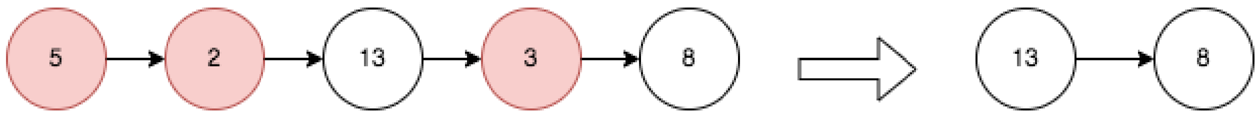
Q1.You are given the head of a linked list. Remove every node which has a node with a greater value anywhere to the right side of it. Return the head of the modified linked list.

Example 1:



Input: head = [5,2,13,3,8]
Output: [13,8]

Explanation: The nodes that should be removed are 5, 2 and 3.

- Node 13 is to the right of node 5.
- Node 13 is to the right of node 2.
- Node 8 is to the right of node 3.

Example 2:
Input: head = [1,1,1,1]
Output: [1,1,1,1]

Explanation: Every node has value 1, so no nodes are removed.

Constraints:

- The number of the nodes in the given list is in the range [1, 105].
- 1 <= Node.val <= 105

Q2.You are given an absolute path for a Unix-style file system, which always begins with a slash '/'. Your task is to transform this absolute path into its simplified canonical path.

The rules of a Unix-style file system are as follows:

- A single period '.' represents the current directory.
- A double period '..' represents the previous/parent directory.
- Multiple consecutive slashes such as '//' and '///' are treated as a single slash '/'.
- Any sequence of periods that does not match the rules above should be treated as a valid directory or file name. For example, '...' and '   ' are valid directory or file names.

The simplified canonical path should follow these rules:

- The path must start with a single slash '/'.
- Directories within the path must be separated by exactly one slash '/'.
- The path must not end with a slash '/', unless it is the root directory.
- The path must not have any single or double periods ('.' and '..') used to denote current or parent directories.

Return the simplified canonical path.

Example 1:
Input: path = "/home/"
Output: "/home"

Explanation:
The trailing slash should be removed.

Example 2:
Input: path = "/home//foo/"
Output: "/home/foo"

Explanation:
Multiple consecutive slashes are replaced by a single one.

Example 3:
Input: path = "/home/user/Documents/../Pictures"
Output: "/home/user/Pictures"

Explanation:
A double period ".." refers to the directory up a level (the parent directory).

Example 4:
Input: path = "/../"
Output: "/"
Explanation:
Going one level up from the root directory is not possible.

Example 5:
Input: path = "/.../a/../b/c/../d/./"
Output: "/.../b/d"

Explanation:
"..." is a valid name for a directory in this problem.

Constraints:

- 1 <= path.length <= 3000
- path consists of English letters, digits, period '.', slash '/' or '_'.
- path is a valid absolute Unix path.

Q 3. You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position. Return true if you can reach the last index, or false otherwise.

Example 1:
Input: nums = [2,3,1,1,4]
Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:
Input: nums = [3,2,1,0,4]
Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints:

- 1 <= nums.length <= 104
- 0 <= nums[i] <= 105

Q4. **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.**

Example 1:
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].

Example 2:
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.

Constraints:

- 1 <= intervals.length <= 104
- intervals[i].length == 2
- 0 <= starti <= endi <= 104

Q 5.You are given an array of non-overlapping intervals intervals where intervals[i] = [starti, endi] represent the start and the end of the ith interval and intervals is sorted in ascending order by starti. You are also given an interval newInterval = [start, end] that represents the start and end of another interval.
Insert newInterval into intervals such that intervals are still sorted in ascending order by starti and intervals still do not have any overlapping intervals (merge overlapping intervals if necessary). Return intervals after the insertion.
Note that you don't need to modify intervals in-place. You can make a new array and return it.

Example 1:
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]

Example 2:
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

Output: [[1,2],[3,10],[12,16]]

Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

Constraints:

- 0 <= intervals.length <= 104
- intervals[i].length == 2
- 0 <= starti <= endi <= 105
- intervals is sorted by starti in ascending order.
- newInterval.length == 2
- 0 <= start <= end <= 105