Q1. Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Example 2:

Input: strs = [""]
Output: [[""]]

Example 3:

Input: strs = ["a"]
Output: [["a"]]

Constraints:

- 1 <= strs.length <= 10^4
- 0 <= strs[i].length <= 100
- strs[i] consists of lowercase English letters.

Q2. You are given a **0-indexed** array of integers nums of length n. You are initially positioned at nums[0].
Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

- 0 <= j <= nums[i] and
- i + j < n

Return *the minimum number of jumps to reach* nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Example 1:

Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [2,3,0,1,4]
Output: 2

Constraints:

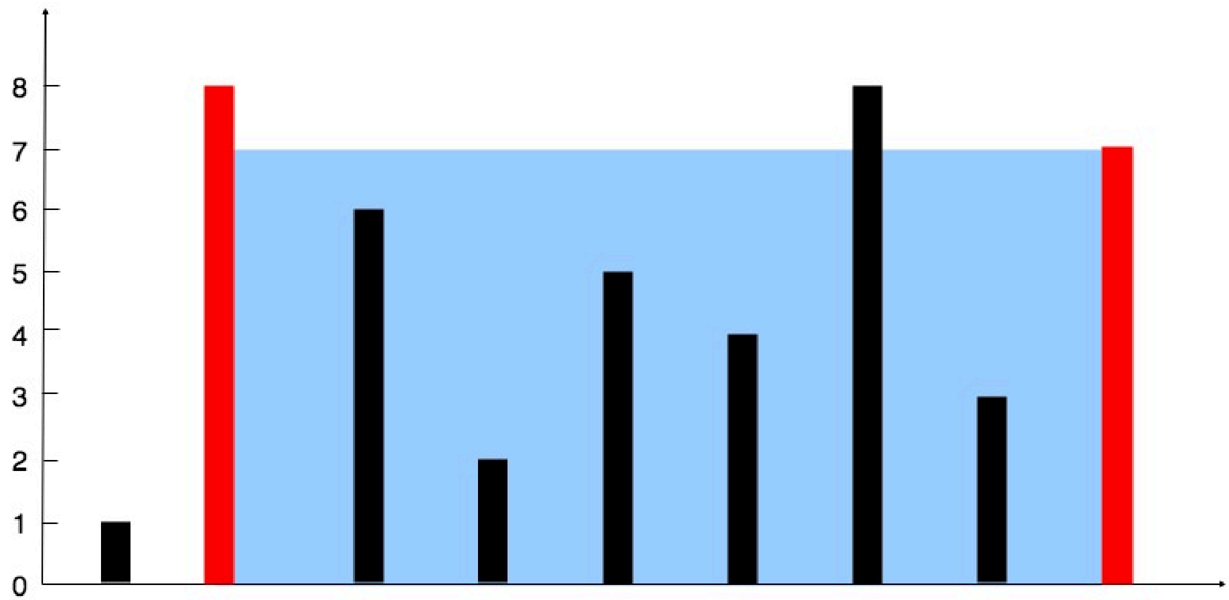
- 1 <= nums.length <= 10^4
- 0 <= nums[i] <= 1000
- It's guaranteed that you can reach nums[n - 1].

Q3. You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i_{th} line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]
Output: 1

Constraints:

- n == height.length
- 2 <= n <= 10^5
- 0 <= height[i] <= 10^4

Q4. Determine if a 9 x 9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: true

Example 2:

Input: board =
[["8","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]

,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: false
Explanation: Same as Example 1, except with the **5** in the top left corner being modified to **8**. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

Constraints:

- board.length == 9
- board[i].length == 9
- board[i][j] is a digit 1-9 or '.'.

Q5. There are n gas stations along a circular route, where the amount of gas at the i_{th} station is gas[i].
You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from the i_{th} station to its next $(i + 1)_{th}$ station. You begin the journey with an empty tank at one of the gas stations.

