Documentation

Intuition

The problem involves identifying the largest subset of integers where every pair satisfies the divisibility condition. Sorting the input array is the key to simplifying the process since smaller numbers are always checked against larger ones for divisibility. This allows us to build the solution iteratively while ensuring the divisibility condition is met throughout the subset.

Approach

The sorted array forms the foundation for our dynamic programming approach. We use a dp array to keep track of the size of the largest divisible subset that ends at each index. Simultaneously, a parent array is used to store the previous index in the subset, enabling efficient reconstruction of the subset after identifying its size.

Dynamic Programming Mechanics

For each element in the sorted array, we compare it with all preceding elements. If a preceding element divides the current element without a remainder, we consider extending the subset ending at the preceding element. The dp array is updated to reflect the largest subset size found so far. The parent array ensures that we can later backtrack to extract the subset.

Subset Reconstruction

Once the dp array is fully populated, the largest value in the array indicates the size of the largest subset. The corresponding index in the parent array helps in tracing back the elements of the subset. Starting from this index, we traverse the parent array to gather the elements of the subset and reverse it to maintain the ascending order.

Optimization Techniques

Sorting the array simplifies the divisibility checks, making it unnecessary to consider unordered pairs. Using the parent array alongside the dp array ensures efficient subset reconstruction without requiring additional computation or nested loops.

Complexity Considerations

The time complexity is dominated by the nested loops that update the dp array, which operate in $O(n^2)$. Sorting the array contributes an additional O(nlogn). The space complexity is O(n), accounting for the dp, parent, and subset storage.

Practical Applications

This versatile method can be applied to solve similar problems involving subsets with specific conditions. It also demonstrates the power of dynamic programming in handling combinatorial problems efficiently, ensuring both correctness and performance.