# 四川师范大学 实验报告

学期： 2024 至 2025 第一学期 实验成绩：

课程名字：《程序设计基础——数据结构(C语言版)》 专业： 信息与计算科学

班级： 2023 级 9 班 实验编号： 09

实验项目： 实验九 指导老师： 冯山

姓名： 刘智恒 学号： 2023060522

**一、实验题目**

内部排序算法

**二、实验目的及要求**

1.熟练掌握各种排序的算法思想。

2.掌握快速排序、堆排序、归并排序等算法的实现方法。

3.熟悉各种排序算法的时空复杂度分析方法。

**三、实验内容：(类C算法的程序实现，任选其二)**

1.选择排序的算法实现(简单选择排序和堆排序)。(必做)

2.交换排序的算法实现(快速排序和冒泡排序)。

3.插入排序的算法实现(直接插入排序、Shell排序和2-路插入排序)。

4.归并排序的算法实现。

5.多关键字排序的算法实现(LSD或MSD)。

**四、实验准备**

1.计算机设备;

2.程序调试环境的准备，本实验采用**Microsoft Visual Studio**环境;

3.实验内容的算法分析与代码设计与分析准备；

4.实验源程序**Exp\_9**准备。

5.实验测试用例：

**（1）第一组**

**关键字数目：8；**

**待排关键字序列：{49，38，65，97，76，13，27，49}。**

**（2）第二组**

**关键字数目：8；**

**待排关键字序列：{-2，29，0，-67，-89，100，999，2147483647}。**

**（3）第三组**

**关键字数目：15；**

**待排关键字序列：{0，4，3，1，0，3，1，2，1，2，0，2，1，0，0}。**

**五、实验内容**

**（一）问题分析**

（1）针对实验内容1，简单**选择排序**（**Select Sort**）的基本思想为：每一趟在**n-i+1**（i=1,2,…,n-1）个记录中**选取关键字最小的记录**作为有序序列中的第**i**个记录。

**堆排序**（**Heap Sort**）的定义为：

其实质是满足如下性质的**完全二叉树**：

**任一非叶子结点均小于（或大于）它的孩子结点**。

基本思想为：若在输出**堆顶的最小值**（或最大值）后，使得剩余**n-1**个元素的序列又建成一个堆，则得到**n**个元素的**次小值**（或次大值）......，如此反复，便能得到一个有序序列，这个过程称之为堆排序。

关于**堆的调整**：（以**小根堆**为例）

**[1]** 输出堆顶元素后，以堆中最后一个元素替代之；

**[2]** 然后将根结点值与左、右子树的根结点值进行比较，并与其中较小者进行交换；

**[3]** 重复上述操作，直至叶子结点，从而得到新的堆，称这个**从堆顶至叶子**的调整过程为“**筛选**”。

（2）针对实验内容2，**快速排序**（**Quick Sort**）是对**冒泡排序**的一种改进，基本思想为：通过一趟排序**将待排记录分割成独立的两部分**，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录继续排序，以达到整个序列有序。

通常选定一个**中间数**（**枢轴**）作为参考，所有元素与之比较，**小的调到其左边，大的调到其右边**（枢轴可以是第一个数、最后一个数、最中间一个数、任选一个数等）。

（3）针对实验内容3，**直接插入排序**（**Straight Insertion Sort**）的基本思想为：每步将一个待排序的对象，按其关键字大小，插入到前面**已经排好序**的一组对象的适当位置上，直到对象全部插入为止。即**边插入边排序**，保证子序列中随时都是排好序的。

**（注：本实验只包含实验内容1、实验内容2中的快速排序、实验内容3中的直接插入排序以上内容，由于其存储结构具有共通性，故将实验内容1，实验内容2以及实验内容3整合为了Exp\_9）**

（4）本实验的存储结构选用**顺序表**结构，结构定义如下：

**typedef struct{**

**KeyType key;**

**}ElemType;**

**typedef struct SqList{**

**ElemType \*R;**

**int length;**

**}SqList;**

**（二）算法描述**

**Exp\_9:**

（1）**Status InitSqList(SqList \*L, int n);**

//本算法用于初始化顺序表：输入顺序表**L**及其表长**n**，为顺序表分配**n+1**个元素内存空间，初始化顺序表长为**n+1**。

（2）**Status DesSqList(SqList \*L);**

//本算法用于销毁顺序表：输入顺序表**L**，若顺序表的元素数组**R**不为空，则释放其内存，将**R**置为空，顺序表长置为**0**。

（3）**void PrintElem(SqList L);**

//本算法用于打印顺序表中的元素：输入顺序表**L**，遍历顺序表**L**的元素数组**R**，打印每个元素的**key**值。

（**选择排序**）

（4）**int SelectMinKey(SqList L, int i);**

//本算法用于在顺序表指定范围内选择最小键值的元素索引：遍历顺序表**L**的元素数组**R**，如果找到键值小于当前**L.R[j].key**的元素，则更新**j**为该元素的索引，最终返回最小键值元素的索引**j**。

