**Project4实验报告**

**小组成员：陈诗翰、易鸣奇、张倬豪**

**1.总体功能描述**

第一题：

该程序通过快速排序算法对数据文件input.txt中的数字进行升序排序，并进行效率分析，最后将排列好的数字依次保存到另一文本文件output.txt中。

project整体要求：从一个数据文件中读入N个整数（N<=100000），且把结果保存到另一个文本文件中，并显示排序所用的时间。

第二题：

选择排序法对于小数组来说比归并排序快，尽管当数组变大时，情况相反。该程序实现了改进的排序算法，找到区别的大小数组的区分点，使之适应于各种大小的数组，把两种策略结合起来，对大数组用合并排序，对小数组使用选择排序。用这种方法重新实现函数SortIntegerArray。

**2.设计关键思路或方法**

**第一题：**

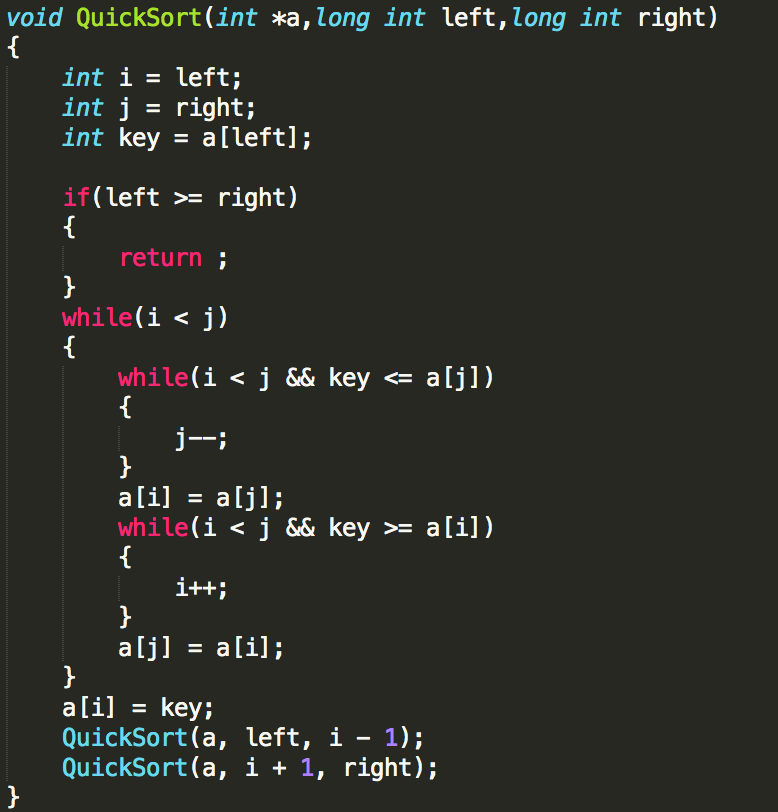
**Main()**

首先打开两个文本文件，当不能打开时，输出"File open error."并退出程序。然后从input.txt中不断读取数字，直到读到最后一个数字或者已经读到第100000个数字为止，记录读入数字个数。调用clock()记录排序开始时间，调用sort()进行排序，结束后再次调用clock()记录排序结束时间。将两个时间相减，得到排序所用时间，最后将排序好的指针从开头到尾部逐个输出到output.txt中。最后关闭两个文本文件，如果不能关闭，输出"Can not close the file."并退出程序。

程序主函数本身比较简单，不做截图，详见源代码文件。

**Void QuickSort(int \*a,long int left,long int right)**

首先对左右索引进行比较，如果左边索引大于或者等于右边的索引就代表已经整理完成一个组了。该条件不满足时，控制在当组内寻找一遍。首先从最右端开始向前寻找，直至找到一个小于key的数且i与j的大小关系没有反转，找到这样一个数后它赋给前面的被拿走的i的值（如果第一次循环且key是a[left]，那么就是给key）。接下来从最左端向后寻找，步骤与上一循环类似，但停止循环时啊a[i]与key的关系与上一循环相反。在改组找完一遍后，把中间数key回归，左右两边的数与key的大小关系相反。最后利用递归，用同样的方法对分出来的左右两个小组进行同上的做法，直到每一组的i==j为止。



