



哈爾濱工業大學

数 据 结 构 实 验 报 告

查 找 与 排 序 方 法

学	院	计算机科学与技术
姓	名	胡 博 涵
学	号	SZ170110113
专	业	计算机类
日	期	2018-06-14

目录

1	问题分析	1
2	详细设计	2
2.1	设计思想	2
2.2	存储结构及操作	2
2.3	程序整体流程	7
3	用户手册	15
4	总结	16
5	结果	17
6	感想与建议	24

1 问题分析

任务一 任务一要求对给出的整数进行重新组合，使之成为最大的整数，对于计算机而言，属于排序算法，只是需要重新定义排序规则。

任务二 任务二要求对 k 个有序列表进行合并，对于计算机而言，是有序列表的归并算法。

任务三 任务三要求对股票信息进行排序。首先需要选择合适的存储结构对股票信息进行存储。输入的数据为字符串形式，则需要对字符串进行解析，并将信息正确存储。在排序方面，有冒泡排序、选择排序、插入排序等效率不同的算法，应采取高效的排序算法对某关键字进行排序。在查找方面，有线性查找、索引查找、折半查找和基于索引树的查找等不同方式，对于不同的查找方式，需要选取不同的数据结构进行存储。例如，对于线性查找，需要采用线性存储结构；折半查找必须采取顺序存储结构；索引查找需要采取线性存储结构；索引树则需要线性和树状存储结构进行结合。

2 详细设计

2.1 设计思想

任务一 任务一给出了一组非负整数，要求重新排列顺序使之成为一个最大的整数。这实际上也是一种排序，只是排序规则需要重新进行定义。

对于这个排序，定义的规则是：如果两个数字 a, b 拼接而成的多位数字 $ab > ba$ ，（例如 34 和 5 组成的 534 和 345），则可以定义 $a > b$ 。而当整个数列达到在以上定义规则下的递减形式时，就可以保证拼接而成的整数是最大的。重新定义比较规则之后，选用任何一种排序算法，就可以得到答案。

任务二 任务二给定了若干个有序数组（其单调性一致），要求合并数组，并使之仍然有序。这里借鉴了归并排序的思想，合并线性表的操作可以看成归并排序算法的中间步骤，即开辟一块新的空间，对于待归并的有序线性表，比较其表首元素（或表尾元素），将较小（或较大）的一方插入新表，并从原始表中删除该数据项，并继续进行第二轮比较。

任务三 任务三给定了百万条数据，对于规模庞大的数据，需要进行快速的随机访问，故采用顺序存储结构进行存储。

排序 实验要求用某种 $O(n^2), O(n \log n)$ 的排序算法和希尔排序算法进行排序。这里 $O(n^2)$ 算法采用选择排序， $O(n \log n)$ 算法采用快速排序。

查找 查找有三种方式：

- **线性查找** 在原始数据上进行顺序查找。其时间复杂度为 $O(n^2)$ 。
- **索引查找** 对原始数据数据分为 m 块建立索引，进行索引查找。其时间复杂度为 $O(\log_2 m + n/m)$ 。
- **索引树查找** 将名称相同的股票看作同一股票，对于每一支股票，以交易日期为关键字建立平衡二叉排序树 (AVL) 进行查询。

2.2 存储结构及操作

任务一：最大数

存储结构 采用一维数组存储输入的数字和排序后的数字。

相关操作 以下是在 C 语言中定义的比较函数。

```

1 bool larger(int a, int b) {
2     char ca[100], cb[100];
3     sprintf(ca, "%d", a);
4     sprintf(cb, "%d", b);
5     int len_a = (int) strlen(ca);
6     int len_b = (int) strlen(cb);
7     // 分别计算 ab, ba 拼接的值
8     unsigned long long aplusb = (unsigned long long) pow(10, len_b) * a + b;
9     unsigned long long bplusa = (unsigned long long) pow(10, len_a) * b + a;
10    return aplusb > bplusa;
11 }
    
```

函数清单 以下为部分函数。

表 1: 函数清单

函数名	函数功能	操作对象	传入参数	返回值
bubble	冒泡排序单轮交换并判断区间是否有序	数据指针数组	单轮交换区间	bool (区间是否有序)
BubbleSort	冒泡排序	数据指针数组	待排序区间	无

在输出时，直接将数组元素顺序输出（其中不加空格）即可。

任务二：合并 K 个有序数组

存储结构 以下是列表类型的定义。

```

1 #define DEFAULT_CAPACITY 100
2 typedef struct _list { //定长顺序表
3     int _elem[DEFAULT_CAPACITY];
4     int size;
5 } List;

```

函数清单 以下为部分函数。

表 2: 任务二函数清单

函数名	函数功能	操作对象	传入参数	返回值
initList	创建顺序表对象	顺序表	无	顺序表对象指针
clrList	清空顺序表	顺序表	待清空的顺序表	无
insert	在顺序表尾部插入元素	顺序表	插入元素、顺序表	无
merge	归并两个有序线性表	顺序表	待归并的两个表	归并后的新表

任务三：股票信息

股票数据信息结构体的定义 以下是股票信息结构体的定义。

```

1 typedef struct _item {
2     char raw[LINE_MAX]; // 股票原始信息
3     char id[LINE_MAX]; // 股票名称
4     long date; // 交易日期
5     double start_price; // 开盘价
6     double highest; // 最高价
7     double lowest; // 最低价
8     double end_price; // 收盘价
9     double quantity; // 交易量
10    char otherInfo[LINE_MAX]; // 其他信息
11 } Item;
12 int numbers; // 设立全局变量存储股票数量
13 Item items[MAX_NUM]; // 数据域
14 Item* ptrs[MAX_NUM]; // 指针域 (存储排序后的变量指针)

```

函数清单 以下为部分函数。

表 3: 函数清单

函数名	函数功能	操作对象	传入参数	返回值
openFile	读取并解析文件信息	信息文件	文件名]	无
ShellSort	希尔排序	数据指针数组	待排序区间	无
selectPivot	选择主元位置	数据指针数组	待排序区间	主元位置
partition	快速排序 以主元为分界划分区间	数据指针数组	待排序区间 主元下标位置	主元新下标位置
QuickSort	快速排序（递归）	数据指针数组	待排序区间	无
linearSearch	线性查找	数据指针数组	查找区间、查找关键字	查找到的元素位置
getIndex	为索引查找创建索引	数据指针数组	无	无
AVLSearch	在 AVL 树查找项目	数据指针数组	对应关键字的索引树树根 查找关键字	查找到的元素位置

注 由于排序涉及大量的交换操作，结构体的交换将对排序的性能造成极大影响。故采用对指针进行交换的方法，即另设一指针数组，将其每一个单元初始化为对应单元编号的结构体的存储地址，在排序时，对指针进行交换，可提高排序效率。（实测：对结构体进行交换时，快速排序用时 1.53s；对指针进行交换时，快速排序用时 0.58s）

