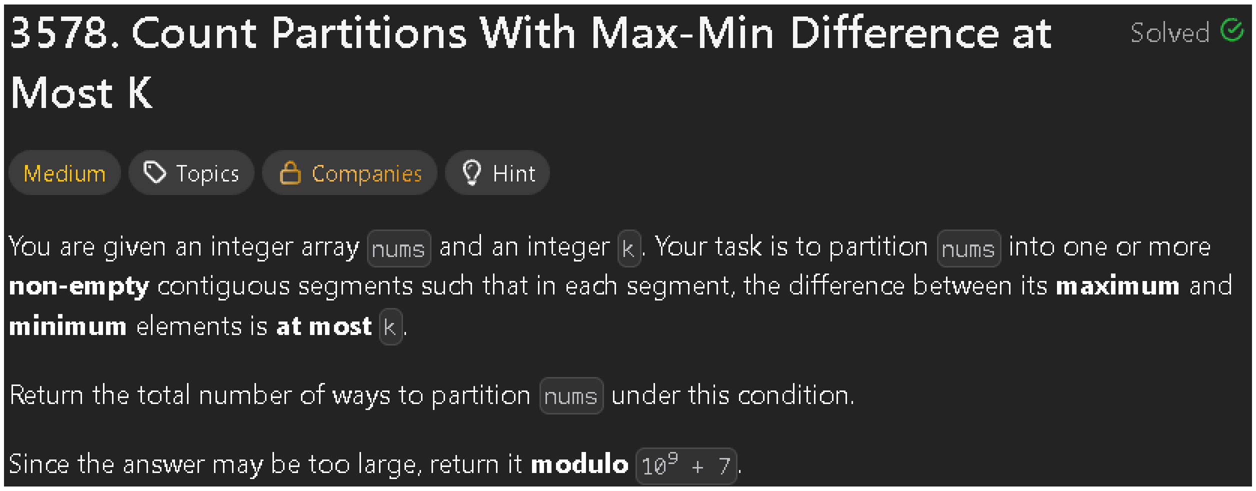


3578. Count Partitions With Max-Min Difference at Most K

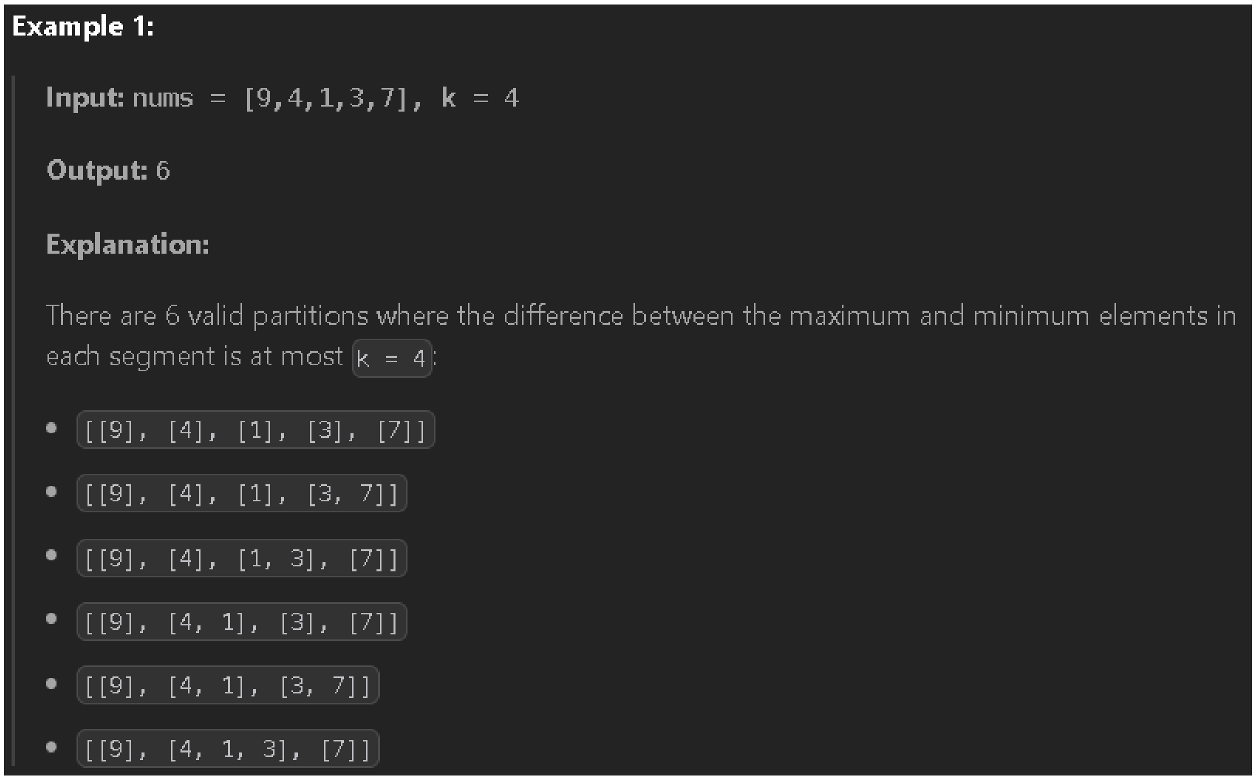
Solved ✓

Medium Topics Companies Hint

You are given an integer array `nums` and an integer `k`. Your task is to partition `nums` into one or more **non-empty** contiguous segments such that in each