# Problem 699: Falling Squares

## Problem Information

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

## Problem Description

There are several squares being dropped onto the X-axis of a 2D plane.

You are given a 2D integer array

positions

where

positions[i] = [left

i

, sideLength

i

]

represents the

i

th

square with a side length of

sideLength

i

that is dropped with its left edge aligned with X-coordinate

left

i

.

Each square is dropped one at a time from a height above any landed squares. It then falls downward (negative Y direction) until it either lands

on the top side of another square

or

on the X-axis

. A square brushing the left/right side of another square does not count as landing on it. Once it lands, it freezes in place and cannot be moved.

After each square is dropped, you must record the

height of the current tallest stack of squares

.

Return

an integer array

ans

where

ans[i]

represents the height described above after dropping the

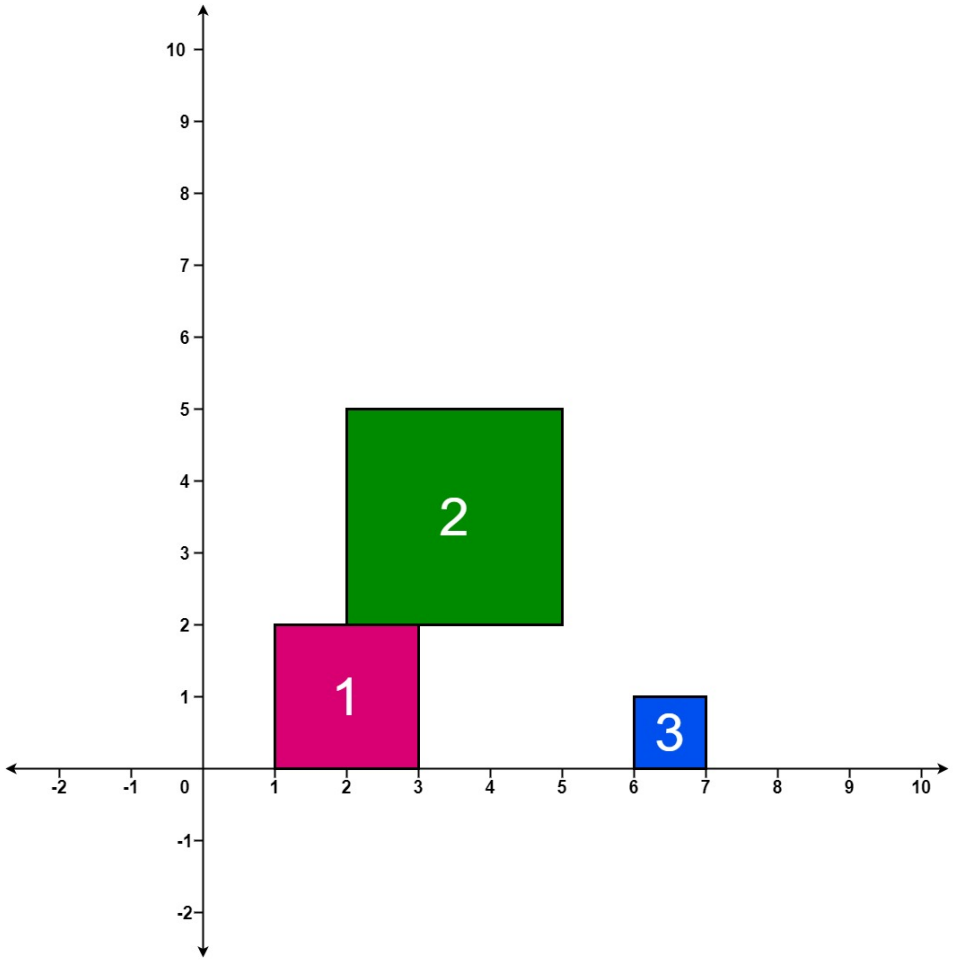
i

th

square

.

Example 1:




Input:

positions = [[1,2],[2,3],[6,1]]

Output:

[2,5,5]

Explanation:

After the first drop, the tallest stack is square 1 with a height of 2. After the second drop, the tallest stack is squares 1 and 2 with a height of 5. After the third drop, the tallest stack is still squares 1 and 2 with a height of 5. Thus, we return an answer of [2, 5, 5].

