

Problem 2149: Rearrange Array Elements by Sign

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

of

even

length consisting of an

equal

number of positive and negative integers.

You should return the array of nums such that the array follows the given conditions:

Every

consecutive pair

of integers have

opposite signs

.

For all integers with the same sign, the

order

in which they were present in

nums

is

preserved

.

The rearranged array begins with a positive integer.

Return

the modified array after rearranging the elements to satisfy the aforementioned conditions

.

Example 1:

Input:

nums = [3,1,-2,-5,2,-4]

Output:

[3,-2,1,-5,2,-4]

Explanation:

The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4]. The only possible way to rearrange them such that they satisfy all conditions is [3,-2,1,-5,2,-4]. Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions.

Example 2:

Input:

nums = [-1,1]

Output:

[1,-1]

Explanation:

1 is the only positive integer and -1 the only negative integer in nums. So nums is rearranged to [1,-1].

Constraints:

$2 \leq \text{nums.length} \leq 2 * 10$

5

nums.length

is

even

$1 \leq |\text{nums}[i]| \leq 10$

5

nums

consists of

equal

number of positive and negative integers.

It is not required to do the modifications in-place.

Code Snippets

C++:

```
class Solution {  
public:  
    vector<int> rearrangeArray(vector<int>& nums) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int[] rearrangeArray(int[] nums) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

```

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

};

```

TypeScript:

```

function rearrangeArray(nums: number[]): number[] {

};

```

C#:

```

public class Solution {
    public int[] RearrangeArray(int[] nums) {

    }
}

```

C:

```

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* rearrangeArray(int* nums, int numsSize, int* returnSize) {

}

```

Go:

```

func rearrangeArray(nums []int) []int {

}

```

Kotlin:

```

class Solution {
    fun rearrangeArray(nums: IntArray): IntArray {

    }
}

```

```
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer[]  
     */  
    function rearrangeArray($nums) {  
  
    }  
}
```

Dart:

```

class Solution {
    List<int> rearrangeArray(List<int> nums) {

    }

}

```

Scala:

```

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

    }

}

```

Elixir:

```

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

    end

end

```

Erlang:

```

-spec rearrange_array(Nums :: [integer()]) -> [integer()].
rearrange_array(Nums) ->

.

```

Racket:

```

(define/contract (rearrange-array nums)
  (-> (listof exact-integer?) (listof exact-integer?))
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: Rearrange Array Elements by Sign

```

```

* Difficulty: Medium
* 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:
vector<int> rearrangeArray(vector<int>& nums) {

}
};

```

Java Solution:

```

/**
* Problem: Rearrange Array Elements by Sign
* Difficulty: Medium
* 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[] rearrangeArray(int[] nums) {

}
}

```

Python3 Solution:

```

"""
Problem: Rearrange Array Elements by Sign
Difficulty: Medium
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 rearrangeArray(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Rearrange Array Elements by Sign
 * Difficulty: Medium
 * 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 rearrangeArray = function(nums) {

};

```

TypeScript Solution:

```

/**
 * Problem: Rearrange Array Elements by Sign
 * Difficulty: Medium
 * 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 rearrangeArray(nums: number[]): number[] {

};

```

C# Solution:

```

/*
 * Problem: Rearrange Array Elements by Sign
 * Difficulty: Medium
 * 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[] RearrangeArray(int[] nums) {

    }
}

```

C Solution:

```

/*
 * Problem: Rearrange Array Elements by Sign
 * Difficulty: Medium
 * 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

```

```

*/

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* rearrangeArray(int* nums, int numsSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Rearrange Array Elements by Sign
// Difficulty: Medium
// 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 rearrangeArray(nums []int) []int {

}

```

Kotlin Solution:

```

class Solution {
    fun rearrangeArray(nums: IntArray): IntArray {

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```
// Problem: Rearrange Array Elements by Sign
// Difficulty: Medium
// 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 rearrange_array(nums: Vec<i32>) -> Vec<i32> {

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

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

    }
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (rearrange-array nums)  
  (-> (listof exact-integer?) (listof exact-integer?))  
)
```