重 庆 大 学

学 生 实 验 报 告

实验课程名称 算法设计与分析

开课实验室 DS1501

学 院 大数据与软件学院 年级 2021 专业班软件工程 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2022 至 2023 学年第 二 学期

总 成 绩	
教师签名	XXX

大数据与软件学院制

《算法设计与分析》实验报告

开课实验室：DS1501 **2023 年 5 月 16 日**

学院	大数据与软件学院	年级、专业、班	2021 级软件工程 X 班	姓名	XXX	成绩	
课程名称	算法设计与分析	实验项目名　　称	分治法实验	指导教师	XXX		
教师评语	教师签名： 年　月　日						

一、 实验目的

- 掌握分治法的设计思想，包括分治法解决的问题特征、分治法的求解过程。
- 熟练掌握“众数问题算法”的实现，熟练掌握“一个整数数组划分为两个子数组问题算法”的实现，提供输入案例检测算法的正确性。
- 学会分析上述分治类算法的时间复杂性。
- 主要任务：实现教材配套实验指导书“第 2 章 2.3.2 小节 求解众数问题算法”和“第 2 章 2.3.5 小节 求解一个整数数组划分为两个子数组问题算法”。

二、使用仪器、材料

PC 微机 Lenovo Legion R9000P2021H；
Windows11 操作系统;
Clion2023 编译环境；

三、实验步骤

2.3.2 实验 2 求解众数问题

给定一个整数序列,每个元素出现的次数称为重数,重数最大的元素称为众数。编写一个实验程序对递增有序序列 a 求众数。例如 S={1,2,2,2,3,5},多重集 S 的众数是 2,其重数为 3。

解:

求众数的方法有多种,这里采用分治法求众数。

用全局变量 mode 和 maxcnt 分别存放 a 的众数和重数(maxcnt 的初始值为 0)。对于至少含有一个元素的序列 a[low..high],以中间位置 mid 为界限，求出 a[mid]元素的重数 cnt,即 a[left..right]均为 a[mid],cnt=right-left+l,若 cnt 大于 maxcnt,置 mode=a[mid],maxcnt=cnt。然后对左序列 a[low..left-1]和右序列 a[right+1..high]递归求解众数,如图 2.14 所示。

