Problem 948: Bag of Tokens

Problem Information

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

Problem Description

You start with an initial

power

of

power

, an initial

score

of

0

, and a bag of tokens given as an integer array

tokens

, where each

tokens[i]

denotes the value of token

i

.

Your goal is to

maximize

the total

score

by strategically playing these tokens. In one move, you can play an

unplayed

token in one of the two ways (but not both for the same token):

Face-up

: If your current power is

at least

tokens[i]

, you may play token

i

, losing

tokens[i]

power and gaining

1

score.

Face-down

: If your current score is

at least

1

, you may play token

i

, gaining

tokens[i]

power and losing

1

score.

Return

the

maximum

possible score you can achieve after playing

any

number of tokens

.

Example 1:

Input:

tokens = [100], power = 50

Output:

0

Explanation

:

Since your score is

0

initially, you cannot play the token face-down. You also cannot play it face-up since your power (

50

) is less than

tokens[0]

(

100

).

Example 2:

Input:

tokens = [200,100], power = 150

Output:

1

Explanation:

Play token

1

(

100

) face-up, reducing your power to

50

and increasing your score to

1

.

There is no need to play token

0

, since you cannot play it face-up to add to your score. The maximum score achievable is

1

.

Example 3:

Input:

tokens = [100,200,300,400], power = 200

Output:

2

Explanation:

Play the tokens in this order to get a score of

2

:

Play token

0

(

100

) face-up, reducing power to

100

and increasing score to

1

.

Play token

3

(

400

) face-down, increasing power to

500

and reducing score to

0

.

Play token

1

(

200

) face-up, reducing power to

300

and increasing score to

1

.

Play token

2

(

300

) face-up, reducing power to

0

and increasing score to

2

.

The maximum score achievable is

2

.

Constraints:

0 <= tokens.length <= 1000

0 <= tokens[i], power < 10

4

Code Snippets

C++:

```
class Solution {
public:
int bagOfTokensScore(vector<int>& tokens, int power) {

}
};
```

Java:

```
class Solution {
public int bagOfTokensScore(int[] tokens, int power) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function bagOfTokensScore(tokens: number[], power: number): number {

};
```

C#:

```
public class Solution {
public int BagOfTokensScore(int[] tokens, int power) {

}
}
```

C:

```
int bagOfTokensScore(int* tokens, int tokensSize, int power) {

}
```

Go:

```
func bagOfTokensScore(tokens []int, power int) int {

}
```

Kotlin:

```
class Solution {
fun bagOfTokensScore(tokens: IntArray, power: Int): Int {

}
}
```

Swift:

```
class Solution {
func bagOfTokensScore(_ tokens: [Int], _ power: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn bag_of_tokens_score(tokens: Vec<i32>, power: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} tokens
# @param {Integer} power
# @return {Integer}
def bag_of_tokens_score(tokens, power)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[] $tokens
* @param Integer $power
* @return Integer
*/
function bagOfTokensScore($tokens, $power) {

}
}
```

Dart:

```
class Solution {
int bagOfTokensScore(List<int> tokens, int power) {

}
}
```

Scala:

```
object Solution {
def bagOfTokensScore(tokens: Array[Int], power: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec bag_of_tokens_score(tokens :: [integer], power :: integer) :: integer
def bag_of_tokens_score(tokens, power) do

end
end
```

Erlang:

```
-spec bag_of_tokens_score(Tokens :: [integer()], Power :: integer()) ->
integer().
bag_of_tokens_score(Tokens, Power) ->
.
```

Racket:

```
(define/contract (bag-of-tokens-score tokens power)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Bag of Tokens
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 bagOfTokensScore(vector<int>& tokens, int power) {

}
};
```

Java Solution:

```
/**
* Problem: Bag of Tokens
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 bagOfTokensScore(int[] tokens, int power) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Bag of Tokens
Difficulty: Medium
Tags: array, greedy, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Bag of Tokens
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[]} tokens
* @param {number} power
* @return {number}
*/
var bagOfTokensScore = function(tokens, power) {

};
```

TypeScript Solution:

```
/**
* Problem: Bag of Tokens
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 bagOfTokensScore(tokens: number[], power: number): number {

};
```

C# Solution:

```
/*
* Problem: Bag of Tokens
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 BagOfTokensScore(int[] tokens, int power) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Bag of Tokens
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 bagOfTokensScore(int* tokens, int tokensSize, int power) {

}
```

Go Solution:

```
// Problem: Bag of Tokens
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 bagOfTokensScore(tokens []int, power int) int {

}
```

Kotlin Solution:

```
class Solution {
fun bagOfTokensScore(tokens: IntArray, power: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func bagOfTokensScore(_ tokens: [Int], _ power: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Bag of Tokens
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 bag_of_tokens_score(tokens: Vec<i32>, power: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} tokens
# @param {Integer} power
# @return {Integer}
def bag_of_tokens_score(tokens, power)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $tokens
* @param Integer $power
* @return Integer
*/
```

```
function bagOfTokensScore($tokens, $power) {

}
}
```

Dart Solution:

```
class Solution {
int bagOfTokensScore(List<int> tokens, int power) {

}
}
```

Scala Solution:

```
object Solution {
def bagOfTokensScore(tokens: Array[Int], power: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec bag_of_tokens_score(tokens :: [integer], power :: integer) :: integer
def bag_of_tokens_score(tokens, power) do

end
end
```

Erlang Solution:

```
-spec bag_of_tokens_score(Tokens :: [integer()], Power :: integer()) ->
integer().
bag_of_tokens_score(Tokens, Power) ->
.
```

Racket Solution:

```
(define/contract (bag-of-tokens-score tokens power)
(-> (listof exact-integer?) exact-integer? exact-integer?)
```

```
)
```