Problem 1039: Minimum Score Triangulation of Polygon

Problem Information

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

Problem Description

You have a convex

n

-sided polygon where each vertex has an integer value. You are given an integer array

values

where

values[i]

is the value of the

i

th

vertex in

clockwise order

.

Polygon

triangulation

is a process where you divide a polygon into a set of triangles and the vertices of each triangle must also be vertices of the original polygon. Note that no other shapes other than triangles are allowed in the division. This process will result in

n - 2

triangles.

You will

triangulate

the polygon. For each triangle, the

weight

of that triangle is the product of the values at its vertices. The total score of the triangulation is the sum of these

weights

over all

n - 2

triangles.

Return the

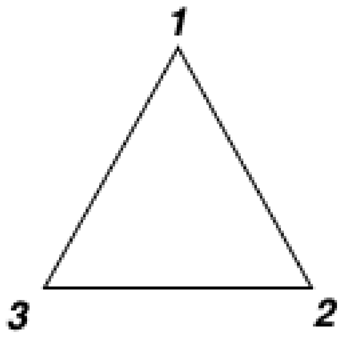
minimum possible score

that you can achieve with some

triangulation

of the polygon.

Example 1:



Input:

values = [1,2,3]

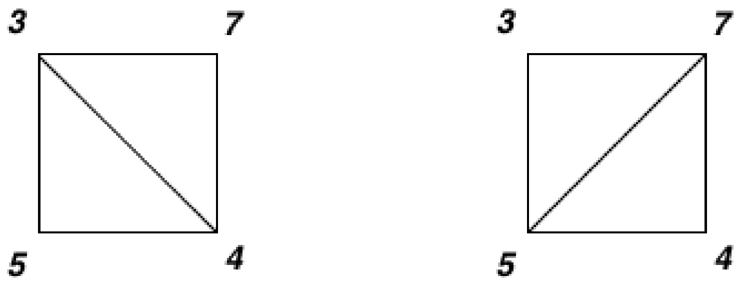
Output:

6

Explanation:

The polygon is already triangulated, and the score of the only triangle is 6.

Example 2:



Input:

values = [3,7,4,5]

Output:

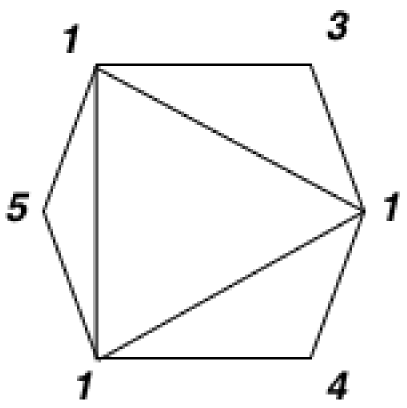
144

Explanation:

There are two triangulations, with possible scores: 3*7*5 + 4*5*7 = 245, or 3*4*5 + 3*4*7 = 144.

The minimum score is 144.

Example 3:



Input:

values = [1,3,1,4,1,5]

Output:

13

Explanation:

The minimum score triangulation is 1*1*3 + 1*1*4 + 1*1*5 + 1*1*1 = 13.

Constraints:

n == values.length

3 <= n <= 50

1 <= values[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int minScoreTriangulation(vector<int>& values) {

}
};
```

Java:

```
class Solution {
public int minScoreTriangulation(int[] values) {

}
}
```

Python3:

```
class Solution:
def minScoreTriangulation(self, values: List[int]) -> int:
```

Python:

```
class Solution(object):
def minScoreTriangulation(self, values):
"""
:type values: List[int]
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minScoreTriangulation(values: number[]): number {

};
```

C#:

```
public class Solution {
public int MinScoreTriangulation(int[] values) {

}
}
```

C:

```
int minScoreTriangulation(int* values, int valuesSize) {

}
```

Go:

```
func minScoreTriangulation(values []int) int {

}
```

Kotlin:

```
class Solution {
fun minScoreTriangulation(values: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minScoreTriangulation(_ values: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_score_triangulation(values: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} values
# @return {Integer}
def min_score_triangulation(values)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $values
* @return Integer
*/
function minScoreTriangulation($values) {

}
}
```

Dart:

```
class Solution {
int minScoreTriangulation(List<int> values) {

}
}
```

Scala:

```
object Solution {
def minScoreTriangulation(values: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_score_triangulation(values :: [integer]) :: integer
def min_score_triangulation(values) do

end
end
```

Erlang:

```
-spec min_score_triangulation(Values :: [integer()]) -> integer().
min_score_triangulation(Values) ->
.
```

Racket:

```
(define/contract (min-score-triangulation values)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Score Triangulation of Polygon
* Difficulty: Medium
* 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 minScoreTriangulation(vector<int>& values) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Score Triangulation of Polygon
* Difficulty: Medium
* 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 minScoreTriangulation(int[] values) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Score Triangulation of Polygon
Difficulty: Medium
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 minScoreTriangulation(self, values: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Score Triangulation of Polygon
* Difficulty: Medium
* 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[]} values
* @return {number}
*/
var minScoreTriangulation = function(values) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Score Triangulation of Polygon
* Difficulty: Medium
* 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 minScoreTriangulation(values: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Score Triangulation of Polygon
* Difficulty: Medium
* 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 MinScoreTriangulation(int[] values) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Score Triangulation of Polygon
* Difficulty: Medium
* 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 minScoreTriangulation(int* values, int valuesSize) {

}
```

Go Solution:

```
// Problem: Minimum Score Triangulation of Polygon
// Difficulty: Medium
// 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 minScoreTriangulation(values []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minScoreTriangulation(values: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minScoreTriangulation(_ values: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Score Triangulation of Polygon
// Difficulty: Medium
// 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 min_score_triangulation(values: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} values
# @return {Integer}
def min_score_triangulation(values)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $values
* @return Integer
*/
function minScoreTriangulation($values) {

}
}
```

Dart Solution:

```
class Solution {
int minScoreTriangulation(List<int> values) {

}
}
```

Scala Solution:

```
object Solution {
def minScoreTriangulation(values: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_score_triangulation(values :: [integer]) :: integer
def min_score_triangulation(values) do

end
end
```

Erlang Solution:

```
-spec min_score_triangulation(Values :: [integer()]) -> integer().
min_score_triangulation(Values) ->
.
```

Racket Solution:

```
(define/contract (min-score-triangulation values)
(-> (listof exact-integer?) exact-integer?)
)
```