Problem 735: Asteroid Collision

Problem Information

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

Problem Description

We are given an array

asteroids

of integers representing asteroids in a row. The indices of the asteroid in the array represent their relative position in space.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:

Input:

asteroids = [5,10,-5]

Output:

[5,10]

Explanation:

The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Example 2:

Input:

asteroids = [8,-8]

Output:

[]

Explanation:

The 8 and -8 collide exploding each other.

Example 3:

Input:

asteroids = [10,2,-5]

Output:

[10]

Explanation:

The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.

Example 4:

Input:

asteroids = [3,5,-6,2,-1,4]

Output:

[-6,2,4]

Explanation:

The asteroid -6 makes the asteroid 3 and 5 explode, and then continues going left. On the other side, the asteroid 2 makes the asteroid -1 explode and then continues going right, without reaching asteroid 4.

Constraints:

2 <= asteroids.length <= 10

4

-1000 <= asteroids[i] <= 1000

asteroids[i] != 0

Code Snippets

C++:

```
class Solution {
public:
vector<int> asteroidCollision(vector<int>& asteroids) {

}
};
```

Java:

```
class Solution {
public int[] asteroidCollision(int[] asteroids) {

}
}
```

Python3:

```
class Solution:
def asteroidCollision(self, asteroids: List[int]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function asteroidCollision(asteroids: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] AsteroidCollision(int[] asteroids) {

}
}
```

C:

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

}
```

Go:

```
func asteroidCollision(asteroids []int) []int {

}
```

Kotlin:

```
class Solution {
fun asteroidCollision(asteroids: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func asteroidCollision(_ asteroids: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn asteroid_collision(asteroids: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} asteroids
# @return {Integer[]}
def asteroid_collision(asteroids)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $asteroids
* @return Integer[]
*/
function asteroidCollision($asteroids) {

}
}
```

Dart:

```
class Solution {
List<int> asteroidCollision(List<int> asteroids) {

}
}
```

Scala:

```
object Solution {
def asteroidCollision(asteroids: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec asteroid_collision(asteroids :: [integer]) :: [integer]
def asteroid_collision(asteroids) do

end
end
```

Erlang:

```
-spec asteroid_collision(Asteroids :: [integer()]) -> [integer()].
asteroid_collision(Asteroids) ->
.
```

Racket:

```
(define/contract (asteroid-collision asteroids)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Asteroid Collision
* Difficulty: Medium
* Tags: array, stack
*
* 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> asteroidCollision(vector<int>& asteroids) {

}
};
```

Java Solution:

```
/**
* Problem: Asteroid Collision
* Difficulty: Medium
* Tags: array, stack
*
* 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[] asteroidCollision(int[] asteroids) {

}
}
```

Python3 Solution:

```
"""
Problem: Asteroid Collision
Difficulty: Medium
Tags: array, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Asteroid Collision
* Difficulty: Medium
* Tags: array, stack
*
* 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[]} asteroids
* @return {number[]}
*/
var asteroidCollision = function(asteroids) {

};
```

TypeScript Solution:

```
/**
* Problem: Asteroid Collision
* Difficulty: Medium
* Tags: array, stack
*
* 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 asteroidCollision(asteroids: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Asteroid Collision
* Difficulty: Medium
* Tags: array, stack
*
* 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[] AsteroidCollision(int[] asteroids) {

}
}
```

C Solution:

```
/*
* Problem: Asteroid Collision
* Difficulty: Medium
* Tags: array, stack
*
* 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* asteroidCollision(int* asteroids, int asteroidsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Asteroid Collision
// Difficulty: Medium
// Tags: array, stack
//
// 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 asteroidCollision(asteroids []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun asteroidCollision(asteroids: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func asteroidCollision(_ asteroids: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Asteroid Collision
// Difficulty: Medium
// Tags: array, stack
//
// 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 asteroid_collision(asteroids: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} asteroids
# @return {Integer[]}
def asteroid_collision(asteroids)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $asteroids
* @return Integer[]
*/
function asteroidCollision($asteroids) {

}
}
```

Dart Solution:

```
class Solution {
List<int> asteroidCollision(List<int> asteroids) {

}
}
```

Scala Solution:

```
object Solution {
def asteroidCollision(asteroids: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec asteroid_collision(asteroids :: [integer]) :: [integer]
def asteroid_collision(asteroids) do

end
end
```

Erlang Solution:

```
-spec asteroid_collision(Asteroids :: [integer()]) -> [integer()].
asteroid_collision(Asteroids) ->
.
```

Racket Solution:

```
(define/contract (asteroid-collision asteroids)
(-> (listof exact-integer?) (listof exact-integer?))
)
```