**DSA – ASSIGNMENT 4**

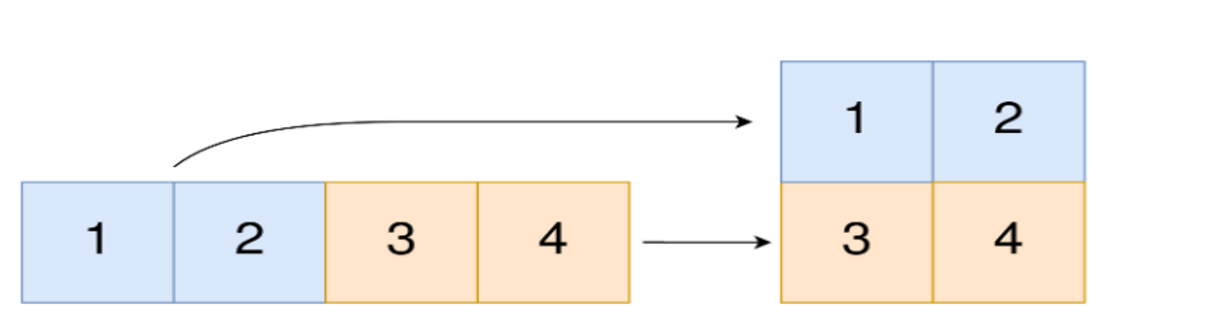
💡 **Question 1** Convert 1D Array Into 2D Array

You are given a **0-indexed** 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with  m rows and n columns using **all** the elements from original.

The elements from indices 0 to n - 1 (**inclusive**) of original should form the first row of the constructed 2D array, the elements from indices n to 2 \* n - 1 (**inclusive**) should form the second row of the constructed 2D array, and so on.

Return an m x n 2D array constructed according to the above procedure, or an empty 2D array if it is impossible.

**Example 1:**



**Input:** original = [1,2,3,4], m = 2, n = 2

**Output:** [[1,2],[3,4]]

**Explanation:** The constructed 2D array should contain 2 rows and 2 columns.

The first group of n=2 elements in original, [1,2], becomes the first row in the constructed 2D array.

The second group of n=2 elements in original, [3,4], becomes the second row in the constructed 2D array.

**Solution. :-**

* Check if the number of elements in the original array is equal to m \* n. If not, return an empty 2D array.
* Initialize an empty 2D array result with m rows and n columns.
* Iterate i from 0 to m - 1 (inclusive):
  + a. Get the i-th group of elements from the original array by slicing the array from i \* n to (i + 1) \* n.
  + b. Assign the group of elements to the i-th row of the result array.
* Return the result array.

**def convertTo2D(original, m, n):**

**if len(original) != m \* n:**

**return []**

**result = [[0] \* n for \_ in range(m)]**

**for i in range(m):**

**group = original[i \* n : (i + 1) \* n]**

**result[i] = group**

**return result**

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

**m = 2**

**n = 2**

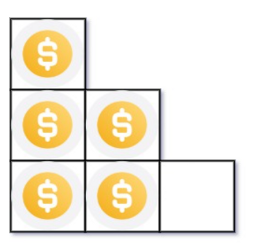
**result = convertTo2D(original, m, n)**

**print(result)**

💡 **Question 2** You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase **may be** incomplete.

Given the integer n, return the number of ***complete rows*** of the staircase you will build.

**Example 1:**



**Input:** n = 5

**Output:** 2

**Explanation:** Because the 3rd row is incomplete, we return 2.

**Solution. :-**

* Initialize variables left and right to 1 and n, respectively. These will represent the range of possible values for the number of rows.
* While left is less than or equal to right:
  + a. Calculate the mid value as mid = (left + right) // 2.
  + b. Calculate the sum of the first mid rows using the formula sum = (mid \* (mid + 1)) // 2.
  + c. If the sum is less than or equal to n, update left = mid + 1 since we can try a larger number of rows.
  + d. Otherwise, update right = mid - 1 since we need to try a smaller number of rows.
* Return right as the number of complete rows.

**def arrangeCoins(n):**

**left, right = 1, n**

**while left <= right:**

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

**sum = (mid \* (mid + 1)) // 2**

**if sum <= n:**

**left = mid + 1**

**else:**

**right = mid - 1**

**return right**

**n = 5**

**result = arrangeCoins(n)**

**print(result)**

💡 **Question 3** Given an integer array nums sorted in **non-decreasing** order, return *an array of* ***the squares of each number*** *sorted in non-decreasing order*.

