Unit-1

Q1. Explain the process of designing an algorithm. Give characteristics of an algorithm.

Q2. Explain asymptotic notations.

Q3. For $T(n)=7T(n/2)+18n^2$, Solve the recurrence relation and find the time complexity.

Q4. Solve the recurrence relation and find the time complexity

$T(n)= T(n-1) + n$ where $T(1)=1$

Q5. Describe performance analysis, space complexity and time complexity.

Q6. Solve the following recurrence relation using substitution method:

$$T(n)= \begin{cases} aT(n/b)+f(n) & n>1 \\ T(1) & n=1 \end{cases}$$

where $a=5, b=4$, and $f(n)=cn^2$.

Q7. Prove that function $f(x)= 5x^4 +7x + 3$ is Theta (x^4).

Q8. Write a recursive algorithm for computing the nth fibonacci number.

Q9. Write an Algorithm using recursion that determines the GCD of two numbers. Determine the time and space complexity.

Q10. Suppose we wish to search a linked list of length n, discuss best, worst and average case of searching a random element.

Q11. Give the general procedure of divide and conquer method.

Q12. Illustrate Merge sort algorithm and discuss its time complexity.

Q13. Simulate Quick sort algorithm for the following example:

25,36,12,4,5,16,58,54,24,16,9,65,78

Q14. Is quick sort a stable sorting method? Is merge sort a stable sorting method? Justify both your answers.

Q15. Can we say that the time for Merge Sort is $\Theta(n \log n)$. What is its worst and best time of procedure for Merge Sort.

Q16. What is stable sorting method? Is it necessary for counting sort to be a stable sorting method? Justify.

Q17. Explain the Quick Sort algorithm with an example and also draw the tree structure of the recursive calls made. Write 2 advantages and 2 disadvantages of Quick Sort.

Q18. Explain the Merge Sort algorithm with an e.g. and also draw the tree structure of the recursive calls made.

Q19. Which sorting algorithm is best if the list is already sorted. Why?

Q20. Illustrate the operation of bucket sort algorithm on following array:

{0.79, 0.13, 0.16, 0.64, 0.56, 0.69, 0.79, .47}

Q21. Sort the following list using heap sort. Display each step:

20, 12, 25, 6, 0, 15, 13

Q22. Define Heap. Sort the given list of Elements using heap sort: 2, 9, 7, 6, 5, 8

Q23. Explain insertion sort with the help of example and also determine its complexity.

Q24. Explain binary search with the help of example and also determine its complexity

Q25. Briefly describe all the following with the help of >>examples >>advantages >>disadvantages >>importance >>definition:

a. Big (O)
b. Small (o)
c. Theta (Θ)
d. Big omega (Ω)
e. Small omega (ω)

Q26. Explain quick sort with the help of example and also determine its complexity.

Q27. Illustrate the operation of the PARTITION on the array A = <13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 >.

Q28. Explain merge sort with the help of example and also determine its complexity.

Q29. Solve the following list using merge sort: - 5, 2, 4, 6, 1, 3, 2, 6.

Q30. Explain run time efficiency and run time analysis of algorithms.

Q31. Explain briefly with the help of example and determine complexity:

a. Substitution method
b. Iteration method
c. Recursion tree method
d. Master method

Q32. Explain run time analysis of divide and conquer algorithm.

Q33. Explain run time analysis of recurrence relations

Q34. Explain Strassen matrix multiplication with the help of example.

Q35. Elaborate and define with the help of example asymptotically non-negative functions, asymptotically positive functions, asymptotically negative functions, asymptotic tight bounds using the general concept of f(n), g(n), c(n).

Q36. Solve the following recurrences:

(i) $T(n) = T(n/2) + 1$

(ii) $T(n)= 2T((n/2)+16)+n$

(iii) $T(n) = T(\sqrt{n}) + 1$

(iv) $T(n)=3T(n/4) + n$

(v) $T(n)=T(n-1) + 1/n$

(vi) $T(n)=2T(n/2) + n^2$

(vii) $T(n)= T(n/3) + T(2n/3) +n$

(viii) $T(n)= 16T(n/4) + n^3$

(ix) $T(n) = T(n – 1) + n^4$

(x) $T(n) = 2T(\sqrt{n}) + \log n$

Q37. Prove the following:

a. $2n+3= \theta(n)$
b. $3^{n+3} = \theta(3^n)$
c. $27\ n^2 + 16n = O(n^{3)}$
d. $3\ n^3 + 4n = \theta(n^{3)}$
e. $3^n = o(3^{3n})$
f. $10\ n^2 + 7 = \Omega(n)$
g. $F(n) = \log n$ (base 2) is $O(n^{@)})$ for any $@>0$
h. $n! = \omega(\ 2^{n)}$
i. $n! =o(n^{n)}$
j. is $2^{n+1} = O(2^{n)}$? Is $2^{2n} = O(2^{n)}$?`

Q38. Prove that $(n+a)^b = \theta(n^b)$, $b > 0$

Q39. “Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?”

Q40. Let f(n) and g(n) be asymptotically non-negative functions. Using the basic definition of θ notation prove that max (f(n), g(n)) = θ(f(n) + g(n)).

Q41. Write an algorithm for the Quick Sort and illustrate the operation of the PARTITION on the array A = <9, 2, 7, 5, 1, 4, 10, 6>.

Q42. Compare and contrast the time complexities of Quick Sort and Merge Sort. What are the implications of these differences for practical use?

Q43. Analyze the performance measurements of an algorithm that has a time complexity of O(n^2) and suggest scenarios where it might still be acceptable to use.

Q44. Examine how the choice of data structures can impact the efficiency of algorithms. Provide examples of specific algorithms where data structure choice is critical.

Q45. Analyze a given recursive algorithm to identify its base case, recursive case, and overall time complexity.

Q46. Evaluate the effectiveness of the Master Method in solving recurrence relations. Are there limitations to its applicability?

