

# Problem 2952: Minimum Number of Coins to be Added

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given a

0-indexed

integer array

coins

, representing the values of the coins available, and an integer

target

.

An integer

x

is

obtainable

if there exists a subsequence of

coins

that sums to

x

.

Return

the

minimum

number of coins

of any value

that need to be added to the array so that every integer in the range

[1, target]

is

obtainable

.

A

subsequence

of an array is a new

non-empty

array that is formed from the original array by deleting some (

possibly none

) of the elements without disturbing the relative positions of the remaining elements.

Example 1:

Input:

coins = [1,4,10], target = 19

Output:

2

Explanation:

We need to add coins 2 and 8. The resulting array will be [1,2,4,8,10]. It can be shown that all integers from 1 to 19 are obtainable from the resulting array, and that 2 is the minimum number of coins that need to be added to the array.

Example 2:

Input:

coins = [1,4,10,5,7,19], target = 19

Output:

1

Explanation:

We only need to add the coin 2. The resulting array will be [1,2,4,5,7,10,19]. It can be shown that all integers from 1 to 19 are obtainable from the resulting array, and that 1 is the minimum number of coins that need to be added to the array.

Example 3:

Input:

coins = [1,1,1], target = 20

Output:

3

Explanation:

We need to add coins 4, 8, and 16. The resulting array will be [1,1,1,4,8,16]. It can be shown that all integers from 1 to 20 are obtainable from the resulting array, and that 3 is the minimum number of coins that need to be added to the array.

Constraints:

$1 \leq \text{target} \leq 10$

5

$1 \leq \text{coins.length} \leq 10$

5

$1 \leq \text{coins[i]} \leq \text{target}$

## Code Snippets

C++:

```
class Solution {
public:
    int minimumAddedCoins(vector<int>& coins, int target) {
        }
};
```

Java:

```
class Solution {
public int minimumAddedCoins(int[] coins, int target) {
    }
```

```
}
```

### Python3:

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

### Python:

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

### JavaScript:

```
/**  
 * @param {number[]} coins  
 * @param {number} target  
 * @return {number}  
 */  
var minimumAddedCoins = function(coins, target) {  
  
};
```

### TypeScript:

```
function minimumAddedCoins(coins: number[], target: number): number {  
  
};
```

### C#:

```
public class Solution {  
    public int MinimumAddedCoins(int[] coins, int target) {  
  
    }  
}
```

**C:**

```
int minimumAddedCoins(int* coins, int coinsSize, int target) {  
  
}
```

**Go:**

```
func minimumAddedCoins(coins []int, target int) int {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun minimumAddedCoins(coins: IntArray, target: Int): Int {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func minimumAddedCoins(_ coins: [Int], _ target: Int) -> Int {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn minimum_added_coins(coins: Vec<i32>, target: i32) -> i32 {  
  
    }  
}
```

**Ruby:**

```
# @param {Integer[]} coins  
# @param {Integer} target  
# @return {Integer}  
def minimum_added_coins(coins, target)
```

```
end
```

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $coins  
     * @param Integer $target  
     * @return Integer  
     */  
    function minimumAddedCoins($coins, $target) {  
  
    }  
}
```

### Dart:

```
class Solution {  
int minimumAddedCoins(List<int> coins, int target) {  
  
}  
}
```

### Scala:

```
object Solution {  
def minimumAddedCoins(coins: Array[Int], target: Int): Int = {  
  
}  
}
```

### Elixir:

```
defmodule Solution do  
@spec minimum_added_coins(coins :: [integer], target :: integer) :: integer  
def minimum_added_coins(coins, target) do  
  
end  
end
```

### Erlang:

```

-spec minimum_added_coins(Coins :: [integer()], Target :: integer()) ->
    integer().
minimum_added_coins(Coins, Target) ->
    .

```

### Racket:

```

(define/contract (minimum-added-coins coins target)
  (-> (listof exact-integer?) exact-integer? exact-integer?))

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Minimum Number of Coins to be Added
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 minimumAddedCoins(vector<int>& coins, int target) {
        }
    };

```

### Java Solution:

```

/**
 * Problem: Minimum Number of Coins to be Added
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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 minimumAddedCoins(int[] coins, int target) {
        }
    }
}

```

### Python3 Solution:

```

"""
Problem: Minimum Number of Coins to be Added
Difficulty: Medium
Tags: array, greedy, sort

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 minimumAddedCoins(self, coins: List[int], target: int) -> int:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def minimumAddedCoins(self, coins, target):
        """
        :type coins: List[int]
        :type target: int
        :rtype: int
        """

```

### JavaScript Solution:

```

/**
 * Problem: Minimum Number of Coins to be Added
 * Difficulty: Medium
 */

```

```

* Tags: array, greedy, sort
*
* 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[]} coins
* @param {number} target
* @return {number}
*/
var minimumAddedCoins = function(coins, target) {

};

```

### TypeScript Solution:

```

/**

* Problem: Minimum Number of Coins to be Added
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minimumAddedCoins(coins: number[], target: number): number {
};


```

### C# Solution:

```

/*
* Problem: Minimum Number of Coins to be Added
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 MinimumAddedCoins(int[] coins, int target) {
        }
    }
}

```

### C Solution:

```

/*
 * Problem: Minimum Number of Coins to be Added
 * Difficulty: Medium
 * Tags: array, greedy, sort
 *
 * 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
*/

int minimumAddedCoins(int* coins, int coinsSize, int target) {
}

```

### Go Solution:

```

// Problem: Minimum Number of Coins to be Added
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 minimumAddedCoins(coins []int, target int) int {
}

```

### Kotlin Solution:

```
class Solution {  
    fun minimumAddedCoins(coins: IntArray, target: Int): Int {  
        }  
        }  
}
```

### Swift Solution:

```
class Solution {  
    func minimumAddedCoins(_ coins: [Int], _ target: Int) -> Int {  
        }  
        }  
}
```

### Rust Solution:

```
// Problem: Minimum Number of Coins to be Added  
// Difficulty: Medium  
// Tags: array, greedy, sort  
//  
// 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 minimum_added_coins(coins: Vec<i32>, target: i32) -> i32 {  
        }  
        }  
}
```

### Ruby Solution:

```
# @param {Integer[]} coins  
# @param {Integer} target  
# @return {Integer}  
def minimum_added_coins(coins, target)  
  
end
```

### PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $coins
     * @param Integer $target
     * @return Integer
     */
    function minimumAddedCoins($coins, $target) {

    }
}

```

### Dart Solution:

```

class Solution {
    int minimumAddedCoins(List<int> coins, int target) {
        return 0;
    }
}

```

### Scala Solution:

```

object Solution {
    def minimumAddedCoins(coins: Array[Int], target: Int): Int = {
        0
    }
}

```

### Elixir Solution:

```

defmodule Solution do
  @spec minimum_added_coins(coins :: [integer], target :: integer) :: integer
  def minimum_added_coins(coins, target) do
    end
  end
end

```

### Erlang Solution:

```

-spec minimum_added_coins(Coins :: [integer()], Target :: integer()) ->
    integer().
minimum_added_coins(Coins, Target) ->

```

**Racket Solution:**

```
(define/contract (minimum-added-coins coins target)
  (-> (listof exact-integer?) exact-integer? exact-integer?))
```