Problem statements:
Solutions found here: https://walkccc.me/LeetCode/problems/0001/

==

==

Problem 1041. : Robot Bounded in a Circle

On an infinite plane, a robot initially stands at (0, 0) and faces north. Note that:

- The north direction is the positive direction of the y-axis.
- The south direction is the negative direction of the y-axis.
- The east direction is the positive direction of the x-axis.
- The west direction is the negative direction of the x-axis.

The robot can receive one of three instructions:

- "G": go straight 1 unit.
- "L": turn 90 degrees to the left (i.e., anti-clockwise direction).
- "R": turn 90 degrees to the right (i.e., clockwise direction).

The robot performs the instructionsgiven in order, and repeats them forever.

Return true if and only if there exists a circle in the plane such that the robot never leaves the circle.

Example 1:

```
Input: instructions = "GGLLGG"
Output: true
Explanation: The robot is initially at (0, 0) facing the north direction.
"G": move one step. Position: (0, 1). Direction: North.
"G": move one step. Position: (0, 2). Direction: North.
"L": turn 90 degrees anti-clockwise. Position: (0, 2). Direction: West.
"L": turn 90 degrees anti-clockwise. Position: (0, 2). Direction: South.
"G": move one step. Position: (0, 1). Direction: South.
"G": move one step. Position: (0, 0). Direction: South.
Repeating the instructions, the robot goes into the cycle: (0, 0) --> (0, 1)
--> (0, 2) --> (0, 1) --> (0, 0).
Based on that, we return true.
```

Example 2:

```
Input: instructions = "GG"
Output: false
Explanation: The robot is initially at (0, 0) facing the north direction.
"G": move one step. Position: (0, 1). Direction: North.
"G": move one step. Position: (0, 2). Direction: North.
Repeating the instructions, keeps advancing in the north direction and does not
go into cycles.
Based on that, we return false.
```

Example 3:

```
Input: instructions = "GL"
Output: true
Explanation: The robot is initially at (0, 0) facing the north direction.
"G": move one step. Position: (0, 1). Direction: North.
"L": turn 90 degrees anti-clockwise. Position: (0, 1). Direction: West.
"G": move one step. Position: (-1, 1). Direction: West.
"L": turn 90 degrees anti-clockwise. Position: (-1, 1). Direction: South.
"G": move one step. Position: (-1, 0). Direction: South.
"L": turn 90 degrees anti-clockwise. Position: (-1, 0). Direction: East.
"G": move one step. Position: (0, 0). Direction: East.
"L": turn 90 degrees anti-clockwise. Position: (0, 0). Direction: North.
Repeating the instructions, the robot goes into the cycle: (0, 0) --> (0, 1)
--> (-1, 1) --> (-1, 0) --> (0, 0).
Based on that, we return true.
```

Constraints:

- 1 <= instructions.length <= 100
- instructions[i] is 'G', 'L'

==

==

Problem 146.: LRU Cache

Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Implement the LRUCache class:

- LRUCache(int capacity)Initialize the LRU cache with positive size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, evict the least recently used key.

The functions get and put must each run in O(1) average time complexity.

Example 1:

```
Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
lRUCache.get(4);    // return 4
```

Constraints:

- 1 <= capacity <= 3000
- 0 <= key <= 10^4
- 0 <= value <= 10^5
- At most 2 * 10^5 calls will be made to get and put.

==
==

Problem. Number of Provinces (547)

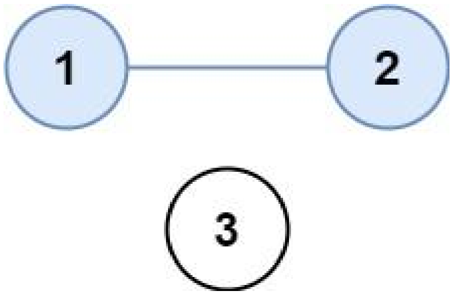
There are n cities. Some of them are connected, while some are not. If city ais connected directly with city b, and city b is connected directly with city c, then city a is connected indirectly with city c.

