3097. Shortest Subarray With OR at Least K II

You are given an array `nums` of **non-negative** integers and an integer `k`.

An array is called **special** if the bitwise `OR` of all of its elements is **at least** `k`.

Return *the length of the* ***shortest special non-empty*** subarray *of* `nums`, *or return* `-1` *if no special subarray exists*.

Example 1:

Input: nums = [1,2,3], k = 2

Output: 1

Explanation:

The subarray `[3]` has `OR` value of `3`. Hence, we return `1`.

Example 2:

Input: nums = [2,1,8], k = 10

Output: 3

Explanation:

The subarray `[2,1,8]` has `OR` value of `11`. Hence, we return `3`.

Example 3:

Input: nums = [1,2], k = 0

Output: 1

Explanation:

The subarray `[1]` has `OR` value of `1`. Hence, we return `1`.

Constraints:

- `1 <= nums.length <= 2 * 10^5`
- `0 <= nums[i] <= 10^9`
- `0 <= k <= 10^9`

3097. Shortest Subarray With OR at Least K II

```
/*
   Naive approach-TLE
   Time complexity: O(n^3)
   Space complexity: O(1)
*/
typedef std::vector<int> vi;

class Solution {
   public:
      int minimumSubarrayLength(vi& nums, int k) {
         int n=nums.size();
         int ans=INT_MAX;
         for(int i=0;i<n;++i){
            for(int j=i;j<n;++j){
               int s=0;
               for(int k=i;k<=j;++k) s|=nums[k];
               if(s>=k) ans=std::min(ans,j-i+1);
            }
         }
         return ans!=INT_MAX?ans:-1;
      }
};
```

3097. Shortest Subarray With OR at Least K II

```
/*
   Optimization - TLE
   Time complexity: O(n^2)
   Space complexity: O(1)
*/
typedef std::vector<int> vi;

class Solution {
   public:
      int minimumSubarrayLength(vi& nums, int k) {
         int n=nums.size();
         int ans=INT_MAX;
         for(int i=0;i<n;++i){
            for(int j=i;j<n;++j){
               int s=0;
               for(int k=i;k<=j;++k) s|=nums[k];
               if(s>=k) ans=std::min(ans,j-i+1);
            }
         }
         return ans!=INT_MAX?ans:-1;
      }
};
```

3097. Shortest Subarray With OR at Least K II

```
/*
   Sliding window - AC
   Time complexity: O(n*32)=O(n)
   Space complexity: O(32)=O(1)
*/
typedef std::vector<int> vi;

class Solution {
   public:
      int minimumSubarrayLength(vi& nums, int k) {
         int n=nums.size();

         // Store the contibution of each bit of each value
         // in the prefix OR result
         vi prefix_bit(32,0);

         // If the  bit at position i of the value `val` is set
         // If `val` contibutes in the prefix OR result: we increment the conribution of that bit by 1
         // If `val` does not contibute in the prefix OR result: we subtract 1 from conribution of that bit
         auto contribute=[&](int val,bool is_contibuted)->void{
            for(int i=0;i<32;++i){
               if(val&(1<<i)) prefix_bit[i]+= (is_contibuted?1:-1);
            }
         };

         // Convert a 32-bits array to an integer.
         auto bin2int=[&]()->int{
            int res=0;
            for(int i=0;i<32;++i) res+= (prefix_bit[i]!=0?(1<<i):0);
            return res;
         };
```

```
        int ans=INT_MAX;
        int i=0,j=0; // Window of size 1
        while(j<n){
            contribute(nums[j],true); // Add the contibution of nums[j] into the ORing operation
            // Keep shrinking the window, while i<=j  and the OR result >= k
            while(i<=j && bin2int()>=k){
                ans=std::min(ans,j-i+1); // Minimize the answer
                contribute(nums[i],false); // Remove the contibution of nums[i] into the ORing operation
                i++; // Shrink the window
            }
            j++; // Extend the window
        }

        return ans!=INT_MAX?ans:-1;
    }
};
```