Roll Number:________________________ Number of Pages: 02

Thapar Institute of Engineering and Technology, Patiala

Department of Computer Science and Engineering

MID SEMESTER EXAMINATION

B. E. (Second Year): Sem-II (2021/22)	Course Code: UCS415 Course Name: Design and Analysis of Algorithms
April 09, 2022 Saturday, 11:00 Hrs – 13:00 Hrs	Time: 2 Hours, M. Marks: 35

Name of Faculty: Rajiv Kumar, Maninder Kaur, Shreelekha Pandey, Rajesh Mehta, Mamta Dabra, Yashwant Singh Patel, Vaibhav Pandey, Shruti Aggarwal

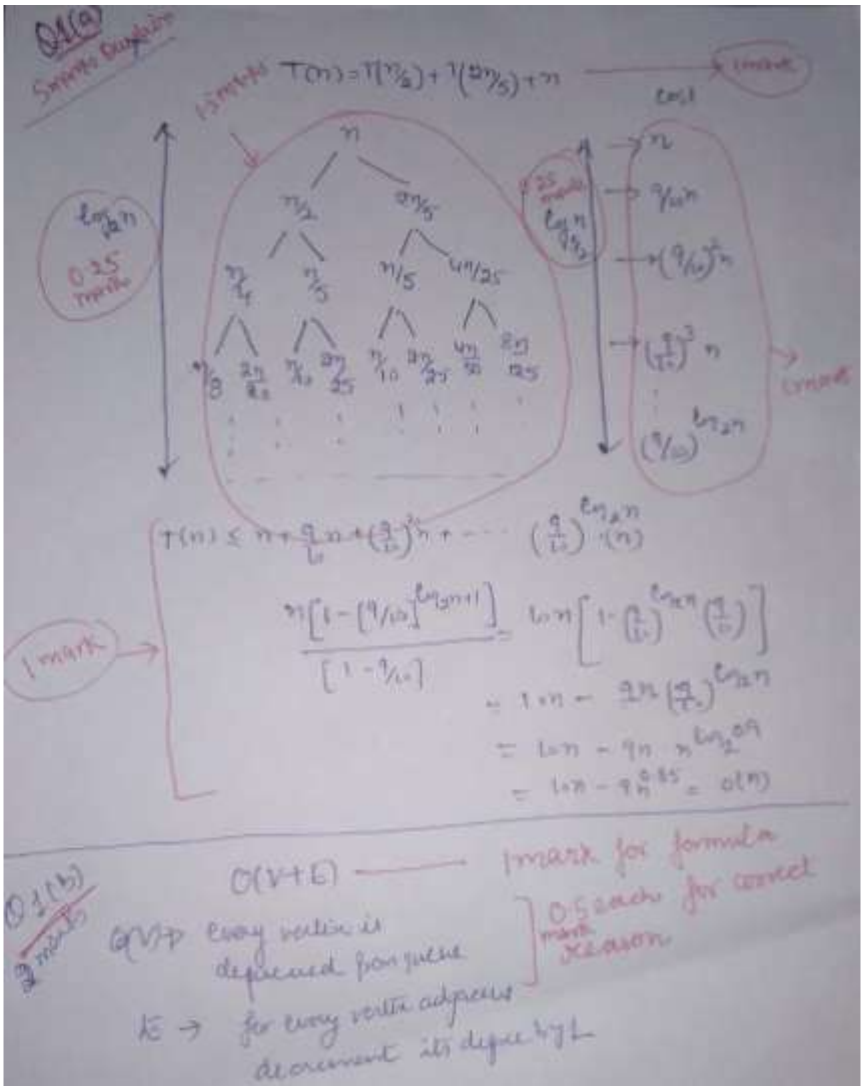
Note: Attempt any 5 questions. Attempt subparts of a question in sequence at one place. Assume missing data, if any, suitably.

