Dezyne École College

Bachelor of Computer Application (B.C.A.) Third Year-5th Semester

Design Analysis & Algorithms

University Exam Probable Questions

UNIT 1

1. Define an algorithm. What are the characteristics of a good algorithm? Provide an example.
2. Differentiate between time complexity and space complexity with examples.
3. Explain Big-O, Big-Theta (Θ), and Big-Omega (Ω) notations with graphical illustrations.
4. Arrange the following functions in increasing order of growth rate: O(n),O(nlog□n),O(n2),O(2n),O(1)O(n), O(n\log n), O(n^2), O(2^n), O(1). Justify your answer.
5. Describe the importance of asymptotic analysis in algorithm design. Give examples where it helps in comparing algorithms.
6. Derive the time complexity of a recursive algorithm using recurrence relation: T(n) = T(n-1) + 1.
7. Analyze the time complexity of the following iterative function:

```
for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
        cout << i + j;
```

8. Give a recursive algorithm to compute factorial of a number. Perform its time complexity analysis.
9. Differentiate between analysis of recursive and non-recursive algorithms with examples.
10. Explain the brute force approach. What are its advantages and limitations?

11. Describe the working of Selection Sort with an example array. Write its time complexity.
12. Compare Bubble Sort and Selection Sort in terms of efficiency and stability.
13. Write pseudocode for Bubble Sort and analyze its best, worst, and average-case complexities.
14. Explain the Sequential Search technique. How is it different from Binary Search in terms of complexity?
15. Describe the Brute Force string matching algorithm. Illustrate its working with an example.
16. Explain the Divide and Conquer technique with a generic recursive structure. Give two real-world examples.
17. Solve the defective chessboard problem using Divide and Conquer strategy. Describe the algorithm.
18. Write a Divide and Conquer algorithm for Binary Search and derive its time complexity.
19. Implement Merge Sort algorithm and perform step-by-step dry run on the array: [38, 27, 43, 3, 9, 82, 10]. Analyze its time complexity.
20. Explain the Quick Sort algorithm. How does pivot selection affect performance? Derive its best, average, and worst-case time complexities.

UNIT 2

1. Explain the general approach of the Greedy Method. How does it differ from Dynamic Programming?
2. List and explain the characteristics an optimization problem must satisfy to be solved by the greedy method.
3. Discuss the greedy-choice property and optimal substructure with an example.
4. Solve the Fractional Knapsack Problem using Greedy strategy for the following data:
    - Items = 3; Weights = [10, 20, 30]; Profits = [60, 100, 120]; Capacity = 50.
5. Why does the Greedy method fail to solve the 0/1 Knapsack problem optimally? Illustrate with an example.
6. Describe the Job Sequencing with Deadlines problem. Provide an algorithm using the greedy approach.
7. Given a list of jobs with deadlines and profits, schedule them using a greedy method and calculate the maximum profit.
8. Differentiate between Kruskal's and Prim's algorithms. When is each more efficient?
9. Using Kruskal's algorithm, construct a Minimum Spanning Tree (MST) for the following weighted undirected graph:
   (Provide a graph if needed.)
10. Apply Prim's algorithm step-by-step on a graph and show the MST formed.
11. Explain the use of greedy strategy in Prim's algorithm. Why does it guarantee an optimal result?
12. Describe the Single-Source Shortest Path problem. Can it be solved using a greedy approach? Give an example.
13. Apply Dijkstra's algorithm on a given graph and find the shortest path from the source to all other vertices.
14. Explain the general structure of a dynamic programming solution. How is it different from Divide and Conquer?
15. Define overlapping subproblems and optimal substructure. How do these apply to Dynamic Programming problems?
16. Write the algorithm for Warshall's algorithm and explain its use in computing transitive closure.

17. Apply Floyd's algorithm to a given graph to find the shortest paths between all pairs of nodes.
18. Solve the 0/1 Knapsack Problem using dynamic programming for the following data:

- Items = 3; Weights = [10, 20, 30]; Profits = [60, 100, 120]; Capacity = 50.

19. Explain how the time and space complexity of 0/1 Knapsack is improved by using dynamic programming.
20. Describe how the Traveling Salesperson Problem is solved using Dynamic Programming. Give its recurrence relation and explain the approach.

UNIT 3

1. What is a lower-bound argument in algorithm analysis? Explain with an example.
2. Explain the role of decision trees in establishing lower bounds. Illustrate with comparison-based sorting.
3. Prove that any comparison-based sorting algorithm has a lower bound of Ω(n log n).
4. Define the classes P, NP, and NP-Complete. Give an example of each.
5. What is the significance of the P vs NP problem? Why is it important in algorithm design?
6. Explain the concept of polynomial-time reduction and how it is used to prove NP-completeness.
7. Give an example of an NP-complete problem and outline how it can be verified in polynomial time.
8. List the main challenges in designing numerical algorithms. How do issues like round-off error and convergence affect them?
9. Explain how algorithmic power is limited when solving certain mathematical problems like irrational roots or high-degree polynomials.
10. What is backtracking? How is it different from brute force search?
11. Solve the 4-Queens problem using backtracking. Show all intermediate steps.
12. Write an algorithm to solve the Hamiltonian Circuit Problem using backtracking.
13. Explain how the Subset Sum Problem is solved using backtracking. Provide an example with at least 4 elements.
14. Differentiate between backtracking and branch-and-bound techniques.
15. Describe the branch-and-bound method for solving the 0/1 Knapsack Problem. Show the solution tree.
16. Using branch-and-bound, solve a small-sized assignment problem and show pruning.
17. Explain how the Traveling Salesperson Problem (TSP) can be solved using branch-and-bound.
18. Why do we need approximation algorithms for NP-hard problems?
19. Describe an approximation algorithm for the Traveling Salesperson Problem. What is the approximation ratio?

20. Explain a greedy approximation approach to the 0/1 Knapsack Problem. When does it fail to give optimal results?