Sorting assignment. 🫠

Q1) Sort colors.
Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.
We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.
You must solve this problem without using the library's sort function.

Code:

```
def sortColors(nums):
    low, mid, high = 0, 0, len(nums) - 1

    while mid <= high:
        if nums[mid] == 0:
            nums[low], nums[mid] = nums[mid], nums[low]
            low += 1
            mid += 1
        elif nums[mid] == 1:
            mid += 1
        else:
            nums[mid], nums[high] = nums[high], nums[mid]
            high -= 1

# Example usage
nums1 = [2, 0, 2, 1, 1, 0]
sortColors(nums1)
print(nums1)  # Output: [0, 0, 1, 1, 2, 2]

nums2 = [2, 0, 1]
sortColors(nums2)
print(nums2)  # Output: [0, 1, 2]
```

Q2)A sentence is a list of words that are separated by a single space with no leading or trailing spaces. Each word consists of lowercase and uppercase English letters.

A sentence can be shuffled by appending the 1-indexed word position to each word then rearranging the words in the sentence.

For example, the sentence "This is a sentence" can be shuffled as "sentence4 a3 is2 This1" or "is2 sentence4 This1 a3".

Given a shuffled sentence s containing no more than 9 words, reconstruct and return the original sentence.

Code:

```
def reconstructSentence(s):
    # Split the sentence into words
    words = s.split()

    # Extract the index and sort the words based on the index
    words.sort(key=lambda word: int(word[-1]))

    # Remove the index from each word
    original_sentence = ' '.join(word[:-1] for word in words)

    return original_sentence

# Example usage
s1 = "is2 sentence4 This1 a3"
print(reconstructSentence(s1))  # Output: "This is a sentence"

s2 = "Myself2 Me1 I4 and3"
print(reconstructSentence(s2))  # Output: "Me Myself and I"
```

Q3)Minimum Number Game

Code:

```
def playGame(nums):
    # Sort the nums array
    nums.sort()

    # Initialize the result array
    arr = []

    # Use two pointers to simulate the game
    left, right = 0, len(nums) - 1

    while left < right:
        # Alice removes the minimum element
        alice_choice = nums[left]
        left += 1
```

```
        # Bob removes the next minimum element
        bob_choice = nums[left]
        left += 1

        # Bob appends first, then Alice
        arr.append(bob_choice)
        arr.append(alice_choice)

    return arr

# Example usage
nums1 = [5, 4, 2, 3]
print(playGame(nums1))  # Output: [3, 2, 5, 4]

nums2 = [2, 5]
print(playGame(nums2))  # Output: [5, 2]
```

Q4)Average Salary Excluding the Minimum and Maximum

Code:

```
def average(salary):
    # Find the minimum and maximum salary
    min_salary = min(salary)
    max_salary = max(salary)

    # Calculate the sum excluding the minimum and maximum salary
    total_sum = sum(salary) - min_salary - max_salary

    # Calculate the number of remaining salaries
    count = len(salary) - 2

    # Compute the average
    average_salary = total_sum / count

    return average_salary

# Example usage
salary1 = [4000, 3000, 1000, 2000]
print(f"{average(salary1):.5f}")  # Output: 2500.00000
```

```
salary2 = [1000, 2000, 3000]
print(f"{average(salary2):.5f}")  # Output: 2000.00000
```

Q5)Sort Even and Odd Indices Independently.

Code:

```
def rearrangeArray(nums):
    # Extract values at odd and even indices
    odd_values = [nums[i] for i in range(1, len(nums), 2)]
    even_values = [nums[i] for i in range(0, len(nums), 2)]

    # Sort odd values in non-increasing order
    odd_values.sort(reverse=True)

    # Sort even values in non-decreasing order
    even_values.sort()

    # Merge the sorted values back into the original array
    result = []
    odd_index, even_index = 0, 0
    for i in range(len(nums)):
        if i % 2 == 0:
            result.append(even_values[even_index])
            even_index += 1
        else:
            result.append(odd_values[odd_index])
            odd_index += 1

    return result

# Example usage
nums1 = [4, 1, 2, 3]
print(rearrangeArray(nums1))  # Output: [2, 3, 4, 1]

nums2 = [2, 1]
print(rearrangeArray(nums2))  # Output: [2, 1]
```

Q6)K Weakest Rows in a Matrix

Code:

```
def kWeakestRows(mat, k):
    # Calculate the number of soldiers in each row
    soldier_count = [(sum(row), i) for i, row in enumerate(mat)]

    # Sort the rows by the number of soldiers and then by row index
    soldier_count.sort()

    # Extract the indices of the first k rows
    weakest_rows = [index for _, index in soldier_count[:k]]

    return weakest_rows

# Example usage
mat1 = [
    [1, 1, 0, 0, 0],
    [1, 1, 1, 1, 0],
    [1, 0, 0, 0, 0],
    [1, 1, 0, 0, 0],
    [1, 1, 1, 1, 1]
]
k1 = 3
print(kWeakestRows(mat1, k1))  # Output: [2, 0, 3]

mat2 = [
    [1, 0, 0, 0],
    [1, 1, 1, 1],
    [1, 0, 0, 0],
    [1, 0, 0, 0]
]
k2 = 2
print(kWeakestRows(mat2, k2))  # Output: [0, 2]
```

Q7) Squares of a Sorted Array.

