JavaScript Merge Intervals

Challenge

Given an array of `intervals` where `intervals[i] = [starti, endi]`, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

1st Example

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6]
             overlap, merge them into [1,6].
```

2nd Example

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5]
             are considered overlapping.
```

Constraints

- `1 <= intervals.length <= 10^4`

- `intervals[i].length == 2`
- `0 <= starti <= endi <= 10^4`

Solution

```
const merge = (intervals) => {
    intervals.sort((a, b) => a[0] - b[0]);

    let i = 0;

    while (i < intervals.length - 1) {
        const [_, firstRight]           = intervals[i],
              [secondLeft, secondRight] = intervals[i + 1];

        if (firstRight >= secondLeft) {
            intervals[i][1] = Math
                              .max(firstRight, secondRight);
            intervals.splice(i + 1, 1);
        } else {
            i++;
        }
    }

    return intervals;
};
```

Explanation

I've coded a function called `merge` that takes an array of intervals as input and returns the merged intervals.

The function starts by sorting the `intervals` array in ascending order based on the first element of each interval. This is done using the `sort` method and a comparator function that compares the first elements of two intervals.

A variable `i` is initialized to `0`, which will be used as an index to iterate through the `intervals` array.

The function enters a `while` loop that will continue until `i` reaches the second-to-last index of the `intervals` array.

Inside the loop, the current interval and the next interval are extracted and assigned to variables using destructuring assignment. The first element of the current interval is ignored (assigned to `_`), and the second element is assigned to `firstRight`. The first element of the next interval is assigned to `secondLeft`, and the second element is assigned to `secondRight`.

A conditional statement checks if the `firstRight` value is greater than or equal to the `secondLeft` value. This condition determines if the two intervals can be merged.

If the condition is true, the function updates the end value of the current interval to the maximum of `firstRight` and `secondRight` using the `Math.max` function. This effectively merges the two intervals.

The `splice` method is used to remove the next interval from the `intervals` array, starting from the index `i+1`. This is done to eliminate the duplicate interval that was merged.

If the condition in the previous step is false, meaning the two intervals cannot be merged, the function increments the index `i`

by `1` to move to the next pair of intervals.

After the while loop finishes, the modified `intervals` array is returned as the result of the function.

In summary, this function merges overlapping intervals in the input array. It sorts the intervals based on the start values and then iterates through the sorted array, merging intervals when necessary. The resulting merged intervals are returned as the output of the function.

Author: Trevor Morin