Q47. Critique the divide-and-conquer strategy in the context of sorting algorithms. In what situations might it not be the best approach?

Q48. Assess the impact of algorithm complexity on real-world applications. Provide examples where algorithm efficiency significantly affects performance.

Q49. Evaluate the trade-offs between using a non-recursive and a recursive algorithm for a specific problem. Which approach would you recommend, and why?

Q50. Design a new algorithm that uses a divide-and-conquer strategy to solve a specific problem. Describe its structure and analyze its time complexity.

Q51. Create a detailed comparison table of various sorting algorithms, including Quick Sort, Merge Sort, Counting Sort, and Heap Sort, focusing on their complexities and use cases.

Q52. Propose an alternative method for analyzing the performance of recursive algorithms that integrates empirical and theoretical approaches. How would you implement it?

Q53. Invent a scenario where a recursive algorithm would outperform an iterative algorithm in terms of efficiency. Describe the algorithm and its benefits.

Q54. What is the divide-and-conquer approach in algorithms, and what are its main steps?

Q55. Compare and contrast Quick Sort and Merge Sort in terms of their algorithmic structure and performance.

Q56. Explain the Master Method and provide an example of how it can be applied to solve a recurrence relation.

Q57. What are the properties of asymptotic notations that make them useful for algorithm analysis?

Q58. How do you combine functions in asymptotic notation?

Q59. Analyze whether quick sort is a stable sorting method and provide reasoning to support your conclusion.

Q60. Describe the steps involved in designing an algorithm and list five characteristics that define an algorithm.

Unit-2

Q1. What is back tracking? Where Back tracking is used to solve the problem.

Q2. What is the difference between 0/1 Knapsack problem and fractional Knapsack problem.

Q3. How 8-Queen's problem can be solved using back tracking and explain with an example. Show the state space tree.

Q4. Explain method of Greedy algorithm. Find the greedy solution for following job sequencing with deadlines problem n = 7,

(p1,p2,p3,p4,p5,p6,p7) = (3,5,20,18,1,6,30),

(d1,d2,d3,d4,…,d7) = (1,3,4,3,2,1,2)

Q5. Define Greedy knapsack. Find the optimal solution of the Knapsack instance n= 7, M=15, (p1, p2, ……p7) = (10,5,15,7,6,18,3) and (w1,w2,. ...w7)=(2,3,5,7,1,4,1).

Q6. Define Greedy knapsack. Find the optimal solution of the Knapsack instance n= 7, M=20, (p1, p2, ……p7) = (8,5,6,7,6,12,3) and (w1,w2,....w7)=(2,10,8,7,6,4,11).

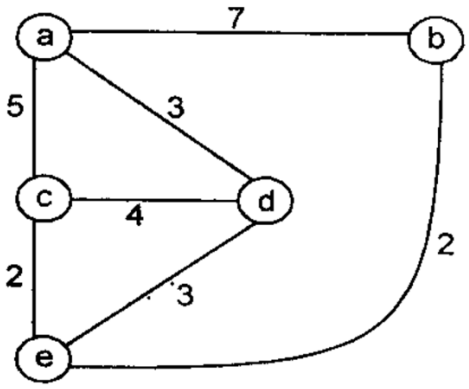
Q7. Give the optimal solution for 0/1 knapsack problem using greedy method. (p1,p2,p3, p4) = (11, 21, 31, 33), (w1, w2, w3, w4) = (2, 11, 22, 15), M=40, n=4.

Q8. Derive time complexity of job sequencing with deadlines. Obtain the optimal solution when n=5, (p1, p2,…)=(20,15,10,5,1) and (d1,d2,…)=(2,2,1,3,3).

Q9. A motorist wishing to ride from city A to B. Formulate greedy based algorithms to generate shortest path and explain with an example graph.

Q10. Write down the Bellman Ford Algorithm to solve single source shortest path problem. Write its time complexity.

Q11. Write Dijikstra's algorithm to find shortest path in a graph. Apply Dijikstra algorithm for following graph (Consider 1 is the source vertex).



Q12. . Let G be an undirected, weighted graph, and s and t be any two vertices. Suppose all the edge weights are positive real numbers except one edge (u, v) ∈ G; the weight of the edge (u, v) is some negative real number. Design a O(m + n log n) time algorithm to find the distance of t from s where n and m denote the number of vertices and edges in G respectively.

Q13. There is a sequence of n activities a1, a2, . . . , an with corresponding utilities u1, u2, . . . , un. You wish to perform all these n activities according to this sequence within k days. If you perform the activities from ai to aj for some 1 <= i<= j<= n on the j-th day, then your utility Uj for the j-th day is max $\{U_l : i <= l <= j\}$. Your total utility is $\sum_{l=1}^{k} U_l$. Design a greedy algorithm to find the sequence of activities you will perform on every day which maximizes your total utility.

Q14. Suppose you have to give Re N to your friend. You have enough number of 500, 200, 100, 50, 20, 10 rupee notes each at your disposal. Your goal is to give Re N to your friend with minimum number of notes. For example, Re 600 can be changed using 3 Re 200 notes as well as using 1 Re 500 note and 1 Re 100 note. However, the later one uses minimum number of notes. B Either prove correctness or provide counter example of the following greedy strategy: keep picking highest denomination as much as you can! B Provide a set of denominations for which the above greedy strategy will fail.

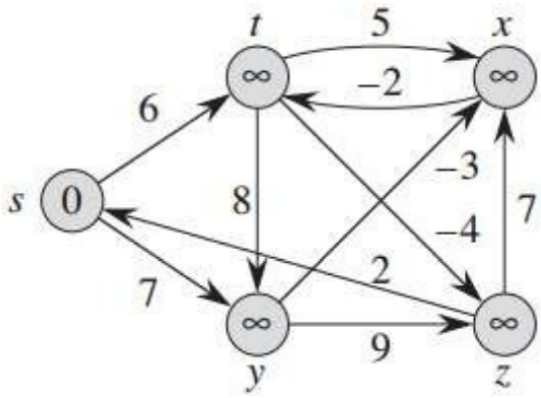
Q15. Using state space tree prove that:

1. There is no solution for a 2 queen problem
2. There are multiple solutions for a 4 queen problem

Q16. What is the central principle of backtracking? Apply backtracking to solve the below instance of sum of subset problem S = {5, 10, 12, 13, 15, 18} d = 30.

