RV College of Engineering
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world

Academic year 2024-2025 (Even Sem)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Date	01st April 2025	Maximum Marks	50+10
Course Code	**CD343AI**	Duration	90+30
Sem	IV	CIE I	
UG		Faculty: **CS:** HKK, KCG, SB, JS, ASP, **CY:** ARA, **CD:** CRM, **IS:** AKB, SWS, **AIML**: SAK, RRM	

Design and Analysis of Algorithms

(Common to CS/CD/CY/IS/AIML)

Note: - Answer all the questions.

S. No	Part A	M	BT	CO
1.1	**Analyze** the worst-case time complexity of a modified Quicksort when the median (found in O(n) time) is always chosen as the pivot.	01	4	3

1.2	**Derive T(n):**
	`fun(int n) {`
	`if(n==1) return;`
	`else { for (int i=1; i<=n; i++)`
	`for (int j=1; j<=n; j++)`
	`printf ("daa");`
	`fun(n-3);`
	`}`
	`}`
		01	4	2

1.3	Find the total **number of swaps** in the 3rd pass of Bubble Sort for [7, 2, 5, 1, 9].	02	3	1
1.4	Apply and solve using the **Master Theorem:** i) $T(n)=3T(n/4) + n \log n$ ii) $T(n)=0.3T(n/2) + ½$	02	3	1

1.5	**Derive T(n):**
	`for(i=n/2; i<=n; i++)`
	`for(j=1; j<=n; j=2*j)`
	`for(k=1; k<=n; j=k*2)`
	`print ("rvce")`
		02	4	2

1.6	Compare the growth of ½ n (n − 1) and n^2 using **limits**	02	3	2
	Part B			
2. a	Design a recursive algorithm for computing 2^n for any nonnegative integer n that is based on the formula $2^n = 2^{n-1} + 2^{n-1}$. Set up a recurrence relation for the number of additions made by the algorithm and solve it. Draw a tree of recursive calls for this algorithm and count the number of calls made by the algorithm. Is it a good algorithm for solving this problem?	05	6	3

2. b	You have **5 jars of pills**. Each pill weighs 10g, except for one jar where pills weigh 9g. Using a weighing scale, design an algorithm to identify the contaminated jar and analyze its efficiency.	05	6	3
3. a	Write the **selection sort** algorithm and analyze its **time complexity**.	05	4	2
3. b	Derive the time complexity of **Strassen's algorithm** and compare it with the conventional $O(n^3)$ matrix multiplication.	05	5	2
4	Write the **MERGESORT** algorithm/s. Derive T(n) and solve the recurrence using **back substitution method**.	10	4	2
5	Detective Alex is analyzing security footage of a warehouse where crates are supposed to be arranged in ascending order of their serial numbers. However, the footage reveals that some crates are misplaced, creating 'disruptions' in the order. To uncover how disorganized the warehouse is, Alex wants to count how many such disruptions (**inversions**) exist in the sequence. **Design** an efficient algorithm to count these inversions and **analyze** its time complexity.	10	6	3
6	Write **Quick Sort Algorithm** and apply the algorithm for following set of characters: **S = {A, D, S, O, R, P, T, I, V, E}** Consider the **first character as the pivot** in each partitioning step. Show all steps in detail and draw the recursion tree.	10	3	1

****

BT-Blooms Taxonomy, CO-Course Outcomes, M-Marks

Marks Distribution	Particulars		CO1	CO2	CO3	CO4	CO5	CO6	L1	L2	L3	L4	L5	L6
	Quiz/ Test	Max Marks	14	25	21	-	-	-	-	-	16	19	5	20

Course Outcomes: After completing the course, the students will be able to: -	
CO 1	Apply knowledge of computing and mathematics to algorithm analysis and design.
CO 2	Analyze a problem and identify the computing requirements appropriate for a solution.
CO 3	Apply algorithmic principles and computer science theory to the modeling for evaluation of computer-based solutions in a way that demonstrates comprehension of the trade-offs involved in design choices.
CO 4	Investigate and use optimal design techniques, development principles, skills and tools in the construction of software solutions of varying complexity.
CO 5	Demonstrate critical, innovative thinking, and display competence in solving engineering problems.
CO 6	Exhibit effective communication and engage in continuing professional development through experiential learning.

Academic year 2024-2025 (Even Sem)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

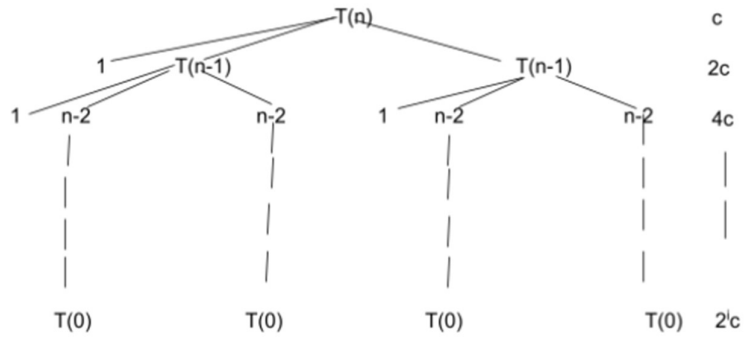
Date	1st April 2025	Maximum Marks	50+10
Course Code	CD343AI	Duration	90+30
Sem	IV	CIE I	
UG	UG	SCHEME AND SOLUTION	

Design and Analysis of Algorithms

(Common to CS/CD/CY/IS/AIML)

