DAA-One Shot For Viva

1. MAX-MIN USING DIVIDE AND CONQUER

1. Time complexity: O(n)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Efficiently finds the maximum and minimum elements in an array using divide and conquer. Also, reduces the number of comparisons from 2(n-1) to (3n/2) – 2.

4. Applications: Used in various optimization problems where finding both maximum and minimum values is required.

2. STASSEN'S MATRIX MULTIPLICATION USING DIVIDE AND CONQUER

1. Time complexity: O(n^2.81) = O(n^log7)

2. Important case in time complexity: Matrix multiplication using Divide and Conquer has Time complexity O(n^3).

3. Anything special about the algorithm: Breaks down matrix multiplication into smaller subproblems to reduce computational complexity.
Generally, Strassen’s Method is not preferred for practical applications for following reasons.

- The constants used in Strassen’s method are high and for a typical application Naive method works better.
- For Sparse matrices, there are better methods especially designed for them.
- The submatrices in recursion take extra space.
- Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen’s algorithm than in Naive Method

4. Applications: Used in applications requiring fast matrix multiplication, such as computer graphics and numerical simulations.

3. QUICK SORT USING DIVIDE AND CONQUER

1. Time complexity: O(n log n) average case, O(n^2) worst case.

2. Important case in time complexity: If array is sorted than Time Complexity = O(n^2)

3. Anything special about the algorithm: Efficient sorting algorithm that recursively divides the array into smaller partitions based on a chosen pivot element.

4. Applications: Widely used for sorting large datasets efficiently in various programming languages and applications.

4. MERGE SORT USING DIVIDE AND CONQUER

1. Time complexity: O(n log n)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Divides the array into smaller subarrays, sorts them individually, and then merges them to produce a sorted output.

4. Applications: Used for sorting large datasets, particularly in scenarios where stability and predictable performance are important. Inversion counting (counting the number of inversions in an array). Finding the median of an array

5. SINGLE SOURCE SHORTEST PATH (DIJKSTRA'S ALGORITHM)

1. Time complexity: O((V + E) log V) using binary heap.

2. Important case in time complexity: If simple array is used, Time Complexity = O(n^2).

3. Anything special about the algorithm: Finds the shortest path from a single source vertex to all other vertices in a weighted graph.

4. Applications: Used in various routing algorithms, network design, and optimization problems in transportation and logistics.

6. ACTIVITY SELECTION PROBLEM

1. Time complexity: O(n log n)

2. Important case in time complexity: If list is sorted then O(n).

3. Anything special about the algorithm: Greedy algorithm that selects the maximum number of non-overlapping activities.

4. Applications: Scheduling tasks, resource allocation, and job sequencing problems.

7. FRACTIONAL KNAPSACK PROBLEM

1. Time complexity: O(n log n)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Greedy algorithm that selects items based on their value-to-weight ratio.

4. Applications: Optimizing resource allocation problems where items have fractional weights and values, such as in finance and resource management.

8. PRIM'S ALGORITHM

1. Time complexity: O((V + E) log V) using Fibonacci heap.

2. Important case in time complexity: O(E*log V) using binary heap and O(V^2) using adjacency matrix.

3. Anything special about the algorithm: Finds the minimum spanning tree in a connected, undirected graph. Works on Vertices. The tree that we are making is always connected.

4. Applications: Network design, clustering, and optimization problems in various fields including telecommunications and transportation.

9. JOB SEQUENCING WITH DEADLINES

1. Time complexity: O(n^2)

2. Important case in time complexity: O(n*log n) using Priority-Queue (Max-Heap)

3. Anything special about the algorithm: Greedy algorithm that schedules jobs to maximize profit, considering their deadlines.

4. Applications: Production scheduling, project management, and task assignment in various industries. Just for Deep in Pink Color :
Airlines Scheduling : Optimizing crew, flight and gate assignments to reduce delays.

10. MATRIX CHAIN MULTIPLICATION

1. Time complexity: O(n^3)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Finds the most efficient way to multiply a series of matrices.

