3355. Zero Array Transformation I

You are given an integer array $nums$ of length n and a 2D array `queries`, where $queries[i]=[l_i,r_i]$.

For each $queries[i]$:

- Select a of indices within the range $[l_i,r_i]$ in $nums$.
- Decrement the values at the selected indices by 1.

A **Zero Array** is an array where all elements are equal to 0.

Return $true$ if it is *possible* to transform $nums$ into a **Zero Array** after processing all the queries sequentially, otherwise return $false$.

Example 1:

Input: $nums=[1,0,1]$, $queries=[[0,2]]$

Output: $true$

Explanation:

- **For i = 0:**
  - Select the subset of indices as $[0,2]$ and decrement the values at these indices by 1
  - The array will become $[0,0,0]$, which is a Zero Array.

Example 2:

Input: $nums=[4,3,2,1]$, $queries=[[1,3],[0,2]]$

Output: $false$

Explanation:

- **For i = 0:**
  - Select the subset of indices as $[1,2,3]$ and decrement the values at these indices by 1.
  - The array will become $[4,2,1,0]$.
- **For i = 1:**
  - Select the subset of indices as $[0,1,2]$ and decrement the values at these indices by 1.
  - The array will become $[3,1,0,0]$, which is not a Zero Array.

Constraints:

- $1 \leq nums.length \leq 10^5$
- $0 \leq nums[i] \leq 10^5$
- $1 \leq queries.length \leq 10^5$
- $queries[i].length == 2$
- $0 \leq l_i \leq r_i < nums.length$

3355. Zero Array Transformation I

```
/*
   Naive: Naively process the queries on the array
   Time complexity: O(n.m) (TLE)
   Space complexity: O(1)
*/
class Solution {
   public:
      bool isZeroArray(vector<int>& nums, vector<vector<int>>& queries) {
         //int n=nums.size();
         //int m=queries.size();

         // For each query
         for(auto& query: queries){
            int l=query[0];
            int r=query[1];
            int val=query[2];

            // Run over the array in range [l,r]
            for(int i=l;i<=r;++i){
               // if the current element is already equal to zero, pass to next
               if(nums[i]==0) continue;

               // Otherwise, decrement 1 from the current element
               nums[i]--;
            }
         }

         // After processing all queries, check il all elements in the given array
         // are transformed to zero or not
         for(auto& e: nums){
            // If at least one element is not equal to zero, return false
            if(e!=0) return false;
         }

         // If all elements are equal to zero
         return true;
      }
};
```

3355. Zero Array Transformation I

Difference array technique

It is a simple technique that can give you an end array after performing range queries of the form $update(l, r, x)$.

NOTE: $update(l, r, x)$ means add x to all the elements in an array within the range l to r .
$A[l] += x, A[l + 1] += x, A[l + 2] += x, \ldots A[r] += x$.

Difference array stores all the differences between the two elements

$$diff[i] = A[i] - A[i-1]$$

<u>Lemma:</u> The difference array work on a simple technique that when you add something to a range in an array the difference between two elements remains the same except for the boundary of the range.

<u>Proof</u>

$$A = \{a_0, a_1, a_2, \ldots, a_{n-1}\}$$
$$diff[i] = \{0, a_1 - a_0, a_2 - a_1, a_3 - a_2, \ldots, a_{n-1} - a_{n-2}\}$$

Performing $update(l, r, x)$ on A :

$$A = \{a_0, a_1, a_2, \ldots, a_{l-1}, a_l + x, a_{l+1} + x, a_{l+2} + x, \ldots, a_{r-1} + x, a_r + x, a_{r+1}, \ldots, a_{n-1}\}$$

so, difference array will be:

$$diff[i] = \{0, a_1 - a_0, a_2 - a_1, \ldots,$$
$$a_l + x - a_{l-1}, a_{l+1} + x - a_l - x, a_{l+2} + x - a_{l+1} - x, \ldots, a_r + x - a_{r-1} - x, a_{r+1} - a_r - x,$$
$$a_{r+2} - a_{r+1}, \ldots, a_{n-1} - a_{n-2}\}$$
$$diff[i] = \{0, a_1 - a_0, a_2 - a_1, \ldots,$$
$$a_l - a_{l-1} + x, a_{l+1} - a_l, a_{l+2} - a_{l+1}, \ldots, a_r - a_{r-1}, a_{r+1} - a_r - x,$$
$$a_{r+2} - a_{r+1}, \ldots, a_{n-1} - a_{n-2}\}$$

We have proved that updating A from l to r by x , is the same by updating the difference array at l by $+x$ and at $r+1$ by $-x$

Restoring final array after updates

$$diff[i]=A[i]-A[i-1] <=> A[i]=diff[i]+A[i-1]$$

$$\begin{cases} A[0]=a_0 \\ A[i]=diff[i]+A[i-1] \end{cases}$$

$$A=\{a_0,a_1,a_2,\dots,a_{n-1}\}$$

$$diff[i]=\{0,a_1-a_0,a_2-a_1,\dots,$$
$$a_l-a_{l-1}+x,a_{l+1}-a_l,a_{l+2}-a_{l+1},\dots,a_r-a_{r-1},a_{r+1}-a_r-x,$$
$$a_{r+2}-a_{r+1},\dots,a_{n-1}-a_{n-2}\}$$

$$A[0]=a_0+0$$

$$A[1]=diff[1]+A[0]=(a_1-a_0)+a_0=a_1$$

$$A[2]=diff[2]+A[1]=(a_2-a_1)+a_1=a_2$$

$$A[l]=diff[l]+A[l-1]=(a_l-a_{l-1}+x)+a_{l-1}=a_l+x$$

$$A[l+1]=diff[l+1]+A[l]=(a_{l+1}-a_l)+a_l+x=a_{l+1}+x$$

....