Example 2:

Input:

positions = [[100,100],[200,100]]

Output:

[100,100]

Explanation:

After the first drop, the tallest stack is square 1 with a height of 100. After the second drop, the tallest stack is either square 1 or square 2, both with heights of 100. Thus, we return an answer of [100, 100]. Note that square 2 only brushes the right side of square 1, which does not count as landing on it.

Constraints:

1 <= positions.length <= 1000

1 <= left

i

<= 10

8

1 <= sideLength

i

<= 10

6

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> fallingSquares(vector<vector<int>>& positions) {

}
};
```

**Java:**

```
class Solution {
public List<Integer> fallingSquares(int[][] positions) {

}
}
```

**Python3:**

```
class Solution:
def fallingSquares(self, positions: List[List[int]]) -> List[int]:
```

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function fallingSquares(positions: number[][]): number[] {

};
```

**C#:**

```
public class Solution {
public IList<int> FallingSquares(int[][] positions) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* fallingSquares(int** positions, int positionsSize, int*
positionsColSize, int* returnSize) {

}
```

**Go:**

```
func fallingSquares(positions [][]int) []int {

}
```

**Kotlin:**

```
class Solution {
fun fallingSquares(positions: Array<IntArray>): List<Int> {
```

```
}
}
```

**Swift:**

```
class Solution {
func fallingSquares(_ positions: [[Int]]) -> [Int] {

}
}
```

**Rust:**

```
impl Solution {
pub fn falling_squares(positions: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer[][]} positions
# @return {Integer[]}
def falling_squares(positions)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $positions
* @return Integer[]
*/
function fallingSquares($positions) {

}
}
```

**Dart:**

```
class Solution {
List<int> fallingSquares(List<List<int>> positions) {

}
}
```

**Scala:**

```
object Solution {
def fallingSquares(positions: Array[Array[Int]]): List[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec falling_squares(positions :: [[integer]]) :: [integer]
def falling_squares(positions) do

end
end
```

**Erlang:**

```
-spec falling_squares(Positions :: [[integer()]]) -> [integer()].
falling_squares(Positions) ->
.
```

**Racket:**

```
(define/contract (falling-squares positions)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Falling Squares
```

```
* Difficulty: Hard
* Tags: array, tree, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> fallingSquares(vector<vector<int>>& positions) {

}
};
```

**Java Solution:**

```
/**
* Problem: Falling Squares
* Difficulty: Hard
* Tags: array, tree, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<Integer> fallingSquares(int[][] positions) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Falling Squares
Difficulty: Hard
Tags: array, tree, stack

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def fallingSquares(self, positions: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Falling Squares
* Difficulty: Hard
* Tags: array, tree, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} positions
* @return {number[]}
*/
var fallingSquares = function(positions) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Falling Squares
* Difficulty: Hard
* Tags: array, tree, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function fallingSquares(positions: number[][]): number[] {

};
```

### C# Solution:

```
/*
* Problem: Falling Squares
* Difficulty: Hard
* Tags: array, tree, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<int> FallingSquares(int[][] positions) {

}
}
```

### C Solution:

```
/*
* Problem: Falling Squares
* Difficulty: Hard
* Tags: array, tree, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* fallingSquares(int** positions, int positionsSize, int*
positionsColSize, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Falling Squares
// Difficulty: Hard
// Tags: array, tree, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func fallingSquares(positions [][]int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun fallingSquares(positions: Array<IntArray>): List<Int> {

}
}
```

### Swift Solution:

```
class Solution {
func fallingSquares(_ positions: [[Int]]) -> [Int] {

}
}
```

### Rust Solution:

```
// Problem: Falling Squares
// Difficulty: Hard
// Tags: array, tree, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn falling_squares(positions: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} positions
# @return {Integer[]}
def falling_squares(positions)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $positions
* @return Integer[]
*/
function fallingSquares($positions) {

}
}
```

### Dart Solution:

```
class Solution {
List<int> fallingSquares(List<List<int>> positions) {

}
}
```

### Scala Solution:

```
object Solution {
def fallingSquares(positions: Array[Array[Int]]): List[Int] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec falling_squares(positions :: [[integer]]) :: [integer]
def falling_squares(positions) do

end
end
```

### Erlang Solution:

```
-spec falling_squares(Positions :: [[integer()]]) -> [integer()].
falling_squares(Positions) ->
.
```

### Racket Solution:

```
(define/contract (falling-squares positions)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```