Q1. (a) Find the recurrence relation and solve it for the function given in **Fig. 1**. **(5)**

(b) Discuss the time complexity of the Kahn's algorithm for topological sorting? **[DR. MANINDER KAUR]** **(2)**

```
int test(int n)
{ if (n == 0)    return 1;
  for (int i = 1, i <= n; i++)    PRINT n;
  return test(n/2) * test(2*n/5);           }
```

Fig. 1



Q1(a) Sample Solution

1 mark $T(n)=T(n/2)+T(2n/5)+n$ → 1 mark

cost

$\log_2 n$ 0.25 mark

0.25 mark $\log_{10/9} n$

n, $\frac{9}{10}n$, $(\frac{9}{10})^2 n$, $(\frac{9}{10})^3 n$, …, $(\frac{9}{10})^{\log_2 n}$ → 1 mark

$T(n) \le n + \frac{9}{10}n + (\frac{9}{10})^2 n + \cdots (\frac{9}{10})^{\log_2 n}(n)$

$\frac{n[1-(9/10)^{\log_2 n+1}]}{[1-9/10]} = 10n\left[1-(\frac{9}{10})^{\log_2 n}(\frac{9}{10})\right]$

$= 10n - 9n(\frac{9}{10})^{\log_2 n}$

$= 10n - 9n \; n^{\log_2 0.9}$

$= 10n - 9n^{0.85} = O(n)$

1 mark

Q1(b) 2 marks

$O(V+E)$ — 1 mark for formula

$O(V)$ → every vertex is dequeued from queue

E → for every vertex adjacent decrement its degree by 1

0.5 each mark for correct reason

Q2. (a) Execute Hierholzer's algorithm on the graph shown in **Fig. 2** explaining each step of the algorithm clearly. **(3)**

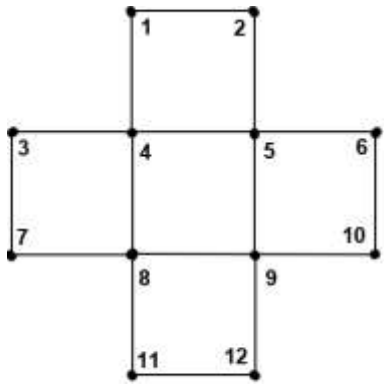


Fig. 2

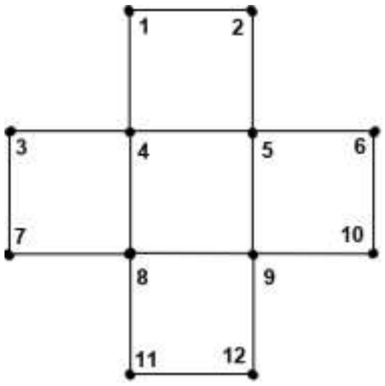
[DR. YASHWANT]

Hierholzer's algorithm:

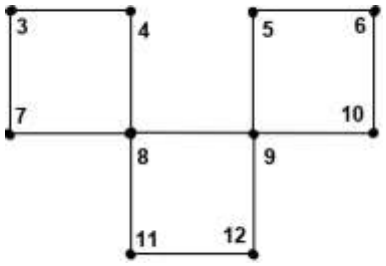
1. Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
2. As long as there exists a vertex u that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from u, following unused edges until returning to u, and join the tour formed in this way to the previous tour.

Execution of Hierholzer's algorithm on the graph in Fig. 2:

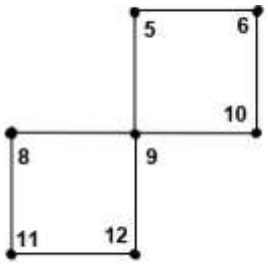
Starting with vertex 1:



