第 3 章　排序与查找

排序与查找可以说是计算机领域最经典的问题，相信很多读者在本科阶段都学过各式各样的排序和查找方法。排序和查找问题在考研机试真题中经常出现，本章旨在讲解一些经典问题，详细说明解题方法，使读者日后在面对此类问题时能够做到游刃有余。

3.1　排序

排序考点在历年机试考点中分布广泛。排序既是考生必须掌握的基本算法，又是考生学习其他大部分算法的前提和基础。

首先学习对基本类型的排序。对基本类型排序，是指对诸如整数、浮点数等计算机编程语言内置的基本类型进行排序的过程。

例题 3.1　排序（华中科技大学复试上机题）

题目描述：

对输入的 n 个数进行排序并输出。

输入：

输入的第一行包括一个整数 n（$1 \le n \le 100$）。接下来的一行包括 n 个整数。

输出：

可能有多组测试数据，对于每组数据，输出排序后的 n 个整数,每个数后面都有一个空格。
每组测试数据的结果占一行。

样例输入：

```
4
1 4 3 2
```

样例输出：

```
1 2 3 4
```

提交网址：

```
http://t.cn/E9dLx5K
```

【分析】

这是在机试中曾经出现的关于排序考点的一个真题。面对这样的考题，读者可能很快就会联想到教科书上各种特点不同的排序算法，如快速排序、归并排序等。然而，要在机试的短时间内快速并正确地写出高效的排序算法，相信大部分考生都是力不从心的。然而，并不需要为此担心，因为 C++内部已为大家编写了基于快速排序的函数 sort，只需调用这个函数，便能轻易地完成排序。下面简单介绍 sort 函数。sort(first,last,comp)函数有三个参数：first 和 last 为待排序序列的起始地址和结束地址；comp 为排序方式，可以不填写，不填写时默认为升序方式。

代码 3.1

```
#include <iostream>
#include <cstdio>
#include <algorithm>

using namespace std;

const int MAXN = 100;                    //数据量定义为常数

int arr[MAXN];

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; ++i) {
            scanf("%d", &arr[i]);
        }
        sort(arr, arr + n);              //默认升序排序
        for (int i = 0; i < n; ++i) {
            printf("%d ", arr[i]);
        }
        printf("\n");
    }
    return 0;
}
```

sort 库函数包含在 algorithm 头文件中，因此要在程序中引入 algorithm 头文件。本题中，sort 函数的两个参数分别代表待排序序列的起始地址 arr 和结束地址 arr+n。该函数调用完成后，arr 数组中的数字就已经通过快速排序变成了升序序列，之后只需对其输出即可。

如果要对给定的数组进行降序排列或以自定义的方式排列，那么应该如何操作呢？此时，sort 函数的第三个参数会起作用。可以通过编写比较函数的方式来实现想要的自定义排序方式。

例题 3.2　成绩排序（清华大学复试上机题）

题目描述：

用一维数组存储学号和成绩，然后按成绩排序输出。

输入：

输入的第一行中包括一个整数 N（$1 \le N \le 100$），它代表学生的个数。
接下来的 N 行中，每行包括两个整数 p 和 q，分别代表每个学生的学号和成绩。

输出：

按照学生的成绩从小到大进行排序，并将排序后的学生信息打印出来。
如果学生的成绩相同，那么按照学号的大小从小到大排序。

样例输入：

```
3
1 90
2 87
3 92
```

样例输出：

```
2 87
1 90
3 92
```

提交网址：

```
http://t.cn/E9d3ysv
```

【分析】

本题和上题类似，都是以升序方式排序，但本题的排序规则多了一些限制：

① 本题不再是对基本类型进行排序，而是对一组基本类型进行排序。因此，需要首先将这一组基本类型构建为一个结构体或类，然后对结构体或类进行排序。

② 排序的要求在题面中已经明确给出，即先按照成绩升序排序，成绩相同时按照学号升序排序。此时，sort 函数的默认排序方式无法满足本题的要求，需要编写一个自定义的比较函数才能完成上述要求。