（5）**void SelectSort(SqList \*L);**

//本算法用于对顺序表进行简单选择排序：输入顺序表**L**，遍历顺序表**L**的元素数组**R**，调用**SelectMinKey**函数找到范围内最小键值元素的索引**j**，如果当前**i**不等于**j**，则交换**R[i]**和**R[j]**的元素。

（**堆排序**）

（6）**void HeapAdjust(SqList \*L, int s, int m);**

//本算法用于调整顺序表中的元素使其满足**小根堆**的性质：输入顺序表**L**，起始索引**s**，结束索引**m**，先取出**R[s]**元素到临时变量**rc**，然后从**j=2\*s**开始，在**R[s…m]**范围内进行循环调整：如果**j<m**且**R[j].key>R[j+1].key**，则**j**指向键值较小的子节点（右子节点）；如果**rc.key<=R[j].key**，则跳出循环，将**R[s]**设为**R[j]**，更新**s**为**j**。循环结束后，最后将**rc**插入到**s**位置。

（7）**void Swap(ElemType root, ElemType last);**

//本函数用于交换两个元素（根结点和最后一个结点）的值：输入两个元素**root**和**last**，借助临时变量**temp**进行交换**root**和**last**的值。

（8）**void HeapSort(SqList \*L, int n);**

//本算法用于对顺序表进行堆排序：输入顺序表**L**，表长**n**，从**i=n/2**到**1**倒序遍历，调用**HeapAdjust**函数建立初始小根堆，然后从**i=n**到**2**倒序遍历，进行**n-1**趟排序，在每一趟排序中，交换**R[1]**和**R[i]**，并调用**HeapAdjust**函数对**R[1]**到**R[i-1]**重新排序建堆。

（**快速排序**）

（9）**int Partition(SqList \*L, int low, int high);**

//本算法用于对顺序表进行快速排序的划分：输入顺序表**L**，低索引**low**，高索引**high**，选择**R[low]**作为枢轴元素并保存到**R[0]**，然后设置枢轴键值**pivotkey**为**R[low].key**；从两端向中间扫描，将小于枢轴键值的元素移到左边，大于枢轴键值的移到右边；最后将枢轴元素放到正确位置，返回枢轴元素的最终位置**low**。

（10）**void QSort(SqList \*L, int low, int high);**

//本算法用于对顺序表进行快速排序：输入顺序表**L**，低索引**low**，高索引**high**，如果**low<high**，调用**Partition**函数进行划分，得到枢轴位置**pivotloc**，递归地对低子表**[low, pivotloc-1]**和高子表**[pivotloc+1, high]**进行快速排序。

