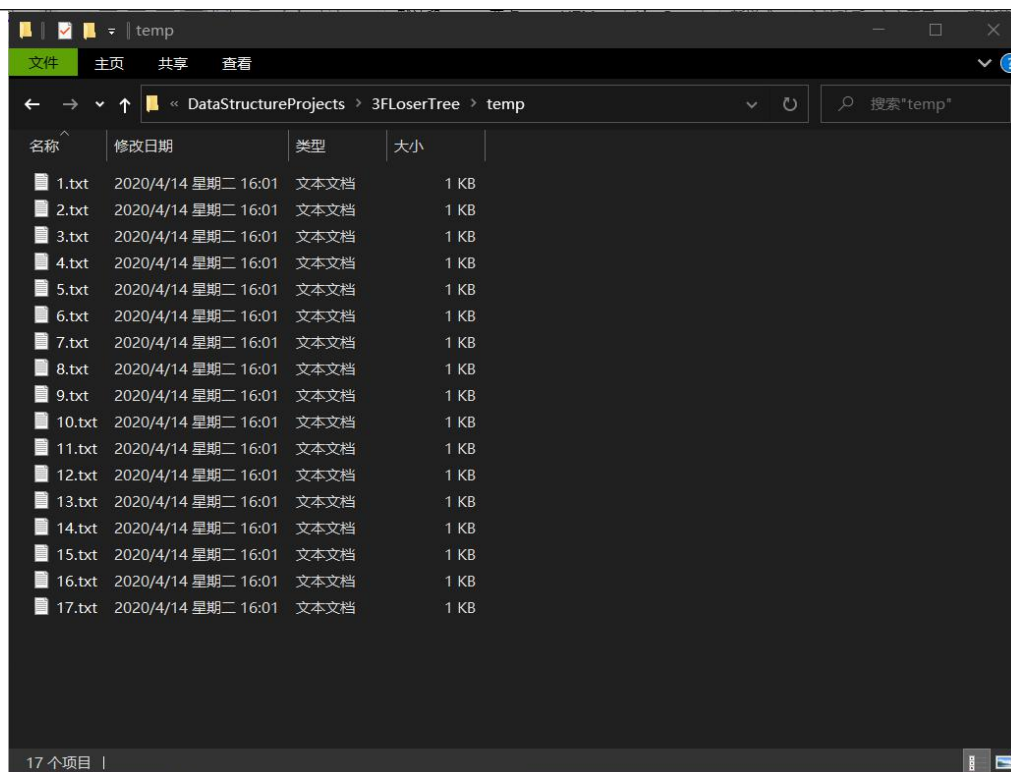


# 山东大学计算机科学与技术学院

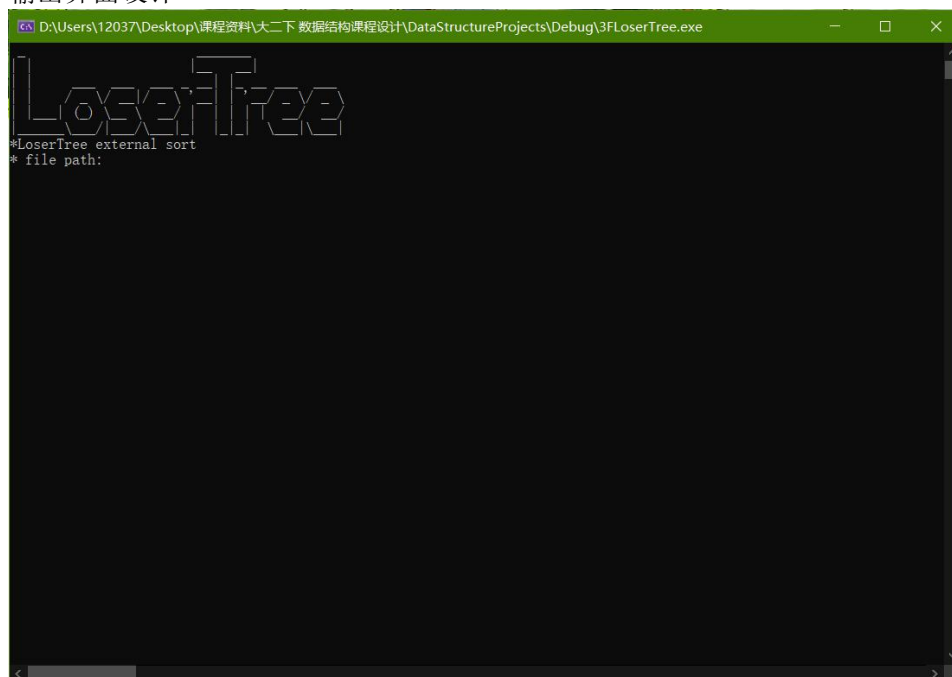
## 数据结构与算法课程设计报告

学号：201700140056	姓名：李港	班级：17.4
上机学时：4	日期：2020.04.11	
课程设计题目：外排序		
软件环境：VS2019		
报告内容： 一、需求描述 1.1 问题描述 应用竞赛树（输者树）结构模拟实现外排序。  1.2 基本要求 1. 设计实现最小输者树结构 ADT，ADT 中应包括初始化、返回赢者、重构等基本操作。 2. 设计实现外排序，外部排序中的生成最初归并串以及 K 路归并都应用最小输者树结构实现； 3. 随机创建一个较长的文件；设置归并路数以及缓冲区的大小；获得外排序的访问磁盘的次数并进行分析。可采用小文件来模拟磁盘块。  1.3 输入说明 1. 输入设计：采用文件输入的形式，文件内仅包含无序的数组。  2. 输入数据样例：采用文件输出的形式，文件内包含排好序的数组。另外 temp 文件夹下包含分割后的小文件。  <pre>1 9790 7143 5220 831 1781 4321 8566 4393 7452 973 5745 1533 4042 3333 6115 8698 147 3100 2965 9565 1087 3517 9816 8343 154 7167 227 2512 6813 5708 9755 7435 1421 818 6133 9068 3347 2297 6296 2280 1788 6886 2951 8252 2770 975 4381 6145 5914 9696 5737 8654 1871 1726 7021 1742 5071 1674 5105 4148 8676 456 8861 4188 8262 6145 1767 3476 3696 8022 6794 5515 9136 3830 6872 8168 6338 3745 7248 910 5295 4909 8131 2463 3282 1561 2228 8031 8939 9873 47 6650 1384 2801 6492 1432 7137 7005 7603 1343 2546 529 3921 3466 6814 2500 3612 1162 5442 3282 2452 3514 1528 8907 7835 2279 2414 8230 2470 8767 1878 6826 820 905 6623 4174 8238 1654 468 1550 1840 5190 3299 3969 3416 8814 512 2330 300 2968 7227 9116 2368 3948 6223 593 1864 1854 45 4828 4955 4984 9609 6334 8681 265 4226 9430 8798 654 3512 2204 