代码 3.2

```
#include <iostream>
#include <cstdio>
#include <algorithm>

using namespace std;
```

```
struct Student {
    int number;                                          //学号
    int score;                                           //成绩
};

const int MAXN = 100;

Student arr[MAXN];

bool Compare(Student x, Student y) {                     //比较函数
    if (x.score == y.score) {                            //成绩相等比较学号
        return x.number < y.number;
    } else {                                             //成绩不等比较成绩
        return x.score < y.score;
    }
}

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%d%d", &arr[i].number, &arr[i].score);
    }
    sort(arr, arr + n, Compare);                         //按照比较函数排序
    for (int i = 0; i < n; ++i) {
        printf("%d %d\n", arr[i].number, arr[i].score);
    }
    return 0;
}
```

① 上述代码定义了一个学生结构体 Student，这个结构体包括两个整型变量，即学号和成绩。读者以后遇到一组基本类型数据时，要学会将其封装为一个结构体或类进行处理。

② 上述代码还定义了一个比较函数 Compare，这个比较函数实现了题目中的排序规则。考生在遇到新的排序规则时，只需记住如下这条黄金法则：当比较函数的返回值为 true 时，表示的是比较函数的第一个参数将会排在第二个参数前面。记住这条规则后，就可在日后面对不同的排序规则时，设计出相应的比较函数。

例题 3.3 成绩排序（清华大学复试上机题）

题目描述：

输入任意(用户,成绩)序列，可以获得成绩从高到低或从低到高的排列，相同成绩都按先录入者排列在前的规则处理。

示例：

```
jack      70
```

```
peter      96
Tom        70
smith      67
从高到低   成绩
peter      96
jack       70
Tom        70
smith      67
从低到高   成绩
smith      67
jack       70
Tom        70
peter      96
```

输入：

```
输入多行，首先输入要排序的人的个数，然后输入排序方法 0（降序）或 1（升序），
再分别输入他们的名字和成绩，以一个空格隔开。
```

输出：

```
按照指定方式输出名字和成绩，名字和成绩之间以一个空格隔开。
```

样例输入：

```
3
0
fang 90
yang 50
ning 70
```

样例输出：

```
fang 90
ning 70
yang 50
```

提交网址：

```
http://t.cn/E9gyHM1
```

【分析】

和上一道题相比，本题最大的特点是排序方式并不唯一，需要根据输入来确定是采用升序排序还是采用降序排序。对于本题而言，最简单的方法是，首先根据题意设计两个不同的比较函数（一个用于升序排序，一个用于降序排序），然后根据不同的输入选择不同的比较函数。

代码 3.3

```
#include <iostream>
#include <cstdio>
#include <algorithm>

using namespace std;

struct Student {
    string name;                                    //姓名
    int score;                                      //成绩
    int order;                                      //次序
};

bool CompareDescending(Student x, Student y) {      //降序比较函数
    if (x.score == y.score) {                       //成绩相等比较次序
        return x.order < y.order;
    } else {                                        //成绩不等比较成绩
        return x.score > y.score;
    }
}

bool CompareAscending(Student x, Student y) {       //升序比较函数
    if (x.score == y.score) {                       //成绩相等比较次序
        return x.order < y.order;
    } else {                                        //成绩不等比较成绩
        return x.score < y.score;
    }
}

int main() {
    int n;
    int type;
    while (scanf("%d%d", &n, &type) != EOF) {
        Student arr[n];                             //数据范围未知
        for (int i = 0; i < n; ++i) {
            cin >> arr[i].name >> arr[i].score;
            arr[i].order = i;
        }
        if (type == 0) {                            //选择比较函数
            sort(arr, arr + n, CompareDescending);
        } else {
            sort(arr, arr + n, CompareAscending);
        }
```

```
            for (int i = 0; i < n; ++i) {
                cout << arr[i].name << " " << arr[i].score << endl;
            }
        }
        return 0;
    }
```

① 上述代码定义了一个学生结构体 Student，这个结构体包括一个字符串变量 name（姓名）及两个整型变量 score（成绩）和 order（次序）。

② 上述代码还定义了两个比较函数，一个用于成绩升序，一个用于成绩降序，当成绩相同时，两个比较函数都按照输入次序进行升序排序。程序会根据输入的不同选择不同的比较函数。

对于结构体或类的自定义排序，除通过设计比较函数外，还有一种常用的方法，即在结构体或类的内部重载小于号。对此方法有兴趣的读者，可以自行阅读相关资料。

相信通过本节的学习与练习，读者不会再对排序问题感到恐惧：只需用 sort 函数按照题面要求定义不同的排序规则。如果读者对 sort 函数还有疑问，或者想要深入学习，那么可以访问网址 http://www.cplusplus.com/reference/algorithm/sort/。