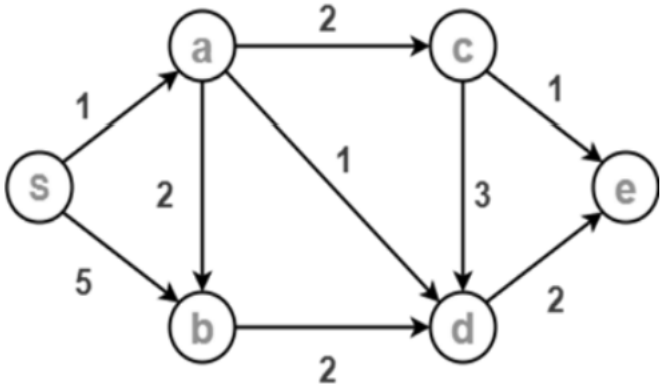
Q17. Explain the following: a] LC Branch and bound b] FIFO Branch and bound

Q18. Apply Dijkstra's Algorithm on the given graph.

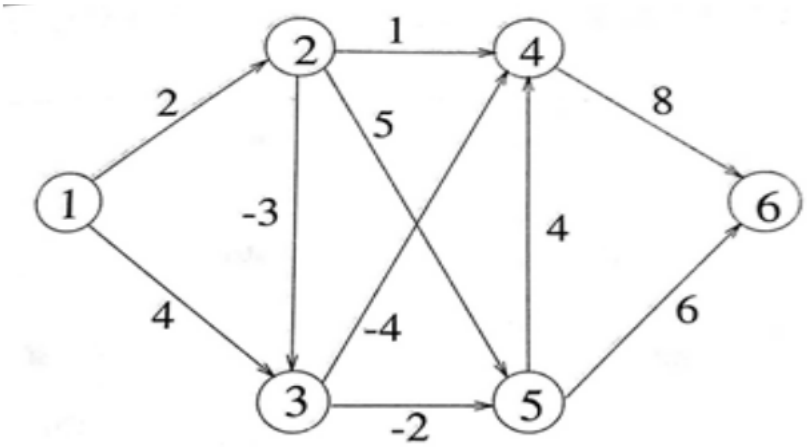


How is Bellman-Ford different from Dijkstra's Algorithm? To what design technique does the algorithm belong to? Explain.

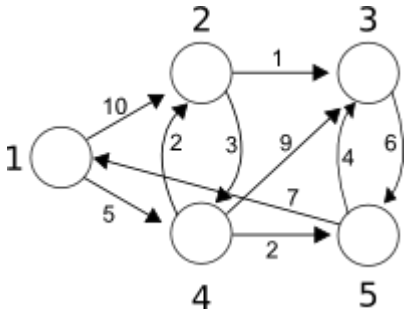
Q19. Design Dijkstra's algorithm and apply the same to find single source shortest path for the given graph by considering 'S' as the source vertex



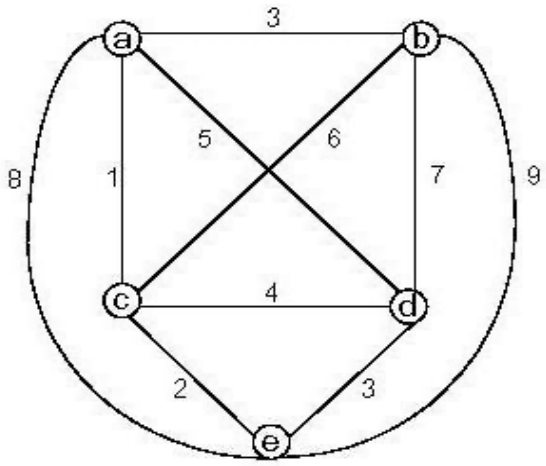
Q20. Find the shortest path from node 1 to every other node in the given graph using Bellman-Ford algorithm.



Q21. Write the Dijkastra's Algorithm for finding the shortest path from a single source. What is its complexity. Also find the shortest paths in the following graph assuming that "1" is the source node.



Q22. Solve the following instance of travelling salesman problem using Branch and Bound.



Q23. What is the greedy algorithm paradigm, and how does it differ from other algorithmic approaches?

Q24. List the key characteristics that make a problem suitable for a greedy solution.

Q25. Explain the process of making greedy choices with an example.

Q26. What are the limitations of the greedy approach?

Q27. How can you prove that a greedy algorithm is optimal for a specific problem?

Q28. Describe a task scheduling problem that can be solved using a greedy approach. What is the objective?

Q29. How would you apply a greedy algorithm to minimize the total completion time of tasks?

Q30. Provide a specific example of a task scheduling problem and outline the steps of your greedy algorithm.

Q31. Explain the fractional knapsack problem and how it differs from the 0/1 knapsack problem.

Q32. Describe the greedy strategy for solving the fractional knapsack problem.

Q33. Write the pseudocode for a greedy algorithm that solves the fractional knapsack problem.

Q34. What is the time complexity of the greedy algorithm for the fractional knapsack problem, and why?

Q35. Explain the single source shortest paths problem. Why is it important in graph theory?

Q36. Compare and contrast Dijkstra's Algorithm and the Bellman-Ford Algorithm in terms of their approach and use cases.

Q37. Describe how Dijkstra's Algorithm works step by step, including its data structures.

Q38. What are the limitations of Dijkstra's Algorithm, and in what scenarios might Bellman-Ford be preferred?

Q39. Explain the Bellman-Ford Algorithm. How does it handle negative weight edges?

