Problem 3695: Maximize Alternating Sum Using Swaps

Problem Information

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

Problem Description

You are given an integer array

nums

.

You want to maximize the

alternating sum

of

nums

, which is defined as 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]...

You are also given a 2D integer array

swaps

where

swaps[i] = [p

i

, q

i

]

. For each pair

[p

i

, q

i

]

in

swaps

, you are allowed to swap the elements at indices

p

i

and

q

i

. These swaps can be performed any number of times and in any order.

Return the maximum possible

alternating sum

of

nums

.

Example 1:

Input:

nums = [1,2,3], swaps = [[0,2],[1,2]]

Output:

4

Explanation:

The maximum alternating sum is achieved when

nums

is

[2, 1, 3]

or

[3, 1, 2]

. As an example, you can obtain

nums = [2, 1, 3]

as follows.

Swap

nums[0]

and

nums[2]

.

nums

is now

[3, 2, 1]

.

Swap

nums[1]

and

nums[2]

.

nums

is now

[3, 1, 2]

.

Swap

nums[0]

and

nums[2]

.

nums

is now

[2, 1, 3]

.

Example 2:

Input:

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

Output:

2

Explanation:

The maximum alternating sum is achieved by not performing any swaps.

Example 3:

Input:

nums = [1,1000000000,1,1000000000,1,1000000000], swaps = []

Output:

-2999999997

Explanation:

Since we cannot perform any swaps, the maximum alternating sum is achieved by not performing any swaps.

Constraints:

2 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

0 <= swaps.length <= 10

5

swaps[i] = [p

i

, q

i

]

0 <= p

i

< q

i

<= nums.length - 1

[p

i

, q

i

] != [p

j

, q

j

]

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
long long maxAlternatingSum(int* nums, int numsSize, int** swaps, int
swapsSize, int* swapsColSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun maxAlternatingSum(nums: IntArray, swaps: Array<IntArray>): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximize Alternating Sum Using Swaps
* Difficulty: Hard
* Tags: array, graph, 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, vector<vector<int>>& swaps) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Alternating Sum Using Swaps
* Difficulty: Hard
* Tags: array, graph, 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, int[][] swaps) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Alternating Sum Using Swaps
Difficulty: Hard
Tags: array, graph, 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], swaps: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Alternating Sum Using Swaps
* Difficulty: Hard
* Tags: array, graph, 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
* @param {number[][]} swaps
* @return {number}
*/
var maxAlternatingSum = function(nums, swaps) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Alternating Sum Using Swaps
* Difficulty: Hard
* Tags: array, graph, 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[], swaps: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Alternating Sum Using Swaps
* Difficulty: Hard
```

```
* Tags: array, graph, 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, int[][] swaps) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Alternating Sum Using Swaps
* Difficulty: Hard
* Tags: array, graph, 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, int** swaps, int
swapsSize, int* swapsColSize) {

}
```

Go Solution:

```
// Problem: Maximize Alternating Sum Using Swaps
// Difficulty: Hard
// Tags: array, graph, 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, swaps [][]int) int64 {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun maxAlternatingSum(nums: IntArray, swaps: Array<IntArray>): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Alternating Sum Using Swaps
// Difficulty: Hard
// Tags: array, graph, 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>, swaps: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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