69. Sqrt(x)

**Example 1:**

**Input:** x = 4

**Output:** 2

**Explanation:** The square root of 4 is 2, so we return 2.

**Example 2:**

**Input:** x = 8

**Output:** 2

**Explanation:** The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

class Solution:

    def mySqrt(self, x: int) -> int:

        start = 0

        end = x

        if x == 1:

            return 1

        while start <= end:

            mid = start + (end - start) // 2

            if mid \* mid <= x < (mid+1) \*\* 2:

                return mid

            elif mid \*\* 2 < x:

                start = mid + 1

            else:

                end = mid - 1

        # #return int(mid)

121. Best Time to Buy and Sell Stock

**Example 1:**

**Input:** prices = [7,1,5,3,6,4]

**Output:** 5

**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

**Example 2:**

**Input:** prices = [7,6,4,3,1]

**Output:** 0

**Explanation:** In this case, no transactions are done and the max profit = 0.

class Solution:

    def maxProfit(self, prices: List[int]) -> int:

        # mn = min(prices)

        # # if mn == prices[-1]:

        # #     return 0

        # mx = mn

        # x = 0

        # for i in range(len(prices)):

        #     if prices[i] == mn:

        #         x = i

        #         break

        # for i in range(x, len(prices)):

        #     if prices[i] > mx:

        #         mx = prices[i]

        # return mx - mn

        output, x = 0, 0

        # for i in range(len(prices)):

        #     for j in range(1, len(prices) - i):

        #         if prices[i] < prices[i + j]:

        #             x = prices[i + j] - prices[i]

        #             if x > output:

        #                 output = x

        # return output

        left = 0

        right = 1

        maxProfit = 0

        while(right < len(prices)):

            if prices[left] < prices[right]:

                curProfit = prices[right] - prices[left]

                maxProfit = max(curProfit, maxProfit)

                # if curProfit > maxProfit:

                #     maxProfit = curProfit

            else:

                left = right

            right += 1

        return maxProfit

        # while(left < len(prices) - 1):

        #     if prices[right] > prices[left]:

        #         curProfit = prices[right] - prices[left]

        #         maxProfit = max(curProfit, maxProfit)

        #     right += 1

        #     if right == len(prices):

        #         left += 1

        #         right = left + 1

        # return maxProfit

125. Valid Palindrome

**Example 1:**

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

**Output:** true

**Explanation:** "amanaplanacanalpanama" is a palindrome.

class Solution:

    def isPalindrome(self, s: str) -> bool:

        s = s.lower()

        # pattern = r'[^A-Za-z0-9]+'

        # s = re.sub(pattern, '', s)

        # extra = ""

        # for char in s:

        #     if char.isalnum():

        #         extra += char

        # s = extra

        # j = -1

        # for i in range(len(s)//2):

        #     if s[i] != s[j]:

        #         return False

        #     j -= 1

        # return True

        def alphaNum(c : str) -> bool:

            if (ord("a") <= ord(c) <= ord("z") or ord('0') <= ord(c) <= ord('9')):

                return True

            else:

                return False

            # return (ord('a') <= ord(c) <= ord('z') or

            #        ord('0') <= ord(c) <= ord('9') )

        l, r = 0, len(s) - 1

        while l <= r:

            while l < r and alphaNum(s[l]) == False:

                l += 1

            while r > l and alphaNum(s[r]) == False:

                r -= 1

            if s[l] != s[r]:

                return False

            l += 1

            r -= 1

        return True

        #Valid Palindrome 2 where you can delete at most one character

        # l, r = 0, len(s) - 1

        # while l < r:

        #     if s[l] != s[r]:

        #         skipL = s[l + 1:r + 1]

        #         skipR = s[l:r]

        #         return skipL == skipL[::-1] or skipR == skipR[::-1]

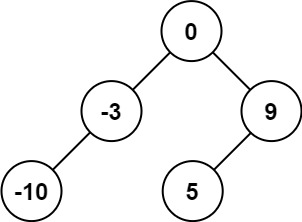
        #     l += 1

        #     r -= 1

        # return True

108. Convert Sorted Array to Binary Search Tree

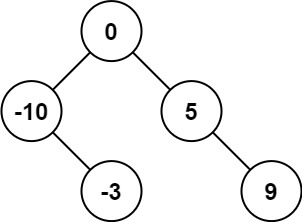
**Example 1:**



**Input:** nums = [-10,-3,0,5,9]

**Output:** [0,-3,9,-10,null,5]

**Explanation:** [0,-10,5,null,-3,null,9] is also accepted:



# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, val=0, left=None, right=None):

#         self.val = val

#         self.left = left

#         self.right = right

class Solution:

    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:

        def helper(l, r) -> TreeNode:

            if r < l:

                return None

            m = l + (r - l) // 2

            root = TreeNode(nums[m])

            root.left = helper(l, m - 1)

            root.right = helper(m + 1, r)

            return root

        if nums is None:

            return None

        return helper(0, len(nums) - 1)