$$A[r]=diff[r]+A[r-1]=(a_r-a_{r-1})+a_{r-1}+x=a_r+x$$

$$A[r+1]=diff[r+1]+A[r]=(a_{r+1}-a_r-x)+a_r+x=a_{r+1}$$

3355. Zero Array Transformation I

```
/*
   Difference array
   Time complexity: O(n+m+n)=O(n+m)
   Space complexity: O(n)
*/
class Solution {
   public:
      bool isZeroArray(std::vector<int>& nums, std::vector<std::vector<int>>& queries) {
         int n=nums.size();
         //int m=queries.size();

         // Create the difference of size (n+1)
         std::vector<int> diff(n+1,0);

         // For each query
         for(auto& query: queries){
            int l=query[0];
            int r=query[1];
            int val=query[2];

            // Update the difference array
            diff[l]++;
            diff[r+1]--;
         }

         // After processing all queries, check il all elements in the given array
         // are transformed to zero or not
         int pre=0;
         for(int i=0;i<n;++i){
            // pre=sum of all values in range[0,i] to decrement from nums[i]
            pre+=diff[i];

            // If at least one element is not equal to zero, return false
            if(nums[i]-pre>0) return false;
         }

         // If all elements are equal to zero
         return true;
      }
};
```

3356. Zero Array Transformation II

You are given an integer array $nums$ of length `n` and a 2D array $querie$ `s` where $queries[i]=[l_i,r_i,val_i]$.

Each $queries[i]$ represents the following action on $nums$:

- Decrement the value at each index in the range $[l_i,r_i]$ in $nums$ by **at most** val_i.
- The amount by which each value is decremented can be chosen **independently** for each index.

A **Zero Array** is an array with all its elements equal to 0.

Return the **minimum** possible **non-negative** value of k, such that after processing the first k queries in **sequence**, $nums$ becomes a **Zero Array**. If no such k exists, return -1.

Example 1:

Input: $nums=[2,0,2]$, $queries=[[0,2,1],[0,2,1],[1,1,3]]$

Output: 2

Explanation:

- **For i = 0 (l = 0, r = 2, val = 1):**
  - Decrement values at indices `[0, 1, 2]` by `[1, 0, 1]` respectively.
  - The array will become `[1, 0, 1]`.
- **For i = 1 (l = 0, r = 2, val = 1):**
  - Decrement values at indices `[0, 1, 2]` by `[1, 0, 1]` respectively.
  - The array will become `[0, 0, 0]`, which is a Zero Array. Therefore, the minimum value of `k` is 2.

Example 2:

Input: $nums=[4,3,2,1]$, $queries=[[1,3,2],[0,2,1]]$

Output: -1

Explanation:

- **For i = 0 (l = 1, r = 3, val = 2):**
  - Decrement values at indices `[1, 2, 3]` by `[2, 2, 1]` respectively.
  - The array will become `[4, 1, 0, 0]`.
- **For i = 1 (l = 0, r = 2, val = 1):**
  - Decrement values at indices `[0, 1, 2]` by `[1, 1, 0]` respectively.
  - The array will become `[3, 0, 0, 0]`, which is not a Zero Array.

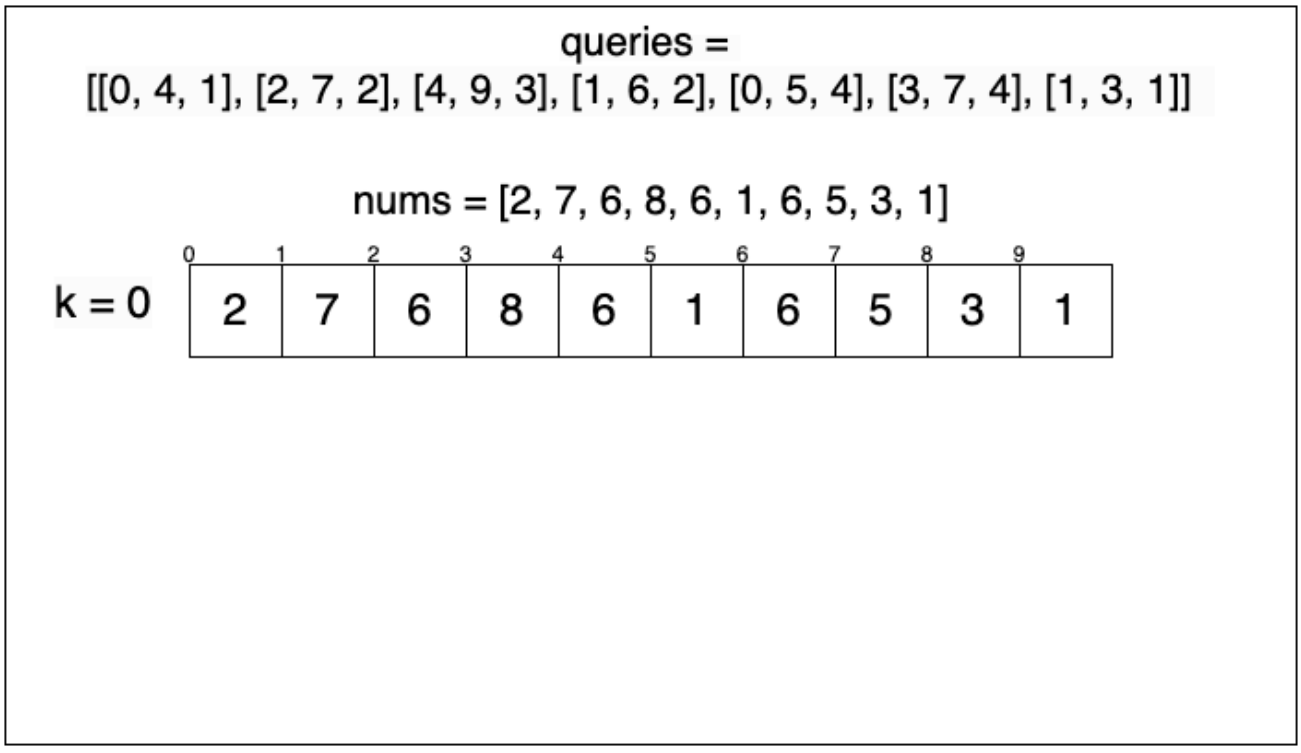
Constraints:

