1. Given binary array and an integer k , return true if all is at last k places array from each other wise flase

```
num=[1,0,0,0,1,0,0,1]
n=[]
p=[]
k = 2
for i in range(len(num)):
   if(num[i]==1):
      n.append(i)
for j in range(len(n)-1):
   r = abs(n[j]-n[j+1])-1
   p.append(r)
for g in range(len(p)):
   if(p[g]==k):
      f=1
      break
if(f==1):
   print("true")
else:
   print("false")
output
true
```

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

```
nums = [10,1,2,4,7,2]
limit = 5
arr = []
ans = 0
j = 0

for i in range(len(nums)):
    arr.append(nums[i])
    arr.sort()

    while arr[-1] - arr[0] > limit:
        arr.remove(nums[j])
        j += 1
```

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

output

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.

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

k = 5

ar = [0]

for i in mat:

ar = sorted(a + b for a in ar for b in i)[:k]

output = ar[-1]

print(output)

output

7

4. Count Triplets That Can Form Two Arrays of Equal XOR

Given an array of integers arr.

We want to select three indices i, j and k where (0 <= i < j <= k < arr.length).

Let's define a and b as follows:

- a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]
- b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]

Note that ^ denotes the bitwise-xor operation.

Return the number of triplets (i, j and k) Where a == b.

Example 1:

Input: arr = [2,3,1,6,7]

Output: 4

Explanation: The triplets are (0,1,2), (0,2,2), (2,3,4) and (2,4,4)

Example 2:

Input: arr = [1,1,1,1,1]

Output: 10

Constraints:

- 1 <= arr.length <= 300
- 1 <= arr[i] <= 108

```
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 = [2, 3, 1, 6, 7]
print(countTriplets(arr1))
```

output

4

5. Minimum Time to Collect All Apples in a Tree

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.

Example 1:

Input: n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,true,true,false]

Output: 8

Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Example 2:

Input: n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,false,true,false]

Output: 6

Explanation: The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Example 3:

Input: n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,false,false,false,false,false]

Output: 0

Constraints:

- 1 <= n <= 105
- edges.length == n - 1
- edges[i].length == 2
- 0 <= ai < bi <= n - 1
- fromi < toi
- hasApple.length == n

```
def minTimeToCollectApples(n, edges, hasApple):
    graph = [[] for _ in range(n)]
    for u, v in edges:
        graph[u].append(v)
        graph[v].append(u)

    def dfs(node, parent):
        time = 0
```

```
        for child in graph[node]:
            if child != parent:
                time += dfs(child, node)
        if (hasApple[node] or node != 0) and time == 0:
            return 2
        return time

    return max(0, dfs(0, -1) - 2)

print(minTimeToCollectApples(7, [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]],
[False,False,True,False,False,True,False]))
```

output

6

6. 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.

Example 1:

Input: pizza = ["A..","AAA","..."], k = 3

Output: 3

Explanation: The figure above shows the three ways to cut the pizza. Note that pieces must contain at least one apple.

Example 2:

Input: pizza = ["A..","AA.","..."], k = 3

Output: 1MOD = 10**9 + 7

```
def waysToCutPizza(pizza, k):
    rows, cols = len(pizza), len(pizza[0])
    dp = [[[0] * (k + 1) for _ in range(cols)] for _ in range(rows)]

    for i in range(rows):
        for j in range(cols):
            dp[i][j][1] = 1 if 'A' in pizza[i][j:] else 0
            for s in range(2, k + 1):
                dp[i][j][s] = 0

    for s in range(2, k + 1):
        for i in range(rows - 1, -1, -1):
            for j in range(cols - 1, -1, -1):
                for x in range(i + 1, rows):
                    if 'A' in pizza[i][j:]:
                        dp[i][j][s] += dp[x][j][s - 1]
                        dp[i][j][s] %= MOD
                for y in range(j + 1, cols):
                    if 'A' in [pizza[r][j] for r in range(i, rows)]:
                        dp[i][j][s] += dp[i][y][s - 1]
                        dp[i][j][s] %= MOD

    return dp[0][0][k]
print(waysToCutPizza(["A..","AA.","..."], 3))
```

output

1