实验报告

一、算法设计思路

1.快速排序算法实现

首先选择一个元素作为基准元素（pivot），然后通过一趟排序（partition）将整个数组分割成以基准元素为界的两部分，其中一部分的元素数值均小于基准元素，另一部分的元素数值均大于基准元素。然后，可以分别对这两部分重复上述操作，继续进行排序，并重复直至整个数组有序。

其中，在进行partition排序时，设置指示位置的变量i，j，从头开始扫描数据，通过数据之间的交换，使得i左边的数据均小于基准元素，i和j之间的数据均大于基准元素，最终成功将数组以基准元素为界分为两个子数组。

针对如何选择基准元素，有四种不同的选择方式：

1.总是使用第一个元素作为基准元素

2.总是使用最后一个元素作为基准元素

3.使用一个随机元素作为基准元素

4.使用“三取中”的方法选取基准元素

分别实现这四种选择方式，并通过样例输入进行比较。

2.统计比较次数

为了统计总比较次数，设置全局变量compare\_num，在每次调用QuickSort函数时，该变量增加当前数组长度减1即可（因为基准元素会和除它本身之外的每个元素发生比较）

二、程序说明

通过以上的思路分析，设计可递归调用的QuickSort函数，以及重要的Partition函数。并针对四种不同的基准元素选择方式，设计ChoosePivot1、ChoosePivot2、ChoosePivot3和ChoosePivot4函数对应四种方式，可以在QuickSort函数中进行改变。

另外设置N表示待输入数组的长度，可以直接在define语句中进行修改。

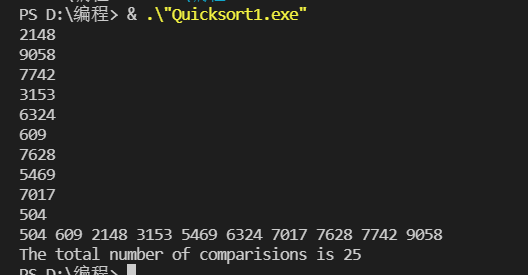
程序详见“QuickSort.cpp”。

三、测试结果及分析

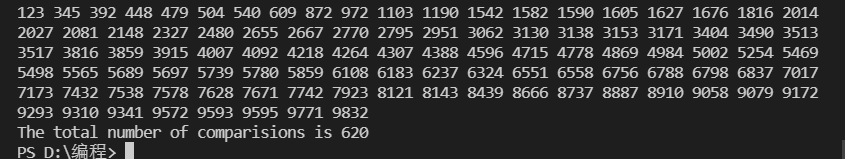
在<http://www.algorithmsilluminated.org/>网页上下载样例输入，进行测试

