

# 重 庆 大 学

## 学 生 实 验 报 告

实验课程名称 数据结构与算法

开课实验室 DS1501

学 院 软件学院 年级 2021 专业班 软件 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2022 至 2023 学年第 1 学期

总 成 绩	
教师签名	XXX

# 《数据结构与算法》实验报告

开课实验室：DS1501

2022 年 11 月 10 日

学院	软件学院	年级、专业、班	2021 级软件工 程 X 班	姓名	XXX	成绩	
课程 名称	数据结构与算法	实验项目 名 称	2022-2023 学年第一学期数 据结构与算法上机练习 004	指导教师	XXX		
教 师 评 语	教师签名： 年 月 日						

## 一、实验目的

- 请完成教材第 7 章 153 页至 155 页的归并排序算法，图 7.9 和 7.10 任选一种实现，并撰写实验报告，附上算法关键代码以及运行代价分析，同时请对算法的稳定性进行分析。
- 请实现第 7 章 155 页至 157 页的快速排序算法，并撰写实验报告，附上算法关键代码以及运行代价分析，同时请对算法的稳定性进行分析。

## 二、使用仪器、材料

PC 微机；  
Windows 操作系统，VS2022 编译环境；

## 三、实验步骤

- 1、完成教材第 7 章 153 页至 155 页的归并排序算法，图 7.9 和 7.10 任选一种实现，并撰写实验报告，附上算法关键代码以及运行代价分析，同时请对算法的稳定性进行分析；
- 2、实现第 7 章 155 页至 157 页的快速排序算法，并撰写实验报告，附上算法关键代码以及运行代价分析，同时请对算法的稳定性进行分析。

#### 四、实验过程原始记录(数据、图表、计算等)

1、完成教材第7章 153 页至 155 页的归并排序算法,图 7.9 和 7.10 任选一种实现,并撰写实验报告,附上算法关键代码以及运行代价分析,同时请对算法的稳定性进行分析;

(1) 关键代码:

```
//书P154 图7.9归并排序的标准实现
template<typename E> <T> 提供 IntelliSense 的示例模板参数
void mergesort(E A[], E temp[], int left, int right) {
    if (left == right) { return; }
    int mid = (left + right) / 2;
    mergesort<E>(A, temp, left, mid);
    mergesort<E>(A, temp, mid + 1, right);
    for (int i = left; i <= right; i++) {
        temp[i] = A[i];
    }
    int i1 = left;
    int i2 = mid + 1;
    for (int curr = left; curr <= right; curr++) {
        if (i1 == mid + 1) {
            A[curr] = temp[i2++];
        }
        else if (i2 > right) {
            A[curr] = temp[i1++];
        }
        else if (temp[i1] < temp[i2]) {
            A[curr] = temp[i1++];
        }
        else {
            A[curr] = temp[i2++];
        }
    }
}
```

(2) 验证程序:

```

int main()
{
    int n;
    int* list;
    list = new int();
    cout << "请输入数组长度: " << endl;
    cin >> n;
    cout << "请输入数组元素: " << endl;
    Input(list, n);
    cout << "归并排序前数组为: " << endl;
    Output(list, n);
    cout << endl;
    int temp[] = { 0 };
    mergesort(list, temp, 0, n-1);
    cout << "归并排序后数组为: " << endl;
    Output(list, n);
    cout << endl;
    return 0;
}

```

(3) 结果:



```

Microsoft Visual Studio 调试
请输入数组长度:
7
请输入数组元素:
20 22 11 17 17 34 12
归并排序前数组为:
20 22 11 17 17 34 12
归并排序后数组为:
34 22 20 17 17 12 11

```

(4) 运行代价分析:

归并算法的实现是一个递归程序, 设  $i$  为两个要归并子数组的总长度, 归并过程需要花费时间为  $\Theta(i)$ , 设要被排序的元素的数目为  $n$ ;

在归并排序的过程中, 需要对当前区间进行对半划分, 直到区间的长度为 1。也就是说, 每一层的子区间, 长度都是上一层的  $1/2$ , 而当划分到第  $\log n$  层的时候, 子区间的长度就是 1 了。所以递归的深度为  $\log n$ ;

第一层递归可以认为是对一个长度为  $n$  的数组的排序, 下一层是对 2 个长度为  $n/2$  的子数组的排序, 再下一层是对 4 个长度为  $n/4$  的子数组的排序, 以此类推, 最后一层是对  $n$  个长度为 1 的子数组的排序。在对  $n$  个长度为 1 的子数组归并时, 时间复杂度为  $\Theta(n)$ , 在对  $n/2$  个长度为 2 的子数组归并时, 时间复杂度为  $\Theta(n)$ , 以此类推, 每一层的时间复杂度均为  $\Theta(n)$ 。因此, 总的时间复杂度为  $\Theta(n \log n)$ ;

综上所述, 归并排序算法的时间复杂度为  $\Theta(n \log n)$ , 且这个时间复杂度是稳定的, 不随需要排序的序列不同而产生波动。

(5) 稳定性分析:

归并算法中, 在合并的时候, 如果两个数相等, 可以将前面的数先放在  $temp$  数组中, 这样就保证了值相同的元素, 在合并前后的先后顺序不变。

所以, 归并排序算法是一个稳定的排序算法。

2、实现第 7 章 155 页至 157 页的快速排序算法，并撰写实验报告，附上算法关键代码以及运行代价分析，同时请对算法的稳定性进行分析。

(1) 关键代码：

```
template <typename E>
inline void swap(E A[], int i, int j) {
    E temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}

template <typename E>
inline int findpivot(E A[], int i, int j) {
    return (i + j) / 2;
}

template <typename E>
inline int partition(E A[], int l, int r, E& pivot) {
    do {
        while (A[++l] > pivot);
        while ((l < r) && (pivot > A[--r]));
        swap(A, l, r);
    } while (l < r);
    return l;
}

template <typename E>
void quicksort(E A[], int i, int j) {
    if (j <= i) return;
    int pivotindex = findpivot(A, i, j);
    swap(A, pivotindex, j);
    int k = partition<E>(A, i - 1, j, A[j]);
    swap(A, k, j);
    quicksort<E>(A, i, k - 1);
    quicksort<E>(A, k + 1, j);
}
```

(2) 验证程序：

```

int main()
{
    int n;
    int* list;
    list = new int();
    cout << "请输入数组长度: " << endl;
    cin >> n;
    cout << "请输入数组元素: " << endl;
    Input(list, n);
    cout << "快速排序前数组为: " << endl;
    Output(list, n);
    cout << endl;
    int temp[] = { 0 };
    quicksort(list, 0, n - 1);
    cout << "快速排序后数组为: " << endl;
    Output(list, n);
    cout << endl;
    return 0;
}

```

(3) 结果:



```

Microsoft Visual Studio 调试
请输入数组长度:
7
请输入数组元素:
20 22 11 17 18 36 5
快速排序前数组为:
20 22 11 17 18 36 5
快速排序后数组为:
36 22 20 18 17 11 5

```

(4) 运行代价分析:

最好的情况下，我们选取的轴刚好就是这个区间的中位数。也就是说，在操作之后，正好将区间分成了数字个数相等的左右两个子区间。此时就和归并排序基本一致了，每一层的总时间复杂度都是  $\Theta(n)$ ，因为需要对每一个元素遍历一次，在最好的情况下，同样也是有  $\log n$  层，所以快速排序最好的时间复杂度为  $\Theta(n \log n)$ ；

最坏的情况下，对于每一个区间，我们在处理的时候，选取的轴刚好就是这个区间的最大值或者最小值。对于  $n$  个数来说，需要操作  $n$  次，才能为  $n$  个数排好序。而每一次操作都需要遍历一次剩下的所有元素，这个操作的时间复杂度是  $\Theta(n)$ ，所以总时间复杂度为  $\Theta(n^2)$ ；

平均情况下，时间复杂度为  $\Theta(n \log n)$ ；

综上所述，快速排序算法的时间复杂度是  $\Theta(n \log n)$ 。

(5) 稳定性分析:

快速排序中，两个值相同的元素在排序前后位置仍有变化，即使待排序元素可基数相等也需要移动待排序元素的位置使得有序，所以快速排序算法是不稳定的。

## 五、实验结果及分析

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。