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

There are n **1-indexed** robots, each having a position on a line, health, and movement direction.

You are given **0-indexed** integer arrays positions, healths, and a string directions (directions[i] is either **'L'** for **left** or **'R'** for **right**). All integers in positions are **unique**.

All robots start moving on the line**simultaneously** at the **same speed**in their given directions. If two robots ever share the same position while moving, they will **collide**.

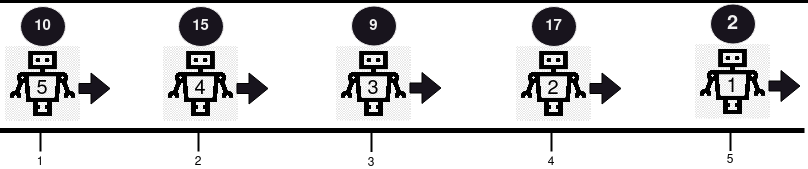
If two robots collide, the robot with **lower health** is **removed** from the line, and the health of the other robot **decreases** **by one**. The surviving robot continues in the **same** direction it was going. If both robots have the **same** health, they are bothremoved from the line.

Your task is to determine the **health** of the robots that survive the collisions, in the same **order**that the robots were given,i.e. final health of robot 1 (if survived), final health of robot 2 (if survived), and so on. If there are no survivors, return an empty array.

Return *an array containing the health of the remaining robots (in the order they were given in the input), after no further collisions can occur.*

**Note:** The positions may be unsorted.

**Example 1:**

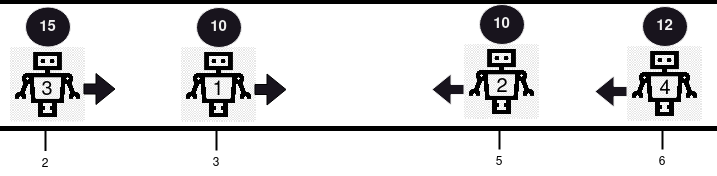


**Input:** positions = [5,4,3,2,1], healths = [2,17,9,15,10], directions = "RRRRR"

**Output:** [2,17,9,15,10]

**Explanation:** No collision occurs in this example, since all robots are moving in the same direction. So, the health of the robots in order from the first robot is returned, [2, 17, 9, 15, 10].

**Example 2:**



**Input:** positions = [3,5,2,6], healths = [10,10,15,12], directions = "RLRL"

**Output:** [14]

**Explanation:** There are 2 collisions in this example. Firstly, robot 1 and robot 2 will collide, and since both have the same health, they will be removed from the line. Next, robot 3 and robot 4 will collide and since robot 4's health is smaller, it gets removed, and robot 3's health becomes 15 - 1 = 14. Only robot 3 remains, so we return [14].

**Example 3:**



**Input:** positions = [1,2,5,6], healths = [10,10,11,11], directions = "RLRL"

**Output:** []

**Explanation:** Robot 1 and robot 2 will collide and since both have the same health, they are both removed. Robot 3 and 4 will collide and since both have the same health, they are both removed. So, we return an empty array, [].

**Constraints:**

1 <= positions.length == healths.length == directions.length == n <= 105

1 <= positions[i], healths[i] <= 109

directions[i] == 'L' or directions[i] == 'R'

All values in positions are distinct

**Attempt 1: 2025-09-08**

**Solution 1: Stack (10 min)**

class Solution {

    public List<Integer> survivedRobotsHealths(int[] positions, int[] healths, String directions) {

        int n = positions.length;

        Integer[] indices = new Integer[n];

        for(int i = 0; i < n; i++) {

            indices[i] = i;

        }

        // Sort indices by position

        Arrays.sort(indices, (a, b) -> positions[a] - positions[b]);

        // Store indices of robots that are moving right or haven't collided yet

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

        for(int index : indices) {

            boolean push = true;

            char direction = directions.charAt(index);

            // Collision will happen only if Previous robot is moving right,

            // current robot is moving left

            while(!stack.isEmpty() && direction == 'L'

                && directions.charAt(stack.peek()) == 'R') {

                // If both robots have the same health, they are both removed from the line

                if(healths[index] == healths[stack.peek()]) {

                    healths[index] = 0; // current robot dies

                    healths[stack.peek()] = 0; // top robot dies

                    stack.pop();

                    push = false;

                    break;

                // Current one has higher health than previous one

                } else if(healths[index] > healths[stack.peek()]) {

                    healths[stack.peek()] = 0; // top robot dies

                    stack.pop();

                    healths[index]--; // current robot decrease health

                    push = true;

                    continue;

                // Current one has lower health than previous one

                } else {

                    healths[stack.peek()]--; // top robot decrease health

                    healths[index] = 0; // current robot dies

                    push = false;

                    break;

                }

            }

            if(push) {

                stack.push(index);

            }

        }

        List<Integer> result = new ArrayList<>();

        for(int i = 0; i < n; i++) {

            if(healths[i] > 0) {

                result.add(healths[i]);

            }

        }

        return result;

    }

}

Time Complexity: O(n log n) for sorting, and each robot is pushed and popped at most once.