Given two integer arrays gas and cost, return *the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return* -1. If there exists a solution, it is **guaranteed** to be **unique**

Example 1:

Input: gas = [1,2,3,4,5], cost = [3,4,5,1,2]
Output: 3
Explanation:
Start at station 3 (index 3) and fill up with 4 units of gas. Your tank = 0 + 4 = 4
Travel to station 4. Your tank = 4 - 1 + 5 = 8
Travel to station 0. Your tank = 8 - 2 + 1 = 7
Travel to station 1. Your tank = 7 - 3 + 2 = 6
Travel to station 2. Your tank = 6 - 4 + 3 = 5
Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3.
Therefore, return 3 as the starting index.

Example 2:

Input: gas = [2,3,4], cost = [3,4,3]
Output: -1
Explanation:
You can't start at station 0 or 1, as there is not enough gas to travel to the next station.
Let's start at station 2 and fill up with 4 units of gas. Your tank = 0 + 4 = 4
Travel to station 0. Your tank = 4 - 3 + 2 = 3
Travel to station 1. Your tank = 3 - 3 + 3 = 3
You cannot travel back to station 2, as it requires 4 units of gas but you only have 3.
Therefore, you can't travel around the circuit once no matter where you start.

Constraints:

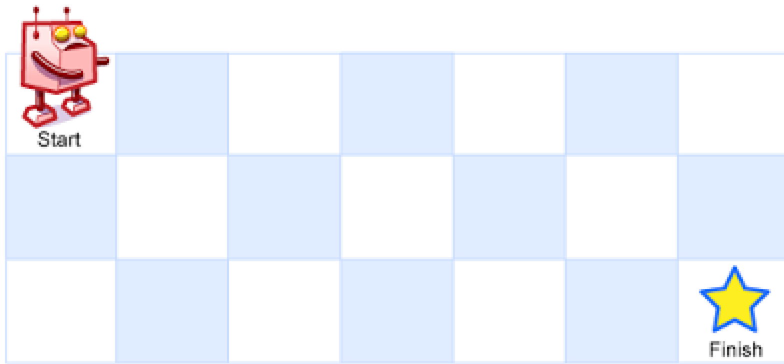
- n == gas.length == cost.length
- 1 <= n <= 10^5
- 0 <= gas[i], cost[i] <= 10^4

Q6 There is a robot on an m x n grid. The robot is initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

Given the two integers m and n, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to 2 * 10^9.

Example 1:



Input: m = 3, n = 7

Output: 28

Example 2:

Input: m = 3, n = 2

Output: 3

Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right -> Down -> Down

2. Down -> Down -> Right

3. Down -> Right -> Down

Constraints:

- 1 <= m, n <= 100

– Optional for 6 ppl groups

Q7. You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 or 3 .. up to k steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2, k = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 3, k = 3

Output: 4

Explanation: There are four ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

4. 3 steps

Constraints:

- 1 <= n <= 45, 1 <= k <= 10

– Optional for 7 ppl groups

Q8. Given an integer array nums, return true *if there exists a triple of indices* (i, j, k) *such that* i < j < k *and* nums[i] < nums[j] < nums[k]. If no such indices exists, return false.

Example 1:

Input: nums = [1,2,3,4,5]

Output: true

Explanation: Any triplet where i < j < k is valid.

Example 2:

Input: nums = [5,4,3,2,1]

Output: false

Explanation: No triplet exists.

Example 3:

Input: nums = [2,1,5,0,4,6]

Output: true

Explanation: The triplet (3, 4, 5) is valid because nums[3] == 0 < nums[4] == 4 < nums[5] == 6.

Constraints:

- 1 <= nums.length <= 5 * 10^5
- -2^{31} <= nums[i] <= 2^{31} - 1

– Optional for 8 ppl groups

Q9. A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true *if it is a **palindrome**, or* false *otherwise*.

Example 1:

Input: s = "A man, a plan, a canal: Panama"

Output: true

Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

Input: s = "race a car"

Output: false

Explanation: "raceacar" is not a palindrome.

Example 3:

Input: s = " "

Output: true

Explanation: s is an empty string "" after removing non-alphanumeric characters.

Since an empty string reads the same forward and backward, it is a palindrome.

Constraints:

- 1 <= s.length <= 2 * 10^5
- s consists only of printable ASCII characters.