# 数据结构与算法专题实践1

## 1．问题描述/需求分析

1. 构建并管理计算机缓冲区模拟器
2. 组织与管理文件中的大型二维数组乘法
3. 统计运算过程中的缓存区缺失次数
4. 绘制缓存区缺失次数与缓存区大小关系图像
5. 绘制缓存区缺失次数与矩阵大小关系图像

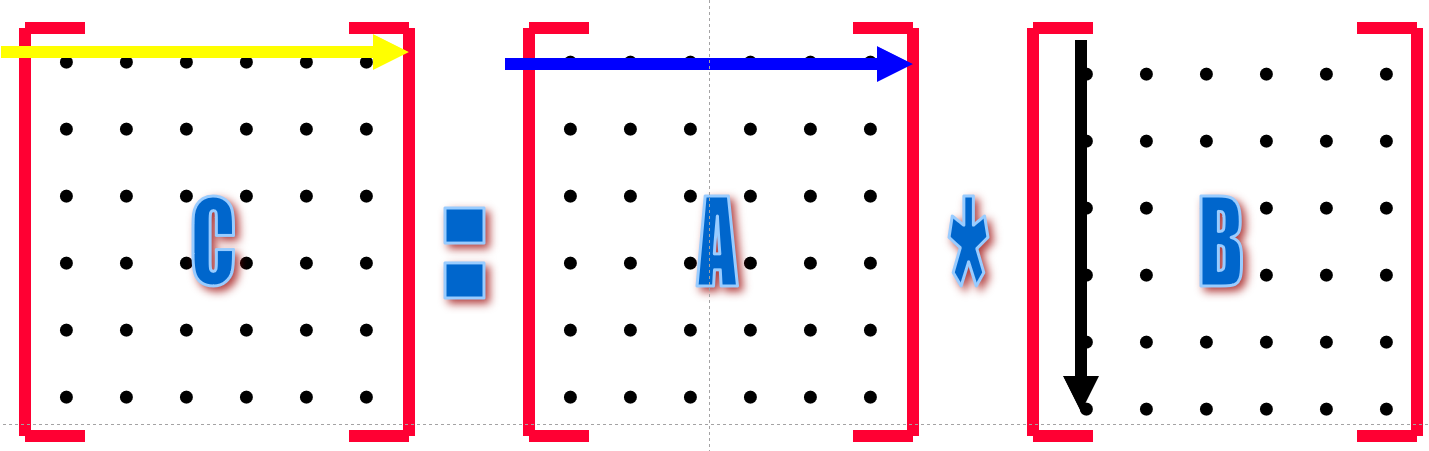
## 2．系统结构/算法思想

### 基本思路：

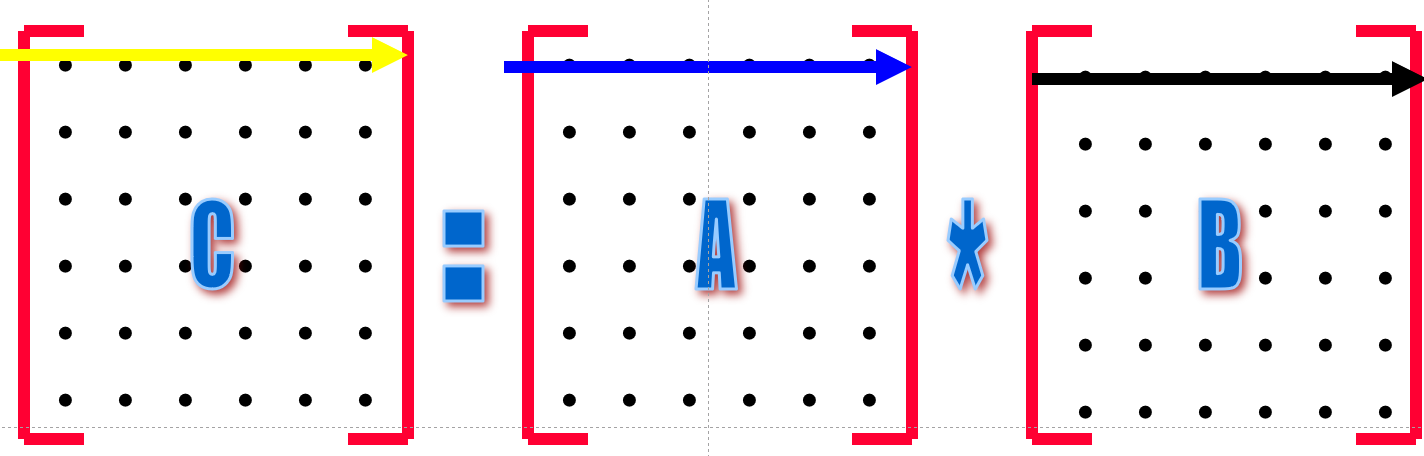
基本思路为：运算层按照操作层提供的指令运算通过缓冲区获取的数据。

运算层包含两种算法：ijk矩阵乘法算法以及ikj矩阵乘法算法

ijk矩阵乘法算法的数据访问过程如图所示：



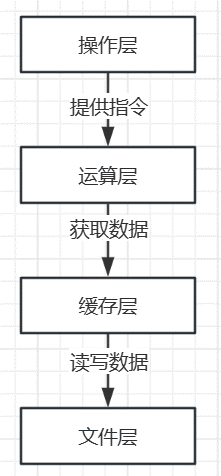
ikj矩阵乘法算法的数据访问过程如图所示：



缓冲区采用页替换模式，出现未命中情况后以缺失位置为基准更新整个缓冲区。

由图可得，ikj矩阵乘法算法数据访问更加规律，因此更容易命中缓冲区。

### 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

**矩阵运算模块：**包含两种矩阵乘法算法，按照操作模块提供的指令进行运算，从缓存模块获取数据。

**缓存模块：**为矩阵运算模块提供数据，在数据缺失时自动从文件中读取数据，保障数据一致性，使得矩阵运算模块如同直接访问文件一般访问数据。

**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：

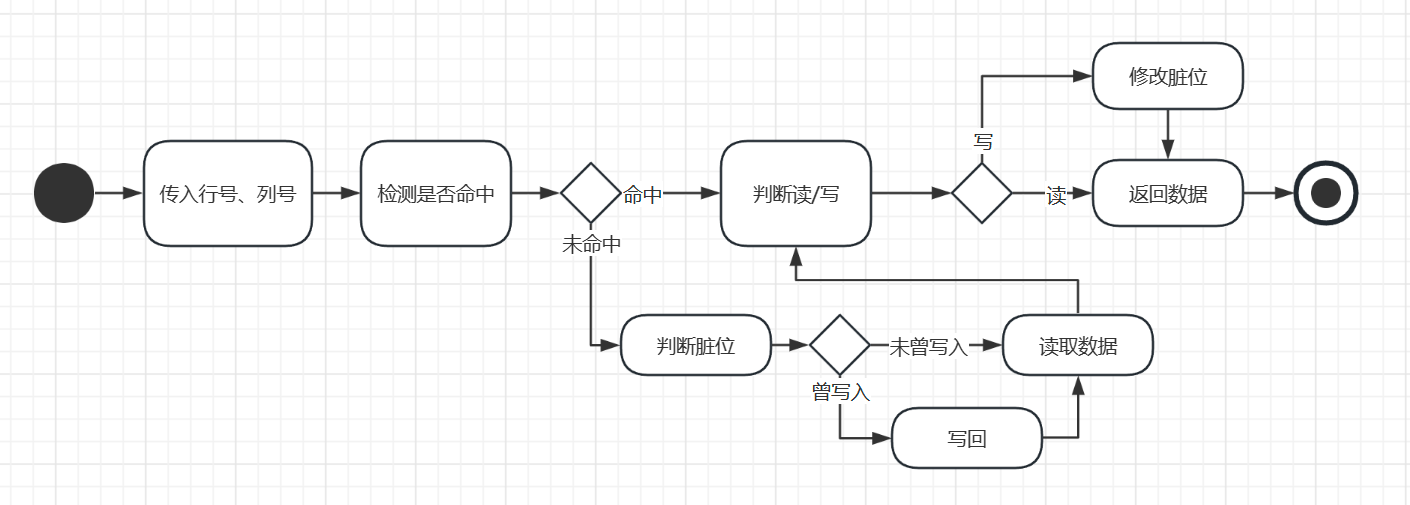
**操作模块：**包含main函数，向矩阵运算模块提供操作指令，将运算结果通过屏幕进行图形化输出，同时输出缓存分析。

**矩阵运算模块：**即Matrix类。主要包含两种矩阵乘法算法，并为操作模块提供打印接口和数据构建接口。按照操作模块提供的指令选择算法进行运算。通过缓存模块提供的接口从中获取数据。

**缓存模块：**即Buff类。为达到快速检索的目的，底层使用哈希图。提供getNum()与setNum()接口为矩阵运算模块提供数据。通过刷新操作在数据缺失时自动从文件中读取数据。通过脏位保障数据一性，在缓冲区内容被修改后设置为脏，从而在刷新数据前将数据写回。使得矩阵运算模块如同直接访问文件一般访问数据。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，前两个数据为整型的行与列，之后是按顺序以双精度浮点型存储的矩阵内容。缓存模块通过文件指针的操作访问数据。

