Problem 1: Customer Order Sorting System

Algorithm: Merge Sort

Concept

Merge Sort is a divide-and-conquer algorithm that recursively divides the array into halves, sorts them, and merges them back together. It guarantees O(n log n) time complexity, making it efficient for sorting large datasets of customer orders by timestamp.

Input Format

```
n (number of orders)
id1 timestamp1
id2 timestamp2
...
idn timestampn
```

Sample Input

```
5
101 1609459200
102 1609455600
103 1609462800
104 1609457400
105 1609460000
```

Sample Output

```
102 1609455600
104 1609457400
101 1609459200
105 1609460000
103 1609462800
```

Time Complexity

- **Best Case:** O(n log n)
- **Average Case:** O(n log n)
- **Worst Case:** O(n log n)
- **Space Complexity:** O(n)

Problem 2: Movie Recommendation System

Algorithm: Quick Sort

Concept

Quick Sort is a divide-and-conquer algorithm that selects a pivot element and partitions the array around it. It recursively sorts the sub-arrays. Quick Sort is efficient with an average time complexity of O(n log n), ideal for sorting movies by various criteria like rating, year, or popularity.

Input Format

```
n criteria (1=rating, 2=year, 3=watch_time)
id1
name1
rating1 year1 watch_time1
...
```

Sample Input

```
5 1
101
The Shawshank Redemption
9.3 1994 15000000
102
The Godfather
9.2 1972 12000000
103
The Dark Knight
9.0 2008 20000000
```

Sample Output

```
101 The Shawshank Redemption 9.3 1994 15000000
102 The Godfather 9.2 1972 12000000
103 The Dark Knight 9.0 2008 20000000
```

Time Complexity

- **Best Case:** O(n log n)
- **Average Case:** O(n log n)
- **Worst Case:** O(n^2)
- **Space Complexity:** O(log n)

Problem 3: Emergency Relief Supply Distribution

Algorithm: Fractional Knapsack

Concept

The Fractional Knapsack problem allows taking fractions of items to maximize value within a weight constraint. Items are sorted by value-per-weight ratio, and greedily selected starting with the highest ratio. This greedy approach guarantees an optimal solution for the fractional variant.

Input Format

```
n capacity
id1
name1
weight1 value1
...
```

Sample Input

```
5 50
1
Medicine Kit
10 60
2
Food Packets
20 100
3
Drinking Water
30 120
```

Sample Output

```
Maximum Utility Value: 240.00

Items Selected:
ID  Name          Weight  Value  Fraction
1   Medicine Kit  10      60     1.00
2   Food Packets  20      100    1.00
3   Drinking Water 30     120    0.67
```

Time Complexity

- **Time Complexity:** O(n log n) [due to sorting]
- **Space Complexity:** O(1)

Problem 4: Smart Traffic Management

Algorithm: Dijkstra's Algorithm

Concept

Dijkstra's algorithm finds the shortest path from a source vertex to all other vertices in a weighted graph with non-negative edges. It uses a priority queue to repeatedly extract the vertex with minimum distance and relaxes adjacent edges. Perfect for finding optimal ambulance routes.

Input Format

```
n m source (vertices, edges, source)
u1 v1 weight1
u2 v2 weight2
...
```

Sample Input

```
6 9 0
0 1 4
0 2 2
1 2 1
1 3 5
2 3 8
2 4 10
3 4 2
3 5 6
4 5 3
```

Sample Output

```
Shortest distances from ambulance location 0:
Hospital  Distance  Path
0         0         0
1         3         0 -> 2 -> 1
2         2         0 -> 2
3         8         0 -> 2 -> 1 -> 3
4         10        0 -> 2 -> 1 -> 3 -> 4
5         13        0 -> 2 -> 1 -> 3 -> 4 -> 5
```

Time Complexity

- **Time Complexity:** O((V + E) log V) with binary heap
- **Space Complexity:** O(V)

Problem 5: Multi-Stage Delivery Route Optimization

Algorithm: Dynamic Programming on DAG

Concept

Multi-stage graph optimization uses dynamic programming to find the shortest path through a directed acyclic graph with distinct stages. Each node belongs to a specific stage, and edges only go forward. The DP approach processes nodes in topological order, ensuring optimal substructure.

