<https://leetcode.com/problems/asteroid-collision/>

We are given an array asteroids of integers representing asteroids in a row.

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.

**Constraints:**

2 <= asteroids.length <= 10^4

-1000 <= asteroids[i] <= 1000

asteroids[i] != 0

**Attempt 1: 2024-10-20**

**Solution 1: Stack (30 min)**

**Style 1: More intuitive to understand**

class Solution {

    public int[] asteroidCollision(int[] asteroids) {

        Stack<Integer> stack = new Stack<>();

        for(int ast : asteroids) {

            boolean push = true;

            // Only when current asteroid is negative and the top of stack is positive, collision happens

            while(!stack.isEmpty() && ast < 0 && stack.peek() > 0) {

                int cur = stack.peek();

                // Top explodes, need push

                if(cur < -ast) {

                    stack.pop();

                    continue;

                // Both explode, pop out top and no need push

                } else if(cur == -ast) {

                    stack.pop();

                    push = false;

                    break;

                // Current asteroid explodes, no need push

                } else {

                    push = false;

                    break;

                }

            }

            if(push) {

                stack.push(ast);

            }

        }

        int[] result = new int[stack.size()];

        for(int i = result.length - 1; i >= 0; i--) {

            result[i] = stack.pop();

        }

        return result;

    }

}

Time Complexity: O(n), each asteroid is pushed and popped at most once.

Space Complexity: O(n) for the stack.

**Refer to Deepseek**

I'll provide a Java solution for LeetCode 735. "Asteroid Collision", along with an explanation and complexity analysis.

**Problem Description**

We are given an array asteroids of integers representing asteroids in a row. For each asteroid:

The absolute value represents its size.

The sign represents its direction (positive: moving right, negative: moving left).

All asteroids move at the same speed.

When two asteroids meet:

The smaller one explodes.

If both are the same size, both explode.

Asteroids moving in the same direction never meet. The problem is to return the state of the asteroids after all collisions.

**Java Solution**

import java.util.Stack;

class Solution {

public int[] asteroidCollision(int[] asteroids) {

Stack<Integer> stack = new Stack<>();

for (int ast : asteroids) {

boolean push = true;

// Only when current asteroid is negative and the top of stack is positive, collision happens

while (!stack.isEmpty() && ast < 0 && stack.peek() > 0) {

int top = stack.peek();

if (top < -ast) {

stack.pop(); // top explodes

continue; // check next top

} else if (top == -ast) {

stack.pop(); // both explode

push = false;

break;

} else {

push = false; // current asteroid explodes

break;

}

}

if (push) {

stack.push(ast);

}

}

// Convert stack to array

int[] result = new int[stack.size()];

for (int i = result.length - 1; i >= 0; i--) {

result[i] = stack.pop();

}

return result;

}

}

**Explanation**

**Use a Stack**:

We traverse each asteroid.

Collisions only occur when a negative asteroid (moving left) meets a positive asteroid (moving right) in the stack.

**Handle Collisions**:

While the stack is not empty, and the current asteroid is negative, and the top of the stack is positive:

If the top asteroid is smaller than the current (absolute value), pop it from the stack (it explodes).

If they are equal, pop the top and skip pushing the current (both explode).

If the top is larger, skip pushing the current (it explodes).

**Push Asteroid**:

If no collision occurs or the current asteroid survives, push it to the stack.

**Convert Stack to Array**:

The stack contains the asteroids that survived, in reverse order. We pop them into an array from the end to the beginning.

**Example Walkthrough**

Let asteroids = [5, 10, -5]:

Push 5: stack = [5]

Push 10: stack = [5,10]

Current asteroid = -5:

Compare with top (10): 10 > 5? So -5 explodes? Wait, actually:

top = 10, current = -5: 10 > |-5|? So current explodes? But wait, the rule: if top is larger, current explodes.

So we break without pushing.

Result: [5,10]

Another example: asteroids = [10, 2, -5]:

Push 10: stack = [10]

Push 2: stack = [10,2]

Current = -5:

Compare with top (2): 2 < |-5|? So pop 2 (explodes). Now stack = [10].