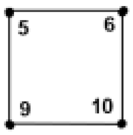
1, 4, 5, 2, 1



1, 4, 3, 7, 8, 4, 5, 2, 1



1, 4, 3, 7, 8, 11, 12, 9, 8, 4, 5, 2, 1



1, 4, 3, 7, 8, 11, 12, 9, 10, 6, 5, 9, 8, 4, 5, 2, 1

Required Trail: 1, 4, 3, 7, 8, 11, 12, 9, 10, 6, 5, 9, 8, 4, 5, 2, 1

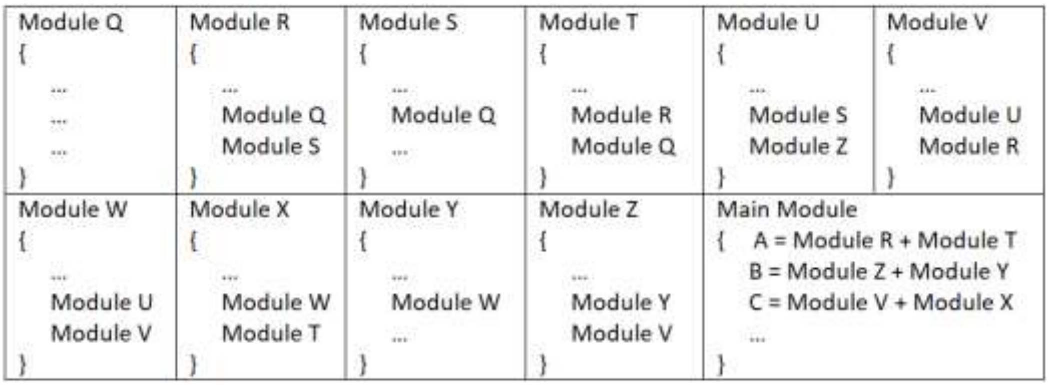
Marking Scheme: (Possible variations in the execution steps are considered)

If all valid trails are given with justification 3 marks (0.6 marks for each step)

If only valid trails are written without any justification 1.5 – 2 Marks

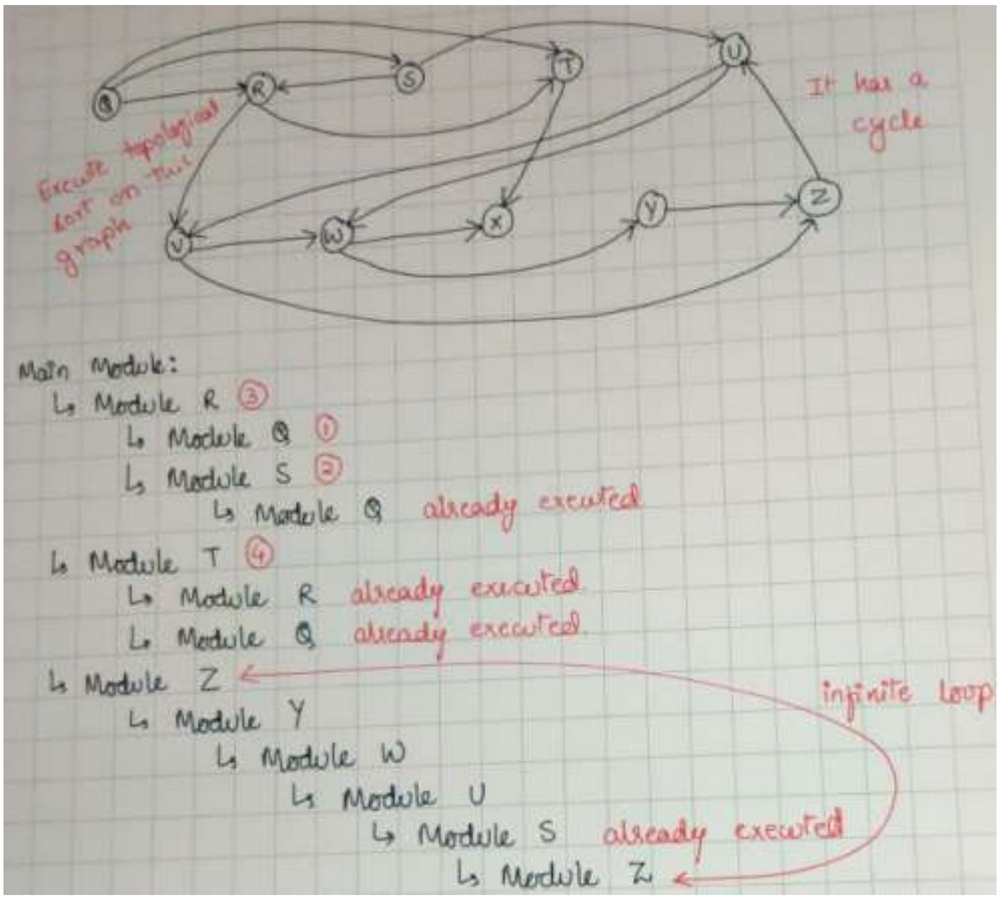
Some just mentioned the Euler circuit (1 mark)

