1 初始思路

1.1 数组定义

假设 num 为一维线性数组，存放总序列，size_of_num = n。
定义一维数组 dp，第 i 个位置用于储存 num[1:i] 的 LIS 长度。

1.2 状态转移方程

dp[i] 应该从前 $i-1$ 个状态进行更新，遍历前 $i-1$ 个元素（下标用 k 表示）。如果发现 num[i] > num[k]，那么可以构成一个递增数列，对 dp[i] 进行更新，每次更新时保留最大值，即：

$$\texttt{dp[i]} = \max\{\texttt{dp[i]}, \texttt{dp[k]} + 1\}$$

1.3 伪代码

```
SET n = size_of_num
FOR i = 1 TO n DO
    dp[i] = 1  // 初始化，一个独立数字可以构成长度为1的递增序列
FOR i = 1 TO n DO
    FOR k = 1 TO i-1 DO
        IF num[i] > num[k] THEN
            dp[i] = MAX(dp[i], dp[k] + 1)
result = dp[n,n]
PRINT(result)
```

Listing 1: O(N^2) 解法

1.4 缺陷与修改思路

-该算法时间复杂度为 $O(N^2)$。
-分析发现，对前 $i-1$ 个 num 进行遍历时使用了比较大小操作，是否可以通过让前 $i-1$ 个 num 有序，并使用二分查找优化到 $O(N \log N)$?

2 改进思路

分析发现，更新递增子序列长度，需要比较原有递增子序列的尾数大小，若有两个原有递增子序列，它们的长度都相同，那么可以“贪心地”只考虑尾数更小的那一个例如：7，2，10，9，3，4。遍历到 4 时，前面原有的最长递增子序列有多个：7，10；7，9；2，10；2，9；2，3。此时只需保留尾数最小的 2，3 即可。

2.1 优化方案

定义：一维数组 tails，第 i 位储存当前指针之前，长度为 i 的所有递增子序列尾数的最小值。
状态转移：对于遍历指针所指向的数，如果它比 tails 中所有的数都大，则直接加到 tails 的末尾；否则，用它替换 tails 中第一个大于它的数。

这样可以保持 tails 的升序性质，每次查找时可以使用二分查找。

2.2 伪代码

```
SET n = size_of_num
DECLARE tails = []  // 初始化空数组
FOR i = 1 TO n Do
    // 二分查找 tails 中第一个比 num[i] 大的数
    pos = BinarySearch(tails, num[i])
    IF pos == len(tails) Then
        tails.append(num[i])  // 若所有数都比 num[i] 小，将其加入
    ELSE
        tails[pos] = num[i]  // 替换更大的值
result = len(tails)
PRINT(result)
```

Listing 2: O(NlogN) 解法