# Problem 492: Construct the Rectangle

## Problem Information

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

## Problem Description

A web developer needs to know how to design a web page's size. So, given a specific rectangular web page’s area, your job by now is to design a rectangular web page, whose length L and width W satisfy the following requirements:

The area of the rectangular web page you designed must equal to the given target area.

The width

W

should not be larger than the length

L

, which means

L >= W

.

The difference between length

L

and width

W

should be as small as possible.

Return

an array

[L, W]

where

L

and

W

are the length and width of the web page you designed in sequence.

Example 1:

Input:

area = 4

Output:

[2,2]

Explanation:

The target area is 4, and all the possible ways to construct it are [1,4], [2,2], [4,1]. But according to requirement 2, [1,4] is illegal; according to requirement 3, [4,1] is not optimal compared to [2,2]. So the length L is 2, and the width W is 2.

Example 2:

Input:

area = 37

Output:

[37,1]

Example 3:

Input:

area = 122122

Output:

[427,286]

Constraints:

1 <= area <= 10

7

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> constructRectangle(int area) {

}
};
```

**Java:**

```
class Solution {
public int[] constructRectangle(int area) {

}
```

```
}
```

**Python3:**

```
class Solution:
def constructRectangle(self, area: int) -> List[int]:
```

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function constructRectangle(area: number): number[] {

};
```

**C#:**

```
public class Solution {
public int[] ConstructRectangle(int area) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* constructRectangle(int area, int* returnSize) {

}
```

**Go:**

```
func constructRectangle(area int) []int {

}
```

**Kotlin:**

```
class Solution {
fun constructRectangle(area: Int): IntArray {

}
}
```

**Swift:**

```
class Solution {
func constructRectangle(_ area: Int) -> [Int] {

}
}
```

**Rust:**

```
impl Solution {
pub fn construct_rectangle(area: i32) -> Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer} area
# @return {Integer[]}
def construct_rectangle(area)
```

```
end
```

**PHP:**

```
class Solution {

/**
* @param Integer $area
* @return Integer[]
*/
function constructRectangle($area) {

}
}
```

**Dart:**

```
class Solution {
List<int> constructRectangle(int area) {

}
}
```

**Scala:**

```
object Solution {
def constructRectangle(area: Int): Array[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec construct_rectangle(area :: integer) :: [integer]
def construct_rectangle(area) do

end
end
```

**Erlang:**

```
-spec construct_rectangle(Area :: integer()) -> [integer()].
construct_rectangle(Area) ->
.
```

**Racket:**

```
(define/contract (construct-rectangle area)
(-> exact-integer? (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Construct the Rectangle
* Difficulty: Easy
* Tags: array, math
*
* 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:
vector<int> constructRectangle(int area) {

}
};
```

### Java Solution:

```
/**
* Problem: Construct the Rectangle
* Difficulty: Easy
* Tags: array, math
*
* 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[] constructRectangle(int area) {

}
}
```

### Python3 Solution:

```
"""
Problem: Construct the Rectangle
Difficulty: Easy
Tags: array, math

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Construct the Rectangle
* Difficulty: Easy
* Tags: array, math
*
```

```
* 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} area
* @return {number[]}
*/
var constructRectangle = function(area) {

};
```

### TypeScript Solution:

```
/**
* Problem: Construct the Rectangle
* Difficulty: Easy
* Tags: array, math
*
* 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 constructRectangle(area: number): number[] {

};
```

### C# Solution:

```
/*
* Problem: Construct the Rectangle
* Difficulty: Easy
* Tags: array, math
*
* 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[] ConstructRectangle(int area) {

}
}
```

### C Solution:

```
/*
* Problem: Construct the Rectangle
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* constructRectangle(int area, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Construct the Rectangle
// Difficulty: Easy
// Tags: array, math
//
// 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 constructRectangle(area int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun constructRectangle(area: Int): IntArray {

}
}
```

**Swift Solution:**

```
class Solution {
func constructRectangle(_ area: Int) -> [Int] {

}
}
```

**Rust Solution:**

```
// Problem: Construct the Rectangle
// Difficulty: Easy
// Tags: array, math
//
// 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 construct_rectangle(area: i32) -> Vec<i32> {

}
}
```

**Ruby Solution:**

```
# @param {Integer} area
# @return {Integer[]}
def construct_rectangle(area)

end
```

**PHP Solution:**

```
class Solution {
```

```
/**
* @param Integer $area
* @return Integer[]
*/
function constructRectangle($area) {

}
}
```

**Dart Solution:**

```
class Solution {
List<int> constructRectangle(int area) {

}
}
```

**Scala Solution:**

```
object Solution {
def constructRectangle(area: Int): Array[Int] = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec construct_rectangle(area :: integer) :: [integer]
def construct_rectangle(area) do

end
end
```

**Erlang Solution:**

```
-spec construct_rectangle(Area :: integer()) -> [integer()].
construct_rectangle(Area) ->
.
```

**Racket Solution:**

```
(define/contract (construct-rectangle area)
(-> exact-integer? (listof exact-integer?))
)
```