1. This problem can be divided into 2 parts -

    a. Finding the power of every element in the array.
    b. Find a subarray of length k, which has the maximum sum of powers.

2. Looking at the constraints, you know that test cases can be really large. Thus, you need to write an efficient algorithm to find the number of 1s in the binary representation of every element of the array. Note that the number of 1s in a number, is nothing but the number of set bits in its binary representation.

    The most efficient method is the bitCount() inbuilt method, which does this in O(1). Do you need extra space to store the powers or can you simply replace the array elements with their powers?

    Remember that if the original number from the array was negative, its power will also be negative. You just need to find the absolute value of the number i.e without any sign, find its power, and then convert the power to its original sign.

3. Now, for the second part, a naive approach would be to find all subarrays of size k, calculate sum of powers for all, and then return the maximum sum.

4. But do you really need to calculate the sum repeatedly for all such subarrays? Note that the subrarrays would be contiguous. Consider the following array: [1,2,-3,4,5] and k = 3

    Now for [1,2,-3], sum of elements is 0

    When we calculate the sum for the next subarray of size 3 i.e [2,-3,4], do we need to recalculate the entire sum? Or we can just look at it as a sliding window of size k. The window has moved one element further. So, we can simply subtract the element which is no longer a part of my current window, i.e 1, and add the element that just became a part of my window i.e 4. Sum for all other elements will remain the same.

    So, 0 - 1 + 4 = 3 would be the sum of this subarray. Keep track of the maxSum, and update it if you find a subarray with a greater sum.

    Can you use this approach to solve the given question?