（**直接插入排序**）

（11）**void InsertSort(SqList \*L);**

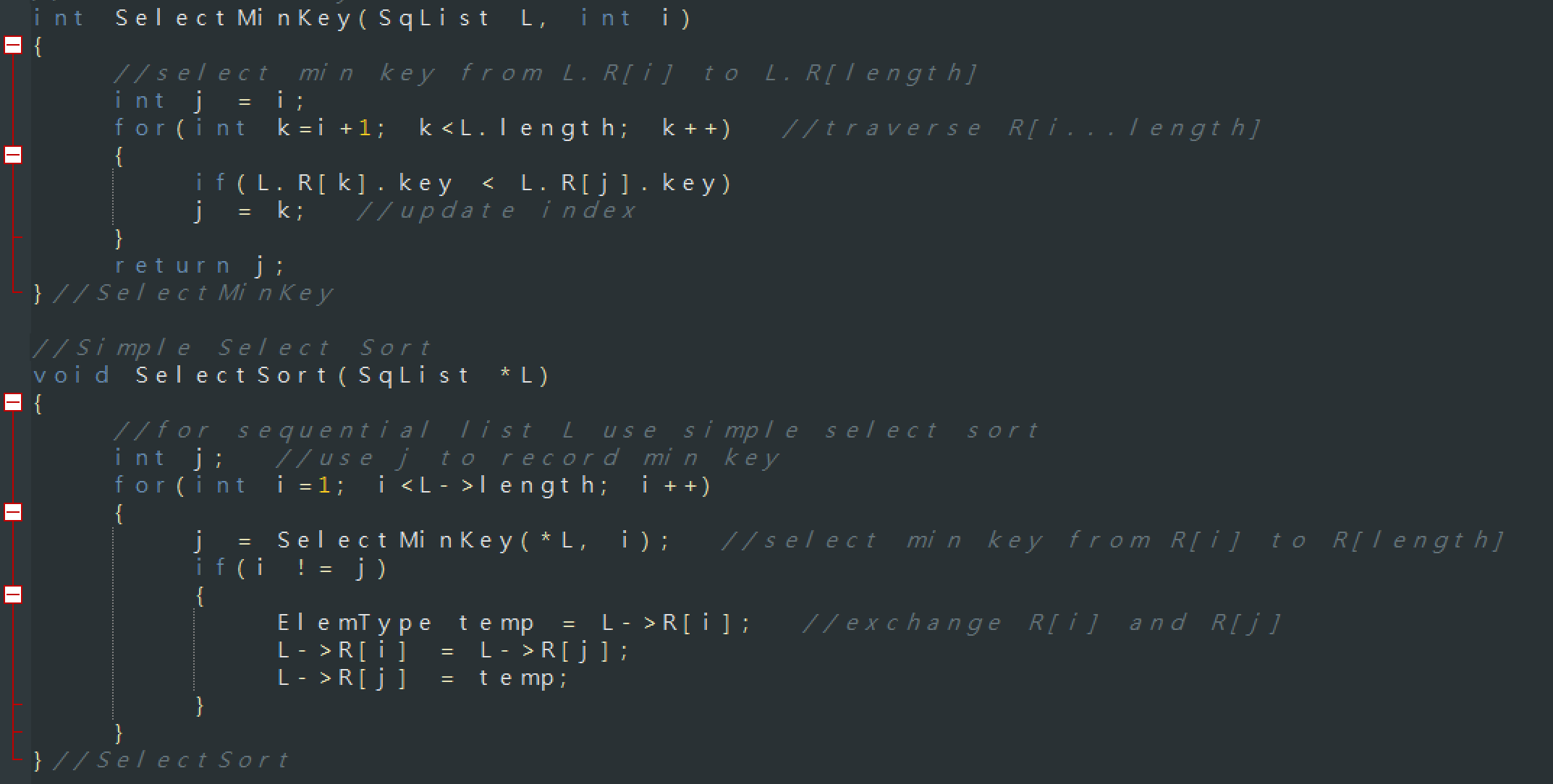
//本算法用于对顺序表进行直接插入排序：输入顺序表**L**，遍历顺序表**L**的元素数组**R**，如果当前**R[i].key<R[i-1].key**，则将**R[i]**插入到前面已经排好序的序列的正确位置。插入过程通过将大于R[i]的元素后移来腾出位置，最后将**R[i]**插入到正确位置。