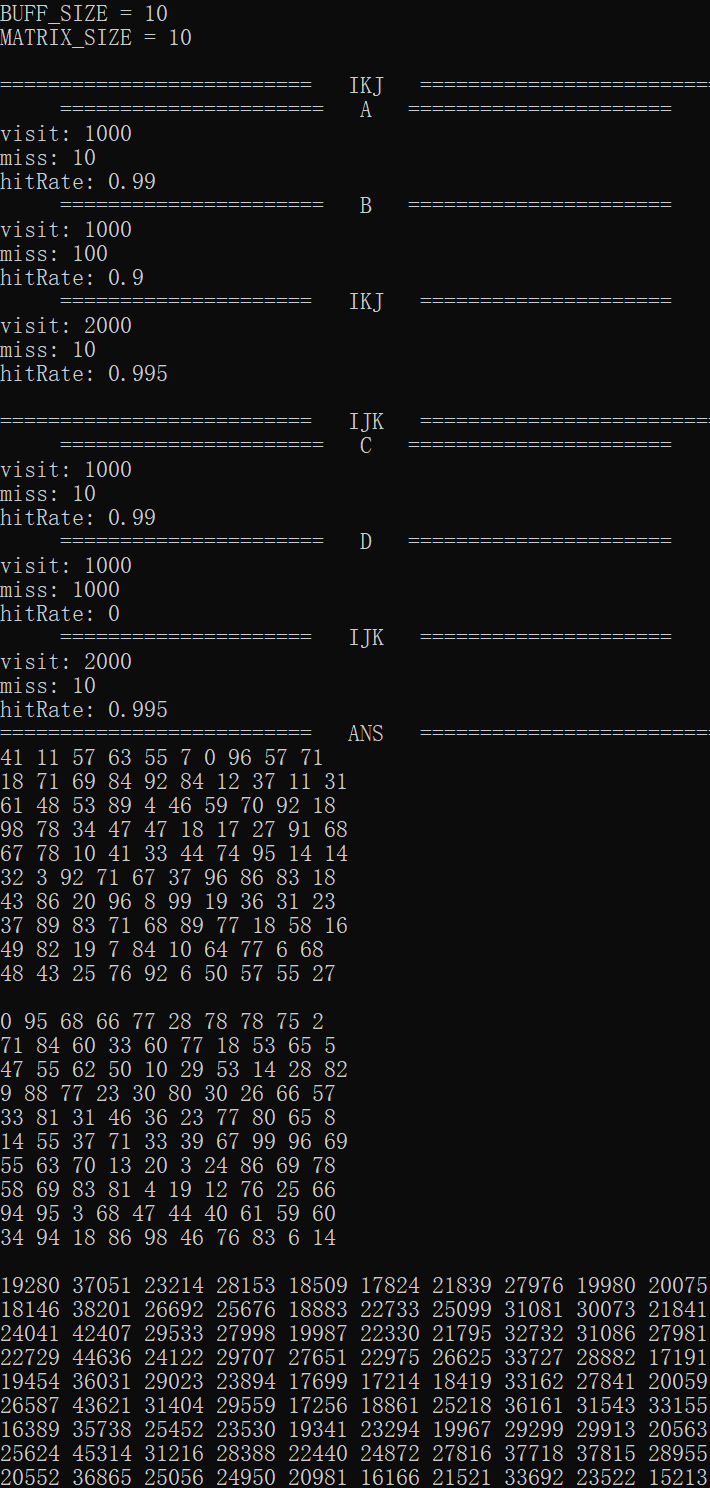
### 缓存流程图：



## 4．测试结果与分析

**测试数据选择或生成办法：**通过rand()随机算法生成指定大小的测试矩阵。

**运行结果截图：**



**性能图：**

如图所示，IKJ算法缺失次数整体小于IJK算法。初始情况下IKJ算法的缺失次数随着缓冲区的增加变化明显。在缓冲区接近列数后两个算法的缺失次数趋于减缓。而当缓冲区大小达到列数后，两者的缺失次数再次出现大幅下降的情况。而当缓冲区大小超过列数后，IJK算法的缺失次数发生骤降后稳定，而IKJ算法的缺失次数开始波动式缓慢下降。

如图所示，IKJ算法缺失次数整体小于IJK算法。两者缺失次数的变化可以用二阶多项式完美拟合(R2>0.99)，拟合IJK算法的二阶多项式系数显著大于IKJ算法，表现为更快的上升。

如图所示，命中率的变化趋势非常接近缺失次数的变化趋势，故不再讨论。

## 5．实验总结

本次实验通过模拟外部大矩阵乘法，探究了矩阵规模、缓存大小对缓存命中率的影响，进而学习算法和缓存之间的相互作用。在本实验中，ikj矩阵乘法算法在大多数情况下都快于ijk矩阵乘法算法，同时，随缓存区大小的增加，ikj矩阵乘法算法的命中率快速上升。

在实验的过程中遇到了以下问题：

问题：文件在读取、写入时会清空原有文件

方案：通过查找资料可知，ofstream默认的ios::out文件操作类型会清空原有文件内容，若使用ios::in|ios::out类型则可以保留文件内容的同时进行IO操作，故需添加ios::in参数。

