# 动态规划上机题分析

## 问题描述

输入一个整数数组A，给出该数组中最长的单调递增子序列长度。算法复杂度不得高于O(n2)。

## 方法1：基于公共最长子序列（LCS）实现

首先对数组A进行快速排序，并且去除连续的相等元素，得到有序的数组B；然后对数组A以及数组B运行最长公共子序列（LCS）算法，得到的最长公共子序列长度记为数组A中最长的单调递增子序列的长度。

快速排序的算法复杂度为O(NlogN)，去除连续的相等元素可以在O(N)时间内完成（连续的相等元素只复制一份到B数组中）。记B数组的长度为M，则有M≤N，而LCS算法的复杂度为O(MN) ≤O(N2)。因此这个算法的总复杂度为O(NlogN)+O(N)+O(N2)=O(N2)。

值得一提的是，使用的LCS算法中需要避免过多的内存占用，因此在动态规划填表的时候，仅在内存中保存了两列，分别用于保存“上一轮”填表的结果，以及“这一轮”填表的结果。

## 方法2：动态规划

方法1由于需要对输入数据进行多轮处理，并且需要开辟大量内存空间用于中间结果处理，因此效率并不是很高。

可以采用动态规划的思路，直截了当地进行计算。使用L[i]表示A数组中A[1..i]中包含元素A[i]的最长单调子序列的长度，那么对于任意的1≤j<i，若A[j]<A[i]，那么有L[j]+1≤L[i]。因此容易得到如下的递归关系：

L[i] = max{L[j]+1|A[i]>A[j], j<i}

对于i为1的平凡情况，我们有L[1]=1。

在计算完毕L[i]之后，最长子序列的长度即为max{L[i]}。

算法实现中，只需一个双重循环，分别使用i和j作为下标访问数组A。易知其复杂度为O(N2)。

## 方法3：动态规划+快速查找

方法2使用了一个简单的数组就完成了动态规划的计算，但是复杂度仍然为O(N2)，和方法1比没有本质的提高。对于原始序列A，考虑其长度为i的单调严格递增子序列，这样的序列可能有多个，我们选取这些子序列的结尾元素的最小值，记为Li，则有L1<L2<...<Lm。假设{Li}中的最大值为Lm，那么m就是最长单调递增子序列的长度。Li的初始值设置为0，并且由如下规则更新：

对于Ai，若：

* Ai<Li，则Li=Ai
* Ai>Lm，则Lm+1=Ai, m=m+1（m是当前见到的最大的L的下标）
* Ls<ai<Ls+1，则Ls+1=Ai

由于L是递增的，因此对于每个Ai可以使用二分查找来确定需要更新的Li值。二分查找复杂度为O(logN)，总共运行N次，因此总的算法复杂度为O(NlogN)，这相比之前的O(N2)算法有很大的提高。

## 评价

在已经批改的作业中，这三种做法的数量差不多。方法2的实现最为简单明了，而且效率也在能够接受的范围内。方法1将本问题转化为了另外一个问题，利用了现成的LCS算法来处理次问题，也是一个不错的思路。方法1的问题在于逻辑复杂，且需要一些技巧避免过多的内存占用，这些技巧又使得代码复杂。方法3的效率最高，但是其递推关系比较难思考明白。