

算法作业 Week 4

1. 统计一个大小为 N 的一维数组中逆序的个数（归并排序求解）

分析：利用归并排序的变种方法求解。用变量 `ans` 记录逆序数，在每次合并时，如果将后面一组的数先插入，则说明存在逆序对，答案变化如下：

$$\text{ans} = \text{ans} + \text{前面一组剩余数的个数} = \text{ans} + \text{mid} - i$$

时间复杂度为 $O(n\log(n))$

代码：

```
// 求逆序数 = 逆序对的数量 = SUM(每个数后面比它小的个数)
// 使用归并排序的方式，在每次归并的时候进行处理。
// 归并时，每次安放后面一组中的数的时候，计数器 count+=前面一组剩余数的数量

#include <iostream>
using namespace std;

const int MAXN = 1000;
int a[MAXN];
int b[MAXN];
int ans = 0;

// 对 a[l]-a[r] 这一区间进行归并排序
void MergeSort(int a[], int l, int r) {
    if (r <= l) return;
    int mid = (l + r) / 2;
    // 对两侧进行归并排序
    MergeSort(a, l, mid);
    MergeSort(a, mid + 1, r);
    // 将待合并数据复制到 b 里
    for (int i = l; i <= r; i++) {
        b[i] = a[i];
    }
    // 原位置归并，并统计逆序数
    int i = l, j = mid + 1;
    for (int k = l; k <= r; k++) {
        if (i > mid)
            a[k] = b[j++];
        else if (j > r)
            a[k] = b[i++];
        else if (b[j] < b[i]) {
            a[k] = b[j++];
            ans = ans + mid - i + 1;
        } else
            a[k] = b[i++];
    }
}
```

```

}

int main() {
    int N;
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> a[i];
    }
    MergeSort(a, 0, N - 1);
    cout << ans << endl;
    return 0;
}

```

运行结果 1: (逆序数对为 21, 41, 43, 31, 一共 4 组)

```

4
2 4 3 1
4

```

运行结果 2: (逆序数对为 32, 31, 21, 54, 51, 41, 一共 6 组)

```

5
3 2 5 4 1
6

```

2. 在一个序列中出现次数最多的元素称为众数。请设计算法寻找众数，并分析算法的时间复杂度。

分析：首先利用快速排序将数组元素从小到大排序，之后遍历两遍数组，第一遍找到最大重复次数，第二遍输出所有众数。排序最优的时间复杂度为 $O(n\log(n))$ ，遍历的时间复杂度为 $O(n)$ ，因此总时间复杂度 $O(n\log(n))$

代码：

```

// 先排序，遍历一遍数组找到最大重复次数，再遍历一遍输出所有众数
// 采用快速排序，时间复杂度  $O(n\log(n))$ 
#include <algorithm>
#include <iostream>
using namespace std;

const int MAXN = 1000;
int a[MAXN]; // 存放输入的数据
int N;       // 输入数据的长度
int main() {
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> a[i];
    }
    sort(a, a + N);
    int max_len = 0, l = 0, r = 0;
    // 扫第一遍，求出最大长度

```

```

    while (r < N) {
        l = r;
        while (r + 1 < N && a[r + 1] == a[r]) r++;
        if (r - l + 1 >= max_len) max_len = r - l + 1;
        r++;
    }
    // 扫第二遍，输出答案
    l = 0, r = 0;
    while (r < N) {
        l = r;
        while (r + 1 < N && a[r + 1] == a[r]) r++;
        if (r - l + 1 == max_len) cout<<a[l]<<endl;
        r++;
    }
    return 0;
}

```

运行结果 1: (众数为 2, 3, 4,)

```

8
4 4 3 2 3 2 1 5
2
3
4

```

运行结果 2: (众数为 9)

```

10
1 2 3 4 5 6 7 8 9 9
9

```

3. 设 M 是一个 $n \times n$ 的整数矩阵，其中每一行（从左到右）和每一列（从上到下）的元素都是按照升序排列的。请设计分治算法确定一个给定的整数 x 是否在 M 中，并分析算法的时间复杂度。

分析：在每一行中使用二分法查找 x 。在最坏的情况下需要在每行都进行查找，因此算法的时间复杂度为 $O(n \log(n))$ 。

代码：

```

// 在升序排列的 n*n 矩阵中查找 x
// 利用二分法进行查找，算法复杂度  $O(n \log(n))$ 

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

const int MAXN = 100;
int a[MAXN][MAXN], N;

```

```

bool find(int x) {
    for (int i = 1; i <= N; i++) {
        // 每一行进行二分查找
        int l = 1, r = N, mid;
        while (l <= r) {
            mid = (l + r) / 2;
            if (a[i][mid] == x) {
                cout << "(" << i << "," << mid << ")" << endl;
                return true;
            } else if (a[i][mid] < x) {
                r = mid - 1;
            } else {
                l = mid + 1;
            }
        }
    }
    return false;
}

int main() {
    int x;
    cin >> N;
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            cin >> a[i][j];
        }
    }
    cin >> x;
    if (find(x))
        cout << "True" << endl;
    else
        cout << "False" << endl;
    return 0;
}

```

运行结果 1: (第一行输入 n, 接下来 n 行输入矩阵, 之后输入 x)

```

3
1 2 3
4 5 6
7 8 9
5
(2,2)
True

```

运行结果 2:

```

5
1 3 5 7 9
2 3 6 9 10
4 5 8 10 11
11 12 13 14 15
12 13 14 15 16
17
False

```

4.

已知：每个飞机只有一个油箱，飞机之间可以相互加油（注意是相互，没有加油机）一箱油可供一架飞机绕地球飞半圈。

问题：为使至少一架飞机绕地球一圈回到起飞时的飞机场，至少需要出动几架飞机？（所有飞机从同一机场起飞，而且必须安全返回机场美不允许中途降落，中间没有飞机场）。

解答：至少需要 3 架飞机，出动 5 架次。

设三架飞机分别为 A、B、C。

1. ABC 同时顺时针出发，至 $1/8$ 圈处，C 为 AB 加满油，C 剩余 $1/4$ 油量返航；
2. AB 飞至 $1/4$ 圈处，B 为 A 加满油，B 剩余 $1/2$ 油量返航；
3. A 用所有油量飞至 $3/4$ 圈处；B 逆时针飞至 $3/4$ 圈处，给 A 加 $1/8$ 的油，两飞机一同飞到 $7/8$ 圈处；
4. C 逆时针飞到 $7/8$ 圈处接应，为 AB 各加 $1/8$ 油量；ABC 一同飞回机场，油恰好用尽。

