<https://leetcode.com/problems/minimum-number-of-days-to-make-m-bouquets/description/>

You are given an integer array bloomDay, an integer m and an integer k.

You want to make m bouquets. To make a bouquet, you need to use k adjacent flowers from the garden.

The garden consists of n flowers, the ith flower will bloom in the bloomDay[i] and then can be used in exactly one bouquet.

Return the minimum number of days you need to wait to be able to make m bouquets from the garden. If it is impossible to make m bouquets return -1.

**Example 1:**

**Input:** bloomDay = [1,10,3,10,2], m = 3, k = 1

**Output:** 3

**Explanation:** Let us see what happened in the first three days. x means flower bloomed and \_ means flower did not bloom in the garden.

We need 3 bouquets each should contain 1 flower.

After day 1: [x, \_, \_, \_, \_] // we can only make one bouquet.

After day 2: [x, \_, \_, \_, x] // we can only make two bouquets.

After day 3: [x, \_, x, \_, x] // we can make 3 bouquets.

The answer is 3.

**Example 2:**

**Input:** bloomDay = [1,10,3,10,2], m = 3, k = 2

**Output:** -1

**Explanation:** We need 3 bouquets each has 2 flowers, that means we need 6 flowers. We only have 5 flowers so it is impossible to get the needed bouquets and we return -1.

**Example 3:**

**Input:** bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3

**Output:** 12

**Explanation:** We need 2 bouquets each should have 3 flowers.

Here is the garden after the 7 and 12 days:

After day 7: [x, x, x, x, \_, x, x]

We can make one bouquet of the first three flowers that bloomed. We cannot make another bouquet from the last three flowers that bloomed because they are not adjacent.

After day 12: [x, x, x, x, x, x, x]

It is obvious that we can make two bouquets in different ways.

**Constraints:**

bloomDay.length == n

1 <= n <= 10^5

1 <= bloomDay[i] <= 10^9

1 <= m <= 10^6

1 <= k <= n

**Attempt 1: 2024-12-07**

**Solution 1: Binary Search + Greedy (10 min)**

class Solution {

    public int minDays(int[] bloomDay, int m, int k) {

        // Add new test case, (int m = 89945, k = 32127),

        // the older solution may cannot pass all test cases,

        // have to convert m \* k to long type

        if((long) m \* k > bloomDay.length) {

            return -1;

        }

        int lo = (int)(1e9 + 1);

        int hi = 0;

        for(int bd : bloomDay) {

            lo = Math.min(lo, bd);

            hi = Math.max(hi, bd);

        }

        while(lo <= hi) {

            int mid = lo + (hi - lo) / 2;

            if(canMakeBouquets(bloomDay, m, k, mid)) {

                hi = mid - 1;

            } else {

                lo = mid + 1;

            }

        }

        return lo;

    }

    private boolean canMakeBouquets(int[] bloomDay, int m, int k, int minWaitDays) {

        int bouquets = 0;

        // Adajacent flower's count whose bloom day no more than minimum wait days

        int flowers = 0;

        // Greedy loop all flowers one by one as adjacent only strategy

        for(int bd : bloomDay) {

            if(bd <= minWaitDays) {

                // Flower can be used

                flowers++;

                // Form a bouquet since all flowers used are adjacent

                if(flowers == k) {

                    // Form a bouquet

                    bouquets++;

                    // Reset count for next section

                    flowers = 0;

                }

                // Make required number of bouquets

                if(bouquets >= m) {

                    return true;

                }

            } else {

                // Reset count if the current flower can't bloom in 'day' days

                flowers = 0;

            }

        }

        return false;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Refer to chatGPT**

**Key Idea:**

This problem can be solved using **binary search**:

The minimum possible day is min(bloomDay), and the maximum possible day is max(bloomDay).

Use binary search to find the smallest day such that it's possible to make m bouquets with k flowers each.

class Solution {

public int minDays(int[] bloomDay, int m, int k) {

int n = bloomDay.length;

// If total flowers are less than required for m bouquets, return -1

if (m \* k > n) {

return -1;

}

// Binary search bounds

int left = Integer.MAX\_VALUE;

int right = Integer.MIN\_VALUE;

// Find the range of bloom days

for (int day : bloomDay) {

left = Math.min(left, day);

right = Math.max(right, day);

}

// Binary search to find the minimum day

while (left <= right) {

int mid = left + (right - left) / 2;

// Check if we can make m bouquets in `mid` days

if (canMakeBouquets(bloomDay, m, k, mid)) {

right = mid - 1; // Try for an earlier day

} else {

left = mid + 1; // Need more days

}

}

return left;

}

private boolean canMakeBouquets(int[] bloomDay, int m, int k, int day) {

int bouquets = 0;

int flowers = 0;

// Iterate through the bloomDay array

for (int bloom : bloomDay) {

if (bloom <= day) {

flowers++; // Flower can be used

if (flowers == k) {

bouquets++; // Form a bouquet

flowers = 0; // Reset flowers

}

} else {

flowers = 0; // Reset if the current flower can't bloom in `day` days

}

if (bouquets >= m) {

return true; // Already formed required bouquets

}

}

return false; // Not enough bouquets

}

}

**Explanation:**

Binary Search:

The range of possible days is [min(bloomDay), max(bloomDay)].

Perform binary search to determine the smallest day d such that m bouquets can be made.

canMakeBouquets **Function**:

Traverse the bloomDay array.

Count consecutive flowers that bloom on or before d.

If k consecutive flowers are found, form a bouquet and reset the count.

Stop early if m bouquets are formed.

Return Result:

If binary search completes, left contains the minimum day that works.

**Refer to**

[L410.Split Array Largest Sum (Ref.L1011,L704,L1482)](note://WEB4d9d6f26a965381b137615128e2648be)

[L1011.Capacity To Ship Packages Within D Days (Ref.L410,L1482)](note://WEB87ca8f9dd9a14fb4535856a4e0f8f4b1)

[L704.Binary Search](note://484532A5D2CB4B98A28218674438D5D8)