Space Complexity: O(n) for the stack and sorting.

**Refer to Deepseek**

I'll provide a Java solution for LeetCode 2751. "Robot Collisions", along with an explanation and complexity analysis.

**Problem Description**

There are n robots in a line. Each robot has:

A position position[i]

A health health[i]

A direction direction[i] (either 'L' for left or 'R' for right)

All robots move at the same speed (1 unit per second). When two robots collide:

The robot with lower health is removed.

The robot with higher health loses 1 health point.

If both have the same health, both are removed.

Robots moving in the same direction never collide. Only robots moving towards each other (one 'R' and one 'L') can collide.

Return the list of remaining robots in the order of their original positions.

**Java Solution**

import java.util.\*;

class Solution {

public List<Integer> survivedRobotsHealths(int[] positions, int[] healths, String directions) {

int n = positions.length;

Integer[] indices = new Integer[n];

for (int i = 0; i < n; i++) indices[i] = i;

// Sort indices by position

Arrays.sort(indices, (a, b) -> positions[a] - positions[b]);

Stack<Integer> stack = new Stack<>(); // store indices of robots that are moving right or haven't collided yet

for (int i : indices) {

if (directions.charAt(i) == 'R') {

stack.push(i);

} else { // current robot is moving left

while (!stack.isEmpty() && directions.charAt(stack.peek()) == 'R') {

int top = stack.pop();

if (healths[top] > healths[i]) {

healths[top]--;

stack.push(top);

healths[i] = 0; // current robot dies

break;

} else if (healths[top] < healths[i]) {

healths[i]--;

healths[top] = 0; // top robot dies

// continue to check next top

} else { // equal health

healths[top] = 0;

healths[i] = 0;

break;

}

}

if (healths[i] > 0) {

stack.push(i);

}

}

}

// Collect results in original order

List<Integer> result = new ArrayList<>();

for (int i = 0; i < n; i++) {

if (healths[i] > 0) {

result.add(healths[i]);

}

}

return result;

}

}

**Explanation**

**Sort by Position**:

We need to process robots from left to right. Create an array of indices and sort them by the robots' positions.

**Use a Stack**:

The stack stores indices of robots that are moving right or haven't collided yet.

For each robot (in sorted order by position):

If it is moving right ('R'), push it to the stack.

If it is moving left ('L'), it may collide with the top of the stack (which is moving right).

Handle collisions:

While the stack is not empty and the top robot is moving right:

If the top robot has more health: top loses 1 health, current robot dies (health=0). Push top back and break.

If the current robot has more health: current loses 1 health, top robot dies. Continue to check the next top.

If equal health: both die and break.

If the current robot survives after all collisions, push it to the stack.

**Collect Results**:

After processing, collect all robots with health > 0 in the original order (by index).

**Example Walkthrough**

Let:

positions = [5, 10, 15]

healths = [3, 2, 1]

directions = "RLR"

Sort indices by position: [0,1,2] (already sorted).

Process:

i=0 (pos=5, health=3, dir='R'): push to stack → stack=[0]

i=1 (pos=10, health=2, dir='L'): collides with top (index0, dir='R').

Compare health: top=3 > current=2 → top health becomes 2, current dies. Push top back → stack=[0] (with health=2).

i=2 (pos=15, health=1, dir='R'): push to stack → stack=[0,2]

Result: healths = [2,0,1] → robots with health>0: [2,1] (but wait, we need to return in original order? Actually, we return the healths of surviving robots in the order of original positions. So for indices 0 and 2: healths are 2 and 1 → result = [2,1].

**Complexity Analysis**

**Time Complexity**: O(n log n) for sorting, and each robot is pushed and popped at most once.

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

This solution efficiently handles collisions using a stack and processes robots in sorted order. Let me know if you need further clarification!

**Refer to**

[L735.Asteroid Collision (Ref.L2211)](note://WEB88a08f8ad07d5d7ac8a64605312c92a6)