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.
```

Example 4:

```
Input:
asteroids = [-2, -1, 1, 2]
Output: [-2, -1, 1, 2]
Explanation:
The -2 and -1 are moving left, while the 1 and 2 are moving right.
Asteroids moving the same direction never meet, so no asteroids will meet each other.
```

Note:

- The length of `asteroids` will be at most `10000`.
- Each asteroid will be a non-zero integer in the range `[-1000, 1000].`.

Solution 1

- at the end, all the negative star has to be on the left, and all the positive star has to be on the right.
- from the left, a negative star will pass through if no positive star on the left;
- keep track of all the positive stars moving to the right, the right most one will be the 1st confront the challenge of any future negative star.
- if it survives, keep going, otherwise, any past positive star will be exposed to the challenge, by being popped out of the stack.

C++

```
class Solution {
public:
    vector<int> asteroidCollision(vector<int>& a) {
        vector<int> s; // use vector to simulate stack.
        for (int i = 0; i < a.size(); i++) {
            if (a[i] > 0) // a[i] is positive star
                s.push_back(a[i]);
            else if (s.empty() || s.back() < 0) // a[i] is negative star and there is no positive on stack
                s.push_back(a[i]); // negative star pass through
            else if (s.back() <= -a[i]) { // a[i] is negative star and stack top is positive star
                if(s.back() < -a[i]) i--; // only positive star on stack top get destroyed, stay on i to check more on stack.
                s.pop_back(); // destroy positive star on the frontier;
            } // else : positive on stack bigger, negative star destroyed.
        }
        return s;
    }
};
```

java

```
class Solution {
    public int[] asteroidCollision(int[] a) {
        LinkedList<Integer> s = new LinkedList<>(); // use LinkedList to simulate stack so that we don't need to reverse at end.
        for (int i = 0; i < a.length; i++) {
            if (a[i] > 0 || s.isEmpty() || s.getLast() < 0)
                s.add(a[i]);
            else if (s.getLast() <= -a[i])
                if (s.pollLast() < -a[i]) i--;
        }
        return s.stream().mapToInt(i->i).toArray();
    }
}
```

written by alexander original link here

Solution 2

How is this test case correct ? [-2,-2,-2,1] Exp Ans : [-2,-2,-2,1].
I lost in this one. For me the right result is [-2,-2,-2] . Can any one help ?

written by mohammedsidik original link here

Solution 3

Not as clean as the given solution, but perhaps easier to understand.

Some observations:

1. Negative asteroids without any positive asteroids on the left can be ignored as they will never interact with the upcoming asteroids regardless of their direction.
2. Positive asteroids (right-moving) may interact with negative asteroids (left-moving) that come later.

We can use a stack called `res` to efficiently simulate the collisions. We can iterate through the list of `asteroids` and handle the following scenarios as such:

1. If `res` is empty, we push that `asteroid` into it regardless of directions. Because negative asteroids will be part of the final result while positive asteroids may interact with future negative asteroids.
2. If the `asteroid` is positive, push it into `res`. It will never interact with existing asteroids in `res` but may interact with future negative asteroids.
3. If the `asteroid` is negative, we need to simulating the collision process by repeatedly popping the positive asteroids from the top of the stack and compare to see which asteroid survives the collision. We may or may not need to push the negative `asteroid` to `res` depending on the value of the positive asteroids it encounters. Push the negative `asteroid` if it survives all the collisions.

- *Yangshun*

```python
class Solution(object):
    def asteroidCollision(self, asteroids):
        res = []
        for asteroid in asteroids:
            if len(res) == 0 or asteroid > 0:
                res.append(asteroid)
            elif asteroid < 0:
                # While top of the stack is positive.
                while len(res) and res[-1] > 0:
                    # Both asteroids are equal, destroy both.
                    if res[-1] == -asteroid:
                        res.pop()
                        break
                    # Stack top is smaller, remove the +ve asteroid
                    # from the stack and continue the comparison.
                    elif res[-1] < -asteroid:
                        res.pop()
                        continue
                    # Stack top is larger, -ve asteroid is destroyed.
                    elif res[-1] > -asteroid:
                        break
                else:
                    # -ve asteroid made it all the way to the
                    # bottom of the stack and destroyed all asteroids.
                    res.append(asteroid)
        return res
```

written by yangshun original link here

From Leetcoder.