数据结构

实验报告（九）

快速排序

学号：3020205015

姓名：石云天

班级：智能机器平台 2 班

日期：2022.12. 23

目　录

一、实验内容描述

本次实验主题是快速排序算法，实现后在图 1 的待排序样例上进行验证：

图 1 实验要求验证的待排序样例

二、实验步骤

（1）根据上课所讲，回顾快速排序的基本概念。
（2）仔细阅读实验要求，考虑快速排序算法的核心思想与实现方法。
（3）利用 CodeBlocks 编译器，配置环境，基于 C++语言将算法用程序实现。
（4）编译运行程序，使用样例进行程序测试，观察所编程序是否实现要求的功能。
（5）考察算法的时间复杂度和空间复杂度，评价算法的优劣，进一步优化程序。
（6）撰写实验报告，进行实验总结与反思。

三、程序设计

（一）抽象数据类型 ADT

本次实验主要是实现快速排序算法，不涉及具体数据结构的刻画，因此无需写出抽象数据类型。

（二）存储结构

本次实验利用顺序存储结构实现快速排序的过程，使用数组储存待排序的数据元素。

（三）算法简述

首先对于快速排序算法的基本思想，其主要步骤如下：

1. 设置两个指针 low 和 high，设枢轴记录的关键字为 pivotkey；

2. 首先从 high 指针所指位置起向前搜索，找到第一个关键字小于 pivotkey 的记录，并将其与枢轴记录交换；

3. 随后从 low 指针所指位置起向后搜索，找到第一个关键字大于 pivotkey 的记录，并将其与枢轴记录交换；

4. 重复 2、3 步，直至 high == low 结束。

将上述步骤利用流程图的形式表示，所得结果见下图 2：

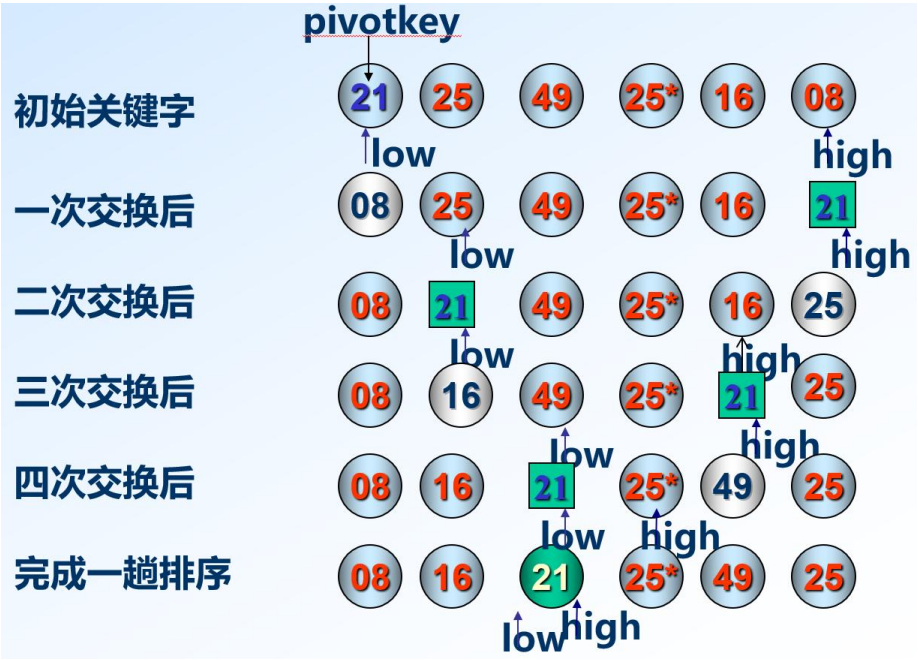

图 2 快速排序算法流程图

（四）程序代码

为保证实验报告的清晰和可读性，将源代码以**附录形式**附于文末。

四、调试分析

（一）调试过程和主要错误

在程序编写完成后，使用 CodeBlocks 编译并运行，基本没有出现问题。最初由于空指针的问题没有注意，编译器报错 Segmentation Fault，仔细检查代码后，经过适当修改，程序顺利运行。

（二）时间复杂度

一般情况下：快速排序的平均计算时间为$O(n\log_2 n)$，实验结果表明：在平均计算时间层面，快速排序是所有排序方法中最优的一种。

五、程序测试

在完成全部程序编写后，输入测试样例进行测试，所得结果均满足要求，具体测试结果见图 3：

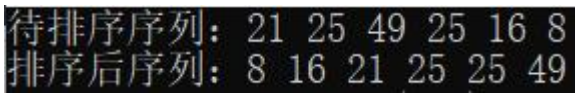

图 3 快速排序算法测试结果

六、实验总结

通过这次试验，我发现我对排序这一部分的理解不够深入全面，需要不断巩固学习，加深理解。同时。在编程过程中需要完成某些特定目标时，我不能很快的想出其对应的操作，需要课下不断练习以熟能生巧，还可以多查阅一些资料以开阔自己的思路。在本次实验中，我编写并实现了快速排序算法，在此过程中不断调试，寻找问题，并不断简化代码，提升函数执行速度。除此之外，在本次实验过程中，编写、调试程序花费了很长时间：首先是快速排序要求在顺序结构中存储，而且其是不稳定排序，利用分而治之的思想逐步完成整个数组的排序。其次对于空指针的理解更加深刻，我最初以为 Segmentation Fault 类型报错是因为数组越界，经过不断查阅资料发现是因为调用了空指针，空指针不会指向任何实体，因此在程序编写过程中需要格外注意各指针变量指向的变化，在 delete 操作完成后，最好在后面加一行将指针置为 NULL 的代码，这可以有效避免调用空指针的错误。最后是关于结果部分不正确的情况，这一问题往往是比较棘手的，此时需要在边界条件上入手，寻找没有关注到的情况，让思考更加全面周到，有助于顺利解决问题。在本次实验后，我还需要精益求精，不断改进程序，优化函数性能，实现预期目标与功能所需。

附录 1：程序源代码：快速排序算法

```
#include <iostream>

using namespace std;

void QuickSort(int *array, int low, int high){//快速排序
    if(low >= high){//若待排序序列只有一个元素，返回空
        return ;
    }
    int i = low;//i 作为指针从左向右扫描
    int j = high;//j 作为指针从右向左扫描
    int key = array[low];//第一个数作为基准数
    while(i < j){
        while(array[j] >= key && i < j){//从右侧寻找小于基准数的元素 （此处仍需判断 i 是否小于 j）
            j--;//找不到则 j-1
        }
        array[i] = array[j];//找到则赋值
        while(array[i] <= key && i < j){//从左边找大于基准数的元素
            i++;//找不到则 i+1
        }
        array[j] = array[i];//找到则赋值
    }
    array[i] = key;//当 i 和 j 相遇，将基准元素赋值到指针 i 处
    QuickSort(array, low, i - 1);//i 左侧序列递归调用快排
    QuickSort(array, i + 1, high);//i 右侧序列递归调用快排
}
int main(){
    int array[] = {21, 25, 49, 25, 16, 8};
    int length = sizeof(array) / sizeof(*array);
    cout << "待排序序列：";
    for(int i = 0; i < length; i++){
        cout << array[i] << " ";
    }
    cout << endl;
```

```
    QuickSort(array, 0, length - 1);
    cout << "排序后序列：";
    for(int i = 0; i < length; i++){
        cout << array[i] << " ";
    }
    return 0;
}
```