F - 15031

Register No. 2 0 I T L 1 1 5

BTech Degree Examination December 2022

Fourth Semester

Information Technology

20ITT43 – DESIGN AND ANALYSIS OF ALGORITHMS

(Regulations 2020)

Time: Three hours

Maximum: 100 marks

Answer all Questions

Part – A ($10 \times 2 = 20$ marks)

1. Estimate how many times faster it will be to find gcd(31415, 14142) by Euclid's algorithm compared with the algorithm based on checking consecutive integers from min{m, n} down to gcd(m, n) [CO1,K3]
2. List the properties of the asymptotic notations: O, Ω and θ [CO1,K1]
3. Consider the merge sort algorithm for sorting a group of 15 values. Write the number of comparisons needed. [CO2,K4]
4. Write the algorithm to find the height of a binary Tree (T). [CO2,K1]
5. How insertion sort is performed? Give example. [CO3,K1]
6. Demonstrate the obstacles in constructing a minimum spanning tree by an exhaustive search. [CO3,K4]
7. Show an algorithm to make change for 1655 using the greedy strategy. [CO4,K3]
   The coins available are {1000, 500, 100, 50, 20, 10, 5}.
8. Write Warshall's algorithm to compute the transitive closure of a graph whose Adjacency matrix A is given. [CO4,K1]
9. What is the difference between backtracking and branch and bound method? [CO5,K2]
10. Give any two applications for travelling sales person problem. [CO5,K2]

Part – B ($5 \times 16 = 80$ marks)

11. a. i) Check if the following inequalities are correct. (8) [CO1,K3]

$6n^2 - 8n = \theta(n^2)$

$12n^2 + 8 = O(n)$.

ii) List out the steps in mathematical analysis of non recursive algorithm with an example algorithm. (8) [CO1,K2]

(OR)

b. i) Discuss important problem types that you face during Algorithm Analysis. (8) [CO1,K2]

ii) Analyse the time efficiency of recursive algorithms and use recurrence to find the number of moves for Towers of Hanoi problem. (8) [CO1,K3]

12. a. i) Explain the convex hull problem and discuss the brute force approach to solve convex-hull with an example. Derive the time complexity. (8) [CO2,K3]

ii) Disect the working of Strassen's Matrix Multiplication with the help of divide and conquer method. How much multiplication is saved in Strassen's method? (8) [CO2,K4]

(OR)

b. i) Outline quick sort algorithm with example. (8) [CO2,K2]

ii) Find order of growth for the following recurrences (8) [CO2,K3]

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

$T(n) = 4T(n/2) + n^3, T(1) = 1$

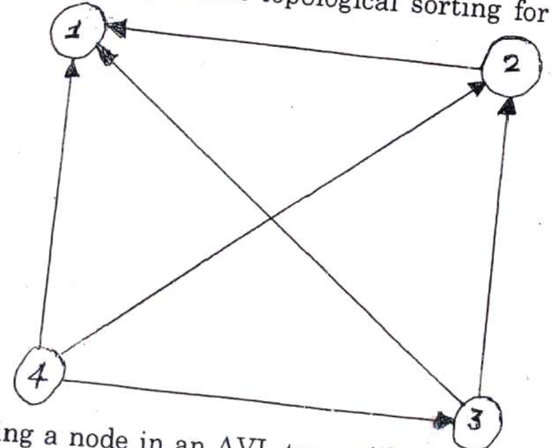
[Use Master Theorem]

13. a. i) Write the algorithm to sort a set of N numbers using insertion sort (8) [CO3,K2]

ii) Design a presorting based algorithm for solving the problem of finding smallest and largest elements in an array of N numbers. (8) [CO3,K3]

(OR)

