Problem 2505: Bitwise OR of All Subsequence Sums

Problem Information

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

Problem Description

Given an integer array

nums

, return

the value of the bitwise

OR

of the sum of all possible

subsequences

in the array

.

A

subsequence

is a sequence that can be derived from another sequence by removing zero or more elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [2,1,0,3]

Output:

7

Explanation:

All possible subsequence sums that we can have are: 0, 1, 2, 3, 4, 5, 6. And we have 0 OR 1 OR 2 OR 3 OR 4 OR 5 OR 6 = 7, so we return 7.

Example 2:

Input:

nums = [0,0,0]

Output:

0

Explanation:

0 is the only possible subsequence sum we can have, so we return 0.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long subsequenceSumOr(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long SubsequenceSumOr(int[] nums) {

}
}
```

C:

```
long long subsequenceSumOr(int* nums, int numsSize) {

}
```

Go:

```
func subsequenceSumOr(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun subsequenceSumOr(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn subsequence_sum_or(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Bitwise OR of All Subsequence Sums
* 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:
long long subsequenceSumOr(vector<int>& nums) {
}
};
```

Java Solution:

```
/**
* Problem: Bitwise OR of All Subsequence Sums
* 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 long subsequenceSumOr(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Bitwise OR of All Subsequence Sums
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 subsequenceSumOr(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Bitwise OR of All Subsequence Sums
* 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[]} nums
* @return {number}
*/
var subsequenceSumOr = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Bitwise OR of All Subsequence Sums
* 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 subsequenceSumOr(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Bitwise OR of All Subsequence Sums
* 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 long SubsequenceSumOr(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Bitwise OR of All Subsequence Sums
* 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
*/

long long subsequenceSumOr(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Bitwise OR of All Subsequence Sums
// 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 subsequenceSumOr(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun subsequenceSumOr(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Bitwise OR of All Subsequence Sums
// 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 subsequence_sum_or(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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