	Description:	Submitted Code Python:	Notes:	Time	Space	Submitted Code JavaScript	Notes:	Time:	Space:

202. Happy Number (EZ)	Write an algorithm to determine if a number n is happy.
	
	A happy number is a number defined by the following process:
	
	Starting with any positive integer, replace the number by the sum of the squares of its digits.
	Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
	Those numbers for which this process ends in 1 are happy.
	Return true if n is a happy number, and false if not.
	
	Example 1:
	
	Input: n = 19
	Output: true
	Explanation:
	12 + 92 = 82
	82 + 22 = 68
	62 + 82 = 100
	12 + 02 + 02 = 1
	Example 2:
	
	Input: n = 2
	Output: false
		class Solution:
		  def isHappy(self, n: int) -> bool:
		  seen = set()
		
		  while n != 1 and n not in seen:
		  seen.add(n)
		  digit_sum = 0
		  for digit in str(n):
		  digit = int(digit) ** 2
		  digit_sum += digit
		  n = digit_sum
		  if n == 1:
		  return True
		  else:
		  return False
			- Created a seen set (does not allow for duplicates)
			- While loop to check if 1 or n is NOT in seen
			- Add n to seen
			- INIT digit_sum
			- separate digits from number
			- Square the digits
			- Increment digit_sum with the new digits
			- Make N digit sum
			- Check if happy
				35 ms / 18.46%
				
				16 mb / 80%					

217. Contains Duplicate (EZ)	Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.
	
	Example 1:
	
	Input: nums = [1,2,3,1]
	Output: true
	Example 2:
	
	Input: nums = [1,2,3,4]
	Output: false
	Example 3:
	
	Input: nums = [1,1,1,3,3,4,3,2,4,2]
	Output: true
		class Solution:
		  def containsDuplicate(self, nums: List[int]) -> bool:
		  s_nums = sorted(nums)
		  for i in range(len(s_nums) - 1):
		  if s_nums[i] == s_nums[i + 1]:
		  return True
			- First sort the array
			- Check the elements in range of length of array (-1 to avoid overflow)
			- If sorted nums element at index i is equal to sorted nums element at ext index return TRUE
				448 ms / 13.1 %
				
				28.10 mb / 98.69%
						/**
						* @param {number[]} nums
						* @return {boolean}
						*/
						var containsDuplicate = function(nums) {
						  let seen = new Set();
						
						  for (let num of nums) {
						  if (seen.has(num)) {
						  return true;
						  }
						  seen.add(num);
						  }
						
						  return false;
						};
							- Define a function named `containsDuplicate` that takes an array `nums` as an argument.
							- Create a new `Set` called `seen` to store unique numbers encountered in the array.
							- Start a loop that iterates over each element in the array `nums`.
							- Check if the `Set` `seen` already contains the current element `num`.
							- If the current element `num` is found in the `Set`, return `true` to indicate a duplicate exists.
							- If the current element `num` is not found in the `Set`, add it to the `Set`.
							- Close the loop.
							- If the loop completes without finding any duplicates, return `false` to indicate no duplicates were found.
							- Close the function definition.
								68 ms / 92.84%	83.12 mb / 22.62%

258. Add Digits (EZ)	Given an integer num, repeatedly add all its digits until the result has only one digit, and return it.
	
	Example 1:
	
	Input: num = 38
	Output: 2
	Explanation: The process is
	38 --> 3 + 8 --> 11
	11 --> 1 + 1 --> 2
	Since 2 has only one digit, return it.
	Example 2:
	
	Input: num = 0
	Output: 0
	
	Constraints:
	
	0 <= num <= 231 - 1
	
	Follow up: Could you do it without any loop/recursion in O(1) runtime?
		class Solution:
		  def addDigits(self, num: int) -> int:
		  n = num
		  seen = set()
		  while n >= 10 and n not in seen:
		  seen.add(n)
		  digit_sum = 0
		  for digit in str(n):
		  digit_sum += int(digit)
		  n = digit_sum
		  if n == 0:
		  return n
			- INIT n as num
			- INIT seen as an empty set
			- While loop the checks if n is greater than or equal to 10 (multiple digits) and is not in the set
			- Add the number to the set
			- INIT digit_sum
			- Increment digit_sum with the new digits
			- Make N digit sum
			- Check if less than two digits
			
			ALT CODE:
			
			class Solution:
			  def addDigits(self, num: int) -> int:
			  while num >= 10:
			  digit_sum = 0
			  for digit in str(num):
			  digit_sum += int(digit)
			  num = digit_sum
			  return num
				34 ms / 70.56%	16.52 mb / 52.82%				

605. Can Place Flowers (EZ)	You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots.
	
	Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.
	