(b) A project is divided into 10 inter-dependent modules as is shown in the template (**Fig. 3**). The system executes a module only once and stores the result that will be used during later function calls. Determine the possible sequence in which modules get executed by the system to complete the project, giving the required explanations. **(4)**



Module Q { … … … }	Module R { … Module Q Module S }	Module S { … Module Q … }	Module T { … Module R Module Q }	Module U { … Module S Module Z }	Module V { … Module U Module R }
Module W { … Module U Module V }	Module X { … Module W Module T }	Module Y { … Module W … }	Module Z { … Module Y Module V }	Main Module { A = Module R + Module T B = Module Z + Module Y C = Module V + Module X … }	

Fig. 3



Main Module:
- ↳ Module R ③
  - ↳ Module Q ①
  - ↳ Module S ②
    - ↳ Module Q already executed
- ↳ Module T ④
  - ↳ Module R already executed
  - ↳ Module Q already executed
- ↳ Module Z
  - ↳ Module Y
    - ↳ Module W
      - ↳ Module U
        - ↳ Module S already executed
        - ↳ Module Z

Marking Scheme:

If only sequence is mentioned without any justification: 1 mark

If only dependency graph is drawn: 1.5 – 2 marks

If only possible sequence & dependency graph is explained without mentioning the cycle:3 marks

If possible sequence & dependency graph is explained with cycle: 4 marks

Q3. Use dynamic programming to fully parenthesize the product of five matrices, i.e. $\boldsymbol{A}\,[2 \times 5], \boldsymbol{B}\,[5 \times 3], \boldsymbol{C}\,[3 \times 6], \boldsymbol{D}\,[6 \times 10], \boldsymbol{E}\,[10 \times 7]$ such that the number of scalar multiplications gets minimized. Show each and every step. **[DR. RAJIV]** **(7)**

Step 1: p[] = {2, 5, 3, 6, 10, 7}, n = p.length – 1 = 6 – 1 = 5.

Step 2: m[1][1] = m[2][2] = m[3][3] = m[4][4] = 0

Step 3: (Chain length = 2)

m[1][2] = m[1][1] + m[2][2] + 2 × 5 × 3 = 30.
m[2][3] = m[2][2] + m[3][3] + 5 × 3 × 6 = 90.
m[3][4] = m[3][3] + m[4][4] + 3 × 6 × 10 = 180.
m[4][5] = m[4][4] + m[5][5] + 6 × 10 × 7 = 420.
s[1][2] = 1, s[2][3] = 2, s[3][4] = 3 and s[4][5] = 4

Step 4: (Chain length = 3)

m[1][3] = min(
m[1][1] + m[2][3] + 2 × 5 × 6 = 0 + 90 + 60 = 150,
m[1][2] + m[3][3] + 2 × 3 × 6 = 30 + 0 + 36 = 66)
m[2][4] = min(
m[2][2] + m[3][4] + 5 × 3 × 10 = 0 + 180 + 150 = 330,
m[2][3] + m[4][4] + 5 × 6 × 10 = 90 + 0 + 300 = 390)
m[3][5] = min(
m[3][3] + m[4][5] + 3 × 6 × 7 = 0 + 420 + 126 = 546,
m[3][4] + m[5][5] + 3 × 10 × 7 = 180 + 0 + 210 = 390)
s[1][3] = 2, s[2][4] = 2 and s[3][5] = 4

Step 5: (Chain length = 4)

m[1][4] = min(
m[1][1] + m[2][4] + 2 × 5 × 10 = 0 + 330 + 100 = 430,
m[1][2] + m[3][4] + 2 × 3 × 10 = 30 + 180 + 60 = 270,
m[1][3] + m[4][4] + 2 × 6 × 10 = 66 + 0 + 120 = 186)
m[2][5] = min(
m[2][2] + m[3][5] + 5 × 3 × 7 = 0 + 390 + 105 = 495,
m[2][3] + m[4][5] + 5 × 6 × 7 = 90 + 420 + 210 = 720,
m[2][4] + m[5][5] + 5 × 10 × 7 = 330 + 0 + 350 = 680)
s[1][4] = 3 and s[2][5] = 2