Compare with top (10): 10 > |-5|? So current explodes.

Result: [10]

**Complexity Analysis**

**Time Complexity**: O(n), each asteroid is pushed and popped at most once.

**Space Complexity**: O(n) for the stack.

This solution efficiently simulates the collisions using a stack. Let me know if you need further clarification!

**Style 2:**

class Solution {

    public int[] asteroidCollision(int[] asteroids) {

        Stack<Integer> stack = new Stack<>();

        for(int a : asteroids) {

            if(a > 0) {

                stack.push(a);

            } else {

                // 'stack.peek() > 0' for negative asteroid on top case,

                // at that moment only one negative asteroid in stack

                while(!stack.isEmpty() && stack.peek() > 0 && stack.peek() < Math.abs(a)) {

                    stack.pop();

                }

                // If no remain asteroids on stack is positive(empty

                // or all negative, e.g [-2,-1,1,2], because all

                // previous smaller positive ones are removed since

                // they explode), then we can push

                if(stack.isEmpty() || stack.peek() < 0) {

                    stack.push(a);

                // Handle corner case as same value but negative + positive

                // e.g [8,-8]

                } else if(stack.peek() + a == 0) {

                    stack.pop();

                }

            }

        }

        int n = stack.size();

        int[] result = new int[n];

        int i = n - 1;

        while(!stack.isEmpty()) {

            result[i--] = stack.pop();

        }

        return result;

    }

}

**Refer to**

<https://algo.monster/liteproblems/735>

**Problem Description**

In this problem, we are dealing with a simulation of asteroids moving in a row, represented by an array called asteroids. Each integer in the array represents an asteroid's size (absolute value) and direction (sign of the integer). Positive numbers mean the asteroid is moving to the right, while negative numbers mean the asteroid is moving to the left. All asteroids move at the same constant speed.

The task is to simulate the collisions that may happen when asteroids meet as they move. When two asteroids collide, the following rules apply:

If one asteroid is larger than the other, the smaller asteroid is destroyed.

If both asteroids are the same size, they both are destroyed.

Asteroids moving in the same direction do not collide.

The goal is to determine the final state of the asteroids after all possible collisions have occurred.

**Intuition**

To solve this problem, we use a stack data structure that will help us manage the asteroid collisions efficiently. The stack is chosen because collisions affect the asteroids in a last-in, first-out manner: the most recently moving right asteroid can collide with the newly encountered left-moving one.

Here's the intuition for the solution approach:

If we encounter an asteroid moving to the right (x > 0), it won't collide with any of the previous asteroids. We add (push) this asteroid to the stack, as it could collide with a future left-moving asteroid.

When an asteroid moving to the left (x < 0) is found, we need to check if it will collide with any of the right-moving asteroids already in the stack.

If a collision is possible (the stack is not empty and the top asteroid in the stack is a positive number, meaning it is moving right), and the top asteroid in the stack is smaller than the current one (absolute size comparison), the top asteroid will explode (be removed by pop from the stack).

If the top asteroid in the stack has the same size (but opposite direction) as the current one, they both explode (we pop the top of the stack without adding the current one).

If the stack is empty or contains only asteroids moving to the left, it means there will be no collision for the current asteroid; therefore, we can safely add (push) the current asteroid to the stack.

By applying these steps, we simulate all possible collisions. Once we finish iterating through the asteroids array, the remaining asteroids in the stack will be the ones surviving after all collisions, which becomes our final result.

**Solution Approach**

The solution approach leverages a stack to keep track of the asteroids that are still in play, i.e., those that haven't been destroyed in a collision.

Here is the step-by-step approach based on the Reference Solution provided:

We initialize an empty list stk to act as our stack.

We iterate through each asteroid x in the asteroids array from left to right.

For an asteroid moving to the right (x > 0), we push it onto the stack since it cannot collide with asteroids already on the stack (as they are either also moving to the right or are larger left-moving asteroids that already survived previous collisions).

For an asteroid moving to the left (x < 0), we check for possible collisions with right-moving asteroids that are on top of the stack.

