Approach

Now that nums is sorted, consider the first element to the left of target as smaller. As smaller is the closest element to target, we want to increment it to equal target. This will cost us target - smaller operations. Now, consider the next element to the left as smaller2. Now this is the element closest to target, so we increment it using target - smaller2 operations. We continue this process until we run out of operations.

As you can see, the number of operations required is simply the difference between target and the numbers we are incrementing. Let's say that the final frequency of target was 4. We would have a sum of 4 * target. The number of operations would be this sum minus the sum of the elements before we incremented them

This brings us to our solution. We will use a sliding window over the sorted nums. For each element nums[right], we will treat target as this element and try to make every element in our window equal to target.

The size of the window is right - left + 1. That means we would have a final sum of (right - left + 1) * target. If we track the sum of our window in a variable curr, then we can calculate the required operations as (right - left + 1) * target - curr. If it requires more than k operations, we must shrink our window. Like in all sliding window problems, we will use a while loop to shrink our window by incrementing left until k operations are sufficient.

Once the while loop ends, we know that we can make all elements in the window equal to target. We can now update our answer with the current window size. The final answer will be the largest valid window we find after iterating right over the entire input.