SEMESTER - VI EXAMINATION SYSTEM

Paper XIV: Algorithm Design Techniques — Comprehensive Question Bank

UNIT - I: Algorithm Specification, Asymptotic Analysis & Recurrences

Section: Questions of 1 Mark Each

Q1. What is pseudo-code?

Q2. Define the term 'Algorithm'.

Q3. What does time complexity measure?

Q4. What is space complexity?

Q5. State the Omega (Ω) notation definition.

Q6. Define Big-O (O) notation.

Q7. What is Theta (Θ) notation?

Q8. What is worst-case time complexity?

Q9. Name the input case for which Insertion Sort performs best.

Q10. Write the recurrence relation for Binary Search.

Q11. What is the primary advantage of the Divide and Conquer strategy?

Q12. Give an example of a divide-and-conquer algorithm.

Q13. What is a recurrence relation?

Q14. When is the Master Method not applicable?

Q15. What is the worst-case time complexity of Insertion Sort?

Q16. What is asymptotic analysis?

Q17. Define a recursion tree.

Q18. What is the best-case time complexity of Insertion Sort?

Q19. What does space complexity account for besides input data?

Q20. What is a basic operation in algorithm analysis?

Section: Questions of 2 Marks Each

Q1. Differentiate between time complexity and space complexity.

Q2. Explain the criteria for a well-defined algorithm.

Q3. Why do we prefer asymptotic analysis over empirical measurements?

Q4. Explain the difference between Big-O and little-o notation.

Q5. State the general form of a recurrence relation solved by the Master Theorem.

Q6. Describe the basic concept of the Substitution Method.

Q7. What is the principle of the Divide and Conquer paradigm?

Q8. Explain how Insertion Sort works in brief.

Q9. Show that f(n) = 3n + 2 is O(n).

Q10. What are the limitations of the Master Method?

Q11. Briefly describe how a recursion tree is formed.

Q12. Define average-case time complexity with an example.

Q13. What is the significance of the upper bound in algorithm analysis?

Q14. Why is the loop invariant useful in verifying Insertion Sort?

Q15. How does the division step in Divide and Conquer affect complexity?

Section: Questions of 5 Marks Each

Q1. Explain the different types of asymptotic notations (Big-O, Omega, Theta) with mathematical definitions and graphical representations.

Q2. Write the complete pseudo-code for Insertion Sort and analyze its best-case and worst-case time complexity.

Q3. Solve the recurrence relation T(n) = 2T(n/2) + n using the Substitution Method.

Q4. Illustrate the Recursion Tree Method by solving the recurrence relation T(n) = 3T(n/3) + n.

Q5. State and explain all three cases of the Master Theorem with suitable examples for each case.

Q6. Discuss the strategy of Divide and Conquer. How does it differ from a simple iterative approach? Give examples.

Q7. Analyze the space complexity of an iterative algorithm versus a recursive algorithm for the same problem.

Q8. Prove that the worst-case runtime of Insertion Sort is quadratic using summation notation.

Q9. Given the recurrence T(n) = T(n/3) + T(2n/3) + n, use a recursion tree to estimate the upper bound.

Q10. Discuss the role of constants and lower-order terms in asymptotic analysis and why they are ignored.

Section: Questions of 8 Marks Each

Q1. Provide a comprehensive formulation of the Master Method. Discuss its formal mathematical basis, outline the structural conditions under which it can be safely applied, and systematically analyze three distinct algorithmic problems that demonstrate each of its canonical cases.

Q2. Analyze the complete design and performance profiles of the Insertion Sort algorithm. Provide its structured pseudo-code, rigorously demonstrate its loop invariant proof of correctness, and derive exact mathematical expressions for its space and time complexities under best, worst, and average input configurations.

Q3. Formulate a detailed theoretical framework comparing the Substitution Method, the Recursion Tree Method, and the Master Theorem for solving recurrences. Apply each method to solve T(n) = 4T(n/2) + n^2, contrasting their analytical steps, human error-proneness, and conceptual elegance.

