125. Valid Palindrome

Created	@June 23, 2021 9:58 PM
Difficulty	Easy
LC Url	https://leetcode.com/problems/valid-palindrome/
Importance	
Tag	Two pointers
Video	

Given a string `s`, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

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.
```

Constraints:

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

Solution

```python
class Solution:
    def isPalindrome(self, s: str) -> bool:
        left, right = 0, len(s) - 1

        while left < right:
            while left < right and not s[left].isalnum():
                left += 1
            while left < right and not s[right].isalnum():
                right -= 1
            if s[left].lower() != s[right].lower():
                return False
            left += 1
            right -= 1

        return True
```

167. Two Sum II - Input array is sorted

Created	@June 26, 2021 5:34 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/
Importance	
Tag	Two pointers
Video	

Given an 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.

Return *the indices of the two numbers (**1-indexed**) as an integer array* `answer` *of size* `2`*, where* `1 <= answer[0] < answer[1] <= numbers.length`.

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

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.
```

Example 2:

```
Input: numbers = [2,3,4], target = 6
Output: [1,3]
```

Example 3:

```
Input: numbers = [-1,0], target = -1
Output: [1,2]
```

Constraints:

- `2 <= numbers.length <= 3 * 10 4`
- `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**.

Solution

```
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        start = 0
        end = len(numbers) - 1

        while start != end:
            if numbers[start] + numbers[end] > target:
                end -= 1
            elif numbers[start] + numbers[end] < target:
                start += 1
            else:
                return [start+1, end+1]
```

```
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        left, right = 0, len(numbers) - 1

        while left < right:
            isum = numbers[left] + numbers[right]

            if isum == target:
                return [left+1, right+1]
            elif isum < target:
                left += 1
            else:
                right -= 1
        return [-1, -1]
```

15. 3Sum

Created	@July 15, 2020 3:07 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/3sum/
Importance	
Tag	Two pointers
Video	

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`
- `10 5 <= nums[i] <= 10 5`

Solution

排序+双指针

详细讲解：

力扣

https://leetcode.cn/problems/3sum/solution/san-shu-zhi-he-by-leetcode-solution/

```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        # 双指针
        # 链接：https://leetcode.cn/problems/3sum/solution/san-shu-zhi-he-by-leetcode-solution/
        n = len(nums)
        nums.sort()
        ans = list()

        # 枚举 a
        for first in range(n):
            # 需要和上一次枚举的数不相同
            if first > 0 and nums[first] == nums[first - 1]:
                continue
            # c 对应的指针初始指向数组的最右端
            third = n - 1
            target = -nums[first]
            # 枚举 b
            for second in range(first + 1, n):
                # 需要和上一次枚举的数不相同
                if second > first + 1 and nums[second] == nums[second - 1]:
                    continue
                # 需要保证 b 的指针在 c 的指针的左侧
                while second < third and nums[second] + nums[third] > target:
                    third -= 1
                # 如果指针重合，随着 b 后续的增加
                # 就不会有满足 a+b+c=0 并且 b<c 的 c 了，可以退出循环
                if second == third:
                    break
                if nums[second] + nums[third] == target:
                    ans.append([nums[first], nums[second], nums[third]])

        return ans
```

复杂度分析

- 时间复杂度：$O(N^2)$，其中 N 是数组 $nums$ 的长度。
- 空间复杂度：$O(\log N)$。我们忽略存储答案的空间，额外的排序的空间复杂度为 $O(\log N)$。然而我们修改了输入的数组 $nums$，在实际情况下不一定允许，因此也可以看成使用了一个额外的数组存储了 $nums$ 的副本并进行排序，空间复杂度为 $O(N)$。