**（三）程序代码**

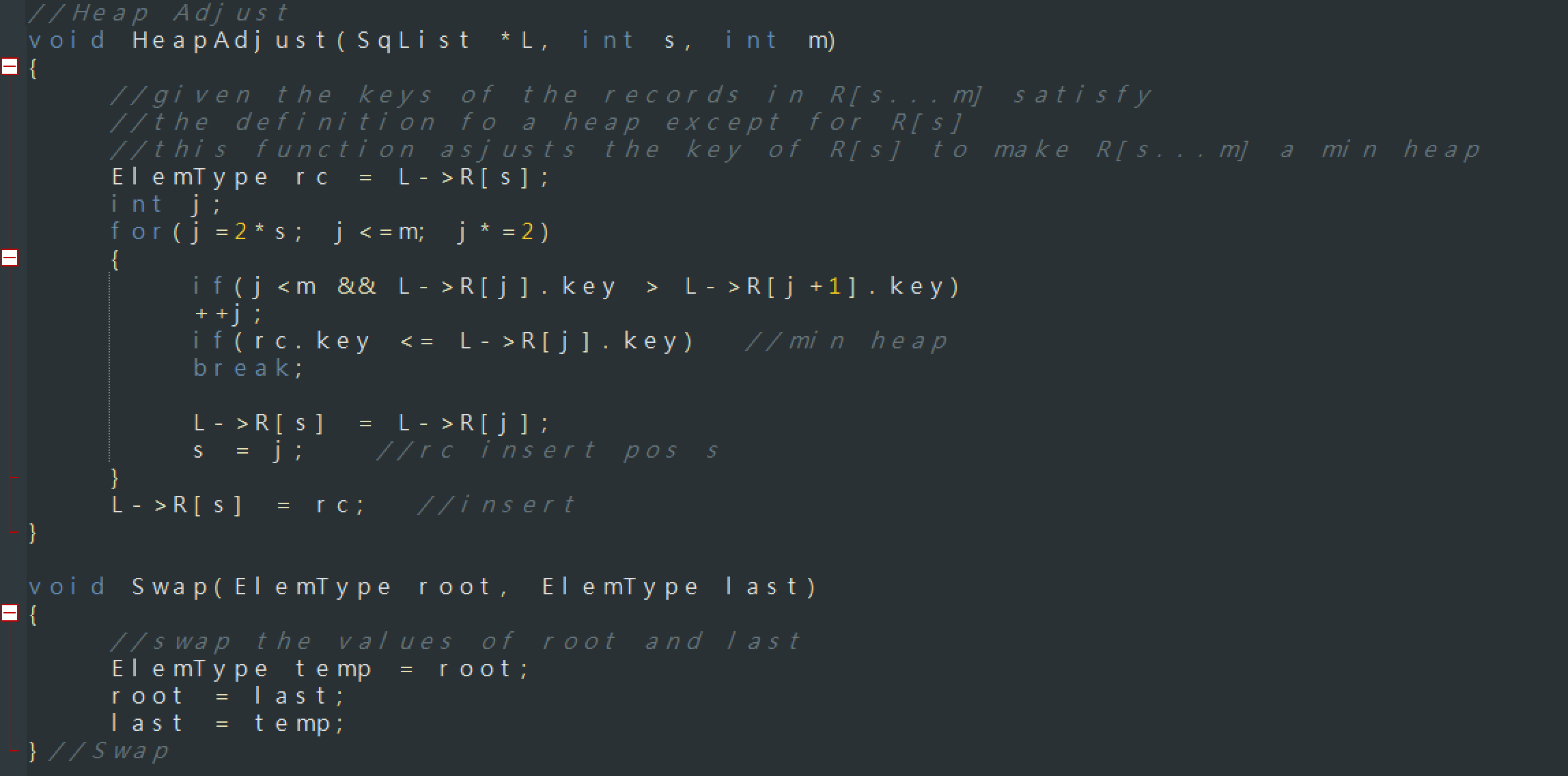
根据以上对**实验内容**的算法描述，求解的程序代码如下：



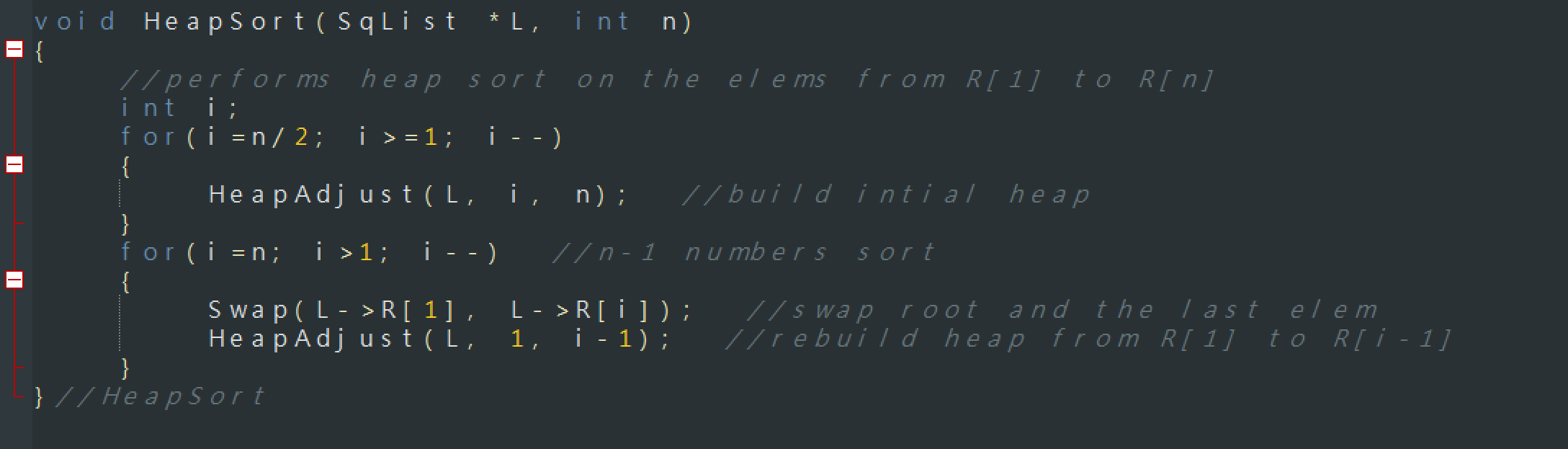
