Problem 1073: Adding Two Negabinary Numbers

Problem Information

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

Problem Description

Given two numbers

arr1

and

arr2

in base

-2

, return the result of adding them together.

Each number is given in

array format

: as an array of 0s and 1s, from most significant bit to least significant bit. For example,

arr = [1,1,0,1]

represents the number

(-2)^3 + (-2)^2 + (-2)^0 = -3

. A number

arr

in

array, format

is also guaranteed to have no leading zeros: either

arr == [0]

or

arr[0] == 1

.

Return the result of adding

arr1

and

arr2

in the same format: as an array of 0s and 1s with no leading zeros.

Example 1:

Input:

arr1 = [1,1,1,1,1], arr2 = [1,0,1]

Output:

[1,0,0,0,0]

Explanation:

arr1 represents 11, arr2 represents 5, the output represents 16.

Example 2:

Input:

arr1 = [0], arr2 = [0]

Output:

[0]

Example 3:

Input:

arr1 = [0], arr2 = [1]

Output:

[1]

Constraints:

1 <= arr1.length, arr2.length <= 1000

arr1[i]

and

arr2[i]

are

0

or

1

arr1

and

arr2

have no leading zeros

Code Snippets

C++:

```
class Solution {
public:
vector<int> addNegabinary(vector<int>& arr1, vector<int>& arr2) {

}
};
```

Java:

```
class Solution {
public int[] addNegabinary(int[] arr1, int[] arr2) {

}
}
```

Python3:

```
class Solution:
def addNegabinary(self, arr1: List[int], arr2: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def addNegabinary(self, arr1, arr2):
"""
:type arr1: List[int]
```

```
:type arr2: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} arr1
* @param {number[]} arr2
* @return {number[]}
*/
var addNegabinary = function(arr1, arr2) {

};
```

TypeScript:

```
function addNegabinary(arr1: number[], arr2: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] AddNegabinary(int[] arr1, int[] arr2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* addNegabinary(int* arr1, int arr1Size, int* arr2, int arr2Size, int*
returnSize) {

}
```

Go:

```
func addNegabinary(arr1 []int, arr2 []int) []int {

}
```

Kotlin:

```
class Solution {
fun addNegabinary(arr1: IntArray, arr2: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func addNegabinary(_ arr1: [Int], _ arr2: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn add_negabinary(arr1: Vec<i32>, arr2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer[]}
def add_negabinary(arr1, arr2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer[]
*/
function addNegabinary($arr1, $arr2) {

}
}
```

Dart:

```
class Solution {
List<int> addNegabinary(List<int> arr1, List<int> arr2) {

}
}
```

Scala:

```
object Solution {
def addNegabinary(arr1: Array[Int], arr2: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec add_negabinary(arr1 :: [integer], arr2 :: [integer]) :: [integer]
def add_negabinary(arr1, arr2) do

end
end
```

Erlang:

```
-spec add_negabinary(Arr1 :: [integer()], Arr2 :: [integer()]) ->
[integer()].
add_negabinary(Arr1, Arr2) ->
.
```

Racket:

```
(define/contract (add-negabinary arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Adding Two Negabinary Numbers
* Difficulty: Medium
* 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:
vector<int> addNegabinary(vector<int>& arr1, vector<int>& arr2) {

}
};
```

Java Solution:

```
/**
* Problem: Adding Two Negabinary Numbers
* Difficulty: Medium
* 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 int[] addNegabinary(int[] arr1, int[] arr2) {

}
}
```

Python3 Solution:

```
"""
Problem: Adding Two Negabinary Numbers
Difficulty: Medium
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 addNegabinary(self, arr1: List[int], arr2: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def addNegabinary(self, arr1, arr2):
"""
:type arr1: List[int]
:type arr2: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Adding Two Negabinary Numbers
* Difficulty: Medium
* 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[]} arr1
* @param {number[]} arr2
* @return {number[]}
*/
var addNegabinary = function(arr1, arr2) {

};
```

TypeScript Solution:

```
/**
* Problem: Adding Two Negabinary Numbers
* Difficulty: Medium
* 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 addNegabinary(arr1: number[], arr2: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Adding Two Negabinary Numbers
* Difficulty: Medium
* 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 int[] AddNegabinary(int[] arr1, int[] arr2) {

}

}
```

C Solution:

```
/*
* Problem: Adding Two Negabinary Numbers
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* addNegabinary(int* arr1, int arr1Size, int* arr2, int arr2Size, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Adding Two Negabinary Numbers
// Difficulty: Medium
// 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 addNegabinary(arr1 []int, arr2 []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun addNegabinary(arr1: IntArray, arr2: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func addNegabinary(_ arr1: [Int], _ arr2: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Adding Two Negabinary Numbers
// Difficulty: Medium
// 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 add_negabinary(arr1: Vec<i32>, arr2: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer[]}
def add_negabinary(arr1, arr2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer[]
*/
function addNegabinary($arr1, $arr2) {

}
}
```

Dart Solution:

```
class Solution {
List<int> addNegabinary(List<int> arr1, List<int> arr2) {

}
}
```

Scala Solution:

```
object Solution {
def addNegabinary(arr1: Array[Int], arr2: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec add_negabinary(arr1 :: [integer], arr2 :: [integer]) :: [integer]
def add_negabinary(arr1, arr2) do

end
end
```

Erlang Solution:

```
-spec add_negabinary(Arr1 :: [integer()], Arr2 :: [integer()]) ->
[integer()].
add_negabinary(Arr1, Arr2) ->
.
```

Racket Solution:

```
(define/contract (add-negabinary arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```