We continue to check the top of the stack (stk[-1]) to ensure the top asteroid is moving right (stk[-1] > 0) and is smaller than the current asteroid (stk[-1] < -x).

If both conditions are met, the top asteroid will explode; we remove it by popping it from the stack.

This pop operation is repeated in a loop until the stack's top asteroid is too large to be destroyed (stk[-1] >= -x) or until we find a left-moving asteroid (stk[-1] < 0), indicating that the current asteroid will not collide with any other asteroids in the stack.

If at any point, the asteroid on top of the stack is equal in size to the current one, both asteroids explode. Therefore, we pop the top asteroid from the stack without pushing the current one.

If, after checking for collisions, the stack is empty or the top asteroid is moving to the left, no collision occurs, and we push the current asteroid into the stack.

After processing all the asteroids in the given array, the remaining asteroids in the stack are those that survived all collisions. We return the stk as the final state of asteroids.

By using this approach, we effectively simulate asteroid movement and collisions, leading us to the final answer.

**Example Walkthrough**

Let's apply the solution approach on an example input of asteroids: asteroids = [5, 10, -5].

We initialize an empty list stk to act as our stack. stk = [].

First, we encounter asteroid 5, which is moving to the right (x > 0). We push it onto the stack. Now stk = [5].

Next, we encounter asteroid 10, which is also moving to the right (x > 0). We push it onto the stack as well. Now stk = [5, 10].

Finally, we encounter asteroid -5, which is moving to the left (x < 0). We now check for possible collisions with asteroids on top of the stack:

The top of the stack is 10 which is moving to the right (stk[-1] > 0), but it is larger than our current asteroid (stk[-1] = 10 > |-5| = 5). Therefore, there's no pop operation, as -5 is destroyed in the collision.

We continue and see that there are no more asteroids in the asteroids list and conclude processing.

The remaining asteroids in stk give us the final state after all possible collisions have occurred. In this case, the surviving asteroids are [5,10]. Therefore, our function would return this list.

This example illustrates how the stack is utilized to process the asteroids one by one, simulating the collisions based on their direction and size and leaving us with the final state of the asteroids.

**Solution Implementation**

class Solution {

public int[] asteroidCollision(int[] asteroids) {

// Initialize a stack to keep track of the asteroids

Deque<Integer> stack = new ArrayDeque<>();

// Iterate through the array of asteroids

for (int asteroid : asteroids) {

// If the asteroid is moving to the right, push it onto the stack

if (asteroid > 0) {

stack.offerLast(asteroid);

} else {

// While there are asteroids in the stack moving to the right, and the

// current asteroid's magnitude is larger, pop the asteroids from the stack

while (!stack.isEmpty() && stack.peekLast() > 0 && stack.peekLast() < -asteroid) {

stack.pollLast();

}

// If the top of the stack is an asteroid of the same magnitude but moving in the opposite direction, destroy both

if (!stack.isEmpty() && stack.peekLast() == -asteroid) {

stack.pollLast();

} else if (stack.isEmpty() || stack.peekLast() < 0) {

// If there are no asteroids on the stack or the top asteroid is moving to the left, push the current asteroid onto the stack

stack.offerLast(asteroid);

}

}

}

// Convert the remaining asteroids in the stack to an array

return stack.stream().mapToInt(Integer::valueOf).toArray();

}

}

**Time and Space Complexity**

The time complexity of the provided code is O(n), where n is the length of the array asteroids. This is due to the fact that the algorithm processes each element of the array exactly once in the worst-case scenario. The while loop inside the for loop does not increase the time complexity because it only processes elements that are potentially colliding, and each element can be pushed and popped from the stack at most once.

The space complexity of the code is also O(n). In the worst-case scenario, this happens when all the asteroids are moving in the same direction and there are no collisions. As such, the stk can potentially grow to include all elements of the asteroids array if no asteroids ever collide.

**Refer to**

[L2211.Count Collisions on a Road](note://WEB54fbc019f7cda043b32ce15a1ff3bf88)