云南大学数学与统计学院

上机实践报告

|  |  |  |
| --- | --- | --- |
| **课程名称**：数据结构与算法实验 | **年级**：2015级 | **上机实践成绩**： |
| **指导教师**：陆正福 | **姓名**：刘鹏 |  |
| **上机实践名称**：算法分析实验 | **学号**：20151910042 | **上机实践日期**：2017-04-10 |
| **上机实践编号**：No.3 | **组号**： | **上机实践时间**：上午三、四节 |

# 一、实验目的

1. 熟悉算法分析的基本概念与方法

# 二、实验内容

1. 实验研究法

任取一些典型算法, 进行Python实现，模仿3.1研究其运行时间，分析算法效率。

2. 渐近分析法

查阅Python文献，绘制图3.4，体会渐近分析法中经常遇到的几种函数的增长率。

# 三、实验平台

Windows 10 1703Enterprise 中文版；

Python 3.6.0；

Wing IDE Professional 6.0.2-1集成开发环境；

MATLAB R2017b。

# 四、实验记录与实验结果分析

1. 对典型算法的运行时间研究，要求用Python进行实现。

**Solution**：

由于目前接触的算法还不算很多，所以这里仅仅考虑不同的排序算法。典型的排序算法有冒泡排序（Bubble Sort）、选择排序（Selection Sort）、快速排序（Quick Sort）。我们这里选择这三种排序方法进行Python实现，然后运行，分析各自的运行时间，进而分析算法效率。这里采取的是实验分析法。

冒泡排序（Bubble Sort）算法解释：

C:\Users\newton\AppData\Local\Microsoft\Windows\INetCache\Content.Word\绘图1.emf

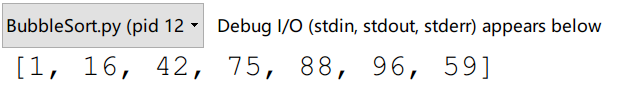
Figure 1

在Figure 1中，我们假设有8个无序的数字，现在我们需要用冒泡法进行由小到大排序。先定义一个指向，如带尾缀的箭头，它指向假定的最小值，然后建立另一个指向，它从该假想值之后的位置开始，遍历剩下的所有元素，如果遇到了比假想值小的元素，就互换假想值与该元素，然后认定该元素为新的假想值。如此一来，经过第一次循环，最左侧就是最小值。然后把带尾缀的箭头指向该表的第二个元素，将该算法运行于除最小值之外的所有元素上。最终尾缀箭头指向该数组的倒数第二个元素，这个时候再与最后一个元素相比，算法完成。

从上面的叙述中我们可以看到，当小箭头遍历完了剩余元素之后，才移动尾缀箭头，而且尾缀箭头的终点在倒数第二个元素位置上。而小箭头的起点都在对应的尾缀箭头的下一个位置上。由此写出Python算法。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # filename: BubbleSort  **def** BubbleSort**(**L**):**  **for** i **in** range**(**0**,**len**(**L**)-**2**):**  **for** j **in** range**(**i**,**len**(**L**)-**1**):**  **if** L**[**i**]** **>** L**[**j**]:**  L**[**i**],**L**[**j**]** **=** L**[**j**],**L**[**i**]**  **return** L  a **=** **[**1**,**96**,**88**,**75**,**42**,**16**,**59**]**  **print(**BubbleSort**(**a**))** |