9239 1696 9641 4848 1188 2014 5981 1486 623 4556 2515 357 9549 1052 1051 2072 5490 8029 8933 4037 4298 2097 5146 3700 1445 547 4314 242 274 7544 9893 4591 5583 7237 638 7556 7792 5138 9037 2409 3758 8286 2420 6774 5650 6979 3284 8025 9475 1179 2752 6423 6987 7762 5410 2966 5185 6190 5149 8938 4947 7517 532 5838 7681 6814 105 3302 1186 2499 6631 7971 9905 6212 660 487 8040 4424 3026 3561 1136 7536 5461 4907 6920 8484 4618 4138 9764 4098 2517 7430 1432 8004 1947 4579 7829 6194 3842 2254 1202 4487 5781 852 3055 7320 1245 5968 422 6627 5980 2531 6052 3028 1932 297 5399 8542 7048 4265 7767 8104 345 5518 4057 821 6259 2644 4882 4233 1329 3092 5081 2041 798 6932 8225 290 4629 8344 6677 140 5148 5822 3589 6048 3485 5753 1136 1064 8117 6363 1106 3431 7312 8121 1792 25 5653 1674 5953 555 8493 688 3007 4942 2666 6362 8686 5192 4950 9561 8093 4828 4666 4753 4139 3611 6192 2322 1948 1419 538 9416 2716 3303 4084 2512 9938 5751 7911 6509 4713 641 3248 9497 8731 5216 1361 2495 1503 4564 8327 2217 7829 3866 5964 453 3583 7966 9805 7870 8594 9358 2622 6680 1021 9569 2236 8315 2282 4215 3651 3492 4787 2434 2722 1864 1718 6502 764 2955 9227 5101 96 4033 6454 5439 6075 7250 7129 2752 6060 915 6476 1337 8280 146 8239 1003 5328 8578 4478 3874 6320 4937 1894 807 3257 2486 6072 4680 4817</pre>		



