Q. Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

SymbolValue

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

* I can be placed before V (5) and X (10) to make 4 and 9.
* X can be placed before L (50) and C (100) to make 40 and 90.
* C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

**Example 1:**

Input: s = "III"

Output: 3

Explanation: III = 3.

Input: s = "LVIII"

Output: 58

Explanation: L = 50, V= 5, III = 3.

**Constraints:**

* 1 <= s.length <= 15
* s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
* It is **guaranteed** that s is a valid roman numeral in the range [1, 3999].

Q. **Longest Substring Without Repeating Characters**

Given a string s, find the length of the **longest substring** without repeating characters.

**Example 1:**

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Input: s = "bbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

Input: s = "pwwkew"

Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

**Constraints:**

* 0 <= s.length <= 50000
* s consists of English letters, digits, symbols and spaces.

Q. <aside> 💡 3. **Majority Element**

Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Input: nums = [3,2,3]

Output: 3

Input: nums = [2,2,1,1,1,2,2]

Output: 2

**Constraints:**

* n == nums.length
* 1 <= n <= 5 \* 10^4
* -10^9 <= nums[i] <= 10^9

Q. <aside> 💡 4. **Group Anagram**

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

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: strs = ["eat","tea","tan","ate","nat","bat"]

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Input: strs = ["a"]

Output: [["a"]]

**Constraints:**

* 1 <= strs.length <= 10000
* 0 <= strs[i].length <= 100
* strs[i] consists of lowercase English letters.

Q.. **Ugly Numbers**

An **ugly number** is a positive integer whose prime factors are limited to 2, 3, and 5.

Given an integer n, return *the* nth ***ugly number***.

**Example 1:**

Input: n = 10

Output: 12

Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the first 10 ugly numbers.

Input: n = 1

Output: 1

Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.

**Constraints:**

* 1 <= n <= 1690

Q. Given an array of strings words and an integer k, return *the* k *most frequent strings*.

Return the answer **sorted** by **the frequency** from highest to lowest. Sort the words with the same frequency by their **lexicographical order**.

**Example 1:**

Input: words = ["i","love","leetcode","i","love","coding"], k = 2

Output: ["i","love"]

Explanation: "i" and "love" are the two most frequent words.

Note that "i" comes before "love" due to a lower alphabetical order.

Input: words = ["the","day","is","sunny","the","the","the","sunny","is","is"], k = 4

Output: ["the","is","sunny","day"]

Explanation: "the", "is", "sunny" and "day" are the four most frequent words, with the number of occurrence being 4, 3, 2 and 1 respectively.

**Constraints:**

* 1 <= words.length <= 500
* 1 <= words[i].length <= 10
* words[i] consists of lowercase English letters.
* k is in the range [1, The number of \*\*unique\*\* words[i]]

Q. **Sliding Window Maximum**

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window*.

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3

Output: [3,3,5,5,6,7]

Explanation:

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 3

1 [3 -1 -3] 5 3 6 7 3

1 3 [-1 -3 5] 3 6 7 5

1 3 -1 [-3 5 3] 6 7 5

1 3 -1 -3 [5 3 6]7 6

1 3 -1 -3 5 [3 6 7] 7

Input: nums = [1], k = 1

Output: [1]

**Constraints:**

* 1 <= nums.length <= 100000
* -10000 <= nums[i] <= 10000
* 1 <= k <= nums.length

Q. **Find K Closest Elements**

Given a **sorted** integer array arr, two integers k and x, return the k closest integers to x in the array. The result should also be sorted in ascending order.

An integer a is closer to x than an integer b if:

* |a - x| < |b - x|, or
* |a - x| == |b - x| and a < b

Input: arr = [1,2,3,4,5], k = 4, x = 3

Output: [1,2,3,4]

Input: arr = [1,2,3,4,5], k = 4, x = -1

Output: [1,2,3,4]

**Constraints:**

* 1 <= k <= arr.length
* 1 <= arr.length <= 10000
* arr is sorted in **ascending** order.
* -10000 <= arr[i], x <= 10000

Answers:

Sure! I can help you with those questions. Here are the solutions:

Q1. Roman to Integer

```python

def romanToInt(s):

values = {

'I': 1,

'V': 5,

'X': 10,

'L': 50,

'C': 100,

'D': 500,

'M': 1000

}

result = 0

prev\_value = 0

for i in range(len(s) - 1, -1, -1):

current\_value = values[s[i]]

if current\_value >= prev\_value:

result += current\_value

else:

result -= current\_value

prev\_value = current\_value

return result

```

Q2. Longest Substring Without Repeating Characters

```python

def lengthOfLongestSubstring(s):

if len(s) == 0:

return 0

char\_map = {}

start = 0

max\_length = 0

for i in range(len(s)):

if s[i] in char\_map and start <= char\_map[s[i]]:

start = char\_map[s[i]] + 1

else:

max\_length = max(max\_length, i - start + 1)

char\_map[s[i]] = i

return max\_length

```

Q3. Majority Element

```python

def majorityElement(nums):

count = 0

majority = None

for num in nums:

if count == 0:

majority = num

if num == majority:

count += 1

else:

count -= 1

return majority

```

Q4. Group Anagrams

```python

from collections import defaultdict

def groupAnagrams(strs):

groups = defaultdict(list)

for word in strs:

key = ''.join(sorted(word))

groups[key].append(word)

return list(groups.values())

```

Q5. Ugly Numbers

```python

def nthUglyNumber(n):

ugly\_numbers = [1]

index\_2 = index\_3 = index\_5 = 0

while len(ugly\_numbers) < n:

next\_ugly = min(ugly\_numbers[index\_2] \* 2, ugly\_numbers[index\_3] \* 3, ugly\_numbers[index\_5] \* 5)

if next\_ugly == ugly\_numbers[index\_2] \* 2:

index\_2 += 1

if next\_ugly == ugly\_numbers[index\_3] \* 3:

index\_3 += 1

if next\_ugly == ugly\_numbers[index\_5] \* 5:

index\_5 += 1

ugly\_numbers.append(next\_ugly)

return ugly\_numbers[-1]

```

Q6. Top K Frequent Words

```python

from collections import Counter

def topKFrequent(words, k):

counter = Counter(words)

sorted\_words = sorted(counter.keys(), key=lambda x: (-counter[x], x))

return sorted\_words[:k]

```

Q7. Sliding Window Maximum

```python

from collections import deque

def maxSlidingWindow(nums, k):

result = []

window = deque()

for i in range(len(nums)):

while window and nums[i] >= nums[window[-1]]:

window.pop()

window.append(i)

if window[0] == i - k:

window.popleft()

if i >= k - 1:

result.append(nums[window[0]])

return result

```

Q8. Find K Closest Elements

```python

def findClosestElements(arr, k, x):

left = 0

right = len(arr) - k

while left < right:

mid = (left + right) // 2

if x - arr[mid] > arr[mid + k] - x:

left = mid + 1

else:

right = mid

return arr[left:left + k]

```