프로그래머스 - 최댓값 만들기 (2)

태그

.sort((a b) => a - b)는 오름차순기법

Arrays.sort(numbers)도 오름차순기법

배열을 한번만 돌면 시간복답도가 빅오 시간복잡도고려

자바스크립트
최댓값 만들기 (2)
문제 설명
정수 배열 numbers가 매개변수로 주어집니다. numbers의 원소 중 두 개를 곱해 만들 수 있는 최댓값을 return 하도록 solution 함수를 완성해주세요.

제한사항
-10,000 ≤ numbers의 원소 ≤ 10,000
2 ≤ numbers 의 길이 ≤ 100
입출력 예 #1
두 수의 곱중 최댓값은 -3 * -5 = 15 입니다.
numbers: [1, 2, -3, 4, -5] , result: 15
입출력 예 #2
numbers: [0, -31, 24, 10, 1, 9] , result:240
두 수의 곱중 최댓값은 10 * 24 = 240 입니다.
입출력 예 #3
numbers: [10, 20, 30, 5, 5, 20, 5] , result: 600
두 수의 곱중 최댓값은 20 * 30 = 600 입니다.

```
function solution(numbers) {
  let max = -Infinity;
  let secondMax = -Infinity;
  let min1 = Infinity;
  let min2 = Infinity;

  for (const num of numbers) {
    if (num > max) {
      secondMax = max;
      max = num;
    } else if (num > secondMax) {
      secondMax = num;
    }
    if (num < min1) {
      min2 = min1;
      min1 = num;
    } else if (num < min2) {
      min2 = num;
    }
  }
```

```
  return Math.max(max * secondMax, min1 * min2);
}
```

배열에서 최대값과 두 번째 최대값, 최소값과 두 번째 최소값을 찾습니다. 그런 다음 최대값과 두 번째 최대값의 곱 중 최대값과 최소값과 두 번째 최소값의 곱을 반환합니다. 이렇게 하면 양수와 음수가 모두 포함된 배열에 대해 솔루션이 올바르게 작동합니다.

배열을 한 번만 반복하면 되므로 시간 복잡도는 O(n)입니다.

```
function solution(numbers) {
    numbers.sort((a, b) => a - b);
    return Math.max(numbers[0]*numbers[1], numbers[numbers.length-1]*numbers[numbers.l
ength-2]);
}

function solution(numbers) {
    var answer = [];
    for(let i = 0; i < numbers.length - 1; i++){
        for(let j = i + 1; j < numbers.length; j++){
            answer.push(numbers[i] * numbers[j]);
        }
    }
    return Math.max(...answer);
}
```

배열에서 가장 높은 두 값만 고려하지만 가장 높은 곱이 두 음수를 곱한 결과일 수 있는 가능성은 고려하지 않는다는 것입니다.

첫 번째 솔루션은 배열을 오름차순으로 정렬한 다음 처음 두 요소의 곱과 마지막 두 요소의 곱의 최대값을 반환합니다. 배열을 정렬하는 데 O(n * log(n)) 시간이 걸리므로 이 솔루션의 시간 복잡도는 O(n * log(n))입니다.

두 번째 솔루션은 중첩 루프를 사용하여 배열의 모든 요소 쌍을 반복하고 각 쌍의 곱을 계산합니다. 그런 다음 이러한 모든 제품의 최대값을 반환합니다. 이 솔루션의 시간 복잡도는 O(n^2)입니다. 최악의 경우 루프가 O(n^2)번 실행되기 때문입니다.

```
import java.util.Arrays;

class Solution {
    public int solution(int[] numbers) {
        Arrays.sort(numbers);
        return numbers[numbers.length-1] * numbers[numbers.length-2] < numbers[0] * nu
mbers[1] ?
            numbers[0] * numbers[1] : numbers[numbers.length-1] * numbers[numbers.leng
th-2];
    }
}
```

```
import java.util.*;

class Solution {
    public int solution(int[] numbers) {
        int len = numbers.length;
        Arrays.sort(numbers);
        return Math.max(numbers[0] * numbers[1], numbers[len - 2] * numbers[len - 1]);
    }
}
```

첫 번째 솔루션은 배열을 오름차순으로 정렬한 다음 처음 두 요소의 곱과 마지막 두 요소의 곱의 최대값을 반환합니다. 배열을 정렬하는 데 O(n * log(n)) 시간이 걸리므로 이 솔루션의 시간 복잡도는 O(n * log(n))입니다.

두 번째 솔루션은 배열을 오름차순으로 정렬한 다음 처음 두 요소의 곱과 마지막에서 두 번째 및 마지막 요소의 곱의 최대값을 반환합니다. 이 솔루션은 또한 배열을 정렬하는 데 O(n * log(n)) 시간이 걸리므로 O(n * log(n))의 시간 복잡도를 갖습니다.