习题 3.1　特殊排序（华中科技大学复试上机题）

输入一系列整数，将其中最大的数挑出（有多个最大数时，挑出一个即可），并对剩下的数排序，如果无剩下的数，那么输出-1。

提交网址：

http://t.cn/E9gio39

习题 3.2　整数奇偶排序（北京大学复试上机题）

输入 10 个整数，彼此以空格分隔。重新排序后输出（也按空格分隔），要求：
1）首先输出其中的奇数，并且按照从大到小的顺序排列；2）然后输出其中的偶数，并且按照从小到大的顺序排列。

提交网址：

http://t.cn/E9glPvp

习题 3.3　小白鼠排队（北京大学复试上机题）

有 N 只小白鼠（$1 \le N \le 100$），每只小白鼠头上戴一顶有颜色的帽子。现在称出每只白鼠的重量，要求按照白鼠重量从大到小的顺序，输出它们头上帽子的颜色。帽子的颜色用“red”“blue”等字符串表示。不同的小白鼠可以戴相同颜色的帽子。白鼠的重量用整数表示。

提交网址：

http://t.cn/E9gXh9Z

习题 3.4　奥运排序问题（浙江大学复试上机题）

给定国家或地区的奥运金牌数、奖牌数和人口数（百万）。
排序有 4 种方式：金牌总数、奖牌总数、金牌人口比例、奖牌人口比例。
对每个国家给出最佳的排名方式和最终排名。

提交网址：

```
http://t.cn/E9gYpyl
```

3.2　查找

查找是另一类必须掌握的基础算法，它不仅会在机试中直接考查，而且是其他某些算法的基础。之所以将查找和排序放在一起讲，是因为二者有较强的联系。排序的重要意义之一便是帮助人们更加方便地进行查找。如果不对数据进行排序，那么在查找某个特定的元素时，需要依次检查所有的元素，这样的方式对于单次或少量的查找来说运行效率是很高的，但查找次数较多时，如果所有元素都是有序的，那么就能更快地进行检索，而不必逐个元素地进行比较。

例题 3.4　找 x（哈尔滨工业大学复试上机题）

题目描述：

输入一个数 n，然后输入 n 个不同的数值，再输入一个值 x，输出这个值在数组中的下标（从 0 开始，若不在数组中则输出-1）。

输入：

测试数据有多组。输入 n（$1 \le n \le 200$），接着输入 n 个数，然后输入 x。

输出：

对于每组输入,请输出结果。

样例输入：

```
2
1 3
0
```

样例输出：

```
-1
```

提交网址：

```
http://t.cn/E9gHFnS
```

【分析】

由此例可以看出，即使是在数组中查找特定数字这样的最基本的问题，也会在考研机试真题中出现，由此可以看出查找在整个算法体系中的重要性。通过此例可以了解查找涉及的几个基本要素。

① 查找空间。也称解空间。所谓查找，就是在查找空间中找寻符合要求的解的过程。在此例中，整个数组包含的整数集就是查找空间。

② 查找目标。需要一个目标来判断查找空间中的各个元素是否符合要求，以便判断查找活动是否成功。在此例中，即判断数组中的数字与目标数字是否相同。

③ 查找方法。利用某种特定的策略在查找空间中查找各个元素。不同的策略对查找的效率和结果有不同的影响。因此，对于某个特定的问题，要选择切实可行的策略来查找解空间，以期事半功倍。在此题中，查找方法为线性地遍历数组。

代码 3.4

```
#include <iostream>
#include <cstdio>

using namespace std;

const int MAXN = 200;

int arr[MAXN];

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; ++i) {
            scanf("%d", &arr[i]);
        }
        int x;
        scanf("%d", &x);
        int answer = -1;
        for (int i = 0; i < n; ++i) {
            if (arr[i] == x) {
                answer = i;
                break;
            }
        }
        printf("%d\n", answer);
    }
    return 0;
}
```

下面介绍一种不同于线性查找的查找策略，而是一种策略性、跳跃性的方法来遍历查找空间，它就是二分查找。二分查找建立在待查找元素排列有序的前提下。假设元素是按升序排列的，将序列中间位置记录的关键字与查找关键字比较，若二者相等，则查找成功；否则利用中间位置记录的关键字将序列分成前、后两个子序列，若中间位置记录的关键字大于查找关键字，则进一步查找前面的子序列，否则进一步查找后面的子序列。重复以上过程，直到找到满足条件的记录（查找成功），或到子序列不存在为止（此时查找不成功）。