**第二题：**

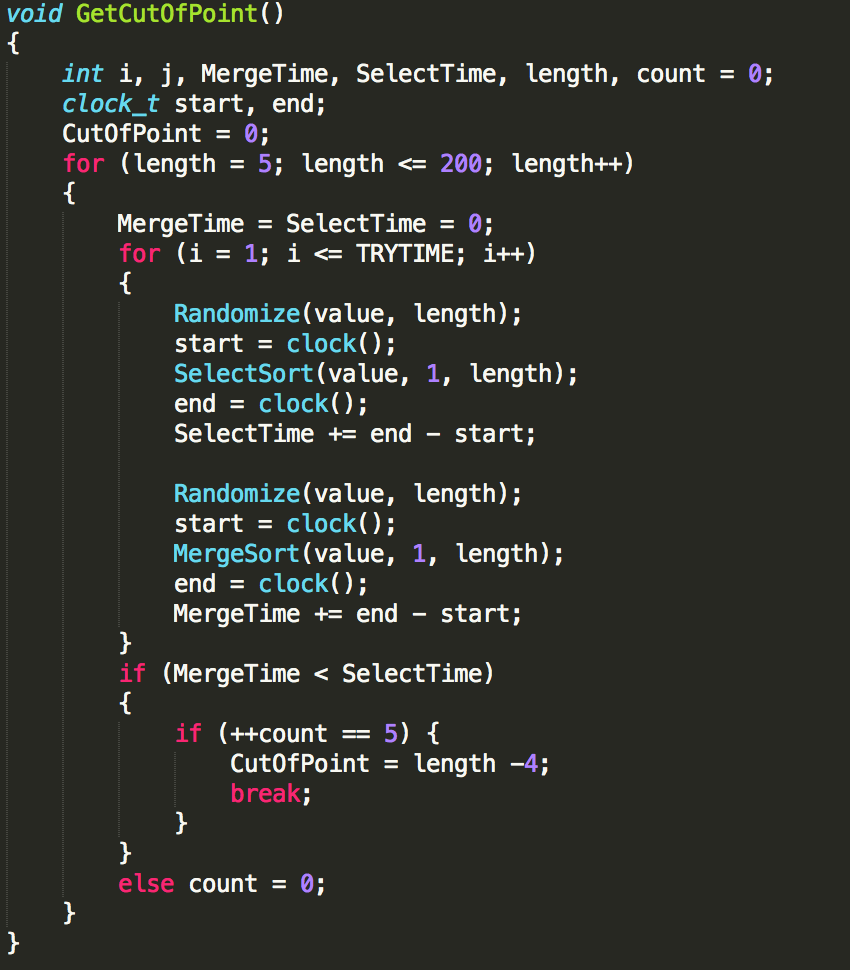
**Main()**

首先先调用GetCutOfPoint函数，根据题目所描述和要求的进行分界点的选择，下文会详细说明，接下来，与第一题程序类似，首先打开两个文本文件，当不能打开时，输出"File open error."并退出程序。然后从input.txt中不断读取数字，直到读到最后一个数字或者已经读到第100000个数字为止，记录读入数字个数。调用clock()记录排序开始时间，调用sort()进行排序，结束后再次调用clock()记录排序结束时间。将两个时间相减，得到排序所用时间，最后将排序好的指针从开头到尾部逐个输出到output.txt中。最后关闭两个文本文件，如果不能关闭，输出"Can not close the file."并退出程序。

程序主函数本身比较简单，不做截图，详见源代码文件。

**Void GetCutOfPoint()**

此函数利用随机生成的函数来生成一系列打乱的数字，并对其分别进行选择排序和归并排序，因为归并排序在小数组的时候仍然需要递归等程序开销。所以即使在时间复杂度比选择排序要小时，仍然还是选择排序更快，根据常识和查阅资料、自己试验，这个归并排序和选择排序快慢的数组大小分界点总是在200以内，所以我们设计了这个从小数组逐渐变大的算法，每一个数组长度计算并比较两种排序算法的快慢，重复10000次累计时间进行比较，防止偶然性误差，如果是选择排序较快，则循环继续，若是归并排序快并且连续出现了五次，则记录当前数组大小，退出循环并返回。上文中，重复10000次和连续出现五次分别为消除偶然性误差和随机数组带来的系统性误差，都是必要的操作。



