Problem 312: Burst Balloons

Problem Information

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

Problem Description

You are given

n

balloons, indexed from

0

to

n - 1

. Each balloon is painted with a number on it represented by an array

nums

. You are asked to burst all the balloons.

If you burst the

i

th

balloon, you will get

nums[i - 1] * nums[i] * nums[i + 1]

coins. If

i - 1

or

i + 1

goes out of bounds of the array, then treat it as if there is a balloon with a

1

painted on it.

Return

the maximum coins you can collect by bursting the balloons wisely

.

Example 1:

Input:

nums = [3,1,5,8]

Output:

167

Explanation:

nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> [] coins = 3*1*5 + 3*5*8 + 1*3*8 + 1*8*1 = 167

Example 2:

Input:

nums = [1,5]

Output:

10

Constraints:

n == nums.length

1 <= n <= 300

0 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int maxCoins(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int maxCoins(int[] nums) {

}
}
```

Python3:

```
class Solution:
def maxCoins(self, nums: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxCoins(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxCoins(int[] nums) {

}
}
```

C:

```
int maxCoins(int* nums, int numsSize) {

}
```

Go:

```
func maxCoins(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun maxCoins(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxCoins(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_coins(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def max_coins(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maxCoins($nums) {

}
```

```
}
```

Dart:

```
class Solution {
int maxCoins(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def maxCoins(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_coins(nums :: [integer]) :: integer
def max_coins(nums) do

end
end
```

Erlang:

```
-spec max_coins(Nums :: [integer()]) -> integer().
max_coins(Nums) ->
.
```

Racket:

```
(define/contract (max-coins nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Burst Balloons
* Difficulty: Hard
* Tags: array, dp
*
* 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:
int maxCoins(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Burst Balloons
* Difficulty: Hard
* Tags: array, dp
*
* 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 int maxCoins(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Burst Balloons
Difficulty: Hard
Tags: array, dp
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Burst Balloons
* Difficulty: Hard
* Tags: array, dp
*
* 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[]} nums
* @return {number}
*/
var maxCoins = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Burst Balloons
* Difficulty: Hard
* Tags: array, dp
*
* 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 maxCoins(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Burst Balloons
* Difficulty: Hard
* Tags: array, dp
*
* 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 int MaxCoins(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Burst Balloons
* Difficulty: Hard
* Tags: array, dp
*
* 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
```

```
*/

int maxCoins(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Burst Balloons
// Difficulty: Hard
// Tags: array, dp
//
// 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 maxCoins(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCoins(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxCoins(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Burst Balloons
// Difficulty: Hard
// Tags: array, dp
```

```
//
// 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 max_coins(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def max_coins(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maxCoins($nums) {

}
}
```

Dart Solution:

```
class Solution {
int maxCoins(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def maxCoins(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_coins(nums :: [integer]) :: integer
def max_coins(nums) do

end
end
```

Erlang Solution:

```
-spec max_coins(Nums :: [integer()]) -> integer().
max_coins(Nums) ->
.
```

Racket Solution:

```
(define/contract (max-coins nums)
(-> (listof exact-integer?) exact-integer?)
)
```