Problem 1833: Maximum Ice Cream Bars

Problem Information

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

Problem Description

It is a sweltering summer day, and a boy wants to buy some ice cream bars.

At the store, there are

n

ice cream bars. You are given an array

costs

of length

n

, where

costs[i]

is the price of the

i

th

ice cream bar in coins. The boy initially has

coins

coins to spend, and he wants to buy as many ice cream bars as possible.

Note:

The boy can buy the ice cream bars in any order.

Return

the

maximum

number of ice cream bars the boy can buy with

coins

coins.

You must solve the problem by counting sort.

Example 1:

Input:

costs = [1,3,2,4,1], coins = 7

Output:

4

Explanation:

The boy can buy ice cream bars at indices 0,1,2,4 for a total price of 1 + 3 + 2 + 1 = 7.

Example 2:

Input:

costs = [10,6,8,7,7,8], coins = 5

Output:

0

Explanation:

The boy cannot afford any of the ice cream bars.

Example 3:

Input:

costs = [1,6,3,1,2,5], coins = 20

Output:

6

Explanation:

The boy can buy all the ice cream bars for a total price of 1 + 6 + 3 + 1 + 2 + 5 = 18.

Constraints:

costs.length == n

1 <= n <= 10

5

1 <= costs[i] <= 10

5

1 <= coins <= 10

Code Snippets

C++:

```
class Solution {
public:
int maxIceCream(vector<int>& costs, int coins) {

}
};
```

Java:

```
class Solution {
public int maxIceCream(int[] costs, int coins) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} costs
* @param {number} coins
```

```
 * @return {number}
 */
var maxIceCream = function(costs, coins) {

};
```

TypeScript:

```
function maxIceCream(costs: number[], coins: number): number {

};
```

C#:

```
public class Solution {
public int MaxIceCream(int[] costs, int coins) {

}
}
```

C:

```
int maxIceCream(int* costs, int costsSize, int coins) {

}
```

Go:

```
func maxIceCream(costs []int, coins int) int {

}
```

Kotlin:

```
class Solution {
fun maxIceCream(costs: IntArray, coins: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} costs
# @param {Integer} coins
# @return {Integer}
def max_ice_cream(costs, coins)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $costs
* @param Integer $coins
* @return Integer
*/
function maxIceCream($costs, $coins) {

}
}
```

Dart:

```
class Solution {
int maxIceCream(List<int> costs, int coins) {

}
```

```
}
```

Scala:

```
object Solution {
def maxIceCream(costs: Array[Int], coins: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_ice_cream(costs :: [integer], coins :: integer) :: integer
def max_ice_cream(costs, coins) do

end
end
```

Erlang:

```
-spec max_ice_cream(Costs :: [integer()], Coins :: integer()) -> integer().
max_ice_cream(Costs, Coins) ->
.
```

Racket:

```
(define/contract (max-ice-cream costs coins)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Ice Cream Bars
* 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 maxIceCream(vector<int>& costs, int coins) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Ice Cream Bars
* 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 maxIceCream(int[] costs, int coins) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Ice Cream Bars
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 maxIceCream(self, costs: List[int], coins: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Ice Cream Bars
* 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[]} costs
* @param {number} coins
* @return {number}
*/
var maxIceCream = function(costs, coins) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Ice Cream Bars
```

```
* 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 maxIceCream(costs: number[], coins: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Ice Cream Bars
* 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 MaxIceCream(int[] costs, int coins) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Ice Cream Bars
* 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 maxIceCream(int* costs, int costsSize, int coins) {

}
```

Go Solution:

```
// Problem: Maximum Ice Cream Bars
// 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 maxIceCream(costs []int, coins int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxIceCream(costs: IntArray, coins: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Ice Cream Bars
// 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 max_ice_cream(costs: Vec<i32>, coins: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} costs
# @param {Integer} coins
# @return {Integer}
def max_ice_cream(costs, coins)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $costs
* @param Integer $coins
* @return Integer
*/
function maxIceCream($costs, $coins) {

}
}
```

Dart Solution:

```
class Solution {
int maxIceCream(List<int> costs, int coins) {

}
}
```

Scala Solution:

```
object Solution {
def maxIceCream(costs: Array[Int], coins: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_ice_cream(costs :: [integer], coins :: integer) :: integer
def max_ice_cream(costs, coins) do

end
end
```

Erlang Solution:

```
-spec max_ice_cream(Costs :: [integer()], Coins :: integer()) -> integer().
max_ice_cream(Costs, Coins) ->
.
```

Racket Solution:

```
(define/contract (max-ice-cream costs coins)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```