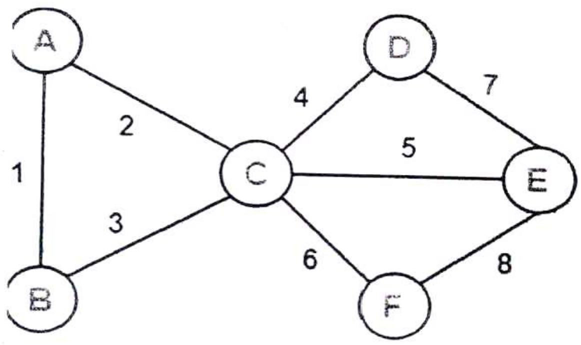
b. i) Apply the source removal method to perform topological sorting for the following diagraph: (8) [CO3,K3]



ii) Illustrate the cases of inserting a node in an AVL tree with suitable examples comment on time complexity involved in inserting a node. (8) [CO3,K4]

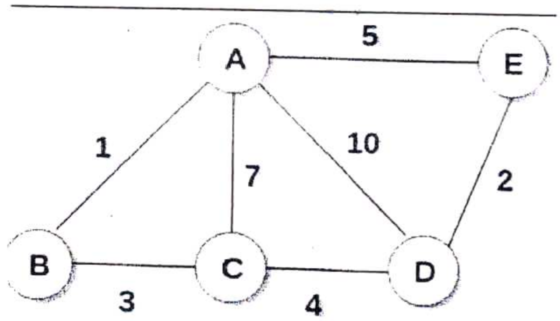
14. a. i) Explain the memory function for the knapsack problem and write the algorithm. (8) [CO4,K2]

ii) Find the minimum spanning tree of the following graph using Prims algorithm. [start from node A]. (8) [CO4,K3]



(OR)

b. i) Find minimum spanning Tree of the following graph using Kruskal's algorithm. (8) [CO4,K3]



ii) Write an algorithm to compute all pair shortest path in a graph. Explain with an example. (8) [CO4,K2]

15. a. i) Solve assignment problem using the following allocation data. Apply branch and Bound method. (8) [CO5,K3]

Job 1	Job 2	Job 3	Job 4	
9	2	7	8	Machine a
6	4	3	7	Machine b
5	8	1	8	Machine c
7	6	9	4	Machine d

ii) Differentiate Deterministic from Non Deterministic algorithms. (8) [CO5,K4]

(OR)

b. i) Demonstrate the stages for solving four queens problem using Backtracking method (8) [CO5,K4]

ii) Show that finding the minimum Hamiltonian cycle problem is NP complete. (8) [CO5,K3]

Bloom's Taxonomy Level	Remembering (K1)	Understanding (K2)	Applying (K3)	Analysing (K4)	Evaluating (K5)	Creating (K6)
Percentage	4	29	47	20	-	-

BTech Degree Examination December 2022
20ITT43 Design and Analysis of Algorithms
(Regulations 2020)

1	The number of divisions made by Euclid's algorithm is 11. The number of divisions made by the consecutive integer checking algorithm on each of its 14142 iterations is either 1 or 2; hence the total number of multiplications is between $1 \cdot 14142$ and $2 \cdot 14142$. Therefore, Euclid's algorithm will be between $1 \cdot 14142/11 \approx 1300$ and $2 \cdot 14142/11 \approx 2600$ times faster.
2	If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.
3	21 comparisons
4	ALGORITHM Height(T) if T = ∅ return −1 else return max{Height(T_{left}), Height(T_{right})} + 1
5	Insertion Sort: It starts with A[1]and ending with A[n – 1], A[i] is inserted in its appropriate place among the first i elements of the array that have been already sorted (but, not in their final positions). This is usually done by scanning the sorted subarray from right to left until the first element smaller than or equal to A[i] is encountered to insert A[i] right after that element. L(1) mark 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 Any example — 1 mark
6	out of syllabus
7	out of syllabus

8	**ALGORITHM** *Warshall*($A[1..n, 1..n]$)
	$R^{(0)} \leftarrow A$
	for $k \leftarrow 1$ **to** n **do**
	    for $i \leftarrow 1$ **to** n **do**
	        for $j \leftarrow 1$ **to** n **do**
	            $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $(R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j])$
	return $R^{(n)}$

9	(see table below)