- $1 \leq nums.length \leq 10^5$
- $0 \leq nums[i] \leq 5 * 10^5$
- $1 \leq queries.length \leq 10^5$
- $queries[i].length == 3$
- $0 \leq l_i \leq r_i < nums.length$
- $1 \leq val_i \leq 5$

Overview

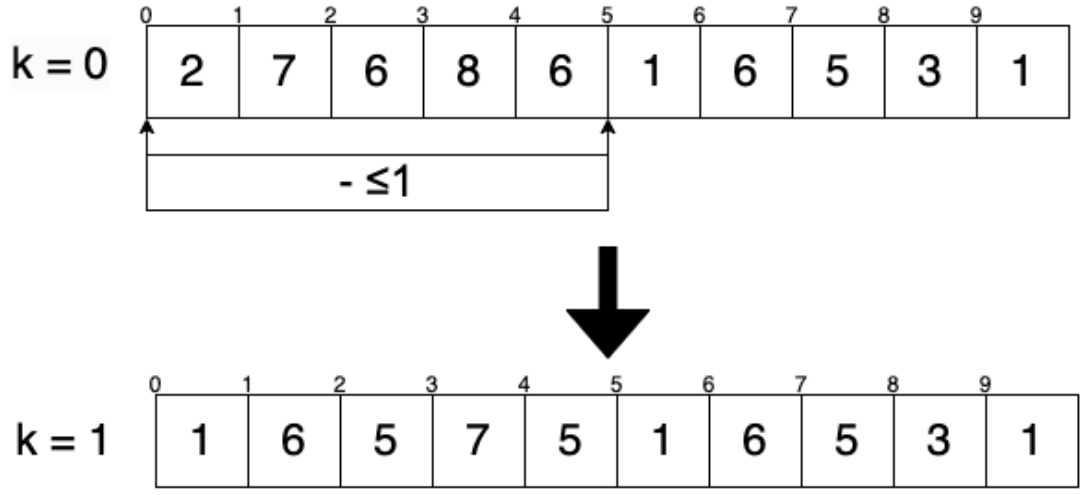
We are given an integer array `nums` of length `n`, and a list of queries that are each in the form `[left, right, val]`. For a given range `[left, right]`, we can decrease each element in that range by at most `val`. Our task is to determine the earliest query that allows us to turn `nums` into an array of all zeroes. If it's not possible, we return `-1`.

We can look at an example of the queries being processed:



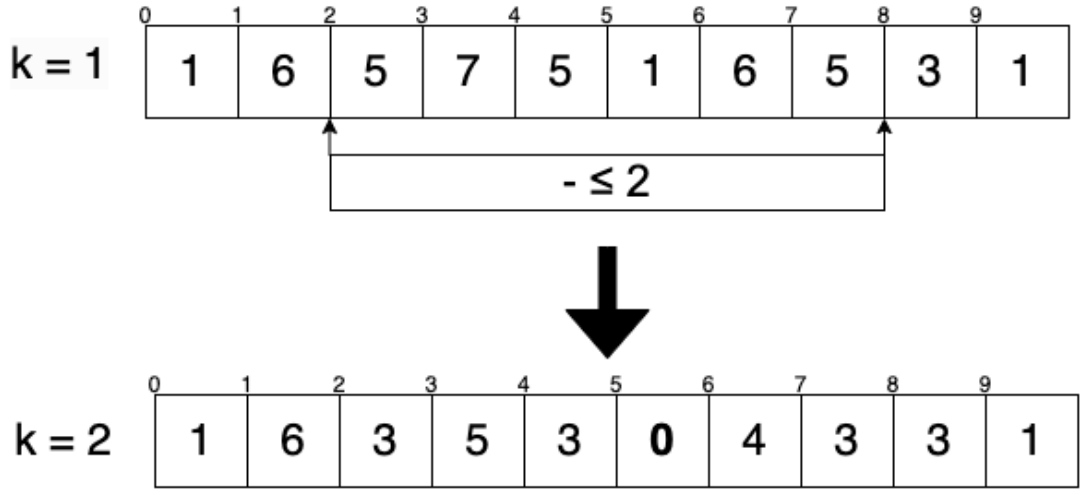
queries =
[**[0, 4, 1]**, [2, 7, 2], [4, 9, 3], [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]