Q4. Explore the foundational pillars of Algorithm Specification and Asymptotic Analysis. Critically evaluate how hardware architectures, compiler optimizations, and language paradigms interact with theoretical space-time bounds, supporting your insights with formal proofs of foundational asymptotic properties.

Q5. Detail the architectural blueprint of the Divide and Conquer paradigm. Show how a complex computational problem is recursively decomposed, transformed, and consolidated, and analyze the theoretical impacts on memory overhead caused by deep activation stacks during runtime.

UNIT - II: Searching, Sorting & Hashing

Section: Questions of 1 Mark Each

Q1. What is the time complexity of Linear Search in the worst case?

Q2. State the best-case time complexity of Binary Search.

Q3. Which sorting algorithm uses a partitioning element called a pivot?

Q4. Is Merge Sort an in-place sorting algorithm?

Q5. What is the worst-case time complexity of Quick Sort?

Q6. What is a hash function?

Q7. Define collision in a hash table.

Q8. What is linear probing?

Q9. What is quadratic probing?

Q10. What is double hashing?

Q11. What is data structure heap?

Q12. State the average-case time complexity of Merge Sort.

Q13. What is chaining in collision resolution?

Q14. What is open addressing?

Q15. What is a max-heap?

Q16. Which sorting algorithm is based on the heap data structure?

Q17. What is the load factor of a hash table?

Q18. Name one advantage of hashing over direct searching.

Q19. What is a stable sorting algorithm?

Q20. What is the worst-case time complexity of Heap Sort?

Section: Questions of 2 Marks Each

Q1. Explain why Binary Search cannot be applied directly to a singly linked list.

Q2. Compare the space complexities of Merge Sort and Quick Sort.

Q3. What makes an algorithm an 'in-place' sorting algorithm? Give an example.

Q4. Describe the uniform hashing assumption.

Q5. Explain the main difference between linear probing and quadratic probing.

Q6. What is the role of a pivot element in Quick Sort?

Q7. How does the concept of a heap differ from a binary search tree?

Q8. Explain how clustering occurs in linear probing.

Q9. What are the characteristics of a good hash function?

Q10. Define the primary clustering problem in open addressing.

Q11. What is secondary clustering, and which technique resolves it?

Q12. How does double hashing eliminate primary and secondary clustering?

Q13. Explain the heapify process in short.

Q14. Why is the worst-case performance of Quick Sort O(n^2)?

Q15. What is a key collision, and why is it unavoidable in practical applications?

Section: Questions of 5 Marks Each

Q1. Explain the mechanism of Quick Sort with an example array. Derive its best-case and worst-case time complexity.

Q2. Describe the Merge Sort algorithm thoroughly. Provide its divide, conquer, and combine phases along with its complete recurrence relation analysis.

Q3. Explain the structure of a binary heap. Show how a given unordered array is transformed into a Max-Heap using the Heapify procedure.

Q4. What is Hashing? Explain the division method, multiplication method, and mid-square method of generating hash addresses.

Q5. Discuss collision resolution using Open Addressing. Explain Linear Probing and Quadratic Probing with an example for each.

Q6. Explain Chaining as a collision resolution mechanism. What are its advantages and disadvantages compared to Open Addressing?

Q7. Compare the operational characteristics and performance metrics of Linear Search and Binary Search.

Q8. Trace the step-by-step execution of Heap Sort on the following sequence: [12, 11, 13, 5, 6, 7].

Q9. Discuss the concept of double hashing. Show mathematically how the secondary hash function provides alternative positions.

Q10. Analyze the impact of the load factor (α) on the performance of search operations in both chaining and open addressing.