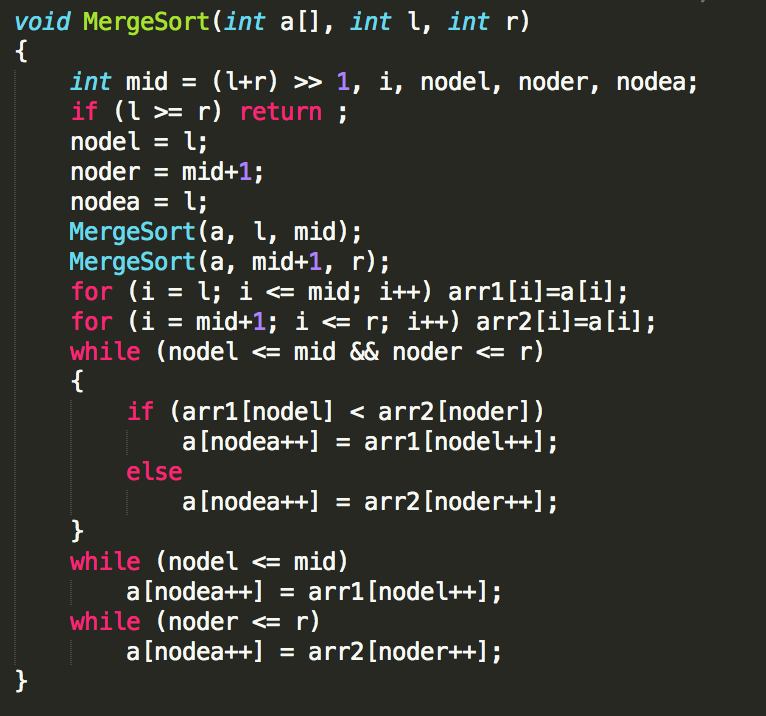
**Void MergeSort(int a[], int l, int r)**

此函数为经典的归并排序，归并排序是利用递归和分而治之的技术将数据序列划分成为越来越小的半子表，再对半子表排序，最后再用递归步骤将排好序的半子表合并成为越来越大的有序序列，归并排序包括两个步骤，分别为：

      1）划分子表

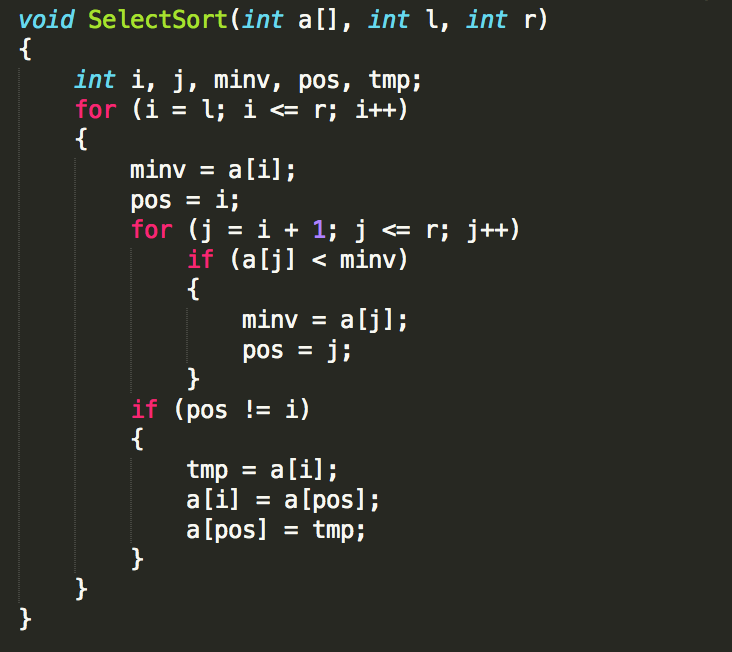
2）合并半子表

归并过程为：比较a[i]和a[j]的大小，若a[i]≤a[j]，则将第一个有序表中的元素a[i]复制到r[k]中，并令i和k分别加上1；否则将第二个有序表中的元素a[j]复制到r[k]中，并令j和k分别加上1，如此循环下去，直到其中一个有序表取完，然后再将另一个有序表中剩余的元素复制到r中从下标k到下标t的单元。归并排序的算法我们通常用递归实现，先把待排序区间[s,t]以中点二分，接着把左边子区间排序，再把右边子区间排序，最后把左区间和右区间用一次归并操作合并成有序的区间[s,t]。



