Course Name:	Competitive Programming Laboratory (216U01L401)	Semester:	IV
Date of Performance:	05 / 02 / 2025	DIV/ Batch No:	E-2
Student Name:	Shreyans Tatiya	Roll No:	16010123325

Experiment No: 1

Title: To implement a competitive programming problem on a platform (e.g., Leetcode) optimized using arrays, hash tables, and a divide-and-conquer approach.

Aim and Objective of the Experiment:

- Understand the concepts of array, hash tables, divide and conquer approach
- Implement the code on leetcode or similar platform

COs to be achieved:

CO2: Analyse and optimize algorithms using amortized analysis and bit manipulation, equipping them to tackle complex computational problems.

Books/ Journals/ Websites referred:

1. https://algo.monster/liteproblems/891

Theory:

The width of a sequence, defined as the difference between its maximum and minimum elements, is a fundamental measure of variability, similar to the **range** in statistics. This problem involves combinatorial enumeration, as an array of size n has 2^n possible subsequences. Since each element contributes differently as a maximum or minimum across multiple subsequences, the solution relies on sorting and power functions to efficiently compute contributions. The use of **modular arithmetic** (mod 10^9 + 7) prevents overflow and ensures computational efficiency. This problem highlights key concepts in mathematical optimization, combinatorics, and number theory, often applied in algorithm design for large datasets.

Problem statement

The width of a sequence is the difference between the maximum and minimum elements in the sequence. Given an array of integers nums, return the sum of the widths of all the non-empty subsequences of nums. Since the answer may be very large, return it modulo 109 + 7.

A subsequence is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements.

Code :

```
class Solution {

public:
    int sumSubseqWidths(vector<int>& nums) {

        sort(nums.begin(),nums.end());
        long mod=1e9+7, n=nums.size(), power=1, res=0;

        for(int i=0;i<n;i++){
            res=(res+nums[i]*power)%mod;
            power=(power*2)%mod;
        }

        power=1;
        for(int i=n-1;i>=0;i--){
            res=(res-nums[i]*power+mod)%mod;
            power=(power*2)%mod;
        }
        return (res+mod)%mod;
    }
};
```

Test Cases

Case 1
[2,1,3]

Case 2
[2]

Case 3
[3,1,6,2,2,7]

Output :

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[2,1,3]

Output

6

Expected

6

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[2]

Output

0

Expected

0

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[3,1,6,2,2,7]

Output

246

Expected

246

Conclusion:

This problem shows how sorting, combinatorics, and modular arithmetic help efficiently compute the sum of widths for all subsequences. It highlights the need for optimized algorithms to handle large inputs while keeping calculations manageable.