二叉平衡查找树（AVL） 下面给出二叉平衡查找树的具体实现。

存储结构 树节点的数据类型定义

```
1 typedef long T;
2 typedef struct __node {
3     T val; // 存储的关键字
4     Item *ptr; // 指向条目地址
5     int height; // 高度
6     struct __node *lChild; // 左子
7     struct __node *rChild; // 右子
8 } Node;
```

索引表中索引项的定义

```
1 typedef struct idx { // 索引表中的元素 区间为[lo,hi)
2     char id[LINE_MAX]; // 股票的名称
3     int lo; // 对应有顺序区间的初始端
4     int hi; // 对应有顺序区间的末尾
5     Node *root; // 对应AVL树的树根
6 } Index;
7 Index indexList[MAX_NUM]; // 索引表
8 int num_index; // 索引项的数量
```

操作接口 宏定义实现部分操作接口

```
1 #define HEIGHT(x) (( (x) == NULL ) ? 0 : (x) -> height)
2 #define MAX(a, b) ( (a) > (b) ? (a) : (b) )
3 #define BalFac(x) ( HEIGHT((x)->lChild) - HEIGHT((x)->rChild))
4 #define Balanced(x) ( (-2 < BalFac(x) ) && (BalFac(x) < 2) )
```

1. HEIGHT(x) 分别处理了 x 为空树节点和非空的情况，在程序中可以避免非法访问造成的错误
2. MAX(a,b) 定义了 a 和 b 之间的最大值
3. BalFac(x) 定义了某子树的平衡因子，借用了上面已经定义的宏 HEIGHT(x)
4. Balanced(x) 定义了一个判断式，判断树是否平衡。

使用宏定义的目的是提高程序的可读性，同时避免过多的函数调用造成过大开销。

函数清单 以下为核心函数清单：

表 4: AVL 树函数清单

函数名	函数功能	操作对象	传入参数	返回值
depth	递归求树的深度	二叉平衡树	根节点指针	树的深度
insert	在 AVL 树中，递归地插入节点	二叉平衡树	根节点指针 数据项指针	插入的节点
initNode	将数据项封装成一个叶节点 并串接左右子	数据项类型	数据项指针	创建后的节点指针
Rotate_LL	对左侧倾的子树进行局部的调整	二叉平衡树	根节点指针	调整后的根节点指针
Rotate_RR/LR/RL: 剩余的旋转函数此处省略，其中 LR/RL 函数调用之前实现的 LL/RR 函数				