	Example 1:
	
	Input: flowerbed = [1,0,0,0,1], n = 1
	Output: true
	Example 2:
	
	Input: flowerbed = [1,0,0,0,1], n = 2
	Output: false
	
	Constraints:
	
	1 <= flowerbed.length <= 2 * 104
	flowerbed[i] is 0 or 1.
	There are no two adjacent flowers in flowerbed.
	0 <= n <= flowerbed.length
		class Solution:
		  def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
		  flowers = 0
		  i = 0
		  while i < len(flowerbed):
		  if (flowerbed[i] == 0 and (i == 0 or flowerbed[i - 1] == 0) and
		  (i == len(flowerbed) - 1 or flowerbed[i + 1] == 0)):
		  flowerbed[i] = 1
		  flowers += 1
		  i += 1
		  return flowers >= n
			- INIT flowers as 0
			- INIT i (position) as 0
			- While i is less than the length of the flower bed go to next line of notes
			- If i is still less than length (Check if the current position in the flowerbed is empty (`flowerbed[i] == 0`) and if either it's the first position or the previous position is empty (`i == 0 or flowerbed[i - 1] == 0`), and if either it's the last position or the next position is empty (`i == len(flowerbed) - 1 or flowerbed[i + 1] == 0`), indicating a valid spot to plant a flower.)
			- If true make position of array 1
			- Also increment flowers by 1
			- And increment position by 1
			- Return true if flowers are equal to n
				135 ms / 39.44%	16.86 mb / 73.02%				

242. Valid Anagram (EZ)	Given two strings s and t, return true if t is an anagram of s, and false otherwise.
	
	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: s = "anagram", t = "nagaram"
	Output: true
	Example 2:
	
	Input: s = "rat", t = "car"
	Output: false
	
	Constraints:
	
	1 <= s.length, t.length <= 5 * 104
	s and t consist of lowercase English letters.
	
	Follow up: What if the inputs contain Unicode characters? How would you adapt your solution to such a case?
		class Solution:
		  def isAnagram(self, s: str, t: str) -> bool:
		  seen_s = []
		  seen_t = []
		  s_str = sorted(s)
		  t_str = sorted(t)
		
		  if len(s_str) == len(t_str):
		  for char in s_str:
		  seen_s.append(char)
		  for char in t_str:
		  seen_t.append(char)
		  if seen_t == seen_s:
		  return True
		  else:
		  return
			- INIT both arrays for s and t
			- Sort the strings
			- Check if arrays are same length before going further
			- Add characters of sorted strings to each array
			- If the arrays are equal return true
				54 ms / 32.40%	18.44 mb / 14.91%				

283. Move Zeroes (EZ)	Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.
	
	Note that you must do this in-place without making a copy of the array.
	
	Example 1:
	
	Input: nums = [0,1,0,3,12]
	Output: [1,3,12,0,0]
	Example 2:
	
	Input: nums = [0]
	Output: [0]
	
	Constraints:
	
	1 <= nums.length <= 104
	-231 <= nums[i] <= 231 - 1
	
	Follow up: Could you minimize the total number of operations done?
		class Solution:
		  def moveZeroes(self, nums: List[int]) -> None:
		  insert_pos = 0
		  for num in nums:
		  if num != 0:
		  nums[insert_pos] = num
		  insert_pos += 1
		  while insert_pos < len(nums):
		  nums[insert_pos] = 0
		  insert_pos +=1
		
		  """
		  Do not return anything, modify nums in-place instead.
		  """
			- Initialize a pointer to keep track of the position to insert non-zero elements
			- Iterate through the list; if the current element `nums[i]` does not equal 0
			- If the current element `nums[i]` is not equal to 0, move it to the current insert position and increment the insert position pointer.
				118 ms / 90.91%	18.09 mb / 78.27%				

1502. Can Make Arithmetic Progression From Sequence(EZ)	A sequence of numbers is called an arithmetic progression if the difference between any two consecutive elements is the same.
	
	Given an array of numbers arr, return true if the array can be rearranged to form an arithmetic progression. Otherwise, return false.
	
	Example 1:
	
	Input: arr = [3,5,1]
	Output: true
	Explanation: We can reorder the elements as [1,3,5] or [5,3,1] with differences 2 and -2 respectively, between each consecutive elements.
	Example 2:
	
	Input: arr = [1,2,4]
	Output: false
	Explanation: There is no way to reorder the elements to obtain an arithmetic progression.
	