**注：此处只展示主要算法的程序代码图。**



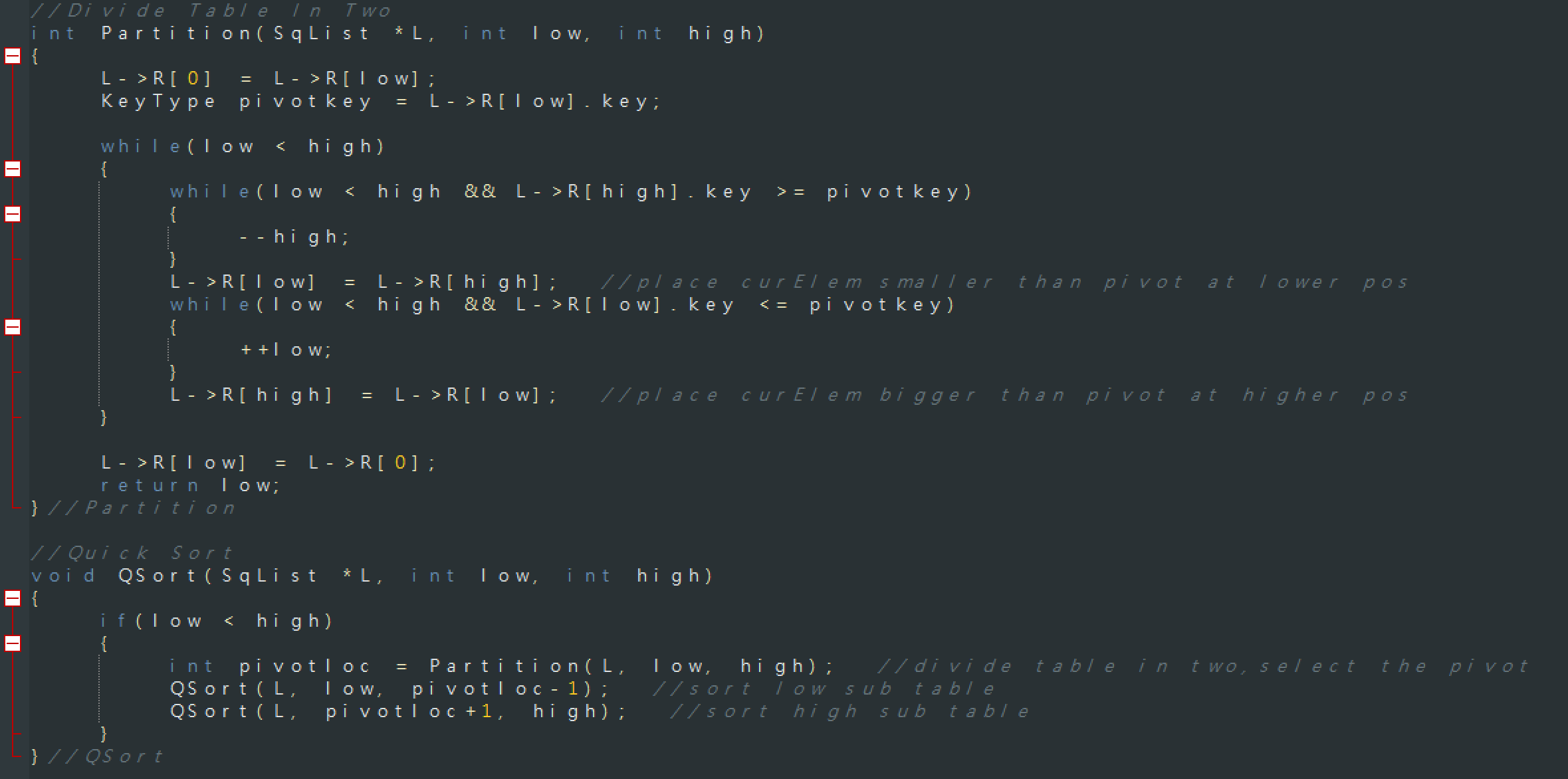
**图5.1选取最小键值**



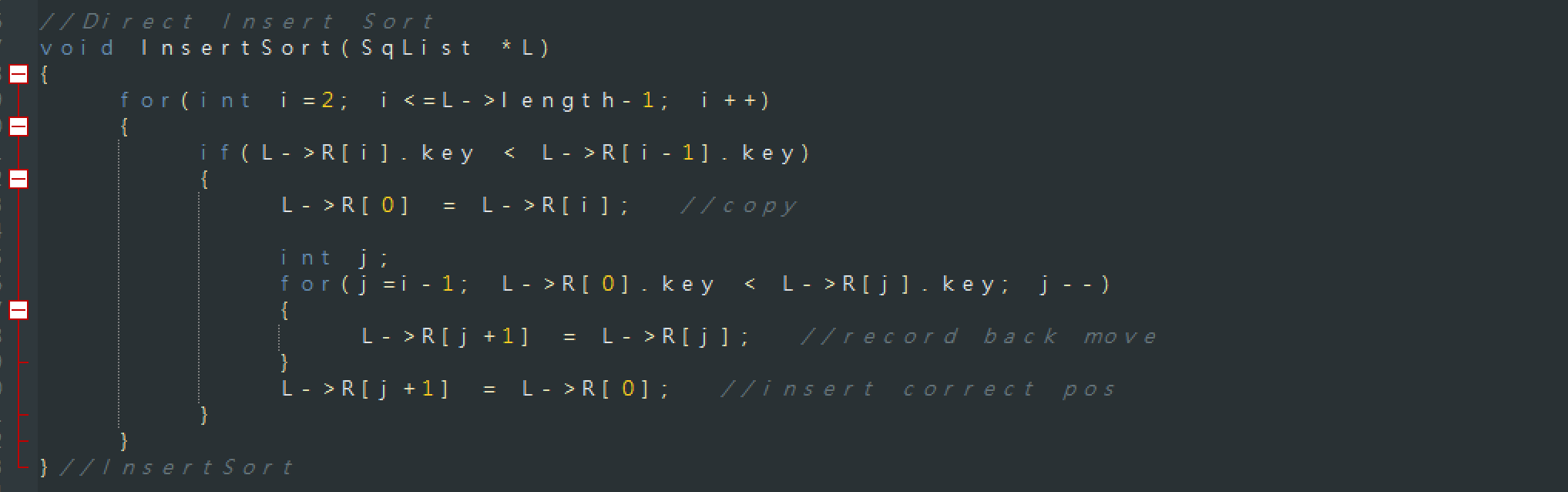