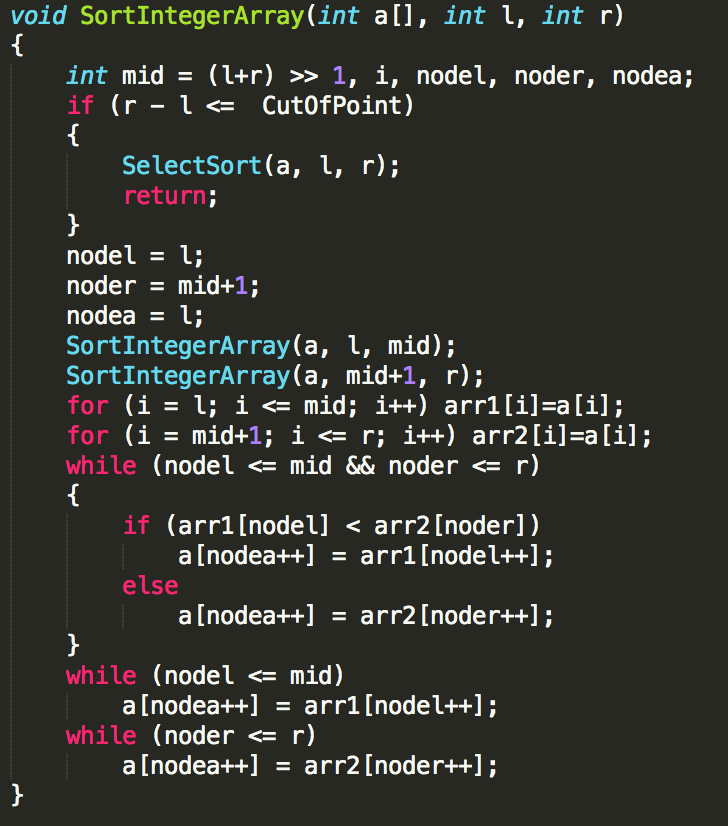
**Void SelectSort(int a[], int l, int r)**

此函数为经典的选择排序，工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的数据元素排完。实现起来比较简单，在此不做赘述。



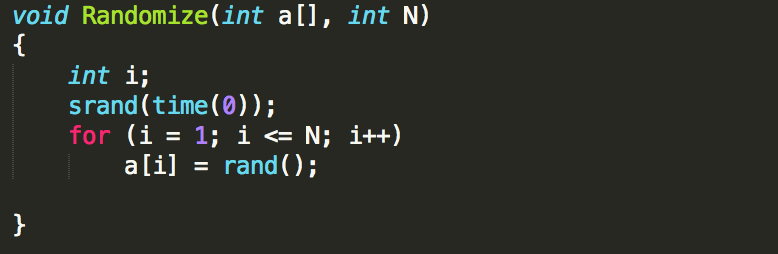
**Void SortIntegerArray(int a[], int l, int r)**

此函数为关键的排序程序，框架其实为归并排序的递归框架，只是改成了在递归数组长度小于分界点（前面求出，GetCutOfPoint）时，调用的是选择排序的函数。其余与前文描述的归并排序的递归部分算法实现基本一致。