Backtracking	**Branch and Bound**
Backtracking is normally used to solve decision problems	Branch and bound is used to solve optimization problems
Nodes in the state-space tree are explored in depth-first order in the backtracking method	Nodes in the tree may be explored in depth-first or breadth-first order in branch and bound method
It realizes that it has made a bad choice & undoes the last choice by backing up.	It realizes that it already has a better optimal solution that the pre-solution leads to so it abandons that pre-solution.
The feasibility function is used in backtracking.	Branch-and-Bound involves a bounding function.
The next move from the current state can lead to a bad choice	The next move is always towards a better solution
On successful search of a solution in state-space tree, the search stops	The entire state space tree is searched in order to find the optimal solution

Any four points - 2M

	Backtracking is more efficient.	Branch-and-Bound is less efficient.	
	Applications: N Queen Problem Knapsack Problem Sum of subsets problem Hamiltonian cycle problem, Graph coloring problem	**Applications**: Travelling salesman problem Knapsack problem Job sequencing problem	
10	vehicle routing problems, logistics, planning and scheduling. Any two -(2M)		

11 a (i)	$6n^2-8n=\Theta(n^2)$ $\lim_{n\to\infty} \frac{6n^2-8n}{n^2} = \lim_{n\to\infty} \frac{12n-8}{2n} = \lim_{n\to\infty} \frac{12}{2} = 6$ implies that both has same order of growth (4M) So, inequality is correct $\lim_{n\to\infty} \frac{12n^2+8}{n} = \lim_{n\to\infty} \frac{24n}{1} = \infty$ (4M) implies that $12n^2+8$ has higher order of growth than n. So inequality is incorrect
(ii)	**General Plan for Analyzing the Time Efficiency of Nonrecursive 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 depends only on the size of an input. If it also depends on some additional property, the worst-case, average-case, and, if necessary, best-case efficiencies have to be investigated separately. (4m) 4. Set up a sum expressing the number of times the algorithm's basic operation is executed. 5. Using standard formulas and rules of sum manipulation either find a closed-form formula for the count or, at the very least, establish its order of growth. (any example algorithm) (4M)
11 b (i)	Sorting Searching String processing Graph problems (2M) Combinatorial problems Geometric problems Numerical problems **Sorting** The sorting problem is to rearrange the items of a given list in nondecreasing order based on the piece of information called a key. **Searching** The searching problem deals with finding a given value, called a search key, in a given set **String Processing** One particular sting processing problem is string matching—searching for a given word in a text. **Graph Problems** Graphs can be used for modeling a wide variety of applications, including

explanation -(6M)

	transportation, communication, social and economic networks, project scheduling, and games. **Combinatorial Problems** Travelling salesman problem and the graph coloring problem are examples of combinatorial problems. These are problems that find a combinatorial object—such as a permutation, a combination, or a subset—that satisfy certain constraints such as a maximum value or a minimum cost. **Geometric Problems** Geometric algorithms deal with geometric objects such as points, lines, and polygons. Two classic problems of computational geometry: the closest-pair problem and the convex-hull problem. **Numerical Problems** Problems that involve mathematical objects of continuous nature: solving equations and systems of equations, computing definite integrals, evaluating functions, and so on.
(ii)	**Tower of Hanoi** Basic Operation: Disk move Recurrence Relation: $M(n) = M(n-1) + 1 + M(n-1)$ for $n > 1$. — (2M) Recurrence Relation and Initial condition $M(n) = 2M(n-1) + 1$ for $n > 1$, $M(1) = 1$. $M(n) = 2M(n-1) + 1$ sub. $M(n-1) = 2M(n-2) + 1$ $= 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1$ sub. $M(n-2) = 2M(n-3) + 1$ $= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3M(n-3) + 2^2 + 2 + 1$. — (6M) generally, after i substitutions, we get $M(n) = 2^iM(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2 + 1 = 2^iM(n-i) + 2^i - 1$. Since the initial condition is specified for $n = 1$, which is achieved for $i = n - 1$, we get the following formula for the solution to recurrence (2.3): $M(n) = 2^{n-1}M(n-(n-1)) + 2^{n-1} - 1$ $= 2^{n-1}M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1$.
12 a (i)	DEFINITION The convex hull of a set S of points is the smallest convex set containing S. — (1M) Convex hull is the smallest region covering given set of points. Polygon is called **convex polygon** if the angle between any of its two adjacent edges is always less than 1800. Otherwise, it is called a concave polygon. Complex polygons are self-intersecting polygons. — (4M) Brute Force Approach The brute force method for determining convex hull is to construct a line connecting two points and then verify whether all points are on the same side or not. There are such $n(n-1)/2$ lines with n points, and each line is compared with the remaining $n-2$ points to see if they fall on the same side. the time efficiency of this algorithm? It is in $O(n^3)$: for each of $n(n-1)/2$ pairs of distinct points, we may need to find the sign of $ax + by - c$ for each of the other $n-2$ points. — (3M)
(ii)	Matrix Multiplication using Strassen's Method Strassen suggested a divide and conquer strategy-based matrix multiplication technique that requires fewer multiplications than the traditional method. The multiplication operation is defined as follows using Strassen's method: $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$ — (6M) C11 = S1 + S4 – S5 + S7 C12 = S3 + S5 C21 = S2 + S4 C22 = S1 + S3 – S2 + S6 Where,