Step 6: (Chain length = 5)

m[1][5] = min(
m[1][1] + m[2][5] + 2 × 5 × 7 = 0 + 495 + 70 = 565,
m[1][2] + m[3][5] + 2 × 3 × 7 = 30 + 390 + 42 = 462,
m[1][3] + m[4][5] + 2 × 6 × 7 = 66 + 420 + 84 = 570,
m[1][4] + m[5][5] + 2 × 10 × 7 = 186 + 0 + 140 = 326)
s[1][5] = 4

m/s	1	2	3	4	5
1	0	30/1	66/2	186/3	326/4
2		0	90/2	330/2	495/2
3			0	180/3	390/4
4				0	420/4
5					0

```
PrintOptimalParens(s,1,5)                   →    (
    PrintOptimalParens(s,1,4)               →    (
        PrintOptimalParens(s,1,3)           →    (
            PrintOptimalParens(s,1,2)       →    (
                PrintOptimalParens(s,1,1)   →    A
                PrintOptimalParens(s,2,2)   →    B )
            PrintOptimalParens(s,3,3)       →    C )
        PrintOptimalParens(s,4,4)           →    D )
    PrintOptimalParens(s,5,5)               →    E )
```

((((AB)C)D)E)

$$2 \times 5 \times 3 + 2 \times 3 \times 6 + 2 \times 6 \times 10 + 2 \times 10 \times 7 = 30 + 36 + 120 + 140 = 326$$

Marking scheme:

7 marks: Everything correct.
6.5 marks: Minor calculation mistakes.
5 – 6 marks: Logical correct but calculation mistakes.
0 – 4 marks: Otherwise (depending upon what is presented)

Q4. (a) Explain how greedy algorithmic strategy differs from dynamic programming by giving at least two distinctions. [DR. RAJESH MEHTA] **(2)**

① Greedy method generates a single decision sequence where as in DP many decision sequence may be generated.

② Dynamic programming is applicable to the problems having overlapping subproblems where as this characteristic does not hold in Greedy method.

③ Greedy method follows Top down approach where as DP follows bottom up approach.

④ Greedy method is less reliable where as DP is highly reliable.

(b) Discuss the recursive definition/equation employed in dynamic programming solution for 0/1 knapsack problem. Utilize it to maximize the profit for a knapsack having a capacity of **W = 5** using four items. The respective weight and profit values of the four items are **w[]: (2, 3, 4, 5)** and **p[]: (3, 4, 5, 6)**. **(5)**

Q4.(b).

w = 2 3 4 5 W = 5

P = 3 4 5 6

Mathematical Formula

$$T(i,j) = \max\{T(i-1,j),\ p_i + T(i-1,\ j-w_i)\}$$

max value of the selected items if we consider items from 1 to i and having wt. restriction of j.

$(n+1) \times (W+1)$

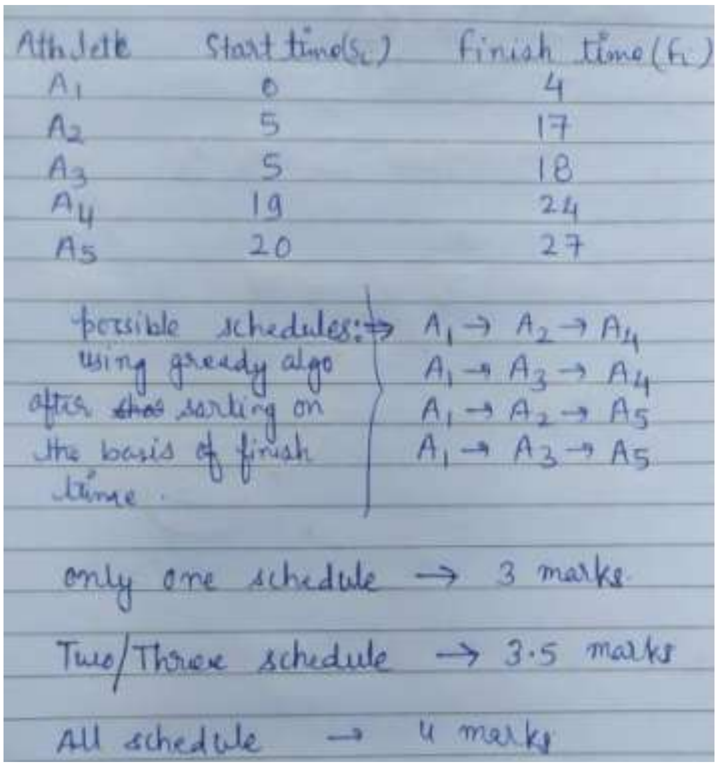
P	W		0	1	2	3	4	5
		0	0	0	0	0	0	0
3	2	1	0	0	3	3	3	3
4	3	2	0	0	3	4	4	7
5	4	3	0	0	3	4	5	7
6	5	4	0	0	3	4	5	$\boxed{7}$

max profit = 7.

item selected = 1, 2

Some Explanation of every entry in the table is also to be mentioned.

