**数据结构与算法专题实践报告**

**Project2**

1、源代码

见code文件夹中的源文件

2、问题描述/需求分析

编写代码实现外部排序（对大量数据进行排序）

需要模拟磁盘-内存操作，外部排序使用归并排序实现，~~初始顺串用插入排序生成~~

初始顺串改用败者树生成，这样可以生成尽可能长的变长顺串

初始顺串生成时加入多线程，使磁盘交互与败者树操作并行

归并时使用最佳归并序

3、系统结构/算法思想

使用vector模拟磁盘和内存缓冲区，用input/outputBufferSize来限制内存缓冲区大小

编写专用的函数来实现模拟的磁盘-内存操作，如内存读入、内存输出

~~使用简单的插入排序生成初始的顺串~~

使用败者树来生成初始的顺串

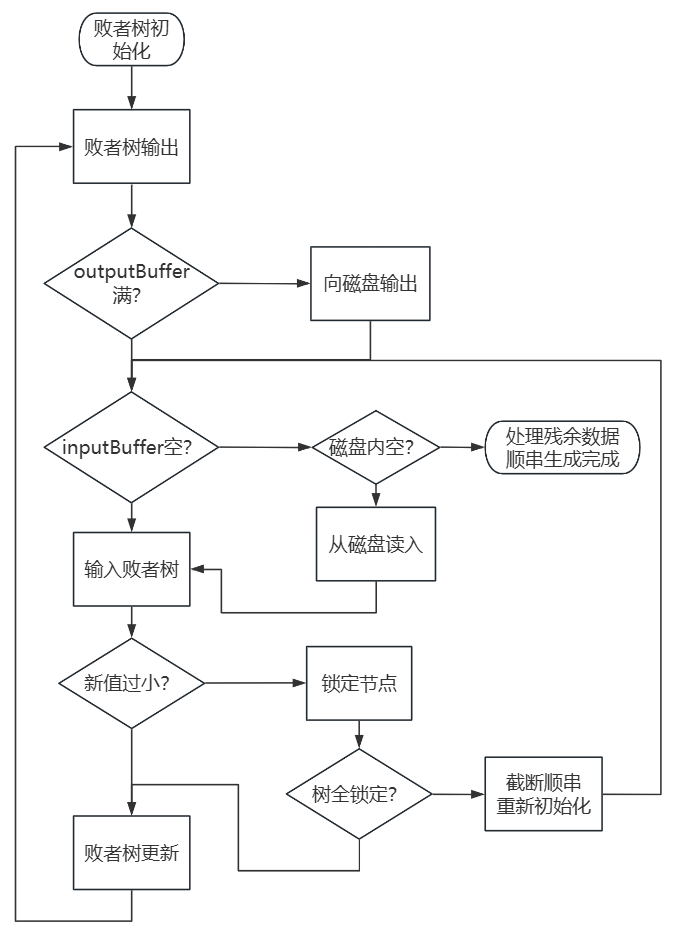
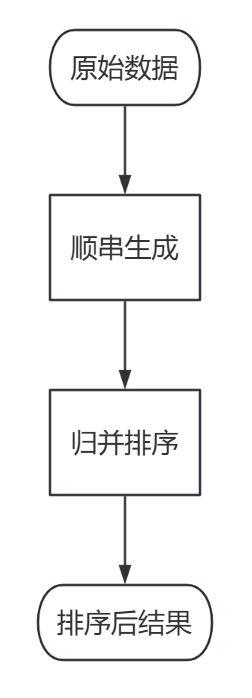
在败者树生成过程中加入多线程，保证在磁盘交互时败者树能保持运转状态

