字符串排序算法的调研与实现

李翠琪

（同济大学,软件学院,上海 200000）

**摘要：** 对于许多的排序,决定顺序的键值都是字符串,于是就想利用字符串的特殊性质来设定专门针对字符串的排序算法,使得排序效率更高。我调研并用C++代码实现了LSD算法,MSD算法以及三向快速排序算法来对字符串进行排序, 并且对这些算法增加了一些改进。之后通过运行程序来比较了几个算法所需要的排序时间, 最后通过查找资料分析归纳了这些算法的特性。

**关键词：**字符串排序，LSD算法,MSD算法，三向快速排序算法，性能对比

**Investigation and Implementation of String Sorting Algorithm**

*LI Cuiqi*

(1. Tongji University, ShangHai 200000, China)

**Abstract:** For many sorts, the key value that determines the order is a string, so I want to use the special properties of the string to set a sort algorithm specifically for the string, so that the sort efficiency is higher. I researched and implemented the LSD algorithm, MSD algorithm, and three-way quick sort algorithm in C ++ code to sort strings, and added some improvements to these algorithms. Then we compared the sorting time of several algorithms by running the program. Finally, we summarized the characteristics of these algorithms by analyzing the data.

**Key words:** String sorting, LSD, MSD, three-way fast sort, performance comparison

在学习多种排序算法时，发现主要是通过比较int类型或者是一些可比较的值进行排序。而字符串排序很常用，并且字符串很有特点，于是我就对字符串排序产生了兴趣。

首先，学习基数排序时，课本介绍了实现多关键码排序有两种常用的方法：最高位优先MSD与最低位优先LSD。我发现与字符串排序非常符合，于是上网查了相关的资料并且借阅了网上推荐的数据结构书籍《算法4》。

在《算法4》中，发现里面的第五章对字符串排序有非常详细的描述，只不过是用Java语言实现的。于是我就学习研究了里面的内容，并且将之转化位C++版本的代码。

学习历程大概是先学习了最低位优先LSD排序算法, 而上课所学的基数排序是典型的LSD排序方法，这里只是针对字符串做了相应的改变而已，所以掌握很快。之后再学习了最高位优先MSD排序算法，学习中，我发现其实和LSD算法差距不大，只是从左往右排序罢了。

之后，通过它的后续章节意识到了，MSD算法其实就是专门针对字符串的快速排序，而之前老师也提过的三向快排，其实也可以用在这里，于是又学习了三向字符串快速排序算法，至此，学完了《算法4》的理论知识。

随后，我用C++代码重新实现了这三个代码，并且参照《算法4》中提出的改进建议给代码做了一些优化改进。之后又受到之前所写的数据结构课程设计习题，写了个小程序来比较几个算法排序所用的时间。最后，参照《算法4》，根据自己的理解，分析归纳这几个字符串排序算法的一些特性。

**1 LSD算法实现**

* 1. **LSD算法描述**

首先,用LSD算法对字符串数组进行排序需要保证字符串数组中的字符串长度是相等的。它从右向左以每个位置的字符作为键，以键值索引法的思想将字符串排序W遍（W为字符串的长度）。

