Longest Increasing Subsequence – Write Up.

Program Overview:

The program aims to find the longest increasing subsequence from a list of numbers entered by the user. A longest increasing subsequence is a subsequence (not necessarily contiguous) of the given sequence in which the subsequence's elements are in sorted order, and the subsequence is as long as possible.

Implementation:

1. **Input Handling:**
    - The program first prompts the user to enter the number of elements in the list.
    - Then, the user is asked to input the elements of the list one by one.
2. **Finding the Longest Increasing Subsequence:**
    - The program uses dynamic programming to find the longest increasing subsequence efficiently.
    - It maintains a list of lists, where each inner list represents a potential increasing subsequence ending at the current index.
    - For each element in the input list, the program checks all previous elements to find the longest increasing subsequence ending at that element.
    - It compares the lengths of these subsequences and updates the longest increasing subsequence found so far.
3. **Output:**
    - Once the longest increasing subsequence is found, the program displays it to the user.

Conclusion:

The program efficiently addresses the task of finding the longest increasing subsequence from a user-provided list of numbers. Through dynamic programming, it optimally computes the lengths of potential increasing subsequences, ultimately identifying the longest one. This approach ensures that the program can handle large input sizes while maintaining a reasonable runtime.

By offering a user-friendly interface for input handling and providing clear output, the program enhances usability and facilitates interaction. Its ability to efficiently handle dynamic input allows users to explore different scenarios easily.

Overall, this program demonstrates an effective implementation of dynamic programming concepts to solve a real-world problem, showcasing its applicability in various contexts where the identification of longest increasing subsequences is essential.

Algorithm: -

1. **Input:** Receive a list of numbers as input.
2. **Initialization:**
    - Create an array **dp** of length equal to the number of elements in the input list. Each element of **dp** will store the length of the longest increasing subsequence ending at that index.
    - Initialize all elements of **dp** to 1 since the longest increasing subsequence for each element initially is 1 (the element itself).
3. **Iterative Process:**
    - Iterate through each element **num** in the input list.
    - For each element **num**, iterate through the previous elements in the input list.
    - For each previous element **prev**, if **num** is greater than **prev** and the length of the increasing subsequence ending at **prev** plus 1 is greater than the length of the increasing subsequence ending at **num**, update **dp[num]** to be the length of the increasing subsequence ending at **prev** plus 1.
    - After updating **dp[num]**, keep track of the maximum length of the increasing subsequence found so far.
4. **Backtracking:**
    - Once the lengths of all increasing subsequences ending at each index are calculated, find the index **endIndex** with the maximum value in the **dp** array. This index represents the end of the longest increasing subsequence.
    - Initialize an empty list **lis** to store the longest increasing subsequence.
    - Starting from **endIndex**, iterate backward through the input list.
    - For each index **i**, if the length of the increasing subsequence ending at **i** is equal to the value at **dp[endIndex] - 1**, add **nums[i]** to the beginning of **lis**.
    - Decrement **endIndex** by 1 and repeat the above step until **endIndex** becomes 0.
5. **Output:** Return the list **lis**, which represents the longest increasing subsequence found.

This algorithm efficiently finds the longest increasing subsequence by iteratively updating the lengths of increasing subsequences ending at each index and then backtracking to reconstruct the longest increasing subsequence.