问题：缓存区是否应当使用最近最久未使用算法LRU算法

方案：通过分析算法特性可得，使用LRU算法不但会增大程序编写难度，更会导致缓存命中率下降，因此舍弃LRU算法。

## 6. 源代码

Matrix.h //矩阵类，内联，包含两种矩阵乘法算法

Buffer.cpp //缓存类实现

Buffer.h //缓存类，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

MatrixMultiplication.cpp //主程序文件，包含main函数

# 数据结构与算法专题实践2

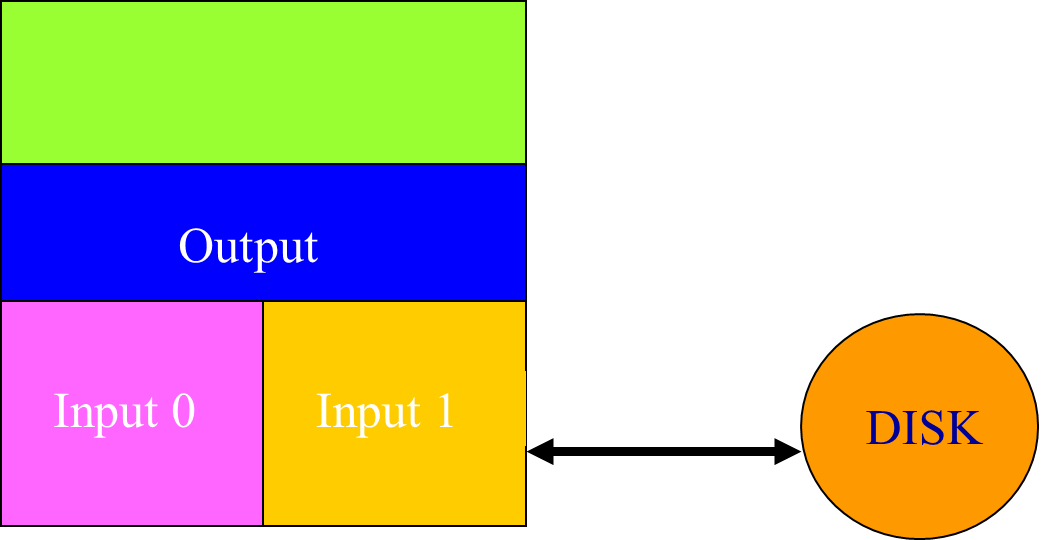
## 1．问题描述/需求分析

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

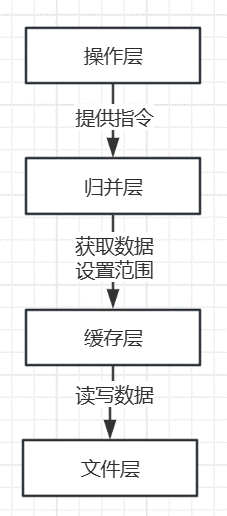
## 2．系统结构/算法思想

### 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。

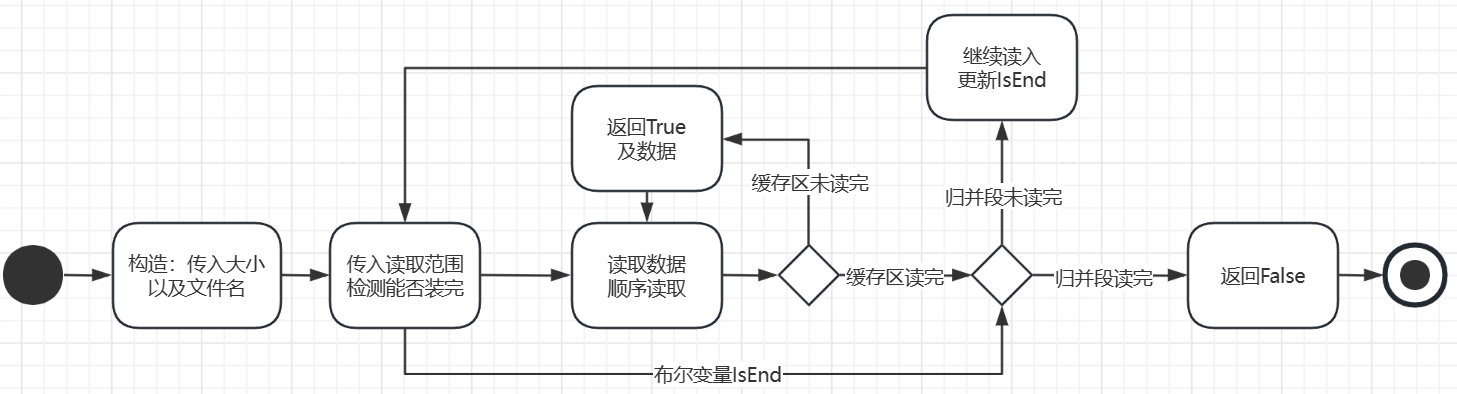
**归并运算模块：**即MergeSort类。主要包含外部归并算法，并为操作模块提供打印接口，将运算结果通过屏幕进行图形化输出，可选输出运算过程。在向缓存模块设置读取范围后按序从缓存模块获取数据。在收到访问越界的消息后开始下一轮归并。

**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

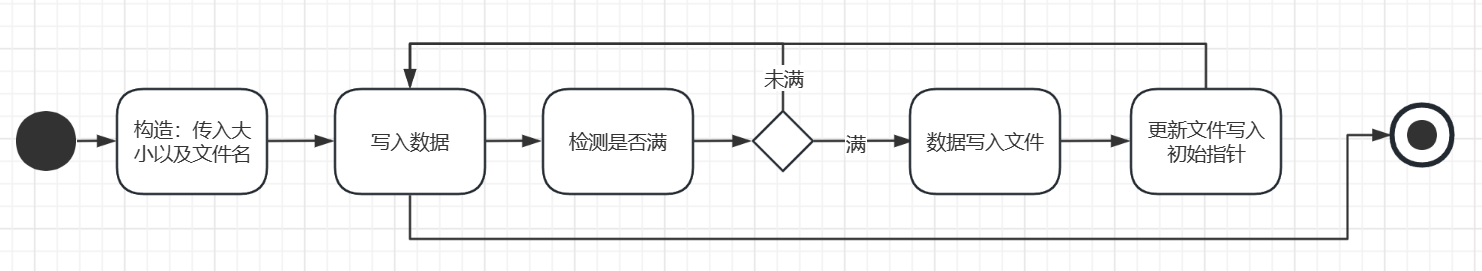
**文件模块：**矩阵以二进制形式(.bin)存储于文件中，按顺序以整型存储的排序内容。缓存模块通过文件指针的操作访问数据。提供临时文件清理代码。

### 缓存流程图：

读取缓冲区



写入缓冲区

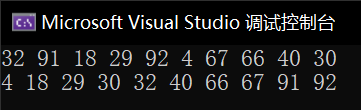


## 4．测试结果与分析

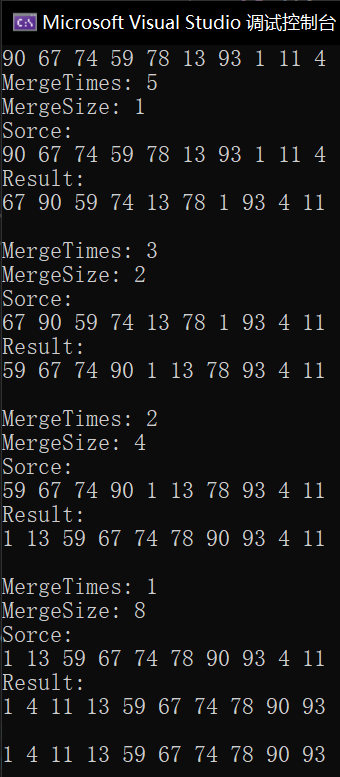
**测试数据选择或生成办法：**通过rand()随机算法生成指定长度与范围的测试数列。

