华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析	**年级**：22 级	**上机实践成绩**：
指导教师：金澈清	**姓名**：石季凡	
上机实践名称：最长上升子序列	**学号**：10225501403	**上机实践日期**：2023.5.15
上机实践编号：No.10	**组号**：1-403	

一、目的

1．熟悉算法设计的基本思想
2．掌握计算最长上升子序列的方法

二、内容与设计思想

1. 基于动态规划编写计算最长上升子序列方法的代码。
2. 基于贪心算法和二分搜索编写计算最长上升子序列方法的代码。
3. 对比两种实现方式不同数据量下运行时间的差异。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. *写出计算最长上升子序列方法的代码*
2. *分别画出各个实验结果的折线图*

五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

【算法实现】

【动态规划】：首先定义一个 MaxSize 数组用于存放以该位置的已知序列数字结尾的最长上升子序列的长度，对于一个长度为 length 已知的序列，分别设置内外两个循环，外循环的变量 i 每次自增 1，这表示我们将依次把已知序列每一个元素设置为目标元素，然后在内循环中利用变量 j 自增对其前面的元素进行遍历，若满足 nums_input[j]<nums_input[i]，此时更新 MaxSize[i]为 max{MaxSize[i],MaxSize[j]+1}，最终返回 MaxSize 数组的最大元素。

【贪心+二分搜索】：主要思想为，当我们想构造一个最长子序列时，对于末尾数字的选择，我们选择的数字越小，构造的子序列更长的可能性更大，因此我们要做的就是在遍历输入序列时，不断更新不同长度的上升子序列的结尾元素，因此我们需要定义一个 MinNum 数组，每一位的数字代表着以该下表+1 为长度的最大子序列的结尾数字，而对于一个遍历到的输入序列元素，由于我们的 MinNum 序列是递增序列，我们可以用二分搜索轻松找到我们需要更新的位置。最终只需要返回末尾元素即可。

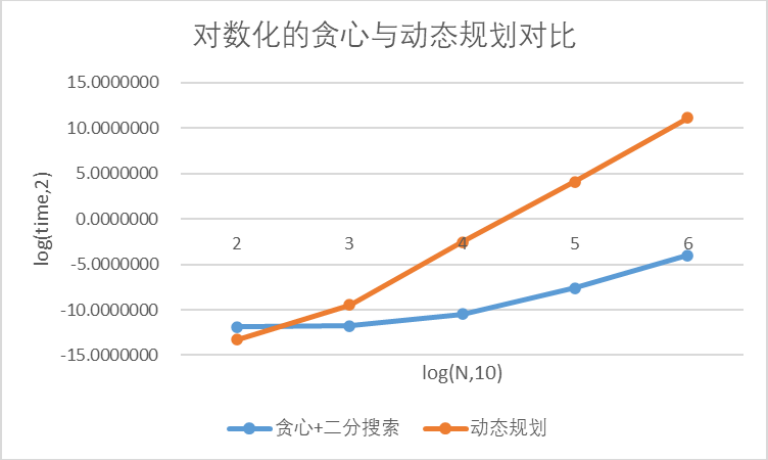
【时间对比】：我们可以很明显的发现，动态规划的时间复杂度为 O(n^2)，而贪心与二分搜索的结合为 O(lgn)，因此在大数据量时贪心与二分搜索的结合将远远快于动态规划法。

下面进行对比实验。

我们输入一个 N，取值为 1e2,1e3,1e4,1e5,1e6，然后生成 N 个随机数作为输入序列，对其求最长上升子序列，分别记录两种算法的时间。

	贪心+二分搜索	动态规划
100	0.0002604	0.0000993
1000	0.0002853	0.0014051
10000	0.0006920	0.1694768
100000	0.0051659	17.3965602
1000000	0.0616327	2209.0270177

其中，横轴为算法类型，纵轴为 N 的大小，表中数据为运行耗时，单位为秒。由于该数据变化太大，不宜绘图，因此我对时间取了以 2 为底的对数，对 N 取了以 10 为底的对数，得到如下图像。



可见，在 N=100 时由于动态规划的单步耗时较少，因此较快，但是随着增大，其耗时急剧增加，并远长于贪心+二分搜索算法。这也验证了我们之前的推测，即动态规划的时间复杂度为 O(n^2)，贪心+二分搜索时间复杂度为 O(nlgn)。

【动态规划的一个小优化】：由于动态规划法在每一个外循环内都要进行一次内循环，加之最后还需要遍历返回最大值，我们不妨最初定义一个 Maximum 用于记录前 i 项的最大值，当内循环中，MaxSize 达到 Maximum+1 时停止循环并且将 Maximum 自增 1。

```
if (MaxSize[i] == maximum+1)
{
    maximum++;
    break;
}
```

代码如上

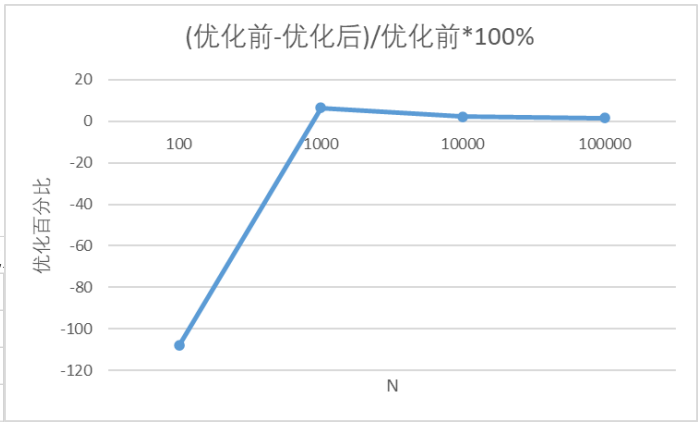
但是同样这也有一定的问题，在数据量比较小时，由于额外的 if 语句也会耗时，因此不一定会有所优化。

我们输入一个 N，取值为 1e2,1e3,1e4,1e5 然后生成 N 个随机数作为输入序

列，对其求最长上升子序列，分别记录优化前后算法的时间。

	优化后	优化前
100	0.000034	0.0000707
1000	0.0015532	0.0014531
10000	0.1714580	0.1677709
100000	17.7224016	17.4320565

如图得到的数据，横轴为算法类型，纵轴为 N 的大小单位为秒，由于不易于观察我将优化前后的耗时之差除以了优化前耗时以算出百分比，得到如下数据与图标。



	优化前-优化
100	-107.94118
1000	6.44475921
10000	2.15043917
100000	1.63829433

我们可以发现确实在数据量增大时耗时有微乎其微的减少，也和我们猜测的结果相类似。但是想要更加准确的验证的话则需要更加完备的实验。另外，由于动态规划本身的算法确实效率过于低，因此想要发生质的耗时改变也只能修改算法。