Problem 3727: Maximum Alternating Sum of Squares

Problem Information

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

Problem Description

You are given an integer array

nums

. You may

rearrange the elements

in any order.

The

alternating score

of an array

arr

is defined as:

score = arr[0]

2

- arr[1]

2

+ arr[2]

2

- arr[3]

2

+ ...

Return an integer denoting the

maximum possible alternating score

of

nums

after rearranging its elements.

Example 1:

Input:

nums = [1,2,3]

Output:

12

Explanation:

A possible rearrangement for

nums

is

[2,1,3]

, which gives the maximum alternating score among all possible rearrangements.

The alternating score is calculated as:

score = 2

2

- 1

2

+ 3

2

= 4 - 1 + 9 = 12

Example 2:

Input:

nums = [1,-1,2,-2,3,-3]

Output:

16

Explanation:

A possible rearrangement for

nums

is

[-3,-1,-2,1,3,2]

, which gives the maximum alternating score among all possible rearrangements.

The alternating score is calculated as:

score = (-3)

2

- (-1)

2

+ (-2)

2

- (1)

2

+ (3)

2

- (2)

2

= 9 - 1 + 4 - 1 + 9 - 4 = 16

Constraints:

1 <= nums.length <= 10

5

-4 * 10

4

<= nums[i] <= 4 * 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Alternating Sum of Squares
* Difficulty: Medium
* Tags: array, greedy, 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:
long long maxAlternatingSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Alternating Sum of Squares
* Difficulty: Medium
* Tags: array, greedy, 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 long maxAlternatingSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Alternating Sum of Squares
Difficulty: Medium
Tags: array, greedy, 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 maxAlternatingSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Alternating Sum of Squares
* Difficulty: Medium
* Tags: array, greedy, 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
* @return {number}
*/
var maxAlternatingSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Alternating Sum of Squares
* Difficulty: Medium
* Tags: array, greedy, 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 maxAlternatingSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Alternating Sum of Squares
* Difficulty: Medium
* Tags: array, greedy, 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 long MaxAlternatingSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Alternating Sum of Squares
* Difficulty: Medium
* Tags: array, greedy, 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
*/

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

```
}
```

Go Solution:

```
// Problem: Maximum Alternating Sum of Squares
// Difficulty: Medium
// Tags: array, greedy, 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 maxAlternatingSum(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Alternating Sum of Squares
// Difficulty: Medium
// Tags: array, greedy, 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 max_alternating_sum(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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