Problem 2035: Partition Array Into Two Arrays to Minimize Sum Difference

Problem Information

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

Problem Description

You are given an integer array

nums

of

2 * n

integers. You need to partition

nums

into

two

arrays of length

n

to

minimize the absolute difference

of the

sums

of the arrays. To partition

nums

, put each element of

nums

into

one

of the two arrays.

Return

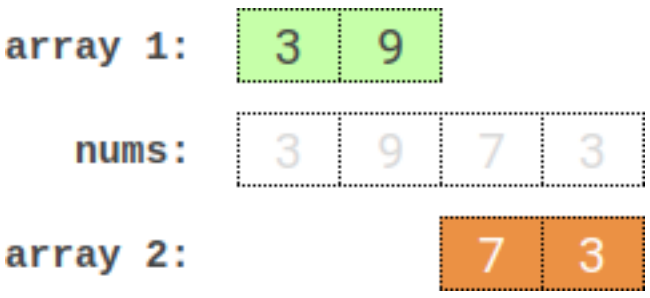
the

minimum

possible absolute difference

.

Example 1:



Input:

nums = [3,9,7,3]

Output:

2

Explanation:

One optimal partition is: [3,9] and [7,3]. The absolute difference between the sums of the arrays is abs((3 + 9) - (7 + 3)) = 2.

Example 2:

Input:

nums = [-36,36]

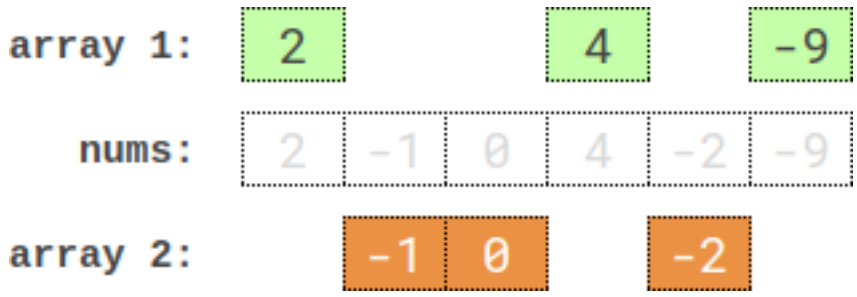
Output:

72

Explanation:

One optimal partition is: [-36] and [36]. The absolute difference between the sums of the arrays is abs((-36) - (36)) = 72.

Example 3:



Input:

nums = [2,-1,0,4,-2,-9]

Output:

0

Explanation:

One optimal partition is: [2,4,-9] and [-1,0,-2]. The absolute difference between the sums of the arrays is abs((2 + 4 + -9) - (-1 + 0 + -2)) = 0.

Constraints:

1 <= n <= 15

nums.length == 2 * n

-10

7

<= nums[i] <= 10

7

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

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]): Int = {

}
}
```

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: Partition Array Into Two Arrays to Minimize Sum Difference
* Difficulty: Hard
* Tags: array, dp, search
*
* 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:
int minimumDifference(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Array Into Two Arrays to Minimize Sum Difference
* Difficulty: Hard
* Tags: array, dp, search
*
* 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 int minimumDifference(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Partition Array Into Two Arrays to Minimize Sum Difference
Difficulty: Hard
Tags: array, dp, search

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: Partition Array Into Two Arrays to Minimize Sum Difference
* Difficulty: Hard
* Tags: array, dp, search
*
* 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: Partition Array Into Two Arrays to Minimize Sum Difference
* Difficulty: Hard
* Tags: array, dp, search
*
* 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: Partition Array Into Two Arrays to Minimize Sum Difference
* Difficulty: Hard
* Tags: array, dp, search
*
* 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 int MinimumDifference(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Partition Array Into Two Arrays to Minimize Sum Difference
* Difficulty: Hard
* Tags: array, dp, search
*
* 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
*/

int minimumDifference(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Partition Array Into Two Arrays to Minimize Sum Difference
// Difficulty: Hard
// Tags: array, dp, search
//
// 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) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Partition Array Into Two Arrays to Minimize Sum Difference
// Difficulty: Hard
// Tags: array, dp, search
//
// 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>) -> i32 {

}
}
```

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]): Int = {

}
}
```

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?)
)
```