

Problem 3207: Maximum Points After Enemy Battles

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

`enemyEnergies`

denoting the energy values of various enemies.

You are also given an integer

`currentEnergy`

denoting the amount of energy you have initially.

You start with 0 points, and all the enemies are unmarked initially.

You can perform

either

of the following operations

zero

or multiple times to gain points:

Choose an

unmarked

enemy,

i

, such that

$\text{currentEnergy} \geq \text{enemyEnergies}[i]$

. By choosing this option:

You gain 1 point.

Your energy is reduced by the enemy's energy, i.e.

$\text{currentEnergy} = \text{currentEnergy} - \text{enemyEnergies}[i]$

.

If you have

at least

1 point, you can choose an

unmarked

enemy,

i

. By choosing this option:

Your energy increases by the enemy's energy, i.e.

$\text{currentEnergy} = \text{currentEnergy} + \text{enemyEnergies}[i]$

.

The

e

nemy

i

is

marked

.

Return an integer denoting the

maximum

points you can get in the end by optimally performing operations.

Example 1:

Input:

enemyEnergies = [3,2,2], currentEnergy = 2

Output:

3

Explanation:

The following operations can be performed to get 3 points, which is the maximum:

First operation on enemy 1:

points

increases by 1, and

currentEnergy

decreases by 2. So,

points = 1

, and

currentEnergy = 0

.

Second operation on enemy 0:

currentEnergy

increases by 3, and enemy 0 is marked. So,

points = 1

,

currentEnergy = 3

, and marked enemies =

[0]

.

First operation on enemy 2:

points

increases by 1, and

currentEnergy

decreases by 2. So,

points = 2

,

currentEnergy = 1

, and marked enemies =

[0]

.

Second operation on enemy 2:

currentEnergy

increases by 2, and enemy 2 is marked. So,

points = 2

,

currentEnergy = 3

, and marked enemies =

[0, 2]

.

First operation on enemy 1:

points

increases by 1, and

currentEnergy

decreases by 2. So,

points = 3

,

currentEnergy = 1

, and marked enemies =

[0, 2]

.

Example 2:

Input:

enemyEnergies =

[2]

, currentEnergy = 10

Output:

5

Explanation:

Performing the first operation 5 times on enemy 0 results in the maximum number of points.

Constraints:

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

5

1 <= enemyEnergies[i] <= 10

9

0 <= currentEnergy <= 10

9

Code Snippets

C++:

```
class Solution {  
public:  
    long long maximumPoints(vector<int>& enemyEnergies, int currentEnergy) {  
  
    }  
};
```

Java:

```
class Solution {  
    public long maximumPoints(int[] enemyEnergies, int currentEnergy) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def maximumPoints(self, enemyEnergies: List[int], currentEnergy: int) -> int:
```

Python:

```
class Solution(object):  
    def maximumPoints(self, enemyEnergies, currentEnergy):  
        """  
        :type enemyEnergies: List[int]
```

```
:type currentEnergy: int
:rtype: int
"""
```

JavaScript:

```
/**
 * @param {number[]} enemyEnergies
 * @param {number} currentEnergy
 * @return {number}
 */
var maximumPoints = function(enemyEnergies, currentEnergy) {

};
```

TypeScript:

```
function maximumPoints(enemyEnergies: number[], currentEnergy: number):
number {

};
```

C#:

```
public class Solution {
    public long MaximumPoints(int[] enemyEnergies, int currentEnergy) {

    }
}
```

C:

```
long long maximumPoints(int* enemyEnergies, int enemyEnergiesSize, int
currentEnergy) {

}
```

Go:

```
func maximumPoints(enemyEnergies []int, currentEnergy int) int64 {

}
```

Kotlin:

```
class Solution {  
    fun maximumPoints(enemyEnergies: IntArray, currentEnergy: Int): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maximumPoints(_ enemyEnergies: [Int], _ currentEnergy: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn maximum_points(enemy_energies: Vec<i32>, current_energy: i32) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} enemy_energies  
# @param {Integer} current_energy  
# @return {Integer}  
def maximum_points(enemy_energies, current_energy)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $enemyEnergies  
     * @param Integer $currentEnergy  
     * @return Integer  
     */  
    function maximumPoints($enemyEnergies, $currentEnergy) {  
  
    }  
}
```

```
}  
}
```

