Assignment 05 Solutions

Q1. 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]]

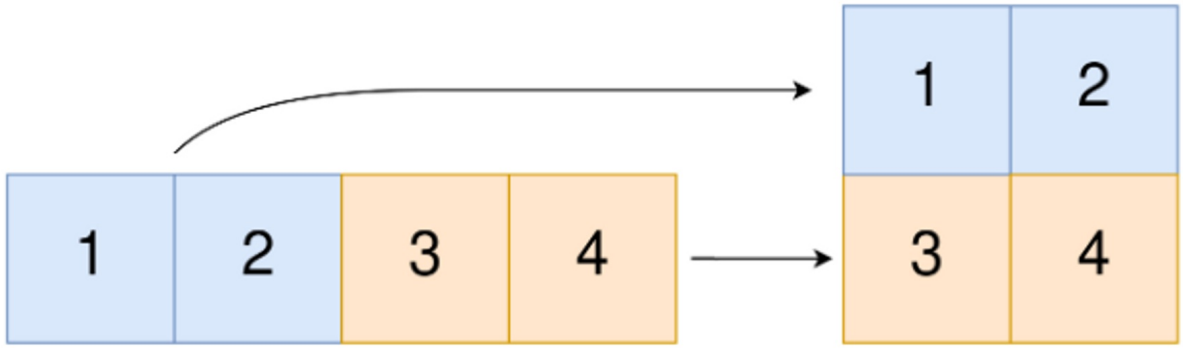
Explanation: The constructed 2D array should contain 2 rows and 2 columns.

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.

In [22]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\Screenshot_2023-05-29_004311.png")
```

Out[22]:



In [23]:
```
from typing import List

def construct2DArray(original: List[int], m: int, n: int) -> List[List[int]]:
    if len(original) != m * n:
        return []

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

    for i, num in enumerate(original):
        ans[i // n][i % n] = num

    return ans
```

In [24]:
```
original = [1, 2, 3, 4]
m = 2
n = 2
result = construct2DArray(original, m, n)
print(result)
```

```
[[1, 2], [3, 4]]
```

Q2. **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:

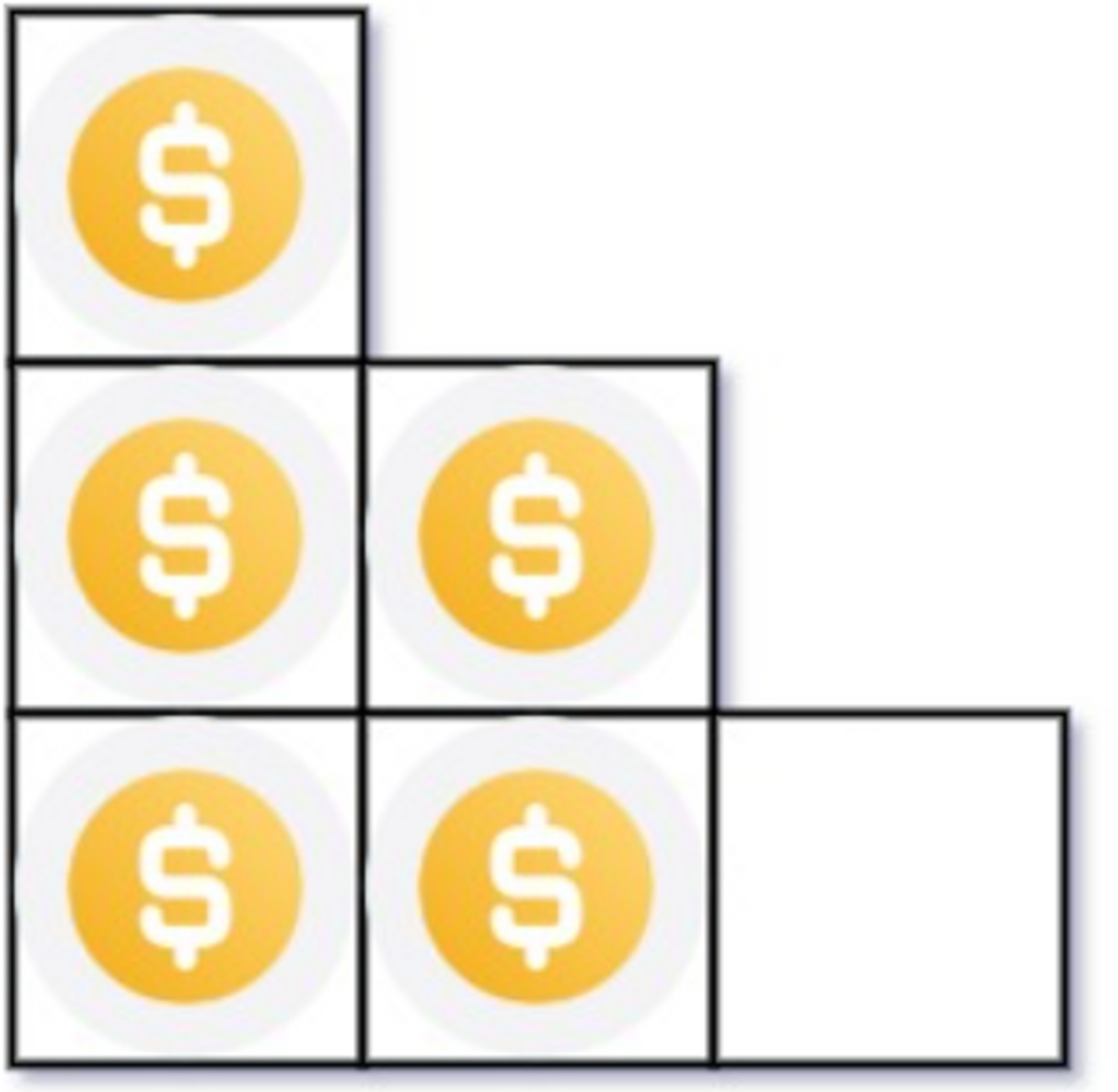
Input: n = 5

Output: 2

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

In [25]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\sda.png")
```