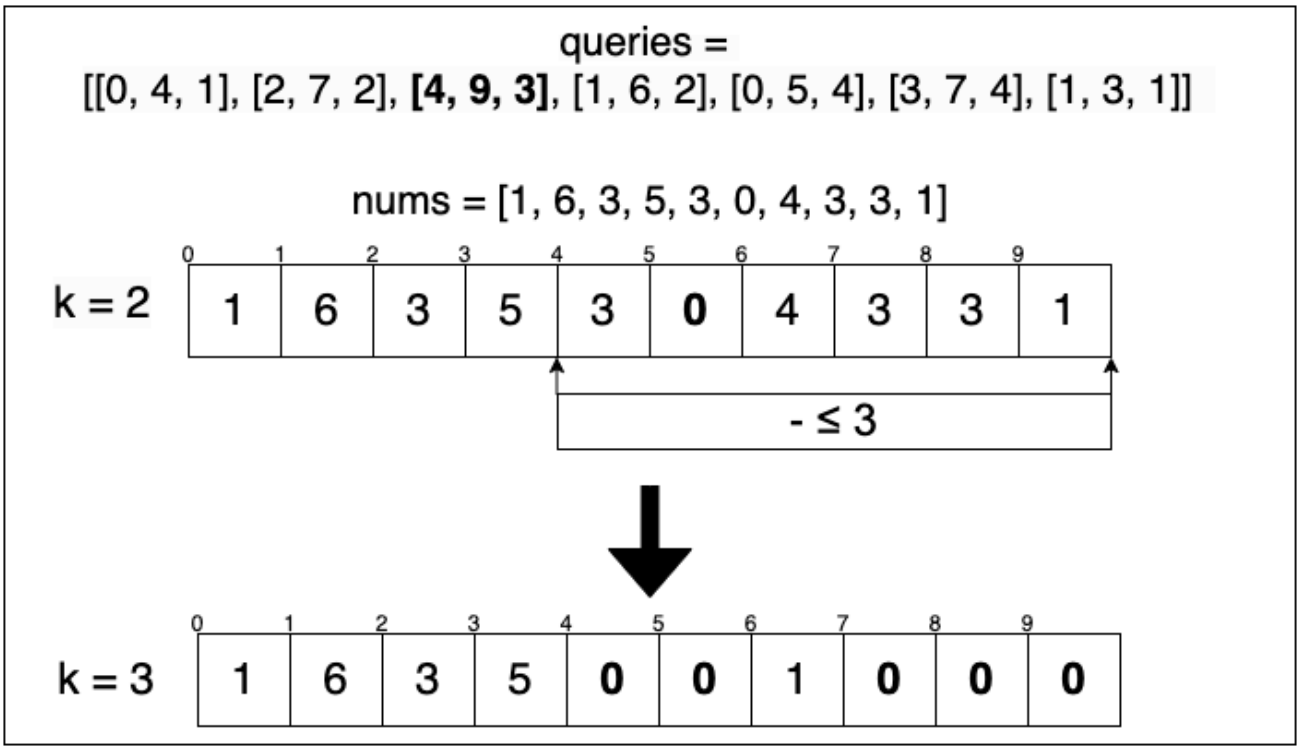
nums = [2, 7, 6, 8, 6, 1, 6, 5, 3, 1]