Code:

```
def sortedSquares(nums):
    n = len(nums)
```

```
    result = [0] * n
    left, right = 0, n - 1
    position = n - 1

    while left <= right:
        left_square = nums[left] ** 2
        right_square = nums[right] ** 2

        if left_square > right_square:
            result[position] = left_square
            left += 1
        else:
            result[position] = right_square
            right -= 1

        position -= 1

    return result

# Example usage
nums1 = [-4, -1, 0, 3, 10]
print(sortedSquares(nums1))  # Output: [0, 1, 9, 16, 100]

nums2 = [-7, -3, 2, 3, 11]
print(sortedSquares(nums2))  # Output: [4, 9, 9, 49, 121]
```

Q8)Height Checker.

Code:

```
def heightChecker(heights):
    # Create the expected array by sorting the heights array
    expected = sorted(heights)

    # Count the number of indices where heights[i] != expected[i]
    count = sum(1 for i in range(len(heights)) if heights[i] != expected[i])

    return count

# Example usage
heights1 = [1, 1, 4, 2, 1, 3]
print(heightChecker(heights1))  # Output: 3
```

```
heights2 = [5, 1, 2, 3, 4]
print(heightChecker(heights2))  # Output: 5

heights3 = [1, 2, 3, 4, 5]
print(heightChecker(heights3))  # Output: 0
```

Q9)Relative Ranks.

Code:

```
def findRelativeRanks(score):
    # Create a list of tuples (score, index)
    score_with_index = [(s, i) for i, s in enumerate(score)]

    # Sort the list by score in descending order
    score_with_index.sort(reverse=True, key=lambda x: x[0])

    # Initialize the result array with the same length as score
    result = [""] * len(score)

    # Assign ranks based on the sorted order
    for rank, (s, i) in enumerate(score_with_index):
        if rank == 0:
            result[i] = "Gold Medal"
        elif rank == 1:
            result[i] = "Silver Medal"
        elif rank == 2:
            result[i] = "Bronze Medal"
        else:
            result[i] = str(rank + 1)

    return result

# Example usage
score1 = [5, 4, 3, 2, 1]
print(findRelativeRanks(score1))  # Output: ["Gold Medal", "Silver Medal", "Bronze
Medal", "4", "5"]
```

```
score2 = [10, 3, 8, 9, 4]
print(findRelativeRanks(score2))  # Output: ["Gold Medal", "5", "Bronze Medal", "Silver Medal", "4"]
```

Q10)Find Target Indices After Sorting Array.

Code:

```
def targetIndices(nums, target):
    # Sort the array in non-decreasing order
    nums.sort()

    # Find the indices of the target element
    result = [i for i, num in enumerate(nums) if num == target]

    return result

# Example usage
nums1 = [1, 2, 5, 2, 3]
target1 = 2
print(targetIndices(nums1, target1))  # Output: [1, 2]

nums2 = [1, 2, 5, 2, 3]
target2 = 3
print(targetIndices(nums2, target2))  # Output: [3]

nums3 = [1, 2, 5, 2, 3]
target3 = 5
print(targetIndices(nums3, target3))  # Output: [4]
```

Q11)Special Array with X elements Greater than or equal X

Code:

```
def specialArray(nums):
```

```
    # Sort the array in non-decreasing order
    nums.sort()

    # Iterate through the array to find the special number
    for x in range(1, len(nums) + 1):
        if len([num for num in nums if num >= x]) == x:
            return x

    return -1

# Example usage
nums1 = [3, 5]
print(specialArray(nums1))  # Output: 2

nums2 = [0, 0]
print(specialArray(nums2))  # Output: -1

nums3 = [0, 4, 3, 0, 4]
print(specialArray(nums3))  # Output: 3
```

Q12)Buy Two Chocolates.

Code:

```
def buyChocolates(prices, money):
    # Sort the prices array
    prices.sort()

    # Initialize two pointers
    left, right = 0, len(prices) - 1

    # Initialize the minimum leftover money to the initial money
    min_leftover = money

    # Iterate through the array to find the best pair of chocolates
    while left < right:
        total_cost = prices[left] + prices[right]

        if total_cost <= money:
            # Calculate the leftover money
```

```
            leftover = money - total_cost
            # Update the minimum leftover money
            min_leftover = min(min_leftover, leftover)
            # Move the left pointer to the right to increase the total cost
            left += 1
        else:
            # Move the right pointer to the left to decrease the total cost
            right -= 1

    return min_leftover

# Example usage
prices1 = [1, 2, 2]
money1 = 3
print(buyChocolates(prices1, money1))  # Output: 0

prices2 = [3, 2, 3]
money2 = 3
print(buyChocolates(prices2, money2))  # Output: 3
```

Q13)How many numbers are smaller than the current.

