**Assignment 24 - Basics of DSA | Revision Class**

**Question 1.** **Roman to Integer**

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.

**Example 2:**

Input: s = "LVIII"

Output: 58

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

**Sol.**

def romanToInt(s: str) -> int:  
 # Dictionary of roman numerals  
 roman\_map = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}  
 # Length of the given string  
 n = len(s)  
 # This variable will store result  
 num = roman\_map[s[n - 1]]  
 # Loop for each character from right to left  
 for i in range(n - 2, -1, -1):  
 # Check if the character at right of current character is bigger or smaller  
 if roman\_map[s[i]] >= roman\_map[s[i + 1]]:  
 num += roman\_map[s[i]]  
 else:  
 num -= roman\_map[s[i]]  
 return num  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(romanToInt('III'))  
 print(romanToInt('LVIII'))

**Question 2**. **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.

**Example 2:**

Input: s = "bbbbb"

Output: 1

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

**Example 3:**

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.

**Sol:**

class Solution:

def lengthOfLongestSubstring(self, s: str) -> int:

ss = set()

i = ans = 0

for j, c in enumerate(s):

while c in ss:

ss.remove(s[i])

i += 1

ss.add(c)

ans = max(ans, j - i + 1)

return ans

**Question 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.

**Example 1:**

Input: nums = [3,2,3]

Output: 3

**Example 2:**

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

**Sol:**

class Solution(object):

def majorityElement(self, nums):

sol = None

cnt = 0

for i in nums:

if cnt == 0:

sol = i

cnt += (1 if i == sol else -1)

return sol

**Question 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"]]

**Example 2:**

Input: strs = [""]

Output: [[""]]

**Example 3:**

Input: strs = ["a"]

Output: [["a"]]

**Constraints:**

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

**Sol:**

words = ["eat", "tea", "tan", "ate", "nat", "bat"]  
  
anagram\_groups = {}  
  
for word in words:  
 sorted\_word = ''.join(sorted(word))  
 if sorted\_word in anagram\_groups:  
 anagram\_groups[sorted\_word].append(word)  
 else:  
 anagram\_groups[sorted\_word] = [word]  
  
result = list(anagram\_groups.values())  
  
print(result)

**Question 5.** **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.

**Example 2:**

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

**Sol:**

class Solution:

def nthUglyNumber(self, n: int) -> int:

h = [1]

vis = {1}

ans = 1

for \_ in range(n):

ans = heappop(h)

for v in [2, 3, 5]:

nxt = ans \* v

if nxt not in vis:

vis.add(nxt)

heappush(h, nxt)

return ans

**Question 6**. **Top K Frequent Words**

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.

**Example 2:**

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]]

**Sol:**

class Solution:  
 def topKFrequent(self, words, k):  
 dico = defaultdict(int)  
 for word in words: dico[word] += 1  
  
 heap = []  
 for key, val in dico.items():  
 heapq.heappush(heap, (val, key))  
 if len(heap) > k: heapq.heappop(heap)  
  
 res = []  
 while heap:  
 res.append([heapq.heappop(heap)[1]])  
 return res[::-1]

**Question 7.** **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.

**Example 1:**

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

**Example 2:**

Input: nums = [1], k = 1

Output: [1]

**Constraints:**

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

**Sol:**

class Solution:

def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:

out = []

out.append(max(nums[:k]))

for i in range(len(nums)-k):

new = nums[i+k]

old = nums[i]

if new>out[-1]:

out.append(new)

else:

if out[-1]==old:

out.append(max(nums[i+1:k+i+1]))

else:

out.append(out[-1])

return(out)

**Question 8.** **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

**Example 1:**

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

Output: [1,2,3,4]

**Example 2:**

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

**Sol:**

class Solution(object):

def findClosestElements(self, arr, k, x):

left = 0

right = len(arr) - k

while left < right:

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

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

left = mid + 1

else:

right = mid

return arr[left : left + k]