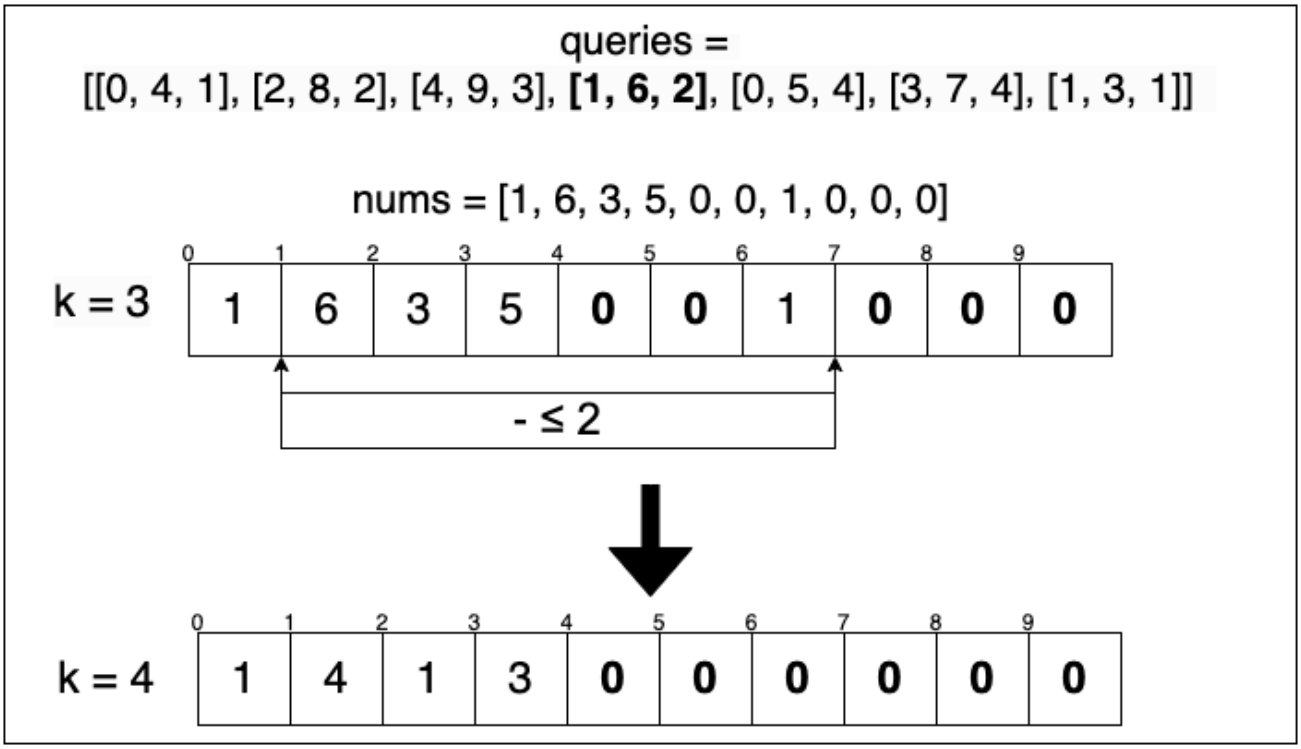


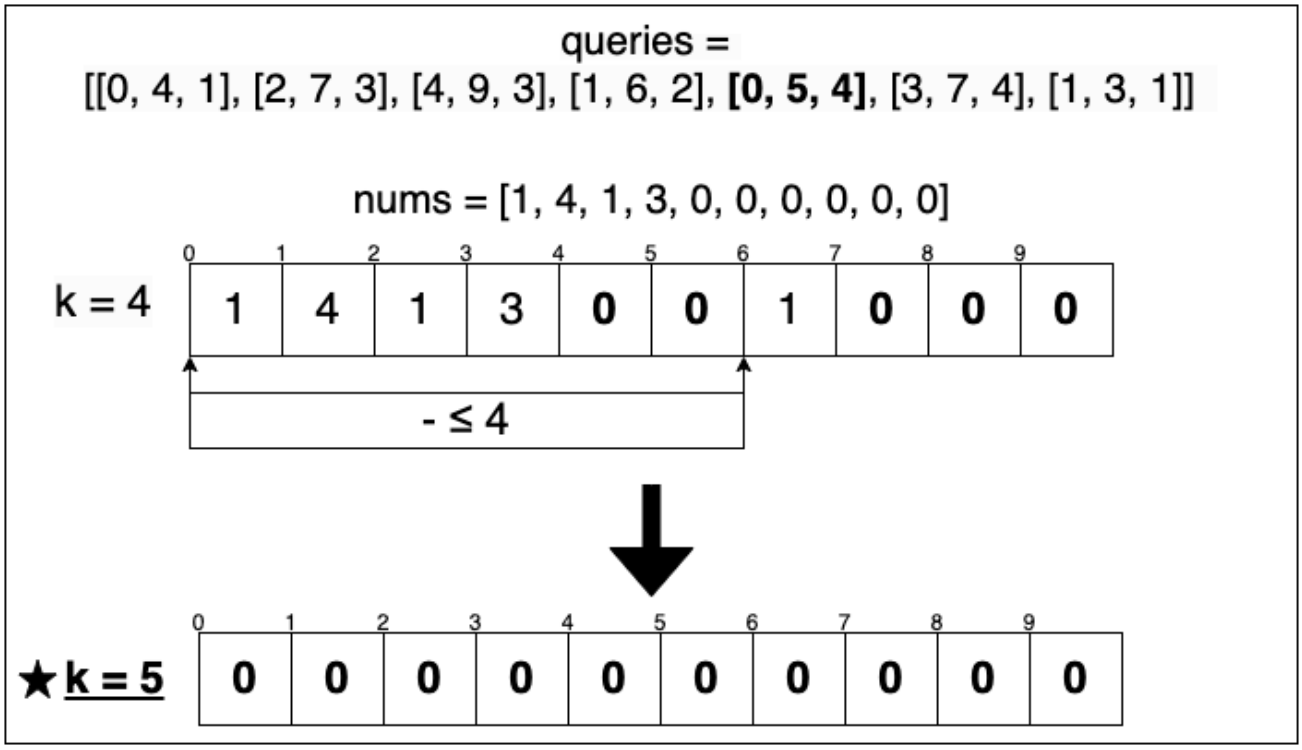
queries =
[[0, 4, 1], **[2, 7, 2]**, [4, 9, 3], [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]

nums = [1, 6, 5, 7, 5, 1, 6, 5, 3, 1]




queries =
[[0, 4, 1], [2, 7, 2], [4, 9, 3], [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]
nums = [1, 6, 3, 5, 3, 0, 4, 3, 3, 1]
k = 2
0 1 2 3 4 5 6 7 8 9
1 6 3 5 3 0 4 3 3 1
- ≤ 3
k = 3
0 1 2 3 4 5 6 7 8 9
1 6 3 5 0 0 1 0 0 0


queries =
[[0, 4, 1], [2, 8, 2], [4, 9, 3], [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]
nums = [1, 6, 3, 5, 0, 0, 1, 0, 0, 0]
k = 3
0 1 2 3 4 5 6 7 8 9
1 6 3 5 0 0 1 0 0 0
- ≤ 2
k = 4
0 1 2 3 4 5 6 7 8 9
1 4 1 3 0 0 0 0 0 0



From this example, we can see that there are two main operations that will occur:

1. Iterating through each element in `queries`.
2. Applying the range and value of each query to `nums`.

3356. Zero Array Transformation II

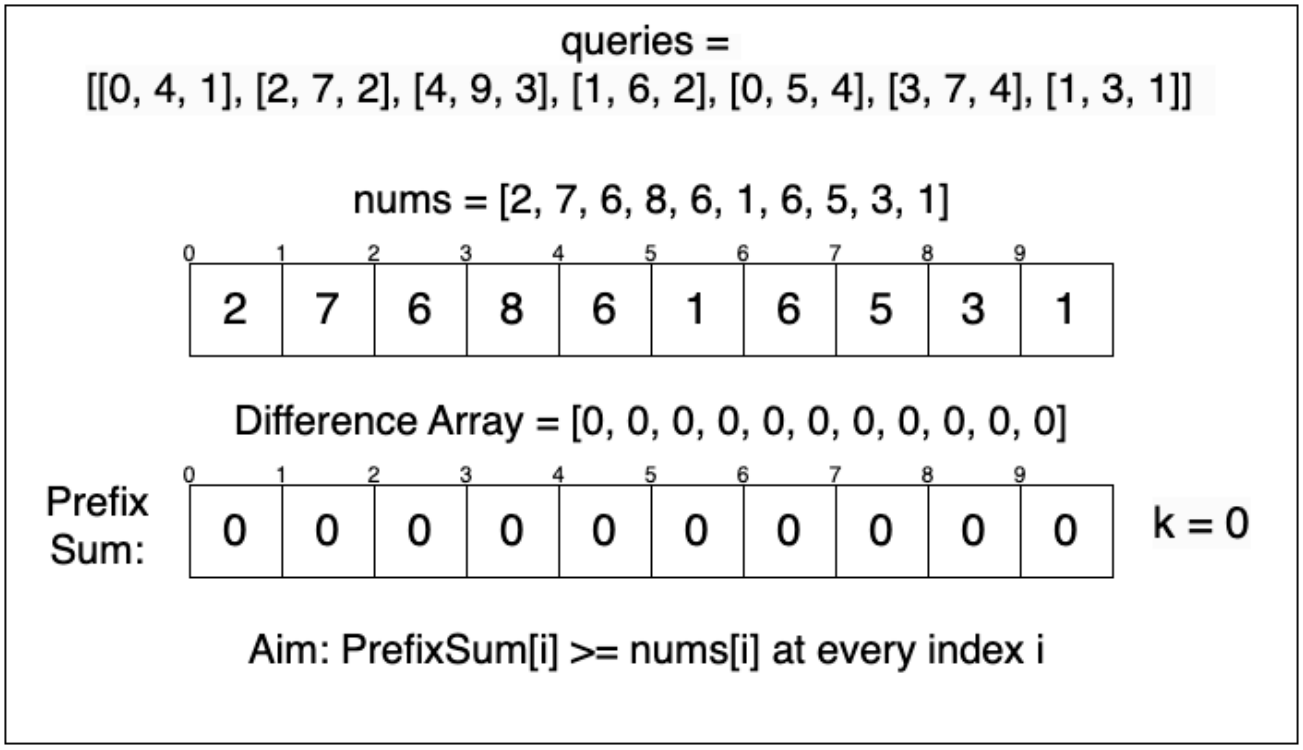
```
/*
   Brute force: Naively process the queries
   Time complexity: O(n+2.m.n) (TLE)
   Space complexity: O(1)
*/
class Solution {
public:
   int minZeroArray(std::vector<int>& nums, std::vector<std::vector<int>>& queries){
      int n=nums.size();
      int m=queries.size();

      // Check if an array is a zero array or not
      auto is_zero_array=[&]()->bool{
         for(auto& e: nums){
            if(e!=0) return false;
         }
         return true;
      };

      if(is_zero_array()) return 0;

      int k=0; // Minimum number of queries to process to transform all elements of the array  to zero
      // For each query
      for(auto& query: queries){
         int l=query[0];
         int r=query[1];
         int max_val=query[2];

         // Perform the current query on the given array
         for(int i=l;i<=r;++i) nums[i]=std::max(nums[i]-val,0);

         // Query is performed, add it to the answer
         k++;

         // After processing the current query,if the array become a zero array,
         //return the number of queries k and stop
         if(is_zero_array()) return k;
      }

      // If all queries are performed, but the array is not transformed to zeros
      return -1;
   }
};
```

3356. Zero Array Transformation II

Difference array

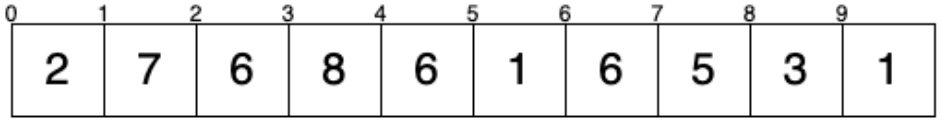
To optimize the above solution, we need a more efficient way to apply queries to $nums$. Instead of modifying each element individually, we can take advantage of a **difference array**. This technique allows us to apply a range update in constant time. The key idea is to store the changes at the boundaries of the range rather than updating every element inside it. For a query $[l, r, val]$, we add val at index l , and subtract val at index $r+1$. ***When we later compute the prefix sum of this difference array, it reconstructs the actual values efficiently.*** This way, instead of updating $nums$ repeatedly, we can process all queries in an optimized manner and then traverse $nums$ just once to check if all elements have become zero.