2.3 程序整体流程

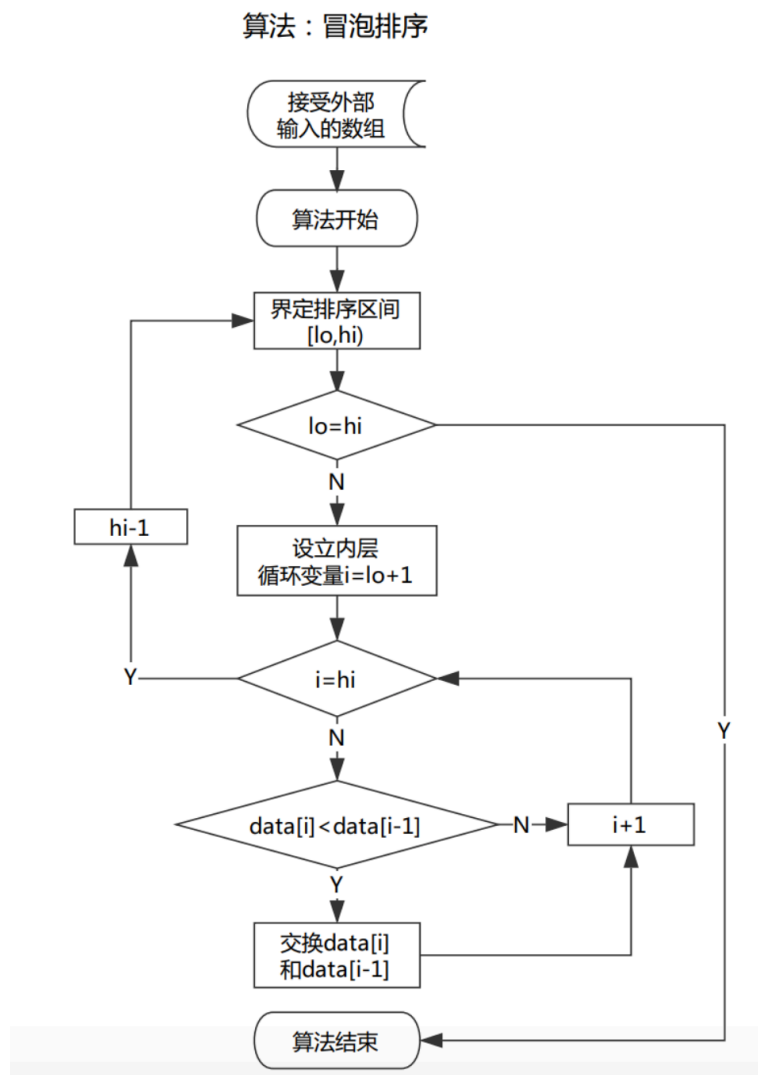


图 1: 任务 1: 冒泡排序

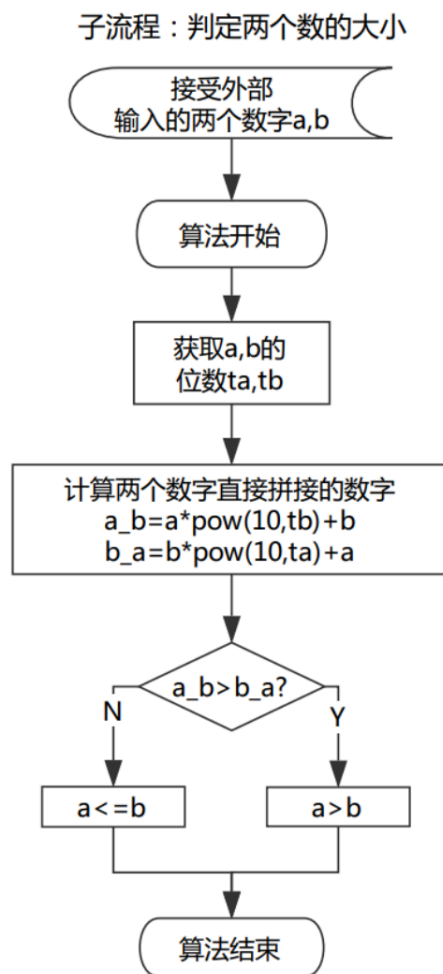


图 2: 任务 1: 比较两个数的顺序

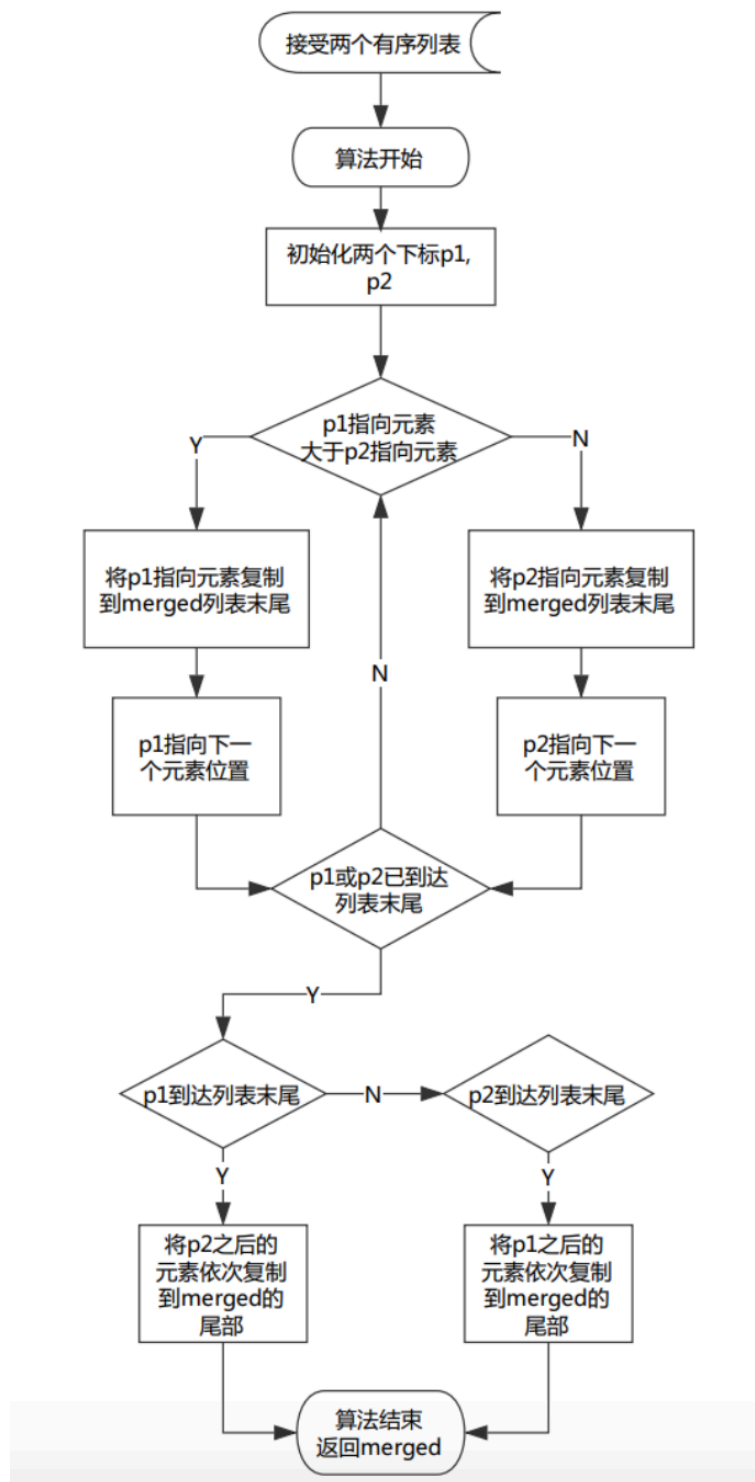


图 3: 任务 2: 归并算法

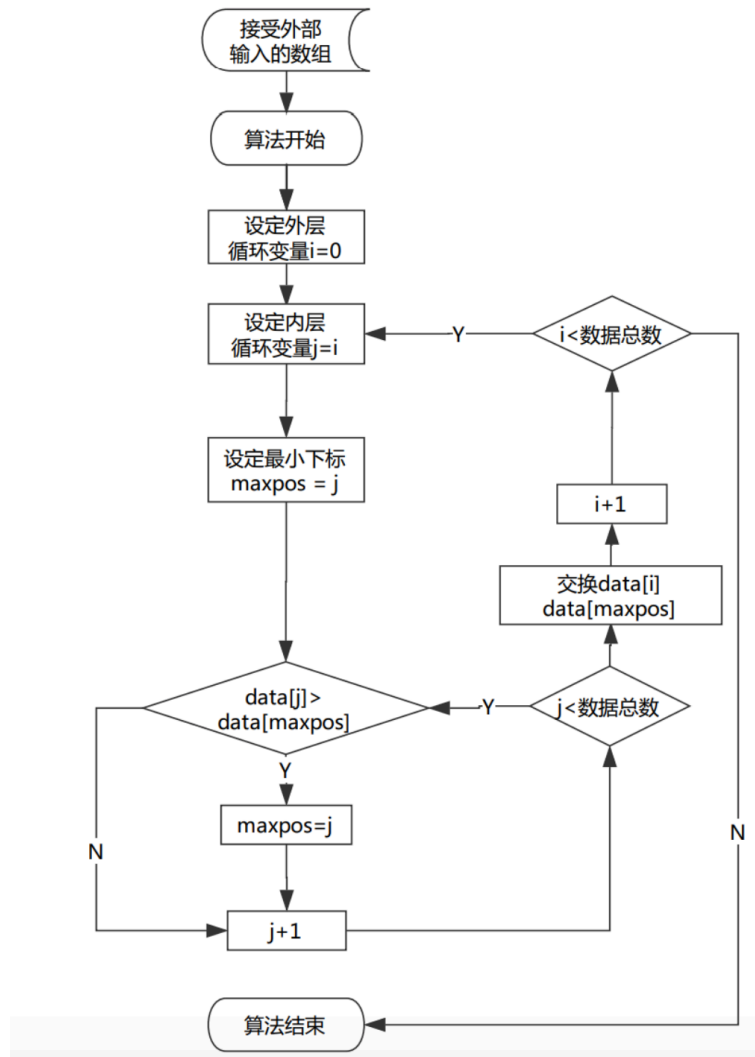