程序代码 1



运行结果 1

快速排序（Quick Sort）：

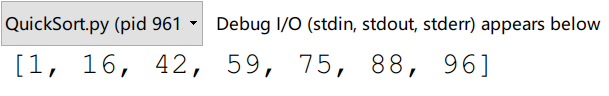


Figure 2

在Figure 2中我们可以看到快速排序的行为方式。白圈就是我们选定的一个基，在它的基础上，我们进行左右分类，分为比它小的，比它大的，与它相等的（包括它自己），然后把这个算法递归性地应用的两类不等的数组中。当数组元素为1，结束划分。最后把这些合并起来，就是最后的有序数组。这个过程有很明显的递归性。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | # Quick Sort for a list  **def** QuickSort**(**L**):**  L\_Left **=** **[]**  L\_Right **=** **[]**  L\_Middle **=** **[]**  **if** len**(**L**)** **<=** 1**:**  **return** L  **else:**  **for** i **in** L**:**  **if** i **<** L**[**0**]:**  L\_Left**.**append**(**i**)**  **elif** i **>** L**[**0**]:**  L\_Right**.**append**(**i**)**  **else:**  L\_Middle**.**append**(**i**)**  L\_Left **=** QuickSort**(**L\_Left**)**  L\_Right **=** QuickSort**(**L\_Right**)**  **return** L\_Left **+** L\_Middle **+** L\_Right  a **=** **[**1**,**96**,**88**,**75**,**42**,**16**,**59**]**  **print(**QuickSort**(**a**))** |

程序代码 2



运行结果 2

基于上面的两种算法，我们用十万个数字对他们进行排序速度测试。由于把数据写入文件再调用，会造成硬盘读写的干扰，为了最大化降低干扰，我们直接把数据写在代码里，运行的时候相当于直接调用内存中的文件，速度会快很多，干扰也会小很多。

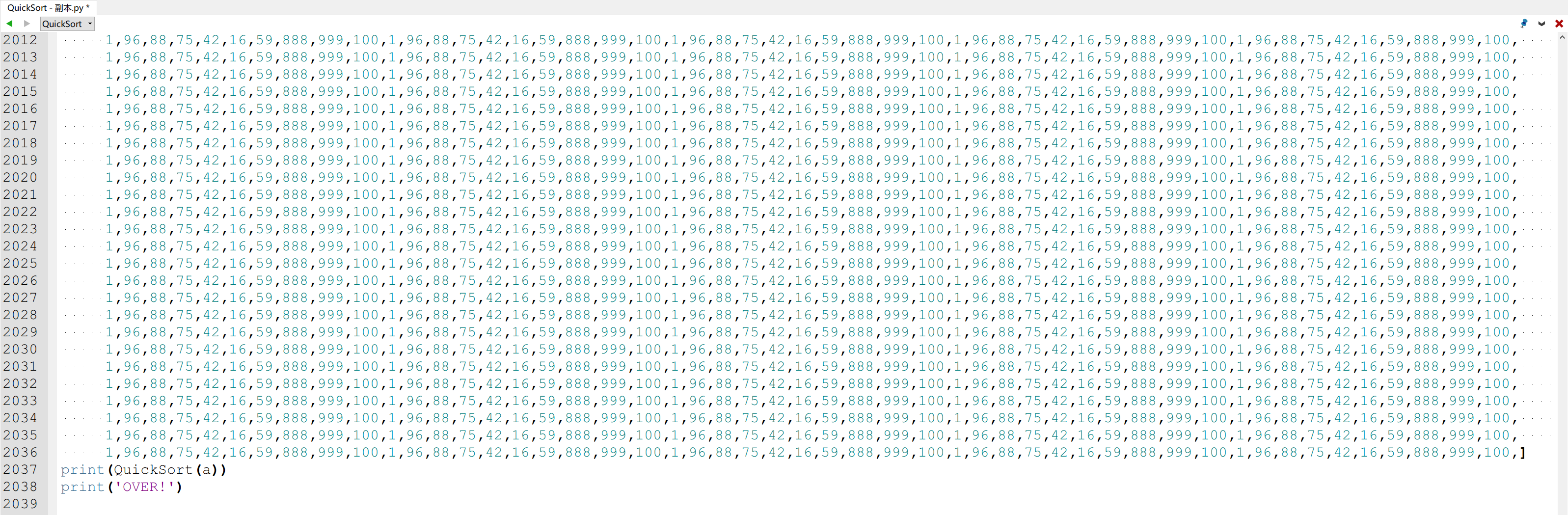


Figure 3

由于Python的运行时间暂时无法查看，所以只能凭借直观感受进行测量。令人欣喜的是，可以明显看出，排序2000行每行50个数字，即一共100.000个数字的时候，快速排序非常迅速，而冒泡排序运行了很久，在Intel i7 6700K @4.6 GHz 下，后者运行了两分钟没有出结果。效率可想而知。