**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

# 数据结构与算法专题实践3

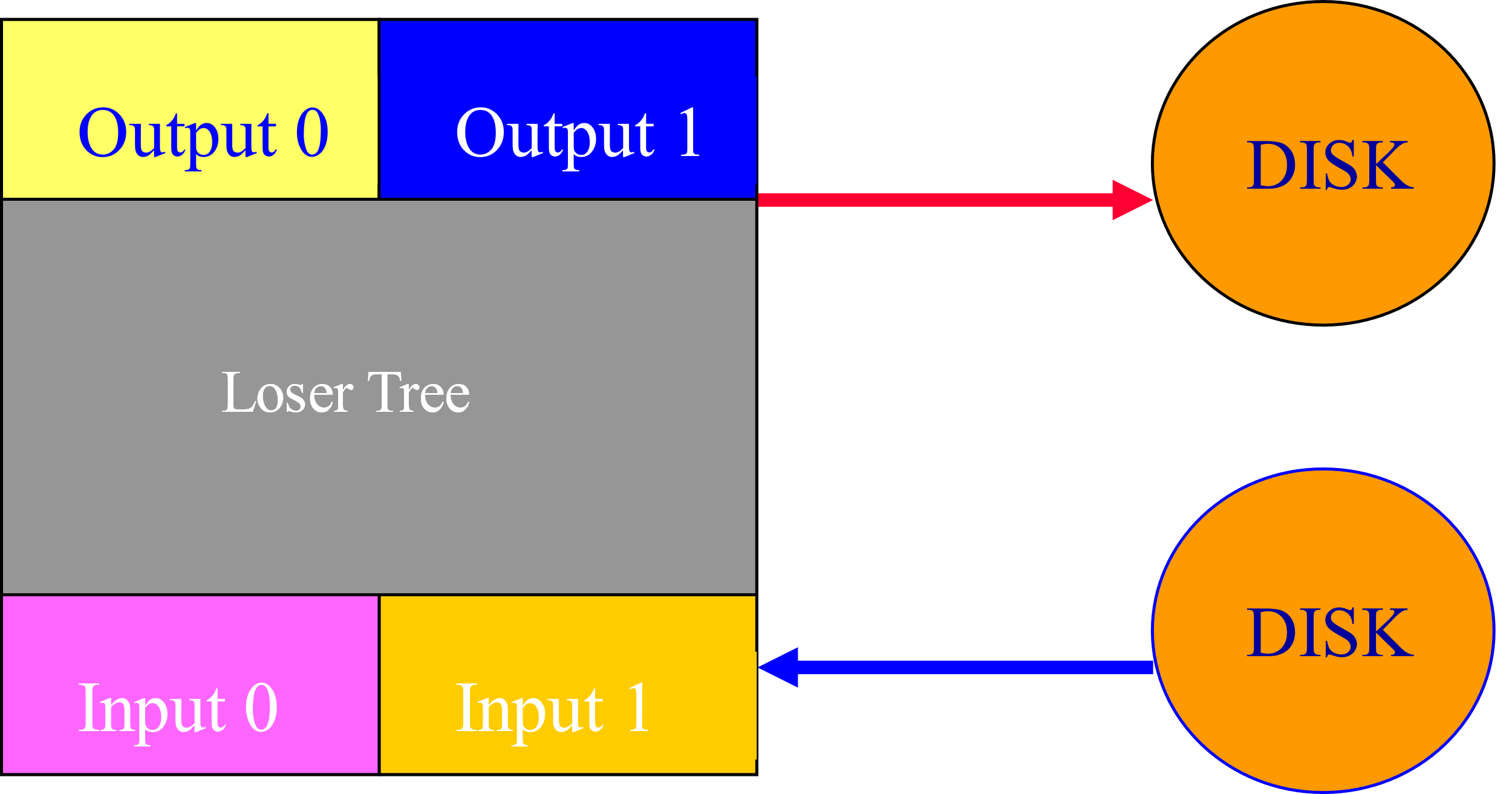
## 1．问题描述/需求分析

1. 数据文件组织
2. 基于并行策略的归并段生成优化算法
3. 最佳归并树
4. 基于最佳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和两个输出缓冲区。两个输出缓存区交替工作，一个写入缓冲区另一个从缓存区写回文件。两个输入缓存区在生成归并段时交替工作，在归并时协同工作。设置一个败者树用于生成归并段、一个最小堆用于构造最优归并树。



### 系统框架：

主体框架与实验二大致相同，归并运算模块实现了多线程。

### 描述各模块功能及其关系：

**操作模块：**包含main函数，向归并运算模块提供操作指令。

**归并运算模块：**包含败者树和最小堆，按照操作模块提供的指令进行生成归并段与归并运算。通过缓存模块提供的接口从中获取数据及消息。依据返回状态实现缓存的自动更换、使得文件读写和归并操作并行。

**缓存模块：**对实验二的缓存进行细节升级，总体无太大变换。为运算模块提供数据。支持预先设置缓存读取范围，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。为支撑多线程，写回可以设置开始位置。

**文件模块：**对实验二无变化，存储矩阵内容以及相关信息。归并时产生的子归并段向后叠加。

## 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，增加检测结果是否正确功能。

**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。先初始化败者树并依据败者树构建归并段。之后依据各个归并段的长度建立最小堆，向缓存模块设置各个归并段的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，将归并结果加入最小堆，并开始下一轮归并。通过设置缓存使用指示与封装将多线程缓存的使用封装为函数，与归并算法分离，便于检测与计算。