	Constraints:
	
	2 <= arr.length <= 1000
	-106 <= arr[i] <= 106
		class Solution:
		  def canMakeArithmeticProgression(self, arr: List[int]) -> bool:
		  new = sorted(arr)
		  checklist = 0
		  x = new[1] - new[0]
		  for i in range(len(new) - 1):
		  if new[+ 1] - new[i] == x:
		  checklist += 1
		  else:
		  return False
		  if checklist == len(new) - 1:
		  return True
		  else:
		  return False
			- **INIT** flowers as 0
			- **INIT** i (position) as 0
			- **WHILE** i is less than the length of the flower bed:
			- **IF** i is still less than length
			- Check if the difference between the current position and the next position in the sorted array is the same as the initial difference (`new[i + 1] - new[i] == x`)
			- **IF** true:
			- Increment checklist by 1
			- **ELSE**:
			- **RETURN** False
			- **IF** checklist is equal to the length of the sorted array minus 1:
			- **RETURN** True
			- **ELSE**:
			- **RETURN** False
				42 ms / 59.24%	16.75 mb / 22.50%				

169. Majority Element (EZ)	Given a string s, reverse only all the vowels in the string and return it.
	
	The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both lower and upper cases, more than once.
	
	Example 1:
	
	Input: s = "hello"
	Output: "holle"
	Example 2:
	
	Input: s = "leetcode"
	Output: "leotcede"
	
	Constraints:
	
	1 <= s.length <= 3 * 105
	s consist of printable ASCII characters.
		class Solution:
		  def majorityElement(self, nums: List[int]) -> int:
		  new = sorted(nums)
		  max_count = 0
		  most_frequent_element = None
		  current_element = None
		  current_count = 0
		  for num in new:
		  if num == current_element:
		  current_count += 1
		  else:
		  if current_count > max_count:
		  max_count = current_count
		  most_frequent_element = current_element
		  current_element = num
		  current_count = 1
		
		  if current_count > max_count:
		  most_frequent_element = current_element
		  return most_frequent_element
			- The `majorityElement` function takes a list of integers as input.
			- It sorts the input list using the `sorted` function and assigns it to the variable `new`.
			- It initializes variables to track the most frequent element (`most_frequent_element`), its count (`max_count`), and the current element being processed (`current_element`) with their respective counts.
			- It iterates through the sorted list and updates the count of the current element.
			- After processing all elements, it returns the most frequent element found in the input list.
			- The return statement is properly indented outside the loop to ensure it only executes after processing the entire list.
				166 ms / 78.05%	18.04 mb / 35.55%				

136. Single Number (EZ)	Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.
	
	You must implement a solution with a linear runtime complexity and use only constant extra space.
	
	Example 1:
	
	Input: nums = [2,2,1]
	Output: 1
	Example 2:
	
	Input: nums = [4,1,2,1,2]
	Output: 4
	Example 3:
	
	Input: nums = [1]
	Output: 1
	
	Constraints:
	
	1 <= nums.length <= 3 * 104
	-3 * 104 <= nums[i] <= 3 * 104
	Each element in the array appears twice except for one element which appears only once.
		class Solution:
		  def singleNumber(self, nums: List[int]) -> int:
		  n = sorted(nums)
		  i = 0
		  while i < len(n) - 1:
		  if n[i] != n[i + 1]:
		  return n[i]
		  i += 2
		  return n[-1]
			- Sort the List: n = sorted(nums) sorts the list nums.
			- Initialize Index: i = 0 to start from the beginning of the list.
			- Iterate Through the List:
			- while i < len(n) - 1: Iterate through the list up to the second-to-last element to avoid index out-of-range errors.
			- if n[i] != n[i + 1]: Check if the current element is not the same as the next element.
			- return n[i]: If they are not the same, return the current element as it is the unique number.
			- i += 2: Skip the next element as it has been paired with the current element.
			- Return the Last Element: If the unique number is the last element in the list, return n[-1].
			- This solution ensures that the unique number is returned correctly as an integer, avoiding the TypeError.
				108 ms / 70.23%	19.14 mb / 34.07%				

657. Robot Return to Origin (EZ)	There is a robot starting at the position (0, 0), the origin, on a 2D plane. Given a sequence of its moves, judge if this robot ends up at (0, 0) after it completes its moves.
	
	You are given a string moves that represents the move sequence of the robot where moves[i] represents its ith move. Valid moves are 'R' (right), 'L' (left), 'U' (up), and 'D' (down).
	
	Return true if the robot returns to the origin after it finishes all of its moves, or false otherwise.
	
	Note: The way that the robot is "facing" is irrelevant. 'R' will always make the robot move to the right once, 'L' will always make it move left, etc. Also, assume that the magnitude of the robot's movement is the same for each move.
	