**Example 1:**

**Input:** nums = [-4,-1,0,3,10]

**Output:** [0,1,9,16,100]

**Explanation:** After squaring, the array becomes [16,1,0,9,100].

After sorting, it becomes [0,1,9,16,100].

**Solution. :-**

* Initialize two pointers, left and right, to point to the start and end of the array nums, respectively.
* Initialize an empty array result to store the squared values.
* While left is less than or equal to right:
  + a. Compare the absolute values of nums[left] and nums[right].
  + b. If the absolute value of nums[left] is greater than or equal to the absolute value of nums[right]: i. Square nums[left] and append the squared value to result. ii. Move left one position to the right.
  + c. If the absolute value of nums[left] is less than the absolute value of nums[right]:
    - i. Square nums[right] and insert the squared value at the beginning of result.
    - ii. Move right one position to the left.
* Return result.

**def sortedSquares(nums):**

**n = len(nums)**

**left, right = 0, n - 1**

**result = [0] \* n**

**for i in range(n - 1, -1, -1):**

**if abs(nums[left]) >= abs(nums[right]):**

**result[i] = nums[left] \* nums[left]**

**left += 1**

**else:**

**result[i] = nums[right] \* nums[right]**

**right -= 1**

**return result**

**nums = [-4, -1, 0, 3, 10]**

**result = sortedSquares(nums)**

**print(result)**

💡 **Question 4** Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:*

* answer[0] *is a list of all* ***distinct*** *integers in* nums1 *which are* ***not*** *present in* nums2\*.\*
* answer[1] *is a list of all* ***distinct*** *integers in* nums2 *which are* ***not*** *present in* nums1.

**Note** that the integers in the lists may be returned in **any** order.

**Example 1:**

**Input:** nums1 = [1,2,3], nums2 = [2,4,6]

**Output:** [[1,3],[4,6]]

**Explanation:**

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].

**Solution. :-**

* Initialize two sets, set1 and set2, to store the distinct elements of nums1 and nums2, respectively.
* Iterate through each element num in nums1 and add it to set1.
* Iterate through each element num in nums2 and add it to set2.
* Initialize two empty lists, result1 and result2, to store the distinct integers not present in the other array.
* Iterate through each element num in set1 and check if it is not in set2. If it is not, append num to result1.
* Iterate through each element num in set2 and check if it is not in set1. If it is not, append num to result2.
* Return a list containing result1 and result2.

**def findDisappearedNumbers(nums1, nums2):**

**set1 = set(nums1)**

**set2 = set(nums2)**

**result1 = [num for num in set1 if num not in set2]**

**result2 = [num for num in set2 if num not in set1]**

**return [result1, result2]**

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

**nums2 = [2, 4, 6]**

**result = findDisappearedNumbers(nums1, nums2)**

**print(result)**

💡 **Question 5** Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

**Example 1:**

**Input:** arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2

**Output:** 2

**Explanation:**

For arr1[0]=4 we have:

|4-10|=6 > d=2

|4-9|=5 > d=2

|4-1|=3 > d=2

|4-8|=4 > d=2

For arr1[1]=5 we have:

|5-10|=5 > d=2

|5-9|=4 > d=2

|5-1|=4 > d=2

|5-8|=3 > d=2

For arr1[2]=8 we have:

**|8-10|=2 <= d=2**

**|8-9|=1 <= d=2**

|8-1|=7 > d=2

**|8-8|=0 <= d=2**

**Solution. :-**

* Initialize a variable distance to 0, which will store the count of elements in arr1 that satisfy the distance condition.
* Iterate through each element num1 in arr1.
* For each element num1, iterate through each element num2 in arr2.
* Check if the absolute difference between num1 and num2 is greater than d. If it is, continue to the next element in arr2.
* If the loop completes without finding any element in arr2 that satisfies the condition, increment distance by 1.
* After the loops, return the value of distance.

**def distanceValue(arr1, arr2, d):**

**distance = 0**

**for num1 in arr1:**

**for num2 in arr2:**

**if abs(num1 - num2) <= d:**

**break**

**else:**

**distance += 1**

**return distance**

**arr1 = [4, 5, 8]**

**arr2 = [10, 9, 1, 8]**

**d = 2**

**result = distanceValue(arr1, arr2, d)**

**print(result)**

💡 **Question 6** Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears **once** or **twice**, return *an array of all the integers that appears* ***twice***.

