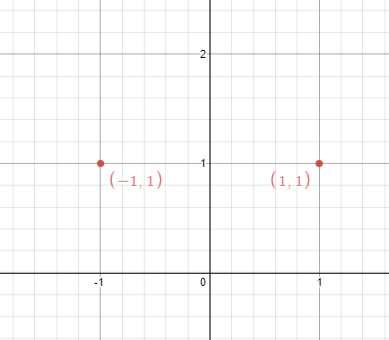
<https://leetcode.ca/all/356.html>

Given n points on a 2D plane, find if there is such a line parallel to the y-axis that reflects the given points symmetrically.

In other words, answer whether or not if there exists a line that after reflecting all points over the given line, the original points' set is the same as the reflected ones.

**Note** that there can be repeated points.

**Example 1:**

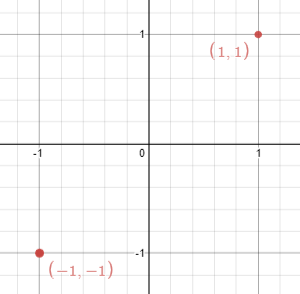


**Input:** points = [[1,1],[-1,1]]

**Output:** true

**Explanation:** We can choose the line x = 0.

**Example 2:**



**Input:** points = [[1,1],[-1,-1]]

**Output:** false

**Explanation:** We can't choose a line.

**Constraints:**

n == points.length

1 <= n <= 10^4

-10^8 <= points[i][j] <= 10^8

**Follow up:** Could you do better than O(n2)?

**Attempt 1: 2025-03-27**

**Solution 1: Math (30 min)**

**Refer to Deepseek**

To solve this problem, we need to determine if a given set of points in a 2D plane can be reflected symmetrically over a vertical line. The key insight is that for the points to be reflectable, there must exist a vertical line such that for every point in the set, its mirror image across this line is also present in the set.

**Approach**

Check for Empty or Single Point: If there are no points or only one point, they trivially form a symmetric line.

**Find Potential Reflection Line: The reflection line must be the average of the smallest and largest x-coordinates of all points. This is because the line must be equidistant from the leftmost and rightmost points.**

Check Symmetry: For each point, check if its mirror image across the potential reflection line exists in the set of points. If all points satisfy this condition, return true; otherwise, return false.

**Solution Code**

class Solution {

public boolean isReflected(int[][] points) {

if (points.length == 0) {

return true;

}

int minX = Integer.MAX\_VALUE;

int maxX = Integer.MIN\_VALUE;

Set<String> pointSet = new HashSet<>();

for (int[] point : points) {

minX = Math.min(minX, point[0]);

maxX = Math.max(maxX, point[0]);

pointSet.add(point[0] + "," + point[1]);

}

int sum = minX + maxX;

for (int[] point : points) {

// Because if the symmetry scenario stands, for the symmetry x1 and x2,

// to find reflect of x1, it will be the reflection line = (x1 + x2) / 2,

// then the delta for both x1 and x2 to the reflection line are the same

// as deltaX = Math.abs((x1 + x2) / 2 - x1), so to find x1's reflect point

// will be reflection line + deltaX = (x1 + x2) / 2 + (x1 + x2) / 2 - x1

// = (x1 + x2) - x1, and x1 + x2 must equal to minX + maxX since if

// symmetry scenario stands

int reflectedX = sum - point[0];

String reflectedPoint = reflectedX + "," + point[1];

if (!pointSet.contains(reflectedPoint)) {

return false;

}

}

return true;

}

}

**Explanation**

Initial Checks: If the input array is empty, return true immediately since an empty set is trivially symmetric.

**Finding Extremes: Iterate through all points to find the smallest (minX) and largest (maxX) x-coordinates. The reflection line will be the average of these two values, calculated as sum = minX + maxX.**

**Checking Symmetry: For each point, compute its mirror image's x-coordinate as sum - point[0] (Because if the symmetry scenario stands, for the symmetry x1 and x2, to find reflect of x1, it will be the reflection line = (x1 + x2) / 2, then the delta for both x1 and x2 to the reflection line are the same as deltaX = Math.abs((x1 + x2) / 2 - x1), so to find x1's reflect point will be reflection line + deltaX = (x1 + x2) / 2 + (x1 + x2) / 2 - x1 = (x1 + x2) - x1, and x1 + x2 must equal to minX + maxX since if symmetry scenario stands). The y-coordinate remains the same. Check if this reflected point exists in the set of original points. If any reflected point is missing, return false.**

Result: If all reflected points are found, return true, indicating the points can be reflected over the vertical line x = sum / 2.

This approach efficiently checks for the existence of a reflection line by leveraging the properties of symmetry and using a hash set for quick lookups. The time complexity is O(n) for iterating through the points, and the space complexity is O(n) for storing the points in the set.

**Refer to**

[L149.Max Points on a Line](note://WEB9621fb1c6a6da80594279dfd17bf65a3)