DAY 5 LAB

1.given an binary array AND integer k written true if all 1s are at last k places away from each other ore else return false

def find\_last\_index(arr,k):

if k<0 or k>=len(arr):

return "invalid index k"

if all(x==1 for x in arr):

return False

zero\_count = 0

for i in range (len(arr)):

if arr[i] == 0:

zero\_count += 1

else :

if zero\_count>2 :

return False

zero\_count = 0

return arr[k] == 1

arr = [0,0,1,0,0,1]

k = 5

result = find\_last\_index(arr,k)

print(result)

2.Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

Given an array of integers nums and an integer limit, return the size of the longest non-empty subarray such that the absolute difference between any two elements of this subarray is less than or equal to limit.

Example 1:

Input: nums = [8,2,4,7], limit = 4

Output: 2

Explanation: All subarrays are:

[8] with maximum absolute diff |8-8| = 0 <= 4.

[8,2] with maximum absolute diff |8-2| = 6 > 4.

[8,2,4] with maximum absolute diff |8-2| = 6 > 4.

[8,2,4,7] with maximum absolute diff |8-2| = 6 > 4.

[2] with maximum absolute diff |2-2| = 0 <= 4.

[2,4] with maximum absolute diff |2-4| = 2 <= 4.

[2,4,7] with maximum absolute diff |2-7| = 5 > 4.

[4] with maximum absolute diff |4-4| = 0 <= 4.

[4,7] with maximum absolute diff |4-7| = 3 <= 4.

[7] with maximum absolute diff |7-7| = 0 <= 4.

Therefore, the size of the longest subarray is 2.

Example 2:

Input: nums = [10,1,2,4,7,2], limit = 5

Output: 4

Explanation: The subarray [2,4,7,2] is the longest since the maximum absolute diff is |2-7| = 5 <= 5.

Example 3:

Input: nums = [4,2,2,2,4,4,2,2], limit = 0

Output: 3

Constraints:

● 1 <= nums.length <= 105

● 1 <= nums[i] <= 109

● 0 <= limit <= 109

def longestSubarray(nums, limit):

i = 0

max\_val = min\_val = nums[0]

for j in range(1, len(nums)):

max\_val = max(max\_val, nums[j])

min\_val = min(min\_val, nums[j])

if max\_val - min\_val > limit:

i += 1

max\_val = max(nums[i:j+1])

min\_val = min(nums[i:j+1])

return j - i + 1

print(longestSubarray([8,2,4,7], 4))

3. Find the Kth Smallest Sum of a Matrix With Sorted Rows

You are given an m x n matrix mat that has its rows sorted in non-decreasing order and an integer k.

You are allowed to choose exactly one element from each row to form an array.

Return the kth smallest array sum among all possible arrays.

Example 1:

Input: mat = [[1,3,11],[2,4,6]], k = 5

Output: 7

Explanation: Choosing one element from each row, the first k smallest sum are:

[1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.

Example 2:

Input: mat = [[1,3,11],[2,4,6]], k = 9

Output: 17

Example 3:

Input: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7

Output: 9

Explanation: Choosing one element from each row, the first k smallest sum are:

[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.

Constraints:

● m == mat.length

● n == mat.length[i]

● 1 <= m, n <= 40

● 1 <= mat[i][j] <= 5000

● 1 <= k <= min(200, nm)

● mat[i] is a non-decreasing array.

import heapq

def kthSmallest(mat, k):

m, n = len(mat), len(mat[0])

pq = [(sum(row[0] for row in mat), [0] \* m)]

for \_ in range(k):

curr\_sum, indices = heapq.heappop(pq)

for i, idx in enumerate(indices):

if idx + 1 < n:

new\_indices = indices[:]

new\_indices[i] += 1

new\_sum = curr\_sum - mat[i][idx] + mat[i][idx + 1]

heapq.heappush(pq, (new\_sum, new\_indices))

return curr\_sum

mat1 = [[1, 3, 11], [2, 4, 6]]

k1 = 5

print(kthSmallest(mat1, k1))

Minimum Time to Collect All Apples in a Tree

4.Given an undirected tree consisting of n vertices numbered from 0 to n-1, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree. Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at vertex 0 and coming back to this vertex.

The edges of the undirected tree are given in the array edges, where edges[i] = [ai, bi] means that exists an edge connecting the vertices ai and bi. Additionally, there is a boolean array hasApple, where hasApple[i] = true means that vertex i has an apple; otherwise, it does not have any apple.

Constraints:

● 1 <= n <= 105

● edges.length == n - 1

● edges[i].length == 2

● 0 <= ai < bi <= n - 1

● fromi < toi

● hasApple.length == n

def countTriplets(arr):

n = len(arr)

count = 0

for i in range(n):

xor = 0

for j in range(i, n):

xor ^= arr[j]

if xor == 0:

count += j - i

return count

arr1 = [1,1,1,1,1]

print(countTriplets(arr1))

5.Number of Ways of Cutting a Pizza

Given a rectangular pizza represented as a rows x cols matrix containing the following characters: 'A' (an apple) and '.' (empty cell) and given the integer k. You have to cut the pizza into k pieces using k-1 cuts.

For each cut you choose the direction: vertical or horizontal, then you choose a cut position at the cell boundary and cut the pizza into two pieces. If you cut the pizza vertically, give the left part of the pizza to a person. If you cut the pizza horizontally, give the upper part of the pizza to a person. Give the last piece of pizza to the last person.

Return the number of ways of cutting the pizza such that each piece contains at least one apple. Since the answer can be a huge number, return this modulo 10^9 + 7.

Constraints:

● 1 <= rows, cols <= 50

● rows == pizza.length

● cols == pizza[i].length

● 1 <= k <= 10

● pizza consists of characters 'A' and '.' only.

def minTime(n, edges, hasApple):

graph = [[] for \_ in range(n)]

for u, v in edges:

graph[u].append(v)

graph[v].append(u)

def dfs(node):

nonlocal res

visited[node] = True

for neighbor in graph[node]:

if not visited[neighbor]:

dfs(neighbor)

if hasApple[neighbor]:

res += 2

else:

res += 1

visited = [False] \* n

res = 0

dfs(0)

return res

n = 7

edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]]

hasApple = [False,False,True,False,True,True,False]

print(minTime(n,edges,hasApple))