**缓存模块：**即Buff类。与实验二基本一致，增加一些属性接口，提供接口为运算模块提供数据。

**文件模块：**与实验二基本一致，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。

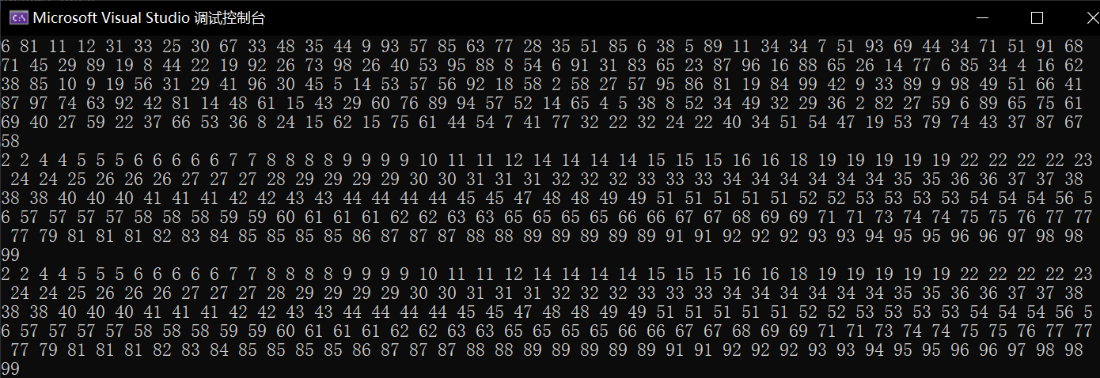
## 4．测试结果与分析

**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。

参数：缓存大小为7，测试数据大小为200，树高为5

**运行结果截图：**

**结果图：**



首行为原始数据，次行为正确排序答案，末行为我的排序结果

**性能分析：**

相较于实验二，本程序大大减少了归并次数，同时使用了最优归并树。多线程读写的实现使得性能进一步提高。

## 5．实验总结

本次实验改进了上一次的外部归并算法，使用了最优归并树与多线程技术，大大提高了效率。在本实验中，拥有两个输入缓冲区以及两个输出缓冲区，通过并行算法交替使用缩短文件读写所用的时间 。同时使用败者树跳过实验二中较小的归并，实现加速。

