Problem 322: Coin Change

Problem Information

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

Problem Description

You are given an integer array

coins

representing coins of different denominations and an integer

amount

representing a total amount of money.

Return

the fewest number of coins that you need to make up that amount

. If that amount of money cannot be made up by any combination of the coins, return

-1

.

You may assume that you have an infinite number of each kind of coin.

Example 1:

Input:

coins = [1,2,5], amount = 11

Output:

3

Explanation:

11 = 5 + 5 + 1

Example 2:

Input:

coins = [2], amount = 3

Output:

-1

Example 3:

Input:

coins = [1], amount = 0

Output:

0

Constraints:

1 <= coins.length <= 12

1 <= coins[i] <= 2

31

- 1

0 <= amount <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int coinChange(vector<int>& coins, int amount) {

}
};
```

Java:

```
class Solution {
public int coinChange(int[] coins, int amount) {

}
}
```

Python3:

```
class Solution:
def coinChange(self, coins: List[int], amount: int) -> int:
```

Python:

```
class Solution(object):
def coinChange(self, coins, amount):
"""
:type coins: List[int]
:type amount: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} coins
* @param {number} amount
* @return {number}
*/
var coinChange = function(coins, amount) {

};
```

TypeScript:

```
function coinChange(coins: number[], amount: number): number {

};
```

C#:

```
public class Solution {
public int CoinChange(int[] coins, int amount) {

}
}
```

C:

```
int coinChange(int* coins, int coinsSize, int amount) {

}
```

Go:

```
func coinChange(coins []int, amount int) int {

}
```

Kotlin:

```
class Solution {
fun coinChange(coins: IntArray, amount: Int): Int {

}
}
```

Swift:

```
class Solution {
func coinChange(_ coins: [Int], _ amount: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn coin_change(coins: Vec<i32>, amount: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} coins
# @param {Integer} amount
# @return {Integer}
def coin_change(coins, amount)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $coins
* @param Integer $amount
* @return Integer
*/
function coinChange($coins, $amount) {

}
}
```

Dart:

```
class Solution {
int coinChange(List<int> coins, int amount) {



}
}
```

Scala:

```
object Solution {
def coinChange(coins: Array[Int], amount: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec coin_change(coins :: [integer], amount :: integer) :: integer
def coin_change(coins, amount) do

end
end
```

Erlang:

```
-spec coin_change(Coins :: [integer()], Amount :: integer()) -> integer().
coin_change(Coins, Amount) ->
.
```

Racket:

```
(define/contract (coin-change coins amount)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Coin Change
* Difficulty: Medium
* Tags: array, dp, search
*
* 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:
int coinChange(vector<int>& coins, int amount) {

}
};
```

Java Solution:

```
/**
* Problem: Coin Change
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 int coinChange(int[] coins, int amount) {

}
}
```

Python3 Solution:

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

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 coinChange(self, coins: List[int], amount: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def coinChange(self, coins, amount):
"""
:type coins: List[int]
:type amount: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Coin Change
* Difficulty: Medium
* Tags: array, dp, search
*
* 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[]} coins
* @param {number} amount
* @return {number}
*/
var coinChange = function(coins, amount) {

};
```

TypeScript Solution:

```
/**
* Problem: Coin Change
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 coinChange(coins: number[], amount: number): number {

};
```

C# Solution:

```
/*
* Problem: Coin Change
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 int CoinChange(int[] coins, int amount) {

}
}
```

C Solution:

```
/*
* Problem: Coin Change
* Difficulty: Medium
* Tags: array, dp, search
*
* 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
```

```
*/

int coinChange(int* coins, int coinsSize, int amount) {

}
```

Go Solution:

```
// Problem: Coin Change
// Difficulty: Medium
// Tags: array, dp, search
//
// 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 coinChange(coins []int, amount int) int {

}
```

Kotlin Solution:

```
class Solution {
fun coinChange(coins: IntArray, amount: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func coinChange(_ coins: [Int], _ amount: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Coin Change
// Difficulty: Medium
// Tags: array, dp, search
```

```
//
// 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 coin_change(coins: Vec<i32>, amount: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} coins
# @param {Integer} amount
# @return {Integer}
def coin_change(coins, amount)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $coins
* @param Integer $amount
* @return Integer
*/
function coinChange($coins, $amount) {

}
}
```

Dart Solution:

```
class Solution {
int coinChange(List<int> coins, int amount) {

}
}
```

Scala Solution:

```
object Solution {
def coinChange(coins: Array[Int], amount: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec coin_change(coins :: [integer], amount :: integer) :: integer
def coin_change(coins, amount) do

end
end
```

Erlang Solution:

```
-spec coin_change(Coins :: [integer()], Amount :: integer()) -> integer().
coin_change(Coins, Amount) ->
.
```

Racket Solution:

```
(define/contract (coin-change coins amount)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```