S.No	Part A	M
1.1	Since the median is chosen in O(n) time, it ensures a balanced partition of the array into two halves. The recurrence relation for this modified Quicksort is: $T(n)=T(n/2) + O(n)$ Solving by backward substitution: $T(n)=T(n/2)+n$ $=T(n/4)+n/2+n$ $=T(n/8)+n/4+n/2+n$ $=\cdots+O(n)$ Expanding until the base case $T(1)=O(1)$, we get O(n log n) complexity. Final Answer: **O(n log n)** Can also be solved using Master Method	01
1.2	The function contains two nested loops, each running n^2 times, followed by a recursive call fun(n-3). Hence $T(n) = O(n^2) + T(n-3)$ for $n > 1$ $T(1) = 1$ Final Answer: $O(n^3)$ (OPTIONAL)	01
1.3	Bubble Sort Process: Initial Array: [7, 2, 5, 1, 9] 1st Pass: [2, 5, 1, 7, 9] (3 swaps) 2nd Pass: [2, 1, 5, 7, 9] (1 swap) 3rd Pass: [1, 2, 5, 7, 9] (1 swap) Final Answer: **1 swap** in the 3rd pass **Tracing – 1 Mark,** **Final answer – 1 Mark**	02
1.4	i) $T(n) = \Theta(n \log n)$ (Case 3) **1 Mark** ii)Does not apply ($a < 1$) **1 Mark**	02
1.5	Innermost loop enters infinite loop, hence invalid algorithm, cannot derive T(n) **– 2 Marks.** **In the case students corrects the innermost loop iterator update to k=k*2, then** • **Outer loop**: i runs from n/2 to n → **O(n)** • **Middle loop**: j doubles each time (logarithmic) → **O(log n)** • **Inner loop**: k doubles each time → **O(log n)** $T(n) = O(n) \times O(\log n) \times O(\log n) =$ **$O(n \log^2 n)$**	02

1.6	$\lim_{n\to\infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2}\lim_{n\to\infty} \frac{n^2-n}{n^2} = \frac{1}{2}\lim_{n\to\infty}(1-\frac{1}{n}) = \frac{1}{2}.$ Since the limit is equal to a positive constant, the functions have the same order of growth or, symbolically, $\frac{1}{2}n(n-1) \in \Theta(n^2)$.	02
	Part B	

2. a	**Solution**
	to compute $\mathbf{2^n}$ for any nonnegative integer n that is based on the formula $\mathbf{2^n = 2^{n-1} + 2^{n-1}}$
	
	Recursive Algorithm: **1 mark**
	`int compute (int n) {`
	`  if (n == 0)`
	`  return 1;`
	`  return compute (n - 1) + compute (n - 1);`
	`}`
	
	Recurrence Relation + Solution: **(1+ 1) = 2 marks**
	Let T(n) be the number of additions, the $T(n)=2T(n-1) + 1$, $T(0) = 0$
	
	Using backward substitution method:
	$T(n)=2(2T(n-2) +1)+1$
	$=4T(n-2) + 2 + 1$
	
	$T(n)=8T(n-3)+4+2+1$
	…
	$T(n)=2^nT(0) + (2^n-1)$
	Since $T(0) = 0$,
	$T(n)=2^n-1 = O(2^n)$. Thus, the algorithm performs exponential additions.
	
	Recursive Call Tree: **1 Mark**
	 
	
	Is it a good algorithm? 1 Mark
	No, it is highly inefficient (**O(2^n)**). Instead, an **iterative approach** or using bitwise shifting (1<<n) achieves **O(1)** time complexity.
		05

2. b	**Solution:** Optimal Algorithm: **4 marks** 1. Label the jars 1 to 5. 2. Take: 1 pill from jar 1 2 pills from jar 2 3 pills from jar 3 4 pills from jar 4 5 pills from jar 5 3. Weigh the total pills (**say W)**. 4. The expected weight (if all were 10g) is: =1(10) + 2(10) + 3(10) + 4(10) + 5(10) = 150g The actual weight will be: X = **150** − W where X (the weight difference) identifies the contaminated jar. Efficiency Analysis: **1 mark** Single weighing operation → **O(1)** Best algorithm for the problem as it minimizes scale usage to just one weighing.	05

3. a	Algorithm: **2 marks**
	
	ALGORITHM *SelectionSort*$(A[0..n-1])$
	//Sorts a given array by selection sort
	//Input: An array $A[0..n-1]$ of orderable elements
	//Output: Array $A[0..n-1]$ sorted in nondecreasing order
	for $i \leftarrow 0$ **to** $n-2$ **do**
	$min \leftarrow i$
	for $j \leftarrow i+1$ **to** $n-1$ **do**
	if $A[j] < A[min]$ $min \leftarrow j$
	swap $A[i]$ and $A[min]$
	
	Time complexity analysis as per the mathematical framework : 3 Marks
	Best = worst = average case = $O(n^2)$ or $\Theta(n^2)$
	
	Time complexity: $O(n^2)$
	Space complexity: O(1)
	In-place/out-of-place? In-place
	Stability? Unstable
	Comparison Sort? Comparison
		05

3. b	**Strassen's algorithm: 2 marks** $$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$ $$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$ where: $$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$$ $$m_2 = (a_{10} + a_{11}) * b_{00}$$ $$m_3 = a_{00} * (b_{01} - b_{11})$$ $$m_4 = a_{11} * (b_{10} - b_{00})$$ $$m_5 = (a_{00} + a_{01}) * b_{11}$$ $$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$$ $$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$$ Time complexity : **2 marks** • Input size – N (matrix order) • Basic operation – Multiplication • Number of multiplications M (n) will be: **M(n) = 7M(n/2) for n > 1,** **M(1) = 1** Solving it by backward substitutions for $n = 2^k$: $M(2^k) = 7M(2^{k-1})$ $= 7[7M(2^{k-2})] = 7^2M(2^{k-2})$ = … $= 7^iM(2^{k-i})$ = … $= 7^kM(2^{k-k}) = 7^k$ Since $k = \log_2 n$ $$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$$ Comparision: **1 mark** Strassen's algorithm ($O(n^{2.81})$) is asymptotically faster than conventional matrix multiplication ($O(n^3)$), but its high constant factors limit practicality to large matrices.	05

4	• Merge sort algorithm: **2 marks** • Merge algorithm: **3 marks** • Deriving recurrence relation + solution**: (2 + 3) = 5 marks**	10

ALGORITHM *Mergesort*$(A[0..n-1])$
//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
Mergesort$(B[0..\lfloor n/2 \rfloor - 1])$
Mergesort$(C[0..\lceil n/2 \rceil - 1])$
Merge(B, C, A)