Q40. Discuss the time complexities of both Dijkstra's and Bellman-Ford algorithms in different scenarios.

Q41. What is backtracking, and how does it differ from other algorithmic strategies?

Q42. Let W= {5,7,10,12,15,18,20} and m=35. Find all possible subsets of W that sum to m using recursive backtracking algorithm. Draw the portion of state space tree that is generated.Describe the N-Queens problem and how it can be approached using backtracking.

Q43. Write pseudocode for a backtracking algorithm that solves the N-Queens problem.

Q44. What is the branch and bound method, and how does it differ from backtracking?

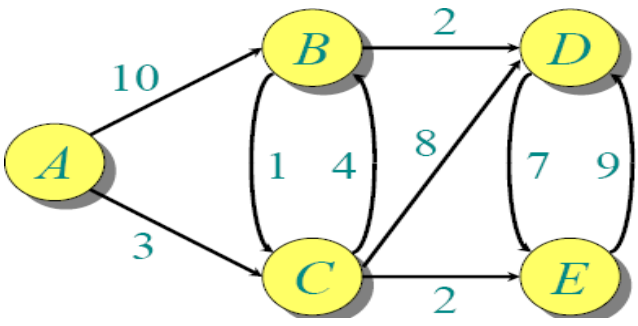
Q45. Explain how branch and bound can be used to solve optimization problems.

Q46. Discuss an example of a problem that can be solved using branch and bound, and outline the algorithm.

Q47. What are the advantages and disadvantages of using the branch and bound approach?

Q48. Define the sum of subsets problem. How does it relate to other combinatorial optimization problems?

Q49. Analyze the shortest path from source node A to node E using Dijkstra's algorithm on the given graph.



Q50. What are the key considerations in designing an efficient algorithm for the sum of subsets problem?

Q51. Provide a sample problem with a specific set of numbers and a target sum, and outline the backtracking steps to find a solution.

Q52. Compare and contrast the greedy paradigm with backtracking and branch and bound. In what scenarios might each be preferred?

Q53. How can you evaluate the performance of greedy algorithms versus backtracking algorithms?

Q54. What role do heuristics play in optimization problems involving the greedy paradigm?

Q55. Describe a real-world application where a greedy algorithm would be suitable.

Q56. How do dynamic programming and greedy algorithms differ in approach?

Q57. Discuss the importance of problem formulation when applying any of these paradigms.

Q58. Can backtracking be used in conjunction with the greedy approach? Provide an example scenario.

Q59. What strategies can be employed to optimize a backtracking solution?

Q60. Explain the term: LC Branch and bound.

Unit-3

1. What steps are used in Dynamic programming approach? Discuss the 0/1 Knapsack problem with respect to Dynamic Programming. Is Greedy method equally applicable for the above problem?
2. Apply the bottom-up dynamic programming algorithm to the following instance of 0/1 Knapsack Problem:

Item	Weight	Value
1	7	₹42
2	3	₹12
3	4	₹40
4	5	₹25

Knapsack Capacity W = 40.

3. Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is <5, 10, 3, 12, 5>.
4. What are the various methods for the traversal of a graph. Explain any one with the algorithm and an example.
5. Briefly explain the following algorithm design techniques with application areas of each:

(i) Dynamic Programming

(ii) Greedy Approach

6. What do you understand by longest common subsequence? What are its applications? Write an algorithm to find the Longest Common Subsequence and determine an LCS of <1, 0, 0, 1, 0, 1, 0, 1> and <0, 1, 0, 1, 1, 0, 1, 1, 0>.
7. What is dynamic programming?
8. Give an $O(n^2)$ time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.
9. Solve following knapsack problem using dynamic approach

   (I, w , v)=<(I1,5,10), (I2, 4, 12), (I3, 5, 8), (I4,4,6)>

   Weight of Knapsack=10
10. What is the matrix chain multiplication problem?
11. Describe the longest common subsequence (LCS) problem.
12. What is the 0/1 knapsack problem?
13. You are developing an application that needs to optimize resource allocation among a set of tasks, each with a specific requirement and limited resources available. You are considering two approaches: a greedy algorithm and a dynamic programming solution.
    - Analyze the asymptotic performance of a basic greedy algorithm for resource allocation. What is its time complexity, and what scenarios might lead to suboptimal solutions?
    - Describe how a dynamic programming approach could be applied to this problem. Outline the key steps involved in your algorithm and explain when it is more suitable than the greedy approach.
    - Identify the major data structures that would be required to implement both the greedy and dynamic programming solutions. Justify your choices based on efficiency and ease of implementation.
    - Discuss whether the resource allocation problem you are solving is NP-complete. Provide reasoning to support your conclusion.
14. Explain the principle of optimality in dynamic programming.
15. Describe how divide and conquer breaks a problem into smaller subproblems
16. What are the advantages of using the Floyd-Warshall algorithm over other shortest path algorithms?
17. Explain the concept of backtracking and illustrate a scenario in which backtracking is effectively used to solve a problem.

18. How does the dynamic programming approach for matrix chain multiplication differ from a brute-force approach?
19. Why is the LCS problem significant in fields like bioinformatics?
20. How does the 0/1 knapsack problem illustrate resource allocation challenges?
21. Given a set of matrices, apply the matrix chain multiplication algorithm to find the optimal multiplication order.
22. Given two sequences, X="ABCBDAB" and Y="BDCAB", write a function to find the length of the longest common subsequence (LCS) between the two sequences. Describe how you would approach solving this problem using dynamic programming.
23. Use the Floyd-Warshall algorithm to find the shortest paths in a given weighted graph.
24. Solve an example of the longest common subsequence using dynamic programming and demonstrate the steps.
25. Apply the 0/1 knapsack algorithm to a real-world scenario, such as packing for a trip, and explain your choices.
26. Demonstrate how the recursive formulation of the LCS problem can be transformed into a dynamic programming solution.
27. Implement a simple version of the Floyd-Warshall algorithm in pseudocode.
28. Compare and contrast dynamic programming and divide and conquer in terms of efficiency and applicability.
29. Analyze the time complexity of the Floyd-Warshall algorithm. When is it more efficient to use it compared to Dijkstra's algorithm?
    - Consider a straight highway with villages alongside. You can represent highway as integer axis and the position of each village as a single integer coordinate. There are no two villages in the same position. The distance between two positions is the absolute value of the difference of their integer coordinates.
    - The task is to build post offices in villages. A village and the post office in it will have the same position. For building the post offices, their positions should be chosen so that the total sum of all distances between each village and its nearest post office is minimum.

