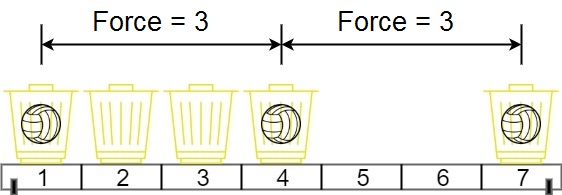
<https://leetcode.com/problems/magnetic-force-between-two-balls/description/>

In the universe Earth C-137, Rick discovered a special form of magnetic force between two balls if they are put in his new invented basket. Rick has n empty baskets, the ith basket is at position[i], Morty has m balls and needs to distribute the balls into the baskets such that the minimum magnetic force between any two balls is maximum.

Rick stated that magnetic force between two different balls at positions x and y is |x - y|.

Given the integer array position and the integer m. Return the required force.

**Example 1:**



**Input:** position = [1,2,3,4,7], m = 3

**Output:** 3

**Explanation:** Distributing the 3 balls into baskets 1, 4 and 7 will make the magnetic force between ball pairs [3, 3, 6]. The minimum magnetic force is 3. We cannot achieve a larger minimum magnetic force than 3.

**Example 2:**

**Input:** position = [5,4,3,2,1,1000000000], m = 2

**Output:** 999999999

**Explanation:** We can use baskets 1 and 1000000000.

**Constraints:**

n == position.length

2 <= n <= 10^5

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

All integers in position are **distinct**.

2 <= m <= position.length

**Attempt 1: 2024-12-08**

**Solution 1: Sorting + Binary Search + Greedy (30 min)**

**Style 1: canDistributeBalls**

class Solution {

    public int maxDistance(int[] position, int m) {

        int len = position.length;

        Arrays.sort(position);

        int lo = 1;

        int hi = position[len - 1] - position[0];

        // Find upper boundary (since we want to find maximum

        // value of minimum magnetic force between any two balls)

        while(lo <= hi) {

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

            // If current maximum value minimum magnetic force between

            // any two balls able to distribute m balls, we can move

            // forward left boundary 'lo' to 'mid + 1' to attempt a larger

            // value, otherwise if cannot distribute m balls, we can move

            // backward right boundary 'hi' to 'mid - 1' to attempt a

            // smaller value

            if(canDistributeBalls(position, m, mid)) {

                lo = mid + 1;

            } else {

                hi = mid - 1;

            }

        }

        return lo - 1;

    }

    private boolean canDistributeBalls(int[] position, int m, int maxMinMagForceBetweenTwoBalls) {

        // Place the first ball at position[0]

        int count = 1;

        int lastPos = position[0];

        // Iterate through the sorted positions, placing the next ball only

        // if the distance from the last placed ball is at least minDist.

        for(int i = 1; i < position.length; i++) {

            if(position[i] - lastPos >= maxMinMagForceBetweenTwoBalls) {

                count++;

                // Update last position only when able to place a ball

                lastPos = position[i];

                // Successfully placed all balls

                if(count == m) {

                    return true;

                }

            }

        }

        // Not enough balls placed

        return false;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Style 2: cannotDistributeBalls**

class Solution {

    public int maxDistance(int[] position, int m) {

        int len = position.length;

        Arrays.sort(position);

        int lo = 1;

        int hi = position[len - 1] - position[0];

        // Find upper boundary (since we want to find maximum

        // value of minimum magnetic force between any two balls)

        while(lo <= hi) {

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

            // If current maximum value minimum magnetic force between

            // any two balls not able to distribute m balls, we can move

            // backward right boundary 'hi' to 'mid - 1' to attempt a

            // smaller value, otherwise if can distribute m balls, we

            // can move backward left boundary 'lo' to 'mid + 1' to

            // attempt a larger value

            if(cannotDistributeBalls(position, m, mid)) {

                hi = mid - 1;

            } else {

                lo = mid + 1;

            }

        }

        return lo - 1;

    }

    private boolean cannotDistributeBalls(int[] position, int m, int maxMinMagForceBetweenTwoBalls) {

        // Place the first ball at position[0]

        int count = 1;

        int lastPos = position[0];

        // Iterate through the sorted positions, placing the next ball only

        // if the distance from the last placed ball is at least minDist.

        for(int i = 1; i < position.length; i++) {

            if(position[i] - lastPos >= maxMinMagForceBetweenTwoBalls) {

                count++;

                // Update last position only when able to place a ball

                lastPos = position[i];

            }

        }

        // If not able to place m balls return true

        return count < m;

    }

}

Time Complexity: O(nlogn)

Space Complexity: O(1)

**Refer to chatGPT**

This is a **binary search on the answer** problem. The task is to maximize the minimum magnetic force by strategically placing balls.

**Key Idea:**

Binary Search:

The minimum distance is 1 (balls placed in adjacent positions).

The maximum distance is max⁡(position)−min⁡(position) (balls placed at the extreme ends of the sorted positions).

Use binary search to find the largest distance x that satisfies the condition.

Feasibility Check:

For a given x, place the balls starting from the smallest position.

If m balls can be placed such that the distance between any two consecutive balls is at least x, then x is feasible.

class Solution {

public int maxDistance(int[] position, int m) {

// Sort the positions

Arrays.sort(position);

// Binary search bounds

int left = 1;

int right = position[position.length - 1] - position[0];

int result = 0;

while (left <= right) {

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

if (canPlaceBalls(position, m, mid)) {

result = mid; // Update result and try for a larger distance

left = mid + 1;

} else {

right = mid - 1; // Decrease the distance

}

}

return result;

}

private boolean canPlaceBalls(int[] position, int m, int minDist) {

int count = 1; // Place the first ball

int lastPlaced = position[0];

for (int i = 1; i < position.length; i++) {

if (position[i] - lastPlaced >= minDist) {

count++;

lastPlaced = position[i];

if (count == m) {

return true; // Successfully placed all balls

}

}

}

return false; // Not enough balls placed

}

}

**Explanation:**

Sorting:

The position array is sorted to allow placing balls in increasing order of positions.

Binary Search:

Start with left=1 and right=max(position) - min(position).

Use the mid-point of the current range to test if it's feasible to place m balls.

canPlaceBalls **Function**:

Place the first ball at the first position.

Iterate through the sorted positions, placing the next ball only if the distance from the last placed ball is at least minDist.

Return true if m balls are successfully placed, otherwise false.

**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)

[L1283.Find the Smallest Divisor Given a Threshold (Ref.L410,L1011,L1482)](note://76952362C4844CB3B68E28EA09F2EE58)

[L1482.Minimum Number of Days to Make m Bouquets (Ref.L410,L1011)](note://WEBd5ce394160da45a79e92dbbc806111ec)

[L1802.Maximum Value at a Given Index in a Bounded Array (Ref.L410)](note://WEB7d060d486a37a8c92696be41f18fb27c)

[L2064.Minimized Maximum of Products Distributed to Any Store (Ref.L410)](note://WEB2ca95ff77a9fee4bcced70f3e62a41df)

[L2226.Maximum Candies Allocated to K Children (Ref.L1802,L1552)](note://WEBbc250dc9fe15c478547298491ba2386f)