Problem 2163: Minimum Difference in Sums After Removal of Elements

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

consisting of

3 * n

elements.

You are allowed to remove any

subsequence

of elements of size

exactly

n

from

nums

. The remaining

2 * n

elements will be divided into two

equal

parts:

The first

n

elements belonging to the first part and their sum is

sum

first

.

The next

n

elements belonging to the second part and their sum is

sum

second

.

The

difference in sums

of the two parts is denoted as

sum

first

- sum

second

.

For example, if

sum

first

= 3

and

sum

second

= 2

, their difference is

1

.

Similarly, if

sum

first

= 2

and

sum

second

= 3

, their difference is

-1

.

Return

the

minimum difference

possible between the sums of the two parts after the removal of

n

elements

.

Example 1:

Input:

nums = [3,1,2]

Output:

-1

Explanation:

Here, nums has 3 elements, so n = 1. Thus we have to remove 1 element from nums and divide the array into two equal parts. - If we remove nums[0] = 3, the array will be [1,2]. The difference in sums of the two parts will be 1 - 2 = -1. - If we remove nums[1] = 1, the array will be [3,2]. The difference in sums of the two parts will be 3 - 2 = 1. - If we remove nums[2] = 2, the array will be [3,1]. The difference in sums of the two parts will be 3 - 1 = 2. The minimum difference between sums of the two parts is min(-1,1,2) = -1.

Example 2:

Input:

nums = [7,9,5,8,1,3]

Output:

1

Explanation:

Here n = 2. So we must remove 2 elements and divide the remaining array into two parts containing two elements each. If we remove nums[2] = 5 and nums[3] = 8, the resultant array will be [7,9,1,3]. The difference in sums will be (7+9) - (1+3) = 12. To obtain the minimum difference, we should remove nums[1] = 9 and nums[4] = 1. The resultant array becomes [7,5,8,3]. The difference in sums of the two parts is (7+5) - (8+3) = 1. It can be shown that it is not possible to obtain a difference smaller than 1.

Constraints:

nums.length == 3 * n

1 <= n <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long minimumDifference(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MinimumDifference(int[] nums) {

}
}
```

C:

```
long long minimumDifference(int* nums, int numsSize) {

}
```

Go:

```
func minimumDifference(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun minimumDifference(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn minimum_difference(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Difference in Sums After Removal of Elements
* Difficulty: Hard
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Difference in Sums After Removal of Elements
* Difficulty: Hard
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long minimumDifference(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Difference in Sums After Removal of Elements
Difficulty: Hard
Tags: array, dp, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Difference in Sums After Removal of Elements
* Difficulty: Hard
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @return {number}
*/
var minimumDifference = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Difference in Sums After Removal of Elements
* Difficulty: Hard
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

function minimumDifference(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Difference in Sums After Removal of Elements
* Difficulty: Hard
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long MinimumDifference(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Difference in Sums After Removal of Elements
* Difficulty: Hard
* Tags: array, dp, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long minimumDifference(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Difference in Sums After Removal of Elements
// Difficulty: Hard
// Tags: array, dp, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumDifference(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumDifference(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Difference in Sums After Removal of Elements
// Difficulty: Hard
// Tags: array, dp, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_difference(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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