] of orderable elements
//Output: Array S[0..n − 1] of A's elements sorted in nondecreasing order
for i ← 0 to n − 1 do Count[i] ← 0
for i ← 0 to n − 2 do
for j ← i + 1 to n − 1 do
if A[i] < A[j]
        Count[j] ← Count[j] + 1
else Count[i] ← Count[i] + 1
for i ← 0 to n − 1 do S[Count[i]] ← A[i]
return S
```

Algorithm Tracing 3 marks

3b	Discuss the procedure used in Boyers Moor algorithm. in Apply the same to search the given pattern in the text	5	3	4

Text: BESS_KNEW_ABOUT_BAOBAB.

Pattern: BAOBAB

B	A	O	B	A	B

Bad shift table is

B	A	O	_	Other
2	1	3	6	6

We need to find the value of d1
Where d1=t(c)-k
t(c) is the value of bad shift table
k is the number of matching character

K	Pattern	No Shifts d2
1	BAOBAB	2
2	BAOBAB	5
3	BAOBAB	5

B	E	S	S	_	K	N	E	W	_	A	B	O	U	T	_	B	A	O	B	A	B

B	A	O	B	A	B

4a	Discuss Presort Element Uniqueness algorithm with its time complexity	5	1	3

```
ALGORITHM PresortElementUniqueness(A[0..n − 1])
//Solves the element uniqueness problem by sorting the array first
//Input: An array A[0..n − 1] of orderable elements
//Output: Returns “true” if A has no equal elements, “false” otherwise
sort the array A
for i ←0 to n − 2 do
        if A[i]= A[i + 1] return false
return true
```

T (n) = Tsort(n) + Tscan(n) → (n log n) + (n) = (n log n)

4b	Discuss how the problems are solved in Dynamic Programming? Find the value of $^4C_3$ using dynamic programming	5	2	2

$$C(n,k) = \begin{cases} 1 & k=0 \text{ or } k=n \\ 0 & k>n \\ C(n-1,\ k)+c(n-1,\ k-1) & k<n \ \& \ k!=0 \end{cases}$$

k →

n ↓

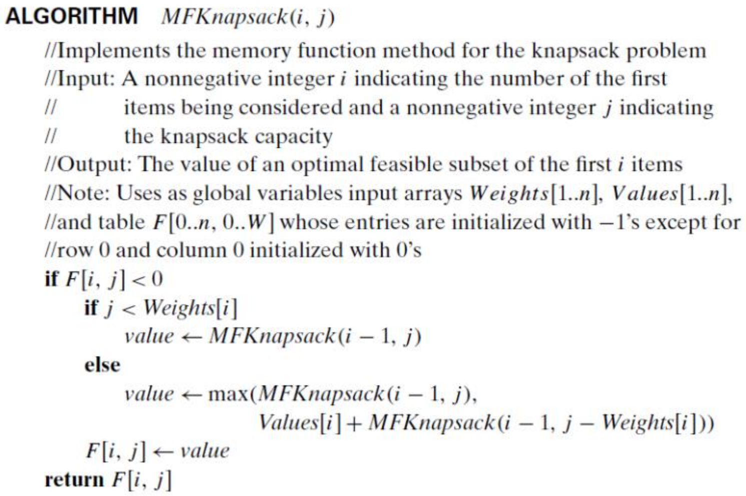
	0	1	2	3
0	1			
1	1	1		
2	1	2	1	
3	1	3	3	1
4	1	4	6	4

5a	Apply Floyd’s algorithm for the weighted matrix given below	4	4	2

$$\begin{matrix} 0 & 2 & \infty & 1 & \infty \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{matrix}$$

D1	D2	D3	D4	Final Matrix

	$\begin{matrix} 0 & 2 & \infty & 1 & \infty \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{matrix}$ $\begin{matrix} 0 & 2 & \infty & 1 & \infty \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & 3 & 0 \end{matrix}$ $\begin{matrix} 0 & 2 & \infty & 1 & \infty \\ 5 & 0 & 3 & 2 & \infty \\ 9 & \infty & 0 & 4 & \infty \\ 8 & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & 3 & 0 \end{matrix}$ $\begin{matrix} 0 & 2 & 6 & 1 & \infty \\ 5 & 0 & 3 & 2 & \infty \\ 9 & \infty & 0 & 4 & \infty \\ 8 & \infty & 2 & 0 & 3 \\ 3 & \infty & 5 & 3 & 0 \end{matrix}$ $\begin{matrix} 0 & 2 & 6 & 1 & \infty \\ 5 & 0 & 3 & 2 & \infty \\ 9 & 11 & 0 & 4 & \infty \\ 8 & 10 & 2 & 0 & 3 \\ 3 & 5 & 9 & 3 & 0 \end{matrix}$			
5b	Write an algorithm to solve 0/1 Knapsack problem using Memory Functions, apply the same to find the maximum profit Algorithm 3 Marks	6	3	4



```
ALGORITHM  MFKnapsack(i, j)
    //Implements the memory function method for the knapsack problem
    //Input: A nonnegative integer i indicating the number of the first
    //        items being considered and a nonnegative integer j indicating
    //        the knapsack capacity
    //Output: The value of an optimal feasible subset of the first i items
    //Note: Uses as global variables input arrays Weights[1..n], Values[1..n],
    //and table F[0..n, 0..W] whose entries are initialized with −1's except for
    //row 0 and column 0 initialized with 0's
    if F[i, j] < 0
        if j < Weights[i]
            value ← MFKnapsack(i − 1, j)
        else
            value ← max(MFKnapsack(i − 1, j),
                            Values[i] + MFKnapsack(i − 1, j − Weights[i]))
        F[i, j] ← value
    return F[i, j]
```

W= 5

w_i	2	1	3	2
v_i	12	10	20	15

Tracing of the input 3 marks
Maximum Profit: 37

Course Outcomes: After completing the course, the students will be able to:-	
CO1	Apply knowledge of computing and mathematics to algorithm analysis and design
CO2	Analyze a problem and identify the computing requirements appropriate for a solution
CO3	Apply mathematical foundations, algorithmic principles, and computer science theory to the modeling, and evaluation of computer-based solutions in a way that demonstrates comprehension of the trade-offs involved in design choices.
CO4	Investigate and apply optimal design, development principles, skills and tools in the construction of software solutions of varying complexity.
CO5	Demonstrate critical, innovative thinking, and display competence in oral, written, and visual communication.
CO6	Exhibits positive group communication exchanges in order to accomplish a common goal and engage in continuing professional development.

Marks Distribution	Particulars	CO1	CO2	CO3	CO4	CO5	CO6	L1	L2	L3	L4
	Max Marks	5	12	15	18	-	-	10	18	22	-