每一次排序都针对字符串的第d个字符进行排序，大致分为一下四个步骤：

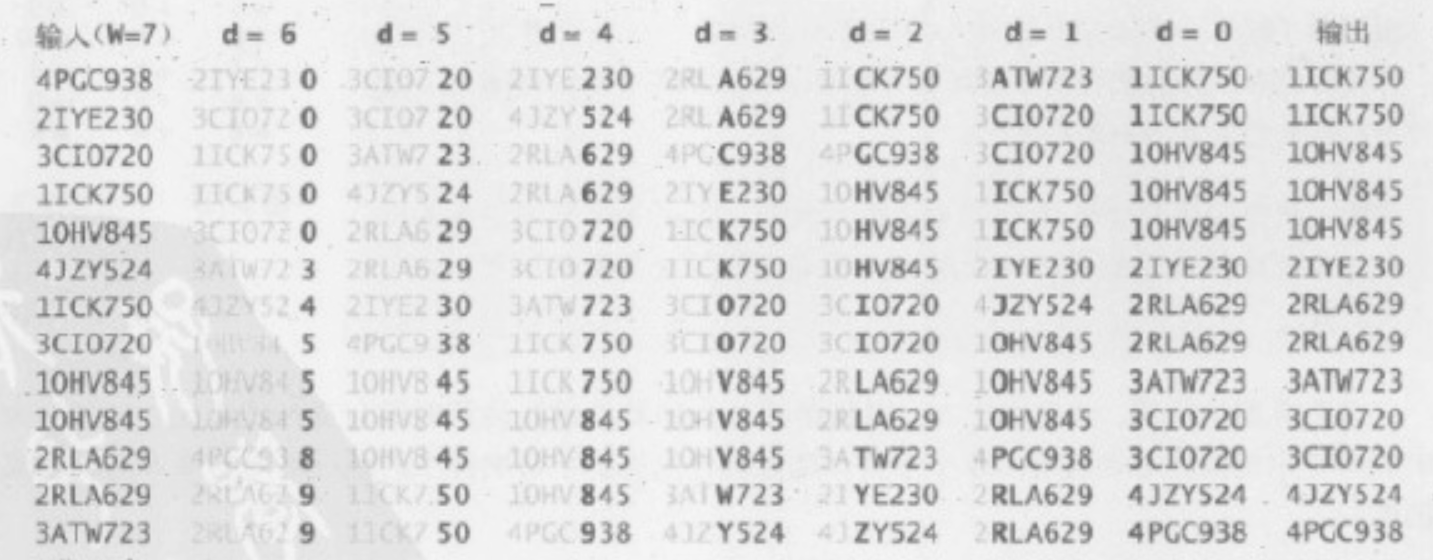
1.计算字符串的第d个字符出现的频率

2.将频率转为索引

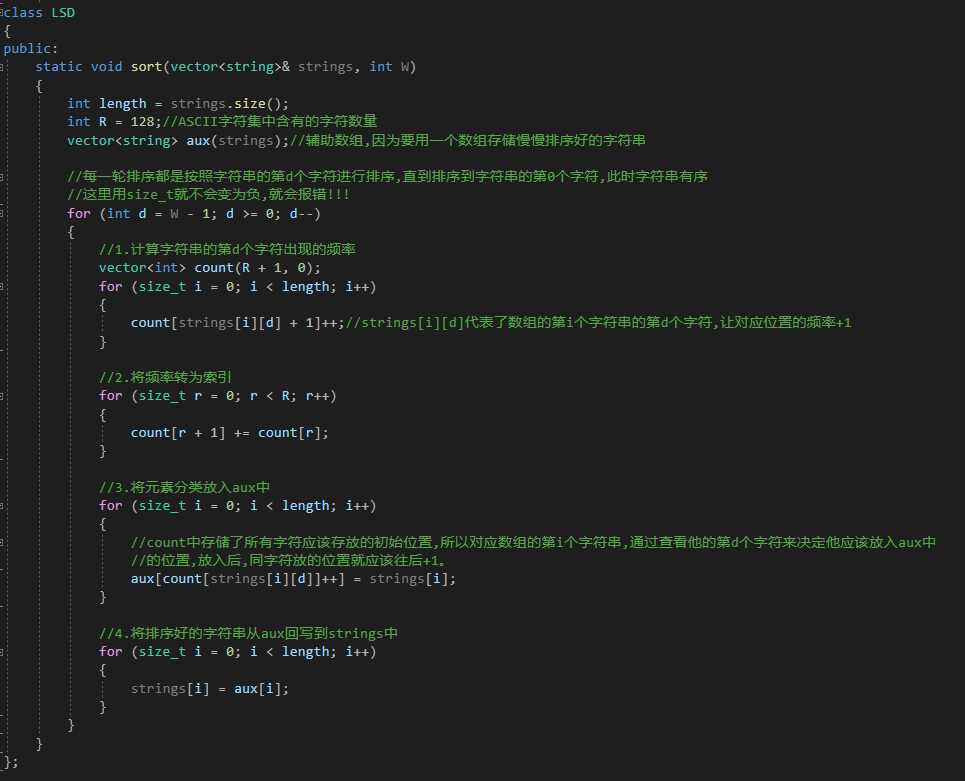
3.将字符串按照对应组的位置放入aux中

4.将排序好的字符串从aux回写到strings中

排序流程：

****

* 1. **LSD算法代码实现**

****

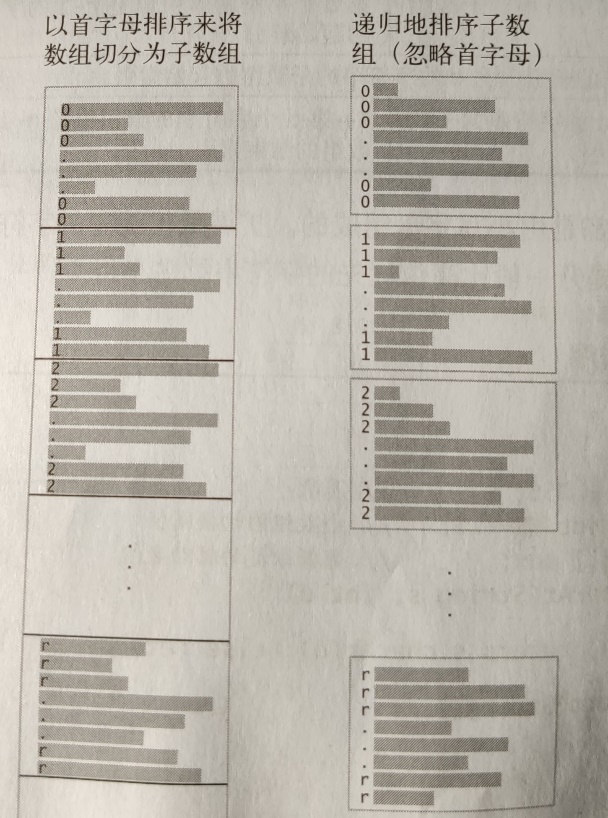
1. **MSD算法实现**
   1. **MSD算法描述**

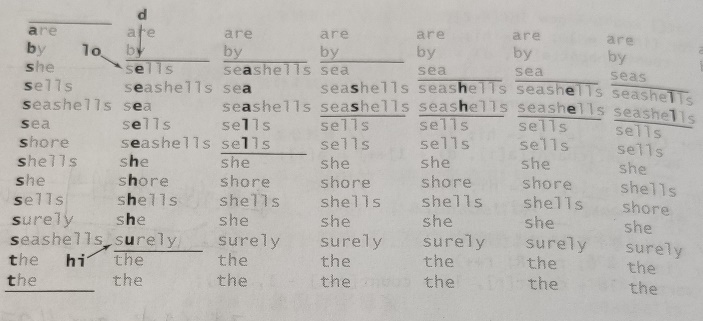
用MSD算法排序字符串数组，实际上是从高位即从左到右对字符串不断排序，它不要求字符串数组中的字符串长度都相同，适用范围更加广。它也以键值索引法为基础进行排序。

MSD的核心思想是分治算法，即将大问题分为小问题来解决，其思想与快速排序类似，具体做法是：先对最高位的字符（即string[0]）进行排序，将排序后的字符串进行分组——最高位相同的在一组；之后递归的对同一组继续进行MSD排序，不过此时以第二位字符进行排序，直到排完最低位，算法结束。

需要注意的是，因为不要求字符串的长度必须一致，那么就要讨论递归到了字符串结尾的情况。一般来说，当一个字符串的所有字符都已经被检查了之后，它应该排在其他字符串的前面。例如abc，abcd，当检查完了c，那么abc肯定要排在abcd的前面。所以需要设置检查完所有字符的字符串放在第一组。

