**Greedy Algorithm Problems for Beginners**

(HackWithInfy Round 1 Style)

1. **Easy 1: Minimum Coins**

**Problem Description:**  
You are given an amount and a list of coin denominations. Your task is to find the minimum number of coins required to make up the given amount using a greedy approach (assume denominations allow greedy solution to work correctly).

**Constraints:**

* 1 <= amount <= 10^6
* 1 <= coin\_i <= amount
* Coins are provided in descending order.

**Input Format:**

amount

coin1 coin2 coin3 ... (space separated)

**Output Format:**

Minimum coins needed

**Sample:**

Input:

43

10 5 2 1

Output:

6

**Explanation:**  
Coins used: 4×10 + 1×2 + 1×1 = 43 (total coins = 6).

**Edge Cases:**

* Amount = 0 → Output should be 0
* Amount smaller than smallest coin → Must return -1 if no solution (optional variant)

1. **Easy 2: Activity Selection**

**Problem Description:**  
Given start and end times of activities, select the maximum number of non-overlapping activities. Activities are selected based on earliest finishing time.

**Constraints:**

* 1 <= n <= 10^4
* 0 <= start\_i < end\_i <= 10^6

**Input Format:**

n

start1 end1

start2 end2

...

**Output Format:**

Maximum number of activities

**Sample:**

Input:

6

1 2

3 4

0 6

5 7

8 9

5 9

Output:

4

**Explanation:**  
Selected: (1,2), (3,4), (5,7), (8,9)

**Edge Cases:**

* Activities completely overlapping → Only one can be selected
* All activities non-overlapping → All can be selected

1. **Medium 1: Fractional Knapsack**

**Problem Description:**  
Given n items with their values and weights, maximize the total value in a knapsack of capacity W. You can take fractional parts of items.

**Constraints:**

* 1 <= n <= 10^4
* 1 <= W <= 10^6
* 1 <= value\_i, weight\_i <= 10^4

**Input Format:**

W

n

value1 weight1

value2 weight2

...

**Output Format:**

Maximum value (rounded to 2 decimal places)

**Sample:**

Input:

50

3

60 10

100 20

120 30

Output:

240.00

**Explanation:**  
Take full items 1 & 2, and 2/3 of item 3.

**Edge Cases:**

* W = 0 → Maximum value is 0
* All items too heavy → Only fractions can be taken

1. **Medium 2: Train Platforms**

**Problem Description:**  
Given arrival and departure times of trains, find the minimum number of platforms required so that no train waits.

**Constraints:**

* 1 <= n <= 10^4
* Time format: HHMM (e.g., 900 for 9:00 AM)

**Input Format:**

n

arrival1 departure1

arrival2 departure2

...

**Output Format:**

Minimum number of platforms required

**Sample:**

Input:

6

900 910

940 1200

950 1120

1100 1130

1500 1900

1800 2000

Output:

3

**Explanation:**  
Maximum overlap occurs when 3 trains are present simultaneously.

**Edge Cases:**

* All trains arriving and leaving at the same time → Need n platforms
* Completely non-overlapping → Only one platform needed

1. **Hard 1: Job Sequencing with Deadlines**

**Problem Description:**  
You are given n jobs with deadlines and profits. Each job takes exactly 1 unit of time. Schedule jobs to maximize total profit. Jobs must be completed before their deadline.

**Constraints:**

* 1 <= n <= 10^4
* 1 <= deadline\_i <= 1000
* 1 <= profit\_i <= 10^4

**Input Format:**

n

deadline1 profit1

deadline2 profit2

...

**Output Format:**

Number of jobs scheduled

Total profit

**Sample:**

Input:

4

4 20

1 10

1 40

1 30

Output:

2

60

**Explanation:**  
Select jobs with highest profit that can be completed on time.

**Edge Cases:**

* Deadlines all 1 → Only one job can be selected
* Deadlines sufficiently large → Can schedule all jobs

1. **Hard 2: Car Refueling Problem**

**Problem Description:**  
You need to travel to a target distance starting with initial fuel. Along the way, there are gas stations at certain distances. Find the minimum number of refuels required to reach the destination. If not possible, return -1.

**Constraints:**

* 1 <= target <= 10^9
* 0 <= start\_fuel <= target
* 0 <= station\_i <= target
* 1 <= station\_fuel\_i <= 10^4

**Input Format:**

target

start\_fuel

n

position1 fuel1

position2 fuel2

...

**Output Format:**

Minimum number of refuels (or -1 if impossible)

**Sample:**

Input:

100

10

4

10 60

20 30

30 30

60 40

Output:

2

**Explanation:**  
Optimal stops: Station at 10 (add 60 fuel), station at 60 (add 40 fuel).

**Edge Cases:**

* No stations → If start fuel is enough, return 0; else -1
* Stations at same location → Handle duplicates

**📚 Usage Guide:**

1. Start solving from Easy to Hard problems.
2. Read input via stdin, output answer via stdout.
3. Validate using provided edge cases and create your own.

**💡 Pro Tips:**

* Always **sort** where necessary (activities, coins, etc.).
* Use **heaps** (priority queues) in job sequencing and car refueling.
* Focus on **greedy choice property and optimal substructure** to justify greedy solutions.

If you want, I can help you create:

* Starter templates in Python/C++ for each problem.
* Detailed solution approaches.
* Mock test format with timer.

Let me know!