**1. 总是使用第一个元素作为基准元素**

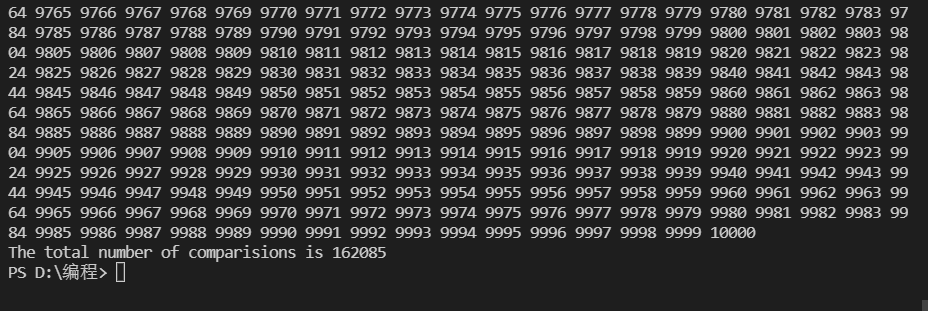
测试样例1



测试样例2



挑战数据集：

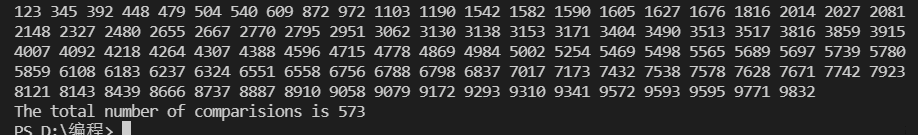


**2. 总是使用最后一个元素作为基准元素**

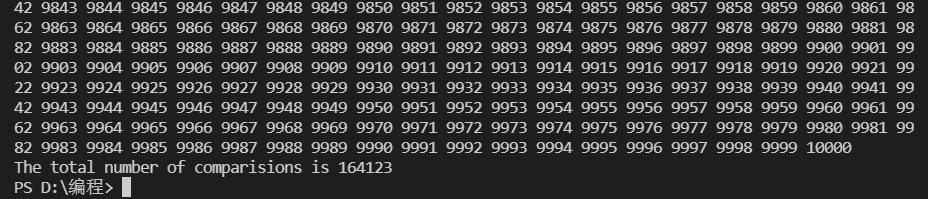
测试样例1



测试样例2



挑战数据集

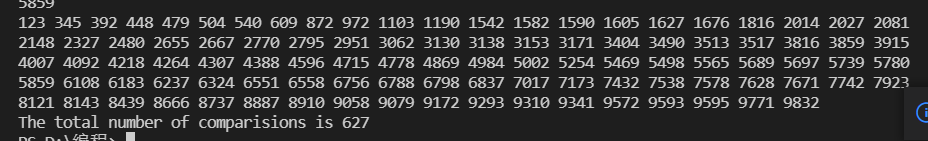


**3. 使用一个随机元素作为基准元素**

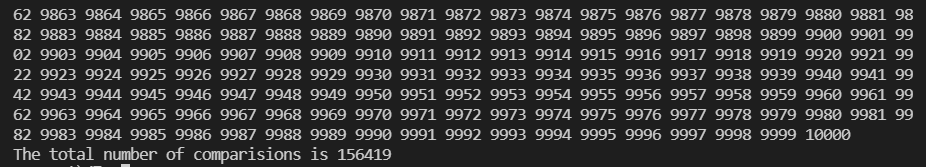
测试样例1



测试样例2



挑战数据集



**4. 使用“三取中”的方法选取基准元素**

测试样例1

如果不计算选择中位元素作为pivot时的比较次数：

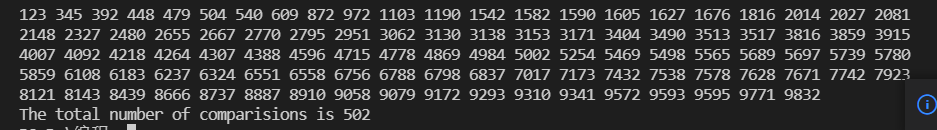


如果计算：

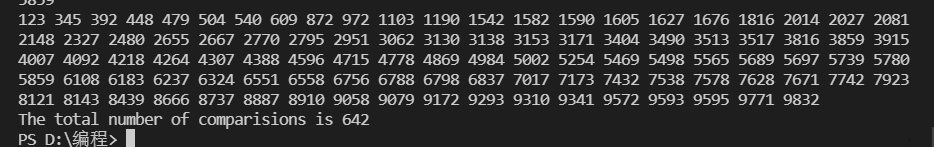


测试样例2

如果不计算选择中位元素作为pivot时的比较次数：

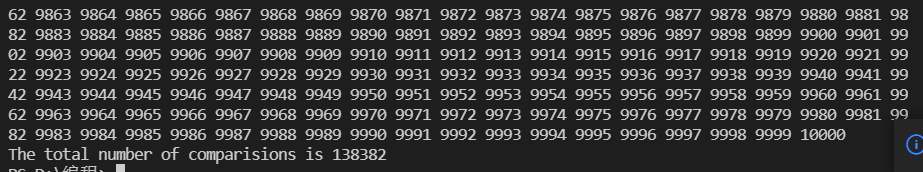


如果计算：

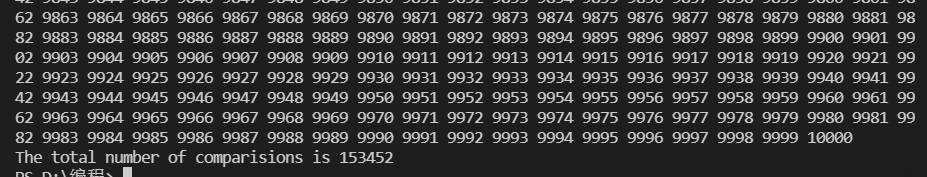


挑战数据集：

如果不计算选择中位元素作为pivot时的比较次数：



如果计算：

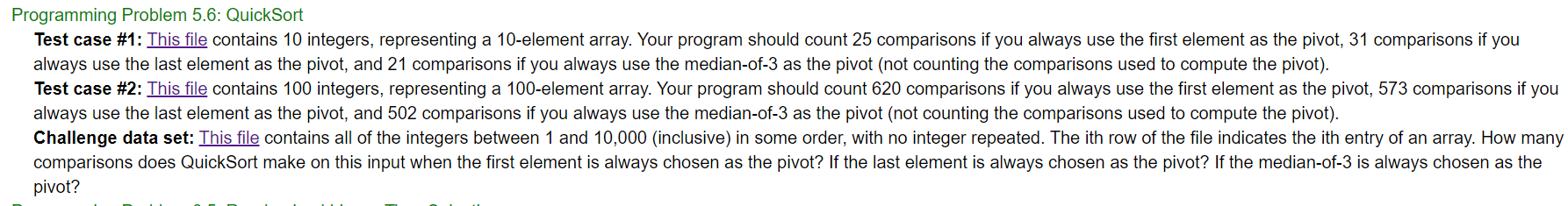


四、算法比较和分析

将上述运行结果制成如下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 选择方法 | | 输入 | | |
| 样例1 | 样例2 | 挑战数据集 |
| 第一个元素 | | 25 | 620 | 162085 |
| 最后一个元素 | | 31 | 573 | 164123 |
| 随机 | | 19 | 627 | 156419 |
| 三取中 | 不计算取中时的比较次数 | 21 | 502 | 138382 |
| 计算取中时的比较次数 | 35 | 642 | 153452 |

可以看到其结果与官网所给的数据是一致的，证明程序正确无误。



从以上结果中可以看出，就整体效果而言，随机选取法和三取中法都比固定位置（第一个或最后一个）选取法要好。因为固定位置选取法在最坏情况（数组已经有序）时，每次划分只能使待排序序列减一，快速排序沦为起泡排序，时间复杂度为Θ(n^2)。而随机选取法中，由于枢轴的位置是随机的，那么产生的分割也不会总是会出现劣质的分割，所以随机法可以对于绝大多数输入数据达到O(nlogn）的期望时间复杂度。三取中法则是通过首尾中间三个数的中间值代替整个数组的中间值，虽然不能完全与中间值相等，但是仍然能比较有效的分割。不过“取中”操作要进行很多次，相应的也会增加很多比较次数（尽管可以不计算）。

综上，QuickSort算法实现时还是选择随机选取pivot或“三取中”法选取pivot的效果更好。