Problem 3232: Find if Digit Game Can Be Won

Problem Information

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

Problem Description

You are given an array of

positive

integers

nums

.

Alice and Bob are playing a game. In the game, Alice can choose

either

all single-digit numbers or all double-digit numbers from

nums

, and the rest of the numbers are given to Bob. Alice wins if the sum of her numbers is

strictly greater

than the sum of Bob's numbers.

Return

true

if Alice can win this game, otherwise, return

false

.

Example 1:

Input:

nums = [1,2,3,4,10]

Output:

false

Explanation:

Alice cannot win by choosing either single-digit or double-digit numbers.

Example 2:

Input:

nums = [1,2,3,4,5,14]

Output:

true

Explanation:

Alice can win by choosing single-digit numbers which have a sum equal to 15.

Example 3:

Input:

nums = [5,5,5,25]

Output:

true

Explanation:

Alice can win by choosing double-digit numbers which have a sum equal to 25.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 99

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean canAliceWin(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function canAliceWin(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CanAliceWin(int[] nums) {

}
}
```

C:

```
bool canAliceWin(int* nums, int numsSize) {

}
```

Go:

```
func canAliceWin(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun canAliceWin(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool canAliceWin(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (can-alice-win nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find if Digit Game Can Be Won
* Difficulty: Easy
* Tags: array, math
*
* 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:
bool canAliceWin(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find if Digit Game Can Be Won
* Difficulty: Easy
* Tags: array, math
*
* 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 boolean canAliceWin(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find if Digit Game Can Be Won
Difficulty: Easy
Tags: array, math

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 canAliceWin(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find if Digit Game Can Be Won
* Difficulty: Easy
* Tags: array, math
*
* 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 {boolean}
*/
var canAliceWin = function(nums) {
};
```

TypeScript Solution:

```
/**
* Problem: Find if Digit Game Can Be Won
* Difficulty: Easy
* Tags: array, math
*
* 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 canAliceWin(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Find if Digit Game Can Be Won
* Difficulty: Easy
* Tags: array, math
*
* 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 bool CanAliceWin(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find if Digit Game Can Be Won
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

bool canAliceWin(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find if Digit Game Can Be Won
// Difficulty: Easy
// Tags: array, math
//
// 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 canAliceWin(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canAliceWin(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find if Digit Game Can Be Won
// Difficulty: Easy
// Tags: array, math
//
// 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 can_alice_win(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool canAliceWin(List<int> nums) {
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-alice-win nums)
(-> (listof exact-integer?) boolean?)
)
```