Q5. (a) Five athletes are practicing on a single lane track of **500 meters.** The first athlete starts immediately and completes in **4 minutes**. Second and third athletes start after **5 minutes** taking **12 and 13 minutes** respectively. The fourth one starts **2 minutes** after the second athlete completes, and finishes in **5 minutes**. The last athlete starts **20 minutes** late and completes in **7 minutes.** Determine all the possible schedules that allow the maximum number of athletes to practice? **[DR. VAIBHAV]** **(4)**



Athlete	Start time(S_i)	Finish time(f_i)
A_1	0	4
A_2	5	17
A_3	5	18
A_4	19	24
A_5	20	27

possible schedules: using greedy algo after ~~the~~ sorting on the basis of finish time. ⇒

$A_1 \to A_2 \to A_4$

$A_1 \to A_3 \to A_4$

$A_1 \to A_2 \to A_5$

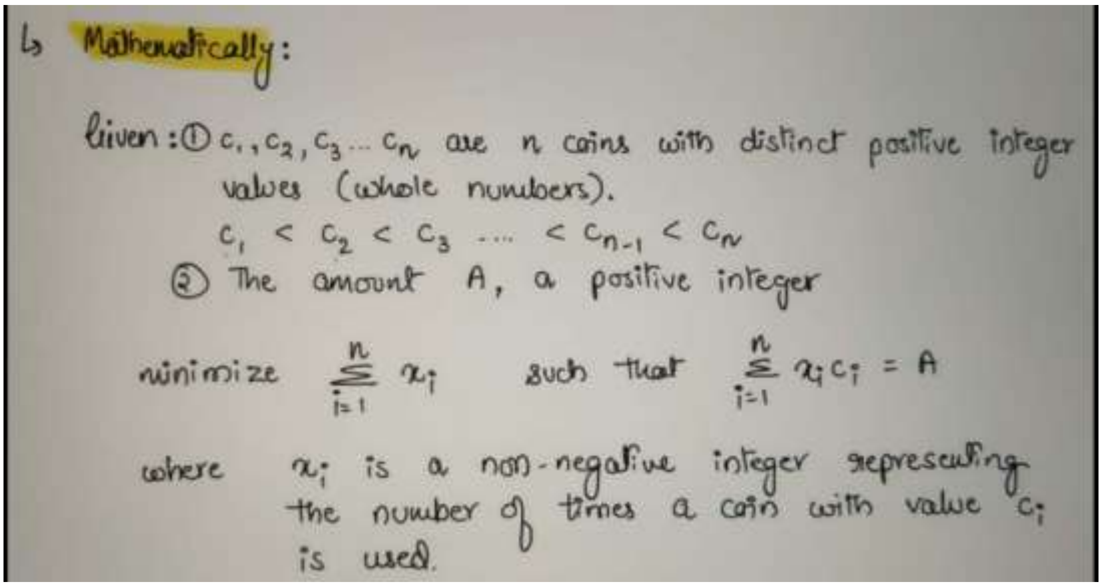
$A_1 \to A_3 \to A_5$

only one schedule → 3 marks

Two/Three schedule → 3.5 marks

All schedule → 4 marks

(b) Mathematically formulate the coin change problem to find the minimum number of coins that add up to the given amount of money. **(3)**



↳ Mathematically:

Given: ① $c_1, c_2, c_3 \dots c_n$ are n coins with distinct positive integer values (whole numbers).

$c_1 < c_2 < c_3 \dots < c_{n-1} < c_n$

② The amount A, a positive integer

minimize $\sum_{i=1}^{n} x_i$ such that $\sum_{i=1}^{n} x_i c_i = A$

where x_i is a non-negative integer representing the number of times a coin with value c_i is used.