You must write an algorithm that runs in O(n) time and uses only constant extra space.

**Example 1:**

**Input:** nums = [4,3,2,7,8,2,3,1]

**Output:**

[2,3]

**Solution. :-**

* Initialize an empty array called duplicates to store the integers that appear twice.
* Iterate through each element num in the nums array.
* Take the absolute value of num and subtract 1 to convert it into a valid index.
* If the value at index nums[abs(num)-1] is negative, it means abs(num) has already appeared before and is a duplicate. Append abs(num) to the duplicates array.
* Otherwise, negate the value at index nums[abs(num)-1] to mark the presence of abs(num).
* After the loop, return the duplicates array.

**def findDuplicates(nums):**

**duplicates = []**

**for num in nums:**

**index = abs(num) - 1**

**if nums[index] < 0:**

**duplicates.append(abs(num))**

**else:**

**nums[index] \*= -1**

**return duplicates**

**nums = [4, 3, 2, 7, 8, 2, 3, 1]**

**duplicates = findDuplicates(nums)**

**print(duplicates)**

💡 **Question 7** Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

* [4,5,6,7,0,1,2] if it was rotated 4 times.
* [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that **rotating** an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in O(log n) time.

**Example 1:**

**Input:** nums = [3,4,5,1,2]

**Output:** 1

**Explanation:**

The original array was [1,2,3,4,5] rotated 3 times.

**Solution. :-**

* Initialize two pointers, left and right, pointing to the first and last elements of the array, respectively.
* While left is less than right, do the following:
  + Calculate the middle index as mid = left + (right - left) // 2.
  + If nums[mid] is greater than nums[right], it means the minimum element is in the right half of the array. Update left = mid + 1.
  + Otherwise, if nums[mid] is less than or equal to nums[right], it means the minimum element is in the left half of the array or it is the current element. Update right = mid.
* At the end of the loop, left and right will point to the minimum element in the array.
* Return nums[left] or nums[right], as they will be equal.

**def findMin(nums):**

**left = 0**

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

**while left < right:**

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

**if nums[mid] > nums[right]:**

**left = mid + 1**

**else:**

**right = mid**

**return nums[left]**

**nums = [3, 4, 5, 1, 2]**

**min\_element = findMin(nums)**

**print(min\_element)**

💡 **Question 8** An integer array original is transformed into a **doubled** array changed by appending **twice the value** of every element in original, and then randomly **shuffling** the resulting array.

Given an array changed, return original *if* changed *is a* ***doubled*** *array. If* changed *is not a* ***doubled*** *array, return an empty array. The elements in* original *may be returned in* ***any*** *order*.

**Example 1:**

**Input:** changed = [1,3,4,2,6,8]

**Output:** [1,3,4]

**Explanation:** One possible original array could be [1,3,4]:

* Twice the value of 1 is 1 \* 2 = 2.
* Twice the value of 3 is 3 \* 2 = 6.
* Twice the value of 4 is 4 \* 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4].

**Solution. :-**

* Initialize an empty dictionary called count\_dict.
* Iterate through each element num in the changed array.
* Check if num is divisible by 2. If it is not, return an empty array since changed cannot be a doubled array.
* Calculate the original value by dividing num by 2.
* If the original value is not present in count\_dict, set its count to 1.
* If the original value is already in count\_dict, increment its count by 1.
* After the loop, iterate through each element in count\_dict and check if the count is not equal to 2. If any element has a count different from 2, return an empty array.
* If all elements have a count of 2, create an empty array called original.
* Iterate through each key-value pair in count\_dict, and for each key num, append num to the original array count number of times.
* Return the original array.

**from collections import defaultdict**

**def findOriginalArray(changed):**

**count\_dict = defaultdict(int)**

**for num in changed:**

**if num % 2 != 0:**

**return []**

**original = num // 2**

**count\_dict[original] += 1**

**for num, count in count\_dict.items():**

**if count != 2:**

**return []**

**original = []**

**for num, count in count\_dict.items():**

**original.extend([num] \* count)**

**return original**

**changed = [1, 3, 4, 2, 6, 8]**

**original = findOriginalArray(changed)**

**print(original)**