Q1

def firstNotRepeatingCharacter(s):

myset = set()

for letter in s:

myset.add(letter)

if not in myset:

return letter

return '\_'

def firstNotRepeatingCharacter(s):

for c in s:

if s.index(c) == s.rindex(c):

return c

return '\_'

Q2

class Solution:

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

maxSum,minSum,curMin,curMax,sum=(nums[0],nums[0],0,0,0)

for val in nums:

curMax=curMax+val if curMax+val>=val else val

maxSum=max(curMax,maxSum)

curMin=curMin+val if curMin+val<=val else val

minSum=min(minSum,curMin)

sum+=val

return maxSum if sum==minSum else max(maxSum,sum-minSum)

Q3

class Solution:

def countStudents(self, students: List[int], sandwiches: List[int]) -> int:

while(len(students)!=0):

if(sandwiches[0] not in students):

return(len(students))

if(students[0]==sandwiches[0]):

students.pop(0)

sandwiches.pop(0)

else:

a=students.pop(0)

students.append(a)

return(len(students))

Q4

class RecentCounter:

def \_\_init\_\_(self):

self.s = []

def ping(self, t: int) -> int:

while self.s and t - self.s[0] > 3000:

self.s.pop(0)

self.s.append(t)

return len(self.s)

Q5

class Solution:

def findTheWinner(self, n: int, k: int) -> int:

def f(n,k):

from collections import deque

q = deque()

for i in range(1,n+1):q.append(i)

if k>n:k = k % n

while len(q)>1:

for i in range(1,k+1):

x = q.popleft()

if i !=k:q.append(x)

return q.pop()

return f(n,k)

Q6

class Solution:

def deckRevealedIncreasing(self, deck: List[int]) -> List[int]:

if not deck:

return []

deck.sort()

ans = []

while deck:

ans.append(deck.pop())

if not deck:

return ans[::-1]

ans.append(ans.pop(0))

return ans[::-1]

Q8

from collections import deque

class DataStream:

def \_\_init\_\_(self, value: int, k: int):

self.value = value

self.k = k

self.deque = deque()

self.count = 0

def consec(self, num: int) -> bool:

if len(self.deque) == self.k:

if self.deque[0] == self.value:

self.count -= 1

self.deque.popleft()

self.deque.append(num)

if num == self.value:

self.count += 1

return self.count == self.k