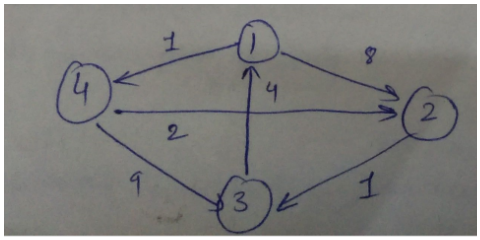
      You have to write a pseudocode to computes the least possible sum of all distances between each village and its nearest post office.
    - Input will be the positions of the villages and the number of post offices.

30. Examine the similarities and differences in the approach to solving the LCS problem and the 0/1 knapsack problem.
31. Discuss the trade-offs between time and space complexity in dynamic programming solutions.
32. Analyze how overlapping subproblems in dynamic programming enhance efficiency.
33. Evaluate the effectiveness of the Floyd-Warshall algorithm in various applications. What are its limitations?
34. Critically assess the dynamic programming approach to matrix chain multiplication and suggest potential improvements.
35. Determine an LCS of strings <1 0 0 1 0 1 0 1> and <0 1 0 1 1 0 1 1 0>
36. Discuss the pros and cons of using dynamic programming for the 0/1 knapsack problem compared to other techniques.
37. Evaluate how different data structures can affect the performance of dynamic programming algorithms.
38. Assess the impact of memoization in dynamic programming and compare it to the tabulation method.
39. Design a new algorithm that combines dynamic programming with another paradigm to solve the 0/1 knapsack problem more efficiently.
40. Create a flowchart that illustrates the steps of the Floyd-Warshall algorithm, highlighting key decisions.
41. Propose a hybrid algorithm that applies both divide and conquer and dynamic programming to solve a complex problem.
42. Develop a case study that outlines the application of matrix chain multiplication in a real-world computing problem.
43. Perform matrix chain multiplication where p = [5, 10, 3, 12, 5]
44. Solve the following instance of the 0/1 knapsack problem, given the knapsack capacity in W=9 kg using dynamic programming:

Item	Weight (kg)	Value (Rs.)
1	4	12
2	3	18
3	2	14
4	5	28

45. Design an experiment to compare the performance of dynamic programming versus divide and conquer on specific problem sets.
46. How can understanding the differences between dynamic programming and divide and conquer help in selecting the right algorithm for a problem?
47. You are tasked with developing an algorithm to solve the 0/1 Knapsack Problem, where you have a set of items, each with a weight and a value, and you want to maximize the total value without exceeding a given weight limit.
    - Choose between a greedy algorithm and a dynamic programming approach for solving the 0/1 Knapsack Problem. Justify your choice based on the characteristics of the problem.
    - Provide a high-level overview of how your chosen algorithm works. Include the main steps involved in the algorithm and any key data structures you would use.
    - Analyze the time and space complexity of your chosen algorithm. How does this complexity impact the algorithm's efficiency for larger inputs?
    - Discuss a real-world scenario where your algorithm could be applied effectively. Explain why your chosen technique is suitable for this scenario.
48. What role does recursion play in both dynamic programming and divide and conquer approaches?
49. In what situations might a greedy approach outperform dynamic programming for solving optimization problems?
50. Imagine you have a collection of N wines placed next to each other on a shelf. For simplicity, let's number the wines from left to right as they are placed on the shelf with integers from 1 to N respectively. The price of the i_{th} wine is p_i. Because wines get better every year, so consider that the price of wine increases over time. So consider the current year as year 1 so in year y the price of the i_{th} wine will be y* p_i i.e. y times of the current value. Your task is to sell all the wines but you can sell exactly one wine per year, and each year you are allowed to sell either the left most or the right most wine placed on the shelf. (You are not allowed to reorder the wines)
    - Find out what is the maximum profit you can earn, if you sell the wines in optimal order.
    - If the prices of the wines are p1= 1, p2= 4 , p3= 2, p4= 3 (in left to right order). Give the optimal solution.

51. Examine the Matrix Chain Multiplication problem and analyze how dynamic programming optimizes the time complexity in determining the optimal parenthesization of the matrix chain. Provide an example to illustrate this process and evaluate the benefits of using dynamic programming for this problem.
52. What insights can be gained from analyzing the performance of algorithms like Floyd-Warshall in large-scale networks?
53. How can dynamic programming be utilized in machine learning algorithms, and what are the implications for data processing?
54. Find all source shortest path in following graph using suitable DP based approach.



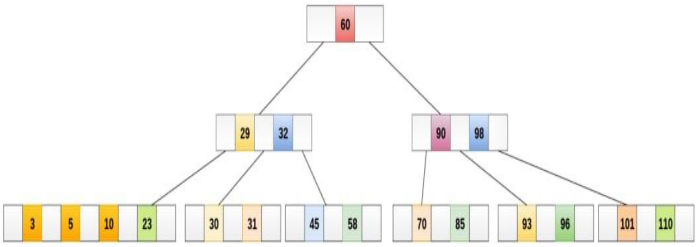
55. Discuss the historical development of dynamic programming and its significance in computer science.
56. How can algorithms like the 0/1 knapsack problem be adapted for multi-dimensional problems?
57. What are the implications of algorithm design choices on the scalability of applications using dynamic programming?
58. Analyze how memory optimization techniques can improve the efficiency of dynamic programming solutions.
59. Discuss the relevance of dynamic programming in solving real-time decision-making problems.
60. Explore the concept of state space in dynamic programming and how it affects problem formulation.