#### 1.4 输出说明

##### 输出界面设计



##### 输出数据样例

$$\begin{array}{l} \quad \quad \quad \diagup -9:64 \\ \quad \quad \quad | \\ \quad \quad \quad \diagdown -7:36 \\ \quad \quad \quad | \quad \quad \quad \diagdown -2:1 \\ \quad \quad \quad | \\ \diagup -1:0 \\ | \\ | \quad \quad \quad \diagup -4:9 \\ | \quad \quad \quad | \\ | \quad \quad \quad \diagdown -3:4 \\ | \\ 10:81 \\ | \\ \quad \quad \quad \diagup -6:25 \\ \quad \quad \quad | \\ \diagdown -5:16 \\ | \\ \quad \quad \quad \diagdown -8:49 \end{array}$$

## 2.4 数据及数据类(型)定义

```
private:
    ...int* tree;.....//输者树内部节点
    ...int* winners;.....//存放赢者

    ...T* _players;.....//输者树外部节点
    ...int _numOfPlayer;//选手的个数

    ...int lowExt;.....//2*(n-s),最底层外部节点的个数,s=2^(log2(n-1))
    ...//s为最底层最左端的内部节点的位置
    ...int offset;.....//2*s-1
```

## 2.5. 算法设计及分析

### 输者树的重排算法

//更新结构, thePlayer 指向的元素已经被替换

```
void replay (int thePlayer) {
    int n = _numOfPlayer;
    //判断 player 范围
    if (thePlayer <= 0 || thePlayer > n) {
        cout << "_players index is illegal" << endl;
        return;
    }

    //获取顺串新元素在树中的新位置
    int fatherPoint;//左右选手的父亲节点
    int left, right;//左右选手
    if (thePlayer <= lowExt) { //如果变化的节点在最底层
        fatherPoint = (thePlayer + offset) / 2; //获取父节点
        left = 2 * fatherPoint - offset; // unify to the left _players
        right = left + 1;
    }
    else { //the _players is on the last but one
        fatherPoint = (thePlayer - lowExt + n - 1) / 2;
        //theplayer 的左兄弟是最后一个内部节点
        if (2 * fatherPoint == n - 1) {
            left = winners[2 * fatherPoint];
            right = thePlayer;
        }
        else {
            left = 2 * fatherPoint - (n - 1) + lowExt;
            right = left + 1;
        }
    }
}

//重新比赛
if (thePlayer == tree[0]) { //重新比赛的选手在之前胜者的位置
    for (; fatherPoint >= 1; fatherPoint /= 2) { //上次比赛的输者已经记录在 tree[] 中
        int loserTemp = _whoLose (tree[fatherPoint], thePlayer); //只需跟败者判断
        winners[fatherPoint] = _whoWin (tree[fatherPoint], thePlayer);
        tree[fatherPoint] = loserTemp; //父节点放置新的败者
        thePlayer = winners[fatherPoint]; //赢者继续比赛
    }
}
```

```

}else { //否则，无法判断新节点与老输者的关系，从该节点到根的路径需重新比赛
    tree[fatherPoint] = _whoLose (left, right);
    winners[fatherPoint] = _whoWin (left, right);

    //向上一级
    if (fatherPoint == n - 1 && n % 2 == 1) {
        fatherPoint /= 2;
        tree[fatherPoint] = _whoLose (winners[n - 1], lowExt + 1);
        winners[fatherPoint] = _whoWin (winners[n - 1], lowExt + 1);
    }

    fatherPoint /= 2;
    for (; fatherPoint >= 1; fatherPoint /= 2) {
        tree[fatherPoint] = _whoLose (winners[2 * fatherPoint], winners[2 * fatherPoint + 1]);
        winners[fatherPoint] = _whoWin (winners[2 * fatherPoint], winners[2 * fatherPoint + 1]);
    }
}
//保存最终胜者
tree[0] = winners[1];
}

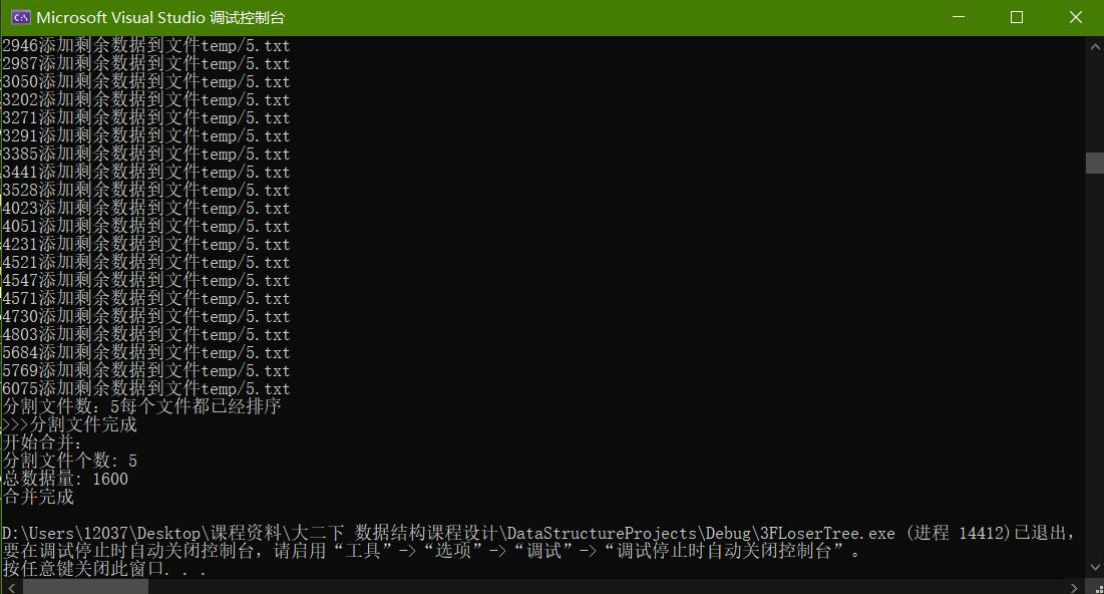
```

