Problem 2244: Minimum Rounds to Complete All Tasks

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

tasks

, where

tasks[i]

represents the difficulty level of a task. In each round, you can complete either 2 or 3 tasks of the

same difficulty level

.

Return

the

minimum

rounds required to complete all the tasks, or

-1

if it is not possible to complete all the tasks.

Example 1:

Input:

tasks = [2,2,3,3,2,4,4,4,4,4]

Output:

4

Explanation:

To complete all the tasks, a possible plan is: - In the first round, you complete 3 tasks of difficulty level 2. - In the second round, you complete 2 tasks of difficulty level 3. - In the third round, you complete 3 tasks of difficulty level 4. - In the fourth round, you complete 2 tasks of difficulty level 4. It can be shown that all the tasks cannot be completed in fewer than 4 rounds, so the answer is 4.

Example 2:

Input:

tasks = [2,3,3]

Output:

-1

Explanation:

There is only 1 task of difficulty level 2, but in each round, you can only complete either 2 or 3 tasks of the same difficulty level. Hence, you cannot complete all the tasks, and the answer is -1.

Constraints:

1 <= tasks.length <= 10

5

1 <= tasks[i] <= 10

9

Note:

This question is the same as

2870: Minimum Number of Operations to Make Array Empty.

Code Snippets

C++:

```
class Solution {
public:
int minimumRounds(vector<int>& tasks) {

}
};
```

Java:

```
class Solution {
public int minimumRounds(int[] tasks) {

}
}
```

Python3:

```
class Solution:
def minimumRounds(self, tasks: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumRounds(tasks: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumRounds(int[] tasks) {

}
}
```

C:

```
int minimumRounds(int* tasks, int tasksSize) {

}
```

Go:

```
func minimumRounds(tasks []int) int {
}
```

Kotlin:

```
class Solution {
fun minimumRounds(tasks: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumRounds(_ tasks: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_rounds(tasks: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} tasks
# @return {Integer}
def minimum_rounds(tasks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $tasks
* @return Integer
*/
function minimumRounds($tasks) {

}
}
```

Dart:

```
class Solution {
int minimumRounds(List<int> tasks) {

}
}
```

Scala:

```
object Solution {
def minimumRounds(tasks: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_rounds(tasks :: [integer]) :: integer
def minimum_rounds(tasks) do

end
end
```

Erlang:

```
-spec minimum_rounds(Tasks :: [integer()]) -> integer().
minimum_rounds(Tasks) ->
.
```

Racket:

```
(define/contract (minimum-rounds tasks)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Rounds to Complete All Tasks
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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:
int minimumRounds(vector<int>& tasks) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Rounds to Complete All Tasks
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 int minimumRounds(int[] tasks) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Rounds to Complete All Tasks
Difficulty: Medium
Tags: array, greedy, hash

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 minimumRounds(self, tasks: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Rounds to Complete All Tasks
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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[]} tasks
* @return {number}
*/
var minimumRounds = function(tasks) {
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Rounds to Complete All Tasks
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 minimumRounds(tasks: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Rounds to Complete All Tasks
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 int MinimumRounds(int[] tasks) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Rounds to Complete All Tasks
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minimumRounds(int* tasks, int tasksSize) {

}
```

Go Solution:

```
// Problem: Minimum Rounds to Complete All Tasks
// Difficulty: Medium
// Tags: array, greedy, hash
//
// 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 minimumRounds(tasks []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumRounds(tasks: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumRounds(_ tasks: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Rounds to Complete All Tasks
// Difficulty: Medium
// Tags: array, greedy, hash
//
// 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 minimum_rounds(tasks: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} tasks
# @return {Integer}
def minimum_rounds(tasks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $tasks
* @return Integer
*/
function minimumRounds($tasks) {

}
}
```

Dart Solution:

```
class Solution {
int minimumRounds(List<int> tasks) {
}
}
```

Scala Solution:

```
object Solution {
def minimumRounds(tasks: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_rounds(tasks :: [integer]) :: integer
def minimum_rounds(tasks) do

end
end
```

Erlang Solution:

```
-spec minimum_rounds(Tasks :: [integer()]) -> integer().
minimum_rounds(Tasks) ->
.
```

Racket Solution:

```
(define/contract (minimum-rounds tasks)
(-> (listof exact-integer?) exact-integer?)
)
```