**图5.2堆的调整**



**图5.3堆排序**



**图5.4快速排序**

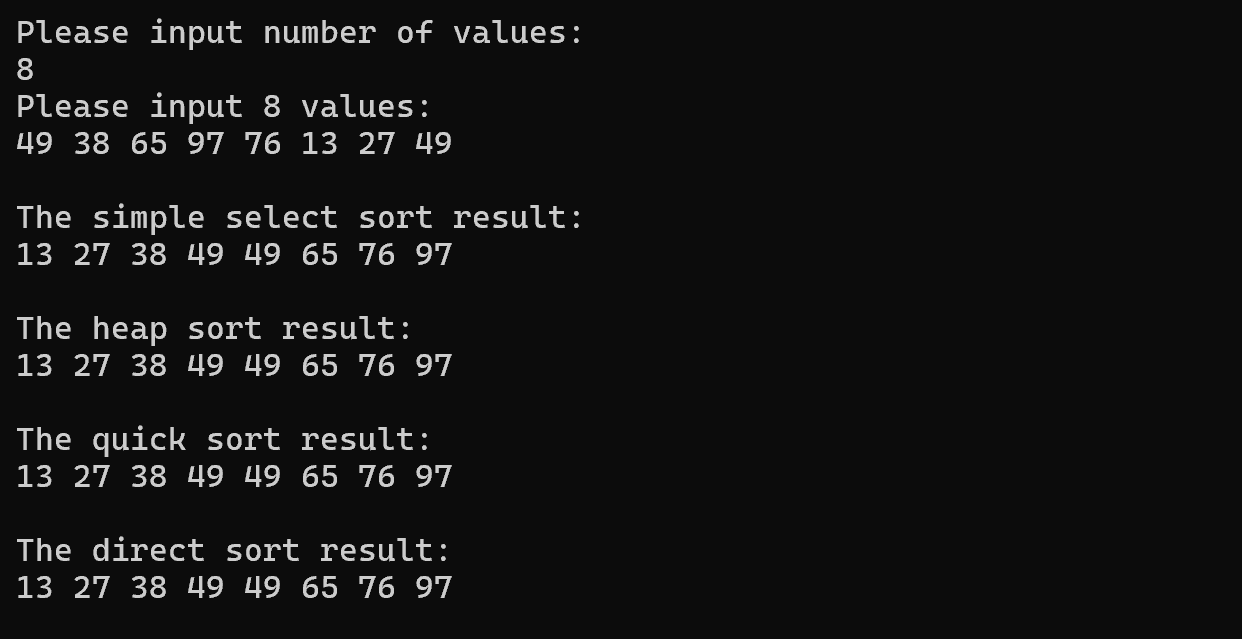


**图5.5直接插入排序**

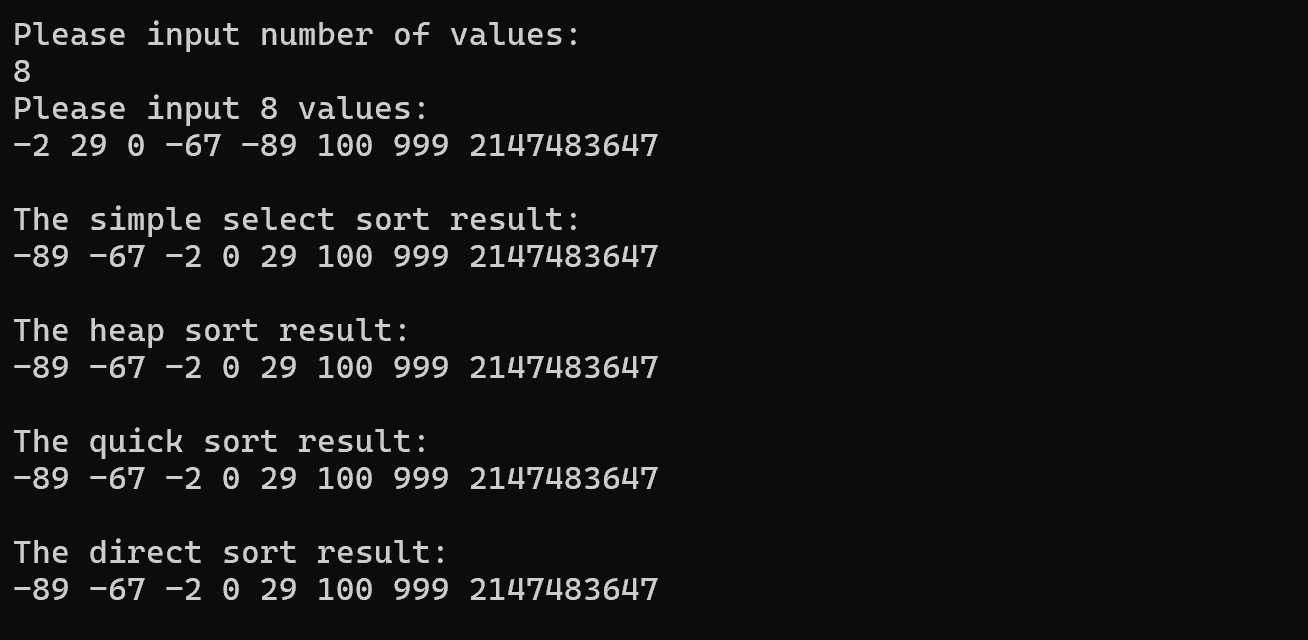