排序的其中一部分：

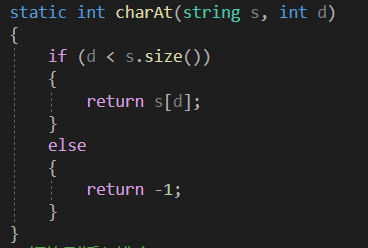




* 1. **MSD算法代码实现**

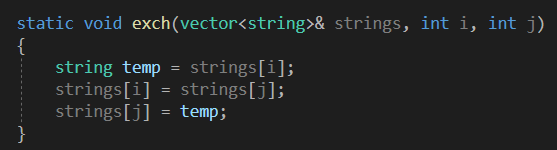
2.2.1 charAt函数

因为高位排序不一定都是同样长度的字符串，所以必须有个函数处理字符串访问越界问题。所有字符都已经被检查过的字符串所在的子数组应该排在子数组的前面。

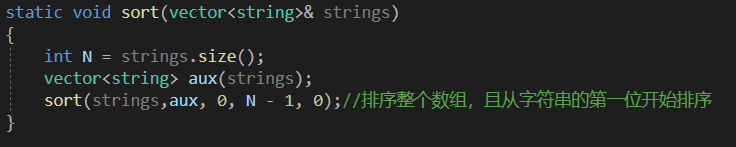


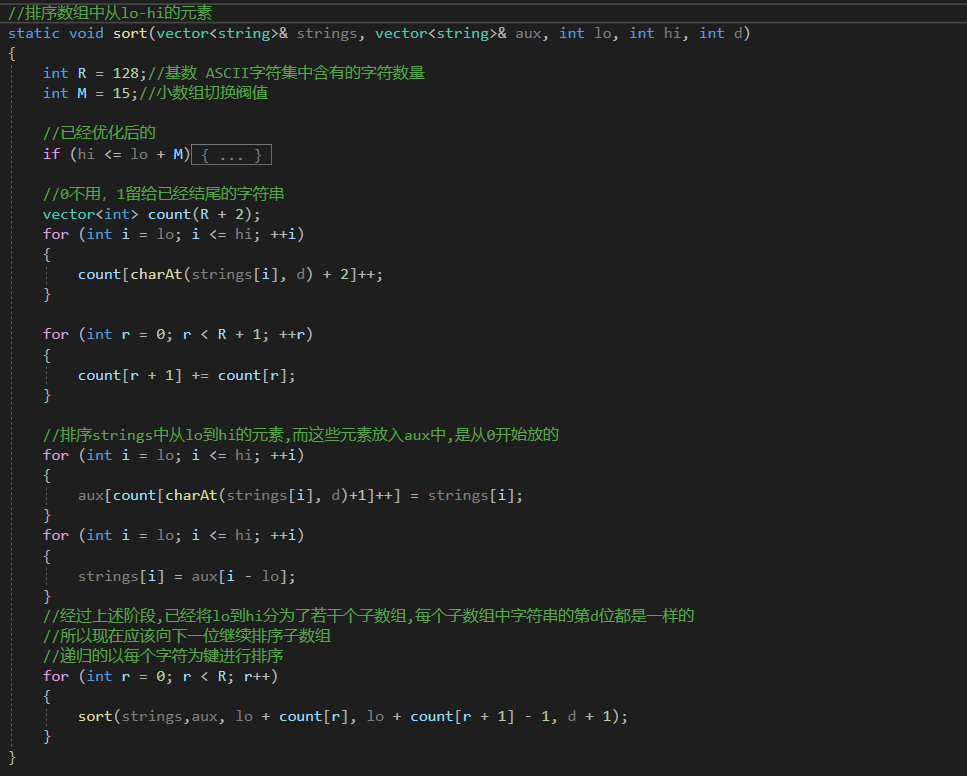
2.2.2 exch函数

一个交换字符串数组中两个元素的函数。



2.2.3 核心函数



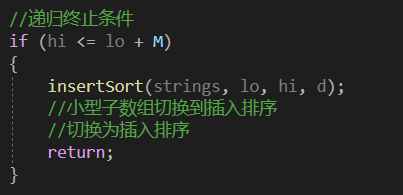


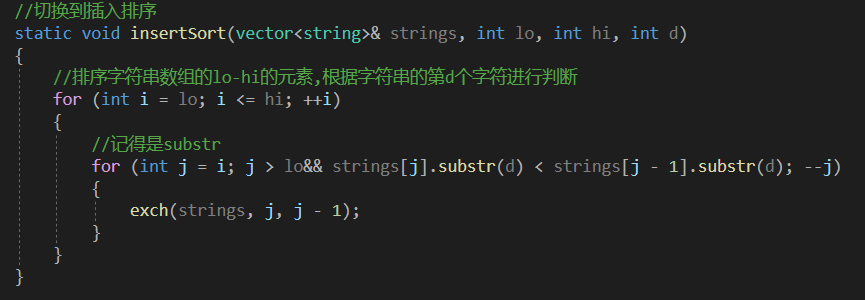
* 1. **MSD算法改进**
     1. 切换插入排序

因为是一种递归算法，所以递归到后面将要处理大量微型数组。如果是当hi<=lo时才返回，那么要递归到只剩下一个字符串的子数组时才会返回，显然会对性能影响很大。

所以，设置一个转换阈值M，当子数组中的元素小于这个阈值时，切换为插入排序。我设置M为15，因为插入排序在数组较小时表现比较好，所以能节约时间。

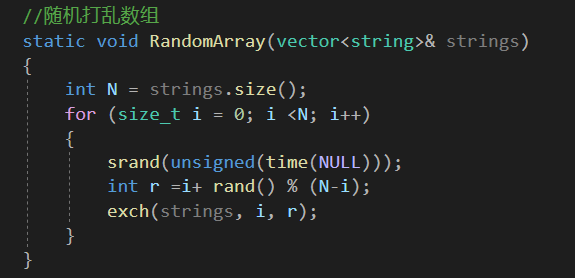
还需要注意的是，切换为插入排序时，应该排序的是从lo到hi的子数组，且排序完就应该返回了，因为此时已经是有序的了。且插入排序比较的是剩余子字符串。