Let's look at how the difference array can be applied to this problem:

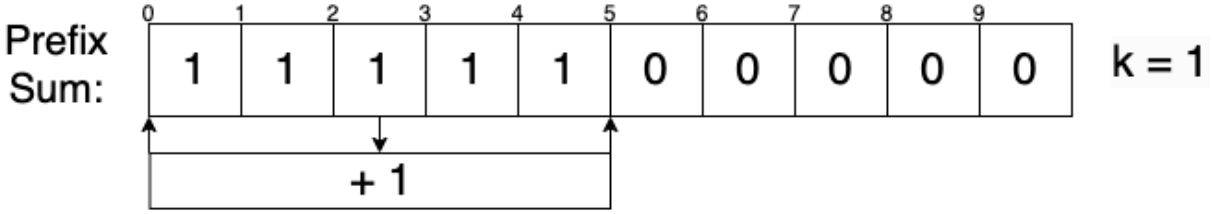


queries =
[[**0, 4, 1**], [2, 7, 3], [4, 9, 3], [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]

nums = [2, 7, 6, 8, 6, 1, 6, 5, 3, 1]

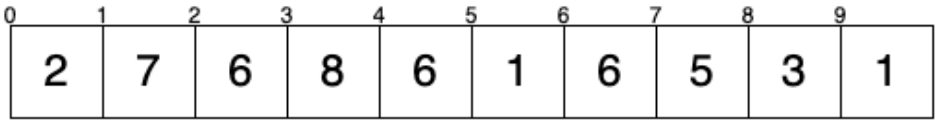


Difference Array = [1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0]

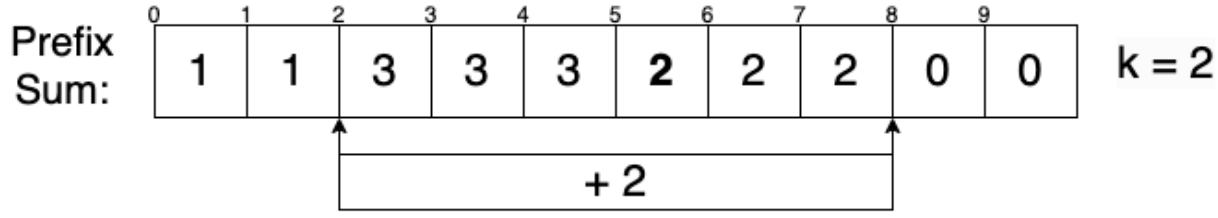


queries =
[[0, 4, 1], **[2, 7, 2]**, [4, 9, 3], [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]

nums = [2, 7, 6, 8, 6, 1, 6, 5, 3, 1]

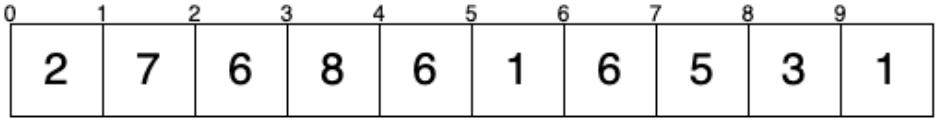


Difference Array = [1, 0, 2, 0, 0, -1, 0, 0, -2, 0, 0]



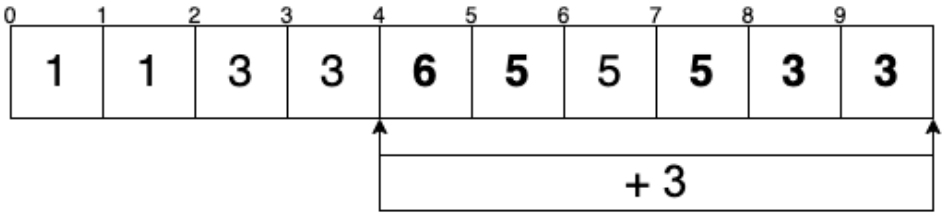
queries =
[[0, 4, 1], [2, 7, 3], **[4, 9, 3]**, [1, 6, 2], [0, 5, 4], [3, 7, 4], [1, 3, 1]]

nums = [2, 7, 6, 8, 6, 1, 6, 5, 3, 1]



Difference Array = [1, 0, 2, 0, 3, -1, 0, 0, -2, 0, -3]

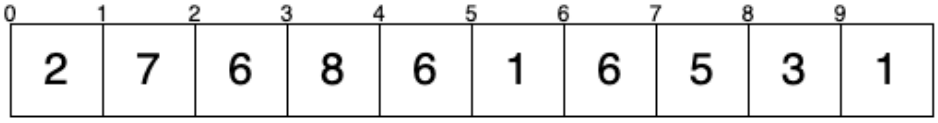
Prefix Sum:



k = 3

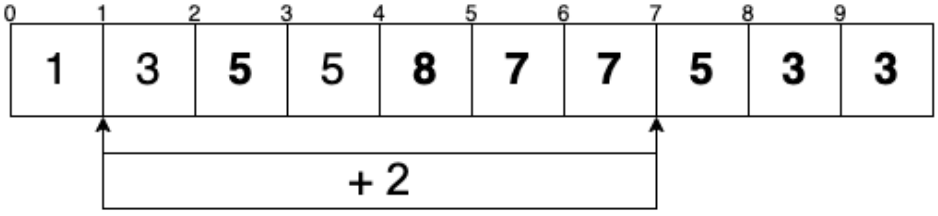
queries =
[[0, 4, 1], [2, 7, 3], [4, 9, 3], **[1, 6, 2]**, [0, 5, 4], [3, 7, 4], [1, 3, 1]]

nums = [2, 7, 6, 8, 6, 1, 6, 5, 3, 1]



