**实验报告七**

1. **实验内容**

查找性能测试

1. **数据结构设计与思路分析**
2. **本题所采用的数据结构：**

顺序表+查找方法

1. **算法设计思路：**

（1）线性查找（顺序查找）

算法设计思路：

1.从列表的第一个元素开始，逐个检查每个元素，直到找到目标元素或者遍历完整个列表。

2.对于每个被检查的元素，与目标元素进行比较，如果相等则返回该元素的索引。

3.如果遍历完整个列表仍未找到目标元素，则返回不存在的标识。

时间复杂度分析：

最坏情况下：需要遍历整个列表来找到目标元素或确认其不存在。

时间复杂度：O(n)，其中n是列表的长度。

（2）非递归二分查找（适用于已排序的列表）

算法设计思路：

1、初始化左指针指向列表的起始位置，右指针指向列表的末尾。

2、计算中间元素的索引。

3、比较中间元素与目标元素的值：

如果中间元素等于目标元素，返回其索引。

如果中间元素大于目标元素，则将右指针移动到中间元素的左侧一位。

如果中间元素小于目标元素，则将左指针移动到中间元素的右侧一位。

重复步骤2和步骤3，直到找到目标元素或左指针大于右指针。

时间复杂度分析：

最坏情况下：每次都能将搜索范围减半，时间复杂度为O(log n)，其中n是列表的长度。

（3）递归二分查找（适用于已排序的列表）

算法设计思路：

1、定义一个递归函数，传入列表、左指针和右指针作为参数。

2、在函数内部，计算中间元素的索引。

3、比较中间元素与目标元素的值：

如果中间元素等于目标元素，返回其索引。

如果中间元素大于目标元素，则在左侧递归调用函数，更新右指针为中间元素的左侧一如果中间元素小于目标元素，则在右侧递归调用函数，更新左指针为中间元素的右侧一位。

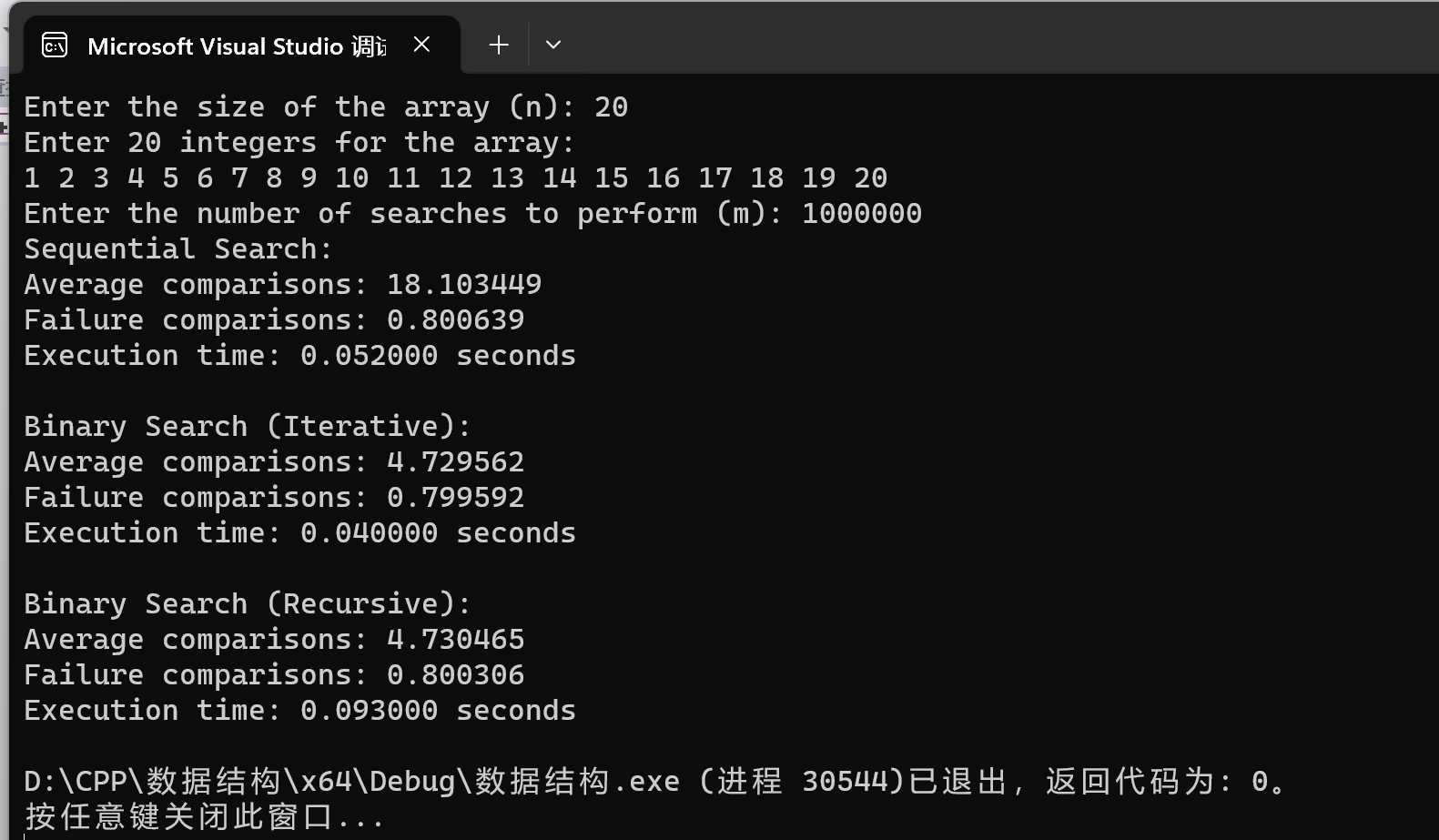
重复步骤2和步骤3，直到找到目标元素或左指针大于右指针。

时间复杂度分析：

最坏情况下：每次都能将搜索范围减半，时间复杂度为O(log n)，其中n是列表的长度。

空间复杂度：递归调用可能消耗额外的内存空间，因此可能会有O(log n)的空间复杂度。

1. **实验结果与分析**
2. **当顺序表为有序表时三种查找方式比较**

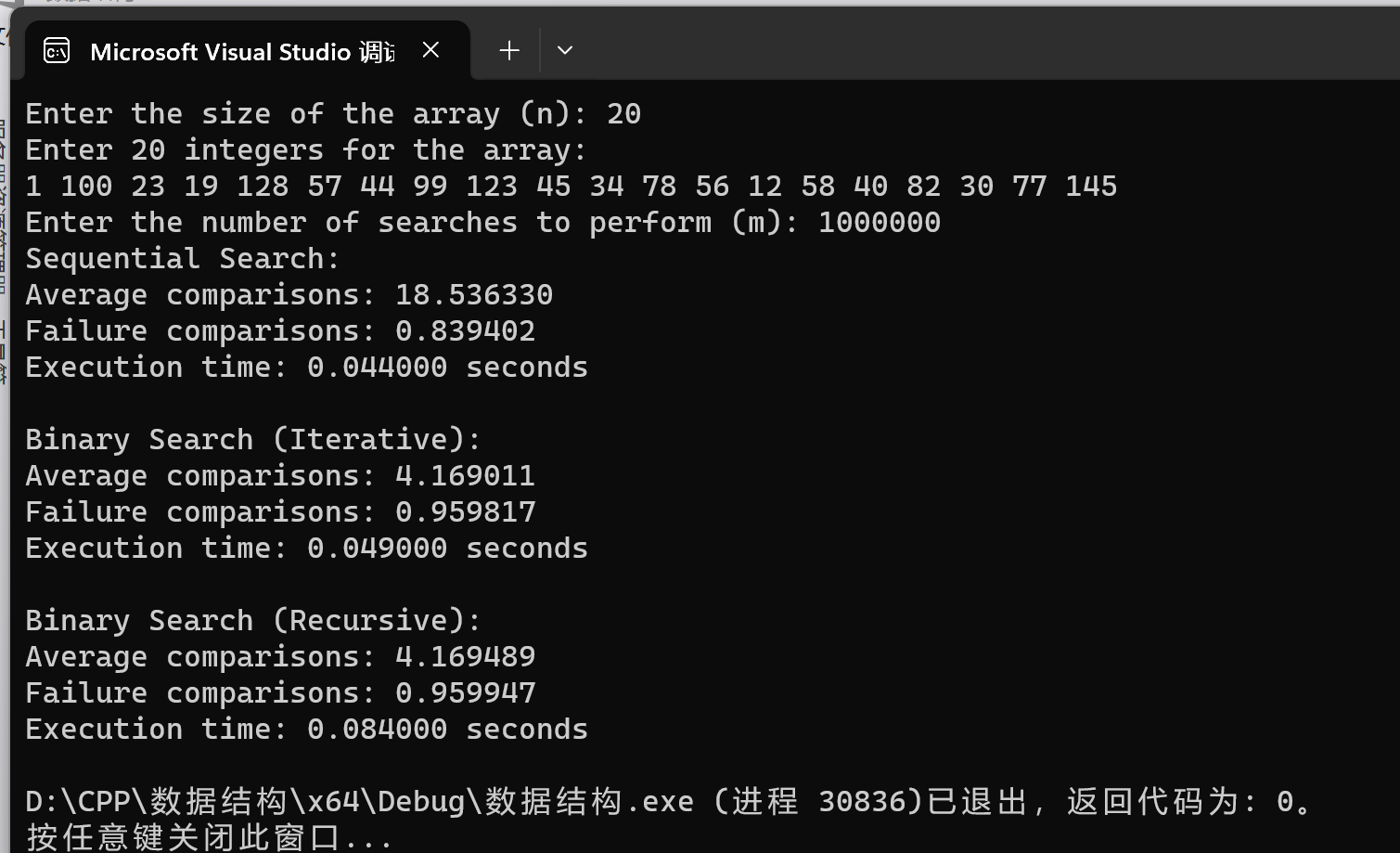


由上图结果可见：

1、在同一有序表中，二分查找的时间效率要高于顺序查找的时间效率。

2、非递归二分查找的时间效率与递归二分查找基本相等。但是如果数据量变得非常大，则非递归二分查找的优势将更加明显，因为递归调用会涉及到函数调用开销、堆栈操作等，需要很多额外的空间，而非递归版本是通过迭代进行查找，避免了这些开销。

**（2）当顺序表为无序表时三种查找方式比较**



比较上面两张图，可以发现：

在有序表中，虽然绝对时间和平均比较次数是线性的，但查找成功时的平均比较次数要比无序表少，这是因为有序性使得在查找成功时可以提前终止。在时间性能上，二者差不多。

总体而言，无序表的顺序查找相对于有序表来说效率较低，因为有序表可以利用有序性进行更有效的搜索。在实际应用中，如果搜索操作频繁，有序表可能更为适合。

1. **实验小结**

通过本次实验，我加深了对三种查找方式的理解，并能通过编写程序进行三种查找方式在有序表与无序表两种不同情况下的性能比较，提高了数据结构的知识掌握程度，同时提高了编程能力。

**实验还可以完善的点**：可以通过随机种子生成大量的随机数据来进行性能测试，使得结果更具准确性，在下一次排序实验中我们会采用此种方法。