# Problem 910: Smallest Range II

## Problem Information

**Difficulty:** Medium
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You are given an integer array

nums

and an integer

k

.

For each index

i

where

0 <= i < nums.length

, change

nums[i]

to be either

nums[i] + k

or

nums[i] - k

.

The

score

of

nums

is the difference between the maximum and minimum elements in

nums

.

Return

the minimum

score

of

nums

after changing the values at each index

.

Example 1:

Input:

nums = [1], k = 0

Output:

0

Explanation:

The score is max(nums) - min(nums) = 1 - 1 = 0.

Example 2:

Input:

nums = [0,10], k = 2

Output:

6

Explanation:

Change nums to be [2, 8]. The score is max(nums) - min(nums) = 8 - 2 = 6.

Example 3:

Input:

nums = [1,3,6], k = 3

Output:

3

Explanation:

Change nums to be [4, 6, 3]. The score is max(nums) - min(nums) = 6 - 3 = 3.

Constraints:

1 <= nums.length <= 10

4

0 <= nums[i] <= 10

4

0 <= k <= 10

4

## Code Snippets

**C++:**

```
class Solution {
public:
int smallestRangeII(vector<int>& nums, int k) {

}
};
```

**Java:**

```
class Solution {
public int smallestRangeII(int[] nums, int k) {

}
}
```

**Python3:**

```
class Solution:
def smallestRangeII(self, nums: List[int], k: int) -> int:
```

**Python:**

```
class Solution(object):
def smallestRangeII(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var smallestRangeII = function(nums, k) {

};
```

**TypeScript:**

```
function smallestRangeII(nums: number[], k: number): number {

};
```

**C#:**

```
public class Solution {
public int SmallestRangeII(int[] nums, int k) {

}
}
```

**C:**

```
int smallestRangeII(int* nums, int numsSize, int k) {

}
```

**Go:**

```
func smallestRangeII(nums []int, k int) int {

}
```

**Kotlin:**

```
class Solution {
fun smallestRangeII(nums: IntArray, k: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func smallestRangeII(_ nums: [Int], _ k: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn smallest_range_ii(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def smallest_range_ii(nums, k)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function smallestRangeII($nums, $k) {
```

```
}
}
```

**Dart:**

```
class Solution {
int smallestRangeII(List<int> nums, int k) {

}
}
```

**Scala:**

```
object Solution {
def smallestRangeII(nums: Array[Int], k: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec smallest_range_ii(nums :: [integer], k :: integer) :: integer
def smallest_range_ii(nums, k) do

end
end
```

**Erlang:**

```
-spec smallest_range_ii(Nums :: [integer()], K :: integer()) -> integer().
smallest_range_ii(Nums, K) ->
.
```

**Racket:**

```
(define/contract (smallest-range-ii nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Smallest Range II
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int smallestRangeII(vector<int>& nums, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Smallest Range II
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int smallestRangeII(int[] nums, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Smallest Range II
```

```
Difficulty: Medium
Tags: array, greedy, math, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def smallestRangeII(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def smallestRangeII(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Smallest Range II
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var smallestRangeII = function(nums, k) {
```

```
};
```

### TypeScript Solution:

```
/**
* Problem: Smallest Range II
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function smallestRangeII(nums: number[], k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Smallest Range II
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int SmallestRangeII(int[] nums, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Smallest Range II
```

```
* Difficulty: Medium
* Tags: array, greedy, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int smallestRangeII(int* nums, int numsSize, int k) {

}
```

### Go Solution:

```
// Problem: Smallest Range II
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func smallestRangeII(nums []int, k int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun smallestRangeII(nums: IntArray, k: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func smallestRangeII(_ nums: [Int], _ k: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Smallest Range II
// Difficulty: Medium
// Tags: array, greedy, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn smallest_range_ii(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def smallest_range_ii(nums, k)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function smallestRangeII($nums, $k) {

}
}
```

### Dart Solution:

```
class Solution {
int smallestRangeII(List<int> nums, int k) {

}
}
```

### Scala Solution:

```
object Solution {
def smallestRangeII(nums: Array[Int], k: Int): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec smallest_range_ii(nums :: [integer], k :: integer) :: integer
def smallest_range_ii(nums, k) do

end
end
```

### Erlang Solution:

```
-spec smallest_range_ii(Nums :: [integer()], K :: integer()) -> integer().
smallest_range_ii(Nums, K) ->
.
```

### Racket Solution:

```
(define/contract (smallest-range-ii nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```