外排序思路：

1. 将乱序输入采用输者树分块存储为多个排好序的小文件。
2. 将所有小文件采用输者树进行多路归并。

### 三、测试

#### 1. 排序过程输出



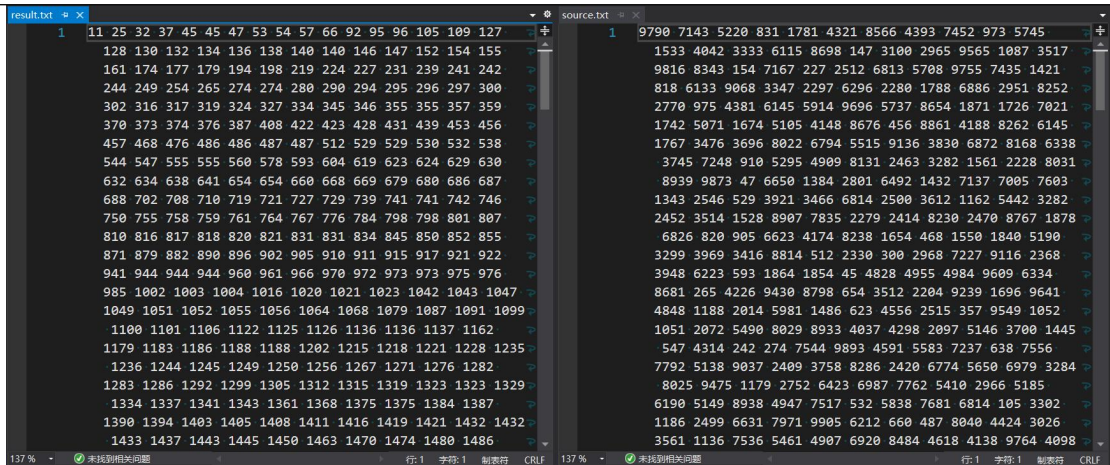
```

Microsoft Visual Studio 调试控制台
2946添加剩余数据到文件temp/5.txt
2987添加剩余数据到文件temp/5.txt
3050添加剩余数据到文件temp/5.txt
3202添加剩余数据到文件temp/5.txt
3271添加剩余数据到文件temp/5.txt
3291添加剩余数据到文件temp/5.txt
3385添加剩余数据到文件temp/5.txt
3441添加剩余数据到文件temp/5.txt
3528添加剩余数据到文件temp/5.txt
4023添加剩余数据到文件temp/5.txt
4051添加剩余数据到文件temp/5.txt
4231添加剩余数据到文件temp/5.txt
4521添加剩余数据到文件temp/5.txt
4547添加剩余数据到文件temp/5.txt
4571添加剩余数据到文件temp/5.txt
4730添加剩余数据到文件temp/5.txt
4803添加剩余数据到文件temp/5.txt
5684添加剩余数据到文件temp/5.txt
5769添加剩余数据到文件temp/5.txt
6075添加剩余数据到文件temp/5.txt
分割文件数: 5每个文件都已经排序
>>>分割文件完成
开始合并:
分割文件个数: 5
总数据量: 1600
合并完成
D:\Users\12037\Desktop\课程资料\大二下 数据结构课程设计\DataStructureProjects\Debug\3FloserTree.exe (进程 14412) 已退出,
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .

```

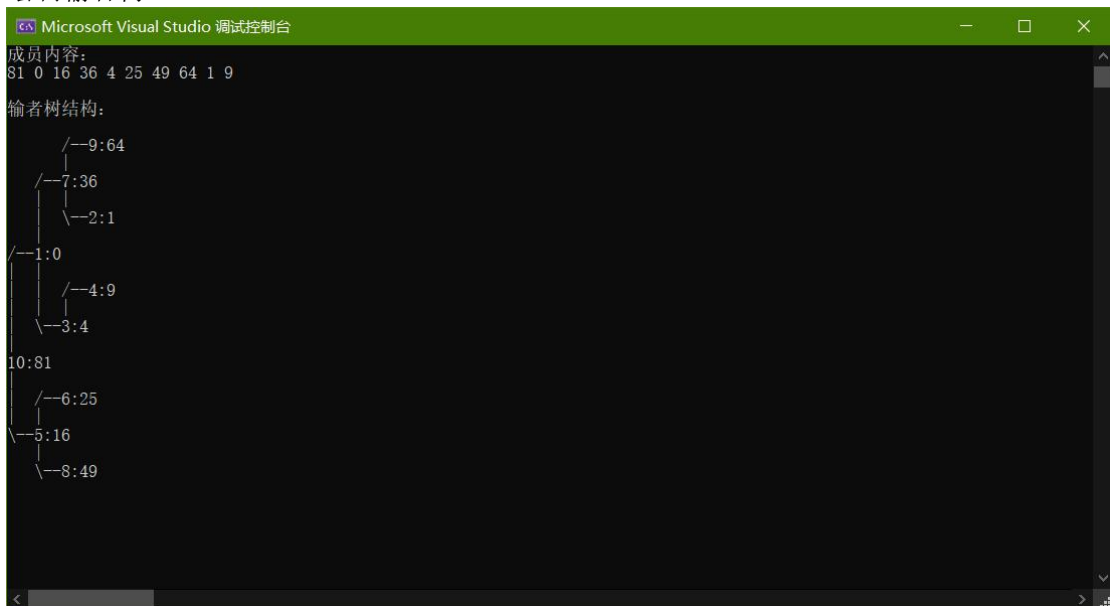
#### 2. 排序前后数据对比





可见，原始乱序数据成为了有序数据。

### 3. 绘制输者树



## 四、分析与探讨

1. 大批量数据算法是面试常考的知识点，外排序只是其中一类。但这些算法都有统一的思想“分治”。
2. 输者树可以降低数据变动对数据结构的影响

## 五、附录：实现源代码

```
#include<iostream>
#include<vector>
#include<map>
#pragma warning(disable:4996)

using namespace std;
template<class T>
class LoserTree {
public:
    LoserTree () { tree = nullptr; };
    LoserTree (T* thePlayers, int n) { initTree (thePlayers, n); } //创建输者树
    ~LoserTree () { delete[] tree; delete[] winners; } //释放资源

    void initTree (T* thePlayers, int n) {
        auto play = [this](int p, int left, int right) ->void {
            //根据 player[l], players[r]的值，设置 tree[p]的值，即在左右子节点 l, r 之间举办比赛
            //之后如果 p 是右孩子节点，则继续举行上一层的比赛

            tree[p] = _whoLose (left, right);
            winners[p] = _whoWin (left, right);
            //内节点 p 是右孩子且非根
        };
    }
};
```

```

        while (p % 2 == 1 && p > 1) {
            tree[p / 2] = _whoLose (winners[p - 1], winners[p]);
            winners[p / 2] = _whoWin (winners[p - 1], winners[p]);
            p /= 2;
        }
    };
    tree = nullptr, winners = nullptr;

    if (n < 2) { return; }
    _players = thePlayers;
    _numOfPlayer = n;

    tree = new int[n];
    winners = new int[n];

    int i, s;

    //计算 s, lowExt, offset
    for (s = 1; 2 * s <= n - 1; s *= 2);
    lowExt = 2 * (n - s);
    offset = 2 * s - 1;

    //对所有底层外部节点进行 play()
    for (i = 2; i <= lowExt; i += 2)
        _play ((i + offset) / 2, i - 1, i);

    //n 为奇数时, 会出现一个内节点其孩子分别是内节点和外节点
    //先对这个内节点和外节点进行 play()
    if (n % 2 == 1) {
        _play (n / 2, winners[n - 1], lowExt + 1);
        i = lowExt + 3;
    }
    else {
        i = lowExt + 2;
    }

    //再对剩余的外部节点 play()
    for (; i <= n; i += 2)