Section: Questions of 8 Marks Each

Q1. Develop an extensive comparative study of advanced sorting paradigms: Quick Sort, Merge Sort, and Heap Sort. Evaluate their algorithmic workflows, structural invariants, asymptotic runtimes across all cases, auxiliary memory footprints, and practical stability profiles.

Q2. Provide an exhaustive analysis of Hashing as an O(1) search paradigm. Mathematically formalize the phenomenon of key collisions, and present a rigorous comparative critique of Chaining versus Open Addressing architectures under variable load factors.

Q3. Deconstruct the Quick Sort partitioning mechanism. Provide detailed pseudo-code for Lomuto and Hoare partitioning schemes, formalize the mathematical proof of its O(n^2) worst-case performance, and explain how randomized pivot selection guarantees O(n log n) expected time.

Q4. Examine the architectural foundations of Heap Sort. Detail the mathematical properties of complete binary trees embedded in arrays, write comprehensive pseudo-code for the max-heapify and build-max-heap processes, and analyze its invariant O(n log n) time complexity.

Q5. Analyze the problem of clustering in open addressing. Conduct a detailed mathematical trace comparing Linear Probing, Quadratic Probing, and Double Hashing while inserting a conflicting sequence of keys into an initially empty hash table.

UNIT - III: Greedy Technique & Dynamic Programming

Section: Questions of 1 Mark Each

Q1. What is the core principle of a Greedy algorithm?

Q2. Does the Greedy method always yield an optimal solution?

Q3. What is the main optimization metric in the Fractional Knapsack problem?

Q4. Can items be broken into parts in the 0/1 Knapsack problem?

Q5. What is the objective of Job Sequencing with Deadlines?

Q6. What type of codes are generated by the Huffman algorithm?

Q7. What is the prefix-free property in Huffman Coding?

Q8. State the fundamental characteristic of Dynamic Programming.

Q9. What is Memoization?

Q10. What is the Bottom-Up approach in Dynamic Programming?

Q11. Define the Principle of Optimality.

Q12. What is the time complexity to find the optimal order of Matrix Chain Multiplication?

Q13. What does LCS stand for in algorithm design?

Q14. What is the main difference between Greedy and Dynamic Programming methods?

Q15. Name an application that uses Huffman Codes.

Q16. What is the optimal substructure property?

Q17. What is the overlapping subproblems property?

Q18. What is the time complexity of the Fractional Knapsack problem?

Q19. State the recurrence relation for the 0/1 Knapsack problem.

Q20. What data structure is typically used to implement Huffman coding efficiently?

Section: Questions of 2 Marks Each

Q1. Explain why the Greedy approach fails for the 0/1 Knapsack problem but works for Fractional Knapsack.

Q2. Define the concept of overlapping subproblems with a simple example.

Q3. What is the role of a greedy choice property in algorithm design?

Q4. Explain the significance of the Principle of Optimality in Dynamic Programming.

Q5. How does Memoization differ from Tabulation?

Q6. What is a prefix code, and why is it important in data compression?

Q7. Explain the objective of the Matrix Chain Multiplication problem.

Q8. What is the Longest Common Subsequence (LCS) problem?

Q9. Compare the space complexities of top-down versus bottom-up Dynamic Programming solutions.

Q10. Why can't Job Sequencing with Deadlines be solved optimally using an arbitrary sorting order?

Q11. How does Dynamic Programming avoid recomputing values?

Q12. What is the computational complexity of evaluating a matrix chain product of dimensions p x q and q x r?

Q13. Give a scenario where a Greedy algorithm provides a sub-optimal solution.

Q14. What is the meaning of a feasible solution versus an optimal solution?

Q15. How does the state table help in reconstructing the optimal solution in Dynamic Programming?

Section: Questions of 5 Marks Each

Q1. Explain the general method of the Greedy Technique. Outline the core stages an algorithm goes through to establish a greedy choice.

Q2. Solve the Fractional Knapsack problem for the following parameters: Knapsack capacity W = 50, Weights = [10, 20, 30], Values = [60, 100, 120].

