Problem 2511: Maximum Enemy Forts That Can Be Captured

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

forts

of length

n

representing the positions of several forts.

forts[i]

can be

-1

,

0

, or

1

where:

-1

represents there is

no fort

at the

i

th

position.

0

indicates there is an

enemy

fort at the

i

th

position.

1

indicates the fort at the

i

th

the position is under your command.

Now you have decided to move your army from one of your forts at position

i

to an empty position

j

such that:

0 <= i, j <= n - 1

The army travels over enemy forts

only

. Formally, for all

k

where

min(i,j) < k < max(i,j)

,

forts[k] == 0.

While moving the army, all the enemy forts that come in the way are

captured

.

Return

the

maximum

number of enemy forts that can be captured

. In case it is

impossible

to move your army, or you do not have any fort under your command, return

0

.

Example 1:

Input:

forts = [1,0,0,-1,0,0,0,0,1]

Output:

4

Explanation:

- Moving the army from position 0 to position 3 captures 2 enemy forts, at 1 and 2. - Moving the army from position 8 to position 3 captures 4 enemy forts. Since 4 is the maximum number of enemy forts that can be captured, we return 4.

Example 2:

Input:

forts = [0,0,1,-1]

Output:

0

Explanation:

Since no enemy fort can be captured, 0 is returned.

Constraints:

1 <= forts.length <= 1000

-1 <= forts[i] <= 1

Code Snippets

C++:

```
class Solution {
public:
int captureForts(vector<int>& forts) {

}
};
```

Java:

```
class Solution {
public int captureForts(int[] forts) {

}
}
```

Python3:

```
class Solution:
def captureForts(self, forts: List[int]) -> int:
```

Python:

```
class Solution(object):
def captureForts(self, forts):
```

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

JavaScript:

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

};
```

TypeScript:

```
function captureForts(forts: number[]): number {

};
```

C#:

```
public class Solution {
public int CaptureForts(int[] forts) {

}
}
```

C:

```
int captureForts(int* forts, int fortsSize) {

}
```

Go:

```
func captureForts(forts []int) int {

}
```

Kotlin:

```
class Solution {
fun captureForts(forts: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func captureForts(_ forts: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn capture_forts(forts: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} forts
# @return {Integer}
def capture_forts(forts)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $forts
* @return Integer
*/
function captureForts($forts) {

}
}
```

Dart:

```
class Solution {
int captureForts(List<int> forts) {

}
}
```

Scala:

```
object Solution {
def captureForts(forts: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec capture_forts(forts :: [integer]) :: integer
def capture_forts(forts) do

end
end
```

Erlang:

```
-spec capture_forts(Forts :: [integer()]) -> integer().
capture_forts(Forts) ->
.
```

Racket:

```
(define/contract (capture-forts forts)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Enemy Forts That Can Be Captured
* Difficulty: Easy
* 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 captureForts(vector<int>& forts) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Enemy Forts That Can Be Captured
* Difficulty: Easy
* 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 captureForts(int[] forts) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Enemy Forts That Can Be Captured
Difficulty: Easy
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 captureForts(self, forts: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Enemy Forts That Can Be Captured
* Difficulty: Easy
* 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[]} forts
* @return {number}
*/
var captureForts = function(forts) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Enemy Forts That Can Be Captured
* Difficulty: Easy
* 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 captureForts(forts: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Enemy Forts That Can Be Captured
* Difficulty: Easy
* 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 CaptureForts(int[] forts) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Enemy Forts That Can Be Captured
* Difficulty: Easy
* 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 captureForts(int* forts, int fortsSize) {

}
```

Go Solution:

```
// Problem: Maximum Enemy Forts That Can Be Captured
// Difficulty: Easy
// 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 captureForts(forts []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun captureForts(forts: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func captureForts(_ forts: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Enemy Forts That Can Be Captured
// Difficulty: Easy
// 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 capture_forts(forts: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} forts
# @return {Integer}
def capture_forts(forts)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $forts
* @return Integer
*/
function captureForts($forts) {

}
}
```

Dart Solution:

```
class Solution {
int captureForts(List<int> forts) {

}
}
```

Scala Solution:

```
object Solution {
def captureForts(forts: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec capture_forts(forts :: [integer]) :: integer
def capture_forts(forts) do

end
end
```

Erlang Solution:

```
-spec capture_forts(Forts :: [integer()]) -> integer().
capture_forts(Forts) ->
.
```

Racket Solution:

```
(define/contract (capture-forts forts)
(-> (listof exact-integer?) exact-integer?)
)
```