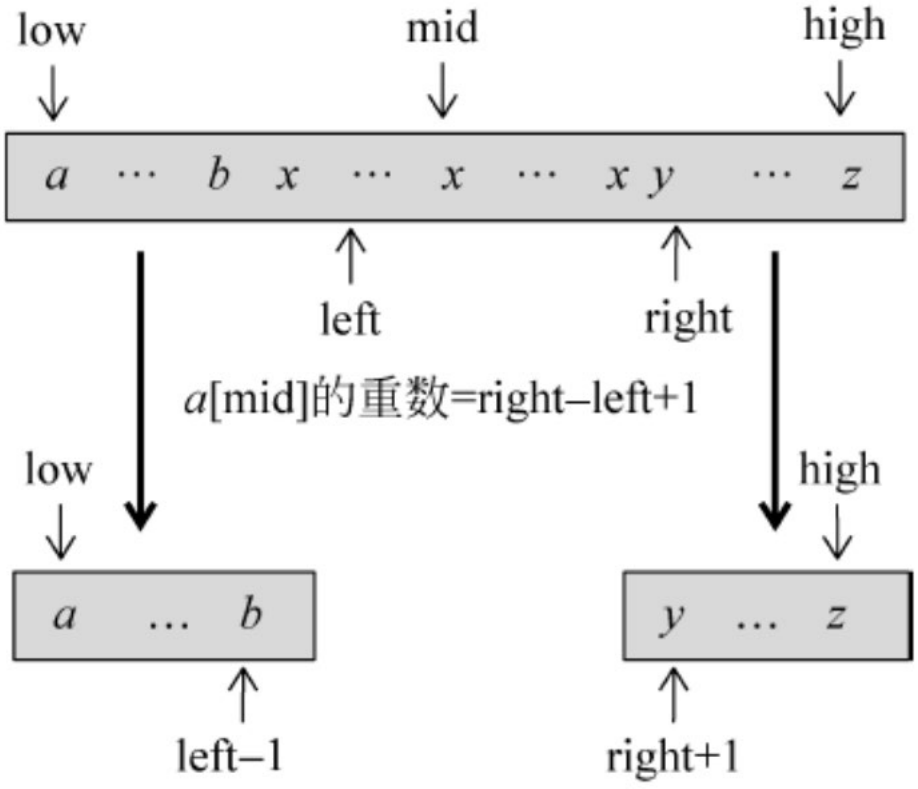


图 2.14　求众数的过程

2.3.5 实验 5 求解一个整数数组划分为两个子数组问题

已知由 n(n≥2)个正整数构成的集合 A={ar}(0≤k<n),将其划分为两个不相交的子集 A1 和 A2,元素个数分别是 n1 和 n2,A1 和 A2 中的元素之和分别为 S1 和 S2。设计一个尽可能高效的划分算法,满足|n1-n2|最小且|S1-S2|最大,算法返回|S1-S2|的结果。

解：

将 A 中最小的⌊n/2⌋个元素放在 A1 中,其他元素放在 A2 中,即得到题目要求的结果。采用递归快速排序思路,查找第 n/2 小的元素,前半部分为 A1 的元素,后半部分为 A2 的元素,这样算法的时间复杂度为 O(n)。如果将 A 中元素全部排序，再进行划分,时间复杂度为 O(nlog2n),不如前面的方法。

四、实验过程原始记录(数据、图表、计算等)

2.3.2 实验 2 求解众数问题

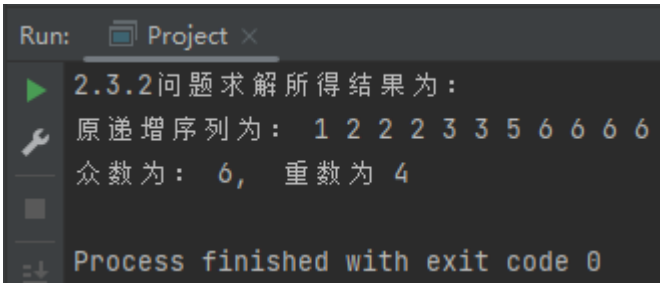
给定一个整数序列,每个元素出现的次数称为重数,重数最大的元素称为众数。编写一个实验程序对递增有序序列 a 求众数。例如 S={1,2,2,2,3,5},多重集 S 的众数是 2,其重数为 3。

源代码：

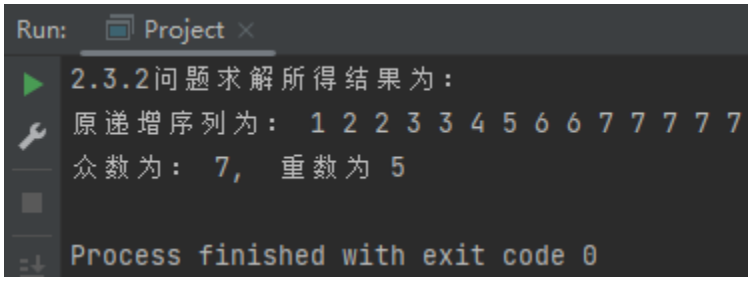
```
//
// Created by 小明同学 on 2023/5/28.
//
#include<iostream>

using namespace std;

int mode;   //保存众数
int maxcnt = 0;   //保存众数出现的次数

// 划分数组为[left, right]和[right+1, end]两段
void divide(const int a[], const int start, const int end, int& mid, int& left, int& right) {
    mid = (start + end) / 2;
    for (left = start; left <= end; left++) {
        if (a[left] == a[mid]) {    //查找左边界
            break;
        }
    }

    for (right = left + 1; right <= end; right++) {