**六、实验结果**

**（一）结果呈现：**

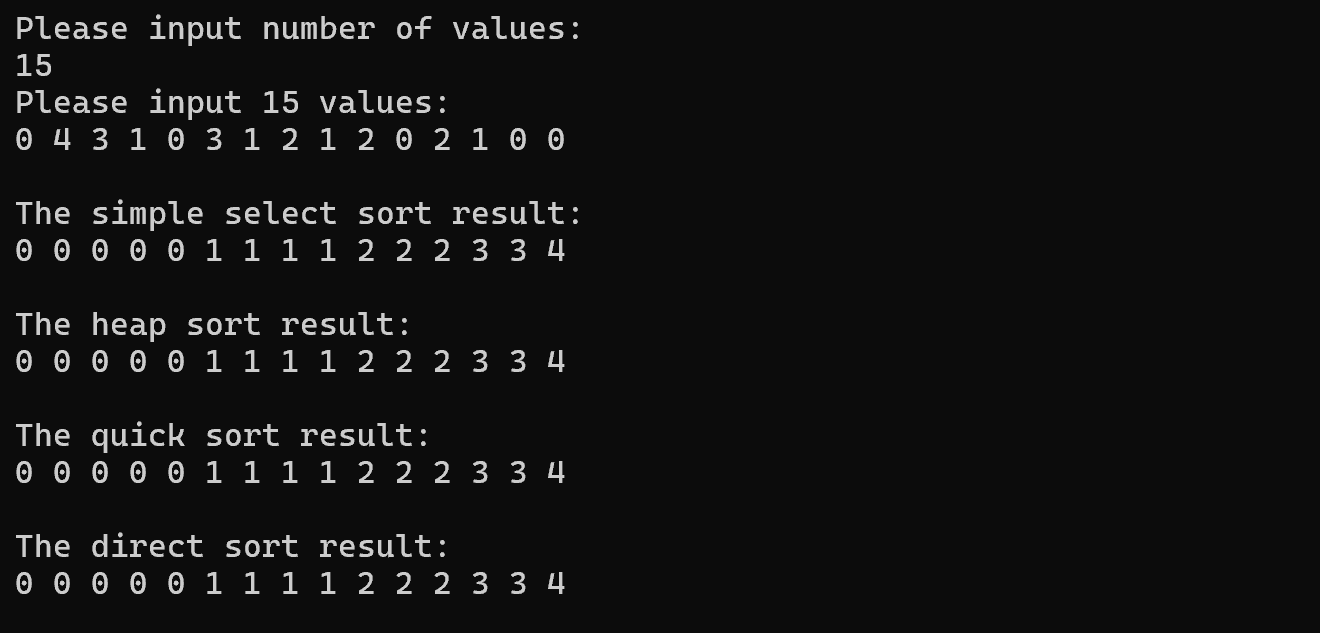
1.根据以上**实验内容1、2、3**的程序代码**Exp\_9**运行后进行多次测试，以下是根据**实验测试用例**进行测试后的结果示意图：

