Sum of Subarray Minimums

Problem Statement: Given an array of integers arr of size n, calculate the sum of the minimum value in each (contiguous) subarray of arr. Since the result may be large, return the answer modulo $10^9 +7$.

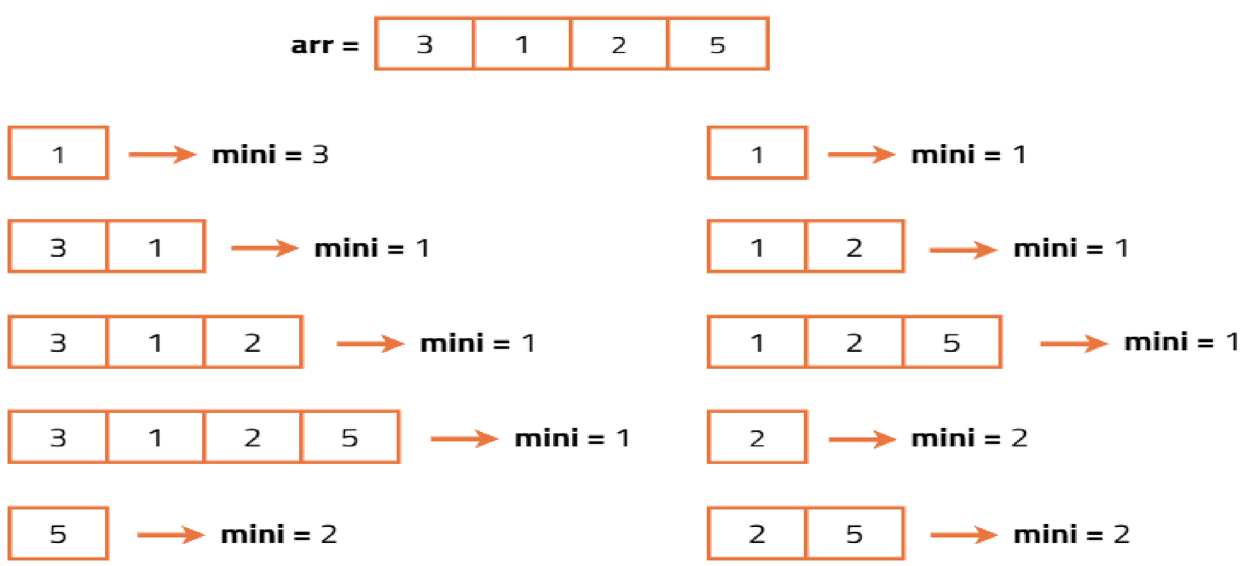
Examples

```
Example 1:
Input: arr = [3, 1, 2, 5]
Output: 18
Explanation: The minimum of subarrays: [3], [1], [2], [5], [3, 1], [1, 2], [2, 5], [3, 1, 2], [1, 2, 5], [3, 1, 2, 5] are 3, 1, 2, 5, 1, 1, 2, 1, 1, 1 respectively and their sum is 18.

Example 2:
Input: arr = [2, 3, 1]
Output: 10
Explanation: The minimum of subarrays: [2], [3], [1], [2,3], [3,1], [2,3,1] are 2, 3, 1, 2, 1, 1 respectively and their sum is 10.
```

Brute Force
Algorithm

- Initialize a variable to hold the total sum, starting from 0
- Start a loop to fix the starting index of the subarray
- Initialize a variable to keep track of the minimum element in the current subarray
- Use an inner loop to extend the subarray to the right
- Update the minimum element as the subarray grows
- Add the current minimum to the total sum
- Repeat this process for all possible subarrays
- Return the total sum after all subarrays are processed



Code

```java
import java.util.*;

class Solution {

    // Function to find the sum of the minimum value in each subarray
    public int sumSubarrayMins(int[] arr) {
        // Size of the array
        int n = arr.length;

        // Modulo value to prevent integer overflow
        int mod = (int)1e9 + 7;

        // Variable to store the total sum
        int sum = 0;

        // Traverse each starting index of subarrays
        for (int i = 0; i < n; i++) {
            // Initialize the minimum as the current element
            int mini = arr[i];

            // Traverse all subarrays starting at index i
            for (int j = i; j < n; j++) {
                // Update the minimum in the current subarray
                mini = Math.min(mini, arr[j]);

                // Add the current minimum to the total sum
                sum = (sum + mini) % mod;
            }
        }

        // Return the total computed sum
        return sum;
    }
}

// Separate class containing the main method
public class Main {
    public static void main(String[] args) {
        // Input array
        int[] arr = {3, 1, 2, 5};

        // Create instance of Solution
        Solution sol = new Solution();

        // Call the function to get the sum of minimums
        int ans = sol.sumSubarrayMins(arr);

        // Print the result
        System.out.println("The sum of minimum value in each subarray is: " +
ans);
    }
}
```

Complexity Analysis

Time Complexity: O(N^2), since we are using two nested loops.

Space Complexity: O(1), as we are not using any extra space except for the input array and a few variables.

Optimal Approach

Algorithm

- Use a stack to find the index of the next smaller element to the right for each position
- Use another stack to find the index of the previous smaller or equal element to the left for each position
- For each element, determine how many subarrays it appears in as the minimum using its NSE and PSEE indices
- Calculate the contribution of each element by multiplying its value with its frequency
- Add each contribution to a total sum
- Return the total sum modulo 10^9 + 7

Code

```
import java.util.*;

class Solution {

    // Function to find indices of Next Smaller Element (NSE)
    private int[] findNSE(int[] arr) {
        int n = arr.length;
        int[] ans = new int[n];
        Stack<Integer> st = new Stack<>();

        // Traverse array from right to left
        for (int i = n - 1; i >= 0; i--) {
            // Pop elements that are greater or equal to current
            while (!st.isEmpty() && arr[st.peek()] >= arr[i]) {
                st.pop();
            }

            // If stack is empty, NSE doesn't exist → set to n
            ans[i] = !st.isEmpty() ? st.peek() : n;

            // Push current index to stack
            st.push(i);
        }

        // Return NSE indices
        return ans;
    }

    // Function to find indices of Previous Smaller or Equal Element (PSEE)
    private int[] findPSEE(int[] arr) {
        int n = arr.length;
        int[] ans = new int[n];
        Stack<Integer> st = new Stack<>();

        // Traverse array from left to right
        for (int i = 0; i < n; i++) {
            // Pop elements greater than current
            while (!st.isEmpty() && arr[st.peek()] > arr[i]) {
                st.pop();
            }

            // If stack is empty, PSEE doesn't exist → set to -1
            ans[i] = !st.isEmpty() ? st.peek() : -1;
```

```
            // Add contribution to sum
            sum = (sum + val) % mod;
        }

        // Return the final sum
        return sum;
    }
}

// Separate class to hold the main method
public class Main {
    public static void main(String[] args) {
        // Input array
        int[] arr = {3, 1, 2, 5};

        // Create an instance of Solution
        Solution sol = new Solution();

        // Call the optimized function
        int ans = sol.sumSubarrayMins(arr);

        // Print the result
        System.out.println("The sum of minimum value in each subarray is: " +
ans);
    }
}
```

Complexity Analysis

Time Complexity: O(N), since finding the indices of next smaller elements and previous smaller elements take O(2*N) time each and calculating the sum of subarrays minimum takes O(N) time.

Space Complexity: O(N), since finding the indices of the next smaller elements and previous smaller elements takes O(N) space each due to stack space and storing the indices of the next smaller elements and previous smaller elements takes O(N) space each.