对顺串不断执行二路归并排序来实现最终的排序

二路归并时使用一个按顺串长度排序的优先队列来提供最佳归并序（类似哈夫曼树）

使用简单的随机数生成器来生成需要排序的数据，按上述顺序进行操作即可实现

4、功能模块设计

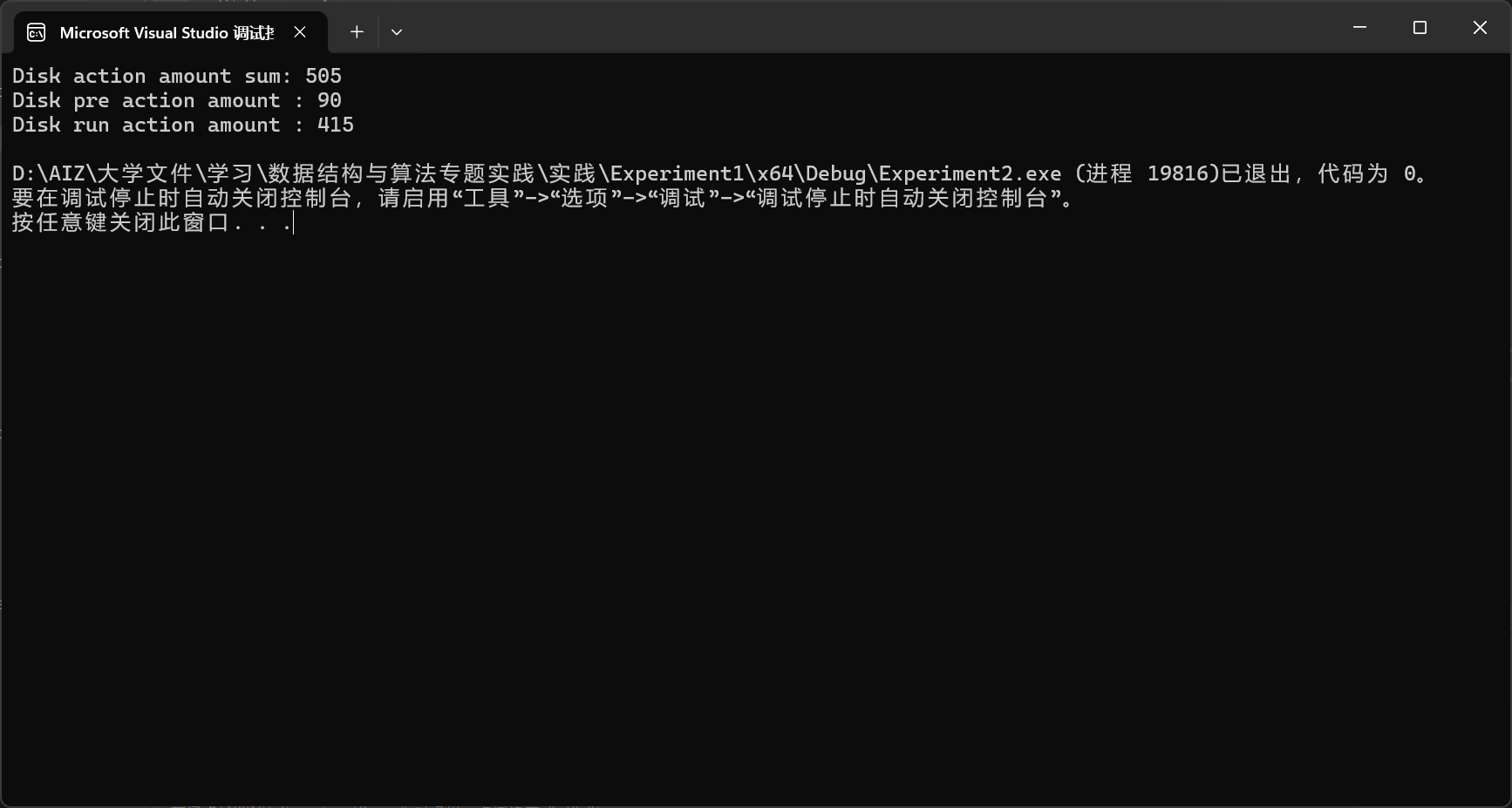


