1.

In [42]:
```
def k_len_apart(nums, k):
    prev_index = -k - 1
    for i, num in enumerate(nums):
        if num == 1:
            if i- prev_index<=k:
                return False
            prev_index = i
    return True
nums = [1,0,0,0,1,0,0,1]
k>=2
print(k_len_apart(nums, k))
```

```
True
```

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

In [58]:
```
from collections import deque
def long_subarray(nums, limit):
    max_deque = deque()
    min_deque = deque()
    left = 0
    max_len = 0
    for right in range(len(nums)):
        while max_deque and nums[right] > max_deque[-1]:
            max_deque.pop()
        max_deque.append(nums[right])
        while min_deque and nums[right] < min_deque[-1]:
            min_deque.pop()
        min_deque.append(nums[right])
        while max_deque[0] - min_deque[0] > limit:
            if nums[left] == max_deque[0]:
                max_deque.popleft()
            if nums[left] == min_deque[0]:
                min_deque.popleft()
            left += 1
        max_len = max(max_len, right - left + 1)
    return max_len
nums = [8, 2, 4, 7]
limit = 4
print(long_subarray(nums, limit))
```

```
2
```

3. kth smallest

In [59]:
```
import heapq
def kthSmallest(mat, k):
    m, n = len(mat), len(mat[0])
    heap = [(sum(row[0] for row in mat), [0] * m)]
    for _ in range(k):
        s, idx = heapq.heappop(heap)
        for i, j in enumerate(idx):
            if j + 1 < n:
                heapq.heappush(heap, (s-mat[i][j]+ mat[i][j + 1],idx[:i]+ [j
    return sum(mat[i][j] for i, j in enumerate(idx))
mat1 = [[1, 3, 11], [2, 4, 6]]
k1 = 5
print(kthSmallest(mat1, k1))
```

```
7
```

count th elements

In [65]:
```
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))
arr2 = [1, 1, 1, 1, 1]
print(counttriplets(arr2))
```

```
4
10
```

In [66]:
```
from collections import defaultdict
from typing import List
class Solution:
    def minTime(self, n: int, edges: List[List[int]], hasApple: List[bool])
        graph = defaultdict(list)
        for u, v in edges:
            graph[u].append(v)
            graph[v].append(u)
        def dfs(node: int) -> int:
            total_time = 0
            for neighbor in graph[node]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    total_time += dfs(neighbor)
            if total_time > 0 or hasApple[node]:
                if node != 0:
                    return total_time + 2
            return total_time
        visited = set()
        visited.add(0)
        return max(dfs(0), 0)
solution = Solution()
print(solution.minTime(7, [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], [False,Fals
```

```
8
```

In [ ]:
```

```