图 4: 任务 3: 选择排序

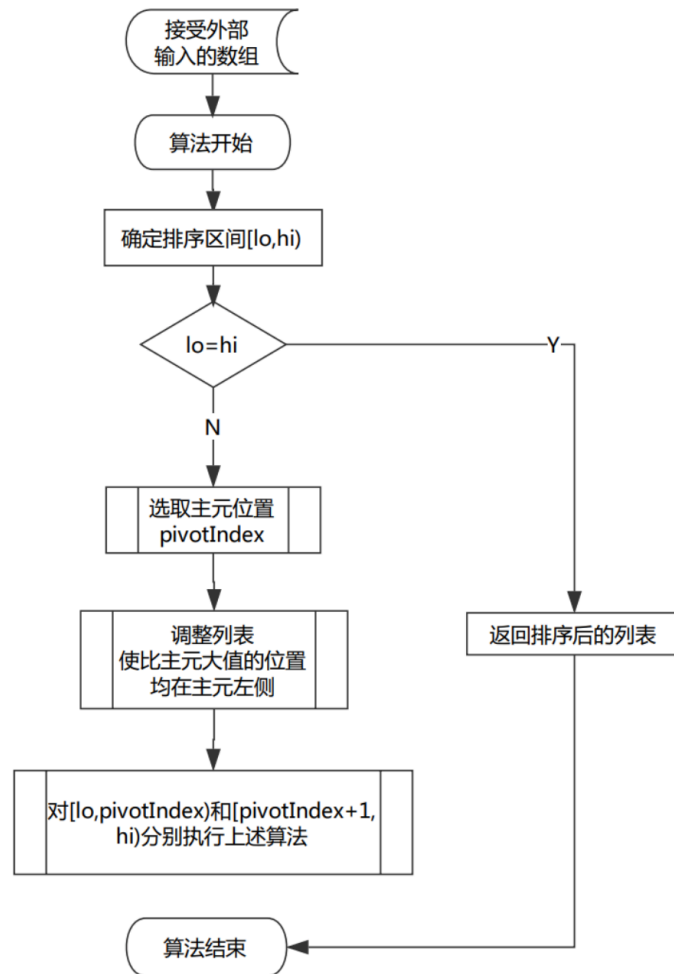


图 5: 任务 3: 快速排序

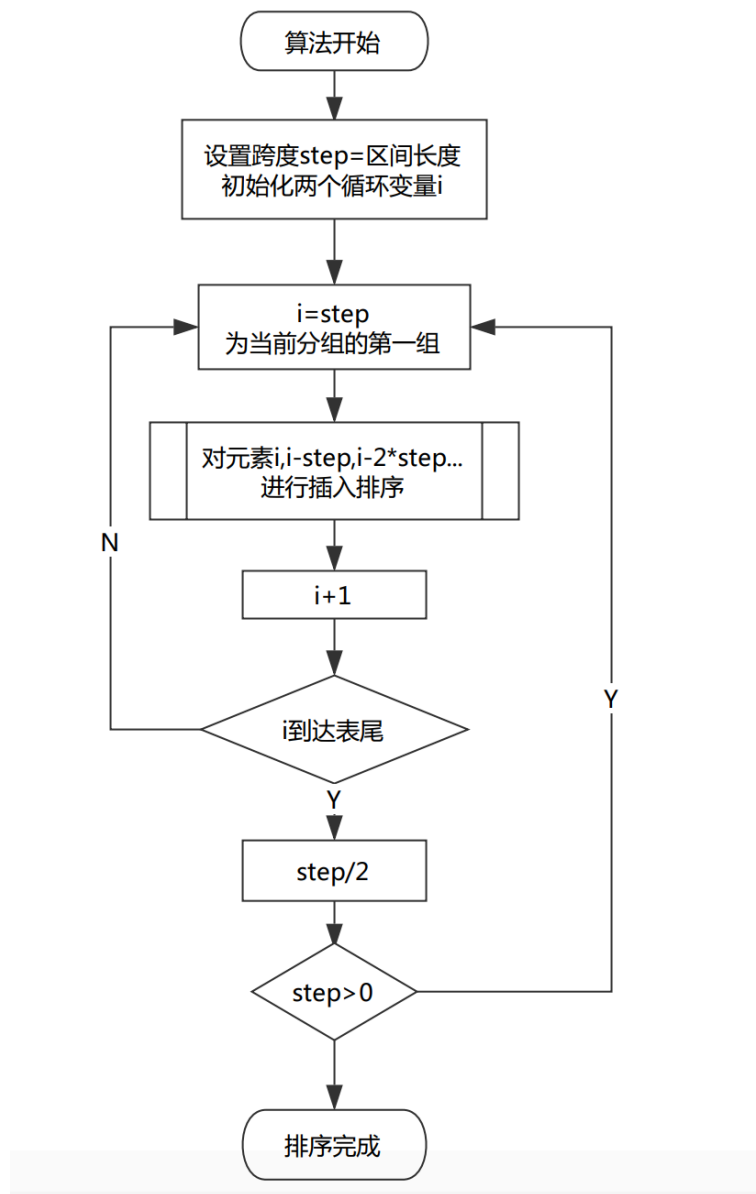


图 6: 任务 3: 希尔排序

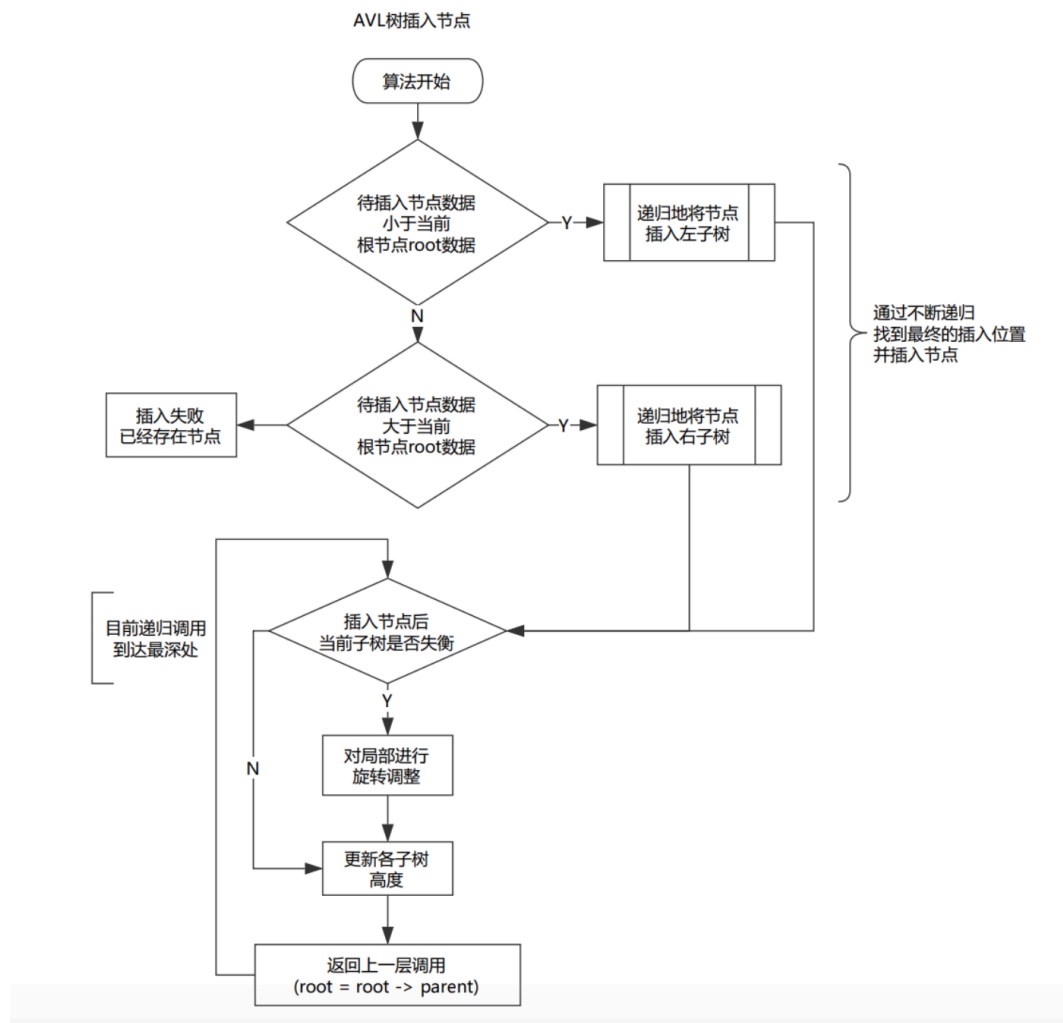


图 7: 任务 3: 创建 AVL 树 (插入节点)