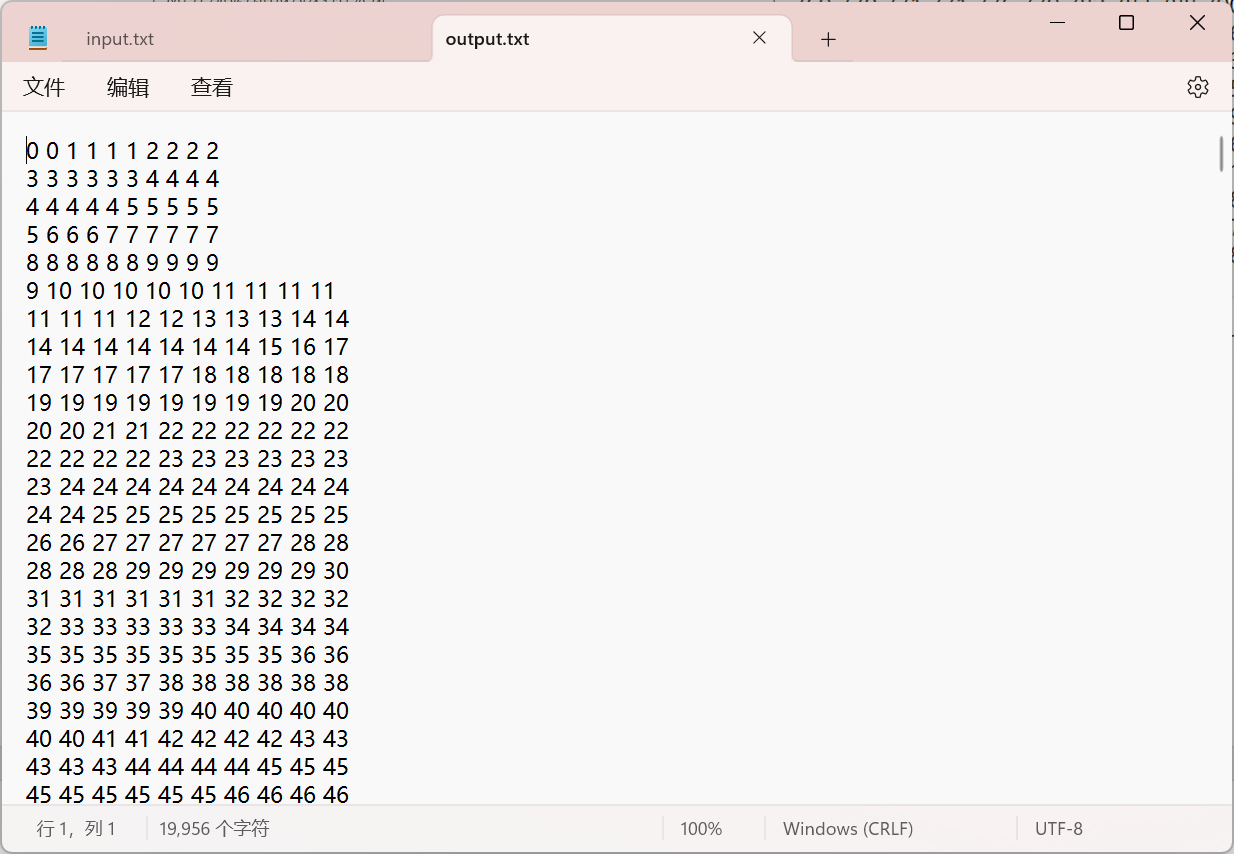
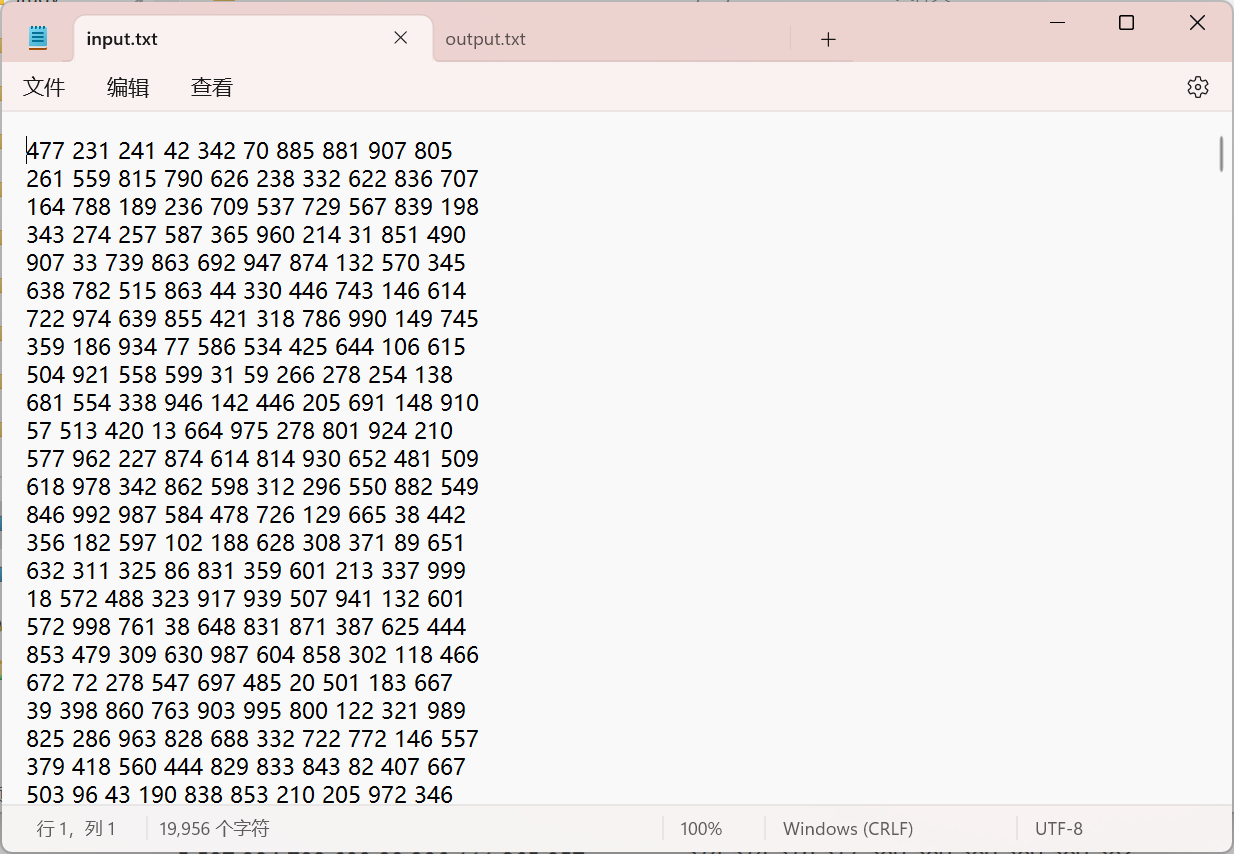
上图中，右图即为顺串生成过程中的败者树流程图，其中菱形框右侧为判断为Yes的情况，下方为判断为No的情况