4. Applications: Computer graphics, numerical simulations, and optimization problems in engineering and finance.

11. ALL PAIR SHORTEST PATH

1. Time complexity: O(V^3) using Floyd-Warshall algorithm.

2. Important case in time complexity: If we use Dijkstra for each node then O(V^3*log v) and If we use Bellman Ford for each node then O(V^4).

3. Anything special about the algorithm: Finds the shortest path between all pairs of vertices in a weighted graph.

4. Applications: Routing algorithms, network analysis, and optimization problems in transportation and logistics.

12. 0/1 KNAPSACK PROBLEM

1. Time complexity: O(n*W), where n is the number of items and W is the capacity of the knapsack.

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Dynamic programming approach to maximize the value of items selected, considering their weights and values.

4. Applications: Resource allocation, financial portfolio optimization, and project selection.

13. COIN CHANGE PROBLEM USING GREEDY AND DYNAMIC PROGRAMMING

1. Time complexity:

   - Greedy approach: O(n)

   - Dynamic programming approach: O(n*amount)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Greedy approach may not always give the optimal solution, while dynamic programming guarantees optimality.

4. Applications: Making change, optimization problems in finance, and resource allocation.

14. LONGEST COMMON SUBSEQUENCE (LCS)

1. Time complexity: O(m * n)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Finds the longest subsequence common to two sequences.

4. Applications: Bioinformatics, text comparison, and version control systems.

15. BELLMAN-FORD ALGORITHM

1. Time complexity: O(V*E)

2. Important case in time complexity: If E = V^2 then O(V^3).

3. Anything special about the algorithm: Finds the shortest path from a single source vertex to all other vertices in a weighted graph, even with negative edge weights.

4. Applications: Network routing algorithms, distance-vector routing protocols, and resource allocation.

16. OPTIMAL BINARY SEARCH TREE (OBST)

1. Time complexity: O(n^3)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Constructs a binary search tree that minimizes the expected search cost for a given set of keys and their probabilities.

4. Applications: Database indexing, compiler optimizations, and information retrieval systems.

17. HAMILTONIAN CYCLE USING BACKTRACKING

1. Time complexity: O(n!)

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Backtracking algorithm that finds a cycle that visits every vertex exactly once in a graph.

4. Applications: Traveling salesman problem, circuit layout design, and job scheduling.

18. GRAPH COLORING USING BACKTRACKING

1. Time complexity: O(m^V) where m is number of colors.

2. Important case in time complexity: No special cases.

3. Anything special about the algorithm: Backtracking algorithm that assigns colors to vertices in a graph such that no two adjacent vertices have the same color.

4. Applications: Register allocation in compilers, scheduling problems, and frequency assignment in wireless networks.

19. N-QUEEN PROBLEM USING BACKTRACKING

1. Time complexity: O(N!)
2. Important case in time complexity: No special cases.
3. Anything special about the algorithm: Backtracking algorithm that finds all possible placements of N queens on an N×N chessboard such that no two queens attack each other.
4. Applications: Puzzle-solving, constraint satisfaction problems, and combinatorial optimization.

20. RABIN-KARP ALGORITHM

1. Time complexity: O(n + m) average case, where n is the length of the text and m is the length of the pattern.
2. Important case in time complexity: No special cases.
3. Anything special about the algorithm: String searching algorithm that efficiently finds patterns in text using hashing.
4. Applications: Plagiarism detection, DNA sequencing, and content-based searching.

21. NAIVE STRING MATCHING

1. Time complexity: O((n-m+1)*m) worst case, where n is the length of the text and m is the length of the pattern.
2. Important case in time complexity: Best Case O(n).
3. Anything special about the algorithm: Simple algorithm that checks for a match between a pattern and all substrings of a text.
4. Applications: Basic string searching, small-scale text processing, and educational purposes.

22. KNUTH-MORRIS-PRATT (KMP) ALGORITHM

1. Time complexity: O(n + m) average case, where n is the length of the text and m is the length of the pattern.
2. Important case in time complexity: No special cases.
3. Anything special about the algorithm: String searching algorithm that efficiently finds patterns in text by avoiding unnecessary character comparisons.
4. Applications: String searching in large texts, bioinformatics, and data mining.