	Example 1:
	
	Input: moves = "UD"
	Output: true
	Explanation: The robot moves up once, and then down once. All moves have the same magnitude, so it ended up at the origin where it started. Therefore, we return true.
	Example 2:
	
	Input: moves = "LL"
	Output: false
	Explanation: The robot moves left twice. It ends up two "moves" to the left of the origin. We return false because it is not at the origin at the end of its moves.
		class Solution:
		  def judgeCircle(self, moves: str) -> bool:
		  count_U = moves.count('U')
		  count_R = moves.count('R')
		  count_L = moves.count('L')
		  count_D = moves.count('D')
		  if len(moves) % 2 != 0:
		  return(False)
		  if len(moves) >= 2:
		  if count_U == count_D and count_R == count_L:
		  return(True)
			- INIT the counts for each move type
			- Check if the length of moves is not even
			- IF NOT return FALSE
			- Check if length of moves are greater than or equal to 2 (after checking if even)
			- IF YES compare the count of the move types with their opposite
			- IF EVEN with their opposites return TRUE
				45 ms / 73.57 %	16.72 mb / 21.80%				

1619. Mean of Array After Removing Some Elements (EZ)	Given an integer array arr, return the mean of the remaining integers after removing the smallest 5% and the largest 5% of the elements.
	
	Answers within 10-5 of the actual answer will be considered accepted.
	
	Example 1:
	
	Input: arr = [1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3]
	Output: 2.00000
	Explanation: After erasing the minimum and the maximum values of this array, all elements are equal to 2, so the mean is 2.
	Example 2:
	
	Input: arr = [6,2,7,5,1,2,0,3,10,2,5,0,5,5,0,8,7,6,8,0]
	Output: 4.00000
	Example 3:
	
	Input: arr = [6,0,7,0,7,5,7,8,3,4,0,7,8,1,6,8,1,1,2,4,8,1,9,5,4,3,8,5,10,8,6,6,1,0,6,10,8,2,3,4]
	Output: 4.77778
	
	Constraints:
	
	20 <= arr.length <= 1000
	arr.length is a multiple of 20.
	0 <= arr[i] <= 105
		class Solution:
		  def trimMean(self, arr: List[int]) -> float:
		  if not arr:
		  return 0.0
		  remove_count = int(len(arr) * 0.05)
		  arr = sorted(arr)
		  arr = arr[remove_count : len(arr) - remove_count]
		
		  total = sum(arr)
		  mean = total / len(arr)
		
		  return mean
			- Check if the input list is empty.
			- if it is, return 0.0.
			- Calculate the number of elements to remove from both ends (5% of the length of the array).
			Sort the array.
			- Remove elements from both ends. Calculate the sum of the remaining elements.
			- Calculate the mean.
			- Return the mean.
				55 ms / 66.86%	16.74 mn / 72.75%				

1748. Sum of Unique Elements (EZ)	You are given an integer array nums. The unique elements of an array are the elements that appear exactly once in the array.
	
	Return the sum of all the unique elements of nums.
	
	Example 1:
	
	Input: nums = [1,2,3,2]
	Output: 4
	Explanation: The unique elements are [1,3], and the sum is 4.
	Example 2:
	
	Input: nums = [1,1,1,1,1]
	Output: 0
	Explanation: There are no unique elements, and the sum is 0.
	Example 3:
	
	Input: nums = [1,2,3,4,5]
	Output: 15
	Explanation: The unique elements are [1,2,3,4,5], and the sum is 15.
	
	Constraints:
	
	1 <= nums.length <= 100
	1 <= nums[i] <= 100
		class Solution:
		  def sumOfUnique(self, nums: List[int]) -> int:
		  nums = sorted(nums)
		  unique = []
		  i = 0
		  n = len(nums)
		  if len(nums) == 1:
		  return(sum(nums))
		  if sum(nums) / len(nums) == nums[0]:
		  return(0)
		  while i < n:
		  if (i == 0 or nums[i] != nums[i - 1]) and (i == n - 1 or nums[i] != nums[i + 1]):
		  unique.append(nums[i])
		  i += 1
		  result = sum(unique)
		  return(result)
			- Sorts the list `nums` and initializes variables for processing.
			- Checks if the list has only one element and returns its sum if true.
			- Checks if the average of the list equals the first element, returning 0 if true.
			- Iterates through the list to find unique elements (elements that are not duplicated).
			- Adds unique elements to a list `unique`.
			- Calculates the sum of the unique elements.
			- Returns the sum of the unique elements.
				34 ms / 80.79%	16.53 / 45.01 %				