败者树中的核心为败者树初始化函数、败者树更新函数、败者树节点锁定和败者树输入输出，实验2中仅为了实现要求所以还未对这部分函数进行模块化，所以该过程全部展开于顺串生成函数中

多线程的实现使用多个thread，在内存输入/输出的时候启动对应线程，同时在下次进行相同操作时执行join()来保证上一次线程执行完成且被正确释放，同时添加对应的互斥锁来保证其的内存访问不会冲突

5、测试结果与分析





对与实验1相同的数进行模拟外部排序，结果显示在归并过程中进行了505次磁盘-内存操作，其中顺串生成时进行了90次，归并时进行了415次

这里使用败者树优化顺串生成后对归并的磁盘-内存操作数优化并不明显，推测是由于实验1中的顺串是BufferSize的整数倍长，所以减少了磁盘-内存操作，而败者树生成的变长顺串往往是不规则的，所以在某些情况下对磁盘-内存操作的优化不那么明显

下图为对5000个随机数进行外部排序的结果，由于数据量较大这里只展示一部分

6、实验总结

编写实验2代码的过程中让我对败者树数据结构有了更深刻的认识，可惜的是这里败者树的Buffer使用了4个Buffer，两个inputBuffer和两个outputBuffer，理论上与败者树进行交互的inputBuffer和outputBuffer是可以合并成一个Buffer的，但由于时间问题和多线程代码问题最终没能实现

多线程代码编写过程中遇到了很多问题，由于多线程在单步调试中不好进行追踪，而且常常出现内存访问错误，所以在编写过程中经常推倒重来。最后在多方学习后采用join()进行线程资源管理，在join()处等待线程执行完成（若线程未执行完成）并释放资源，并且在操作进行完后对所有线程进行一次判断和释放，保证所有线程资源都被正确释放