S1 = (A11 + A22) * (B11 + B22)
S2 = (A21 + A22) * B11
S3 = A11 * (B12 – B22)
S4 = A22 * (B21 – B11)
S5 = (A11 + A12) * B22
S6 = (A21 – A11) * (B11 + B12)
S7 = (A12 – A22) * (B21 + B22)

running time of Strassen's matrix multiplication algorithm O($n^{2.81}$), which is less than cubic order of traditional approach. – (2M)

12 b (i)

```
void  quicksort(a, low, high)
{
    if (low < high)
        s = partition(a, low, high);
        quicksort(a, low, s - 1);
        quicksort(a, s + 1, high);
}
void  partition(a, low, high)
{
     p = a[low];
     i=low+1;
     j=high;
     while (i>=j)
     {
         while (i<=j && a[j]>=p)
              j=j-1;
        while (i<=j && a[i]<=p)
              i=i+1;
        if (i<=j)
             swap(a[i], a[j])
        else
             break;
     }
     swap(a[l], a[j]);
     return j;
}
```

– (5M)

any example problem – (3M)

(ii)

a. $T(n) = 4T(n/2) + n$. Here, $a = 4$, $b = 2$, and $d = 1$. Since $a > b^d$, $T(n) \in \Theta(n^{\log_2 4}) = \Theta(n^2)$. – (4M)

$T(n) = 4T(n/2) + n^3$. Here, $a = 4$, $b = 2$, and $d = 3$. Since $a < b^d$, $T(n) \in \Theta(n^3)$. – (4M)

13 a (i)

```
ALGORITHM InsertionSort(A[0..n − 1])
        for i ←1 to n − 1 do
            v ←A[i]
            j ←i − 1
        while j ≥ 0 and A[j ]> v do
                A[j + 1]←A[j ]
                j ←j − 1
                A[j + 1]←v
```

– (8M)

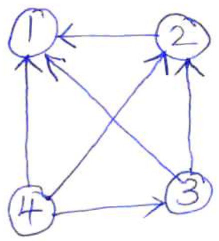
(ii)

```
#include <stdio.h>
int main()
  {
     int i, j, a, n, number[30];
    printf("Enter the value of N \n");
```

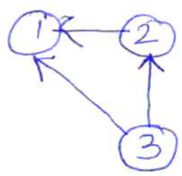
```c
    scanf("%d", &n);
    printf("Enter the numbers \n");
    for (i = 0; i < n; ++i)
      scanf("%d", &number[i]);
    for (i = 0; i < n; ++i)
    {
       for (j = i + 1; j < n; ++j)
       {
          if (number[i] > number[j])
          {
             a =  number[i];
            number[i] = number[j];
            number[j] = a;
          }
       }
    }
    printf("smallest element=%d, largest element=%d\n", number[0],number[n-1]);
}
```

–(8M)

13 b (i)



delete 4



delete 3

—(6M)

delete 2

Topological Sorted order is
{4, 3, 2, 1}. —(2M)