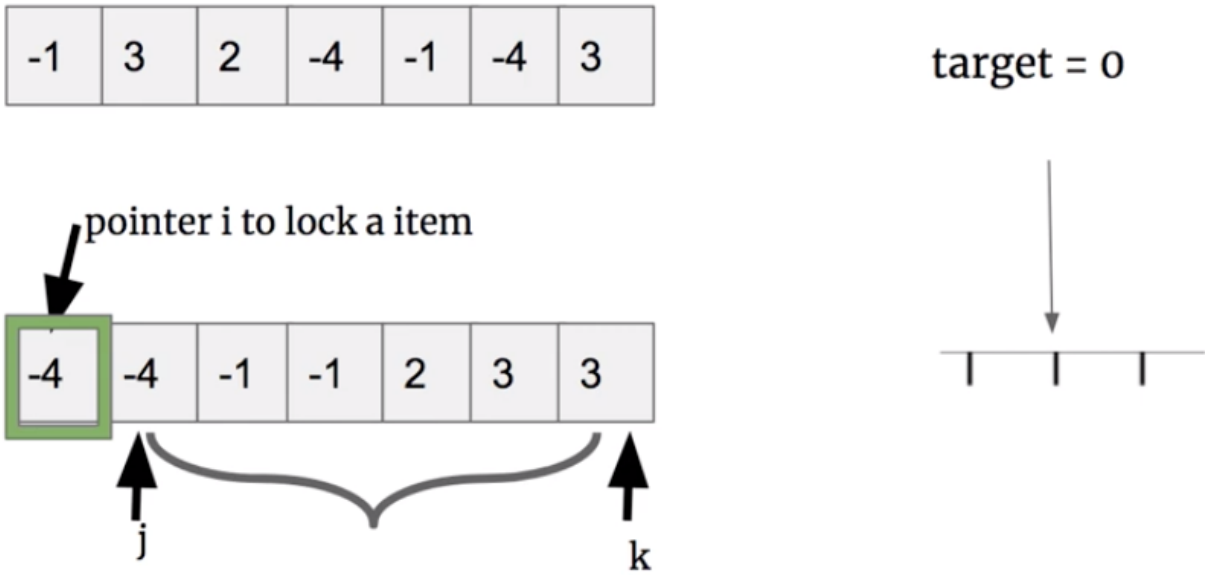
```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        ArrayList<List<Integer>> res = new ArrayList<List<Integer>>();
        if (nums == null || nums.length <= 2) return res;
        int n = nums.length;
        int i = 0;
        Arrays.sort(nums);
        while (i < n-2) {
            int base = nums[i];
            int left = i + 1;
            int right = n - 1;

            while (left < right) {
                int sum = base + nums[left] + nums[right];
                if (sum == 0) {
                    LinkedList<Integer> list = new LinkedList<Integer>();
                    list.add(base);
                    list.add(nums[left]);
                    list.add(nums[right]);
                    res.add(list);
                    left = moveRight(nums, left+1);
                    right = moveLeft(nums, right-1);
                } else if (sum > 0) {
                    right = moveLeft(nums, right-1);
                } else {
                    left = moveRight(nums, left+1);
                }
            }
            i = moveRight(nums, i+1);
        }
        return res;
    }

    public int moveLeft(int[] nums, int right) {
        while (right == nums.length-1 || (right >= 0 && nums[right] == nums[right+1])) {
            right--;
        }
        return right;
    }
```

```
    public int moveRight(int[] nums, int left) {
        while (left == 0 || left < nums.length && nums[left] == nums[left-1]) {
            left++;
        }
        return left;
    }
}
```

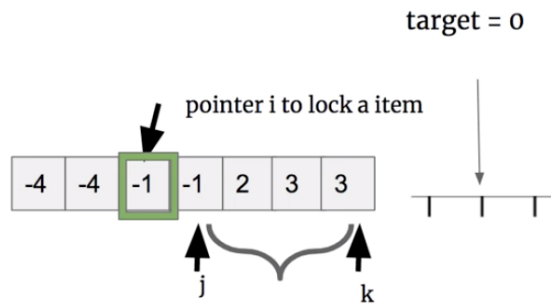
https://www.youtube.com/watch?v=2tbi1W7ce1c