* + 1. 随机化

和快速排序一样，为了避免出现最坏情况，可以将数组打乱后再排序。



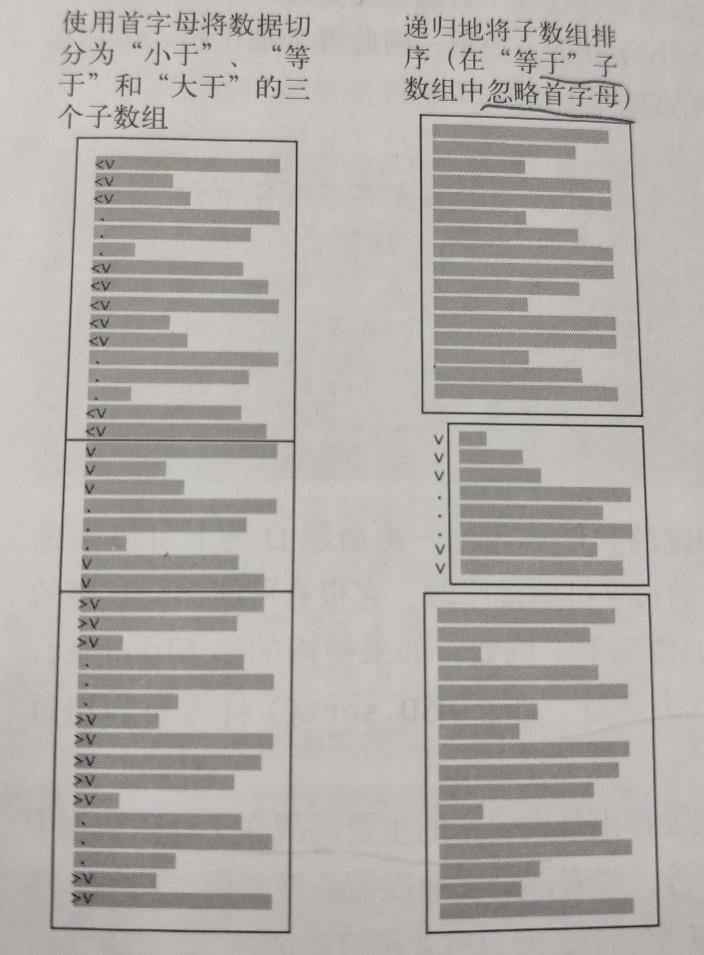
**3三向快速排序算法实现**

**3.1三向快速排序算法描述**

MSD算法在处理含有大量公共前缀的字符串数组时表现不太好。于是针对这个情况改进成字符串三向快速排序算法。其实这个算法和普通的三向快排很像。

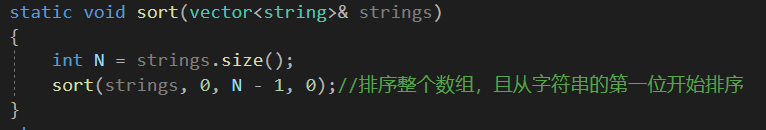
大致是根据字符串的首字母将字符串数组分为三个子数组，然后递归地将这三个子数组排序：一个含有所有首字母小于切分字符的字符串子数组，一个含有所有首字母等于切分字符的字符串子数组（接下来的排序将忽略他们当前这个首字符），还有一个含有所有首字符大于切分字符的子字符串数组。