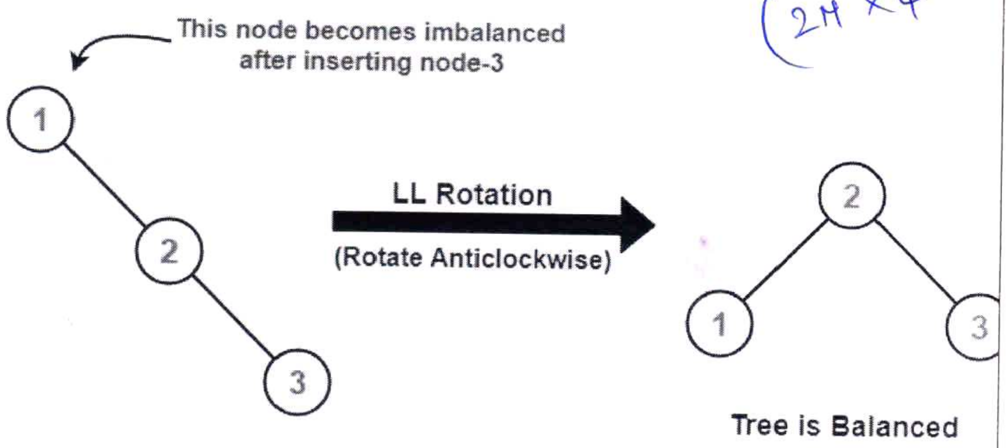
(ii)

1. Left Rotation (LL Rotation)
2. Right Rotation (RR Rotation)
3. Left-Right Rotation (LR Rotation)
4. Right-Left Rotation (RL Rotation)

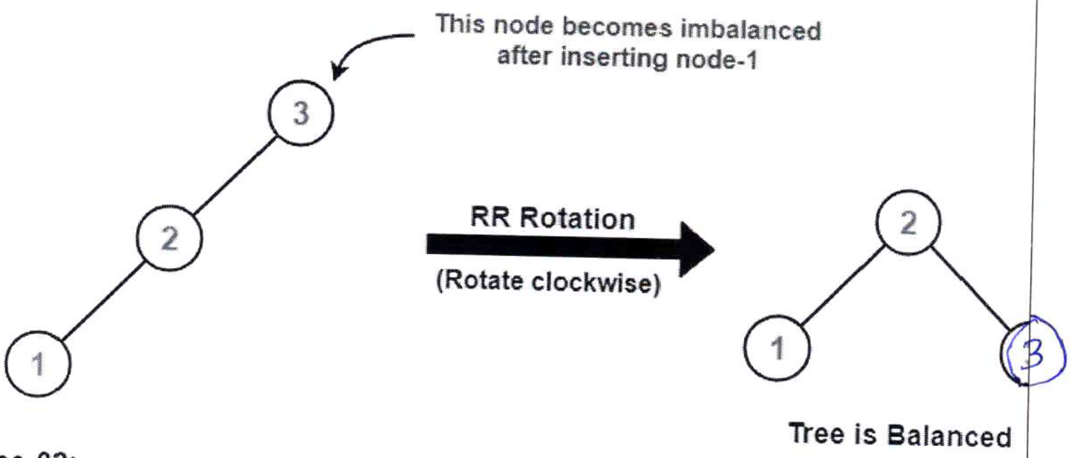
Each type (2M)

(2M × 4 = 8M)