Medium

3. Longest Substring Without Repeating Characters(Medium)

**Example 1:**

**Input:** s = "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

**Example 2:**

**Input:** s = "bbbbb"

**Output:** 1

**Explanation:** The answer is "b", with the length of 1.

class Solution:

    def lengthOfLongestSubstring(self, s: str) -> int:

        st = set()

        l = 0

        res = 0

        #sliding window problem

        for r in range(len(s)):

            while s[r] in st:

                st.remove(s[l])

                l += 1

            st.add(s[r])

            res = max(res, r - l + 1)

        return res

5. Longest Palindromic Substring (Medium)

**Example 1:**

**Input:** s = "babad"

**Output:** "bab"

**Explanation:** "aba" is also a valid answer.

**Example 2:**

**Input:** s = "cbbd"

**Output:** "bb"

class Solution:

    def longestPalindrome(self, s: str) -> str:

        #res = ""

        resLen = 0

        resL, resR = 0, 0

        for i in range(len(s)):

            #odd length

            l, r = i, i

            while l >= 0 and r < len(s) and s[l] == s[r]:

                if r - l + 1 > resLen:

                    resLen = r - l + 1

                    resL, resR = l, r

                l -= 1

                r += 1

            #even length

            l, r = i, i + 1

            while l >= 0 and r < len(s) and s[l] == s[r]:

                if r - l + 1 > resLen:

                    resLen = r - l + 1

                    resL, resR = l, r

                l -= 1

                r += 1

        return s[resL: resR + 1]

        # res = ""

        # resLen = 0

        # for i in range(len(s)):

        #     #odd length

        #     l, r = i, i

        #     while l >= 0 and r < len(s) and s[l] == s[r]:

        #         if r - l + 1 > resLen:

        #             resLen = r - l + 1

        #             res = s[l:r+1]

        #         l -= 1

        #         r += 1

        #     #even length

        #     l, r = i, i + 1

        #     while l >= 0 and r < len(s) and s[l] == s[r]:

        #         if r - l + 1 > resLen:

        #             resLen = r - l + 1

        #             res = s[l:r+1]

        #         l -= 1

        #         r += 1

        # return res

11. Container with Most Water (Medium)

**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.

class Solution:

    def maxArea(self, height: List[int]) -> int:

        area = 0

        l, r = 0, len(height) - 1

        while l < r:

            a = (r - l) \* min(height[l], height[r])

            area = max(a, area)

            if height[l] < height[r]:

                l += 1

            else:

                r -= 1

        return area

        #mn = 0

        # for i in range(len(height)):

        #     for j in range(1, len(height) - i):

        #         mn = min(height[i], height[i + j])

        #         #print("Height, y = ", mn)

        #         #print("x = ", j - i)

        #         a = (i + j - i) \* mn

        #         #print("Area = ",a)

        #         area = max(area, a)

        # return area

        # for i in range(len(height)):

        #     for j in range(i + 1, len(height)):

        #         a = (j - i) \* min(height[i], height[j])

        #         area = max(a, area)

        # return area

33. Search in Rotated Sorted Array (Medium)

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the possible rotation and an integer target, return *the index of*target*if it is in*nums*, or*-1*if it is not in*nums.

You must write an algorithm with O(log n) runtime complexity.

**Example 1:**

**Input:** nums = [4,5,6,7,0,1,2], target = 0

**Output:** 4

**Example 2:**

**Input:** nums = [4,5,6,7,0,1,2], target = 3

**Output:** -1

**Example 3:**

**Input:** nums = [1], target = 0

**Output:** -1

class Solution:

    def search(self, nums: List[int], target: int) -> int:

l, r = 0, len(nums) - 1

        while l <= r:

            m = l + (r - l) // 2

            if nums[m] == target:

                return m

            if nums[m] >= nums[l]:

                #left sorted portion

                if nums[l] > target or nums[m] < target:

                    l = m + 1

                else:

                    r = m - 1

            else:

                #right sorted portion

                if nums[r] < target or nums[m] > target:

                    r = m - 1

                else:

                    l = m + 1

        return -1

34. Find First and Last Position of Element in Sorted Array (Medium)

**Example 1:**

**Input:** nums = [5,7,7,8,8,10], target = 8

**Output:** [3,4]

**Example 2:**

**Input:** nums = [5,7,7,8,8,10], target = 6

**Output:** [-1,-1]

Input: nums = [5, 8,8,8,8,8, 10], target = 8

Output: [1, 5]

class Solution:

    def searchRange(self, nums: List[int], target: int) -> List[int]:

        def binSearch(leftMost : bool) -> int:

            l, r = 0, len(nums) - 1

            index = -1

            while l <= r:

                m = l + (r - l) // 2

                if nums[m] > target:

                    r = m - 1

                elif nums[m] < target:

                    l = m + 1

                else:

                    index = m

                    if leftMost == True:

                        r = m - 1

                    else:

                        l = m + 1

            return index

        l = binSearch(True)

        r = binSearch(False)

        return l, r #[l, r]