segment, the difference between its **maximum** and **minimum** elements is **at most** `k`.

Return the total number of ways to partition `nums` under this condition.

Since the answer may be too large, return it **modulo** $10^9 + 7$.



Example 1:

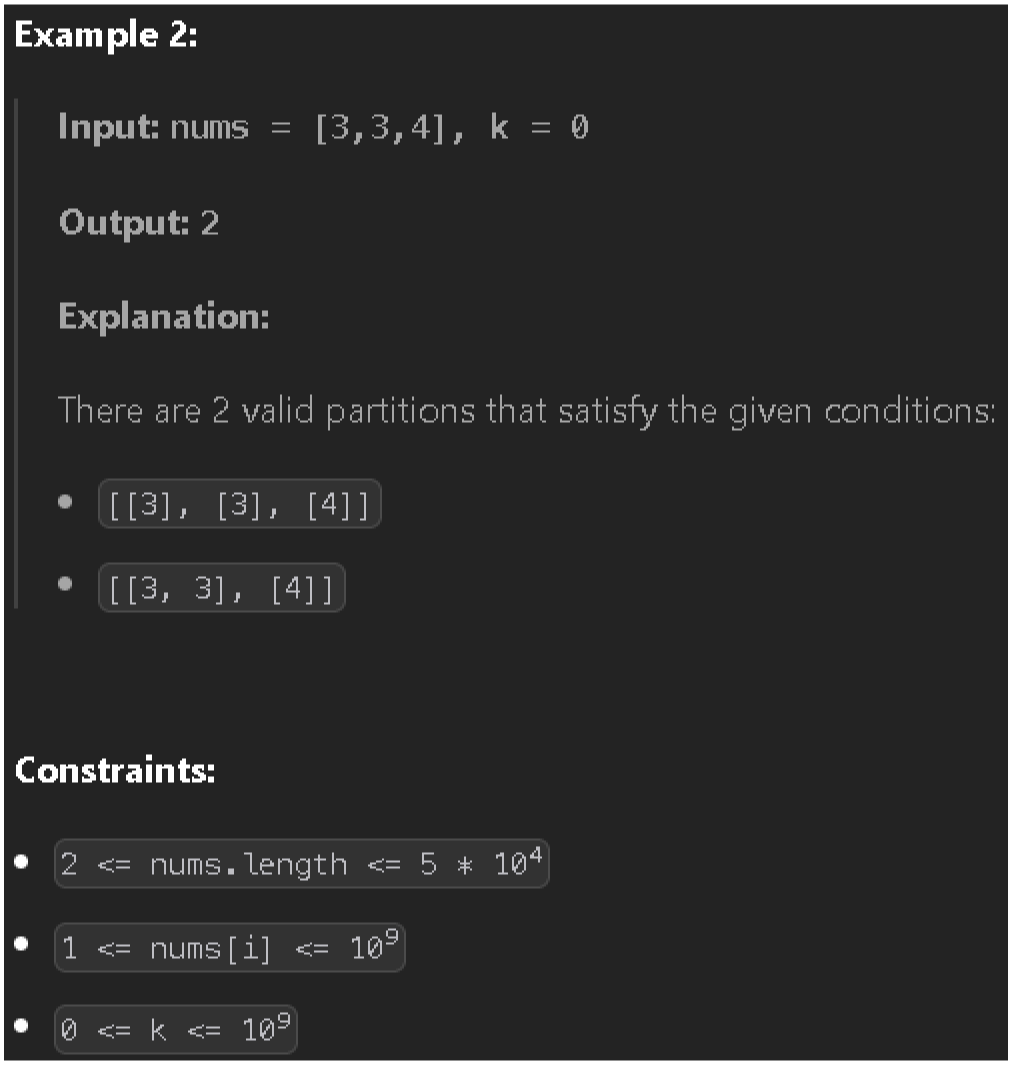
Input: nums = `[9,4,1,3,7], k = 4`

Output: 6

Explanation:

There are 6 valid partitions where the difference between the maximum and minimum elements in each segment is at most `k = 4`:

- `[[9], [4], [1], [3], [7]]`
- `[[9], [4], [1], [3, 7]]`
- `[[9], [4], [1, 3], [7]]`
- `[[9], [4, 1], [3], [7]]`
- `[[9], [4, 1], [3, 7]]`
- `[[9], [4, 1, 3], [7]]`


Example 2:

> **Input:** nums = [3,3,4], k = 0
>
> **Output:** 2
>
> **Explanation:**
>
> There are 2 valid partitions that satisfy the given conditions:
>
> - `[[3], [3], [4]]`
> - `[[3, 3], [4]]`

Constraints:

- `2 <= nums.length <= 5 * 10`^{4}
- `1 <= nums[i] <= 10`^{9}
- `0 <= k <= 10`^{9}

Python:

```
class Solution:
    def countPartitions(self, nums: List[int], k: int) -> int:

        left, cnt, mod_ = 0, 1, 1_000_000_007
        mnQueue, mxQueue, dp = deque(), deque(), [cnt]

        for rght, num in enumerate(nums):
            while mxQueue and num > nums[mxQueue[-1]]:
                mxQueue.pop()
            while mnQueue and num < nums[mnQueue[-1]]:
                mnQueue.pop()

            mxQueue.append(rght)
            mnQueue.append(rght)
            while nums[mxQueue[0]] - nums[mnQueue[0]] > k:
                cnt-= dp[left]
                left+= 1

                if left > mnQueue[0]: mnQueue.popleft()
                if left > mxQueue[0]: mxQueue.popleft()

            dp.append(cnt)
            cnt*= 2
            cnt%= mod_

        return dp[-1] %mod_
```

JavaScript:

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @return {number}
 */
var countPartitions = function(nums, k) {
    const MOD = 1000000007;
    const n = nums.length;

    const dp = new Array(n + 1).fill(0);
    const pref = new Array(n + 1).fill(0);
    dp[0] = 1;
    pref[0] = 1;

    const minq = [];
    const maxq = [];
    let l = 0;

    for (let r = 0; r < n; r++) {
        const x = nums[r];

        while (minq.length > 0 && nums[minq[minq.length - 1]] >= x) {
            minq.pop();
        }
        minq.push(r);
        while (maxq.length > 0 && nums[maxq[maxq.length - 1]] <= x) {
            maxq.pop();
        }
        maxq.push(r);
        while (nums[maxq[0]] - nums[minq[0]] > k) {
            if (minq[0] === l) minq.shift();
            if (maxq[0] === l) maxq.shift();
            l++;
        }

        const base = (l > 0) ? pref[l - 1] : 0;
        let val = (pref[r] - base) % MOD;
        if (val < 0) val += MOD;

        dp[r + 1] = val;
        pref[r + 1] = (pref[r] + dp[r + 1]) % MOD;
    }

    return dp[n] % MOD;
};
```

Java:

```
import java.util.*;

class Solution {
    static final long MOD = 1_000_000_007;

    public int countPartitions(int[] nums, int k) {
        int n = nums.length;

        long[] dp = new long[n + 1];
        long[] prefix = new long[n + 1];

        dp[0] = 1;
        prefix[0] = 1;

        Deque<Integer> minQ = new ArrayDeque<>();
        Deque<Integer> maxQ = new ArrayDeque<>();

        int l = 0;

        for (int r = 0; r < n; r++) {

            while (!minQ.isEmpty() && nums[minQ.peekLast()] > nums[r])
                minQ.pollLast();
            minQ.addLast(r);

            while (!maxQ.isEmpty() && nums[maxQ.peekLast()] < nums[r])
                maxQ.pollLast();
            maxQ.addLast(r);

            while (!minQ.isEmpty() && !maxQ.isEmpty()
                    && nums[maxQ.peekFirst()] - nums[minQ.peekFirst()] > k) {

                if (minQ.peekFirst() == l) minQ.pollFirst();
                if (maxQ.peekFirst() == l) maxQ.pollFirst();
                l++;
            }

            long ways = prefix[r] - (l == 0 ? 0 : prefix[l - 1]);
            ways = (ways % MOD + MOD) % MOD;

            dp[r + 1] = ways;
            prefix[r + 1] = (prefix[r] + dp[r + 1]) % MOD;
        }

        return (int) dp[n];
    }
}
```