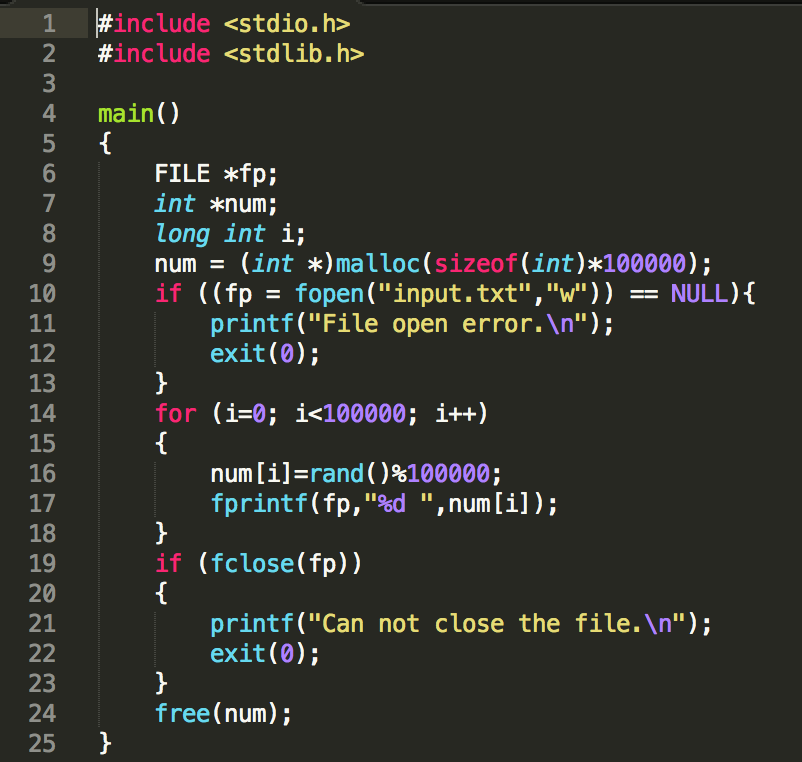


**Void SortIntegerArray(int a[], int l, int r)**

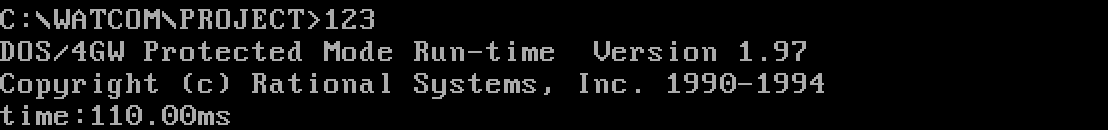
这是随机生成测试分界点数组的函数，比较简单，不做赘述。

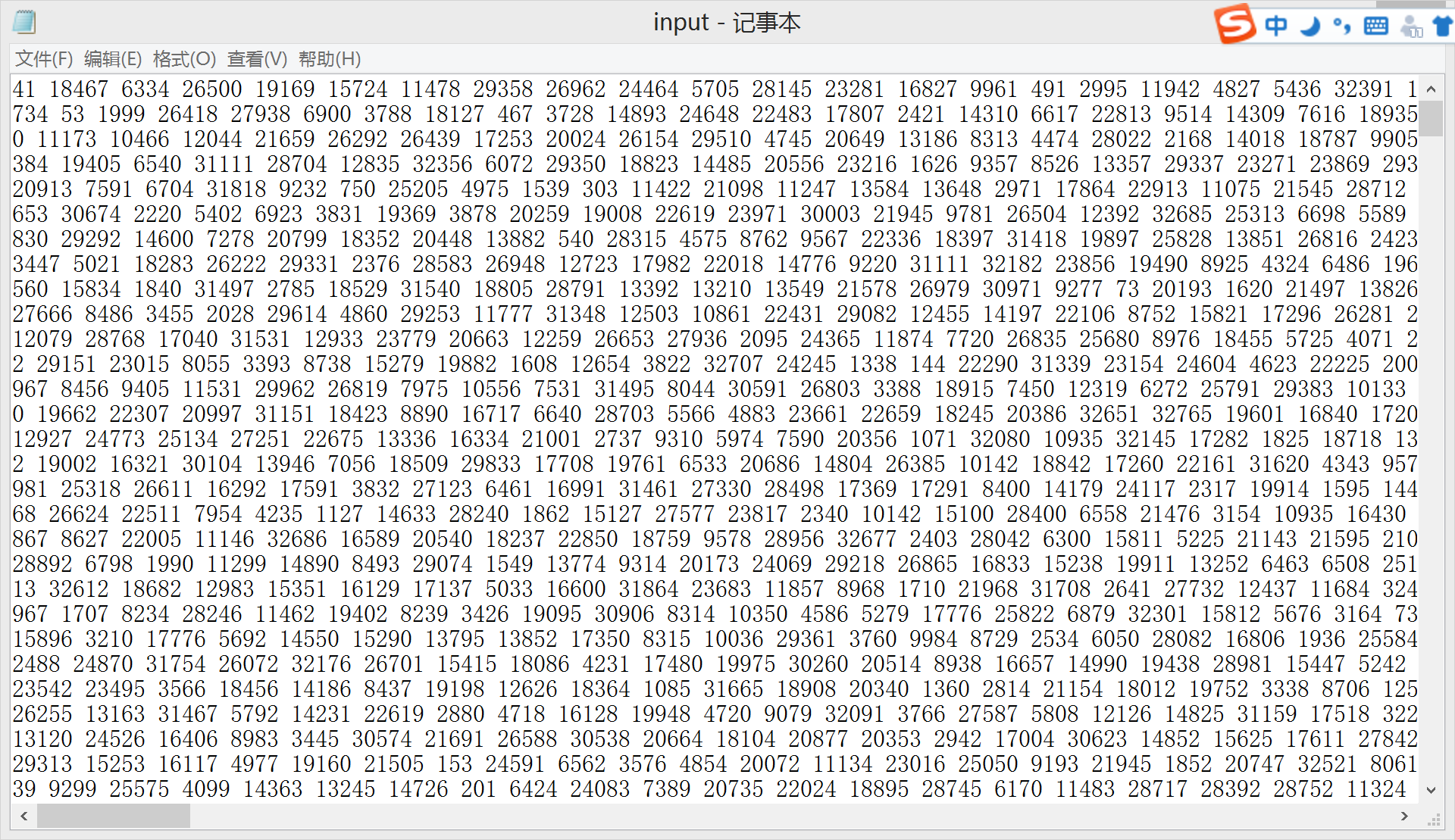


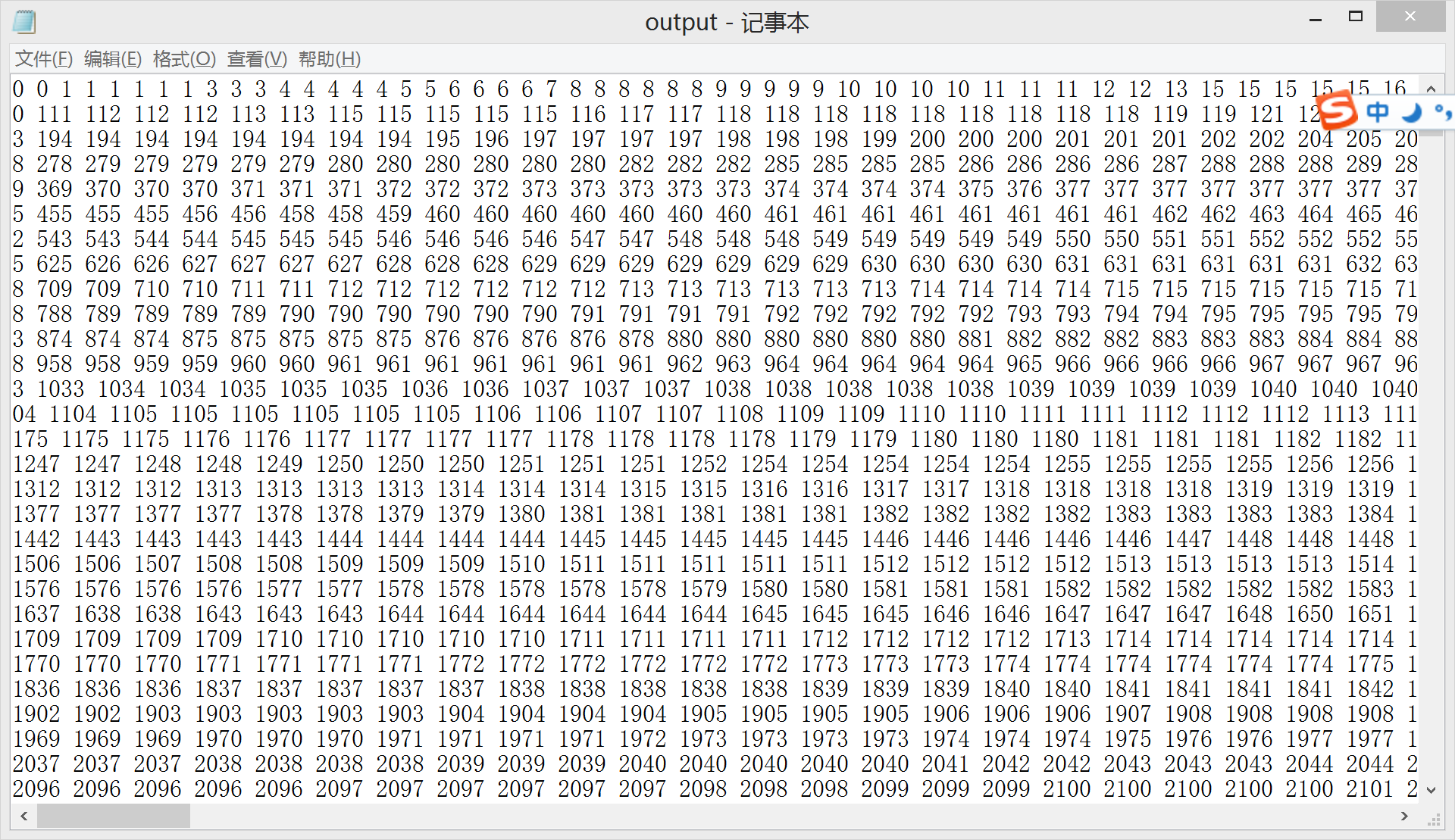
**3.测试报告**

为了生成大量的测试数据，我们写了testing.c这个生成测试数据的程序，将随机生成的100000个随机数组数据存入input.txt以便主程序调用使用。

