Problem: Strictly Increasing Beauty Partition

You are given an array A of size N. You need to partition it into contiguous, non-empty subarrays such that the beauty of each subarray forms a strictly increasing sequence.

Beauty of a subarray is defined as the sum of its elements.

Return the total number of such valid partitions, modulo 10^9 + 7.

Input:
- int N: the number of elements in array A
- List<Integer> A: the array elements

Constraints:
- 1 <= N <= 1000
- -10^9 <= A[i] <= 10^9

Output:
- Integer: Number of valid partitions modulo 10^9 + 7

Example:
Input: A = [1, 2, 3]
Output: 3

Explanation:
Valid partitions are:
- [1, 2, 3] => beauty: [6]
- [1], [2, 3] => beauty: [1, 5]
- [1], [2], [3] => beauty: [1, 2, 3]

```
import java.util.*;

public class Main {
    static final int MOD = 1_000_000_007;

    public static int getAnswer(int N, List<Integer> A) {
        long[] prefix = new long[N + 1];
        for (int i = 0; i < N; i++) {
            prefix[i + 1] = prefix[i] + A.get(i);
        }

        Map<Integer, TreeMap<Long, Integer>> memo = new HashMap<>();
        return dfs(0, Long.MIN_VALUE, A, prefix, memo);
    }

    private static int dfs(int start, long prevSum, List<Integer> A, long[] prefix,
                           Map<Integer, TreeMap<Long, Integer>> memo) {
        if (start == A.size()) return 1;

        TreeMap<Long, Integer> memAtStart = memo.getOrDefault(start, new TreeMap<>());
        if (memAtStart.containsKey(prevSum)) return memAtStart.get(prevSum);

        int count = 0;
```

```
        for (int end = start + 1; end <= A.size(); end++) {
            long currSum = prefix[end] - prefix[start];
            if (currSum > prevSum) {
                count = (count + dfs(end, currSum, A, prefix, memo)) % MOD;
            }
        }

        memAtStart.put(prevSum, count);
        memo.put(start, memAtStart);
        return count;
    }

    public static void main(String[] args) {
        System.out.println(getAnswer(3, Arrays.asList(1, 2, 3))); // 3
        System.out.println(getAnswer(4, Arrays.asList(1, 1, 1, 1))); // 5
                  System.out.println(getAnswer(3,  Arrays.asList(1000000000,  1000000000,
1000000000))); // 3
        System.out.println(getAnswer(2, Arrays.asList(-5, 10))); // 2
    }
}
```