差分数组：高效处理区间加减操作

已知数组a，多个操作 [l, r, val]（对区间[l, r]加上val）

差分数组 ⇒ O(1)时间处理每个操作

```
diff = [0] * (n+1)
for l, r, val in operations:          # 差分操作，两次修改
    diff[l] += val
    diff[r+1] -= val                  (若r+1 < len(diff))
a[0] = diff[0]
for i in range(1, n):
    a[i] = a[i-1] + diff[i]           # 用diff由前缀和还原a
```

奥德

3362. 零数组变换 III

已解答

中等 | 相关标签 | 相关企业 | 提示 | A文

给你一个长度为 `n` 的整数数组 `nums` 和一个二维数组 `queries` ，其中 `queries[i] = [l_i, r_i]` 。

每一个 `queries[i]` 表示对于 `nums` 的以下操作：

- 将 `nums` 中下标在范围 `[l_i, r_i]` 之间的每一个元素 **最多** 减少 1 。
- 坐标范围内每一个元素减少的值相互 **独立** 。

零数组 指的是一个数组里所有元素都等于 0 。

请你返回 **最多** 可以从 `queries` 中删除多少个元素，使得 `queries` 中剩下的元素仍然能将 `nums` 变为一个 **零数组** 。如果无法将 `nums` 变为一个 **零数组** ，返回 -1 。

```python
def maxRemoval(self, nums: List[int], queries: List[List[int]]) -> int:
    # 先对queries排序 确保left小的在前面 | right不用管 因为大根堆会处理
    queries.sort(key = lambda x:x[0])
    heap = []
    # 记录遍历处理的queries的下标
    j=0
    # 当前通过queries累加后的值
    operations=0
    # 差分数组 处理的时候只关注结尾的 del[right+1]-=1 而不用记录+=1
    # 因为operations已经记录了累加的情况
    deltaArray = [0]*(len(nums)+1)
    for i,num in enumerate(nums):
        # 先进行差分累加
        operations+=deltaArray[i]
        # 如果当前query的left等于现在的i 就进堆
        # 那假设现在i==1 query=[0,2]怎么办,这种不直接等于的情况都会在i==0的时候被处理了 不会有漏掉的情况
        while j<len(queries) and queries[j][0]==i:
            # 注意负号 因为python默认建立的是小根堆
            heapq.heappush(heap,-queries[j][1])
            j+=1
        # 当operations还是不够大 就要从堆取（前提是堆里还有数）
        while operations<num and len(heap)>0 and -heap[0]>=i:
            # 取出来 在差分[right+1]的位置-=1 差分常规操作
            deltaArray[-heapq.heappop(heap)+1]-=1
            # 只有从堆取出来的时候才加1
            operations+=1
        # 如果都取完了 还是不够 那queries再往后也没用了 因为已经排序
        # left的值都将小于当前i 所以可以终止
        if operations<num:
            return -1
    # 第一个while确保了我们将所有query都放进堆了
    # 所以没取出来 也没有返回-1 说明就是剩下的没用上的 故直接返回堆的大小
    return len(heap)
```