2. 查阅Python文献，绘制图3.4，体会渐近分析法中经常遇到的几种函数的增长率。

**Solution**:

To sum up, Table 3.1 shows, in order, each of the seven commo functions used in algorithm analysis.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **constant** | **logarithm** | **Linear** | **n-log-n** | **quadratic** | **cubic** | **exponential** |
| 1 |  |  |  |  |  |  |

**Table 3.1: Classes of functions. Here we assume that  is a constant.**

Ideally, we would like data structure operations to run in times proportional to the constant or logarithm function, and we would like our algorithms to run in linear or *n*-log-*n* time. Algorithm with quadratic or cubic running times are less practical, and algorithms with exponential running times are infeasible for all but the smallest sized inputs. Plots of the seven functions are shown in Figure 3.4.



|  |
| --- |
| **Figure 3.4：**Growth rates for the seven fundamental functions used in algorithm used in algorithm analysis. We use base a = 2 for the exponential function. The functions are plotted on a log-log chart, to compare the growth rates primarily as slopes. Even so, the exponential function grows too fast to display all its values on the chart. |

上面的图标是用MATLAB做出的。MATLAB代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | %% 输入自变量n的数量级  n **=** logspace**(**0**,**15**,**16**);**  %% 定义函数  f\_constant **=** n **-** n **+** 10e2**;**  f\_logarithm **=** log2**(**n**);**  f\_linear **=** n**;**  f\_nlogn **=** n **.\*** log2**(**n**);**  f\_quadratic **=** n**.^**2**;**  f\_cubic **=** n**.^**3**;**  f\_exponentional **=** 2.**^**n**;**  %% 作图  loglog**(**n**,**f\_constant**,**'-d'**);**  hold on**;**  loglog**(**n**,**f\_logarithm**,**'-o'**);**  loglog**(**n**,**f\_linear**,**'-s'**);**  loglog**(**n**,**f\_nlogn**,**'-p'**);**  loglog**(**n**,**f\_quadratic**,**'-h'**);**  loglog**(**n**,**f\_cubic**,**'-v'**);**  loglog**(**n**,**f\_exponentional**,**'-<'**);**  axis**([**1 10e15 1 10e44**]);**  xlabel**(**'n'**);**  ylabel**(**'f(n)'**);**  hold off |

程序代码 3

# 五、实验体会

Translation

Chapter 3 Algorithm Analysis

＊第三章 算法分析

In this book, we are interested in the design of “good” data structures and algorithms. Simply put, a data structure is a systematic way of organizing and accessing data, and an algorithm is a step-by-step procedure for performing some task in a finite amount of time. These concepts are central to computing, but to be able to classify some data structures and algorithms as “good”, we must have precise ways of analyzing them.

＊在这本书中，我们致力于设计优秀的数据结构以及算法。简单说来，数据结构就是系统化的组织和调用数据的方法，而算法就是为了在有限的时间内能一步一步地完成一些任务的程序。这是计算的核心概念，但是为了辨别算法与数据结构的好坏，我们必须要有准确的分析方式。

END

根据书中给出的分析方式，可以初步分析一些带有循环的算法的时间复杂度。但是有些还是很难自己证明。这有待进一步的学习。

在排序中，两种算法的效率之迥然，令人咂舌。足以说明追求低时间复杂度的算法是重要的任务。

# 六、参考文献

[1] Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser, *Data Structures and Algorithms in Python*

[2] 实验教材：汪萍，陆正福等编著，数据结构与算法的问题与实验 第1章