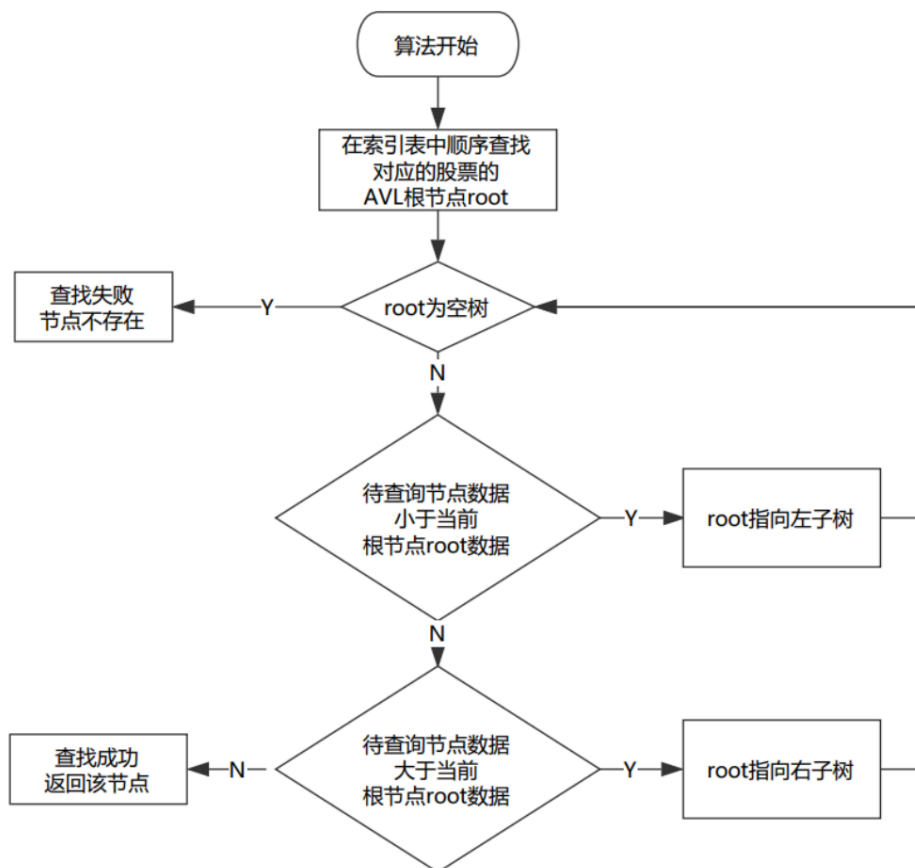


图 8: 任务 3:AVL 搜索算法

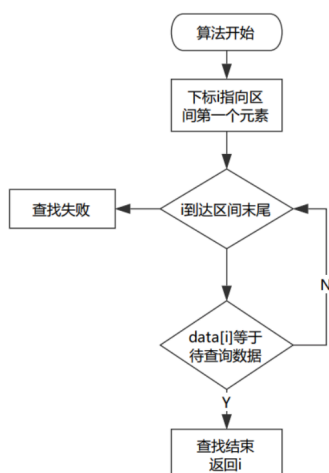


图 9: 任务 3: 线性查找算法

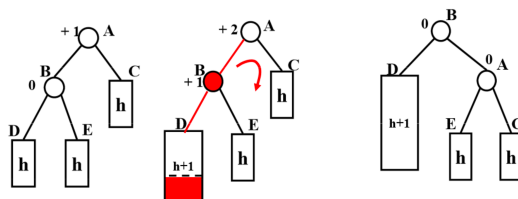


图 10: 任务 3: 二叉树重平衡（单侧）

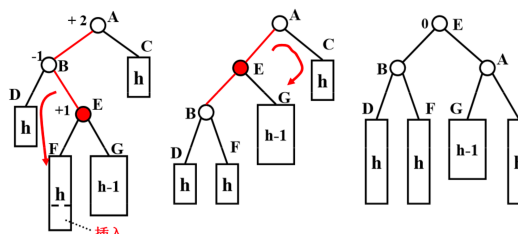


图 11: 任务 3: 二叉树重平衡（异侧）

3 用户手册

输入数据的格式

任务一 输入一组数据，用方括号包围，以逗号分隔，例如:[1,23,41]

任务二 输入若干行数据，要求必须有序，每行数据用方括号包围，以逗号分隔。

任务三 从文件读取数据，其中每条数据的格式为“股票代码 _ 交易日期 _ 开盘价 _ 最高价 _ 最低价 _ 收盘价 _ 交易量 _ 其他若干信息”，每个字段以“_”分割。

程序运行时的操作

任务一、二 无具体操作

任务三 程序运行时，会自动“data.txt”读取数据，读取完成后，会输出文字

打开文件成功!
读取文件时间 1.87895 s.

之后将输出一个菜单，输入对应选项回车后即可运行。

1. 快速排序
 2. 选择排序
 3. 希尔排序
 4. 创建索引 + 二叉平衡树 (AVL)
 5. 线性查找
 6. 索引 + 二叉平衡树 (AVL) 查找
- 输入操作:

注意 若未建立 AVL 树而直接选择索引 + 二叉平衡树 (AVL) 查找，系统将会先自动建立 AVL 树。为保证程序计时结果的正确性，每次进行排序或线性查找之前，将对序列进行重置。

输出数据的格式

任务一 输出排序后能组成的最大整数。

任务二 输出归并后的有序数组，以方括号包围，数字之间用逗号分隔。

任务三 排序：将排序后的信息按照原格式输出到外部文件 `result.txt` 文件；查询：输出查询到的股票详细信息。

4 总结

涉及到的数据结构 本实验涉及到的数据结构包括顺序表（任务一、二、三），索引表（任务三）和二叉平衡搜索树（AVL）（任务三）。

涉及到的算法 本实验涉及到的主要为排序算法。其中，任务一可以采取任意的排序算法，只是要重新定义比较规则。任务二主要借鉴了归并排序的思想，事实上就是归并排序的中间步骤。任务三涉及到不同时间复杂度的排序算法，例如：冒泡排序 ($O(n^2)$)、选择排序 ($O(n^2)$)、希尔排序 ($O(n\log n)$)、快速排序 ($O(n\log n)$) 等，查找算法主要涉及了线性查找和索引树查找。其中，线性查找不需要预处理，但平均查找的时间复杂度与序列长度成正比，为 $O(n)$ 。而索引树的查找，需要先定位索引的下标，由于索引并不多，这里使用顺序查找。在定位到索引后，对其所指的平衡二叉排序树进行深入查找。

