Problem 561: Array Partition

Problem Information

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

Problem Description

Given an integer array

nums

of

2n

integers, group these integers into

n

pairs

(a

1

, b

1

), (a

2

, b

2

), ..., (a

n

, b

n

)

such that the sum of

min(a

i

, b

i

)

for all

i

is

maximized

. Return

the maximized sum

.

Example 1:

Input:

nums = [1,4,3,2]

Output:

4

Explanation:

All possible pairings (ignoring the ordering of elements) are: 1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3 2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3 3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4 So the maximum possible sum is 4.

Example 2:

Input:

nums = [6,2,6,5,1,2]

Output:

9

Explanation:

The optimal pairing is (2, 1), (2, 5), (6, 6). min(2, 1) + min(2, 5) + min(6, 6) = 1 + 2 + 6 = 9.

Constraints:

1 <= n <= 10

4

nums.length == 2 * n

-10

4

<= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Array Partition
* 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 arrayPairSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Array Partition
* 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 arrayPairSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Array Partition
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 arrayPairSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Array Partition
* 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 arrayPairSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Array Partition
* 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 arrayPairSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Array Partition
* 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 ArrayPairSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Array Partition
* 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 arrayPairSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Array Partition
// 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 arrayPairSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Array Partition
// 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 array_pair_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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