Next Smaller Element of Array Element CodeStudio

Given an array of integers, for each element, find the next smaller element in the array. If there is no smaller element to the right of an element, mark it as **-1**.

Example

Consider the following array:

Array: [2, 4, 9, 3, 1]

Output: [1, 3, 3, 1, -1]

Approach 1: Function to find the next smaller elements using a brute force approach

In this approach, for each element in the array, we iterate through the remaining elements to find the next smaller element. We initialize the result array with **-1** assuming that no smaller element is found. If a smaller element is found while iterating, we update the result array accordingly.

Steps:

1. Initialize an array **ans** of size **n**, where **n** is the size of the input array. Initialize all elements of **ans** as **-1**.
2. For each element **arr[i]** in the input array:
    - Iterate through the elements starting from **i+1** to the end of the array.
    - If an element **arr[j]** is smaller than **arr[i]**, update **ans[i]** with **arr[j]** and break from the loop.
3. The **ans** array will now hold the next smaller elements for each element.

Time Complexity:

- For each element in the array, the approach involves iterating through the remaining elements.
- **Overall, the time complexity is approximately O(n^2), where n is the size of the input array.**

Space Complexity:

- The space complexity is determined by the additional **ans** array.
- **Space Complexity**: **O(n), where n is the size of the input array.**

Approach 2: Function to find the next smaller elements using a stack-based approach

In this approach, we use a stack to keep track of elements. We iterate through the array in reverse order. For each element **arr[i]**, we pop elements from the stack until we find a smaller element. The stack holds the elements that have not yet found their next smaller element.

Steps:

1. Initialize an empty stack and an array **ans** of size **n**, where **n** is the size of the input array.
2. Iterate through the array in reverse order (from the last element to the first element):
    - Pop elements from the stack until finding a smaller element.
    - If a smaller element is found in the stack, update **ans[i]** with that element.
    - Push the current element onto the stack.
3. The **ans** array will now hold the next smaller elements for each element.

Time Complexity:

- The approach involves iterating through the array once and performing stack operations.
- The stack operations are constant time.
- **Time Complexity**: **O(n), where n is the size of the input array.**

Space Complexity:

- The space complexity is determined by the stack's maximum size, which depends on the number of elements that have not found their next smaller element.
- In the worst case, the stack could hold all elements.
- **Space Complexity**: **O(n), where n is the size of the input array.**