ALGORITHM *Merge*$(B[0..p-1], C[0..q-1], A[0..p+q-1])$
//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0;\ j \leftarrow 0;\ k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
if $B[i] \le C[j]$
$A[k] \leftarrow B[i];\ i \leftarrow i+1$
else $A[k] \leftarrow C[j];\ j \leftarrow j+1$
$k \leftarrow k+1$
if $i = p$
copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

1. input's size: n – number of elements to be sorted. (Assuming for simplicity that n is a power of 2)
2. basic operation: comparison
3. No worst, average, and best cases
4. Let T(n) = number of times the basic operation is executed.
   $T(n) = 2T(n/2) + T_{divide\_merge}(n)$ for $n > 1$, $T(1) = 0$

Solve using back substitution:

$$T(n) = 2T(n/2) + O(n)$$

$$= 4T(n/4) + 2O(n/2) + O(n)$$

$$= 8T(n/8) + 4O(n/4) + 2O(n/2) + O(n)$$

General Form:

$$T(n) = 2^k T(n/2^k) + kO(n)$$

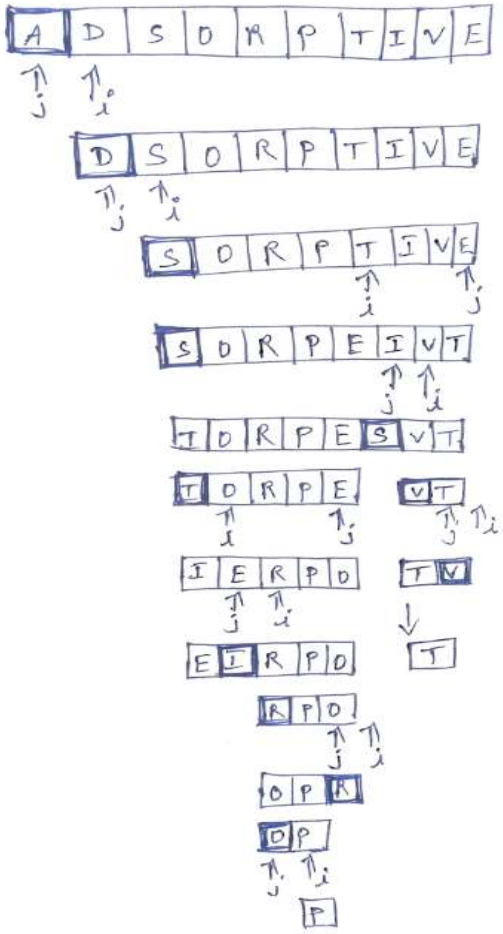
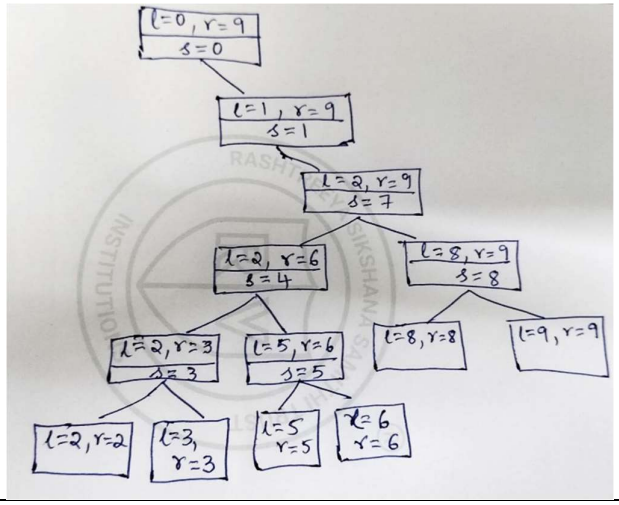
Stopping Condition:

When $n/2^k = 1$, then $k = \log_2 n$.

Final Complexity:

$$T(n) = O(n \log n)$$

5	**Counting Inversions Algorithm (Using Modified Merge Sort): (3+4) = 7 marks**
	Time analysis: 3 marks
	
	`ALGORITHM CountInversions(A, left, right)`
	`  IF left >= right THEN`
	`  RETURN 0`
	`  END IF`
	`  mid ← (left + right) / 2`
	`  invLeft ← CountInversions(A, left, mid)`
	`  invRight ← CountInversions(A, mid + 1, right)`
	`  invMerge ← MergeAndCount(A, left, mid, right)`
	`  RETURN (invLeft + invRight + invMerge)`
	
	`ALGORITHM MergeAndCount(A, left, mid, right)`
	`  i ← left, j ← mid + 1, k ← 0`
	`  invCount ← 0`
	`  CREATE Temp[right - left + 1]`
	
	`  WHILE i ≤ mid AND j ≤ right DO`
	`  IF A[i] ≤ A[j] THEN`
	`  Temp[k] ← A[i]`
	`  i ← i + 1`
	`  ELSE`
	`  Temp[k] ← A[j]`
	`  invCount ← invCount + (mid - i + 1)`
	`  j ← j + 1`
	`  END IF`
	`  k ← k + 1`
	`  END WHILE`
	
	`  COPY remaining elements from A[left..mid] to Temp[]`
	`  COPY remaining elements from A[mid+1..right] to Temp[]`
	`  COPY Temp[] back to A[left..right]`
	
	`  RETURN invCount`
	
	Time Complexity Analysis: This approach modifies Merge Sort to count inversions while sorting. The recurrence is: $T(n)=2T(n/2)+O(n)$ which solves to: $O(n \log n)$
	
	Thus, the algorithm efficiently counts inversions in $O(n\log n)$, much better than the naive $O(n2)$ approach.
	