Input Format

```
n m stages
stage1 stage2 ... stagen (stage for each node)
u1 v1 cost1
u2 v2 cost2
...
```

Sample Input

```
8 13 4
0 1 1 2 2 2 3 3
0 1 2
0 2 3
0 3 6
1 4 4
1 5 5
2 4 2
```

Sample Output

```
Minimum delivery cost: 12
Optimal route: 0 -> 2 -> 4 -> 6

Stage breakdown:
Node 0 (Stage 0) -> Node 2 (Stage 1)
-> Node 4 (Stage 2) -> Node 6 (Stage 3)
```

Time Complexity

- **Time Complexity:** O(V + E)
- **Space Complexity:** O(V)

Problem 6: Disaster Relief Resource Allocation

Algorithm: 0/1 Knapsack (Dynamic Programming)

Concept

The 0/1 Knapsack problem requires selecting items to maximize value without exceeding capacity, where each item must be either taken completely or left. Dynamic programming builds a table where dp[i][w] represents the maximum value achievable with first i items and weight limit w.

Input Format

```
n capacity
id1
name1
weight1 value1 priority1
...
```

Sample Input

```
7 50
1
Medicine Kit
15 100 1
2
Antibiotics
8 80 1
7
First Aid
5 50 1
```

Sample Output

```
Maximum Utility Value: 300
Total Weight Used: 48 kg (Capacity: 50 kg)

Items Selected:
ID  Name          Weight  Value  Priority
1   Medicine Kit  15      100    High
2   Antibiotics   8       80     High
7   First Aid     5       50     High
3   Food Packets  20      60     Medium
```

Time Complexity

- **Time Complexity:** O(n × W)
- **Space Complexity:** O(n × W)

Problem 7: University Timetable Scheduling

Algorithm: Graph Coloring (Greedy & DSATUR)

Concept

Graph coloring assigns colors to vertices such that no adjacent vertices share the same color. DSATUR (Degree of Saturation) selects vertices with maximum saturation degree (number of different colors used by adjacent vertices), providing better coloring than simple greedy in most cases.

Input Format

```
n m algorithm (1=Greedy, 2=DSATUR)
course_name1
course_name2
...
u1 v1 (conflict edges)
u2 v2
...
```

Sample Input

```
6 7 2
Data Structures
Algorithms
Database Systems
Operating Systems
Computer Networks
Software Engineering
0 1
0 2
0 3
```

Sample Output

```
Using DSATUR Coloring Algorithm

Minimum exam slots needed: 3

Time Slot 1:
  Course 0: Data Structures
  Course 4: Computer Networks

Time Slot 2:
  Course 1: Algorithms
  Course 3: Operating Systems
```

Time Complexity

- **Greedy:** O(V + E)
- **DSATUR:** $O(V^2)$
- **Space Complexity:** O(V + E)

Problem 8: N-Queens Problem

Algorithm: Backtracking

Concept

The N-Queens problem places N chess queens on an N×N chessboard such that no two queens attack each other. Using backtracking, the algorithm systematically tries placing queens row by row, checking constraints at each step. If a placement violates constraints (same row, column, or diagonal), it backtracks and tries alternative positions.

Input Format

```
N (size of chessboard)
```

Sample Input

```
8
```

Sample Output

```
N-Queens Problem (N = 8)
Total solutions found: 92

Showing first 4 solutions:

Solution 1:
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
Queen positions (row, col): (0,0) (1,4) (2,7) (3,5) (4,2) (5,6) (6,1) (7,3)
```

Time Complexity

- **Time Complexity:** O(N!)
- **Space Complexity:** O(N^2)