Difference Array = [1, 2, 2, 0, 3, -1, 0, -2, -2, 0,-3]

Prefix Sum:

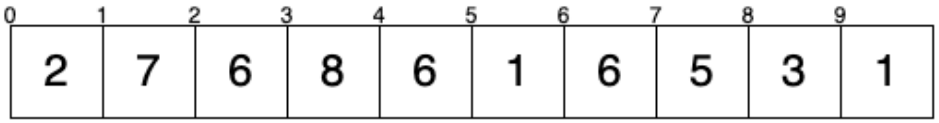


k = 4

queries =

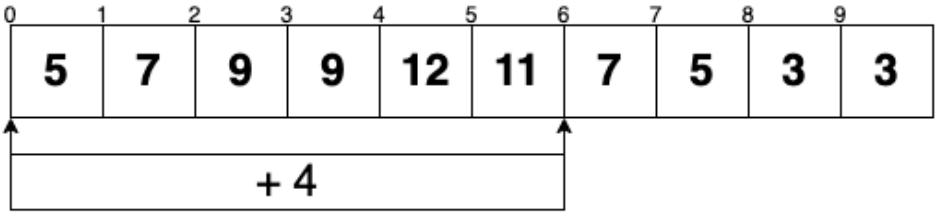
[[0, 4, 1], [2, 7, 3], [4, 9, 3], [1, 6, 2], **[0, 5, 4]**, [3, 7, 4], [1, 3, 1]]

nums = [2, 7, 6, 8, 6, 1, 6, 5, 3, 1]



Difference Array = [5, 2, 2, 0, 3, -1, -4, -2, -2, 0,-3]

Prefix Sum:



★ **k = 5**

3356. Zero Array Transformation II

```
/*
   Binary search on difference arrays
   Time complexity: O(m+log m . (m+n))
   Space complexity: O(n)
*/
class Solution {
public:
   int minZeroArray(std::vector<int>& nums, std::vector<std::vector<int>>& queries){
      int n=nums.size();
      int m=queries.size();

      // Check if it is possible to obtain a zero array after processing k queries
      auto is_possible=[&](int k)->bool{
         // Create difference array
         std::vector<int> diff(n+1,0);

         for(int i=0;i<k;++i){
            int l=queries[i][0];
            int r=queries[i][1];
            int val=queries[i][2];

            // Update difference array
            diff[l]+=val;
            diff[r+1]-=val;
         }

         // After processing all queries, check il all elements in the given array
         // are transformed to zero or not
         int pre=0;
         for(int i=0;i<n;++i){
            // pre=sum of all values in range[0,i] to decrement from nums[i]
            pre+=diff[i];

            // If at least one element is not equal to zero, return false
            if(nums[i]-pre>0) return false;
         }

         // If all elements are equal to zero
         return true;
      };
```

```
        // If is not possible to transform the array to a zero array after performing all the queries
        if(!is_possible(m)) return -1;

        // If it is possible to transform the array to a zero array after performing all the queries
        // determine the minimum number of queries to achieve that:

        // Do a binary search on the queries
        // return the first met where is possible to transform the array to a zero array
        int lo=0,hi=m-1;
        while(lo<=hi){
            int k=(lo+hi)>>1;
            if(is_possible(k)) hi=k-1;
            else lo=k+1;
        }
        return lo;
    }
};
```

3355. Zero Array Transformation I

```
/*
   Line sweep + Difference array: apply only the necessary queries at the right moment.
   Time complexity: O(n+m+n)=O(n+m)
   Space complexity: O(n)
*/
class Solution {
   public:
      bool isZeroArray(std::vector<int>& nums, std::vector<std::vector<int>>& queries) {
         int n=nums.size();
         int m=queries.size();
         // Create the difference of size (n+1)
         std::vector<int> diff(n+1,0);
         int pre=0; // Prefix sum of difference array
         int query=0; // Number of the current query
         // For each element in the given array
         for(int i=0;i<n;++i){
            // While the current element in not zero
            while(nums[i]-(pre+diff[i])>0){
               query++; // Perform the current query
               // If we have performed all queries
               if(query>m) return false;
               // Get range
               int l=queries[query-1][0];
               int r=queries[query-1][1];
               // If the index of the current element is the range
               if(i<=r){
                  diff[std::max(i,l)]++;  // Update the start of the range
                  diff[r+1]--;
               }
            }
            // Update the prefix sum at current index
            pre+=diff[i];
         }
         // The original array transformed to a zero array
         return true;
      }
};
```

3356. Zero Array Transformation II

```
/*
   Line sweep technique: apply only the necessary queries at the right moment.
   Time complexity: O(n+m)
   Space complexity: O(n)
*/
class Solution {
public:
   int minZeroArray(std::vector<int>& nums, std::vector<std::vector<int>>& queries){
      int n=nums.size();
      int m=queries.size();

      std::vector<int> diff(n+1,0);

      int pre=0,k=0;

      for(int i=0;i<n;++i){
         while(nums[i]-(pre+diff[i])>0){
            k++;

            if(k>m) return -1;

            int l=queries[k-1][0];
            int r=queries[k-1][1];
            int val=queries[k-1][2];

            if(i<=r){
               diff[std::max(i,l)]+=val;
               diff[r+1]-=val;
            }
         }
         pre+=diff[i];
      }

      return k;
   }
};
```