

Two pointers

125. Valid Palindrome

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 * 105`
- `s` consists only of printable ASCII characters.

167. Two Sum II - Input Array Is Sorted

Given a **1-indexed** array of integers `numbers` that is already **sorted in non-decreasing order**, find two numbers such that they add up to a specific `target` number. Let these two numbers be `numbers[index1]` and `numbers[index2]` where `1 <= index1 < index2 <= numbers.length`. Return the indices of the two numbers, `index1` and `index2`, **added by one** as an integer array `[index1, index2]` of length 2.

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

```
Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].
```

Example 2:

```
Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].
```

Example 3:

```
Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].
```

Constraints:

- `2 <= numbers.length <= 3 * 104`
- `1000 <= numbers[i] <= 1000`
- `numbers` is sorted in **non-decreasing order**.
- `1000 <= target <= 1000`
- The tests are generated such that there is **exactly one solution**.

15. 3Sum

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: `nums = [-1,0,1,2,-1,-4]`

Output: `[[-1,-1,2],[-1,0,1]]`

Explanation:

`nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.`

`nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.`

`nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.`

The distinct triplets are `[-1,0,1]` and `[-1,-1,2]`.

Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input: `nums = [0,1,1]`

Output: `[]`

Explanation: The only possible triplet does not sum up to 0.

Example 3:

Input: `nums = [0,0,0]`

Output: `[[0,0,0]]`

Explanation: The only possible triplet sums up to 0.

Constraints:

- `3 <= nums.length <= 3000`
- `105 <= nums[i] <= 105`

11. Container With Most Water

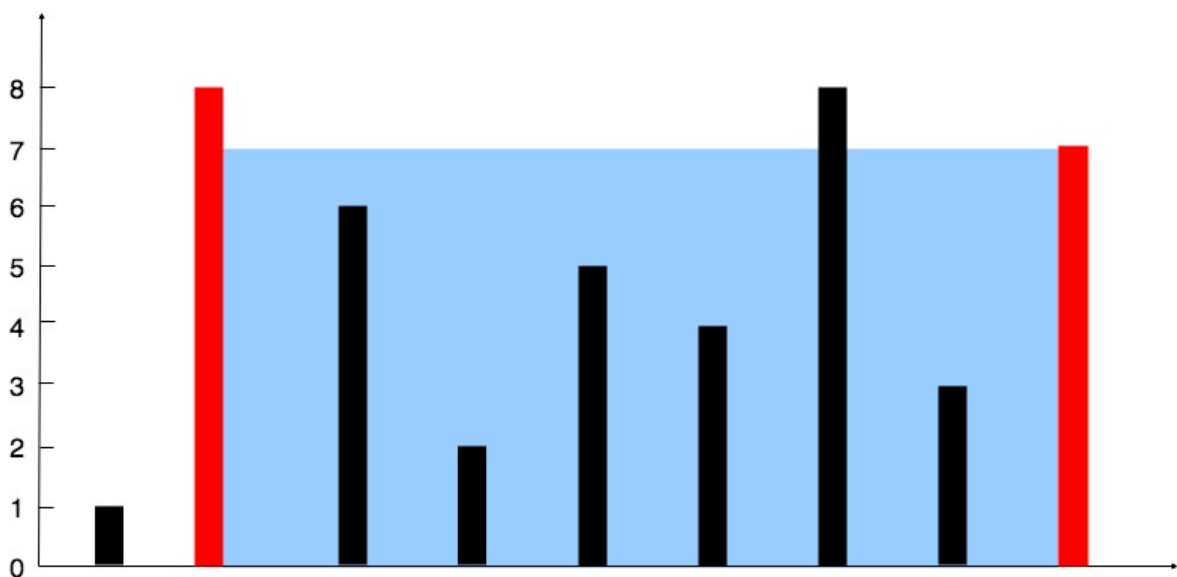
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i`th line are `(i, 0)` and `(i, height[i])`.

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

42. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]
Output: 9

Constraints:

- `n == height.length`
- `1 <= n <= 2 * 104`
- `0 <= height[i] <= 105`