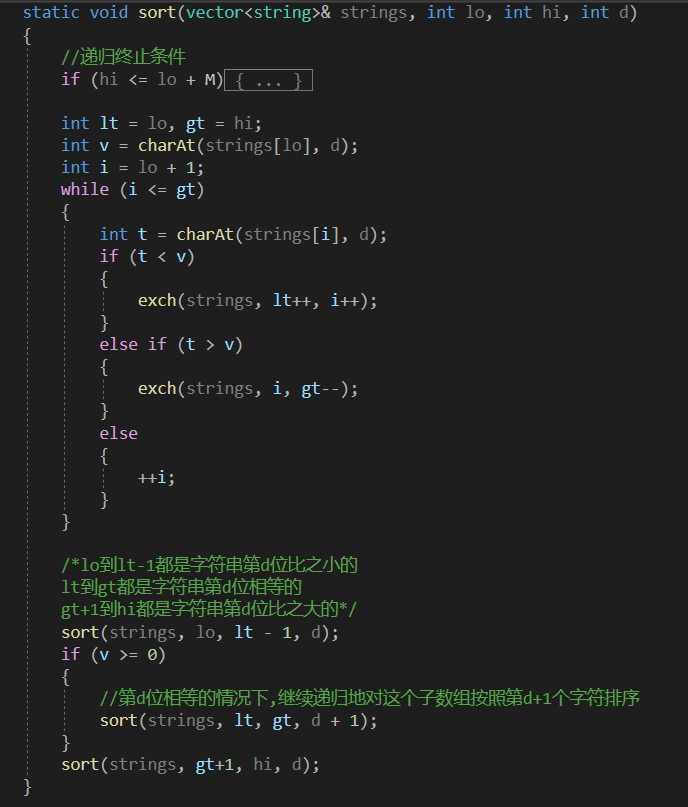
三向字符串快速排序示意图：



**3.2三向快速排序算法代码实现**

有一些和MSD相同的辅助函数就不列出来了

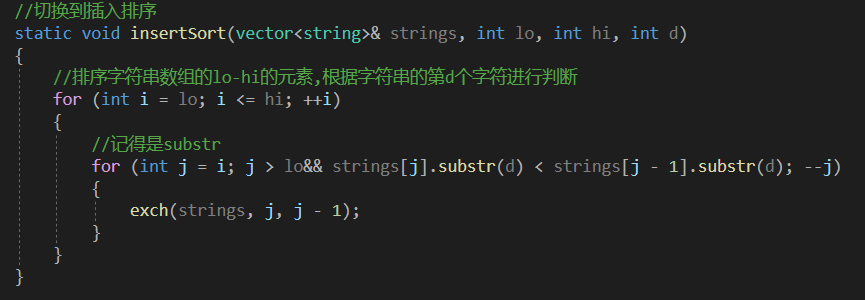
****

****

**3.3三向快速排序算法改进**

3.3.1 切换为插入排序

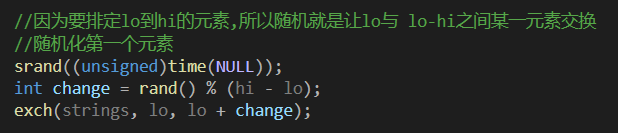
和MSD算法一样，在处理小子数组时，需要切换位插入排序以提高速度。hi<=lo+M时切换。



3.3.2 随机打乱数组

和快速排序一样，最好在排序之前将数组打乱或者将第一个元素和一个随机位置的元素交换以得到一个随机的切分元素，这样就可以预防数组已经有序或是接近有序的最坏情况。

我在这里的随机化算法和MSD采用的不一样，我是在递归排序子数组的函数中，将数组中的lo与lo-hi之间某一元素交换。



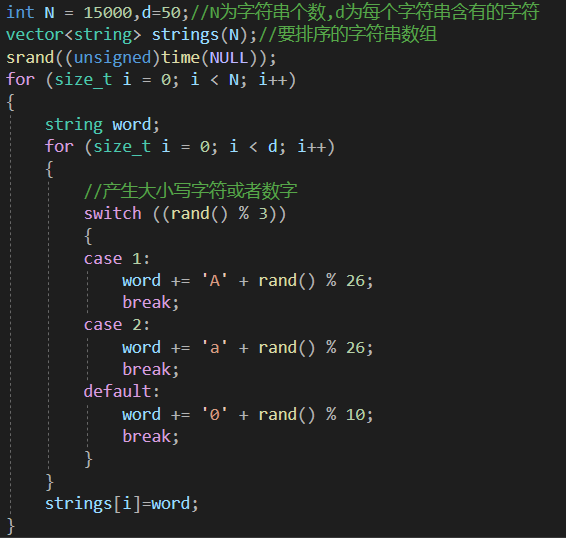
**4 三种算法排序时间比较**

**4.1比较程序说明与实现**

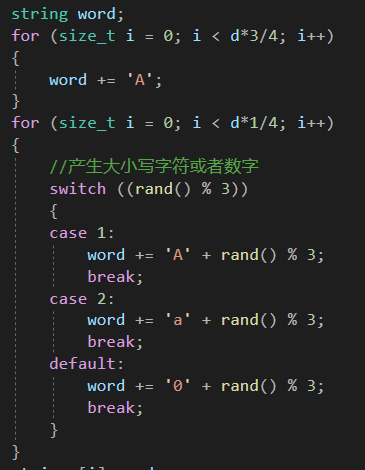
4.1.1生成随机字符串数组并存储

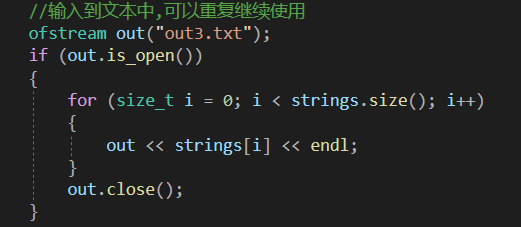
字符串排序需要给定数据，于是通过程序生成一个随机的存储字符串的数组，并且把它存储在txt文件中，以供后续实验能用同一个随机字符串数组，重现之前的实验结果。并且可以控制变量N与d来控制数组中字符串的个数以及每个字符串的长度。

