

Problem 3180: Maximum Total Reward Using Operations I

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

rewardValues

of length

n

, representing the values of rewards.

Initially, your total reward

x

is 0, and all indices are

unmarked

. You are allowed to perform the following operation

any

number of times:

Choose an

unmarked

index

i

from the range

[0, n - 1]

.

If

rewardValues[i]

is

greater

than your current total reward

x

, then add

rewardValues[i]

to

x

(i.e.,

$x = x + \text{rewardValues}[i]$

), and

mark

the index

i

Return an integer denoting the

maximum

total reward

you can collect by performing the operations optimally.

Example 1:

Input:

rewardValues = [1,1,3,3]

Output:

4

Explanation:

During the operations, we can choose to mark the indices 0 and 2 in order, and the total reward will be 4, which is the maximum.

Example 2:

Input:

rewardValues = [1,6,4,3,2]

Output:

Explanation:

Mark the indices 0, 2, and 1 in order. The total reward will then be 11, which is the maximum.

Constraints:

$1 \leq \text{rewardValues.length} \leq 2000$

$1 \leq \text{rewardValues}[i] \leq 2000$

Code Snippets

C++:

```
class Solution {
public:
    int maxTotalReward(vector<int>& rewardValues) {
        }
};
```

Java:

```
class Solution {
    public int maxTotalReward(int[] rewardValues) {
        }
}
```

Python3:

```
class Solution:
    def maxTotalReward(self, rewardValues: List[int]) -> int:
```

Python:

```
class Solution(object):
    def maxTotalReward(self, rewardValues):
```

```
"""
:type rewardValues: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxTotalReward(rewardValues: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxTotalReward(int[] rewardValues) {

}
```

C:

```
int maxTotalReward(int* rewardValues, int rewardValuesSize) {

}
```

Go:

```
func maxTotalReward(rewardValues []int) int {

}
```

Kotlin:

```
class Solution {  
    fun maxTotalReward(rewardValues: IntArray): Int {  
        }  
    }  
}
```

Swift:

```
class Solution {  
    func maxTotalReward(_ rewardValues: [Int]) -> Int {  
        }  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn max_total_reward(reward_values: Vec<i32>) -> i32 {  
        }  
    }  
}
```

Ruby:

```
# @param {Integer[]} reward_values  
# @return {Integer}  
def max_total_reward(reward_values)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $rewardValues  
     * @return Integer  
     */  
    function maxTotalReward($rewardValues) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int maxTotalReward(List<int> rewardValues) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def maxTotalReward(rewardValues: Array[Int]): Int = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec max_total_reward(reward_values :: [integer]) :: integer  
    def max_total_reward(reward_values) do  
  
    end  
end
```

Erlang:

```
-spec max_total_reward(RewardValues :: [integer()]) -> integer().  
max_total_reward(RewardValues) ->  
.
```

Racket:

```
(define/contract (max-total-reward rewardValues)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Maximum Total Reward Using Operations I
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 maxTotalReward(vector<int>& rewardValues) {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximum Total Reward Using Operations I
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 maxTotalReward(int[] rewardValues) {

}
}

```

Python3 Solution:

```

"""
Problem: Maximum Total Reward Using Operations I
Difficulty: Medium
Tags: array, dp

```

```

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 maxTotalReward(self, rewardValues: List[int]) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def maxTotalReward(self, rewardValues):
"""
:type rewardValues: List[int]
:rtype: int
"""

```

JavaScript Solution:

```

/**
 * Problem: Maximum Total Reward Using Operations I
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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[]} rewardValues
 * @return {number}
 */
var maxTotalReward = function(rewardValues) {

};


```

TypeScript Solution:

```

/**
 * Problem: Maximum Total Reward Using Operations I
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 maxTotalReward(rewardValues: number[]): number {
}

```

C# Solution:

```

/*
 * Problem: Maximum Total Reward Using Operations I
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 MaxTotalReward(int[] rewardValues) {
        return 0;
    }
}

```

C Solution:

```

/*
 * Problem: Maximum Total Reward Using Operations I
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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
 */

```

```
*/\n\nint maxTotalReward(int* rewardValues, int rewardValuesSize) {\n\n}
```

Go Solution:

```
// Problem: Maximum Total Reward Using Operations I\n// Difficulty: Medium\n// Tags: array, dp\n//\n// Approach: Use two pointers or sliding window technique\n// Time Complexity: O(n) or O(n log n)\n// Space Complexity: O(n) or O(n * m) for DP table\n\nfunc maxTotalReward(rewardValues []int) int {\n\n}
```

Kotlin Solution:

```
class Solution {\n    fun maxTotalReward(rewardValues: IntArray): Int {\n\n    }\n}
```

Swift Solution:

```
class Solution {\n    func maxTotalReward(_ rewardValues: [Int]) -> Int {\n\n    }\n}
```

Rust Solution:

```
// Problem: Maximum Total Reward Using Operations I\n// Difficulty: Medium\n// Tags: array, dp
```

```

// 
// 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 max_total_reward(reward_values: Vec<i32>) -> i32 {

}
}

```

Ruby Solution:

```

# @param {Integer[]} reward_values
# @return {Integer}
def max_total_reward(reward_values)

end

```

PHP Solution:

```

class Solution {

/**
 * @param Integer[] $rewardValues
 * @return Integer
 */
function maxTotalReward($rewardValues) {

}
}

```

Dart Solution:

```

class Solution {
int maxTotalReward(List<int> rewardValues) {

}
}

```

Scala Solution:

```
object Solution {  
    def maxTotalReward(rewardValues: Array[Int]): Int = {  
        }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec max_total_reward(reward_values :: [integer]) :: integer  
  def max_total_reward(reward_values) do  
  
  end  
end
```

Erlang Solution:

```
-spec max_total_reward(RewardValues :: [integer()]) -> integer().  
max_total_reward(RewardValues) ->  
.
```

Racket Solution:

```
(define/contract (max-total-reward rewardValues)  
  (-> (listof exact-integer?) exact-integer?)  
)
```