Problem 3147: Taking Maximum Energy From the Mystic Dungeon

Problem Information

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

Problem Description

In a mystic dungeon,

n

magicians are standing in a line. Each magician has an attribute that gives you energy. Some magicians can give you negative energy, which means taking energy from you.

You have been cursed in such a way that after absorbing energy from magician

i

, you will be instantly transported to magician

(i + k)

. This process will be repeated until you reach the magician where

(i + k)

does not exist.

In other words, you will choose a starting point and then teleport with

k

jumps until you reach the end of the magicians' sequence,

absorbing all the energy

during the journey.

You are given an array

energy

and an integer

k

. Return the

maximum

possible energy you can gain.

Note

that when you are reach a magician, you

must

take energy from them, whether it is negative or positive energy.

Example 1:

Input:

energy = [5,2,-10,-5,1], k = 3

Output:

3

Explanation:

We can gain a total energy of 3 by starting from magician 1 absorbing 2 + 1 = 3.

Example 2:

Input:

energy = [-2,-3,-1], k = 2

Output:

-1

Explanation:

We can gain a total energy of -1 by starting from magician 2.

Constraints:

1 <= energy.length <= 10

5

-1000 <= energy[i] <= 1000

1 <= k <= energy.length - 1

Code Snippets

C++:

```
class Solution {
public:
int maximumEnergy(vector<int>& energy, int k) {

}
};
```

Java:

```
class Solution {
public int maximumEnergy(int[] energy, int k) {

}
}
```

Python3:

```
class Solution:
def maximumEnergy(self, energy: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def maximumEnergy(self, energy, k):
"""
:type energy: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} energy
* @param {number} k
* @return {number}
*/
var maximumEnergy = function(energy, k) {

};
```

TypeScript:

```
function maximumEnergy(energy: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximumEnergy(int[] energy, int k) {

}
}
```

C:

```
int maximumEnergy(int* energy, int energySize, int k) {

}
```

Go:

```
func maximumEnergy(energy []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximumEnergy(energy: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumEnergy(_ energy: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_energy(energy: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} energy
# @param {Integer} k
# @return {Integer}
def maximum_energy(energy, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $energy
* @param Integer $k
* @return Integer
*/
function maximumEnergy($energy, $k) {

}
}
```

Dart:

```
class Solution {
int maximumEnergy(List<int> energy, int k) {

}
}
```

Scala:

```
object Solution {
def maximumEnergy(energy: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_energy(energy :: [integer], k :: integer) :: integer
def maximum_energy(energy, k) do
```

```
end
end
```

Erlang:

```
-spec maximum_energy(Energy :: [integer()], K :: integer()) -> integer().
maximum_energy(Energy, K) ->
.
```

Racket:

```
(define/contract (maximum-energy energy k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Taking Maximum Energy From the Mystic Dungeon
* Difficulty: Medium
* Tags: array
*
* 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 maximumEnergy(vector<int>& energy, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Taking Maximum Energy From the Mystic Dungeon
```

```
* Difficulty: Medium
* Tags: array
*
* 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 maximumEnergy(int[] energy, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Taking Maximum Energy From the Mystic Dungeon
Difficulty: Medium
Tags: array

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 maximumEnergy(self, energy: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumEnergy(self, energy, k):
"""
:type energy: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Taking Maximum Energy From the Mystic Dungeon
* Difficulty: Medium
* Tags: array
*
* 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[]} energy
* @param {number} k
* @return {number}
*/
var maximumEnergy = function(energy, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Taking Maximum Energy From the Mystic Dungeon
* Difficulty: Medium
* Tags: array
*
* 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 maximumEnergy(energy: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Taking Maximum Energy From the Mystic Dungeon
* Difficulty: Medium
* Tags: array
```

```
*
* 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 MaximumEnergy(int[] energy, int k) {

}
}
```

C Solution:

```
/*
* Problem: Taking Maximum Energy From the Mystic Dungeon
* Difficulty: Medium
* Tags: array
*
* 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 maximumEnergy(int* energy, int energySize, int k) {

}
```

Go Solution:

```
// Problem: Taking Maximum Energy From the Mystic Dungeon
// Difficulty: Medium
// Tags: array
//
// 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 maximumEnergy(energy []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumEnergy(energy: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumEnergy(_ energy: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Taking Maximum Energy From the Mystic Dungeon
// Difficulty: Medium
// Tags: array
//
// 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_energy(energy: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} energy
# @param {Integer} k
# @return {Integer}
def maximum_energy(energy, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $energy
* @param Integer $k
* @return Integer
*/
function maximumEnergy($energy, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumEnergy(List<int> energy, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumEnergy(energy: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_energy(energy :: [integer], k :: integer) :: integer
def maximum_energy(energy, k) do

end
end
```

Erlang Solution:

```
-spec maximum_energy(Energy :: [integer()], K :: integer()) -> integer().
maximum_energy(Energy, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-energy energy k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```