Solution: Two Pointers
1. Sort the array
2. Lock one pointer and do two sum with the other two

```java
public List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> result = new ArrayList<>();
    int target = 0;
    Arrays.sort(nums);
    // corner case
    if (nums.length < 3) return result;

    for (int i = 0; i < nums.length; i++) {
      if (i > 0 && nums[i] == nums[i - 1]) continue;   // skip duplicates
      int j = i + 1;
      int k = nums.length - 1;
      while (j < k) {
        if (nums[i] + nums[j] + nums[k] < target) {
          j++;
          while (j < k && nums[j] == nums[j - 1]) j++; // skip duplicates
        } else if (nums[i] + nums[j] + nums[k] > target) {
          k--;
          while (j < k && nums[k] == nums[k + 1]) k--; // skip duplicates
        } else {
          result.add(Arrays.asList(nums[i], nums[j], nums[k]));
          j++;
          k--;
          while (j < k && nums[j] == nums[j - 1]) j++; // skip duplicates
          while (j < k && nums[k] == nums[k + 1]) k--; // skip duplicates
        }
      }
    }
    return result;
}
```



TIME: O(n^2)
SPACE: O(1)

11. Container With Most Water

Created	@July 15, 2020 2:22 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/container-with-most-water/
Importance	****
Tag	Two pointers
Video	

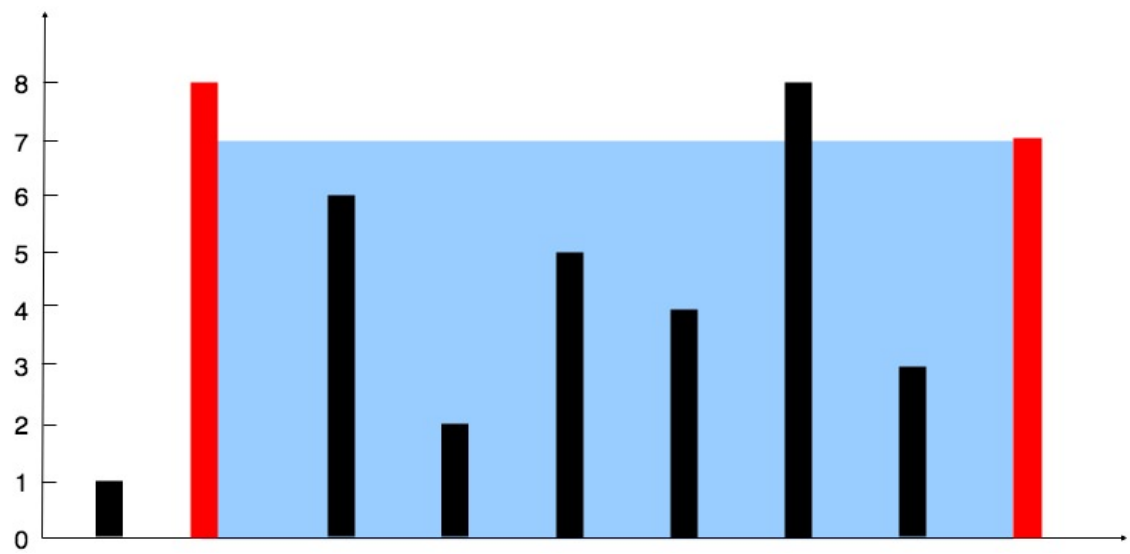
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i th` line are `(i, 0)` and `(i, height[i])`.

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 s
ection) 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] <= 104`

Solution

```
class Solution:
    def maxArea(self, height: List[int]) -> int:
        # 双指针
        # 链接：https://leetcode.cn/problems/container-with-most-water/solution/sheng-zui-duo-shui-de-rong-qi-by-leetcode-solution/
        left, right = 0, len(height) - 1
        ans = 0

        while left < right:
            area = min(height[left], height[right]) * (right - left)
            ans = max(ans, area)

            if height[left] <= height[right]:
                left += 1
            else:
                right -= 1
        return ans
```

