题目：

300. 最长上升子序列

难度 中等 1033

给定一个无序的整数数组，找到其中最长上升子序列的长度。

示例:

```
输入: [10,9,2,5,3,7,101,18]
输出: 4
解释: 最长的上升子序列是 [2,3,7,101]，它的长度是 4。
```

说明:

- 可能会有多种最长上升子序列的组合，你只需要输出对应的长度即可。
- 你算法的时间复杂度应该为 $O(n^2)$ 。

进阶: 你能将算法的时间复杂度降低到 $O(n \log n)$ 吗?

通过次数 151,940 | 提交次数 336,195

解题思路：

1 动态规划：

定义 dp[i]是考虑前i个元素，以第i个数字结尾的最长子序列的长度。注意nums[i]是被选取的，从小到大计算dp[]数组的值，在计算dp[i]之前，计算出dp[0...i-1]的值，则状态转移方程：

dp[i] = max(dp[j])+ 1, 其中0 <= j < i 且 num[j] < num[i]

往dp[0...i-1]中最长子序列后面再加一个nums[i]。由于dp[j] 代表nums[0...j]中以nums[j]结尾的最长上升子序列，所以如果能从dp[j]这个状态转移过来，那么nums[i] > nums[j]，才能将nums[i]放在nums[j]后面形成的更长的上升子序列。

所以，这个数组的最长上升子序列即所有dp[i]的最大值。

LIS = max(dp[i]), 其中0<=i < n

```
def lengthOfLIS(self, nums: List[int]) -> int:
        if not nums:
            return 0
        dp = []
        for i in range(len(nums)):
            dp.append(1)
            for j in range(i):
                # 只有当nums[i] 比 nums[j]大时，才能进行状态转移。
                if nums[i] > nums[j]:
                    dp[i] = max(dp[i], dp[j] + 1)
         return max(dp)
```

时间复杂度：O(n^2)

空间复杂度：O(n)

二：贪心+ 二分查找

维护一个数组d[i]， 表示长度为i的最长上升子序列的末尾元素的最小值

len：记录目前最长上升子序列的长度，

起始：len = 1， d[1] = nums[0]

d[i] 是关于i单调递增的。因为如果d[j]≥d[i] 且 j < ij<i，我们考虑从长度为 ii 的最长上升子序列的末尾删除 i-j个元素，那么这个序列长度变为 jj ，且第 jj 个元素 xx（末尾元素）必然小于 d[i]，也就小于 d[j]。那么我们就找到了一个长度为 jj 的最长上升子序列，并且末尾元素比 d[j] 小，从而产生了矛盾。因此数组 d[]d[] 的单调性得证。

依次遍历数组nums[]中的每个元素，并更新数组d[]和len的值。如果nums[i] > d[len]，则更新len = len + 1，否则在d[1...len]

中找满足d[i-1]<nums[j] <d[i]的下标i，并更新d[i] = nums[j]

根据d数组的单调性，可以使用二分查找寻找下标i，优化时间复杂度

最后，这个算法的流程为：

设当前已求出的最长上升子序列的长度为 len（初始时为 11），从前往后遍历数组 nums，在遍历到 nums[i] 时：

如果 nums[i]>d[len]，则直接加入到 d 数组末尾，并更新len=len+1；

否则，在 d 数组中二分查找，找到第一个比nums[i] 小的数 d[k]，并更新 d[k+1]=nums[i]。

```python
def lengthOfLIS(self, nums: List[int]) -> int:
        d = [ ]
        for n in nums:
            if not d or n > d[-1]:
                d.append(n)
            else:
                l, r = 0, len(d) - 1
                loc = r
                while l <= r:
                    mid = (l + r) // 2
                    if d[mid] >= n:
                        loc = mid
                        r = mid - 1
                    else:
                        l = mid + 1
                 d[loc] = n
         return len(d)
```