Documentation for Insert Interval Algorithm

Overview

The "Insert Interval" algorithm is designed to insert a new interval into a list of non-overlapping intervals while maintaining the sorted order and ensuring that no intervals overlap after insertion. This algorithm takes two main inputs: a list of non-overlapping intervals, sorted in ascending order by start value, and a new interval to be inserted into this list. The algorithm returns the updated list of intervals after the insertion.

Problem Statement

Given a list `intervals` of non-overlapping intervals, represented as pairs of start and end values, and a new interval `newInterval`, the task is to insert `newInterval` into `intervals` such that the resulting list is still sorted by start values and contains no overlapping intervals.

Example

Input:

intervals = [[1,3],[6,9]]

newInterval = [2,5]

Output:

[[1,5],[6,9]]

Explanation:

The new interval `[2,5]` overlaps with `[1,3]`, so it merges with it to form `[1,5]`. The resulting intervals become `[[1,5],[6,9]]`.

Algorithm

1. **Initialization:** Initialize an empty list `result` to store the updated intervals. Set `i` to 0, representing the current index in the intervals list, and `n` to the length of the intervals list.
2. **Add Intervals Before New Interval:** Iterate through the intervals list until reaching an interval whose end value is less than the start value of the new interval. Append each interval encountered to the `result` list.

3. **Merge Overlapping Intervals:** While iterating through the intervals list, if the start value of the current interval is less than or equal to the end value of the new interval, merge the intervals. Update the start value of the new interval to the minimum of its current start value and the start value of the current interval. Update the end value of the new interval to the maximum of its current end value and the end value of the current interval.

4. **Add Merged New Interval:** After merging all overlapping intervals, append the merged new interval to the `result` list.

5. **Add Remaining Intervals:** Append the remaining intervals in the original intervals list to the `result` list.

6. **Return Result:** Return the `result` list containing the updated intervals.

Complexity Analysis

- **Time Complexity:** The algorithm traverses the intervals list once, performing constant time operations for each interval. Hence, the time complexity is O(n), where n is the number of intervals in the list.

- **Space Complexity:** The space complexity is O(n) since the algorithm uses additional space to store the merged intervals in the `result` list.