        _play ((i - lowExt + n - 1) / 2, i - 1, i);

    //将最终的赢家记录在 tree[0]
    tree[0] = winners[i];
}

//更新结构, thePlayer 指向的元素已经被替换
void replay (int thePlayer) {
    int n = _numOfPlayer;
    //判断 player 范围
    if (thePlayer <= 0 || thePlayer > n) {
        cout << "_players index is illegal" << endl;
        return;
    }

    //获取顺序新元素在树中的新位置
    int fatherPoint; //左右选手的父亲节点
    int left, right; //左右选手
    if (thePlayer <= lowExt) { //如果变化的节点在最底层
        fatherPoint = (thePlayer + offset) / 2; //获取父节点
        left = 2 * fatherPoint - offset; // unify to the left _players
        right = left + 1;
    }
    else { //the _players is on the last but one
        fatherPoint = (thePlayer - lowExt + n - 1) / 2;
        //theplayer 的左兄弟是最后一个内部节点
        if (2 * fatherPoint == n - 1) {
            left = winners[2 * fatherPoint];
            right = thePlayer;
        }
        else {
            left = 2 * fatherPoint - (n - 1) + lowExt;
            right = left + 1;
        }
    }
}

//重新比赛
if (thePlayer == tree[0]) { //重新比赛的选手在之前胜者的位置
    for (; fatherPoint >= 1; fatherPoint /= 2) { //上次比赛的输者已经记录在 tree[] 中
        int loserTemp = _whoLose (tree[fatherPoint], thePlayer); //只需跟败者判断
        winners[fatherPoint] = _whoWin (tree[fatherPoint], thePlayer);
        tree[fatherPoint] = loserTemp; //父节点放置新的败者
        thePlayer = winners[fatherPoint]; //赢家继续比赛
    }
}
else { //否则, 无法判断新节点与老输者的关系, 从该节点到根的路径需重新比赛
    tree[fatherPoint] = _whoLose (left, right);
    winners[fatherPoint] = _whoWin (left, right);

    //向上一级
    if (fatherPoint == n - 1 && n % 2 == 1) {
        fatherPoint /= 2;
        tree[fatherPoint] = _whoLose (winners[n - 1], lowExt + 1);
        winners[fatherPoint] = _whoWin (winners[n - 1], lowExt + 1);
    }

    fatherPoint /= 2;
    for (; fatherPoint >= 1; fatherPoint /= 2) {