        if (a[right] != a[mid]) {   //查找右边界
            break;
        }
    }

    right--;    //right指向重复元素序列的最后一个位置
}
```

```
//递归函数，求解众数和重数
void Obtainmaxcnt(int a[], int start, int end) {
    if (start <= end) {
        int mid, left, right;
        divide(a, start, end, &: mid, &: left, &: right);  //将数组划分为三部分
        int cnt;
        cnt = right - left + 1;     //计算出重数（众数出现的次数）
        if (cnt > maxcnt) {         //如果当前的重数大于之前保存的最大重数，则更新众数和最大重数
            mode = a[mid];
            maxcnt = cnt;
        }
        Obtainmaxcnt(a, start, end: left - 1);   //递归求解左半部分
        Obtainmaxcnt(a, start: right + 1, end);    //递归求解右半部分
    }
}

int main() {
    int a[] = { [0]: 1, [1]: 2, [2]: 2, [3]: 2, [4]: 3, [5]: 3, [6]: 5, [7]: 6, [8]: 6, [9]: 6, [10]: 6};  //原数组
    int a_length = sizeof(a) / sizeof(*a);    //计算数组的长度
    cout << "2.3.2问题求解所得结果为: " << endl;
    cout << "原递增序列为:  ";
    for (int i : a) {     //输出原数组
        cout << i << " ";
    }
    cout << endl;
    Obtainmaxcnt(a, start: 0, end: a_length-1);   //求解众数和重数
    cout << "众数为:  " << mode << ",  重数为 " << maxcnt << endl;   //输出结果
    return 0;
}
```

