DAA IMPORTANT QUESTIONS

1. Types of Algorithms Commonly Used in Computer Science:

- Sorting Algorithms: QuickSort, MergeSort, Bubble Sort, Insertion Sort
- Search Algorithms: Binary Search, Depth-First Search (DFS), Breadth-First Search (BFS)
- Graph Algorithms: Dijkstra's Algorithm, Bellman-Ford Algorithm, Kruskal's Algorithm, Prim's Algorithm
- Dynamic Programming Algorithms: Fibonacci Sequence, Knapsack Problem, Longest Common Subsequence
- Greedy Algorithms: Fractional Knapsack, Huffman Coding, Activity Selection
- Divide and Conquer Algorithms: MergeSort, QuickSort, Binary Search
- Backtracking Algorithms: N-Queens Problem, Sudoku Solver
- Brute Force Algorithms: Traveling Salesman Problem, String Matching

2. What is an Algorithm, and What Are Its Primary Properties?

- Algorithm: A finite sequence of well-defined instructions to solve a specific problem or perform a computation.
- Primary Properties:
    - Correctness: The algorithm produces the correct output for all valid inputs.
    - Efficiency: The algorithm performs the task within a reasonable amount of time and space.
    - Finiteness: The algorithm terminates after a finite number of steps.
    - **Definiteness**: Each step of the algorithm is precisely defined.
    - Input and Output: The algorithm receives input and produces output.

3.How do you determine the lower, upper, and Tight bounds of an algorithm?

Determining Lower, Upper, and Tight Bounds of an Algorithm:

- **Lower Bound (Ω):** The minimum time complexity that any algorithm can take to solve a problem.
- **Upper Bound (O):** The maximum time complexity that the algorithm will take in the worst case.

- **Tight Bound (Θ):** When an algorithm's upper and lower bounds are asymptotically the same, representing the precise time complexity.

4. What are the key parameters to consider when writing an algorithm analysis?

Key Parameters to Consider When Writing an Algorithm Analysis:

- **Time Complexity:** How the execution time grows with the input size.
- **Space Complexity:** The amount of memory the algorithm uses relative to the input size.
- **Correctness:** Ensuring the algorithm produces the expected output.
- **Scalability:** How well the algorithm performs as the input size increases.
- **Optimality:** Whether the algorithm provides the best possible solution.

5.In the Fractional Knapsack problem, how does the Greedy algorithm choose items?

- The greedy algorithm chooses items based on the highest value-to-weight ratio until the knapsack is full.

6.Repeated - Key Parameters to Consider When Writing an Algorithm Analysis:

- (Already answered in question 4)

7.How do you analyze control statements and loop invariants to ensure the correctness of an algorithm

- **Control Statements:** Verify that conditional and iterative structures correctly manage the algorithm's flow.
- **Loop Invariants:** Conditions that hold true before and after each iteration of a loop, ensuring the algorithm maintains correctness throughout execution.

8.How does the master theorem apply to solve divide-and-conquer recurrences of the form T(n)=aT(n/b)+f(n))?

Master Theorem for Divide-and-Conquer Recurrences (T(n) = aT(n/b) + f(n)):

- Case 1: If $f(n) = O(n^c)$ where $c < \log_b a$, then $T(n) = O(n^{\log_b a})$
- Case 2: If $f(n) = O(n^c)$ where $c = \log_b a$, then $T(n) = O((n^{\log_b a}) \log n)$
- Case 3: If $f(n) = O(n^c)$ where $c > \log_b a$, then $T(n) = O(f(n))$

9. Which of the following problems is typically solved using a greedy algorithm?

a) Traveling Salesman Problem b) Knapsack Problem (0/1)
c) Minimum Spanning Tree d) N-Queens Problems

- c) Minimum Spanning Tree

10. What is the primary use of Huffman coding?
a) Sorting data b) Searching data
c) Data compression d) Data encryption

- **c) Data compression**

11.What does "stable sorting algorithm" mean? a) The algorithm never crashes b) The relative order of equal elements is preserved c) The performance is consistent across all inputs d) The algorithm uses minimal memory

- b) The relative order of equal elements is preserved

12. What is the time Complexity of Kruskal's Algorithm for Finding the Minimum Spanning Tree:

- a) O(E log V)

13. **Fractional Knapsack Problem - Greedy Algorithm Item Selection:**

- c) By value-to-weight ratio

14. Sorting Algorithm Following the "Divide and Conquer" Paradigm:

- MergeSort, QuickSort

15. **Type of Complexity Providing the Best-Case Scenario for an Algorithm's Performance:**

- Best-case complexity

16. Characteristic of Divide and Conquer Algorithms:

- Breaking a problem into smaller subproblems, solving each subproblem recursively, and combining their solutions.

17. Binary Search Algorithm Maximum Comparisons for Element Not Found:

- The maximum number of comparisons required is $\log_2$**n**

18. Algorithm for Finding Shortest Path from Single Source Vertex in Weighted Graph with Non-Negative Edge Weights:

- Dijkstra's Algorithm

19. Greedy Algorithm for Coin Change Problem with Denominations 1, 3, and 4:

- Choose the largest denomination coin which is not greater than the remaining amount.

20. Problem Not Directly Solved Using a Greedy Strategy in a Weighted Undirected Graph:

- Minimum Cut Problem (implied from the context as others can be solved using greedy algorithms)

Fill in the blanks

1. The Max-Min problem aims to find the maximum and minimum elements in an array with the minimum number of **comparisons**.
2. Greedy algorithms make the **locally optimal** choice at each step with the hope of finding the **global optimal** solution.
3. The **best** case represents the least time taken by an algorithm, while the **worst** case represents the maximum time.
4. The greedy solution to the fractional knapsack problem involves taking items in order of their **highest** value per **weight**.
5. Big Oh (O) notation describes the **upper** bound of an algorithm's time complexity.
6. Huffman coding is a greedy algorithm used for lossless **data** compression.
7. The primary properties of an algorithm include **input**, **output**, finiteness, definiteness, and effectiveness.
8. Dijkstra's algorithm finds the shortest path from a starting node to all other nodes in a graph with **non-negative** edge weights.

9. Strassen's matrix multiplication is applicable to **matrix** multiplication.
10. The recurrence relation for the time complexity of the Merge Sort algorithm is T(n)=2T(n/2)+∗∗O(n)∗∗T(n) = 2T(n/2) + **O(n)**T(n)=2T(n/2)+∗∗O(n)∗∗.
11. Quick sort selects a **pivot** element, partitions the array, and recursively sorts the subarrays.
12.The notation O(n)O(n)O(n) is used to describe the **worst** case time complexity of an algorithm.
13. The time required by an algorithm to complete as a function of the size of the input is referred to as its **time** complexity.
14. Performance profiling is a dynamic program analysis that measures where a program spends its time or which functions consume the most resources.
15. Benchmarking involves running a program or algorithm on a set of **standardized** tasks or inputs to evaluate its performance.
16. The conquer step in the Max-Min problem involves finding the maximum and minimum elements in each half of the **array**.
17. Huffman coding is a method of **lossless** data compression that uses variable-length codes to represent symbols with shorter codes for more frequent symbols.
18. The Knapsack Problem involves selecting a subset of items to maximize the total **value**, while not exceeding a given weight constraint.
19. Dijkstra's algorithm is particularly useful for finding the shortest path in a **weighted** graph.
20. In the Activity Selection Problem, a greedy algorithm selects activities based on their **finish** time.