Problem 3745: Maximize Expression of Three Elements

Problem Information

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

Problem Description

You are given an integer array

nums

.

Choose three elements

a

,

b

, and

c

from

nums

at

distinct

indices such that the value of the expression

a + b - c

is maximized.

Return an integer denoting the

maximum possible value

of this expression.

Example 1:

Input:

nums = [1,4,2,5]

Output:

8

Explanation:

We can choose

a = 4

,

b = 5

, and

c = 1

. The expression value is

4 + 5 - 1 = 8

, which is the maximum possible.

Example 2:

Input:

nums = [-2,0,5,-2,4]

Output:

11

Explanation:

We can choose

a = 5

,

b = 4

, and

c = -2

. The expression value is

5 + 4 - (-2) = 11

, which is the maximum possible.

Constraints:

3 <= nums.length <= 100

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximize-expression-of-three nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Expression of Three Elements
* Difficulty: Easy
* 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:
int maximizeExpressionOfThree(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Expression of Three Elements
* Difficulty: Easy
* 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 int maximizeExpressionOfThree(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Expression of Three Elements
Difficulty: Easy
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 maximizeExpressionOfThree(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Expression of Three Elements
* Difficulty: Easy
* 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 maximizeExpressionOfThree = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Expression of Three Elements
* Difficulty: Easy
* 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 maximizeExpressionOfThree(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Maximize Expression of Three Elements
* Difficulty: Easy
* 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 int MaximizeExpressionOfThree(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Expression of Three Elements
* Difficulty: Easy
* 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
*/

int maximizeExpressionOfThree(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximize Expression of Three Elements
// Difficulty: Easy
```

```
// 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 maximizeExpressionOfThree(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Expression of Three Elements
// Difficulty: Easy
// 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 maximize_expression_of_three(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximize-expression-of-three nums)
(-> (listof exact-integer?) exact-integer?)
)
```