```

```

        tree[fatherPoint] = _whoLose (winners[2 * fatherPoint], winners[2 * fatherPoint + 1]);
        winners[fatherPoint] = _whoWin (winners[2 * fatherPoint], winners[2 * fatherPoint + 1]);
    }
    //保存最终胜者
    tree[0] = winners[1];
}
void output () const {
    for (int i = 0; i < _numOfPlayer; i++) {
        cout << _players[tree[i]] << endl;
    }
}
int* theTree () { return tree; } //返回输者树
int winner () { return winners[1]; }

private:
int* tree; //输者树内部节点
int* winners; //存放赢者

T* _players; //输者树外部节点
int _numOfPlayer; //选手的个数

int lowExt; //2*(n-s),最底层外部节点的个数,s=2^(log2(n-1))
int offset; //s为最底层最左端的内部节点的位置
//2*s-1

int _whoWin (int x, int y) { return _players[x] <= _players[y] ? x : y; } //返回赢者
int _whoLose (int x, int y) { return _players[x] <= _players[y] ? y : x; } //返回输者
};

#include "loserTree.h"
#include <limits.h>
#include <fstream>
#include <cstdio>
using namespace std;

//输者树的元素
struct node {
    int value; //元素值
    int index; //顺串号

    operator int () { return value; } //将类转化为 int 类型
    bool operator <= (node b) {
        if (index < b.index)
            return true;
        else if (index == b.index)
            return value <= b.value;
        else
            return false;
    }
};

class externalsort {
public:
    string filepath;
    externalsort (int buf_size, string filepath) : _bufferLength(buf_size), _filepath(filepath) {}
    ~externalsort () {}

    //进行文件分割
    void split () {
        _clearfile (300);

        cout << ">>>分割文件开始" << endl;

        //打开文件
        ifstream in ( _filepath);
        if (in.fail ()) {
            cout << "no such file!";
            exit (0);
        }

        //新建 p+1 个选手
        node* players = new node[_bufferLength + 1];
        //

        //读取文件, 初始化所有选手
        int value;
        int q = 0;
        while (q < _bufferLength && in >> value) {
            _countReadDiskOnce ();
            q++;

            players[q].value = value;
            players[q].index = 1;
            cout << "排序输者树: " << players[q].value << endl;
        }
        q++;
        if (_bufferLength > q)
            _bufferLength = q;

        //使用这些选手初始化输者树
        LoserTree<node> _spliter;
        _spliter.initTree (players, _bufferLength);
    }
};