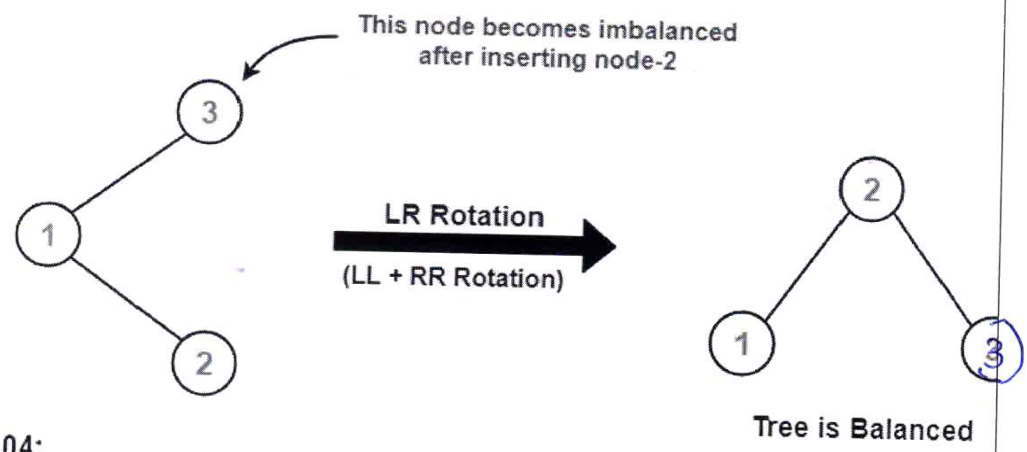
Case-01:



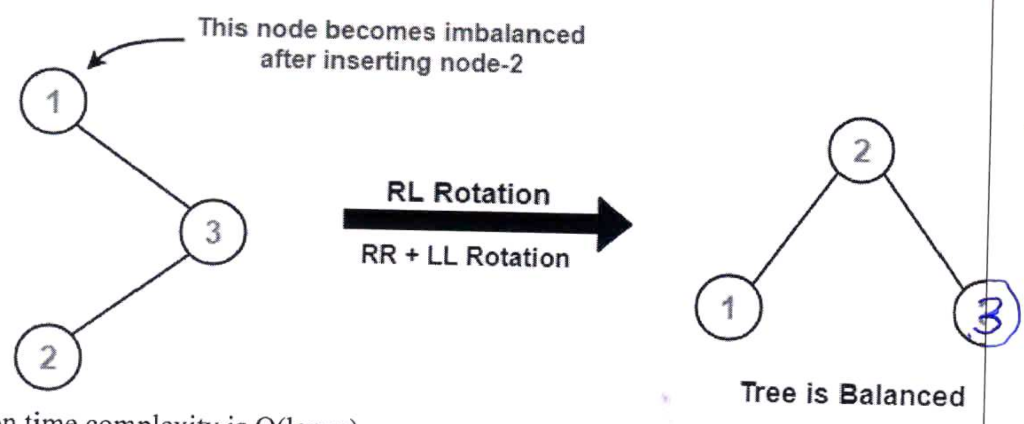
Case-02:



Case-03:



Case-04:



Insertion time complexity is $\Theta(\log_2 n)$

14 a (i) Memory Functions

Dynamic programming solves problems that have a recurrence relation. Using the recurrences directly in a recursive algorithm is a top-down technique. It has the disadvantage that it solves common sub problem multiple times. This leads to poor efficiency, exponential. The dynamic programming technique is bottom-up, and solving all the sub-problems only once. This has the disadvantage that some of the sub-problems may not have been necessary to solve. Illustrate in the table. We would like to have the best of both worlds, i.e. all the necessary sub-problem solved only once. This is possible using memory functions.

The technique uses a top-down approach, recursive algorithm, with table of sub-problem solution. Before determining the solution recursively the algorithm checks if the sub problem has already been solved by checking the table. If the table has a valid value then the algorithm uses the table value else it proceeds with the recursive solution. —(4M)

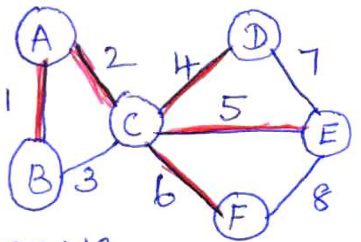
Memory function algorithm for the knapsack problem initializes V[i, j] to -1 except row 0 and column 0, which is initialize to 0.