	Note: Marks will only be awarded for efficient strategies; inefficient approaches may not be considered
		10

5	Quicksort algorithm/s: **(2 + 3) = 5 marks** Tracing**: 3 marks** Recursion tree**: 2 marks**  Recursion call tree 	10

RV College of Engineering
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Academic year 2024-2025 (Even Sem)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

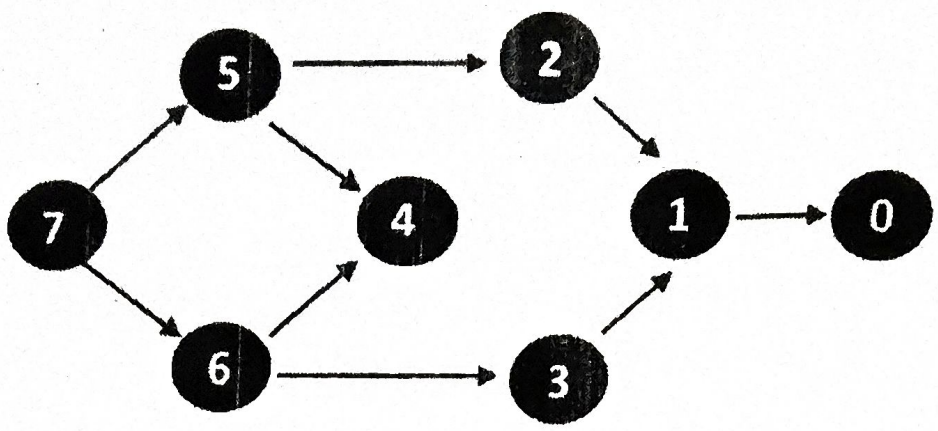
Date: May 2025	CIE II	Maximum Marks: 50 + 10
Course Code: CD343AI	Sem: 4th UG	Duration: 120 Minutes

Design and Analysis of Algorithms
(Common to CS/CD/CY/IS/AIML)

Note: - Answer all the questions.

S.No	Part A	M	BT	CO
1.1	_____ and _____ are examples of sorting algorithms that can achieve linear time complexity under certain conditions.	02	1	1
1.2	Consider the pattern 00001 and a binary text consisting of 1000 zeros (000...0). Using the Boyer-Moore algorithm, analyze and calculate the number of character comparisons made during the pattern search.	02	3	2
1.3	Assuming that the set of possible list values is {a, b, c, d}, sort the following list in alphabetical order by the distribution-counting algorithm: b, c, d, c, b, a, a, b.	02	3	3
1.4	How can you determine the number of paths of length two between any two vertices in a graph using its adjacency matrix? Provide an example to illustrate this process.	02	3	3
1.5	Consider the problem of searching for genes in DNA sequences using Horspool's algorithm. A DNA sequence is represented by a text on the alphabet {A, C, G, T}, and the gene or gene segment is the pattern. Construct the shift table for the following gene segment of a chromosome TCCTATTCTT	01	3	3
1.6	Justify why is Binary search not considered a good example of a pre-sorting technique.	01	3	3
	Part B			
1	You are given the results of a completed round-robin tournament in which n teams played each other exactly once. Each game ended with a victory for one of the teams, and these results are represented in the form of a directed graph (digraph a) — where an edge from team U to team V indicates that team U defeated team V. Design and write an algorithm using Depth First Search(DFS) to list the teams in a sequence such that no team in the list has lost to the team immediately after it. Analyze the time efficiency of your algorithm. Then, apply your algorithm to digraph (a) and show the step-by-step solution.	10	3	4

Academic year 2024-2025 (Even Sem)

	 Fig. a			
2 a	Design a decrease-by-one algorithm for generating the power set of a set of n elements. (The power set of a set S is the set of all the subsets of S, including the empty set and S itself.)	5	3	
2 b	Design a presorting-based algorithm for computing the mode in a given list of numbers and determine its efficiency.	5	2	
3 a	Write the algorithm for Sorting by Comparison Counting and apply it to the following list of elements: 62, 31, 84, 96, 19, 47. Show the complete trace of the sorting process, including the count array and the resulting sorted list.	05	3	
3 b	Consider the problem of finding the smallest and largest elements in an array of n numbers. Design a presorting-based algorithm for solving this problem and determine its efficiency class.	05	2	
4	Write the Heap Sort algorithm along with the heapify procedure used to maintain the max-heap property. Construct a Max Heap using bottom-up heap construction for the following list of elements 2, 9, 7, 6, 5, 8. Show the complete trace of the heap construction pictorially at each step. Briefly discuss the time complexity of Heap Sort algorithm.	10	3	
5	Write the pseudocode for Horspool's algorithm to search for the pattern in the text. Summarize the efficiency of the algorithm in comparison with brute force approach and its Boyer-Moore algorithm. Demonstrate Brute force, Horspool and its Boyer-Moore approach for following example. Text: ababcababcabc Pattern: abc	10	3	2

BT-Blooms Taxonomy, CO-Course Outcomes, M-Marks

Marks Distribution	Particulars		CO1	CO2	CO3	CO4	CO5	L1	L2	L3	L4	L5	L6
	Quiz/ Test	Max Marks	02	12	21	25		2	10	48			

Course Outcomes: After completing the course, the students will be able to: -

CO 1	Apply knowledge of computing and mathematics to algorithm analysis and design.
CO 2	Analyze a problem and identify the computing requirements appropriate for a solution.
CO 3	Apply algorithmic principles and computer science theory to the modeling for evaluation of computer-based solutions in a way that demonstrates comprehension of the trade-offs involved in design choices.
CO 4	Investigate and use optimal design techniques, development principles, skills and tools in the construction of software solutions of varying complexity.
CO 5	Demonstrate critical, innovative thinking, and display competence in solving engineering problems.
CO 6	Exhibit effective communication and engage in continuing professional development through experiential learning.

RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world

Academic year 2024-2025 (Even Sem)

DEPARTMENT OF **COMPUTER SCIENCE & ENGINEERING**

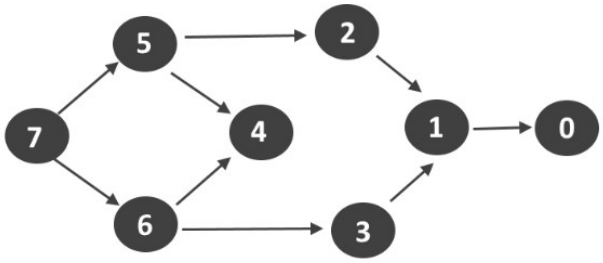
Date: May 2025	**CIE II**	Maximum Marks: 50 + 10
Course Code: CD343AI	Sem: 4th UG	Duration: 120 Minutes

Design and Analysis of Algorithms

(Common to CS/CD/CY/IS/AIML)

SOLUTION and SCHEME

S.No	Part A	M	BT	CO
1.1	Insertion Sort , Distribution Counting Sort	02	1	1
1.2	a. For the pattern 00001, the shift tables will be filled as follows: the bad-symbol table c | 0 | 1 $t_1(c)$ | 1 | 5 . the good-suffix table k | the pattern | d_2 1 | 0000**1** | 5 2 | 000**01** | 5 3 | 00**001** | 5 4 | 0**0001** | 5 On each of its trials, the algorithm will make one unsuccessful comparison and then shift the pattern by $d_1 = \max\{t_1(0) - 0, 1\} = 1$ position to the right without consulting the good-suffix table: 0 0 0 0 0 0 — 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 etc. 0 0 0 0 1 The total number of character comparisons will be $C = 1 \cdot 996 = 996$.	02	3	2
1.3	Input: A: b, c, d, c, b, a, a, b Frequencies: a 2, b 3, c 2, d 1 — Distribution values: a 2, b 5, c 7, d 8 $D[a..d]$ / $S[0..7]$: $A[7] = b$: 2 **5** 7 8 — $S[5] = b$ $A[6] = a$: **2** 4 7 8 — $S[1] = a$ $A[5] = a$: **1** 4 7 8 — $S[0] = a$ $A[4] = b$: 0 **4** 7 8 — $S[4] = b$ $A[3] = c$: 0 3 **7** 8 — $S[6] = c$ $A[2] = d$: 0 3 6 **8** — $S[7] = d$ $A[1] = c$: 0 3 **6** 7 — $S[3] = c$ $A[0] = b$: 0 **3** 5 7 — $S[2] = b$	02	3	3
1.4	Multiply adjacency matrix by itself ----- 1M Example ----- 1M	02	3	3
1.5	A-5, C-2, G-10, T-1	01	3	3
1.6	The brute-force solution/sequential search takes n comparisons in the worst case. Whereas, Binary search, which requires pre-sorting takes (n log n) comparisons in the worst case.	01	3	3

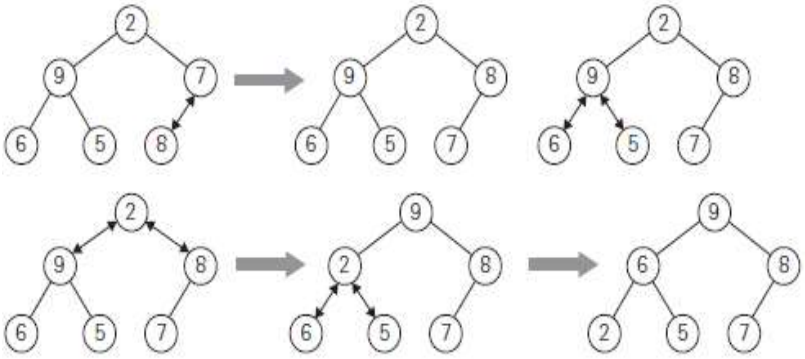
	Part B			
1	 Fig. a **Algorithm(4M)** TOPOSORT(G): Input: A directed acyclic graph G = (V, E) Output: A list representing topological ordering of vertices 1. Create an empty stack `S` to store the topological order 2. Mark all vertices in V as unvisited 3. For each vertex v in V: If v is not visited: DFS(v) 4. Return the contents of stack S (reversed) Procedure DFS(v): 1. Mark v as visited 2. For each neighbor u of v: If u is not visited: DFS(u) 3. Push v onto stack S Time complexit :O(V+E) for adjacency list O(V^2) for adjacency matrix **(2M)** **step-by-step solution (4M)** 7 (1, 8) 5(2, 5,) 2 (3, 3) 1 (4, 2) 0 (5, 1) 4 (6, 4) 6(7, 7) 3(8, 6) **Reverse the pop order will give the final team sequence: 7, 6, 3, 5, 4, 2, 1, 0**	10	3	4

2 a	Algorithm PowerSet(S)
	Input: A set S
	Output: All subsets of S (the power set)
	
	1. If S is empty:
	2. Return { {} } // Only the empty set
	3. Remove one element x from S
	4. SubsetsWithoutX = PowerSet(S) // Recursively find subsets of smaller set
	5. SubsetsWithX = add x to each subset in SubsetsWithoutX
	6. Return SubsetsWithoutX ∪ SubsetsWithX
		5	3	4

2 b	**ALGORITHM** *PresortMode*$(A[0..n-1])$
	//Computes the mode of an array by sorting it first
	//Input: An array $A[0..n-1]$ of orderable elements
	//Output: The array's mode
	sort the array A
	$i \leftarrow 0$ //current run begins at position i
	$modefrequency \leftarrow 0$ //highest frequency seen so far
	while $i \leq n-1$ **do**
	$runlength \leftarrow 1$; $runvalue \leftarrow A[i]$
	while $i + runlength \leq n-1$ **and** $A[i+runlength] = runvalue$
	$runlength \leftarrow runlength + 1$
	if $runlength > modefrequency$
	$modefrequency \leftarrow runlength$; $modevalue \leftarrow runvalue$
	$i \leftarrow i + runlength$
	return $modevalue$
		5	3	3

3 a	Show the complete trace of the sorting process, including the count array and the resulting sorted list.
	Write the Algorithm for sorting by comparison counting. ---------- 3M
	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 $i \leftarrow 0$ **to** $n-1$ **do** $Count[i] \leftarrow 0$
	for $i \leftarrow 0$ **to** $n-2$ **do**
	for $j \leftarrow i+1$ **to** $n-1$ **do**
	if $A[i] < A[j]$
	$Count[j] \leftarrow Count[j] + 1$
	else $Count[i] \leftarrow Count[i] + 1$
	for $i \leftarrow 0$ **to** $n-1$ **do** $S[Count[i]] \leftarrow A[i]$
	return S
		05	3	4