```



```

    _fileNum = 1;
    //分割文件
    cout << "开始分割文件: " << endl;
    totalSize = _bufferLength;
    int temp_in_num;
    //将剩余数据添加到各小文件中
    while (in >> temp_in_num) {
        _totalSize++;
        _countReadDiskOnce ();

        node newNode;

        //输入的数字比原有的小
        if (temp_in_num < players[_splitter.winner ()].value) {
            newNode.index = players[_splitter.winner ()].index + 1;

            if (newNode.index > _fileNum)
                _fileNum = newNode.index;
        } else
            newNode.index = players[_splitter.winner ()].index;

        newNode.value = temp_in_num;

        char temp_file_name[50];
        sprintf (temp_file_name, "winners/%d.txt", players[_splitter.winner ()].index);

        cout << players[_splitter.winner ()].value << "添加到文件 " << temp_file_name << endl;

        ofstream out;
        out.open (temp_file_name, ios::app); //追加模式写
        out << players[_splitter.winner ()].value << " ";
        out.close ();
        _countReadDiskOnce ();

        int i = _splitter.winner ();
        players[i] = newNode;
        _splitter.replay (i);
    }

    //将所有数据清空
    for (int i = 0; i < _bufferLength; i++) {
        _countReadDiskOnce ();

        char temp_file_name[50];
        sprintf (temp_file_name, "winners/%d.txt", players[_splitter.winner ()].index);

        cout << players[_splitter.winner ()].value << "添加剩余数据到文件" << temp_file_name << endl;

        ofstream out_file_stream;
        out_file_stream.open (temp_file_name, ios::app); //追加模式写
        out_file_stream << players[_splitter.winner ()].value << " ";
        out_file_stream.close ();

        //替换元素
        players[_splitter.winner ()] = { INT_MAX ,INT_MAX };

        //重排
        _splitter.replay (_splitter.winner ()); //重排
    }
    delete[] players;
    cout << "分割文件数: " << _fileNum << "每个文件都已经排序" << endl;

    cout << ">>>分割文件完成" << endl;
}

void merges ();
void visitstime () { cout << "访问磁盘次数: " << _readDiskCount << endl; }
private:
    //int _bufferLength; //初始化顺串时最小竞赛树的规模, 初始顺串的平均长度为 2p
    int _fileNum; //输入顺串数
    int _totalSize; //待排序元素总数
    int _readDiskCount; //访问磁盘次数
    int _bufferLength;
    static int _bufferUsed;

    //清空所有文件
    void _clearfile (int n) {
        cout << ">>>正在清空临时文件: " << endl;

        char a[100];
        for (int i = 1; i <= n; i++) {
            sprintf (a, "winners/%d.txt", i);
            remove (a);
        }

        cout << ">>>临时文件清空完成: " << endl;
    }
}

```

```

//模拟读硬盘
void _countReadDiskOnce () {
    _bufferUsed++;
    if (_bufferUsed > _bufferLength) {
        _readDiskCount++;
        _bufferUsed = 0;
    }
}
};
int externalsort::_bufferUsed = 0;

//排序过程
void externalsort::merges () {
    //分割
    split ();

    //用于文件归并排序
    LoserTree<int> _final_sorter;
    cout << "开始合并: " << endl;
    cout << "分割文件个数: " << _fileNum << endl;
    cout << "总数据量: " << _totalSize << endl;

    //打开 k 个文件输入流
    char a[50];
    ifstream* in_files = new ifstream[_fileNum + 1];
    for (int i = 1; i <= _fileNum; i++) {
        sprintf (a, "winners/%d.txt", i);
        in_files[i].open (a);
        if (!in_files[i].is_open ()) {
            cout << "open winners file error" << endl;
            return;
        }
    }

    //读入所有文件的头一个数据
    int* da = new int[_fileNum + 1]; //头数据数组
    for (int i = 1; i <= _fileNum; i++) {
        in_files[i] >> da[i]; //数值表示 key, 下标表示顺串号
    }

    //初始化合并输着树
    _final_sorter.initTree (da, _fileNum);

    //遍历各文件中所有数据
    {
        ofstream out_file ("result.txt"); //出作用域自动关闭文件
        for (int i = 0; i < _totalSize; i++) {
            //每次从竞赛树中弹出冠军读取其顺串号, 再输出 key 值到文件
            int winner_file_id = _final_sorter.winner (); //竞赛树中冠军的下标, 顺串号
            out_file << da[winner_file_id] << " ";

            //根据顺串号从相应文件读取下一个数据, 若不存在则用 INT_MAX 代替, 替换冠军, 重排
            _countReadDiskOnce ();

            //获取下一个数字+顺串空判断
            int x;
            if (in_files[winner_file_id] >> x)
                da[winner_file_id] = x;
            else
                da[winner_file_id] = INT_MAX;

            //输者树重新比赛
            _final_sorter.replay (winner_file_id);
        }
    }
    delete[] in_files; //删除顺串输入流数组
    cout << "合并完成" << endl;
}

#pragma warning(disable:4996)
#include "externalSort.h"
#include <string>
#include <iostream>

int main(){
    cout << R"(
    _____
   |L|O|S|E|R|T|r|e|e|
   |_____|
);";
    cout << "LoserTree external sort"<<endl;
    cout << "file path:";
    string filepath;
    cin >> filepath;
    cout << "buffer size:";
    int size;
    cin >> size;

```

```
externalsort es(size,filepath);  
//进行归并  
es.merges();  
return 0;  
}
```