ALGORITHM MFKnapsack(i, j)
if $F[i, j] < 0$
    if $j < Weights[i]$
        $value \leftarrow MFKnapsack(i-1, j)$
    else
        $value \leftarrow \max(MFKnapsack(i-1, j),$ —(4M)
            $Values[i] + MFKnapsack(i-1, j - Weights[i]))$
    $F[i, j] \leftarrow value$
return $F[i, j]$

(ii)

~~Ordered~~ Tree Vertices	Remaining Vertices
A(–, –)	B(A,1) C(A,2) D(–,∞) E(–,∞) F(–,∞)
B(A,1)	C(A,2) D(–,∞) E(–,∞) F(–,∞)
C(A,2)	D(C,4) C(E,5) F(C,6)
D(C,4)	C(E,5) F(C,6)
C(E,5)	F(C,6)
F(C,6)	–

—(8M)



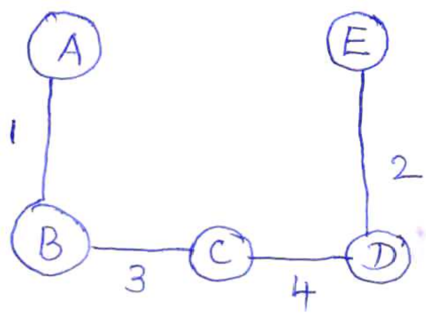
CoST : 18

14 b (i)

ordered edges

AB - 1
ED - 2
BC - 3
CD - 4
AE - 5
AC - 7
AD - 10

MST



CoST : 10

—(8M)

(ii) ALGORITHM Floyd(W[1..n, 1..n])

