算法复杂度 $O(n^2)$.

伪代码：

```
def findLIS(arr):
    n = len(arr)
    dp = list, size = n, initialized to 1.
    prev = list, size = n, initialized to -1.
    max_length = 1
    max_index = 0.

    for i in range(1, n):
        for j in range(i):
            if arr[j] < arr[i] and dp[j] + 1 > dp[i]:
                dp[i] = dp[j] + 1
                prev[i] = j.

    if dp[i] > max_length:
        max_length = dp[i]
        max_index = i.

    LIS = list
    while max_index != -1:
        LIS.append(arr[max_index]).
        max_index = prev[max_index].
    return LIS.
```

在这个函数中，我使用了一个列表 dp[i] 来存储以 arr[i] 结尾的 LIS 的长度，用了另一个列表 prev[i] 来存以 arr[i] 结尾的 LIS 的上一个元素的位置，用来最后重建 LIS.

从头开始遍历数组，在内层循环中遍历 i 之前的所有元素，如果 arr[j] < arr[i] 且 dp[j]+1 > dp[i]，就表示，此时是目前的最优情况，更新 dp[i]，prev[i]. 在外层循环中，此时我们已经找到了以 arr[i] 结尾的 LIS 的长度，如果这个长度大于目前的最大长度的话就更新 max_length 和 max_index.

最后我们通过用 prev 即可建立 LIS.

举例：arr = [10, 22, 9, 33, 21, 50, 41, 60, 80].

dp = [1, 1, ···, 1], prev = [-1, ···, -1].

对于 arr[1] (22), dp[1] = 2, prev[1] = 0.

arr[3] (33), dp[3] = 3, prev[3] = 1.

arr[4] (50), dp[4] = 4, prev[4] = 3.

arr[6] (41), dp[6] = 4, prev[6] = 3

arr[7] (60), dp[7] = 5, prev[7] = 4.

arr[8] [80], dp[8] = 6, prev[8] = 7.

∴ LIS = [10, 22, 33, 50, 60, 80].

算法复杂度 O(nlogn):

伪代码:

```
def findLIS(arr):
    n=len(arr)
    tails = list()
    prev = list, len(list)=n, initialized to -1.
    indices = list().

    for i in rang(n):
        pos = binarySearch(tails, arr[i]).
        if pos == len(tails):
            tails.append(arr[i]).
            indices.append(i).
        else:
            tails[pos] = arr[i]
            indices[pos] = i
        if pos >0:
            prev[i] = indices[pos-1].

    LIS = list().
    k = indices[len(tails)-1]
    while k!=1:
        LIS.prepend(arr[k])
        k = prev.
    return LIS.
```

思路：使用 tails 存最长严格递增子序列的尾部元素。
遍历 arr，对于每个 arr[i]，使用二分查找找出在 tails
中第一个大于或等于 arr[i] 的元素的位置，并替换，
或者如果 arr[i] 大于所有 tails 中的元素，则将其添
加到 tails 末尾。
通过记录每个元素在 tails 中的位置，我们即可
重建 LIS。