	Apply the same for the list : 62, 31, 84, 96, 19, 47 and Show the complete trace. ---------- 3M Array *A*[0..5] | 62 | 31 | 84 | 96 | 19 | 47 | Initially — *Count* [] | 0 | 0 | 0 | 0 | 0 | 0 | After pass *i* = 0 — *Count* [] | 3 | 0 | 1 | 1 | 0 | 0 | After pass *i* = 1 — *Count* [] | | 1 | 2 | 2 | 0 | 1 | After pass *i* = 2 — *Count* [] | | | 4 | 3 | 0 | 1 | After pass *i* = 3 — *Count* [] | | | | 5 | 0 | 1 | After pass *i* = 4 — *Count* [] | | | | | 0 | 2 | Final state — *Count* [] | 3 | 1 | 4 | 5 | 0 | 2 | Array *S*[0..5] | 19 | 31 | 47 | 62 | 84 | 96 |			
3 b	1. Sort the array A using an efficient comparison-based sorting algorithm (e.g., Merge Sort, Heap Sort, or Quick Sort) 2. Return A[0] as the smallest element 3. Return A[n−1] as the largest element • Sorting takes **O(n log n)** time using efficient algorithms (like Merge Sort or Heap Sort). • Accessing the first and last elements takes **O(1)** time.	05	3	4
4	**Heap sort algorithm – 5M** **Algorithm Heap Sort** 1. Build a Max-Heap from the input array using the bottom-up heapify approach. 2. Repeat the following until the heap size is greater than 1: a. Swap the first element (maximum) with the last element. b. Reduce the heap size by 1. c. Heapify the root element to restore the max-heap property. **Algorithm Heapify** Input: Array A[1..n] (1-based indexing) Output: A is transformed into a max-heap for i = floor(n/2) downto 1: siftDown(A, i, n) Function siftDown(A, i, n): k = i v = A[k] heap = false			

	while not heap and 2*k ≤ n:
	j = 2*k
	if j < n and A[j] < A[j+1]:
	j = j + 1
	if v ≥ A[j]:
	heap = true
	else:
	A[k] = A[j]
	k = j
	A[k] = v
	
	Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8. ----3M
	 
	The heap construction stage of the algorithm is in O(n) and sort is O(n log n) we get O(n) + O(n log n) = O(n log n). **(2M)**			

5	**Write the pseudocode for Horspool's algorithm to search for the pattern in the text. ------- 3M**
	ALGORITHM *HorspoolMatching*($P[0..m-1]$, $T[0..n-1]$)
	//Implements Horspool's algorithm for string matching
	//Input: Pattern $P[0..m-1]$ and text $T[0..n-1]$
	//Output: The index of the left end of the first matching substring
	// or −1 if there are no matches
	ShiftTable($P[0..m-1]$) //generate *Table* of shifts
	$i \leftarrow m-1$ //position of the pattern's right end
	while $i \leq n-1$ **do**
	$k \leftarrow 0$ //number of matched characters
	while $k \leq m-1$ **and** $P[m-1-k] = T[i-k]$ **do**
	$k \leftarrow k+1$
	if $k = m$
	return $i-m+1$
	else $i \leftarrow i + Table[T[i]]$
	return −1
		10	3	3

	Summarize the efficiency of the algorithm in comparison with brute force approach and its predecessor algorithm. ------- 2M • A simple example can demonstrate that the worst-case efficiency of Horspool's algorithm is in *O(nm)*. • But for random texts, it is in ~~Θ~~*(n)*, and, although in the same efficiency class, Horspool's algorithm is obviously faster on average than the brute-force algorithm. • Horspool is **much faster than brute force** in practice and **easier to implement** than Boyer-Moore, though slightly less efficient than Boyer-Moore in some cases. It strikes a good **balance between simplicity and performance**. solution			

BT-Blooms Taxonomy, CO-Course Outcomes, M-Marks

Marks Distribution	Particulars		CO1	CO2	CO3	CO4	CO5	L1	L2	L3	L4	L5	L6
	Quiz/ Test	Max Marks	02	12	21	25		2	10	48			

Course Outcomes: After completing the course, the students will be able to: -	
CO 1	Apply knowledge of computing and mathematics to algorithm analysis and design.
CO 2	Analyze a problem and identify the computing requirements appropriate for a solution.
CO 3	Apply algorithmic principles and computer science theory to the modeling for evaluation of computer-based solutions in a way that demonstrates comprehension of the trade-offs involved in design choices.
CO 4	Investigate and use optimal design techniques, development principles, skills and tools in the construction of software solutions of varying complexity.
CO 5	Demonstrate critical, innovative thinking, and display competence in solving engineering problems.
CO 6	Exhibit effective communication and engage in continuing professional development through experiential learning.

RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Go, change the world

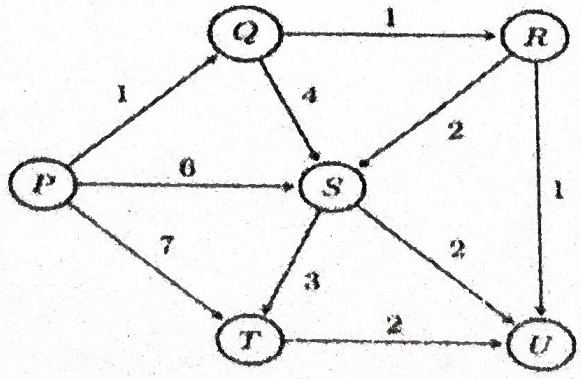
Academic year 2024-2025 (Even Sem)