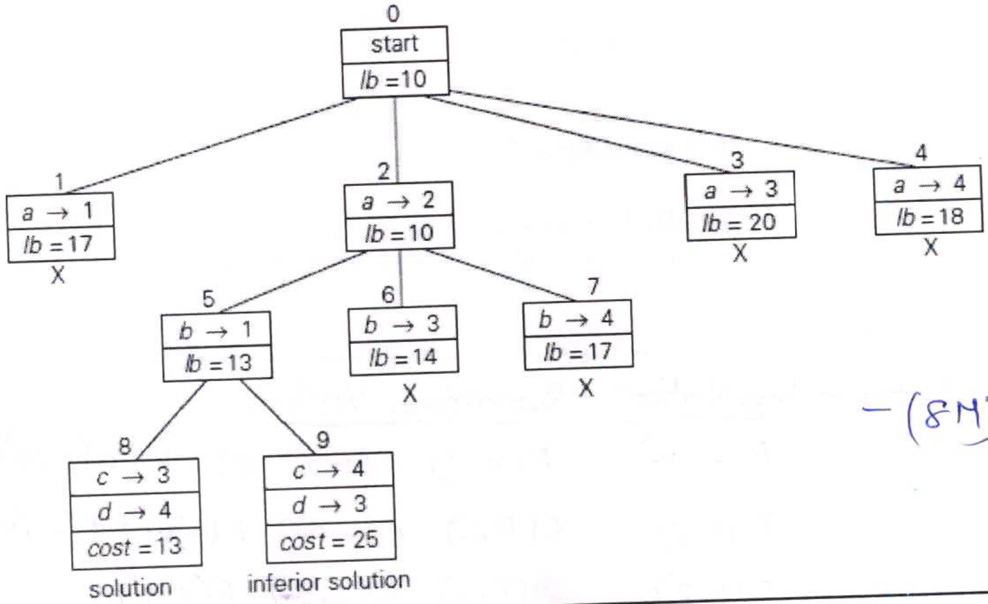
$D \leftarrow W$ //is not necessary if W can be overwritten

```
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```

—(8M)

any example

15 a (i)

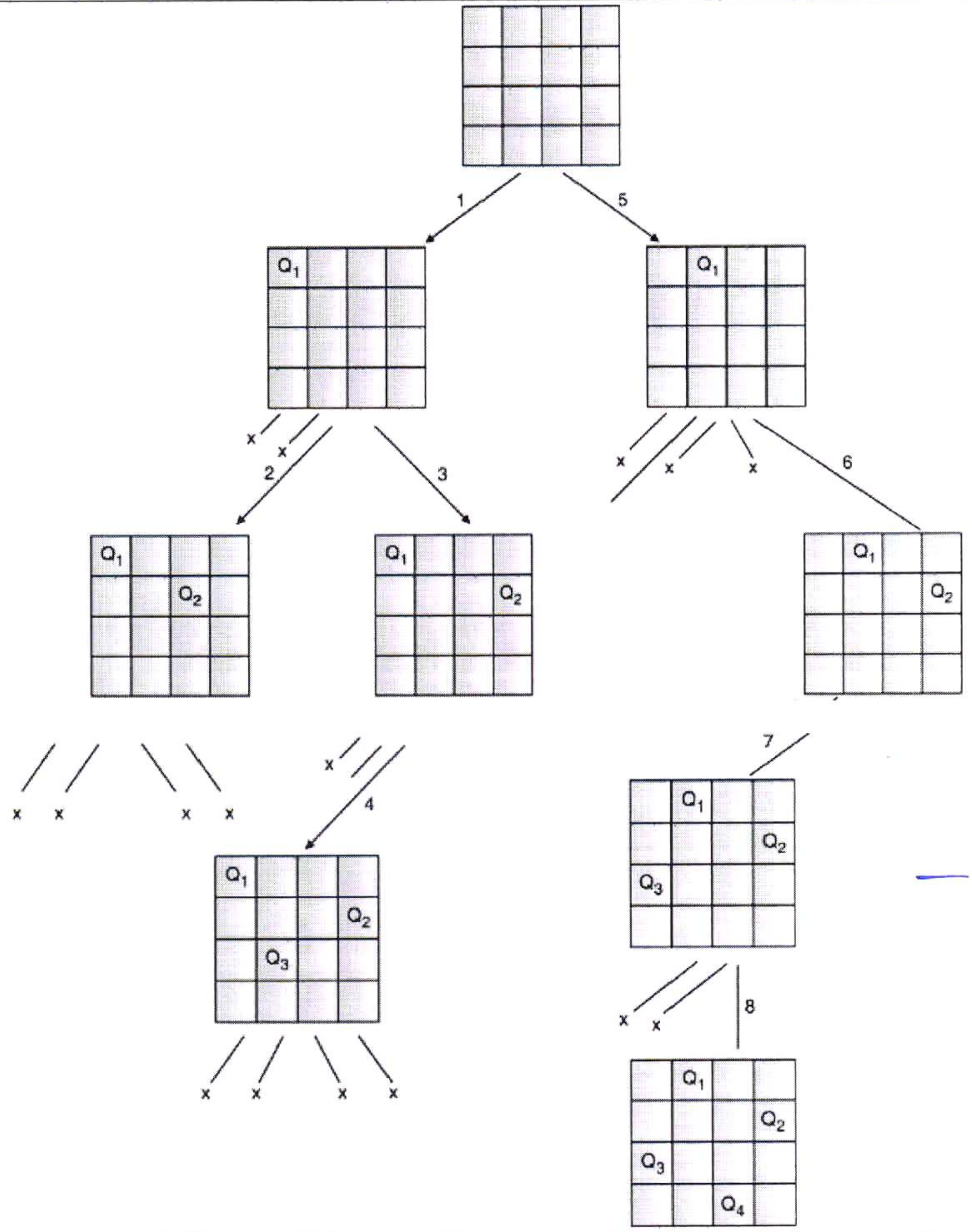


—(8M)

(ii)

Deterministic Algorithm	Non-deterministic Algorithm
For a particular input, the computer will give always the same output.	For a particular input the computer will give different outputs on different execution.
Can solve the problem in polynomial time.	Can't solve the problem in polynomial time.
Can determine the next step of execution.	Cannot determine the next step of execution due to more than one path the algorithm can take.
Operations are uniquely defined.	Operations are not uniquely defined.
Like linear search and binary search	like 0/1 knapsack problem.

—(8M)

15 b (i)	 (8M)
(ii)	**Hamiltonian Cycle is in NP** If any problem is in NP, then, given a *'certificate'*, which is a solution to the problem and an instance of the problem (a graph G and a positive integer k, in this case), we will be able to verify (check whether the solution given is correct or not) the certificate in polynomial time. The certificate is a sequence of vertices forming Hamiltonian Cycle in the graph. We can validate this solution by verifying that all the vertices belong to the graph and each pair of vertices belonging to the solution are adjacent. This can be done in polynomial time, that is **O(V +E)** (8M)