# 数据结构与算法专题实践2

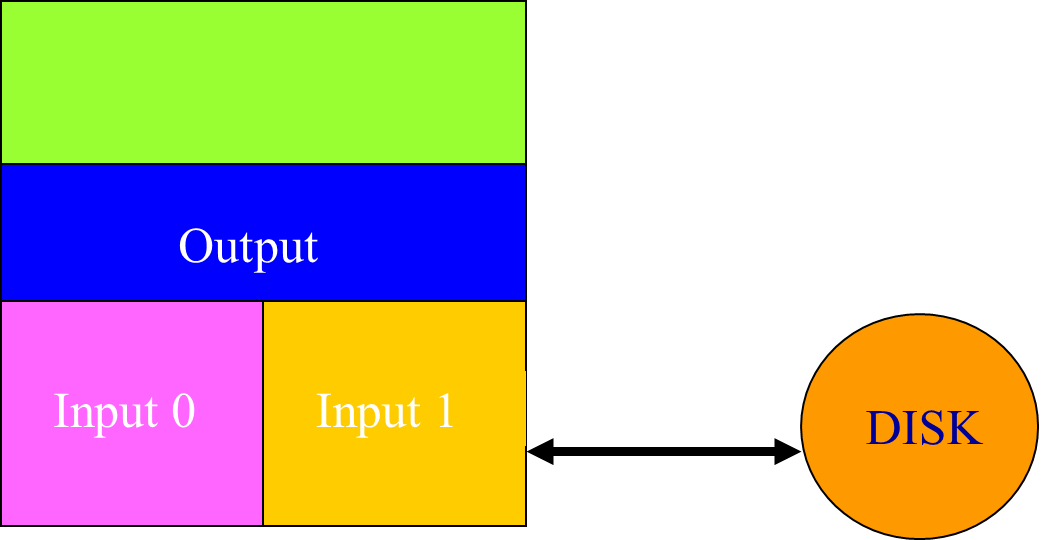
## 1．问题描述/需求分析

1. 数据文件组织与管理（中间结果）
2. 缓冲区管理
3. 2路归并树
4. 基于归并树的归并外排算法

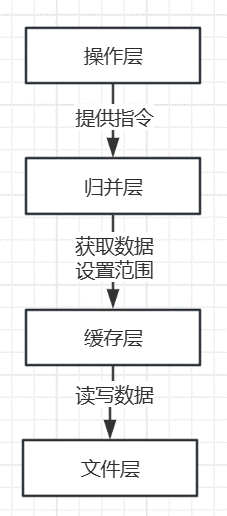
## 2．系统结构/算法思想

### 基本思路：

基本思路为：设置两个输入缓冲区存储需要归并的两个组。将归并的结果存放至输出缓冲区中。输入缓冲空后从文件读取，输出缓冲区满后向文件写入。



### 系统框架：



### 描述各模块功能及其关系：

**操作模块：**包含main函数，向矩阵运算模块提供操作指令

**归并运算模块：**包含外部归并排序算法，按照操作模块提供的指令进行运算。按照操作模块提供的指令进行运算。通过缓存模块提供的接口从中获取数据及消息。

**缓存模块：**为运算模块提供数据。在访问未越界的情况下，数据缺失时自动从文件中读取数据，保障数据一致性，使得运算模块如同直接访问文件一般访问数据。在访问越界的情况下返回消息至运算模块。

**文件模块：**存储矩阵内容以及相关信息。

## 3．功能模块设计

### 模块设计思想：

**操作模块：**包含main函数，向运算模块提供操作指令。包含随机生成测试样本的函数。

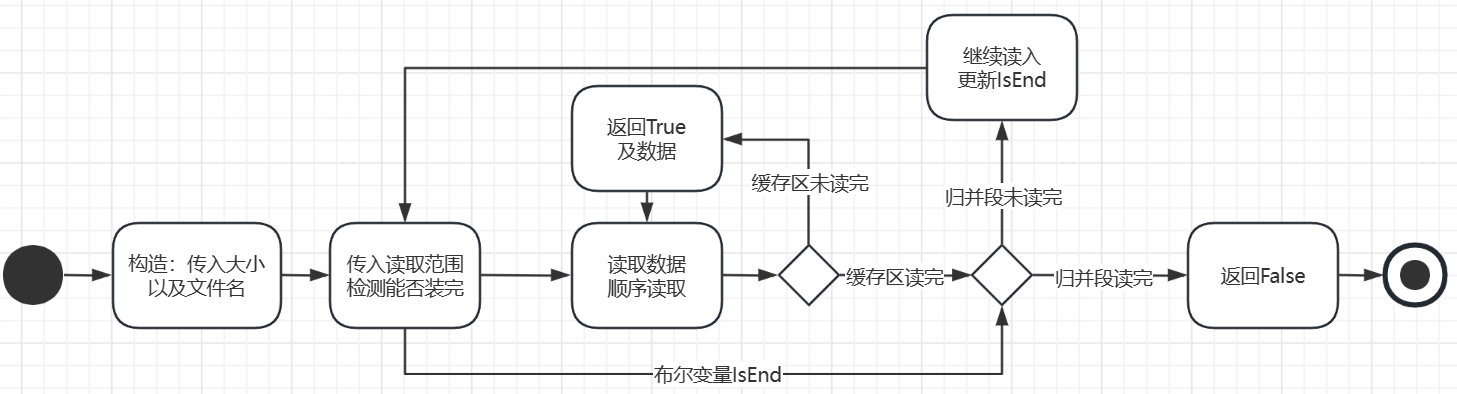
**归并运算模块：**即MergeSort类。主要包含外部归并算法，并为操作模块提供打印接口，将运算结果通过屏幕进行图形化输出，可选输出运算过程。在向缓存模块设置读取范围后按序从缓存模块获取数据。在收到访问越界的消息后开始下一轮归并。