****

**图6.1 Exp\_9第一组实验测试用例结果**

****

**图6.2 Exp\_9第二组实验测试用例结果**

****

**图6.3 Exp\_9第三组实验测试用例结果**

**（二）结果分析：**

**时间与空间复杂度分析**

**【SelectMinKey】—— 选取最小键值**

函数通过一个循环遍历从索引**i+1**到**L.length-1**的元素来查找最小值的索引，循环执行次数最多为**L.length-i-1**，在最坏、平均、最好情况下时间复杂度均为**O(n - i)**（这里**n**可看作**L.length**）。只使用了有限的几个额外变量（如 **j**、**k**等），空间复杂度为**O(1)**。

【**SelectSort**】**—— 选择排序**

它包含两层循环，外层循环执行**L->length-1**次，每次都调用**SelectMinKey**函数，而**SelectMinKey**函数在最坏、平均、最好情况下时间复杂度与元素个数有关，所以整体最坏、平均、最好情况下时间复杂度均为 **O(n^2)**（**n为L->length**）。同理，除了函数内定义的有限几个变量（如**i**、**j**等）外，没有额外的空间占用，空间复杂度为**O(1)**。

【**HeapAdjust**】**—— 堆的调整**

循环中**j**每次乘以**2** ，最多会循环到**m**，循环次数与完全二叉树的高度有关，时间复杂度为 **O(logm)**，这里**m**表示参与调整堆的元素个数范围。同理，只使用了常数个额外变量（如**rc**、**j**等），空间复杂度为**O(1)**。

【**HeapSort**】**—— 堆排序**

第一个循环用于构建初始堆，执行次数大约为**O(n)**（**n**为元素总个数），每次调用**HeapAdjust**时间复杂度为**O(logn)**；第二个循环执行**n-1**次，每次交换后又调用**HeapAdjust**，整体时间复杂度为**O(nlogn)**。同理，除了有限的几个函数内变量外，没有额外空间使用，空间复杂度为**O(1)**。

【**Partition**】**—— 划分序列**

函数中有两个嵌套的循环，在最坏情况下（如序列已经有序）需要遍历整个待划分区间，时间复杂度为**O(n)**（**n**为**high-low+1**表示区间内元素个数），平均情况时间复杂度也接近**O(n)**。同理，只使用了常数个额外变量（如**pivotkey**等），空间复杂度为**O(1)**。

【**QSort**】**—— 快速排序**

在最坏情况下（如每次划分都极度不平衡）时间复杂度为**O(n^2)**，平均情况下时间复杂度为**O(nlogn)**（**n**为待排序元素个数）。由于是递归调用，递归深度最坏情况为**O(n)**（对应划分极度不平衡时），平均情况为**O(logn)**，但函数内本身局部变量占用空间复杂度为**O(1)**，整体考虑空间复杂度最坏为**O(n)**，平均为**O(logn)**。

【**InsertSort**】**—— 直接插入排序**

有两层循环，外层循环执行**n-1**次（**n**为**L->length**），内层循环在最坏情况下（如逆序序列）需要移动较多元素，时间复杂度为**O(n^2)**，最好情况（如序列已经有序）时间复杂度为**O(n)**，平均情况时间复杂度为**O(n^2)**。只使用了常数个额外变量（如**i**、**j**等），空间复杂度为**O(1)**。

**综上所述：**

**最坏情况下，像SelectSort、QSort（最坏划分情况）、InsertSort（逆序输入）等函数可能达到O(n^2)的时间复杂度，其中n通常表示待排序元素个数，综合考虑这些函数在最坏组合使用场景下，整体时间复杂度可以达到O(n^2)。平均情况下，HeapSort、QSort（平均划分情况）等排序算法平均时间复杂度为O(nlogn)，所以整体平均时间复杂度为O(nlogn)。综合而言，整体时间复杂度在最坏情况下是O(n^2)，平均情况下是O(nlogn)。**

**这些函数大多只使用了有限的额外变量，本身局部空间复杂度多为O(1)，不过像 QSort因为递归调用，其空间复杂度取决于递归深度，最坏为O(n)（递归深度最大时）。平均为O(logn)。综合而言，整体空间复杂度在最坏情况下是O(n)，平均情况下是O(logn)。**

**七、实验总结**

1. **SelectMinKey**函数中，可以考虑优化每次找最小元素时的比较次数，比如使用类似**锦标赛排序**的思想，先分组比较选出每组最小，再逐步合并比较，减少不必要的重复比较，以此提升效率。

2. **InsertSort**函数中，可以考虑采用**二分查找**的方式来确定插入位置，减少逐个比较移动的次数，提高插入元素到合适位置的效率，改进后的排序也叫**二分插入排序**，能在一定程度上优化时间复杂度。

3. 快速排序中，可以考虑优化**Partition**函数，可以采用**随机选取基准元素**或者**三数取中**等策略，避免在输入数据有序等特殊情况下出现最坏时间复杂度的情况，使划分更均匀，提升整体性能。