**🛰️ Challenge Name**

**Packet Router Simulation**

**📝 Description**

Simulate a memory-limited packet router handling add, forward, and query operations efficiently.

**🔍 Problem Statement**

A network router maintains a queue of packets with a fixed memory capacity. Each packet has a source, destination, and a timestamp. The router must perform operations such as adding packets, forwarding the oldest one, and querying the number of packets sent to a destination within a time window. Duplicate packets must be ignored, and if the router reaches its memory limit, the oldest packet must be removed to make space.

You are to implement the function runRouterCommands(commands, parameters) that processes a list of router commands and returns the result of each operation.

**📥 Input Format**

* commands: A list of strings representing commands ("Router", "addPacket", "getCount", "forwardPacket").
* parameters: A list of lists corresponding to parameters for each command.

**Command Behaviors:**

* "Router": Takes one integer [limit] for memory capacity.
* "addPacket": Takes [source: str, destination: str, timestamp: int].
* "getCount": Takes [destination: str, startTime: int, endTime: int].
* "forwardPacket": Takes an empty list [].

**🚦 Constraints**

* 1 ≤ number of commands ≤ 10^4
* 1 ≤ memory limit ≤ 1000
* 1 ≤ timestamp ≤ 10^9
* All strings are alphanumeric and ≤ 10 characters
* Timestamps are strictly increasing over time

**📤 Output Format**

Return a list containing the result of each executed command:

* For "addPacket": return True if added, False if duplicate
* For "getCount": return integer count
* For "forwardPacket": return list [source, destination, timestamp] or empty list []

**🏷️ Tags**

queues, hashing, simulation, deques, networking, memory-management

Test case 1:

Input:

commands = ["Router", "addPacket", "addPacket", "addPacket", "getCount", "forwardPacket", "getCount", "getCount"]

parameters = [[2], [1, 2, 50], [2, 1, 60], [1, 2, 50], [2, 50, 70], [], [1, 50, 70], [2, 60, 60]]

Output:

[True, True, False, 1, [1, 2, 50], 1, 0]

Test case 2:

Input:

commands = ["Router", "addPacket", "addPacket", "getCount", "forwardPacket", "forwardPacket", "forwardPacket"]

parameters = [[3], [5, 6, 100], [5, 6, 101], [6, 100, 101], [], [], []]

Output:

[True, True, 2, [5, 6, 100], [5, 6, 101], []]

Test case 3:

Input:

commands = ["Router", "addPacket", "addPacket", "addPacket", "getCount", "addPacket", "getCount"]

parameters = [[2], [1, 2, 10], [3, 4, 20], [5, 6, 30], [4, 10, 30], [3, 4, 20], [4, 20, 20]]

Output:

[True, True, True, 1, False, 1]

Test case 4:

Input:

commands = ["Router", "addPacket", "addPacket", "getCount", "addPacket", "getCount", "forwardPacket", "forwardPacket"]

parameters = [[2], [9, 1, 5], [10, 1, 6], [1, 5, 6], [11, 2, 7], [1, 5, 6], [], []]

Output:

[True, True, 2, True, 1, [10, 1, 6], [11, 2, 7]]

Test case 5:

Input:

commands = ["Router", "addPacket", "addPacket", "addPacket", "forwardPacket", "addPacket", "getCount"]

parameters = [[2], [1, 1, 1], [2, 2, 2], [3, 3, 3], [], [2, 2, 2], [2, 2, 2]]

Output:

[True, True, True, [2, 2, 2], True, 1]

**🧮 Challenge Name**

**Prime Gap Balanced Subarrays**

**📝 Description**

Count subarrays where prime numbers are tightly grouped within a given gap limit.

**🔍 Problem Statement**

You're given a list of integers representing an array and a positive integer k. A subarray is considered *prime gap balanced* if it contains at least two prime numbers and the difference between the maximum and minimum prime in that subarray is less than or equal to k.

Implement the function count\_prime\_gap\_balanced\_subarrays(nums, k) which returns the total number of such subarrays.

**📥 Input Format**

* nums: List of integers (length ≤ 5000), each in the range [1, 50,000]
* k: Integer gap threshold (1 ≤ k ≤ 5000)

**🚦 Constraints**

* 1 ≤ len(nums) ≤ 5000
* 1 ≤ nums[i] ≤ 50000
* 1 ≤ k ≤ 5000
* Prime check should consider all primes up to 50,000
* Subarrays must contain **at least two distinct primes**

**📤 Output Format**

* Return a single integer representing the number of prime gap balanced subarrays.

**🏷️ Tags**

primes, subarrays, brute-force, sliding-window, number-theory, range-analysis

Test case 1:

Input:

nums = [1, 2, 3]

k = 1

Output:

2

Test case 2:

Input:

nums = [2, 3, 5, 7]

k = 3

Output:

4

Test case 3:

Input:

nums = [4, 6, 8, 10]

k = 5

Output:

0

Test case 4:

Input:

nums = [2, 11, 17, 29]

k = 5

Output:

0

Test case 5:

Input:

nums = [1, 2, 4, 3, 5, 6]

k = 2

Output:

6

**📞 Challenge Name**

**Number of Recent Calls**

**📝 Description**

Track how many requests happened in the last 3000 milliseconds.

**🔍 Problem Statement**

You are building a time-based request handler for an online service. Each time a client sends a request, a timestamp (in milliseconds) is recorded using the ping(t) method. The goal is to count how many requests occurred in the last 3000 milliseconds, including the current one.

The timestamp t will always be strictly increasing. Implement the function ping(t) that returns the number of recent requests.

**📥 Input Format**

* A series of function calls to ping(t)
* Each call includes:
  + t: an integer timestamp (in milliseconds)

**🚦 Constraints**

* 1 ≤ t ≤ 10^9
* At most 10^4 calls to ping
* All timestamps are strictly increasing

**📤 Output Format**

For each ping(t) call, return an integer representing the number of requests received within the last 3000 milliseconds.

**🏷️ Tags**

queues, sliding-window, time-based, deque, simulation, requests-tracking

Test case 1:

Input:

1

Output:

1

Test case 2:

Input:

100

Output:

2

Test case 3:

Input:

3001

Output:

3

Test case 4:

Input:

3002

Output:

3

Test case 5:

Input:

6000

Output:

3