Q3. Explain the Job Sequencing with Deadlines problem. Find an optimal sequence for jobs with profits [100, 19, 27, 25, 15] and deadlines [2, 1, 2, 1, 3].

Q4. Describe the Huffman Coding algorithm. Construct a Huffman Tree for characters with frequencies: a:45, b:13, c:12, d:16, e:9, f:5.

Q5. Explain the general principles of Dynamic Programming. Clearly distinguish it from the Divide and Conquer approach.

Q6. Formulate the Dynamic Programming recurrence relation for the Matrix Chain Multiplication problem and explain its parameters.

Q7. Discuss the 0/1 Knapsack problem using Dynamic Programming. Write its step-by-step algorithmic formulation.

Q8. Explain the Longest Common Subsequence (LCS) problem. Write the recurrence relation used to fill the LCS data matrix.

Q9. Trace the LCS algorithm for the sequences X = 'ABCBDAB' and Y = 'BDCABA'. Show the final matrix.

Q10. Compare Greedy and Dynamic Programming techniques based on efficiency, correctness proofs, and applicability.

Section: Questions of 8 Marks Each

Q1. Deconstruct the Matrix Chain Multiplication problem using Dynamic Programming. Provide the formal optimization recurrence, write a complete bottom-up tabulation algorithm, analyze its cubic time complexity, and trace the parenthesization matrix for a specific chain sequence.

Q2. Provide an authoritative analytical comparison between the Greedy Paradigm and Dynamic Programming. Evaluate how they exploit optimal substructure, contrast greedy choices against multi-stage decision policies, and back your analysis with a rigorous comparative study of 0/1 and Fractional Knapsack.

Q3. Formulate the Huffman Coding data compression standard. Detail the constructive greedy mechanism, prove its prefix-free character, write comprehensive pseudo-code for its execution using priority queues, and compute the total bits saved on a text corpus with skewed symbol distributions.

Q4. Systematically resolve the 0/1 Knapsack problem via Dynamic Programming. Define its state space representation, derive the optimal sub-structural recurrence equation, construct the complete bottom-up tabular execution grid, and present a backtracking algorithm to isolate the exact items included.

Q5. Examine the Job Sequencing with Deadlines problem under the Greedy framework. Detail the mathematical layout of deadlines and profit optimization, write its formal pseudo-code, prove its algorithmic correctness, and demonstrate its step-by-step execution on an array of conflicting jobs.

UNIT - IV: Graph Algorithms

Section: Questions of 1 Mark Each

Q1. What is a directed acyclic graph (DAG)?

Q2. Define a Minimum Spanning Tree (MST).

Q3. What is the objective of Topological Sorting?

Q4. Name a greedy algorithm used to find an MST.

Q5. Can Topological Sort be applied to a graph with cycles?

Q6. What is the time complexity of Dijkstra's algorithm using a binary heap?

Q7. Does Dijkstra's algorithm work correctly with negative edge weights?

Q8. Which algorithm can detect negative cycles in a graph?

Q9. What is the main advantage of the Bellman-Ford algorithm over Dijkstra's?

Q10. What is a single-source shortest path problem?

Q11. State the time complexity of Kruskal's algorithm.

Q12. What data structure is used to optimize Kruskal's algorithm?

Q13. What is the running time of Prim's algorithm using an adjacency matrix?

Q14. What is the relaxing step in shortest-path algorithms?

Q15. How many edges are present in a spanning tree of a graph with V vertices?

Q16. Is Topological Sort unique for a given DAG?

Q17. Which graph traversal technique can be modified to perform a Topological Sort?

Q18. What is the primary greedy choice made in Prim's algorithm?

Q19. What is the time complexity of the Bellman-Ford algorithm?

Q20. What does a negative cycle mean in a shortest-path context?

