重庆师范大学

实验报告

实验课程名称	算法设计与分析
实验序号	5
实验内容	快速排序三种方法
班级	23 计科 6 班
姓名	周子依
学号	2023051603162

2024 年 3 月 28 日

实验目的与要求

1. 学习目标

掌握快速排序数组划分的三种方法，分别运用以下的三种方法对待排序数组进行排序。

从两边往中间和主元对比，当右边有小的，左边有大的，停止搜索，两者交换；重复过程直到搜索过程相遇。(Partition1)

从右往左搜索，当有比主元小的，停止搜索并和主元交换；
从左往右搜索，当有比主元大的，停止搜索并和主元交换；
重复过程直到搜索过程相遇。(Partition2)

以最后一个数为主元，同时从左往右搜索，比主元小的放前面，比主元大的放后面，最后把主元放两者之间。(Partition3)

待排序数组：

13	8	17	10	15	18	12	20	9	14	6	19

实验内容

1. 算法原理与设计思路

（1）数组划分法一

从两边往中间和主元对比，j 从最右边开始，i 从主元后一个数开始扫描，当;扫描到有大于主元的数，j 扫描到有小于主元的数(这两个条件需同时满足)，交换 i 和 j 指向的数，直到 i≥j，最后交换主元与 j 指向的数。

下图 1 笔记对课上给出的例子进行了该方法的演示。

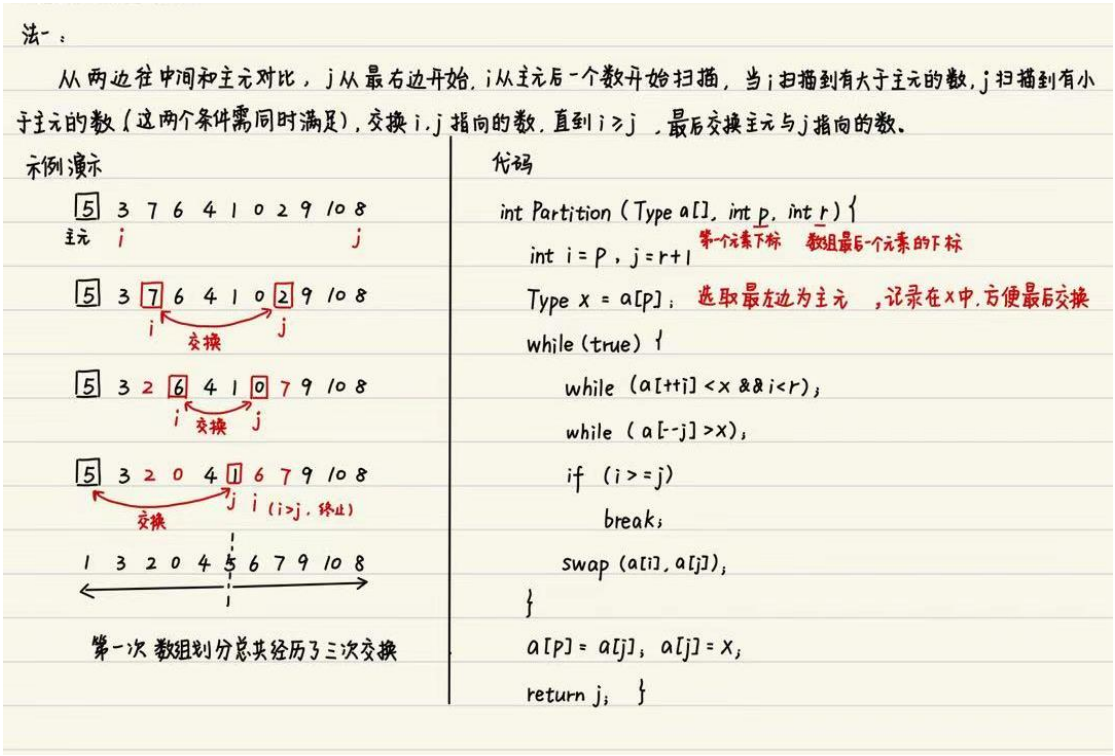

图 1 数组划分法一笔记

法一数组划分的代码如图 2：

```
int Partiton(Type a[], int p, int r) {
    // 初始化左指针 i 为起始索引 p
    int i = p;
    // 初始化右指针 j 为结束索引 r 加 1，方便后续操作
    int j = r + 1;
    // 选择数组的第一个元素作为基准元素
    Type x = a[p];

    // 进入循环，不断交换元素，直到左右指针相遇或交叉
    while (true) {
        // 左指针 i 向右移动，找到第一个大于等于基准元素 x 的元素
        // 同时要保证 i 不超过结束索引 r
        while (a[++i] < x && i < r);
        // 右指针 j 向左移动，找到第一个小于等于基准元素 x 的元素
        while (a[--j] > x);
        // 如果左指针 i 已经大于等于右指针 j，说明划分完成，退出循环
        if (i >= j)
            break;
        // 交换左指针 i 和右指针 j 所指向的元素
        swap(a[i], a[j]);
    }

    // 将基准元素放到正确的位置，即右指针 j 所指向的位置
    a[p] = a[j];
    a[j] = x;
    // 返回基准元素最终所在的位置
    return j;
}
```

图 2 数组划分法一代码

（2）数组划分法二

i 初始指向主元的后一个位置，j 初始指向最后一个位置，首先 j 向前移动，搜索到小于主元的数，将 j 指向的数与主元交换，i 再向后移动，搜索到大于主元的数，将指向的数与主元交换，直到 i 与 j 相遇。

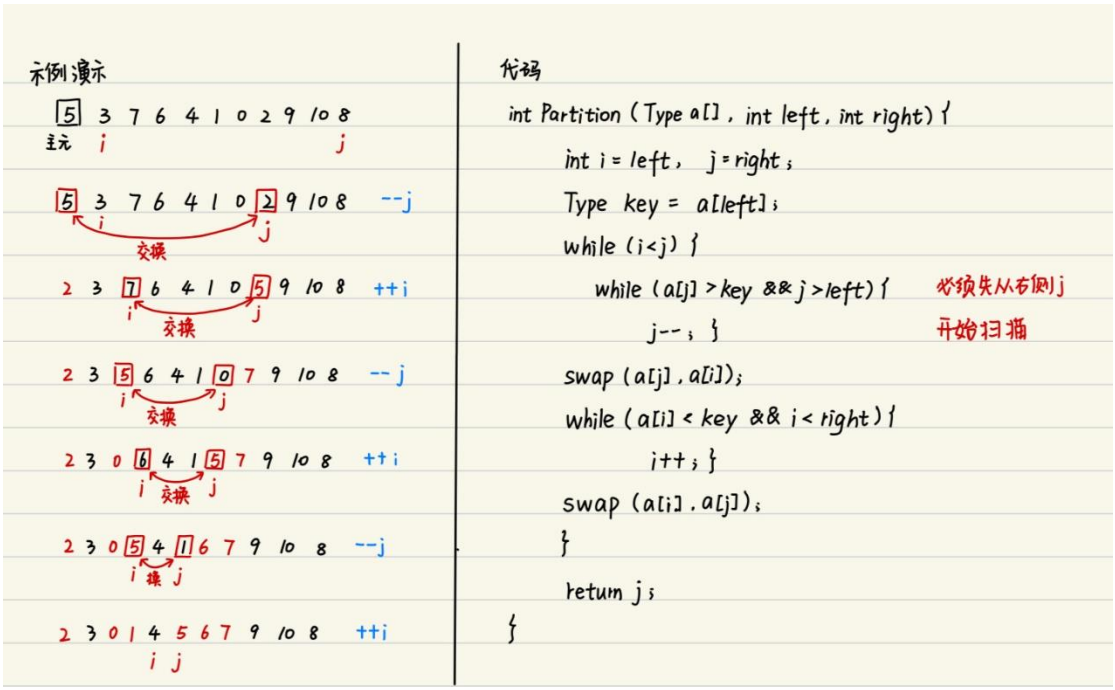

图 3 数组划分法二笔记

法一数组划分的代码如图 4：

```
// 分区函数，将数组分为两部分
template<typename Type>
int QuickSort2(Type a[], int left, int right) {
    int i = left, j = right;
    Type key = a[left];
    while (i < j) {
        // 从右向左找第一个小于等于 key 的元素
        while (a[j] > key && j > left) {
            j--;
        }
        swap(a[j], a[i]);
        // 从左向右找第一个大于等于 key 的元素
        while (a[i] < key && i < right) {
            i++;
        }
        swap(a[i], a[j]);
    }
    return j;
}
```

图 4 数组划分法二代码

（3）数组划分法三

以最后一个数为主元,初始 i ， j 均指向第一个元素， j 向右移动直到搜索到小于主元的数.交换 i 指向的后一个数和 j 指向的数,此时 i 后移一个位置，直到 j 扫描到主元的位置,交换 i 指向的后一个数和主元。

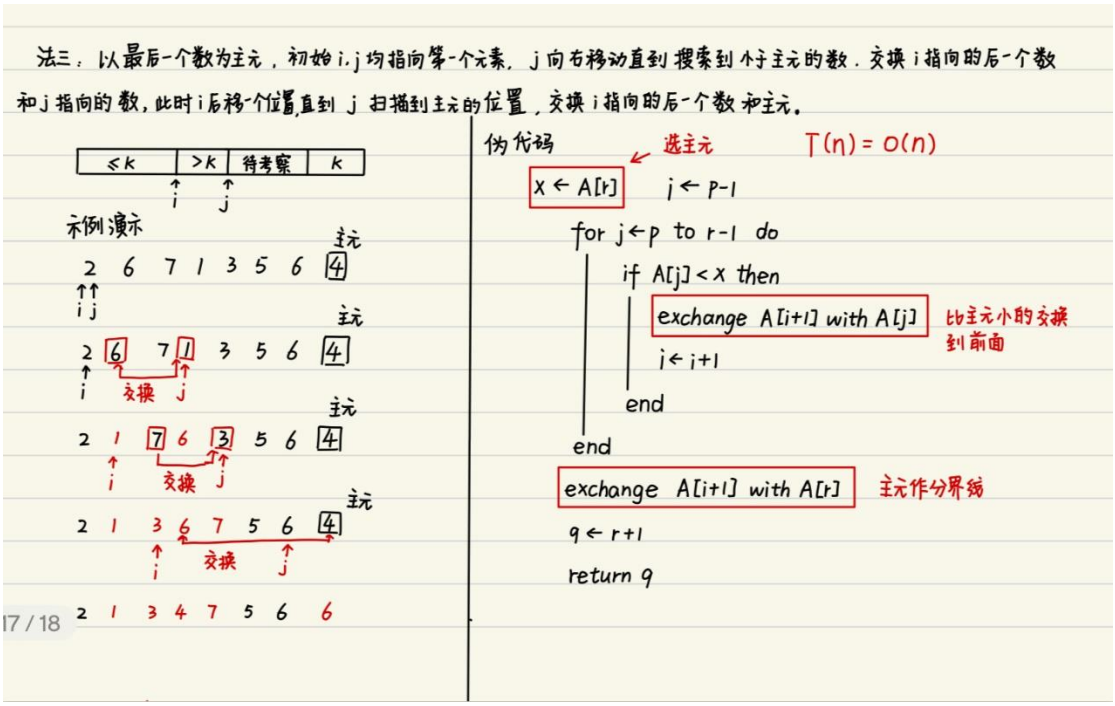

图 5 数组划分法二笔记

法三数组划分的代码如图 6。

```
11.  // 分区模板函数，以最后一个元素作为主元
12.  template<typename Type>
13.  int partition(Type A[], int p, int r) {
14.      // 选取数组最后一个元素作为主元
15.      Type x = A[r];
16.      // i 初始化为 p - 1，用于标记小于主元的元素区域的右边界
17.      int i = p - 1;
18.
19.      // 遍历从 p 到 r - 1 的元素
20.      for (int j = p; j < r; ++j) {
21.          // 如果当前元素 A[j] 小于主元 x
22.          if (A[j] < x) {
23.              // i 右移一位，扩大小于主元的元素区域
24.              ++i;
25.              // 交换 A[i] 和 A[j]，将小于主元的元素放到左边
26.              swap(A[i], A[j]);
27.          }
28.      }
29.      // 遍历结束后，将主元放到正确的位置，即 i + 1 处
30.      // 此时 A[i + 1] 到 A[r - 1] 的元素都大于等于主元
31.      swap(A[i + 1], A[r]);
32.      // 返回主元最终所在的位置
33.      return i + 1;
34.  }
```

图 6 数组划分法三代码

实验问题分析

1. 复杂度分析

最好情况，每次数组划分后，主元都是中间大的数字。最坏情况，每次数组划分后，主元都在一侧，如下图 7 分析。

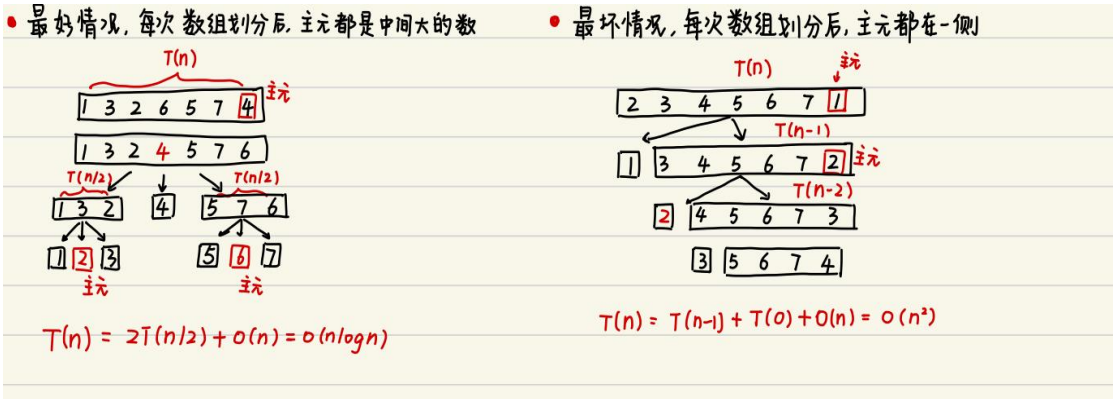

图 7 时间复杂度分析

2. 最坏情况的改进

解决最坏情况，采用随机选取主元的方法，如图 8，改进数组划分的代码。

```
int RandomizedPartition(Type a[], int p, int r) {
    srand(static_cast<unsigned int>(time(nullptr)));
    int i = p + rand() % (r - p + 1);
    swap(a[i], a[r]);

    Type x = a[r];
    int j = p - 1;
    for (int k = p; k < r; k++) {
        if (a[k] <= x) {
            j++;
            swap(a[j], a[k]);
        }
    }
    swap(a[j + 1], a[r]);
    return j + 1;
}
```

图 8 随机选取主元划分

总结与体会

本次实验围绕快速排序的三种数组划分方法展开，收获颇丰。在原理学习上，深入理解了不同划分策略，法一从两边向中间对比主元，法二通过指针交替移动交换元素，法三以最后元素为主元移动指针划分。实践中，亲手编写代码实现，遇到指针操作、边界条件处理等问题，经调试解决，强化了对代码逻辑的把控。复杂度分析让我明晰算法性能，随机选取主元改进策略有效避免最坏情况。此次实验提升了我对算法设计与分析的认知，锻炼了编码和问题解决能力，也让我明白理论与实践结合的重要性，后续会不断深化学习，提升专业素养。

附完整 C++代码

（法三使用随机选取主元的数组划分方法）

法一：

```
#include <iostream>
using namespace std;

typedef int Type;

// 该函数用于对数组进行划分，返回基准元素最终所在的位置
// 参数 a 是待划分的数组，p 是数组的起始索引，r 是数组的结束索引
int Partiton(Type a[], int p, int r) {
    // 初始化左指针 i 为起始索引 p
    int i = p;
    // 初始化右指针 j 为结束索引 r 加 1，方便后续操作
    int j = r + 1;
    // 选择数组的第一个元素作为基准元素
    Type x = a[p];

```

```
16.     // 进入循环，不断交换元素，直到左右指针相遇或交叉
17.     while (true) {
18.         // 左指针 i 向右移动，找到第一个大于等于基准元素 x 的元素
19.         // 同时要保证 i 不超过结束索引 r
20.         while (a[++i] < x && i < r);
21.         // 右指针 j 向左移动，找到第一个小于等于基准元素 x 的元素
22.         while (a[--j] > x);
23.         // 如果左指针 i 已经大于等于右指针 j，说明划分完成，退出循环
24.         if (i >= j)
25.             break;
26.         // 交换左指针 i 和右指针 j 所指向的元素
27.         swap(a[i], a[j]);
28.     }
29.
30.     // 将基准元素放到正确的位置，即右指针 j 所指向的位置
31.     a[p] = a[j];
32.     a[j] = x;
33.     // 返回基准元素最终所在的位置
34.     return j;
35. }
36.
37. // 该函数用于实现快速排序算法
38. // 参数 a 是待排序的数组，p 是数组的起始索引，r 是数组的结束索引
39. void QuickSort(Type a[], int p, int r) {
40.     // 如果起始索引 p 小于结束索引 r，说明数组中至少有两个元素，需要继续排序
41.     if (p < r) {
42.         // 调用 Partiton 函数对数组进行划分，得到基准元素的位置 q
43.         int q = Partiton(a, p, r);
44.         // 对基准元素左边的子数组进行快速排序
45.         QuickSort(a, p, q - 1);
46.         // 对基准元素右边的子数组进行快速排序
47.         QuickSort(a, q + 1, r);
48.     }
49. }
50.
51. int main() {
52.     Type arr[] = {13, 8, 17, 10, 15, 18, 12, 20, 9, 14, 6, 19};
53.     // 计算数组的长度
54.     int n = sizeof(arr) / sizeof(arr[0]);
55.     // 调用 QuickSort 函数对数组进行排序
56.     QuickSort(arr, 0, n - 1);
57.     // 遍历排序后的数组，并输出每个元素
```

```
58.         for (int i = 0; i < n; i++) {
59.             cout << arr[i] << " ";
60.         }
61.         cout << endl;
62.         return 0;
63. }
```

法二：

```
1.  #include <iostream>
2.
3.  // 交换两个元素的函数
4.  template<typename Type>
5.  void swap(Type& a, Type& b) {
6.      Type temp = a;
7.      a = b;
8.      b = temp;
9.  }
10.
11. // 分区函数，将数组分为两部分
12. template<typename Type>
13. int QuickSort2(Type a[], int left, int right) {
14.     int i = left, j = right;
15.     Type key = a[left];
16.     while (i < j) {
17.         // 从右向左找第一个小于等于 key 的元素
18.         while (a[j] > key && j > left) {
19.             j--;
20.         }
21.         swap(a[j], a[i]);
22.         // 从左向右找第一个大于等于 key 的元素
23.         while (a[i] < key && i < right) {
24.             i++;
25.         }
26.         swap(a[i], a[j]);
27.     }
28.     return j;
29. }
30.
31. // 递归实现快速排序
32. template<typename Type>
```

```
33. void recursiveQuickSort(Type a[], int left, int right) {
34.     if (left < right) {
35.         int key = QuickSort2(a, left, right);
36.         recursiveQuickSort(a, left, key - 1);
37.         recursiveQuickSort(a, key + 1, right);
38.     }
39. }
40.
41. using namespace std;
42. int main() {
43.     // 测试数组
44.     int arr[] = {13, 8, 17, 10, 15, 18, 12, 20, 9, 14, 6, 19};
45.     int n = sizeof(arr) / sizeof(arr[0]);
46.
47.     // 调用快速排序函数
48.     recursiveQuickSort(arr, 0, n - 1);
49.
50.     // 输出排序后的数组
51.     for (int i = 0; i < n; i++) {
52.         cout << arr[i] << " ";
53.     }
54.     cout << endl;
55.
56.     return 0;
57. }
```

法三：

```
1. #include <iostream>
2. #include <cstdlib>
3. #include <ctime>
4.
5. // 交换函数
6. template<typename Type>
7. void swap(Type& a, Type& b) {
8.     Type temp = a;
9.     a = b;
10.    b = temp;
11. }
12.
13. // 随机化分区函数
14. template<typename Type>
```

```
15. int RandomizedPartition(Type a[], int p, int r) {
16.     srand(static_cast<unsigned int>(time(nullptr)));
17.     int i = p + rand() % (r - p + 1);
18.     swap(a[i], a[r]);
19.
20.     Type x = a[r];
21.     int j = p - 1;
22.     for (int k = p; k < r; k++) {
23.         if (a[k] <= x) {
24.             j++;
25.             swap(a[j], a[k]);
26.         }
27.     }
28.     swap(a[j + 1], a[r]);
29.     return j + 1;
30. }
31.
32. // 随机化快速排序函数
33. template<typename Type>
34. void RandomizedQuickSort(Type a[], int p, int r) {
35.     if (p < r) {
36.         int q = RandomizedPartition(a, p, r);
37.         RandomizedQuickSort(a, p, q - 1);
38.         RandomizedQuickSort(a, q + 1, r);
39.     }
40. }
41.
42. using namespace std;
43.
44. int main() {
45.     int arr[] = {13, 8, 17, 10, 15, 18, 12, 20, 9, 14, 6, 19};
46.     int n = sizeof(arr) / sizeof(arr[0]);
47.
48.     RandomizedQuickSort(arr, 0, n - 1);
49.
50.     for (int i = 0; i < n; i++) {
51.         cout << arr[i] << " ";
52.     }
53.     cout << endl;
54.
55.     return 0;
56. }
```