Problem 3592: Inverse Coin Change

Problem Information

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

Problem Description

You are given a

1-indexed

integer array

numWays

, where

numWays[i]

represents the number of ways to select a total amount

i

using an

infinite

supply of some

fixed

coin denominations. Each denomination is a

positive

integer with value

at most

numWays.length

.

However, the exact coin denominations have been

lost

. Your task is to recover the set of denominations that could have resulted in the given

numWays

array.

Return a

sorted

array containing

unique

integers which represents this set of denominations.

If no such set exists, return an

empty

array.

Example 1:

Input:

numWays = [0,1,0,2,0,3,0,4,0,5]

Output:

[2,4,6]

Explanation:

Amount

Number of ways

Explanation

1

0

There is no way to select coins with total value 1.

2

1

The only way is

[2]

.

3

0

There is no way to select coins with total value 3.

4

2

The ways are

[2, 2]

and

[4]

.

5

0

There is no way to select coins with total value 5.

6

3

The ways are

[2, 2, 2]

,

[2, 4]

, and

[6]

.

7

0

There is no way to select coins with total value 7.

8

4

The ways are

[2, 2, 2, 2]

,

[2, 2, 4]

,

[2, 6]

, and

[4, 4]

.

9

0

There is no way to select coins with total value 9.

10

5

The ways are

[2, 2, 2, 2, 2]

,

[2, 2, 2, 4]

,

[2, 4, 4]

,

[2, 2, 6]

, and

[4, 6]

.

Example 2:

Input:

numWays = [1,2,2,3,4]

Output:

[1,2,5]

Explanation:

Amount

Number of ways

Explanation

1

1

The only way is

[1]

.

2

2

The ways are

[1, 1]

and

[2]

.

3

2

The ways are

[1, 1, 1]

and

[1, 2]

.

4

3

The ways are

[1, 1, 1, 1]

,

[1, 1, 2]

, and

[2, 2]

.

5

4

The ways are

[1, 1, 1, 1, 1]

,

[1, 1, 1, 2]

,

[1, 2, 2]

, and

[5]

.

Example 3:

Input:

numWays = [1,2,3,4,15]

Output:

[]

Explanation:

No set of denomination satisfies this array.

Constraints:

1 <= numWays.length <= 100

0 <= numWays[i] <= 2 * 10

8

Code Snippets

C++:

```
class Solution {
public:
vector<int> findCoins(vector<int>& numWays) {

}
};
```

Java:

```
class Solution {
public List<Integer> findCoins(int[] numWays) {

}
}
```

Python3:

```
class Solution:
def findCoins(self, numWays: List[int]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findCoins(numWays: number[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindCoins(int[] numWays) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findCoins(int* numWays, int numWaysSize, int* returnSize) {
```

```
}
```

Go:

```
func findCoins(numWays []int) []int {

}
```

Kotlin:

```
class Solution {
fun findCoins(numWays: IntArray): List<Int> {

}
}
```

Swift:

```
class Solution {
func findCoins(_ numWays: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_coins(num_ways: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} num_ways
# @return {Integer[]}
def find_coins(num_ways)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $numWays
* @return Integer[]
*/
function findCoins($numWays) {

}
}
```

Dart:

```
class Solution {
List<int> findCoins(List<int> numWays) {

}
}
```

Scala:

```
object Solution {
def findCoins(numWays: Array[Int]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_coins(num_ways :: [integer]) :: [integer]
def find_coins(num_ways) do

end
end
```

Erlang:

```
-spec find_coins(NumWays :: [integer()]) -> [integer()].
find_coins(NumWays) ->
.
```

Racket:

```
(define/contract (find-coins numWays)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Inverse Coin Change
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> findCoins(vector<int>& numWays) {

}
};
```

Java Solution:

```
/**
* Problem: Inverse Coin Change
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public List<Integer> findCoins(int[] numWays) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Inverse Coin Change
Difficulty: Medium
Tags: array, dp, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findCoins(self, numWays: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Inverse Coin Change
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Inverse Coin Change
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findCoins(numWays: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Inverse Coin Change
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public IList<int> FindCoins(int[] numWays) {

}
}
```

C Solution:

```c
/*
* Problem: Inverse Coin Change
* Difficulty: Medium
* Tags: array, dp, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findCoins(int* numWays, int numWaysSize, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Inverse Coin Change
// Difficulty: Medium
// Tags: array, dp, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func findCoins(numWays []int) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun findCoins(numWays: IntArray): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func findCoins(_ numWays: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Inverse Coin Change
// Difficulty: Medium
// Tags: array, dp, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_coins(num_ways: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} num_ways
# @return {Integer[]}
def find_coins(num_ways)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $numWays
* @return Integer[]
*/
function findCoins($numWays) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findCoins(List<int> numWays) {

}
}
```

Scala Solution:

```
object Solution {
def findCoins(numWays: Array[Int]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_coins(num_ways :: [integer]) :: [integer]
def find_coins(num_ways) do

end
end
```

Erlang Solution:

```
-spec find_coins(NumWays :: [integer()]) -> [integer()].
find_coins(NumWays) ->
.
```

Racket Solution:

```
(define/contract (find-coins numWays)
(-> (listof exact-integer?) (listof exact-integer?))
)
```