**DSA – ASSIGNMENT 1**

💡 **Q1.** Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

**Example:** Input: nums = [2,7,11,15], target = 9 Output0 [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1]

**Solution. :-**

* Create an empty hash table to store the complement of each number.
* Iterate through the array nums using a for loop, with index and num as the loop variables.
  + Calculate the complement as complement = target - num.
  + Check if the complement exists in the hash table. If it does, return the indices [hash\_table[complement], index].
  + If the complement does not exist in the hash table, store the current number and its index in the hash table as hash\_table[num] = index.
* If no two numbers add up to the target, return an empty list or handle the error case as desired.

**def twoSum(nums, target):**

**hash\_table = {}**

**for index, num in enumerate(nums):**

**complement = target - num**

**if complement in hash\_table:**

**return [hash\_table[complement], index]**

**hash\_table[num] = index**

**# Handle error case if no two numbers add up to the target**

**return []**

**nums = [2, 7, 11, 15]**

**target = 9**

**result = twoSum(nums, target)**

**print(result)**

💡 **Q2.** Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val.

Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

* Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
* Return k.

**Example :** Input: nums = [3,2,2,3], val = 3 Output: 2, nums = [2,2,*\*,*\*]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 2. It does not matter what you leave beyond the returned k (hence they are underscores)[

**Solution. :-**

* Initialize two pointers, slow and fast, both initially set to 0.
* Iterate through the array with the fast pointer:
  + If the value at nums[fast] is not equal to the given val, assign it to nums[slow] and increment slow by 1.
* After the iteration, the elements from index 0 to slow - 1 will contain the elements that are not equal to val. Assign val to the remaining elements from slow to the end of the array.
* Return the value of slow, which represents the count of elements remaining.

**def removeElement(nums, val):**

**slow = 0**

**for fast in range(len(nums)):**

**if nums[fast] != val:**

**nums[slow] = nums[fast]**

**slow += 1**

**for i in range(slow, len(nums)):**

**nums[i] = val**

**return slow**

**nums = [3, 2, 2, 3]**

**val = 3**

**count = removeElement(nums, val)**

**print(count)**

**print(nums)**

💡 **Q3.** Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

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

**Example 1:** Input: nums = [1,3,5,6], target = 5

Output: 2

**Solution. :-**

* Initialize two pointers, left and right, to the start and end of the array respectively.
* While left is less than or equal to right, calculate the middle index mid as the average of left and right.
* Compare the value at index mid with the target value:
  + If the value at mid is equal to the target, return mid.
  + If the value at mid is greater than the target, update right to mid - 1.
  + If the value at mid is less than the target, update left to mid + 1.
* If the loop terminates without finding the target value, return the value of left, which represents the index where the target would be inserted.

**def searchInsert(nums, target):**

**left = 0**

**right = len(nums) - 1**

**while left <= right:**

**mid = (left + right) // 2**

**if nums[mid] == target:**

**return mid**

**elif nums[mid] < target:**

**left = mid + 1**

**else:**

**right = mid - 1**

**return left**

**nums = [1, 3, 5, 6]**

**target = 5**

**result = searchInsert(nums, target)**

**print(result)**

💡 **Q4.** You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

**Example 1:** Input: digits = [1,2,3] Output: [1,2,4]

**Explanation:** The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

**Solution. :-**

* Start from the least significant digit (digits[-1]) and add 1 to it. Initialize a carry variable to keep track of the carry-over, initially set to 1.
* Iterate over the digits array in reverse order, starting from the second least significant digit (digits[-2]).
  + Add the carry-over to the current digit.
  + If the sum is less than 10, update the current digit to the sum and set the carry variable to 0.
  + If the sum is 10 or greater, update the current digit to the sum modulo 10 (to get the ones digit) and set the carry variable to 1.
* After the iteration, if there is still a carry-over (carry = 1), insert it as a new most significant digit in the digits array.
* Return the modified digits array.