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an n x n matrix isConnected where isConnected[i][j] = 1 if the ith city and the jth city are directly connected, and isConnected[i][j] = 0 otherwise.

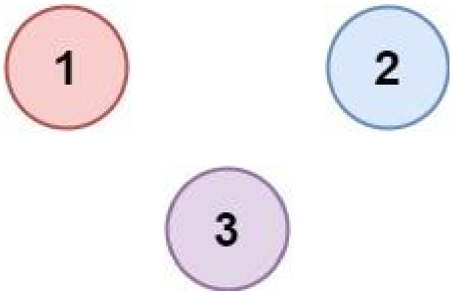
Return *the total number of provinces*.

Example 1:



```
Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2
```

Example 2:



```
Input: isConnected = [[1,0,0],[0,1,0],[0,0,1]]
Output: 3
```

Constraints:

- `1 <= n <= 200`
- `n == isConnected.length`
- `n == isConnected[i].length`
- `isConnected[i][j]` is `1` or `0`.
- `isConnected[i][i] == 1`
- `isConnected[i][j] == isConnected[j][i]`

==
==

Given an array of `intervals` where intervals[i] = [start$_i$, end$_i$], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Constraints:

- 1 <= intervals.length <= 10_4
- intervals[i].length == 2
- 0 <= $start_i$ <= end_i <= 10_4

==

==

56. Merge Intervals

Medium

14219

535

Add to List

Share

Given an array of intervals where intervals[i] = [$start_i$, end_i], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Constraints:

- $1 <= intervals.length <= 10^4$
- $intervals[i].length == 2$
- $0 <= start_i <= end_i <= 10^4$

==
==

973. K Closest Points to Origin

Medium

5755

212

Add to List

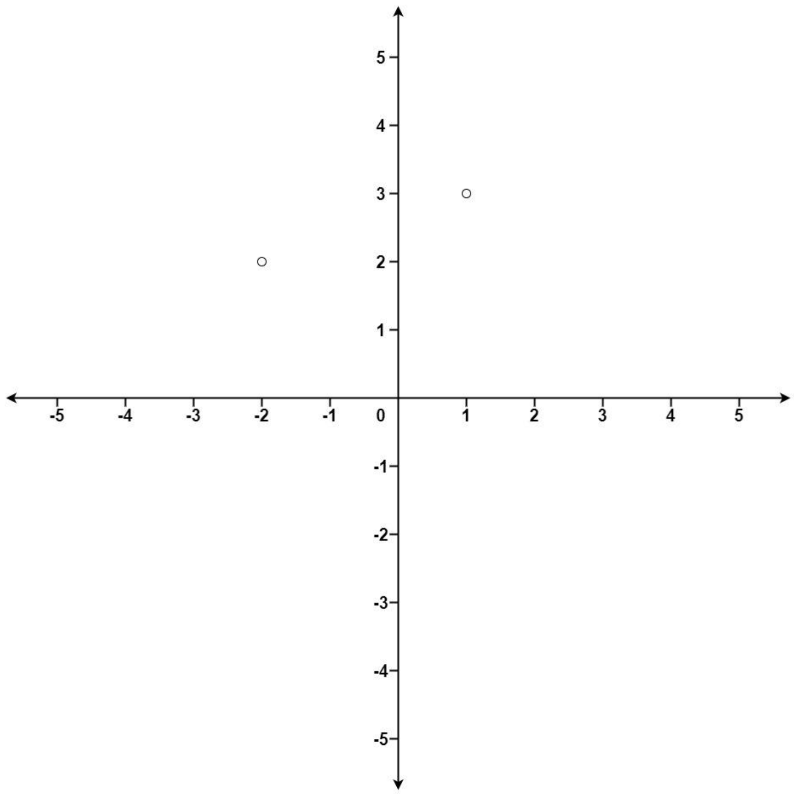
Share

Given an array of points where $points[i] = [x_i, y_i]$ represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

The distance between two points on the X-Y plane is the Euclidean distance (i.e., $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$).