生成随机字符串数组具体实现为下图：



生成有较长公共前缀的字符串数组：



将生成的字符串数组存储在txt文件中实现：

4.1.2判断字符串数组是否排序

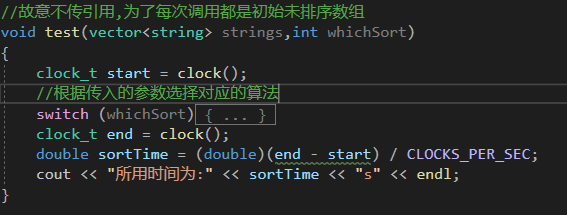
比较排序算法性能的前提是算法能正确将数组排序，但是当输入很大的时候，很难看出是否排序，于是通过程序判断，遍历排序之后的数组，如果后面的元素小于前面的元素，那么就没有排序，就给出提示并且返回。

4.1.3比较字符串数组排序时间函数实现

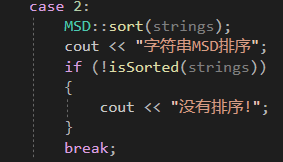
大致实现就是在排序之前计时，然后调用相关算法排序，之后再停止计时，这样就可以查看排序所用的时间了。

需要注意的是，比较不同算法的时候，传入的待排序字符串数组要相同，所以不能传引用，不然之后传入的就是已经排序的数组了，显然不对。

核心代码如下：

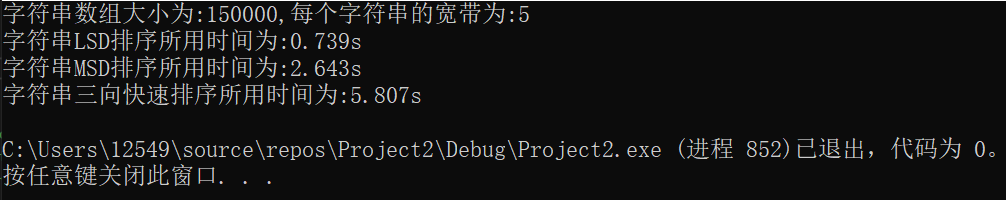


switch中执行某个算法（以MSD为例）：



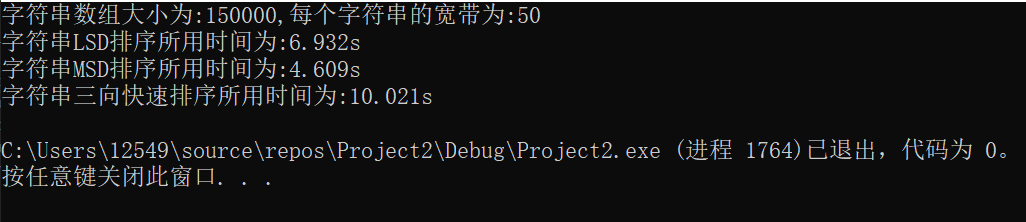
**4.2 不同输入下三种算法运行时间比较**

4.2.1随机字符串数组 N=150000 d=5



由此看出,当字符串宽带比较小的时候LSD算法所用时间显著比较短,而三向快排所用时间最长。

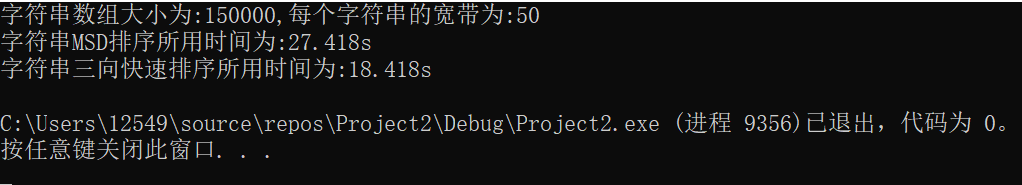
4.2.2随机字符串数组 N=150000 d=50



可以看出，当字符串宽带比较长的时候，MSD算法优势比较大，所用时间最短，而三向快排还是比其他两个算法时间长很多。

4.2.3大量重复字符串数组 N=150000 d=50

说明：这里给出的字符串不一定是定长的，而且有很长的公共前缀。所以不能用LSD算法进行排序，主要比较MSD算法和三向字符串快速排序算法的排序时间。



由此可以看出，对于有较长公共前缀的字符串组成的数组中，三向字符串快速排序算法的排序时间显著低于MSD算法。

**5 三种字符串排序算法的特性**

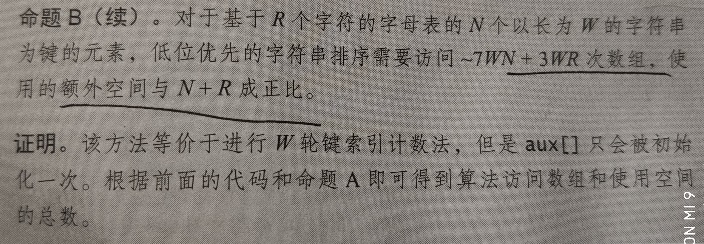
**5.1 排序性能理论分析**

5.1.1 LSD字符串排序算法性能