**缓存模块：**即Buff类。提供接口为运算模块提供数据。为减少临时文件的生成，缓存采用双文件互换操作，至多操作两个文件。具体内容见流程图

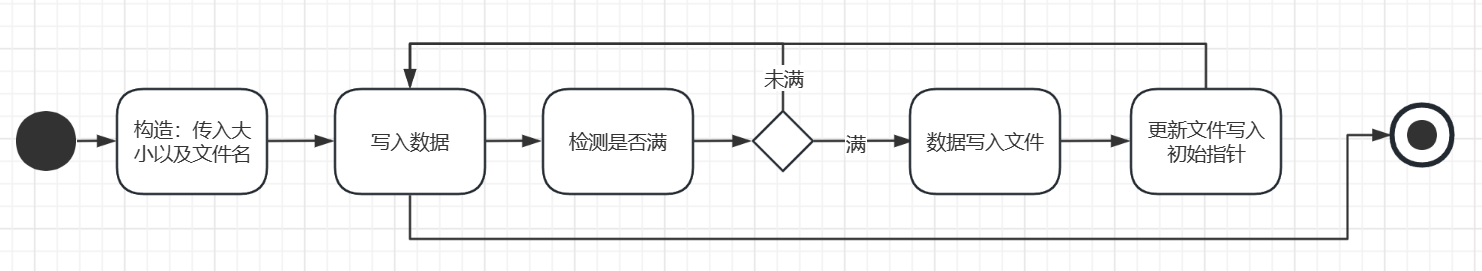
**文件模块：**矩阵以二进制形式(.bin)存储于文件中，按顺序以整型存储的排序内容。缓存模块通过文件指针的操作访问数据。提供临时文件清理代码。

### 缓存流程图：

读取缓冲区



写入缓冲区

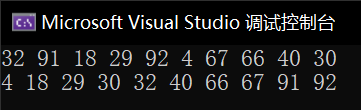


## 4．测试结果与分析

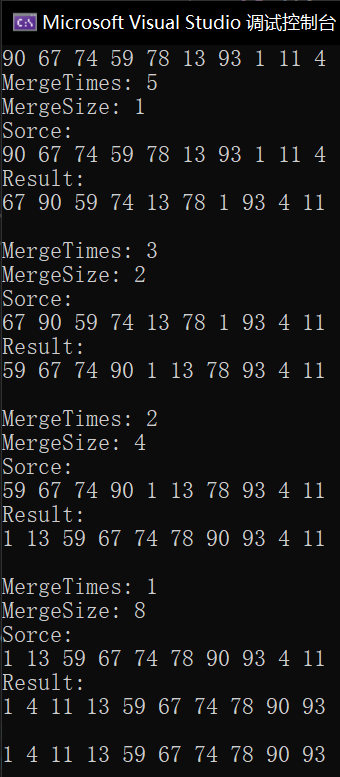
**测试数据选择或生成办法：**通过rand()随机算法生成指定长度与范围的测试数列。

**运行结果截图：**

**结果图：**



**过程图：**



**性能分析：**

如过程图所示， 归并大小较小时，算法会进行多次归并，此时缓冲区并未完全利用而且会产生多次文件访问，性能较差。

## 5．实验总结

本次实验模拟实现外部归并算法，为后续的改进实验提供基础。在本实验中，拥有两个输入缓冲区以及一个输出缓冲区，通过基础的迭代归并算法进行排序，文件访问次数较多，性能较差。

在实验的过程中遇到了以下问题：

问题：如何确定归并次数以及每次归并缓冲区的读写位置？

方案：

需要归并 fileLength / ( 2 \* curMergeSize) 次,向上取整。每次归并 2 \* curMergeSize 个数 两个缓存各读入 curMergeSize 个数。

每次交换文件后，第1次归并时，第一个缓存读入 0 ~ curMergeSize - 1 个数，第二个缓存读入 curMergeSize ~ 2 \* curMergeSize - 1 个数。第2次归并时，第一个缓存读入 2 \* curMergeSize ~ 3 \* curMergeSize - 1 个数，第二个缓存读入 3 \* curMergeSize ~ 4 \* curMergeSize - 1 个数。第3次归并时，第一个缓存读入 4 \* curMergeSize ~ 5 \* curMergeSize - 1 个数，第二个缓存读入 5 \* curMergeSize ~ 6 \* curMergeSize - 1 个数……

第n次归并时，第一个缓存读入 (2 \* n - 2) \* curMergeSize ~ (2 \* n - 1) \* curMergeSize - 1 个数，第二个缓存读入 (2 \* n - 1) \* curMergeSize ~ 2 \* n \* curMergeSize - 1 个数。

如果文件不够长，则第二个缓存读入的数目可能不足 curMergeSize 个数，此时出现单支现象，单支现象出现在第二个缓存读入的数目不足 curMergeSize 个数时，此时第一个缓存读入的数目一定是 curMergeSize 个数

## 6. 源代码

MergeSort.h //归并排序类，内联，包含归并排序算法

Buffer.h //缓存类，内联，包含写入读取以及自动写回

Definition.h //宏定义，缓存大小与矩阵大小的定义

main.cpp //主程序文件，包含main函数