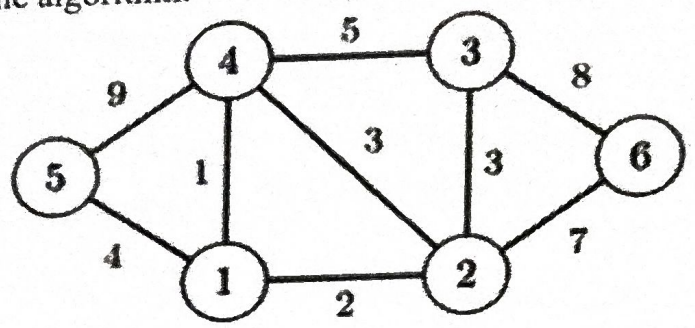
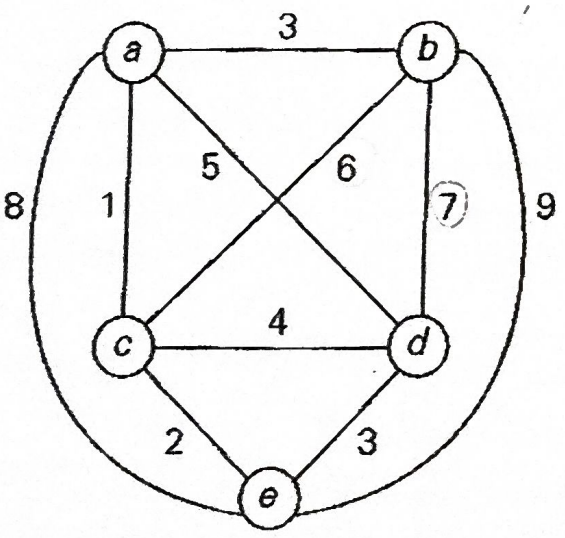
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Date: June 2025	**Improvement Test**	Maximum Marks: 10 + 50
Course Code: CD343AI	Sem: 4th UG	Duration: 120 Minutes

Design and Analysis of Algorithms
(Common to CS/CD/CY/IS/AIML)

Note: - Answer all the questions.

S.No	Part A	M	BT	CO
1.1	Write two important differences between backtracking and branch & bound design techniques.	02	1	2
1.2	A networking company uses a compression technique to encode the message before transmitting over the network. Suppose the message contains the following characters with their frequency: Char: a (5), b (9), c (12), d (13), e (16), f (45) frequency Encode word "**dead**".	02	2	3
1.3	What are promising and non-promising nodes in state space tree?	02	1	1
1.4	State assignment problem. Which design techniques can be used to solve it?	02	1	1
1.5	A problem that contains _____ subproblems, where the same subproblem is solved multiple times, is a good candidate for dynamic programming.	01	1	2
1.6	Which type of search is typically used in the branch and bound algorithm?	01	1	2
	Part B			
1	Using Dynamic Programming, solve the given instance of 0/1 Knapsack problem. Consider the capacity of Knapsack (m) = 5. Write the algorithm. **Item**: 1, 2, 3, 4 **Weight**: 2, 1, 3, 2 **Profit**: 12, 10, 29, 15	10	3	5
2	Write Dijkstra's algorithm and apply it to find shortest path from vertex '**P**'. 	10	3	2

3	Apply Prim's algorithm to obtain a minimum cost spanning tree for the given graph. Write the algorithm. 	10	3	3
4	Apply Backtracking technique to solve the Sum of Subset Problem for the instance d = 15 and S = {3, 5, 6, 7}. Write the algorithm.	10	2	4
5	Consider the graph given below representing an instance of Travelling Salesperson Problem, where source vertex is **"b"**. Apply Branch and Bound technique and obtain the solution where **"c"** comes before "a". Explain the procedure to compute lower bound. Write the state-space tree and number the nodes. 	10	4	5

BT-Blooms Taxonomy, CO-Course Outcomes, M-Marks

Marks Distribution	Particulars		CO1	CO2	CO3	CO4	CO5	L1	L2	L3	L4	L5	L6
	Quiz/ Test	Max Marks	4	14	12	10	20	8	12	30	10	-	-

Course Outcomes: After completing the course, the students will be able to: -	
CO 1	Apply knowledge of computing and mathematics to algorithm analysis and design.
CO 2	Analyze a problem and identify the computing requirements appropriate for a solution.
CO 3	Apply algorithmic principles and computer science theory to the modeling for evaluation of computer-based solutions in a way that demonstrates comprehension of the trade-offs involved in design choices.
CO 4	Investigate and use optimal design techniques, development principles, skills and tools in the construction of software solutions of varying complexity.
CO 5	Demonstrate critical, innovative thinking, and display competence in solving engineering problems.
CO 6	Exhibit effective communication and engage in continuing professional development through experiential learning.

USN | 1 | R | V | 2 | 3 | C | D | 6 | 0 | 3 |

RV College of Engineering

R V College of Engineering

R V Vidyanikethan Post
Mysuru Road Bengaluru - 560 059

IV Semester BE Regular /Supplementary Examinations June/July - 2025.
(Common to CS/CD/CY/IS/AI&ML)
Course : Design and Analysis of Algorithms-CD343AI

Time : 3 Hours **Maximum Marks : 100**

Instructions to the students

1. Answer all questions from Part A. Part A questions should be answered in the first three pages of the answer book only.
2. Answer FIVE full questions from Part B. In Part B question number 2, is compulsory. Answer any one full question from 3 and 4, 5 and 6, 7 and 8, and 9 and 10.