在实验的过程中遇到了以下问题：

问题：如何实现多线程的封装？

方案：

设置读取标记，表示当前可用的缓存号。通过一各返回缓存的引用的函数代替缓存，通过在写回等操作中封装修改标记、设立线程的代码实现通过一个函数自动操作多线程缓存区。

## 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

# 数据结构与算法专题实践4

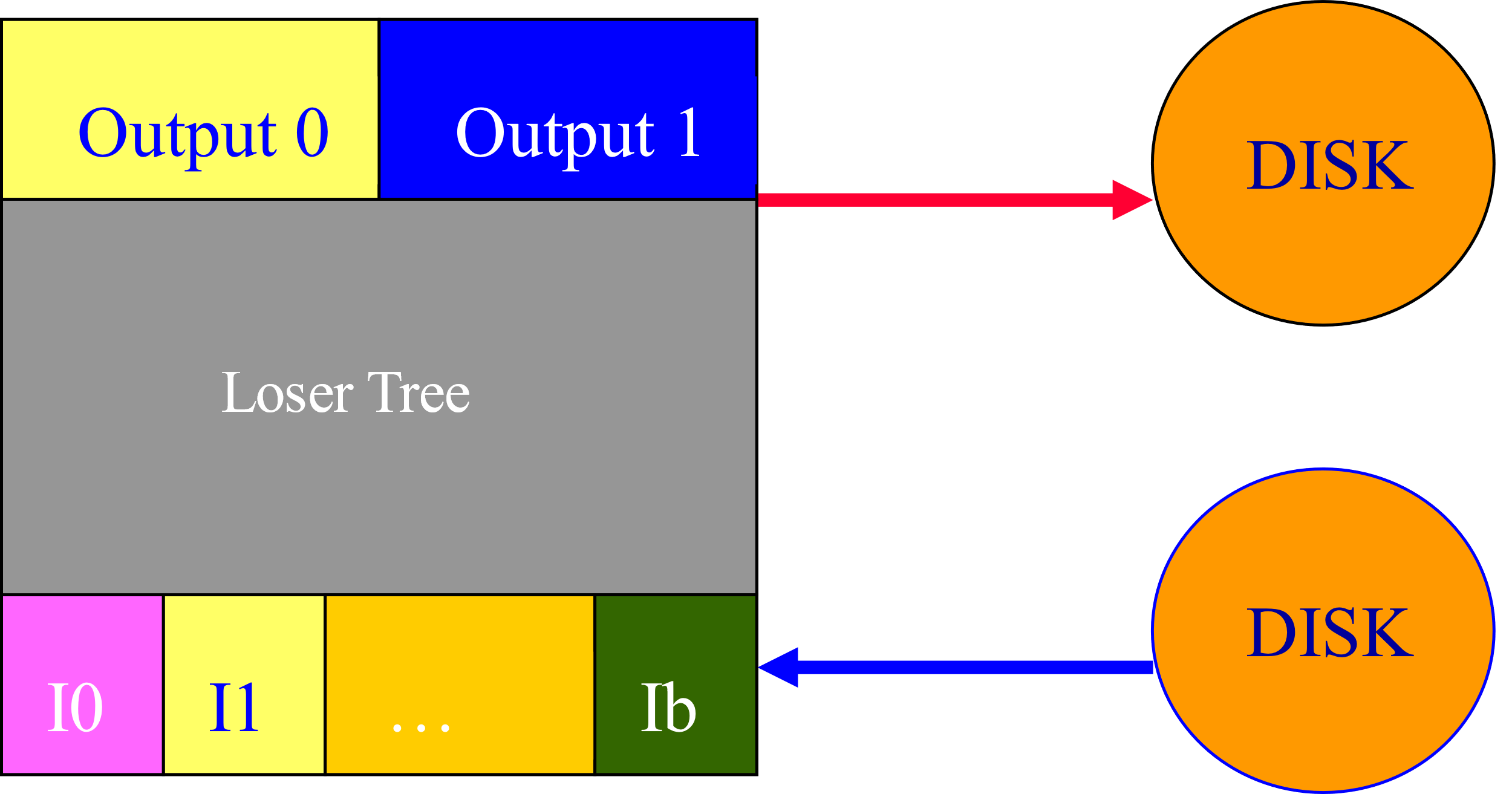
## 1．问题描述/需求分析

1. K路归并树
2. 归并缓冲区(2k + 2)管理策略
3. 基于并行策略的k路归并优化算法

## 2．系统结构/算法思想

### 基本思路：

基本思路为：主体与实验三相同。不同点在于本实验设置了2k个读入缓冲区以实现k路归并操作。通过创建状态列表保存每一个输入缓存区的状态，通过在每一次归并时动态分配读入缓冲区以实现其最大利用。