Unit-4

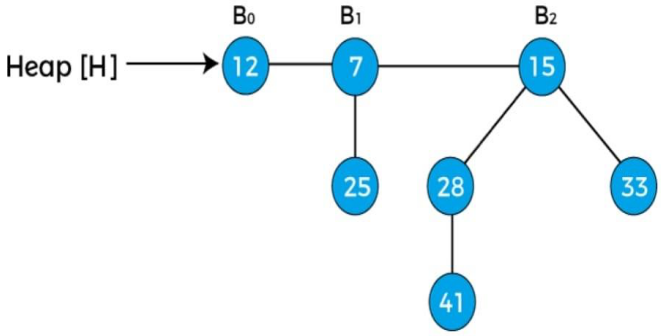
1. Create a RB Tree by adding numbers in sequence: 8, 18, 5, 15, 17, 25, 40 and 80.
2. Delete items in sequence from RB tree created in Q1 in order 17, 25, 15, 5, 8, 18, and 80.
3.

Insert 5, 3, 21, 9, 13, 22, 7, 10, 11, 14, 8, 16 into a B tree

4. Write an algorithm to delete an element from RB Tree.
5. Differentiate between B Tree, RB Tree and AVL Trees.
6. Write a short note on Hashing.
7. Describe the properties of B-Trees. Given a B-Tree of order 5, demonstrate the process of inserting a node with the value 8 into the tree. Evaluate the effects of this insertion on the structure of the B-Tree.



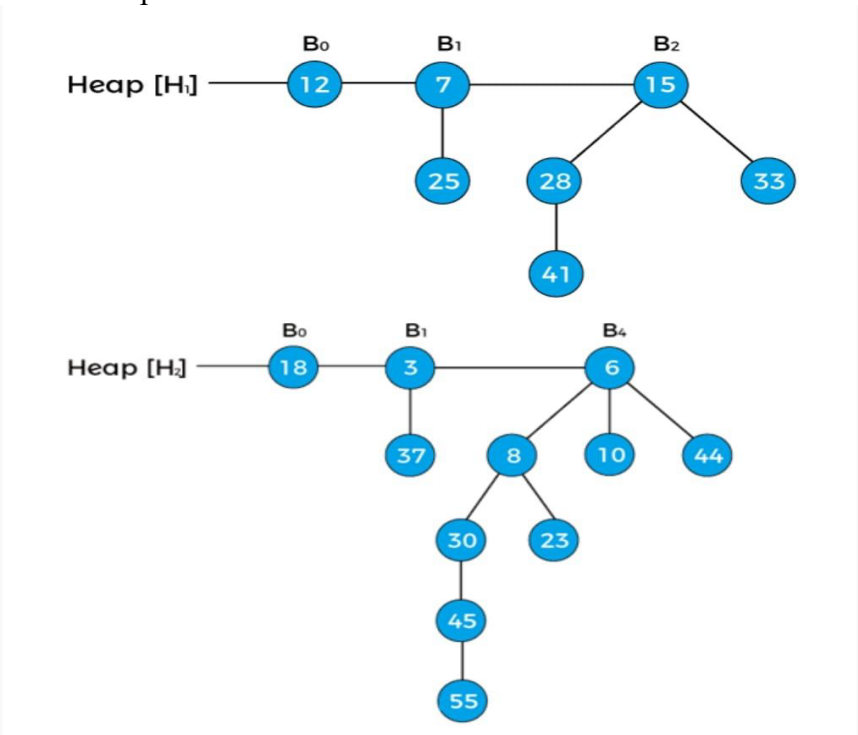
8. What are the key properties of a Red-Black Tree, and how do these properties ensure that the tree remains balanced after insertions and deletions?
9. Explain the process of inserting a new node into a Red-Black Tree. What steps are involved, and how do you handle violations of the Red-Black properties?
10. Describe the deletion process in a Red-Black Tree. What challenges arise during deletion, and how are they resolved to maintain the tree's properties?
11. Compare and contrast the performance of Red-Black Trees with other self-balancing binary search trees, such as AVL Trees. In what scenarios might one be preferred over the other?
12. Discuss the applications of Red-Black Trees in real-world systems. How do their properties make them suitable for use in data structures like associative arrays and databases?
13. Outline the properties of a Binomial Heap. Given the current structure of a Binomial Heap, illustrate the steps required to insert a node with the value 15.



14. Establish the relation between number of nodes and height of AVL tree.
15. Justify the reason of using height Balanced Trees? Explain AVL Rotations with the help of a suitable example.

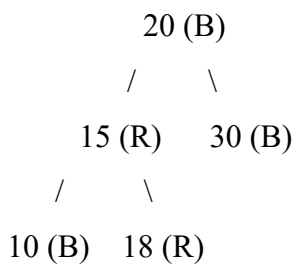
16. Explain the structure of a B-Tree and describe its key properties, including order, height, and balancing mechanisms.
17. Discuss the advantages of using B-Trees over binary search trees in database indexing. How do B-Trees improve search, insertion, and deletion operations?
18. Given a sequence of integers, demonstrate the insertion process into a B-Tree of order 3. Illustrate each step and show the resulting tree.
19. What are the algorithms for deleting a key from a B-Tree? Describe the different cases that need to be handled during deletion and provide an example.
20. Demonstrate the insertion of following elements in a red black tree.

    10, 18, 7, 15, 16, 30, 25
21. Find the minimum and maximum height of any AVL-tree with 11nodes. Assume that height of the root is 0.
22. Analyze the time complexity of various operations (search, insert, delete) in a B-Tree. How do these complexities compare to those of other tree structures, such as Red-Black Trees?
23. What is difference between Binomial Heap and Fibonacci Heap. Perform union of following 2 binomial heaps.