遇到的问题 在进行文件写入操作时，写入文件后，忘记进行 `fclose` 操作，导致最终输出的文件被意外截断，少了几行。同时，由于数据量较大，数据域应该放在堆上而不是栈上，并且对于任何函数的传参，都应该采取指针的形式，避免由于传参过程中复制形参副本，栈空间不足，导致栈溢出 (Stack Overflow)。

5 结果

程序运行正确结果截图

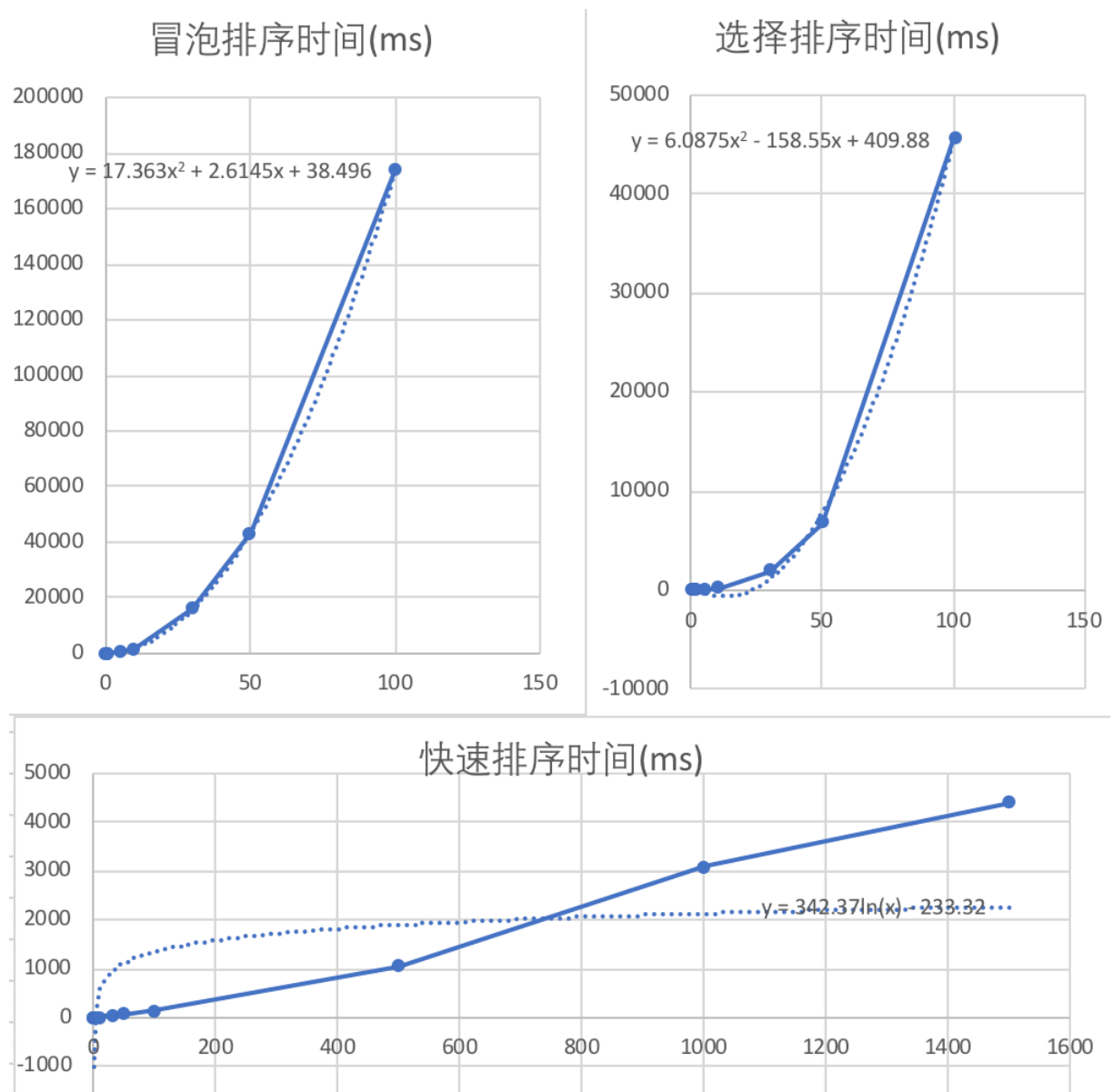


图 12: 排序时间对比

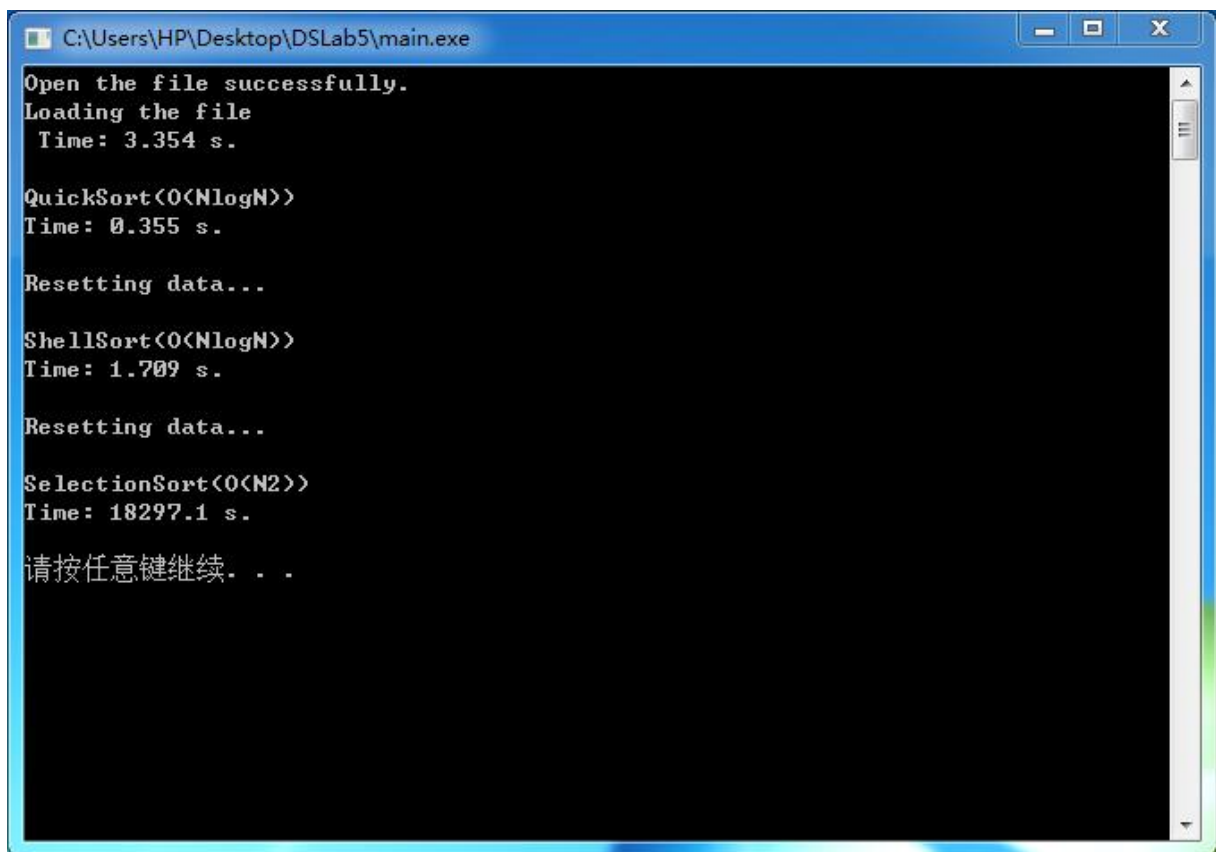


图 13: 排序时间截图（环境：机房电脑 + 编译器 O3 优化）

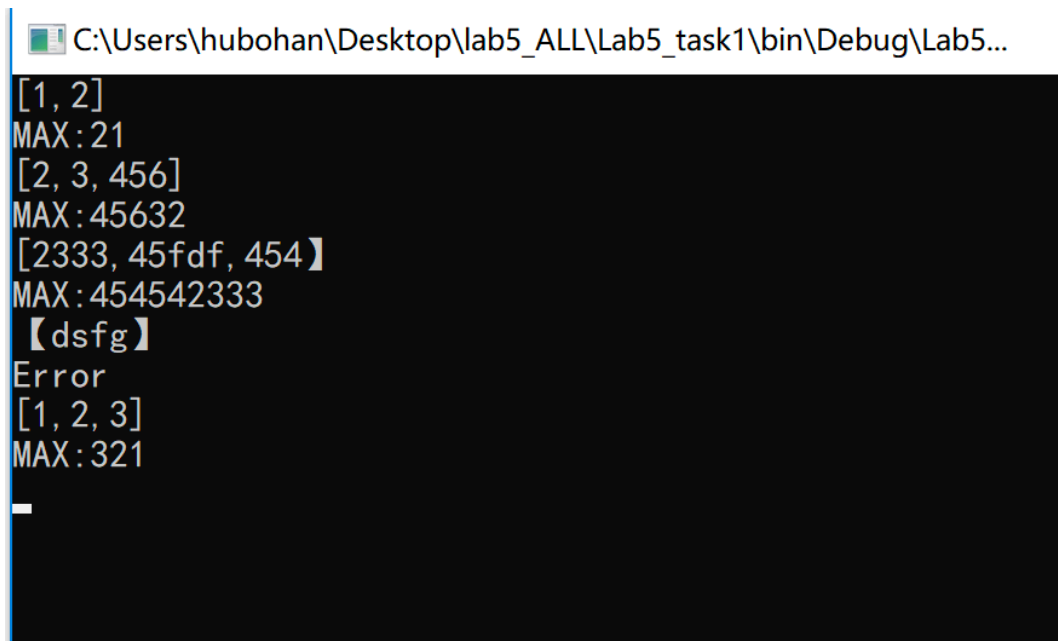


图 14: 任务一（包括异常输入处理）

```
C:\Users\hubohan\Desktop\lab5_ALL\Lab5_task2\bin\Debug\Lab5_task2.exe
[1, 2, 3]
[2, 3, 5]
[345, 45566]
-1
合并后的数组为:
[1, 2, 2, 3, 3, 5, 345, 45566]

Process returned 0 (0x0)    execution time : 12.766 s
Press any key to continue.
```

图 15: 任务二

```
C:\Users\hubohan\Desktop\lab5_ALL\Lab5_task2\bin\Debug\Lab5_task2.exe
[1, 2, 444]
[2, 3, dgfgdghgdhgdrtel]