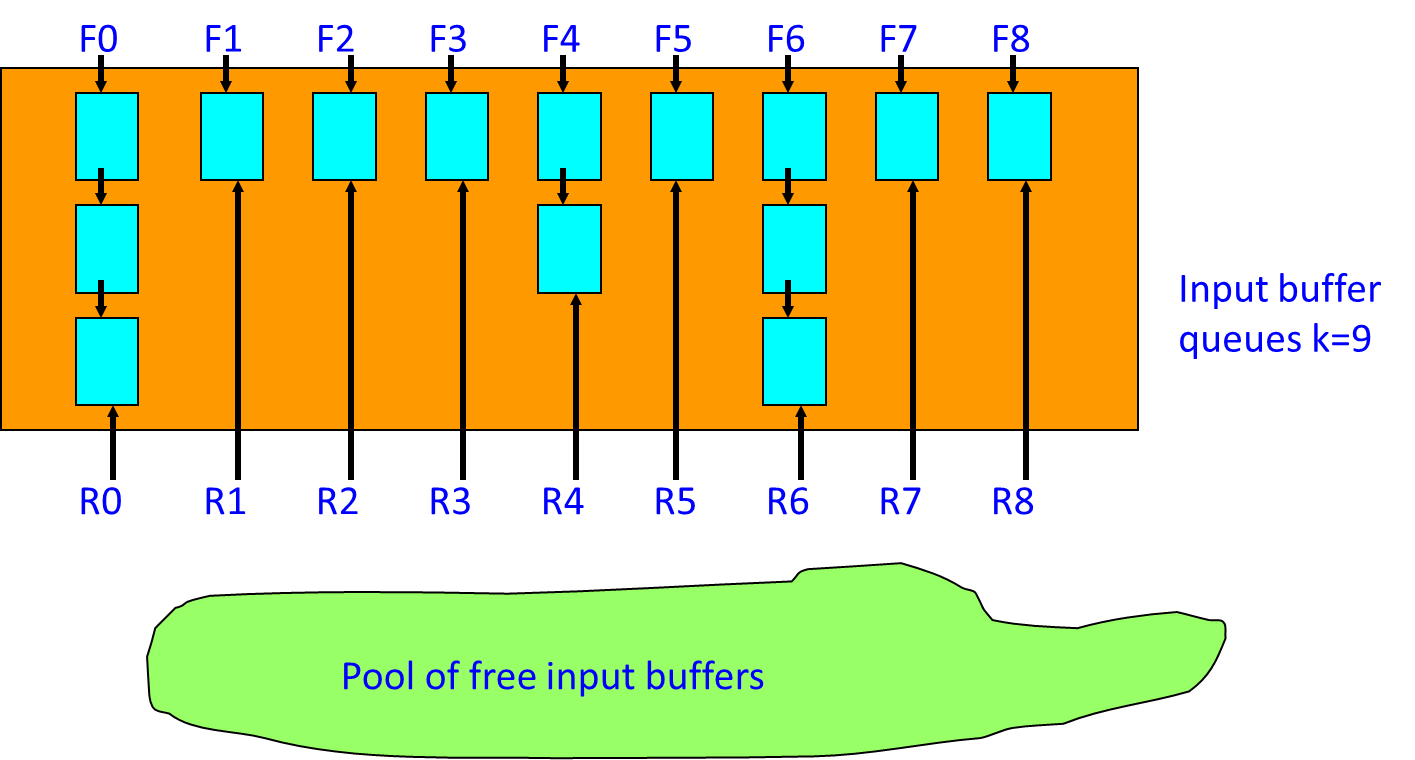
### 系统框架：

主体框架与实验三大致相同，归并运算模块的归并区域由二路归并更改为k路归并。

### 描述各模块功能及其关系：

**操作模块：**与实验三相同。

**归并运算模块：**与实验三大致相同，K路归并采用败者树进行。建立长度为2k的缓冲区数组、缓冲区状态数组、缓冲区分配数组。每次归并时选取未被装入缓存区的最长归并段分配空缓冲区。某个缓冲区读完后重置并回归空闲池



**缓存模块：**与实验三相同。

**文件模块：**与实验三相同。

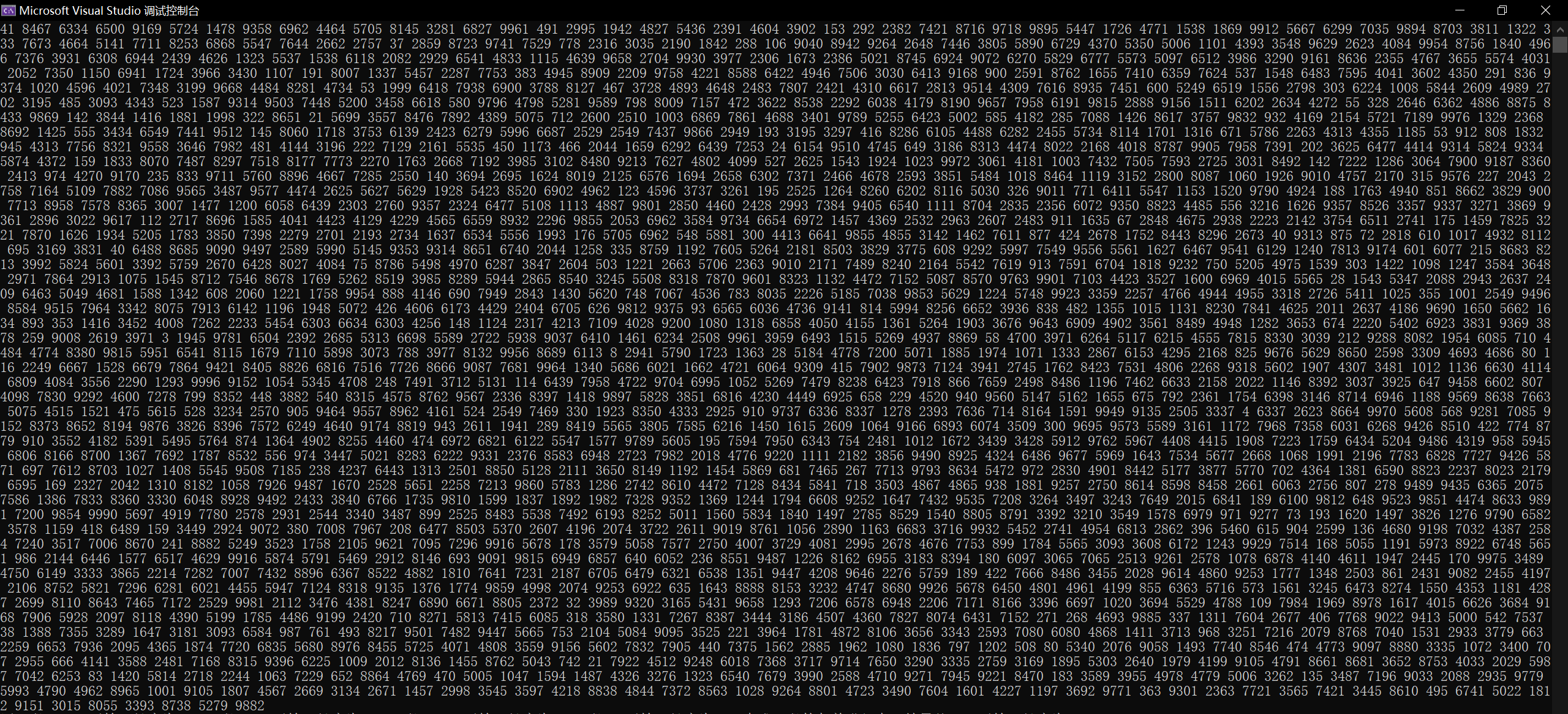
## 4．测试结果与分析

**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。

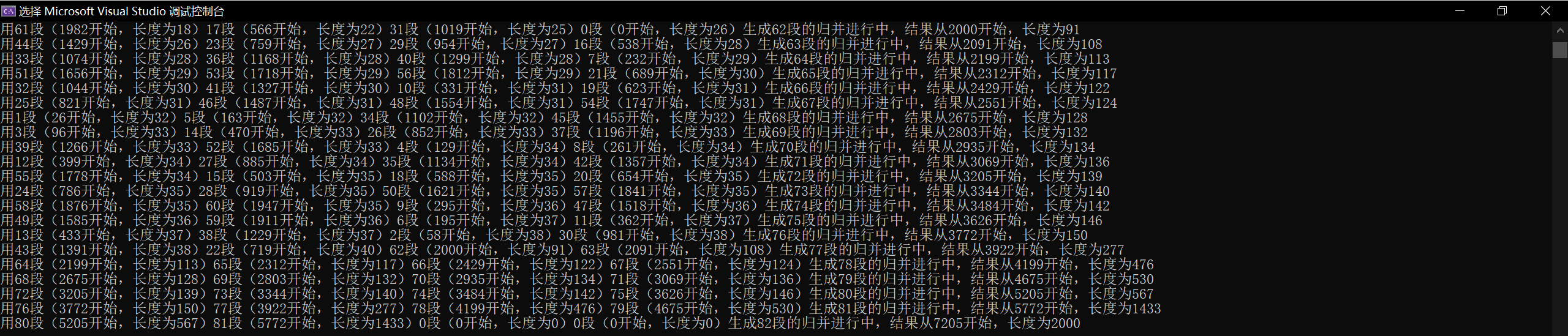
参数：缓存大小为3，测试数据大小为2000，树高为5，采用4路归并

**运行结果截图：**

**结果图：**



原始数据



归并过程（0-1999为初始归并段）



排序结果

**性能分析：**

K路归并带来的性能影响相较于上一个实验的提升不是线性的。随着k的增大，性能提升先增大后因归并负担减小。

## 5．实验总结

本次实验改进了上一次的外部归并算法，使用了k路归并，采用2k+2缓冲区管理算法，进一步提高了效率。在本实验中，拥有2k个输入缓冲区以及两个输出缓冲区，通过并行算法交替使用缩短文件读写所用的时间。多线程问题要注意同步，否则会产生随机性错误。

## 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

Def.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