Problem 2944: Minimum Number of Coins for Fruits

Problem Information

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

Problem Description

You are given an

0-indexed

integer array

prices

where

prices[i]

denotes the number of coins needed to purchase the

(i + 1)

th

fruit.

The fruit market has the following reward for each fruit:

If you purchase the

(i + 1)

th

fruit at

prices[i]

coins, you can get any number of the next

i

fruits for free.

Note

that even if you

can

take fruit

j

for free, you can still purchase it for

prices[j - 1]

coins to receive its reward.

Return the

minimum

number of coins needed to acquire all the fruits.

Example 1:

Input:

prices = [3,1,2]

Output:

4

Explanation:

Purchase the 1

st

fruit with

prices[0] = 3

coins, you are allowed to take the 2

nd

fruit for free.

Purchase the 2

nd

fruit with

prices[1] = 1

coin, you are allowed to take the 3

rd

fruit for free.

Take the 3

rd

fruit for free.

Note that even though you could take the 2

nd

fruit for free as a reward of buying 1

st

fruit, you purchase it to receive its reward, which is more optimal.

Example 2:

Input:

prices = [1,10,1,1]

Output:

2

Explanation:

Purchase the 1

st

fruit with

prices[0] = 1

coin, you are allowed to take the 2

nd

fruit for free.

Take the 2

nd

fruit for free.

Purchase the 3

rd

fruit for

prices[2] = 1

coin, you are allowed to take the 4

th

fruit for free.

Take the 4

t

h

fruit for free.

Example 3:

Input:

prices = [26,18,6,12,49,7,45,45]

Output:

39

Explanation:

Purchase the 1

st

fruit with

prices[0] = 26

coin, you are allowed to take the 2

nd

fruit for free.

Take the 2

nd

fruit for free.

Purchase the 3

rd

fruit for

prices[2] = 6

coin, you are allowed to take the 4

th

, 5

th

and 6

th

(the next three) fruits for free.

Take the 4

t

h

fruit for free.

Take the 5

t

h

fruit for free.

Purchase the 6

th

fruit with

prices[5] = 7

coin, you are allowed to take the 8

th

and 9

th

fruit for free.

Take the 7

t

h

fruit for free.

Take the 8

t

h

fruit for free.

Note that even though you could take the 6

th

fruit for free as a reward of buying 3

rd

fruit, you purchase it to receive its reward, which is more optimal.

Constraints:

1 <= prices.length <= 1000

1 <= prices[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
```

```
int minimumCoins(vector<int>& prices) {

}
};
```

Java:

```
class Solution {
public int minimumCoins(int[] prices) {

}
}
```

Python3:

```
class Solution:
def minimumCoins(self, prices: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumCoins(prices: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumCoins(int[] prices) {

}
}
```

C:

```
int minimumCoins(int* prices, int pricesSize) {

}
```

Go:

```
func minimumCoins(prices []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumCoins(prices: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumCoins(_ prices: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_coins(prices: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} prices
# @return {Integer}
def minimum_coins(prices)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $prices
* @return Integer
*/
function minimumCoins($prices) {

}
}
```

Dart:

```
class Solution {
int minimumCoins(List<int> prices) {

}
}
```

Scala:

```
object Solution {
def minimumCoins(prices: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_coins(prices :: [integer]) :: integer
def minimum_coins(prices) do
```

```
end
end
```

Erlang:

```
-spec minimum_coins(Prices :: [integer()]) -> integer().
minimum_coins(Prices) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Coins for Fruits
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* 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 minimumCoins(vector<int>& prices) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Coins for Fruits
```

```
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* 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 minimumCoins(int[] prices) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Coins for Fruits
Difficulty: Medium
Tags: array, dp, queue, heap

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 minimumCoins(self, prices: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Coins for Fruits
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* 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[]} prices
* @return {number}
*/
var minimumCoins = function(prices) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Coins for Fruits
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* 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 minimumCoins(prices: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Coins for Fruits
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* 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 MinimumCoins(int[] prices) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Coins for Fruits
* Difficulty: Medium
* Tags: array, dp, queue, heap
*
* 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 minimumCoins(int* prices, int pricesSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Coins for Fruits
// Difficulty: Medium
// Tags: array, dp, queue, heap
//
// 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 minimumCoins(prices []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCoins(prices: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumCoins(_ prices: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Coins for Fruits
// Difficulty: Medium
// Tags: array, dp, queue, heap
//
// 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 minimum_coins(prices: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} prices
# @return {Integer}
def minimum_coins(prices)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $prices
* @return Integer
*/
function minimumCoins($prices) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCoins(List<int> prices) {

}
}
```

Scala Solution:

```
object Solution {
def minimumCoins(prices: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_coins(prices :: [integer]) :: integer
def minimum_coins(prices) do

end
end
```

Erlang Solution:

```
-spec minimum_coins(Prices :: [integer()]) -> integer().
minimum_coins(Prices) ->
.
```

Racket Solution:

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