Problem 3701: Compute Alternating Sum

Problem Information

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

Problem Description

You are given an integer array

nums

.

The

alternating sum

of

nums

is the value obtained by

adding

elements at even indices and

subtracting

elements at odd indices. That is,

nums[0] - nums[1] + nums[2] - nums[3]...

Return an integer denoting the alternating sum of

nums

.

Example 1:

Input:

nums = [1,3,5,7]

Output:

-4

Explanation:

Elements at even indices are

nums[0] = 1

and

nums[2] = 5

because 0 and 2 are even numbers.

Elements at odd indices are

nums[1] = 3

and

nums[3] = 7

because 1 and 3 are odd numbers.

The alternating sum is

nums[0] - nums[1] + nums[2] - nums[3] = 1 - 3 + 5 - 7 = -4

.

Example 2:

Input:

nums = [100]

Output:

100

Explanation:

The only element at even indices is

nums[0] = 100

because 0 is an even number.

There are no elements on odd indices.

The alternating sum is

nums[0] = 100

.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (alternating-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Compute Alternating Sum
* Difficulty: Easy
* Tags: array
*
* 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 alternatingSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Compute Alternating Sum
* Difficulty: Easy
* Tags: array
*
* 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 alternatingSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Compute Alternating Sum
Difficulty: Easy
Tags: array

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Compute Alternating Sum
* Difficulty: Easy
* Tags: array
*
* 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 alternatingSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Compute Alternating Sum
* Difficulty: Easy
* Tags: array
*
* 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 alternatingSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Compute Alternating Sum
* Difficulty: Easy
* Tags: array
*
* 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 AlternatingSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Compute Alternating Sum
* Difficulty: Easy
* Tags: array
*
* 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 alternatingSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Compute Alternating Sum
// Difficulty: Easy
// Tags: array
//
// 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 alternatingSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Compute Alternating Sum
// Difficulty: Easy
// Tags: array
//
// 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 alternating_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (alternating-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```