Out[25]:



In [26]:
```
def arrangeCoins(n: int) -> int:
    rows = 0
    i = 1

    while n >= i:
        n -= i
        rows += 1
        i += 1

    return rows

n = 5
result = arrangeCoins(n)
print(result)
```

```
2
```

Q3. **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].

In [27]:
```
def sorted_squares(nums):
    result = []
    for num in nums:
        square = num * num
        result.append(square)
    return sorted(result)
```

In [28]:
```
sorted_squares([-4, -1, 0, 3, 10])
```

Out[28]:
```
[0, 1, 9, 16, 100]
```

Q4. 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].

In [29]:
```
def findDisappearedNumbers(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)
    result1 = []
    result2 = []

    for num in nums1:
        if num not in set2:
            result1.append(num)

    for num in nums2:
        if num not in set1:
            result2.append(num)

    return [result1, result2]
```

In [30]:
```
nums1 = [1, 2, 3]
nums2 = [2, 4, 6]
print(findDisappearedNumbers(nums1, nums2))
```

```
[[1, 3], [4, 6]]
```

Q5. 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

In [31]:
```
def findDistanceValue(arr1, arr2, d):
    distance = 0
```

```
    for num1 in arr1:
        found = False
        for num2 in arr2:
            if abs(num1 - num2) <= d:
                found = True
                break
        if not found:
            distance += 1
    return distance
```

In [32]:
```
arr1 = [4, 5, 8]
arr2 = [10, 9, 1, 8]
d = 2
print(findDistanceValue(arr1, arr2, d))
```

```
2
```

Q6. **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]

In [33]:
```
def findDuplicates(nums):
    result = []
    for num in nums:
        index = abs(num) - 1
        if nums[index] > 0:
            nums[index] = -nums[index]
        else:
            result.append(abs(num))
    return result
```

In [34]:
```
nums = [4, 3, 2, 7, 8, 2, 3, 1]
print(findDuplicates(nums))
```

```
[2, 3]
```

Q7. **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.

In [35]:
```
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]
```

In [36]:
```
nums = [3, 4, 5, 1, 2]
print(findMin(nums))
```

```
1
```

Q8. **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].

In [37]:
```
from collections import defaultdict

def findOriginalArray(changed):
    if len(changed) % 2 != 0:
        return []

    freq = defaultdict(int)
    original = []

    for num in changed:
        freq[num] += 1

    for num in sorted(changed):
        if freq[num] > 0:
            if freq[num * 2] > 0:
                freq[num * 2] -= 1
                original.append(num)
            else:
                return []

    return original
```

In [38]:
```
changed = [1, 3, 4, 2, 6, 8]
print(findOriginalArray(changed))
```

```
[1, 3, 4]
```

In [ ]: