Problem 2491: Divide Players Into Teams of Equal Skill

Problem Information

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

Problem Description

You are given a positive integer array

skill

of

even

length

n

where

skill[i]

denotes the skill of the

i

th

player. Divide the players into

n / 2

teams of size

2

such that the total skill of each team is

equal

.

The

chemistry

of a team is equal to the

product

of the skills of the players on that team.

Return

the sum of the

chemistry

of all the teams, or return

-1

if there is no way to divide the players into teams such that the total skill of each team is equal.

Example 1:

Input:

skill = [3,2,5,1,3,4]

Output:

22

Explanation:

Divide the players into the following teams: (1, 5), (2, 4), (3, 3), where each team has a total skill of 6. The sum of the chemistry of all the teams is: 1 * 5 + 2 * 4 + 3 * 3 = 5 + 8 + 9 = 22.

Example 2:

Input:

skill = [3,4]

Output:

12

Explanation:

The two players form a team with a total skill of 7. The chemistry of the team is 3 * 4 = 12.

Example 3:

Input:

skill = [1,1,2,3]

Output:

-1

Explanation:

There is no way to divide the players into teams such that the total skill of each team is equal.

Constraints:

2 <= skill.length <= 10

5

skill.length

is even.

1 <= skill[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
long long dividePlayers(vector<int>& skill) {

}
};
```

Java:

```
class Solution {
public long dividePlayers(int[] skill) {

}
}
```

Python3:

```
class Solution:
def dividePlayers(self, skill: List[int]) -> int:
```

Python:

```
class Solution(object):
def dividePlayers(self, skill):
"""
:type skill: List[int]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function dividePlayers(skill: number[]): number {

};
```

C#:

```
public class Solution {
public long DividePlayers(int[] skill) {

}
}
```

C:

```
long long dividePlayers(int* skill, int skillSize) {

}
```

Go:

```
func dividePlayers(skill []int) int64 {

}
```

Kotlin:

```
class Solution {
fun dividePlayers(skill: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func dividePlayers(_ skill: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn divide_players(skill: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} skill
# @return {Integer}
def divide_players(skill)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $skill
* @return Integer
*/
function dividePlayers($skill) {

}
}
```

Dart:

```
class Solution {
int dividePlayers(List<int> skill) {

}
}
```

Scala:

```
object Solution {
def dividePlayers(skill: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec divide_players(skill :: [integer]) :: integer
def divide_players(skill) do

end
end
```

Erlang:

```
-spec divide_players(Skill :: [integer()]) -> integer().
divide_players(Skill) ->
.
```

Racket:

```
(define/contract (divide-players skill)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Players Into Teams of Equal Skill
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
long long dividePlayers(vector<int>& skill) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Players Into Teams of Equal Skill
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public long dividePlayers(int[] skill) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide Players Into Teams of Equal Skill
Difficulty: Medium
Tags: array, hash, sort
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def dividePlayers(self, skill: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Players Into Teams of Equal Skill
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} skill
* @return {number}
*/
var dividePlayers = function(skill) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide Players Into Teams of Equal Skill
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function dividePlayers(skill: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Divide Players Into Teams of Equal Skill
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public long DividePlayers(int[] skill) {

}
}
```

C Solution:

```
/*
* Problem: Divide Players Into Teams of Equal Skill
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

long long dividePlayers(int* skill, int skillSize) {

}
```

Go Solution:

```
// Problem: Divide Players Into Teams of Equal Skill
// Difficulty: Medium
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func dividePlayers(skill []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun dividePlayers(skill: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func dividePlayers(_ skill: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Divide Players Into Teams of Equal Skill
// Difficulty: Medium
// Tags: array, hash, sort
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn divide_players(skill: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} skill
# @return {Integer}
def divide_players(skill)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $skill
* @return Integer
*/
function dividePlayers($skill) {

}
}
```

Dart Solution:

```
class Solution {
int dividePlayers(List<int> skill) {

}
}
```

Scala Solution:

```
object Solution {
def dividePlayers(skill: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec divide_players(skill :: [integer]) :: integer
def divide_players(skill) do

end
end
```

Erlang Solution:

```
-spec divide_players(Skill :: [integer()]) -> integer().
divide_players(Skill) ->
.
```

Racket Solution:

```
(define/contract (divide-players skill)
(-> (listof exact-integer?) exact-integer?)
)
```