Problem 3186: Maximum Total Damage With Spell Casting

Problem Information

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

Problem Description

A magician has various spells.

You are given an array

power

, where each element represents the damage of a spell. Multiple spells can have the same damage value.

It is a known fact that if a magician decides to cast a spell with a damage of

power[i]

, they

cannot

cast any spell with a damage of

power[i] - 2

,

power[i] - 1

,

power[i] + 1

, or

power[i] + 2

.

Each spell can be cast

only once

.

Return the

maximum

possible

total damage

that a magician can cast.

Example 1:

Input:

power = [1,1,3,4]

Output:

6

Explanation:

The maximum possible damage of 6 is produced by casting spells 0, 1, 3 with damage 1, 1, 4.

Example 2:

Input:

power = [7,1,6,6]

Output:

13

Explanation:

The maximum possible damage of 13 is produced by casting spells 1, 2, 3 with damage 1, 6, 6.

Constraints:

1 <= power.length <= 10

5

1 <= power[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maximumTotalDamage(vector<int>& power) {

}
};
```

Java:

```
class Solution {
public long maximumTotalDamage(int[] power) {

}
}
```

Python3:

```
class Solution:
def maximumTotalDamage(self, power: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumTotalDamage(power: number[]): number {

};
```

C#:

```
public class Solution {
public long MaximumTotalDamage(int[] power) {

}
}
```

C:

```
long long maximumTotalDamage(int* power, int powerSize) {

}
```

Go:

```
func maximumTotalDamage(power []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumTotalDamage(power: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maximumTotalDamage(_ power: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_total_damage(power: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} power
# @return {Integer}
def maximum_total_damage(power)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $power
* @return Integer
*/
function maximumTotalDamage($power) {

}
}
```

Dart:

```
class Solution {
int maximumTotalDamage(List<int> power) {

}
}
```

Scala:

```
object Solution {
def maximumTotalDamage(power: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_total_damage(power :: [integer]) :: integer
def maximum_total_damage(power) do

end
end
```

Erlang:

```
-spec maximum_total_damage(Power :: [integer()]) -> integer().
maximum_total_damage(Power) ->
.
```

Racket:

```
(define/contract (maximum-total-damage power)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Total Damage With Spell Casting
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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:
long long maximumTotalDamage(vector<int>& power) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Total Damage With Spell Casting
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 long maximumTotalDamage(int[] power) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Total Damage With Spell Casting
Difficulty: Medium
Tags: array, dp, hash, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Total Damage With Spell Casting
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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[]} power
* @return {number}
*/
var maximumTotalDamage = function(power) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Total Damage With Spell Casting
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 maximumTotalDamage(power: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Total Damage With Spell Casting
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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 long MaximumTotalDamage(int[] power) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximum Total Damage With Spell Casting
* Difficulty: Medium
* Tags: array, dp, hash, sort, search
*
* 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
*/

long long maximumTotalDamage(int* power, int powerSize) {

}
```

Go Solution:

```
// Problem: Maximum Total Damage With Spell Casting
// Difficulty: Medium
// Tags: array, dp, hash, sort, search
//
// 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 maximumTotalDamage(power []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumTotalDamage(power: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maximumTotalDamage(_ power: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Total Damage With Spell Casting
// Difficulty: Medium
// Tags: array, dp, hash, sort, search
//
// 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 maximum_total_damage(power: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} power
# @return {Integer}
def maximum_total_damage(power)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $power
* @return Integer
*/
function maximumTotalDamage($power) {

}
}
```

Dart Solution:

```
class Solution {
int maximumTotalDamage(List<int> power) {

}
}
```

Scala Solution:

```
object Solution {
def maximumTotalDamage(power: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_total_damage(power :: [integer]) :: integer
def maximum_total_damage(power) do

end
end
```

Erlang Solution:

```
-spec maximum_total_damage(Power :: [integer()]) -> integer().
maximum_total_damage(Power) ->
.
```

Racket Solution:

```
(define/contract (maximum-total-damage power)
(-> (listof exact-integer?) exact-integer?)
)
```