Problem 1913: Maximum Product Difference Between Two Pairs

Problem Information

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

Problem Description

The

product difference

between two pairs

(a, b)

and

(c, d)

is defined as

(a * b) - (c * d)

.

For example, the product difference between

(5, 6)

and

(2, 7)

is

(5 * 6) - (2 * 7) = 16

.

Given an integer array

nums

, choose four

distinct

indices

w

,

x

,

y

, and

z

such that the

product difference

between pairs

(nums[w], nums[x])

and

(nums[y], nums[z])

is

maximized

.

Return

the

maximum

such product difference

.

Example 1:

Input:

nums = [5,6,2,7,4]

Output:

34

Explanation:

We can choose indices 1 and 3 for the first pair (6, 7) and indices 2 and 4 for the second pair (2, 4). The product difference is (6 * 7) - (2 * 4) = 34.

Example 2:

Input:

nums = [4,2,5,9,7,4,8]

Output:

64

Explanation:

We can choose indices 3 and 6 for the first pair (9, 8) and indices 1 and 5 for the second pair (2, 4). The product difference is (9 * 8) - (2 * 4) = 64.

Constraints:

4 <= nums.length <= 10

4

1 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Scala:

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

}
}
```

Racket:

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

)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product Difference Between Two Pairs
* Difficulty: Easy
* Tags: array, 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:
int maxProductDifference(vector<int>& nums) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Product Difference Between Two Pairs
* Difficulty: Easy
* Tags: array, 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 int maxProductDifference(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product Difference Between Two Pairs
Difficulty: Easy
Tags: array, 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 maxProductDifference(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product Difference Between Two Pairs
* Difficulty: Easy
* Tags: array, 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 maxProductDifference = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product Difference Between Two Pairs
* Difficulty: Easy
* Tags: array, 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 maxProductDifference(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product Difference Between Two Pairs
* Difficulty: Easy
* Tags: array, 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 int MaxProductDifference(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product Difference Between Two Pairs
* Difficulty: Easy
* Tags: array, 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
*/

int maxProductDifference(int* nums, int numsSize){

}
```

Go Solution:

```
// Problem: Maximum Product Difference Between Two Pairs
// Difficulty: Easy
// Tags: array, 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 maxProductDifference(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product Difference Between Two Pairs
// Difficulty: Easy
// Tags: array, 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_product_difference(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```

Racket Solution:

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

)
```