Part A

Question No	Question	M	CO	BT
1.1	Explain how the time complexity of an algorithm can be affected by changes in input size and algorithmic optimizations, giving an example.	02	1	2
1.2	What is the difference between internal and external sorting?	02	1	2
1.3	What does it mean when we say that an algorithm X is asymptotically more efficient than Y?	02	1	3
1.4	How does using the median of the array as the pivot in Quick Sort improve its performance in scenarios where the array is nearly sorted, potentially avoiding the typical degradation to $O(n^2)$ time complexity seen in standard Quick Sort implementations? What will be the time complexity?	02	2	3
1.5	What is the time complexity of following recurrence relation? $T(n) = 3T(n/2) + n^2$	02	2	2
1.6	What is the disadvantage of a space-time trade-off?	02	1	1
1.7	Provide an example of Instance Simplification in the Transform-and-Conquer strategy	01	1	1
1.8	Indicate whether each of the following properties is true for every Huffman code.	02	1	2

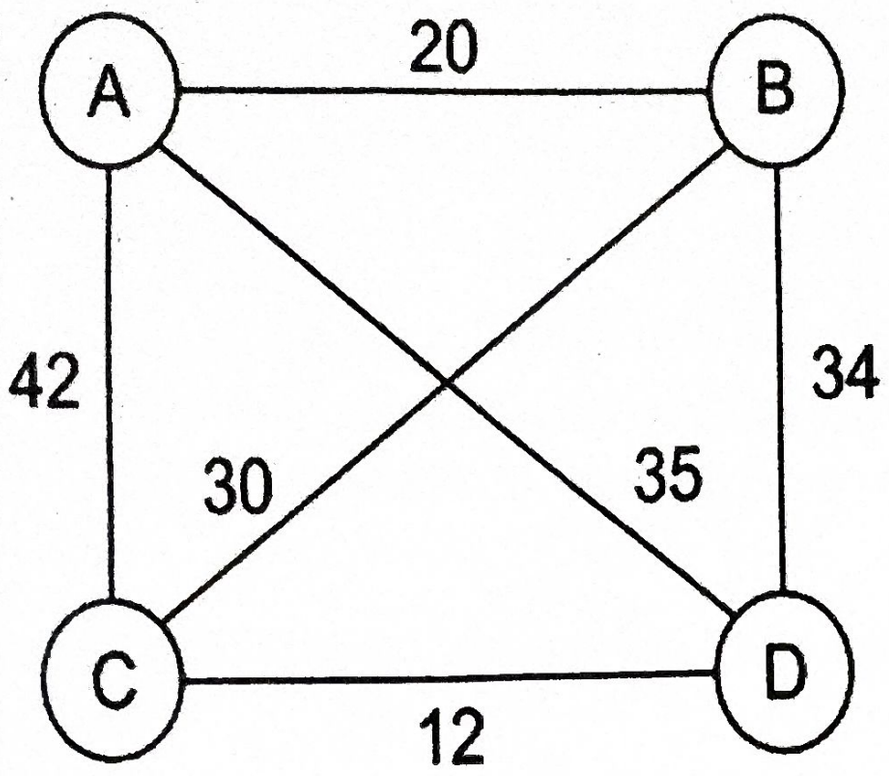
	a. The code words of the two least frequent symbols have the same length. b. The code word's length of a more frequent symbol is always smaller than or equal to the code word's length of a less frequent one.			
1.9	What does the term 'memory function' refer to in the context of dynamic programming?	02	1	2
1.10	Define the leaf structure of a state space tree.	02	2	2
1.11	The problem of determining if a number is prime is in class _____.	01	1	1

Part B

Question No	Question	M	CO	BT
2a	Prove that $O(f(n)) \subseteq O(g(n))$ if $f(n) \in O(g(n))$.	06	1	3
2b	For a given algorithm with time complexity O(n^2+nlogn), what is the simplified time complexity and why?	06	1	3
2c	With an example analyze the asymptotic time complexity of selection sort in worst case	04	2	2
3a	Write the recurrence relation for divide and conquer long integer multiplication algorithm and solve for its time complexity.	08	2	4
3b	Analyse any three scenarios where Binary Search is not an optimal choice.	08	2	4
	OR			
4a	In a software project, the following source files have dependencies: • main.cpp depends on ui.cpp and core.cpp • ui.cpp depends on graphics.cpp and config.cpp • core.cpp depends on utils.cpp and config.cpp • utils.cpp depends on log.cpp a) Represent the above dependencies using a directed graph. b) Determine a valid compilation order of the source files using topological sorting.	08	4	3
4b	Write the Quick Sort algorithm and demonstrate its working on the following list of elements: `10, 80, 30, 90, 40, 50, 70`. Use the **last element** as the pivot during partitioning.	08	4	3

5a	Apply Boyer Moore algorithm to search for the pattern: P = "ABCDABD" in the text T = "ABC-ABCDAB-ABCDABCDABDE"	08	3	3
5b	Consider the problem of Checking element uniqueness in an array. Design a presorting-based algorithm for solving this problem. Compare its efficiency with the brute-force approach in terms of time complexity.	08	3	3
	OR			
6a	Given the input array A = [14, 2, 2, 18, 36, 18, 1,17,2], demonstrate step-by-step how Comparison Counting Sort works to produce the sorted array and discuss its efficiency	08	2	3
6b	Describe variations of Transform-and-Conquer algorithmic strategy. In what scenarios is Transform-and-Conquer preferable over Brute Force and divide-and conquer? Support your answer with suitable examples.	08	2	2
7a	Explain how one can generate a Huffman code without an explicit generation of a Huffman coding tree.	08	4	4
7b	Compare Dynamic Programming and Greedy Design technique	08	3	4
	OR			
8a	Design a linear-time algorithm for solving the single-source shortest-paths problem for DAG (Directed Acyclic Graph) represented by their adjacency lists.	08	4	6
8b	Apply 0/1 Knapsack Memory Functions, find the maximum profit, write recurrence Relation Knapsack size 5	08	3	3
9a	The following map shows set of cities and distance between them. ABC travels is planning to visit the cities A, B, C and D. Solve using "branch and bound technique", to find the shortest possible route that visits every city exactly once and returns to the starting point. Consider City D as the starting point.	10	4	3

Objects (n)	1	2	3
Weight	2	3	2
Profit	5	10	8



9b	Explain the P vs NP problem and its implications in computational complexity.	06	2	2

OR

10a	There are 3 workers and 3 jobs. The cost of assigning each worker to each job is given by the cost matrix: Write "branch and bound technique" based algorithm to solve the given problem. Using the same solve the problem to assign each worker to one job (no repeats) such that the total cost is minimized.	10	3	3

	Job 1	Job 2	Job 3
Worker 1	9	2	7
Worker 2	6	4	3
Worker 3	5	8	1

10b	Solve the N-Queens problem for N = 4 using backtracking. Explain the backtracking steps and provide the board configuration(s).	06	4	2