Problem 2815: Max Pair Sum in an Array

Problem Information

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

Problem Description

You are given an integer array

nums

. You have to find the

maximum

sum of a pair of numbers from

nums

such that the

largest digit

in both numbers is equal.

For example, 2373 is made up of three distinct digits: 2, 3, and 7, where 7 is the largest among them.

Return the

maximum

sum or -1 if no such pair exists.

Example 1:

Input:

nums = [112,131,411]

Output:

-1

Explanation:

Each numbers largest digit in order is [2,3,4].

Example 2:

Input:

nums = [2536,1613,3366,162]

Output:

5902

Explanation:

All the numbers have 6 as their largest digit, so the answer is

2536 + 3366 = 5902.

Example 3:

Input:

nums = [51,71,17,24,42]

Output:

88

Explanation:

Each number's largest digit in order is [5,7,7,4,4].

So we have only two possible pairs, 71 + 17 = 88 and 24 + 42 = 66.

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
```

```
function maxSum($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Max Pair Sum in an Array
* Difficulty: Easy
* Tags: array, 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 maxSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Max Pair Sum in an Array
* Difficulty: Easy
* Tags: array, 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 maxSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Max Pair Sum in an Array
Difficulty: Easy
Tags: array, 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 maxSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Max Pair Sum in an Array
* Difficulty: Easy
* Tags: array, 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[]} nums
* @return {number}
*/
var maxSum = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Max Pair Sum in an Array
* Difficulty: Easy
* Tags: array, 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 maxSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Max Pair Sum in an Array
* Difficulty: Easy
* Tags: array, 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 MaxSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Max Pair Sum in an Array
* Difficulty: Easy
```

```
* Tags: array, 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 maxSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Max Pair Sum in an Array
// Difficulty: Easy
// Tags: array, 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 maxSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Max Pair Sum in an Array
// Difficulty: Easy
// Tags: array, 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 max_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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