**def plusOne(digits):**

**carry = 1 # Start with a carry of 1**

**for i in range(len(digits) - 1, -1, -1):**

**digits[i] += carry**

**if digits[i] < 10:**

**carry = 0**

**break**

**else:**

**digits[i] %= 10**

**if carry == 1:**

**digits.insert(0, 1)**

**return digits**

**digits = [1, 2, 3]**

**result = plusOne(digits)**

**print(result)**

💡 **Q5.** You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

**Example 1:** Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 Output: [1,2,2,3,5,6]

**Explanation:** The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

**Solution. :-**

* Initialize three pointers: p1 for nums1 (starting from m - 1), p2 for nums2 (starting from n - 1), and p for the merged array (starting from m + n - 1).
* While p1 and p2 are both greater than or equal to 0 (indicating that there are still elements to compare):
  + If the element at nums1[p1] is greater than the element at nums2[p2], set nums1[p] to nums1[p1] and decrement p1 by 1.
  + Otherwise, set nums1[p] to nums2[p2] and decrement p2 by 1.
  + Decrement p by 1.
* After the while loop, there might be remaining elements in nums2. If p2 is greater than or equal to 0, iterate over the remaining elements in nums2 and place them in the correct positions in nums1.

**def merge(nums1, m, nums2, n):**

**p1 = m - 1**

**p2 = n - 1**

**p = m + n - 1**

**while p1 >= 0 and p2 >= 0:**

**if nums1[p1] > nums2[p2]:**

**nums1[p] = nums1[p1]**

**p1 -= 1**

**else:**

**nums1[p] = nums2[p2]**

**p2 -= 1**

**p -= 1**

**while p2 >= 0:**

**nums1[p] = nums2[p2]**

**p2 -= 1**

**p -= 1**

**nums1 = [1, 2, 3, 0, 0, 0]**

**m = 3**

**nums2 = [2, 5, 6]**

**n = 3**

**merge(nums1, m, nums2, n)**

**print(nums1)**

💡 **Q6.** 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

**Solution. :-**

* Initialize an empty hash set called seen.
* Iterate through each element num in the nums array.
  + If num is already in the seen set, return True as it indicates a duplicate.
  + Otherwise, add num to the seen set.
* After the iteration is complete, return False as no duplicates were found.

**def containsDuplicate(nums):**

**seen = set()**

**for num in nums:**

**if num in seen:**

**return True**

**seen.add(num)**

**return False**

**nums = [1,2,3,1]**

**result = containsDuplicate(nums)**

**print(result)**

💡 **Q7.** Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the nonzero 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]

**Solution. :-**

* Initialize two pointers, left and right, both starting from index 0.
* Iterate through the array with the right pointer:
  + If the element at the right pointer is non-zero, swap it with the element at the left pointer and increment both pointers.
  + If the element at the right pointer is zero, only increment the right pointer.
* Continue this process until the right pointer reaches the end of the array.
* All non-zero elements will now be placed towards the left side of the array, and all zeros will be towards the right side.
* Finally, iterate from the current position of the left pointer to the end of the array and set all remaining elements to zero.

**def moveZeroes(nums):**

**left = 0**

**right = 0**

**while right < len(nums):**

**if nums[right] != 0:**

**nums[left], nums[right] = nums[right], nums[left]**

**left += 1**

**right += 1**

**while left < len(nums):**

**nums[left] = 0**

**left += 1**

**nums = [0, 1, 0, 3, 12]**

**result = moveZeroes(nums)**

**print(result)**

💡 **Q8.** You have a set of integers s, which originally contains all the numbers from 1 to n. Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in repetition of one number and loss of another number.

You are given an integer array nums representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

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

**Solution. :-**

* Initialize an empty hash set called seen.
* Initialize two variables, duplicate and missing, to store the duplicated and missing numbers, respectively.
* Iterate through each element num in the nums array.
  + If num is not present in the seen set, add it to the set.
  + Otherwise, num is a duplicate, so assign it to the duplicate variable.
* Iterate from 1 to the length of nums + 1.
* If the current number is not present in the seen set, assign it to the missing variable.
* Return an array [duplicate, missing].

**def findErrorNums(nums):**

**seen = set()**

**duplicate = -1**

**missing = -1**

**for num in nums:**

**if num in seen:**

**duplicate = num**

**seen.add(num)**

**for num in range(1, len(nums) + 1):**

**if num not in seen:**

**missing = num**

**return [duplicate, missing]**

**n= int(input("Enter an integer value: "))**

**nums = [2, 3, 1, 4,4]**

**result = findErrorNums(nums)**

**print(result)**