24. Define RB Tree. Analyze why RB tree is preferred over Binary Search Tree. Perform insertion operation in empty Red Black Tree for following nodes values: 38, 31, 12, 19, 8. Show all steps of inserting nodes.
25. Define a binomial heap and explain its structure. How does it differ from other heap structures, such as binary heaps?

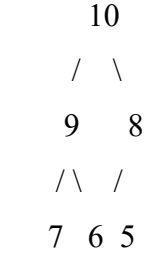
26. Describe the process of inserting a new element into a binomial heap. What steps are involved, and how does this maintain the properties of the heap?
27. Explain how to perform a union operation on two binomial heaps. What algorithm is used, and what are the key steps to ensure the resulting heap is valid?
28. Illustrate the process of deleting the minimum element from a binomial heap. What steps must be taken, and how does the heap structure change as a result?
29. Analyze the time complexity of common operations (insert, delete, find minimum, union) in a binomial heap. How do these complexities compare to those of other heap implementations, such as Fibonacci heaps?
30. What is a Fibonacci heap, and how does its structure differ from that of other heap types, such as binary heaps and binomial heaps?
31. Describe the process of inserting an element into a Fibonacci heap. What steps are involved, and how does the insertion maintain the properties of the heap?
32. Explain the decrease-key operation in a Fibonacci heap. Why is it considered more efficient compared to other heap structures, and what are the implications for the heap's structure?
33. Illustrate how to delete the minimum element from a Fibonacci heap. What are the steps involved in this operation, and how does it affect the heap's organization?
34. Analyze the time complexities of the main operations (insert, find minimum, delete minimum, decrease-key, and delete) in a Fibonacci heap. How do these complexities support its use in graph algorithms, such as Dijkstra's and Prim's?
35. Given the following sequence of numbers: 10, 20, 30, 15, and 25, illustrate the step-by-step process of inserting these values into a Red-Black Tree. Show the tree after each insertion and indicate any necessary rotations or color changes.
36. Consider a Red-Black Tree with the following structure after several insertions:



If you delete the node with the value 15, illustrate the resulting tree and describe the steps taken to maintain the Red-Black properties.

37. Given a Red-Black Tree with 7 nodes, if the height of the tree is 3, what is the maximum number of leaves the tree can have? Explain your reasoning.

38. You have a Red-Black Tree that initially contains the following elements: 5, 10, 15, 20, and 25.If you perform a series of deletions (removing 5 and then 20), show the state of the tree after each deletion and identify any necessary rotations and color adjustments.
39. If you have a Red-Black Tree containing 15 nodes, what is the minimum height of the tree? Show the calculation process based on the properties of Red-Black Trees.
40. Given a B-Tree of order 3, insert the following sequence of keys: 10, 20, 5, 15, and 25. Show the B-Tree after each insertion, indicating how the tree splits when necessary.
41. Consider a B-Tree of order 4 that currently contains the following keys: 10, 20, 30, 40, and 50. If you delete the key 30, illustrate the resulting structure of the B-Tree and explain any adjustments that were made.
42. You have a B-Tree of order 5 with the following keys: 12, 17, 24, 31, and 45. If you perform a search for the key 24, show the path taken through the B-Tree. Then, if you insert the key 29, demonstrate the new structure of the tree.
43. A B-Tree of order 3 contains 12 keys. What is the maximum number of children this B-Tree can have? Explain your reasoning based on the properties of B-Trees.
44. Given a B-Tree of order 4, which currently has the following keys in sorted order: 5, 10, 15, 20, 25, 30, and 35, show how the tree would be structured after inserting the key 18. Indicate any necessary splits and adjustments.
45. Explain the difference between a min-heap and a max-heap. Given the following set of numbers: 3, 1, 4, 1, 5, 9, and 2, construct both a min-heap and a max-heap, showing the structure of each heap.
46. Describe the process of inserting the number 7 into an existing max-heap:



Illustrate the heap after the insertion, including any necessary adjustments to maintain the heap property

47. Consider a min-heap with the following elements: 2, 4, 5, 8, 10, and 15. If you perform a delete-min operation, what will the new structure of the heap look like? Show the steps involved in the deletion process.

48. Given a binary heap with the following elements represented as an array: [20, 15, 30, 10, 8, 25], demonstrate how to convert this array into a valid max-heap using the heapify process. Show each step and the resulting heap.
49. Explain how heaps are used in the heap sort algorithm. Given the following unsorted array: [9, 7, 8, 5, 6, 4], demonstrate the steps of the heap sort process and show the sorted array as a result.
50. Explain the structure of a Fibonacci heap. How do the characteristics of its nodes, such as degree and mark status, contribute to its efficiency?
51. Demonstrate the process of performing a decrease-key operation in a Fibonacci heap. Given the following nodes:
    Node A: value 30, degree 2
    Node B: value 20, degree 1
    Node C: value 25, degree 0
    Show how to decrease the value of Node A to 15 and illustrate the resulting heap structure.
52. Given a Fibonacci heap with the following minimum node values: 10, 20, 15, and 5, illustrate the steps involved in deleting the minimum node. What adjustments are made to the heap structure?
53. Consider the following sequence of operations on a Fibonacci heap: insert(3), insert(8), insert(5), decrease-key(8, 4), and delete-min(). Illustrate the state of the Fibonacci heap after each operation and provide a final result after the delete-min() operation.
54. Analyze the amortized time complexities of the main operations in a Fibonacci heap: insert, find minimum, delete minimum, decrease-key, and delete. Discuss how these complexities make Fibonacci heaps particularly suitable for graph algorithms like Dijkstra's and Prim's.
55. Define the structure of a binomial heap. How does it differ from a binary heap in terms of organization and properties?
56. Given two binomial heaps with the following roots:
    Heap 1: B0 with value 5, B1 with value 10
    Heap 2: B0 with value 3, B1 with value 8. Illustrate the process of merging these two heaps into a single binomial heap and show the resulting structure.
57. Explain the process of deleting the minimum element from a binomial heap. Given a binomial heap with the following root nodes: 2 (B1), 5 (B2), and 8 (B0), demonstrate how to perform the delete operation and illustrate the resulting heap.
58. Insert the following sequence of elements into a binomial heap: 4, 3, 7, 1, and 5. Show the heap after each insertion and explain any necessary merges or structural changes that occur.