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

Example 1:



```
Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest k = 1 points from the origin, so the answer is just
[[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]
Explanation: The answer [[-2,4],[3,3]] would also be accepted.
```

Constraints:

- $1 <= k <= points.length <= 10^4$
- $-10^4 < x_i, y_i < 10^4$

===

===

1710. Maximum Units on a Truck

Easy

1655

112

Add to List

Share

You are assigned to put some amount of boxes onto one truck. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxes$_i$, numberOfUnitsPerBox$_i$]:

- numberOfBoxes$_i$ is the number of boxes of type i.
- numberOfUnitsPerBox$_i$ is the number of units in each box of the type i.

You are also given an integer truckSize, which is the maximumnumber of boxes that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize.

Return *the maximum total number of units that can be put on the truck.*

Example 1:

```
Input: boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4
Output: 8
Explanation: There are:
- 1 box of the first type that contains 3 units.
- 2 boxes of the second type that contain 2 units each.
- 3 boxes of the third type that contain 1 unit each.
You can take all the boxes of the first and second types, and one box of the
third type.
The total number of units will be = (1 * 3) + (2 * 2) + (1 * 1) = 8.
```

Example 2:

```
Input: boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10
Output: 91
```

Constraints:

- 1 <= boxTypes.length <= 1000
- 1 <= numberOfBoxes_i, $\text{numberOfUnitsPerBox}_i$ <= 1000
- 1 <= truckSize <= 10^6

David Gross,

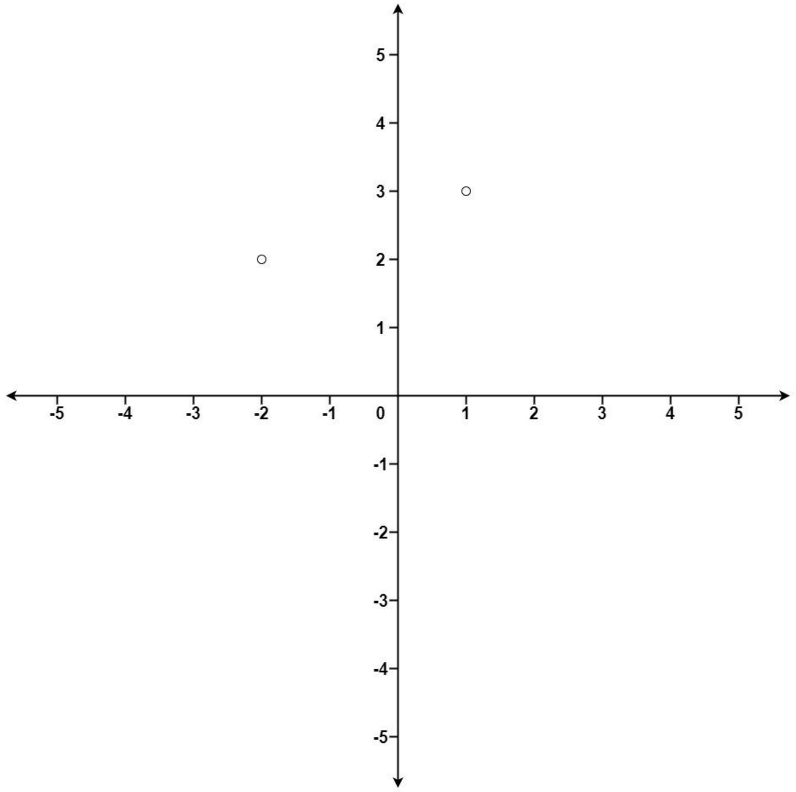
973. K Closest Points to Origin

Given an array of points where points[i] = [x_i, y_i] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

The distance between two points on the X-Y plane is the Euclidean distance (i.e., $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$).

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).

Example 1:



```
Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest k = 1 points from the origin, so the answer is just
[[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]
Explanation: The answer [[-2,4],[3,3]] would also be accepted.
```

Constraints:

- $1 <= k <= points.length <= 10_4$
- $-10_4 < x_i, y_i < 10_4$