Dart:

```
class Solution {  
  int maximumPoints(List<int> enemyEnergies, int currentEnergy) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def maximumPoints(enemyEnergies: Array[Int], currentEnergy: Int): Long = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec maximum_points(enemy_energies :: [integer], current_energy :: integer)  
    :: integer  
  def maximum_points(enemy_energies, current_energy) do  
  
  end  
end
```

Erlang:

```
-spec maximum_points(EnemyEnergies :: [integer()], CurrentEnergy ::  
integer()) -> integer().  
maximum_points(EnemyEnergies, CurrentEnergy) ->  
.
```

Racket:

```
(define/contract (maximum-points enemyEnergies currentEnergy)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Points After Enemy Battles
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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:
    long long maximumPoints(vector<int>& enemyEnergies, int currentEnergy) {

    }
};
```

Java Solution:

```
/**
 * Problem: Maximum Points After Enemy Battles
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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 long maximumPoints(int[] enemyEnergies, int currentEnergy) {

    }
}
```

Python3 Solution:

```

"""
Problem: Maximum Points After Enemy Battles
Difficulty: Medium
Tags: array, greedy

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 maximumPoints(self, enemyEnergies: List[int], currentEnergy: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def maximumPoints(self, enemyEnergies, currentEnergy):
        """
        :type enemyEnergies: List[int]
        :type currentEnergy: int
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Maximum Points After Enemy Battles
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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[]} enemyEnergies
 * @param {number} currentEnergy
 * @return {number}
 */

```

```
var maximumPoints = function(enemyEnergies, currentEnergy) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Maximum Points After Enemy Battles  
 * Difficulty: Medium  
 * Tags: array, greedy  
 *  
 * 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 maximumPoints(enemyEnergies: number[], currentEnergy: number):  
number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Maximum Points After Enemy Battles  
 * Difficulty: Medium  
 * Tags: array, greedy  
 *  
 * 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 long MaximumPoints(int[] enemyEnergies, int currentEnergy) {  
  
    }  
}
```

C Solution:

```

/*
 * Problem: Maximum Points After Enemy Battles
 * Difficulty: Medium
 * Tags: array, greedy
 *
 * 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
 */

long long maximumPoints(int* enemyEnergies, int enemyEnergiesSize, int
currentEnergy) {

}

```

Go Solution:

```

// Problem: Maximum Points After Enemy Battles
// Difficulty: Medium
// Tags: array, greedy
//
// 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 maximumPoints(enemyEnergies []int, currentEnergy int) int64 {

}

```

Kotlin Solution:

```

class Solution {
    fun maximumPoints(enemyEnergies: IntArray, currentEnergy: Int): Long {

    }
}

```

Swift Solution:

```

class Solution {
    func maximumPoints(_ enemyEnergies: [Int], _ currentEnergy: Int) -> Int {

```

```
}  
}
```

Rust Solution:

```
// Problem: Maximum Points After Enemy Battles  
// Difficulty: Medium  
// Tags: array, greedy  
//  
// 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 maximum_points(enemy_energies: Vec<i32>, current_energy: i32) -> i64 {  
  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} enemy_energies  
# @param {Integer} current_energy  
# @return {Integer}  
def maximum_points(enemy_energies, current_energy)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $enemyEnergies  
     * @param Integer $currentEnergy  
     * @return Integer  
     */  
    function maximumPoints($enemyEnergies, $currentEnergy) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
  int maximumPoints(List<int> enemyEnergies, int currentEnergy) {  
  
  }  
}
```

Scala Solution:

```
object Solution {  
  def maximumPoints(enemyEnergies: Array[Int], currentEnergy: Int): Long = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec maximum_points(enemy_energies :: [integer], current_energy :: integer)  
    :: integer  
  def maximum_points(enemy_energies, current_energy) do  
  
  end  
end
```

Erlang Solution:

```
-spec maximum_points(EnemyEnergies :: [integer()], CurrentEnergy ::  
integer()) -> integer().  
maximum_points(EnemyEnergies, CurrentEnergy) ->  
.
```

Racket Solution:

```
(define/contract (maximum-points enemyEnergies currentEnergy)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```