The question asks for formulation, not the solution. For recursive solution 0.5 or 01 mark is given.

Q6. (a) Given a sorted array **A** of **n** distinct positive and negative integers, propose an efficient algorithm to find an index **i** such that **1 ≤ i ≤ n** and **A[i] = i**, provided such an index exists. If there are many such indices, then the algorithm can return anyone of them. What is the time complexity of the proposed algorithm? **[DR. MAMTA]** **(5)**

The key observation is that if $A[j] > j$ and $A[i] = i$, then $i < j$. Similarly if $A[j] < j$ and $A[i] = i$, then $i > j$. So if we look at the middle element of the array, then half of the array can be eliminated. The algorithm below (INDEX-SEARCH) is similar to binary search and runs in $\Theta(\log n)$ time. It returns -1 if there is no answer.

```
INDEX-SEARCH($A, b, e$)
  **if** ($e > b$)
  **return** -1
  $m = \lceil \frac{e+b}{2} \rceil$
  **if** $A[m] = m$
     **then return** $m$
  **if** $A[m] > m$
     **then return** INDEX-SEARCH($A, b, m$)
     **else** **return** INDEX-SEARCH($A, m, e$)
```

Marking Scheme:

If a linear search algorithm is written with complexity. It is considered as a trivial solution - 1.5

Searching a particular key using binary search with complexity – 3.5

For the above solution - 5

(b) Can comparison based sorting algorithm be made stable **(Yes/No)**? If **YES**, then explain how the running times of both the versions, stable and unstable, are related? If **NO**, then explain why? **(2)**

Solution: To make a comparison based sorting algorithm stable, we just tag all elements with their original positions in the array. Now, if A[i] = A[j], then we compare i and j, to decide the position of the elements. This increases the running time at a factor of 2 (atmost)

Marking scheme: Yes (.5)

If the answer is just 'Yes' and no valid explanation: 0.5

If a valid reason is given: 1.5 – 2 (depending upon the explanation)

Q7. (a) Discuss possible improvements, with respect to space complexity and time complexity, in the Longest Common Subsequence algorithm using dynamic programming approach. **[DR. SHREELEKHA]** **(2)**

Eliminate the b table altogether. Each c[i, j] entry depends on only three other c table entries: c[i – 1, j – 1], c[i – 1, j], and c[i, j – 1]. Given the value of c[i, j], one can determine in O(1) time which of these three values was used to compute c[i, j], without inspecting table b, and can reconstruct an LCS in O(m + n) time. Although this saves Θ(mn) space, the auxiliary space requirement for computing an LCS does not asymptotically decrease, since Θ(mn) space is still needed for the c table.

Reduce the asymptotic space requirements for LCS-LENGTH. It needs only two rows of table c at a time: the row being computed and the previous row. This improvement works if only the length of an LCS is to be computed; to reconstruct the elements of an LCS, the smaller table does not keep enough information to retrace steps in O(m + n) time.

Marking Scheme:

1 mark for each improvement. 0.5 for mentioning + 0.5 for explanation

(b) Write an efficient algorithm or pseudocode for the given scenario. **(5)**

An **N×N** chessboard with **M** marked squares is there. The row and column numbers of all the marked squares are stored in an array **markedSquares[M][2]**. The task is to visit each of the marked squares exactly once following the rules mentioned below. **(5)**

- Any of the marked squares can be chosen as the start point.
- Each next visit must be to a marked square in the same row or the same column.
- The directions of the moves must alternate, i.e. one cannot visit in the same row or in the same column twice in any two consecutive moves.

NOTE: It is guaranteed that at least one such visit exists.

Marks are given to the correct logic for the following steps:

Construct bipartite graph where Left represents the rows and Right the columns of the marked squares. **2 Marks**

For each marked square at (*r*,*c*), add an edge connecting vertex *r* from Left to vertex *c* from Right. **2 Marks**

An eulerian path in such graph gives the order in which the marked squares can be visited. **1 Mark**

OR

Next visit to a marked square in the same row. **1 Mark**

Next visit to a marked square in the same column. **1 Mark**

No visit in the same row twice. **1 Mark**

No visit in the same column twice. **1 Mark**

Correctness of the remaining steps. **1 Mark**