LSD算法是稳定排序算法，因为在将他们的最后i个字符作为键进行排序之后,在对i-1字符排序时,

要么因为它们的倒数第i个字符不同，所以排序算法已经将他们的顺序摆放正确,要么它们的倒数第i个字符相同，所以由于排序的稳定性它们仍然有序。排序的稳定性证明：在对最后i个字符作为键进行排序时,因为是不断遍历字符串数组放入,所以键相同的元素在排序之后会聚集在一起,但相对顺序没有变化，所以是稳定的。

访问数组次数：



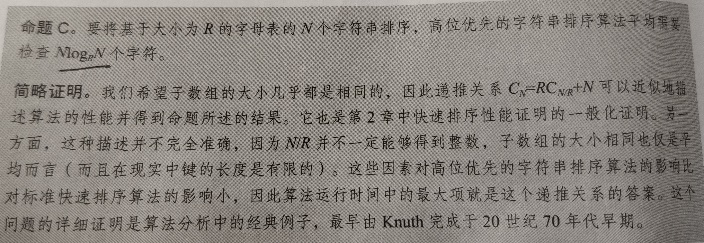
程序运行时间：NW（N为数组大小，W为字符串长度）

排序所需要的额外空间：N(N为数组大小)

5.1.2 MSD字符串排序算法性能

如果不将之随机化，那么MSD算法也是稳定的，原因于LSD算法一致。

比较次数：



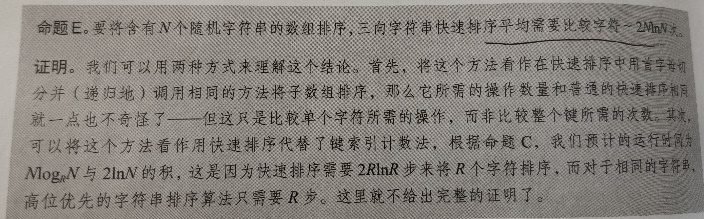
程序运行时间：N到Nw之间（N为数组大小，w为字符串平均长度）

排序所需要的额外空间：N+WR（N为数组大小，W为字符串最大长度,R为字母表大小，本文中是ascii码表码表的大小）

5.1.3三向字符串快速排序算法性能

三向字符串快速排序不是稳定排序算法。

比较次数：



程序运行时间：N到Nw之间（N为数组大小，w为字符串平均长度）

排序所需要的额外空间：W+log N（N为数组大小，W为字符串最大长度）

**5.2 三种算法简单归纳**

LSD字符串算法与MSD字符串算法都是以键值索引法为基础，只不过LSD算法从右往左排序N次，而MSD算法从左往右排序。

LSD算法与MSD算法都是稳定排序算法，但不是原地排序算法。而三向字符串快速排序算法不是稳定排序算法，但是是原地排序的。

LSD算法只适用于定长字符串，而MSD算法于三向字符串快速排序算法对于任意长度的字符串组成的数组都适用。

LSD算法适用于较短的定长字符串，MSD算法适用于随机字符串，而三向字符串快速排序算法适用于含有较长公共前缀的字符串。

这三种字符串排序算法都各有优缺点，我们要根据实际情况选择适合的算法，而不是认定了只使用一种算法。

**参考文献：**

[1] Robert Sedgewick,Kevin Wayne 著,谢路云 译.算法（第4版）[M]. 北京:人民邮电出版社,2012年10月第一版