【2, 3, 4】
-1
合并后的数组为:
[1, 2, 2, 2, 3, 3, 4, 444]

Process returned 0 (0x0)    execution time : 23.286 s
Press any key to continue.
```

图 16: 任务二: 异常数据处理



图 17: 任务三: 启动界面



图 18: 任务三: 快速排序

```

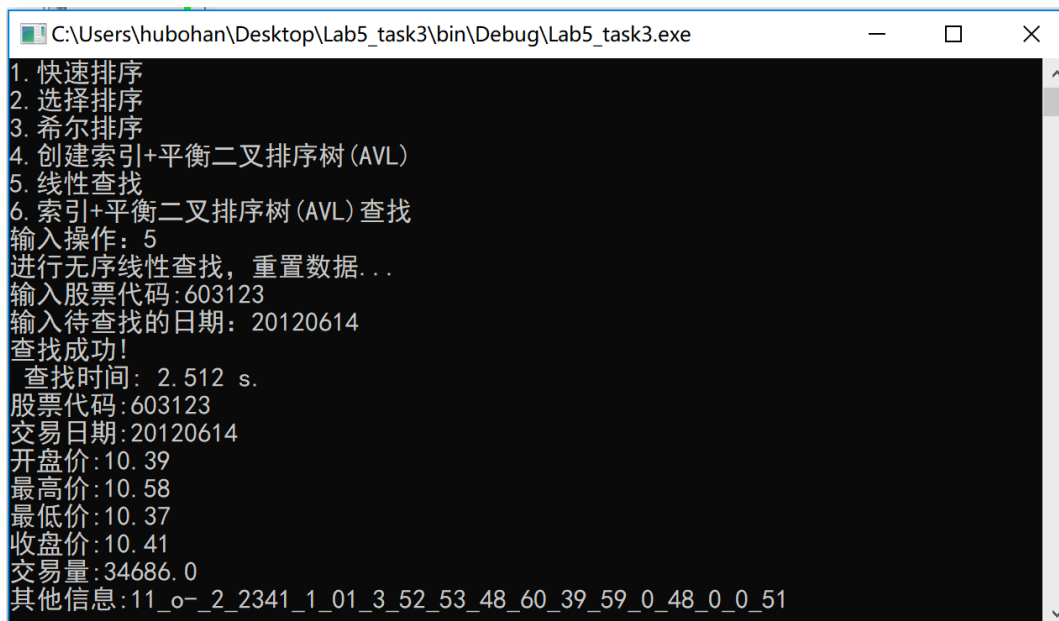
C:\Users\hubohan\Desktop\Lab5_task3\bin\Debug\Lab5_task3.exe
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
输入操作: 3
重置数据...
开始希尔排序 (O(NlogN))
排序时间: 8.093 s.
文件已输出。
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
    
```

图 19: 任务三: 希尔排序

```

C:\Users\hubohan\Desktop\Lab5_task3\bin\Debug\Lab5_task3.exe
打开文件成功!
读取文件时间 5.539 s.
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
输入操作: 5
进行无序线性查找, 重置数据...
输入股票代码: asdfs
输入待查找的日期: 332424
查找失败!
查找时间: 0.025 s.
1. 快速排序
    
```