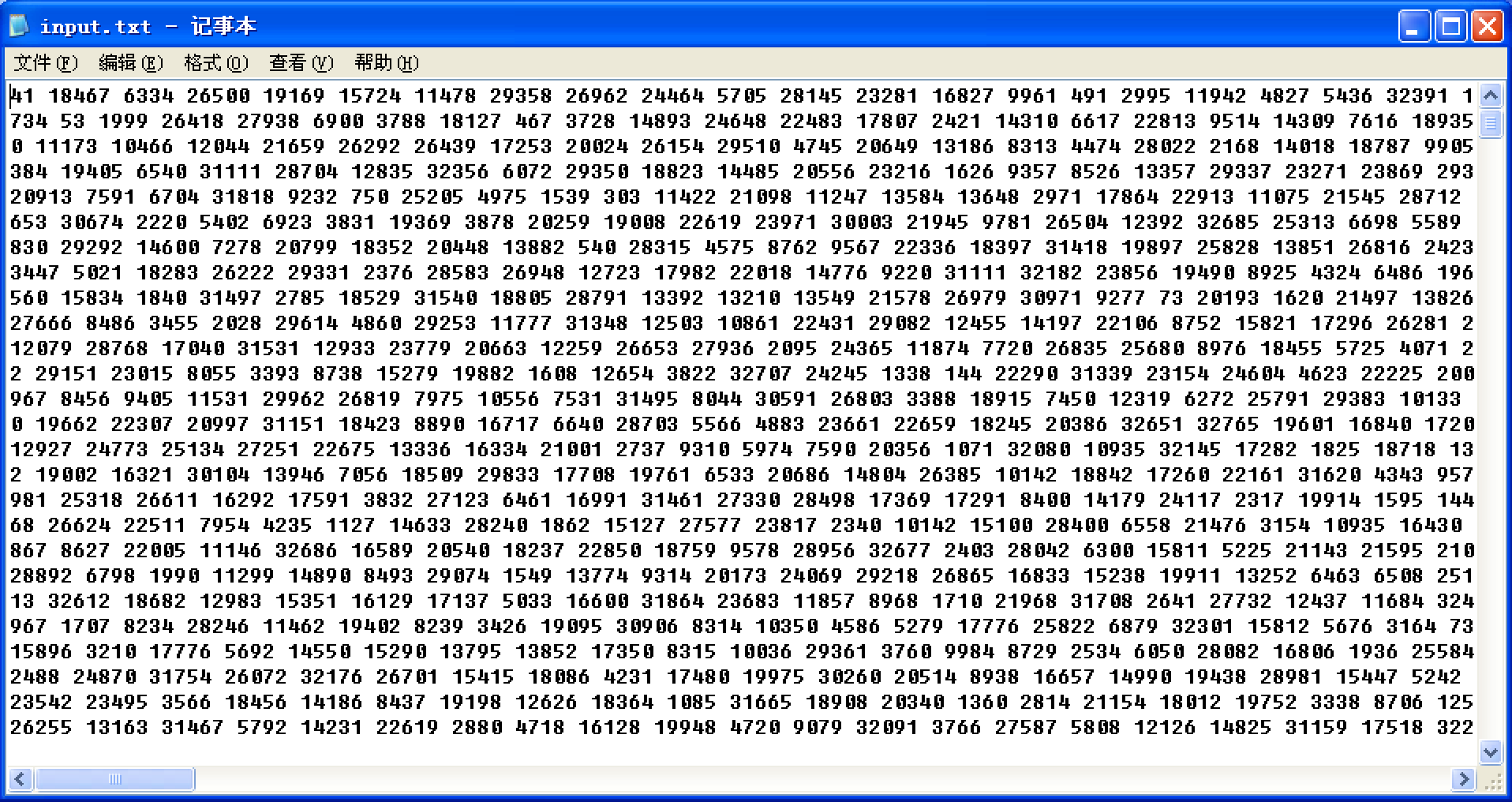
**第一题：**

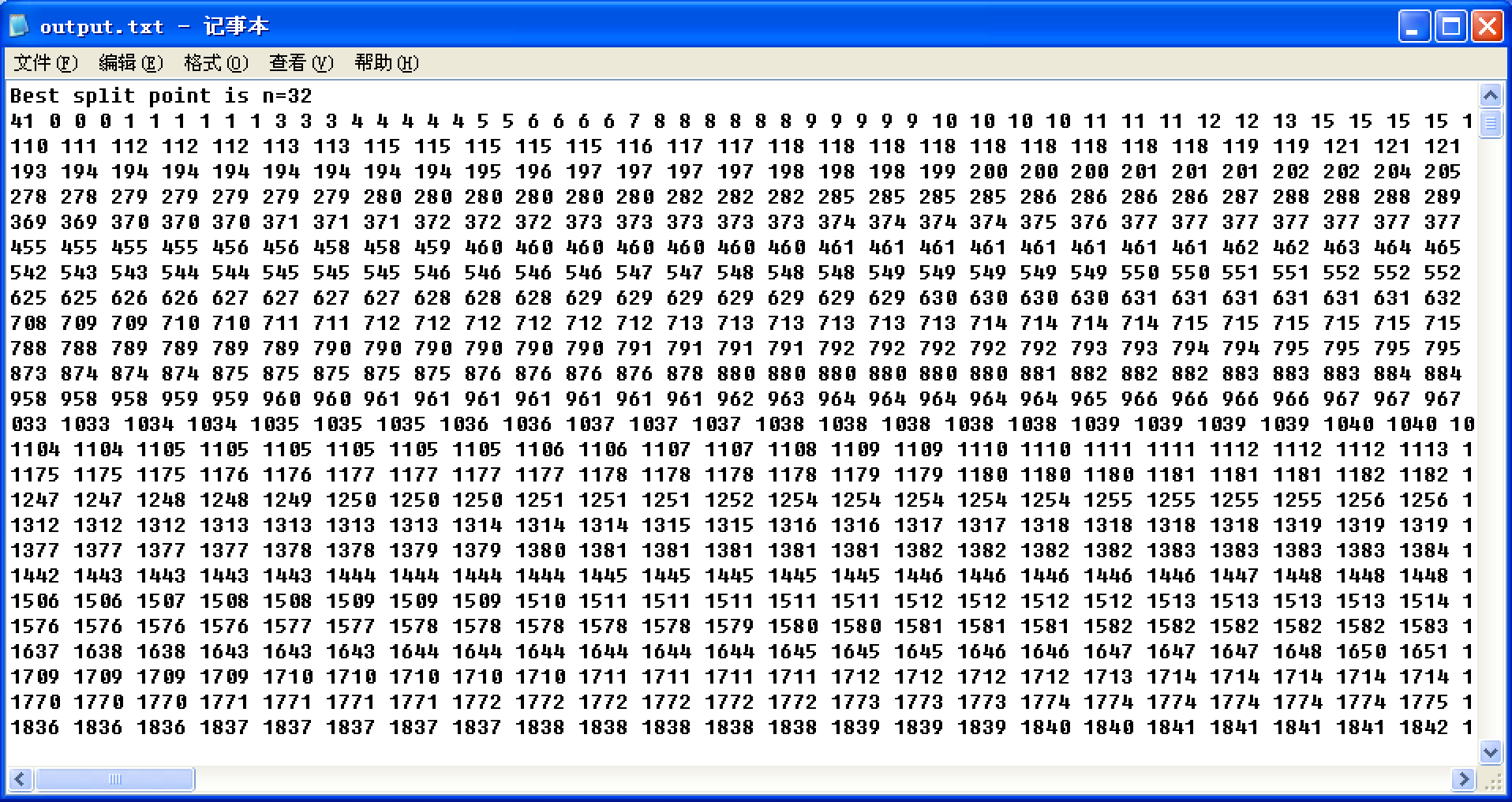


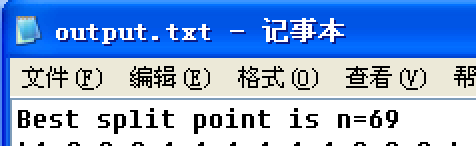


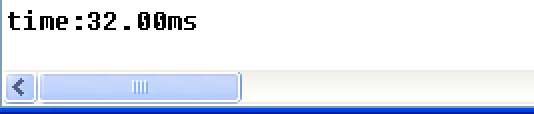
../../Desktop/屏幕快照%202016-06-29%20下午10.14.02.png

**第二题：**









**4.开发体会**

在这一次的project第一题编写过程中，我们了解了快速排序算法这一经典算法，认识到了该算法在处理大量数据时相对冒泡排序，选择排序等算法在消耗时间上的优势。与此同时，这一次的project也让我们加深了对于一段标准的具有可移植性的代码的理解与运用。

在这一次的project第二题编写过程中，我们了解了选择排序和归并排序两种归并排序的融合应用，认识到了他们在时间复杂度上的不同和在实际应用当中与理论的不同，与此同时，这一次的project也让我们加深了对于一段标准的具有可移植性的代码的理解与运用。

**任务职责及互评（不分先后）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 角色 | 学号 | 姓名 | 分数 | 职责 |
| 组长 | 3150102418 | *张倬豪* | *3* | * 总体策划，负责协调活动 * 组内成员任务分工 * 代码实现 * 测试 * 撰写《project报告》 |
| 组员 | 3150102390 | *易鸣奇* | *3* | * 信息收集 * 记日志、拍照、视频 * 代码实现 * 测试 * 撰写《project报告》 |
| 组员 | 3150103690 | *陈诗瀚* | *3* | * 信息收集 * 记日志、拍照、视频 * 代码实现 * 测试   撰写《project报告》 |