Problem 1675: Minimize Deviation in Array

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an array

nums

of

n

positive integers.

You can perform two types of operations on any element of the array any number of times:

If the element is

even

,

divide

it by

2

.

For example, if the array is

[1,2,3,4]

, then you can do this operation on the last element, and the array will be

[1,2,3,

2

].

If the element is

odd

,

multiply

it by

2

.

For example, if the array is

[1,2,3,4]

, then you can do this operation on the first element, and the array will be

[

2

,2,3,4].

The

deviation

of the array is the

maximum difference

between any two elements in the array.

Return

the

minimum deviation

the array can have after performing some number of operations.

Example 1:

Input:

nums = [1,2,3,4]

Output:

1

Explanation:

You can transform the array to [1,2,3,

2

], then to [

2

,2,3,2], then the deviation will be 3 - 2 = 1.

Example 2:

Input:

nums = [4,1,5,20,3]

Output:

3

Explanation:

You can transform the array after two operations to [4,

2

,5,

5

,3], then the deviation will be 5 - 2 = 3.

Example 3:

Input:

nums = [2,10,8]

Output:

3

Constraints:

n == nums.length

2 <= n <= 5 * 10

4

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minimumDeviation(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int minimumDeviation(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var minimumDeviation = function(nums) {

};
```

TypeScript:

```
function minimumDeviation(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumDeviation(int[] nums) {

}
}
```

C:

```
int minimumDeviation(int* nums, int numsSize) {

}
```

Go:

```
func minimumDeviation(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumDeviation(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumDeviation(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_deviation(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def minimum_deviation(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function minimumDeviation($nums) {

}
}
```

Dart:

```
class Solution {
int minimumDeviation(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def minimumDeviation(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_deviation(nums :: [integer]) :: integer
def minimum_deviation(nums) do

end
end
```

Erlang:

```
-spec minimum_deviation(Nums :: [integer()]) -> integer().
minimum_deviation(Nums) ->
.
```

Racket:

```
(define/contract (minimum-deviation nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Deviation in Array
* Difficulty: Hard
* Tags: array, greedy, queue, heap
*
* 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 minimumDeviation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Deviation in Array
* Difficulty: Hard
* Tags: array, greedy, queue, heap
*
* 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 minimumDeviation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Deviation in Array
Difficulty: Hard
Tags: array, greedy, queue, heap

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 minimumDeviation(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize Deviation in Array
* Difficulty: Hard
* Tags: array, greedy, queue, heap
*
* 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
* @return {number}
*/
var minimumDeviation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Deviation in Array
* Difficulty: Hard
* Tags: array, greedy, queue, heap
*
* 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 minimumDeviation(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimize Deviation in Array
* Difficulty: Hard
* Tags: array, greedy, queue, heap
*
* 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 MinimumDeviation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Deviation in Array
* Difficulty: Hard
* Tags: array, greedy, queue, heap
*
* 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 minimumDeviation(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimize Deviation in Array
// Difficulty: Hard
// Tags: array, greedy, queue, heap
//
// 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 minimumDeviation(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumDeviation(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumDeviation(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimize Deviation in Array
// Difficulty: Hard
// Tags: array, greedy, queue, heap
//
// 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 minimum_deviation(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def minimum_deviation(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function minimumDeviation($nums) {

}
}
```

Dart Solution:

```
class Solution {
int minimumDeviation(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def minimumDeviation(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_deviation(nums :: [integer]) :: integer
def minimum_deviation(nums) do

end
end
```

Erlang Solution:

```
-spec minimum_deviation(Nums :: [integer()]) -> integer().
minimum_deviation(Nums) ->
.
```

Racket Solution:

```
(define/contract (minimum-deviation nums)
(-> (listof exact-integer?) exact-integer?)
)
```