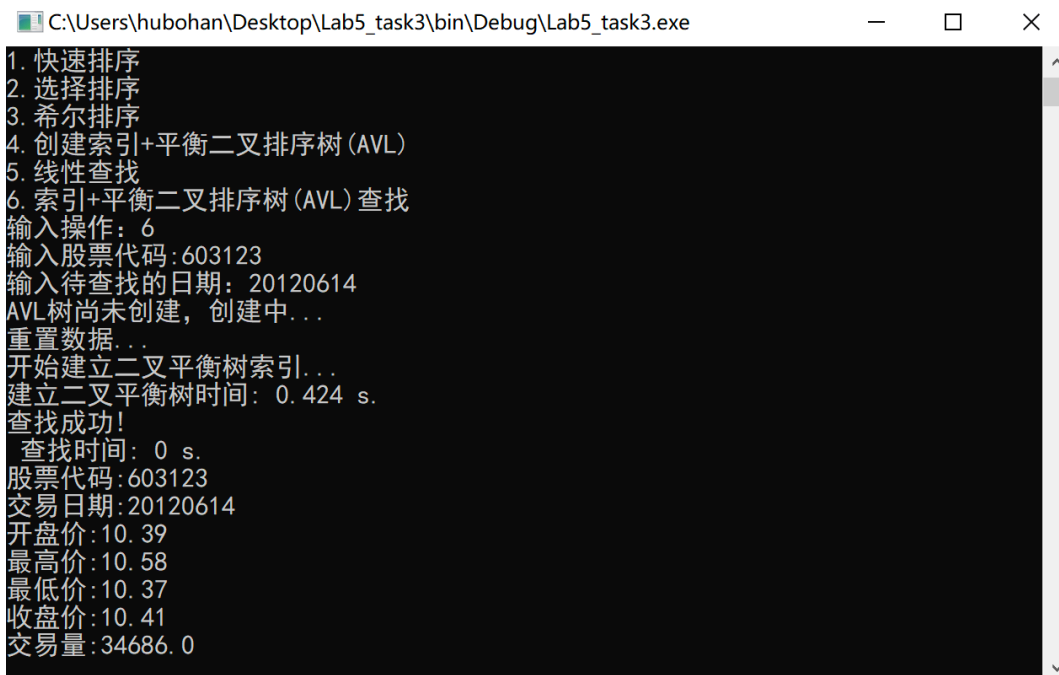
图 20: 任务三: 线性查找失败



```

C:\Users\hubohan\Desktop\Lab5_task3\bin\Debug\Lab5_task3.exe
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
输入操作: 5
进行无序线性查找, 重置数据...
输入股票代码: 603123
输入待查找的日期: 20120614
查找成功!
  查找时间: 2.512 s.
股票代码: 603123
交易日期: 20120614
开盘价: 10.39
最高价: 10.58
最低价: 10.37
收盘价: 10.41
交易量: 34686.0
其他信息: 11_o-_2_2341_1_01_3_52_53_48_60_39_59_0_48_0_0_51
  
```

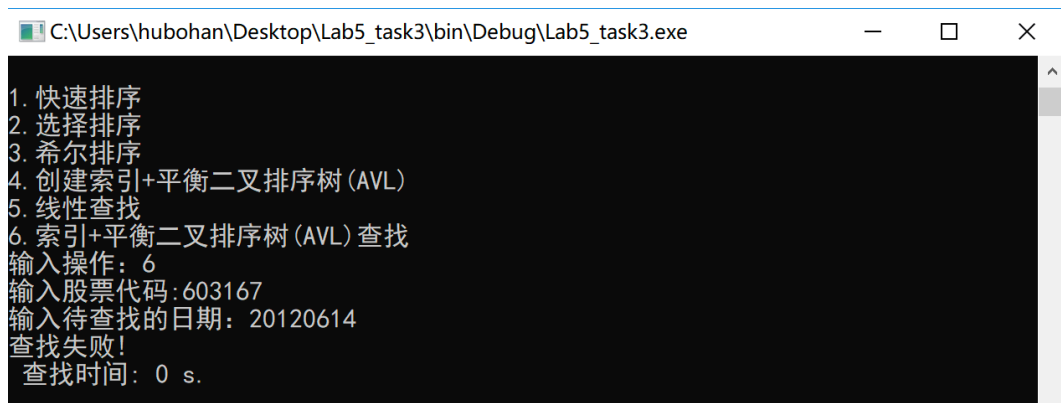
图 21: 任务三: 线性查找成功



```

C:\Users\hubohan\Desktop\Lab5_task3\bin\Debug\Lab5_task3.exe
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
输入操作: 6
输入股票代码: 603123
输入待查找的日期: 20120614
AVL 树尚未创建, 创建中...
重置数据...
开始建立二叉平衡树索引...
建立二叉平衡树时间: 0.424 s.
查找成功!
  查找时间: 0 s.
股票代码: 603123
交易日期: 20120614
开盘价: 10.39
最高价: 10.58
最低价: 10.37
收盘价: 10.41
交易量: 34686.0
  
```


图 22: 任务三: 首次查找, 创建索引树 + 查找成功



```

C:\Users\hubohan\Desktop\Lab5_task3\bin\Debug\Lab5_task3.exe
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
输入操作: 6
输入股票代码: 603167
输入待查找的日期: 20120614
查找失败!
查找时间: 0 s.
    
```

图 23: 任务三: 已有索引树 + 索引树查找失败



```

C:\Users\hubohan\Desktop\Lab5_task3\bin\Debug\Lab5_task3.exe
1. 快速排序
2. 选择排序
3. 希尔排序
4. 创建索引+平衡二叉排序树 (AVL)
5. 线性查找
6. 索引+平衡二叉排序树 (AVL) 查找
输入操作: 6
输入股票代码: 603167
输入待查找的日期: 20120925
查找成功!
查找时间: 0 s.
股票代码: 603167
交易日期: 20120925
开盘价: 9.15
最高价: 9.27
最低价: 8.8
收盘价: 8.92
交易量: 72638.4
其他信息: 31_o+_2_3421_1_01_3_52_22_35_50_38_19_0_0_0_0_32
    
```

图 24: 任务三: 已有索引树 + 查找成功

6 感想与建议

经过一个学期的数据结构的学习，我有了很大的收获。第一，由浅入深学习了不同的数据结构：从最简单的线性结构，到图状结构，再到树形结构，并了解了这些数据结构的特征和性质。第二，学习了部分算法，学会了分析算法的复杂度，并可以尝试设计算法。第三，编码和调试能力得到了提升，在数据结构的实验中，自己去实现特定的数据结构，总会遇到各种各样的问题，经过耐心的调试，最终都能得到解决；同时，在实现数据结构的过程中，还学习到一些程序的优化技巧和提高代码可读性的一些方法。

建议：实验课上提供的测试样例过少，建议模仿 lintcode 的模式，提供处理输入输出的模板，答题者只需完成 Solution 函数的编写，使学生在实验课上专注于算法的实现，而非纠结类似于 [1,2,...] 等字符串的处理。同时，在评测时，建议使用 Online Judge 的方式而非人工验收，以减少工作量，增加公平性。课下实验报告的撰写有时会耗费比编写代码更长的时间，建议优化实验报告的结构，对于代码部分可以交由 Online Judge 评测，并统一流程图标准。使用 Microsoft Word 进行实验报告的排版经常耗费过多时间。本次实验报告用 L^AT_EX 排版，建议可以同时提供 L^AT_EX 模板，提高实验报告书写效率。