复杂度分析

- 时间复杂度：$O(N)$，双指针总计最多遍历整个数组一次。
- 空间复杂度：$O(1)$，只需要额外的常数级别的空间。

力扣

https://leetcode.cn/problems/container-with-most-water/solution/sheng-zui-duo-shui-de-rong-qi-by-leetcode-solution/

https://www.bilibili.com/video/BV1a4411e7oh?p=8

```
class Solution {
    public int maxArea(int[] height) {
        if (height == null || height.length <2) return 0;
        int maxArea = 0;
        int left = 0, right = height.length - 1;
        while (left < right) {
            maxArea = Math.max(maxArea, (right - left) * Math.min(height[right], height[left]));
            if (height[right] < height[left]) {
                right--;
            } else {
                left++;
            }
        }
        return maxArea;
    }

    public static void main(String[] args) {
        Solution solver = new Solution();
        int[] arr = {1,8,6,2,5,4,8,3,7};
        System.out.println(solver.maxArea(arr));
    }
}
```

42. Trapping Rain Water

Created	@October 9, 2022 3:43 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/trapping-rain-water/
Importance	
Tag	Two pointers
Video	https://www.youtube.com/watch?v=ZI2z5pq0TqA

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:

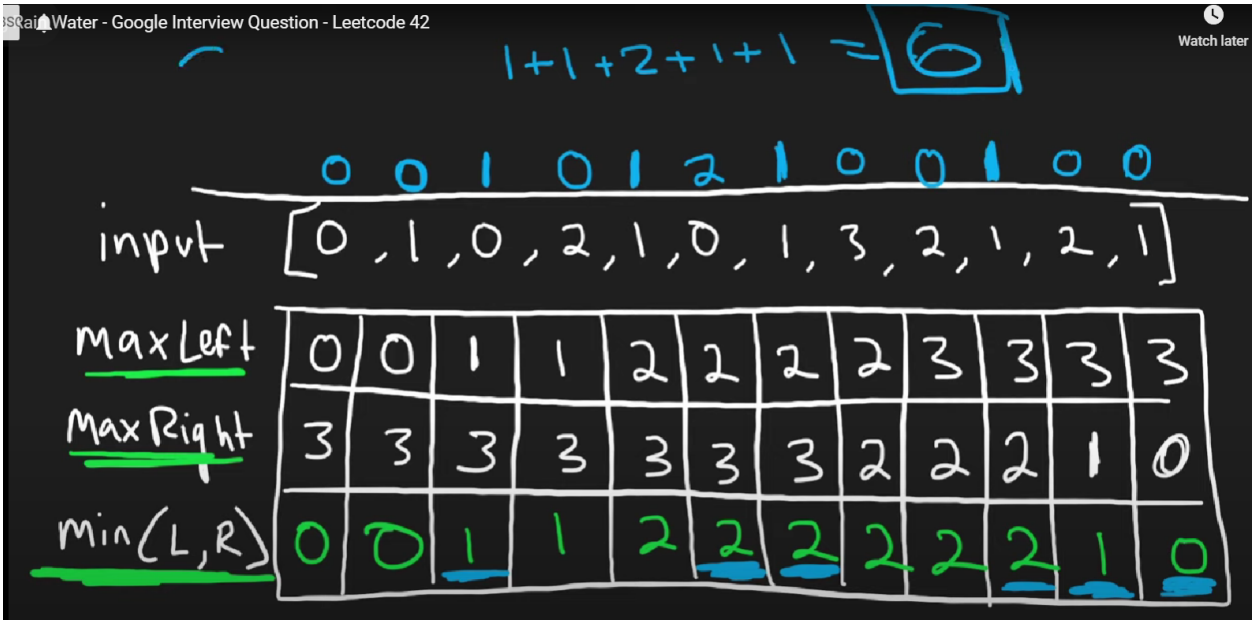
```
Input: height = [4,2,0,3,2,5]
Output: 9
```

Constraints:

- `n == height.length`
- `1 <= n <= 2 * 10 4`
- `0 <= height[i] <= 10 5`

Solution

O(n)




```
class Solution:
    def trap(self, height: List[int]) -> int:
        if not height:
            return 0

        left, right = 0, len(height) - 1
        leftMax, rightMax = height[left], height[right]
        res = 0

        while left < right:
            if leftMax <= rightMax:
                left += 1
                leftMax = max(leftMax, height[left])
                res += leftMax - height[left]
            else:
                right -= 1
```

```
                rightMax = max(rightMax, height[right])
                res += rightMax - height[right]
        return res
```