59. What is the time complexity of the union operation in a binomial heap? Explain how the merging of binomial trees affects the overall structure of the heap and provides an example.
60. Analyze the space complexity of a binomial heap. How does the storage of nodes in different binomial trees contribute to its overall efficiency compared to other heap types?

Unit-5

1. Describe in detail a common NP-complete problem.
2. Compare and contrast min heaps and max heaps by outlining their key differences.
3. Examine Rabin Karp algorithm for string matching.
4. Explain how the satisfiability problem can be applied to solve real-world scenarios.
5. Describe the Naive String Matching Algorithm.

6. What is the Rabin-Karp Algorithm used for?

7. Explain the significance of polynomial-time reductions in the context of NP-Complete problems.

8. Describe an example of an NP-Complete problem and outline its characteristics.

9. How does amortized analysis help in evaluating the performance of a data structure?

10. What are approximation algorithms, and why are they important for solving NP-Complete problems?

11. Explain how the randomized Quick Sort algorithm differs from the standard Quick Sort.

12. What is the main idea behind the Vertex Cover Problem?

13. Given a set of cities with distances, demonstrate how to formulate the Travelling Salesperson Problem.

14. Apply amortized analysis to a specific data structure, such as a dynamic array, and illustrate the results.

15. Implement the Naive String Matching Algorithm on the string "abcdefg" to search for "cde". Show the steps.

16. Use the Rabin-Karp Algorithm to search for the substring "abc" in the string "abcdefghabc". Show the steps and results.

17. Solve a small instance of the Vertex Cover Problem and provide the reasoning for your choices.

18. Demonstrate how randomized Quick Sort can be implemented using a random pivot selection.

19. Analyze the time complexity of the Naive String Matching Algorithm in the worst-case scenario.
20. Compare and contrast NP-Complete and NP-Hard problems in terms of their definitions and examples.
21. Examine the effectiveness of approximation algorithms for the Travelling Salesperson Problem. What are the trade-offs?
22. Analyze the average-case time complexity of the randomized Quick Sort algorithm. How does it compare to the worst-case scenario?
23. Investigate the limitations of using naive approaches for solving NP-Complete problems.
24. Evaluate the performance of the Rabin-Karp Algorithm in terms of its average-case and worst-case complexities.
25. Critically assess the importance of recognizing NP-Complete problems in algorithm design. What are the implications?
26. Evaluate the effectiveness of different approximation algorithms for the Vertex Cover Problem. Which approach yields better results?
27. Discuss the benefits and drawbacks of using randomized algorithms in algorithm design.
28. Evaluate the significance of randomized Quick Sort in practical applications. When would it be preferred over deterministic sorting algorithms?
29. Assess the impact of approximation ratios in solving the Travelling Salesperson Problem. What are the implications for real-world applications?
30. Analyze how string matching algorithms can be optimized for specific types of data, such as DNA sequences.
31. Design a new approximation algorithm for the Travelling Salesperson Problem and outline its steps.
32. Create a hybrid algorithm that combines the strengths of the Naive String Matching and Rabin-Karp algorithms. Describe how it works.
33. Propose a novel method for solving the Vertex Cover Problem that leverages randomization.
34. Invent a new problem that is NP-Complete, and describe how it could be approached using known techniques.
35. Design a comprehensive tutorial for implementing the randomized Quick Sort algorithm, including examples and edge cases.
36. Create a case study that explores the application of NP-Complete problems in a specific industry, detailing the challenges and solutions.

37. How can understanding NP-Completeness influence the design of algorithms for complex problems?
38. Discuss the role of randomization in improving algorithm performance. What are some potential pitfalls?
39. Explore the connections between string matching algorithms and text processing applications.
40. What insights can be gained from analyzing the time and space complexities of the Rabin-Karp Algorithm compared to other string matching techniques?
41. How do approximation algorithms impact decision-making in real-world scenarios, particularly in resource allocation?
42. Discuss the importance of empirical analysis in evaluating the performance of algorithms, particularly for NP-Complete problems.
43. What are the implications of the P vs. NP problem on the study of algorithm design and analysis?
44. How can graph theory be applied to better understand the Vertex Cover Problem?
45. Evaluate the use of randomized algorithms in contemporary machine learning applications.
46. How can amortized analysis be applied to optimize algorithms in dynamic programming scenarios?
47. Discuss the potential future trends in the study of NP-Complete problems and their solutions.
48. Analyze how modern computational power affects the feasibility of solving NP-Complete problems.
49. Explore the ethical implications of using approximation algorithms in decision-making processes.
50. How does the field of bioinformatics benefit from advancements in string matching algorithms?
51. Analyze the distinction between NP, NP-Complete, and NP-Hard problems in detail. Provide examples to illustrate the differences.
52. Explain the Travelling Salesperson Problem (TSP) and describe a common approximation algorithm used to solve it. What is the performance guarantee of this algorithm?
53. Define NP-Complete and NP-Hard classes.
54. Analyze how asymptotically efficient algorithms leverage the properties of Fibonacci heaps.
55. Demonstrate that CLIQUE problem is NP-Complete.
56. Analyze the steps used to show a given problem is NP-Complete.

57. Conduct a detailed analysis of the Vertex Cover Problem. Discuss an approximation algorithm for solving the Vertex Cover Problem, including its approximation ratio, and evaluate the effectiveness of this algorithm in practice.

58. Explain the randomized Quick Sort algorithm and analyze its average-case time complexity. Discuss the advantages of randomization in this algorithm and evaluate how it contributes to improved performance.

59. What is the definition of NP-Complete problems?

60. Define NP-Hard problems and explain how they differ from NP-Complete problems.