Leetcode problem no. 2289

Link- (https://leetcode.com/problems/steps-to-make-array-non-decreasing/description/)

Problem Description:

Given an integer array arr and an integer k, modify the array by repeating it k times.

For example, if arr = [1, 2] and k = 3 then the modified array will be [1, 2, 1, 2, 1, 2].

Return the maximum sub-array sum in the modified array. Note that the length of the sub-array can be 0 and its sum in that case is 0.

As the answer can be very large, return the answer **modulo** $10^9 + 7$.

Example 1:

Input: arr = [1,2], k = 3
Output: 9

Intuition:

The intuition here is to find the maximum subarray sum in an array repeated k times: if k = 1, it's just the normal Kadane's algorithm on the array; if k > 1, the maximum subarray could either lie fully within one copy, or span across the boundary between two copies, so we duplicate the array twice and run Kadane's to capture boundary-spanning subarrays; finally, if the total sum of the array is positive,

then repeating it many times contributes extra, so we add (k-2)*sum to cover the middle repetitions, and take the maximum among all cases, returning the result modulo 1e9+7.

Code:

```
class Solution {
public int mod = (int)(1e9)+7;
public int kConcatenationMaxSum(int[] arr, int k) {
long ans = 0;
int sum = 0;
for(int i : arr){
sum += i;
ans = Math.max(ans,i);
}
ans = Math.max(ans,(long)(sum*k)%mod);
if(k>1){
int[] narr = new int[arr.length*2];
for(int i=0;i<arr.length;i++){
narr[i] = arr[i];
narr[i+arr.length] = arr[i];
}
int csum = 0;
int max = 0;
for(int i=0;i<arr.length*2;i++){
csum += narr[i];
if(csum < 0) csum = 0;
max = Math.max(max,csum);
}
ans = Math.max(ans,(max+Math.max(0l,(long)sum*(k-2)))%mod);
}
return (int)ans;
}
}
```

Time Complexity:

O(2n) = O(n)

Space Complexity:

O(2n) = O(n)

Best Time Complexity:

O(n)

Best space complexity:

O(n)