Code:

```
def smallerNumbersThanCurrent(nums):
    result = []
    for i in range(len(nums)):
        count = 0
        for j in range(len(nums)):
            if nums[j] < nums[i]:
                count += 1
        result.append(count)
    return result

# Example usage
nums1 = [8, 1, 2, 2, 3]
print(smallerNumbersThanCurrent(nums1))  # Output: [4, 0, 1, 1, 3]

nums2 = [6, 5, 4, 8]
print(smallerNumbersThanCurrent(nums2))  # Output: [2, 1, 0, 3]
```

```
nums3 = [7, 7, 7, 7]
print(smallerNumbersThanCurrent(nums3))  # Output: [0, 0, 0, 0]
```

Q14)Sort Array by Increasing Frequency.

Code:

```
from collections import Counter

def frequencySort(nums):
    # Count the frequency of each number
    freq = Counter(nums)

    # Sort the numbers based on frequency and value
    nums.sort(key=lambda x: (freq[x], -x))

    return nums

# Example usage
nums1 = [1, 1, 2, 2, 2, 3]
print(frequencySort(nums1))  # Output: [3, 1, 1, 2, 2, 2]

nums2 = [2, 3, 1, 3, 2]
print(frequencySort(nums2))  # Output: [1, 3, 3, 2, 2]

nums3 = [-1, 1, -6, 4, 5, -6, 1, 4, 1]
print(frequencySort(nums3))  # Output: [5, -1, 4, 4, -6, -6, 1, 1, 1]
```

Q15)Set Mismatch.

Code:

```
def findErrorNums(nums):
    n = len(nums)
    sum_n = n * (n + 1) // 2
    sum_n_sq = n * (n + 1) * (2 * n + 1) // 6
```

```
    sum_nums = sum(nums)
    sum_nums_sq = sum(x * x for x in nums)

    diff = sum_n - sum_nums
    sq_diff = sum_n_sq - sum_nums_sq

    sum_diff = sq_diff // diff

    missing = (diff + sum_diff) // 2
    duplicate = sum_diff - missing

    return [duplicate, missing]

# Example usage
nums1 = [1, 2, 2, 4]
print(findErrorNums(nums1))  # Output: [2, 3]

nums2 = [1, 1]
print(findErrorNums(nums2))  # Output: [1, 2]
```

Q16)Find all numbers disappeared in an array.

Code:

```
def findDisappearedNumbers(nums):
    # Mark each number's corresponding index as visited
    for num in nums:
        index = abs(num) - 1
        if nums[index] > 0:
            nums[index] = -nums[index]

    # Collect all indices that were not visited
    result = [i + 1 for i in range(len(nums)) if nums[i] > 0]

    return result

# Example usage
nums1 = [4, 3, 2, 7, 8, 2, 3, 1]
print(findDisappearedNumbers(nums1))  # Output: [5, 6]
```

```
nums2 = [1, 1]
print(findDisappearedNumbers(nums2))  # Output: [2]
```

Q17)Find the Duplicate number.

Code:

```
def findDuplicate(nums):
    # Initialize the tortoise and hare
    tortoise = hare = nums[0]

    # Phase 1: Finding the intersection point of the two runners
    while True:
        tortoise = nums[tortoise]
        hare = nums[nums[hare]]
        if tortoise == hare:
            break

    # Phase 2: Finding the entrance to the cycle
    tortoise = nums[0]
    while tortoise != hare:
        tortoise = nums[tortoise]
        hare = nums[hare]

    return hare

# Example usage
nums1 = [1, 3, 4, 2, 2]
print(findDuplicate(nums1))  # Output: 2

nums2 = [3, 1, 3, 4, 2]
print(findDuplicate(nums2))  # Output: 3

nums3 = [3, 3, 3, 3, 3]
print(findDuplicate(nums3))  # Output: 3
```

Q18)Find All Duplicates in an Array.

Code:

```
def findDuplicates(nums):
    result = []

    for num in nums:
        index = abs(num) - 1
        if nums[index] < 0:
            result.append(abs(num))
        else:
            nums[index] = -nums[index]

    return result

# Example usage
nums1 = [4, 3, 2, 7, 8, 2, 3, 1]
print(findDuplicates(nums1))  # Output: [2, 3]

nums2 = [1, 1, 2]
print(findDuplicates(nums2))  # Output: [1]

nums3 = [1]
print(findDuplicates(nums3))  # Output: []
```

Q19)Missing Number.

Code:

```
def missingNumber(nums):
    n = len(nums)
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(nums)
    return expected_sum - actual_sum
```

```
# Example usage
nums1 = [3, 0, 1]
print(missingNumber(nums1))  # Output: 2

nums2 = [0, 1]
print(missingNumber(nums2))  # Output: 2

nums3 = [9, 6, 4, 2, 3, 5, 7, 0, 1]
print(missingNumber(nums3))  # Output: 8
```