运行结果：
（1）a[] = {1,2,2,2,3,3,5,6,6,6,6};



（2）a[] = {1,2,2,3,3,4,5,6,6,7,7,7,7,7};



时间复杂度分析：

该算法的时间复杂度为 O(nlogn)。

在该算法中，每次递归都会将数组划分为三部分，这一步需要进行线性的扫描操作，因此其时间复杂度为 O(n)，其中 n 是当前序列中元素的个数。接着，该算法会对左半部分和右半部分进行递归求解。由于每次递归都会将序列的长度减半，因此总共需要进行 logn 次递归。所以该算法的时间复杂度为 O(nlogn)。

2.3.5 实验 5 求解一个整数数组划分为两个子数组问题

已知由 n(n≥2)个正整数构成的集合 A={ar}(0≤k<n),将其划分为两个不相交的子集 A1 和 A2,元素个数分别是 n1 和 n2,A1 和 A2 中的元素之和分别为 S1 和 S2。设计一个尽可能高效的划分算法,满足|n1-n2|最小且|S1-S2|最大,算法返回|S1-S2|的结果。

源代码：

```
#include<iostream>

using namespace std;

// 划分函数，返回轴值在数组 a 中的位置
int divide(int a[], int low, int high) {

    // 取第一个元素作为轴值
    int povit = a[low];
    int i = low, j = high; // i 表示从左边开始遍历的位置，j 表示从右边开始遍历的位置
```

```
    while (i < j) {
        // 从右边开始找到第一个小于等于轴值的元素
        while (i < j && a[j] >= povit)
            j--;
        a[i] = a[j]; // 将该元素放到左边（a[i] 处），a[j] 右边就空出来了

        // 从左边开始找到第一个大于等于轴值的元素
        while (i < j && a[i] <= povit)
            i++;
        a[j] = a[i]; // 将该元素放到右边（a[j] 处），a[i] 左边就空出来了
    }

    // 将轴值放回中间
    a[i] = povit;
    return i;
}
// 划分函数的封装，用于寻找数组 a 的中位数
int resolve(int a[], int n) {
    int low = 0, high = n - 1;
    bool flag = true;

    // 在满足条件的情况下不断划分数组
    while (flag)
    {
        int i = divide(a, low, high);
        if (i == n / 2 - 1) // 已经找到了中位数
            flag = false;
        else if (i < n / 2 - 1) // i 在左半部分，需要在右半部分查找
            low = i + 1;
        else if (i > n / 2 - 1) // i 在右半部分，需要在左半部分查找
            high = i - 1;
    }

    // 计算右半部分元素之和与左半部分元素之和的差
    int S1 = 0, S2 = 0;
    for (int i = 0; i < n / 2; i++)
        S1 = S1 + a[i];
    for (int j = n / 2; j < n; j++) {
        S2 = S2 + a[j];
    }
    return S2 - S1;
}
```

```cpp
int main() {
    cout<<"2.5.5问题求解所得结果为："<<endl;

    //第一个测试样例A
    int a[] = { [0]: 1, [1]: 3, [2]: 5, [3]: 7, [4]: 9, [5]: 2, [6]: 4, [7]: 6, [8]: 8};

    // 计算数组长度
    int a_length = sizeof(a) / sizeof(*a);

    // 输出初始序列
    cout << "初始序列A为：";
    for (int item : a) {
        cout << item << " ";
    }
    cout << endl;

    // 使用划分函数进行划分，并计算右半部分元素之和与左半部分元素之和的差
    int diff_a = resolve(a, n: a_length);

    // 输出划分结果及其差
    cout << "划分结果：" << endl;
    cout << "A1为：";
    for (int i = 0; i < a_length/2; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    cout << "A2为：";
    for (int i = a_length/2; i < a_length; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
    cout <<"S1与S2的差值为：  " <<diff_a<<endl<<endl;
```

```
    //第二个测试样例B
    int b[] = {  [0]: 1, [1]: 3, [2]: 5, [3]: 7, [4]: 9, [5]: 10, [6]: 2, [7]: 4, [8]: 6, [9]: 8};

    // 计算数组长度
    int b_length = sizeof(b) / sizeof(*b);

    // 输出初始序列
    cout << "初始序列B为: ";
    for (int item : b) {
        cout << item << " ";
    }
    cout << endl;

    // 使用划分函数进行划分，并计算右半部分元素之和与左半部分元素之和的差
    int diff_b = resolve( a: b,  n: b_length);

    // 输出划分结果及其差
    cout << "划分结果: " << endl;
    cout << "B1为: ";
    for (int i = 0; i < b_length/2; i++) {
        cout << b[i] << " ";
    }
    cout << endl;
    cout << "B2为: ";
    for (int i = b_length/2; i < b_length; i++)
    {
        cout << b[i] << " ";
    }
    cout << endl;
    cout <<"S1与S2的差值为:  " <<diff_b<<endl;
    return 0;
}
```

运行结果：

```
2.5.5问题求解所得结果为：
初始序列A为：1 3 5 7 9 2 4 6 8
划分结果：
A1为: 1 2 3 4
A2为: 5 6 7 9 8
S1与S2的差值为:  25

初始序列B为：1 3 5 7 9 10 2 4 6 8
划分结果：
B1为: 1 2 3 4 5
B2为: 6 7 10 9 8
S1与S2的差值为:  25
```

时间复杂度分析：

该算法的时间复杂度为 O(n)。

在划分函数 divide 中，其核心操作是在数组中寻找轴值，并将小于轴值的元素放到轴值左边，大于轴值的元素放到轴值右边。具体来说，该函数使用了类似快速排序的思想，在数组中选择一个轴值（这里是数组的第一个元素），然后从数组的两端开始向中间遍历，不断交换位置以满足轴值左边的元素都小于等于轴值，轴值右边的元素都大于等于轴值。因为该函数只对输入的一部分进行处理，所以时间复杂度与输入规模有关，可以表示为 O(n)。

在求中位数并计算左右子数组的差值的函数 resolve 中，该函数基于划分函数 divide 实现。首先确定输入数组的中位数，然后依次判断轴值所处的位置和中位数的大小关系，如果轴值在中位数的左边，就在轴值的右半部分继续查找中位数；如果轴值在中位数的右边，就在轴值的左半部分继续查找中位数。这个过程可以保证最后找到的轴值是中位数所在的位置。接着，函数计算输入数组中，中位数左边的元素之和，以及中位数右边的元素之和，然后用两者之差作为返回值。这个过程需要遍历整个数组，因此时间复杂度为 O(n)。

综上所述，该算法仅仅只是根据轴值大小将序列分成大于轴值和小于轴值的两部分，并没有将序列进行排序，所以其时间复杂度为 O(n)。

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。