Section: Questions of 2 Marks Each

Q1. Explain why topological sorting is not possible for a graph containing a directed cycle.

Q2. State the cut property used as the foundation for Minimum Spanning Tree algorithms.

Q3. Contrast the basic expansion strategies of Prim's and Kruskal's algorithms.

Q4. Why does Dijkstra's algorithm fail when a graph contains negative edge weights?

Q5. Explain how the Bellman-Ford algorithm detects the presence of negative cycles.

Q6. Define a strongly connected component in a directed graph.

Q7. What is the role of the Disjoint-Set (Union-Find) data structure in Kruskal's algorithm?

Q8. Explain the term 'edge relaxation' with a mathematical expression.

Q9. What is the difference between a dense graph and a sparse graph in terms of algorithm selection?

Q10. Why does Bellman-Ford relax all edges exactly |V| - 1 times?

Q11. How can you check if a directed graph is a DAG?

Q12. Explain how Prim's algorithm handles disconnected graphs.

Q13. What is an source vertex and an sink vertex in topological sorting?

Q14. Can a graph have multiple Minimum Spanning Trees? If so, when?

Q15. What is the structural meaning of infinite distance infinity in shortest path algorithms?

Section: Questions of 5 Marks Each

Q1. Explain the concept of Topological Sorting. Describe Kahn’s algorithm (indegree-based) with suitable pseudo-code.

Q2. Describe Prim's algorithm for finding a Minimum Spanning Tree. Detail its step-by-step greedy choice mechanism.

Q3. Explain Kruskal's algorithm for finding an MST. Show how the Disjoint-Set data structure helps avoid cycles.

Q4. Detail Dijkstra's algorithm for single-source shortest paths. Provide its complete algorithmic steps.

Q5. Explain the Bellman-Ford algorithm for single-source shortest paths. Describe how it handles negative edge weights.

Q6. Trace Prim's algorithm on a 5-vertex weighted undirected graph starting from an arbitrary node.

Q7. Trace Kruskal's algorithm on a given edge-weighted graph, explicitly listing the sorted edge processing order.

Q8. Prove the correctness of the Bellman-Ford algorithm for graphs without negative cycles.

Q9. Show with a concrete graph example how Dijkstra's algorithm produces incorrect results given negative weights.

Q10. Compare the time and space complexity profiles of Prim's and Kruskal's algorithms for dense versus sparse graphs.

Section: Questions of 8 Marks Each

Q1. Provide an authoritative analysis of Single-Source Shortest Path (SSSP) paradigms. Present a detailed mathematical dissection of Dijkstra's and Bellman-Ford algorithms, contrast their internal edge relaxation policies, analyze their operational complexities, and formalize the proof of Bellman-Ford's negative cycle detection.

Q2. Deconstruct the theoretical principles behind Minimum Spanning Trees (MSTs). Provide extensive pseudo-code for Prim's and Kruskal's algorithms, mathematically validate their respective greedy choices via the Cut Property, and analyze their performance profiles utilizing advanced data structures like Fibonacci Heaps and Disjoint-Set Forests.

Q3. Explore the concept of Topological Sorting in Directed Acyclic Graphs (DAGs). Formalize its definition, present full algorithmic implementations for both the Depth-First Search (DFS) based approach and Kahn’s indegree-based queue paradigm, prove their linear-time operations, and detail their applications in system dependency resolution.

Q4. Conduct a rigorous architectural review of the Disjoint-Set (Union-Find) ADT. Show how its integration via path compression and union-by-rank optimizations directly affects the asymptotic runtime of Kruskal's algorithm, proving the near-linear amortized performance using the Inverse Ackermann function.

Q5. Analyze structural failures in network routing algorithms caused by edge anomalies. Design a complex edge-weighted graph containing negative arcs and cycles, and show step-by-step how Dijkstra's greed terminates prematurely while Bellman-Ford systematically stabilizes or triggers its cycle warning flag.