由于每次查找都会舍弃一半的元素，因此二分查找的时间复杂度可由线性查找的 $O(n)$ 降至 $O(\log n)$。有的读者可能会有疑问，虽然二分查找的时间复杂度要低于线性查找，但排序的时间复杂度至少是 $O(n\log n)$，如此说来线性查找岂不是要优于二分查找？如果只要求对序列进行单次查找，那么线性查找不失为一个好策略，但如果要进行多次查找，那么线性查找将不再具有优势。假设查找次数为 m，那么线性查找的时间复杂度为 $O(nm)$，二分查找的时间复杂度为 $O(n\log n + m\log n)$，可见当 m 特别小时线性查找很有优势，但当 m 大到一定程度时，二分查找的性能便会远远优于线性查找。

例题 3.5　查找（北京邮电大学复试上机题）

题目描述：

输入数组长度 n，输入数组 $a[1\ldots n]$，输入查找次数 m，输入查找数字 $b[1\ldots m]$，输出 YES 或 NO，找到则输出 YES，否则输出 NO。

输入：

输入有多组数据。
每组输入 n，然后输入 n 个整数，再输入 m，最后输入 m 个整数（$1 \le m, n \le 100$）。

输出：

若在 n 个数组中，则输出 YES，否则输出 NO。

样例输入：

```
5
1 5 2 4 3
3
2 5 6
```

样例输出：

```
YES
YES
NO
```

提交网址：

```
http://t.cn/E9g8aaR
```

【分析】

读题并分析后，读者肯定会发现本题的数据量和查询次数都比较小，因此仍然可以使用线性查找的方法求解。然而，当读者在考场上遇到题目的数据量和查询量都比较大时，如果仍然采用线性查找的方法求解，那么肯定会导致程序超时。这时，必须使用二分查找方法乃至更高性能的查找方法。本题的目的就是向读者介绍二分查找方法。

代码 3.5

```
#include <iostream>
#include <cstdio>
#include <algorithm>

using namespace std;

const int MAXN = 100;

int arr[MAXN];

bool BinarySearch(int n, int target) {          //二分查找
    int left = 0;
    int right = n - 1;
    while (left <= right) {
        int middle = (left + right) / 2;        //中间位置
        if (arr[middle] < target) {             //小于关键字，舍弃左边
            left = middle + 1;
        } else if (target < arr[middle]) {      //大于关键字，舍弃右边
            right = middle - 1;
        } else {
            return true;                        //查找成功
        }
    }
    return false;
}

int main() {
    int n, m;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; ++i) {
            scanf("%d", &arr[i]);
        }
        sort(arr, arr + n);                     //先排序再查找
        scanf("%d", &m);
```

```
            for (int i = 0; i < m; ++i) {
                int target;
                scanf("%d", &target);
                if (BinarySearch(n, target)) {
                    printf("YES\n");
                } else {
                    printf("NO\n");
                }
            }
        }
        return 0;
    }
```

结合上面的代码和之前的讲解，相信读者可以很快地掌握二分查找的特点，并能够独立地写出相关的代码。

注意中间位置的那行代码 `int middle = (left+right)/2`。通常情况下，这样写既没有错误，又方便大家理解。不过，有时会出现 `left` 和 `right` 非常接近整型最大值的情况，这时如果再这样求中间位置就会出现溢出问题，因此要用代码 `int middle = left + (right - left)/2` 代替。二者的功能等价，并且后者不会出现溢出问题。

习题 3.5　找最小数（北京邮电大学复试上机题）

第一行输入一个数 n，$1 \le n \le 1000$；接着输入 n 行数据，每一行有两个数，分别是 x 和 y，输出格式是“$x\ y$”。输出一组“$x\ y$”，这组数据在所有数据中，x 是最小的，且 x 相等时 y 是最小的。

提交网址：

http://t.cn/E9gekWa

习题 3.6　打印极值点下标（北京大学复试上机题）

在一个整数数组上，对于下标为 i 的整数，如果它大于所有与它相邻的整数，或者小于所有与它相邻的整数，那么称该整数为一个极值点，极值点的下标就是 i。

提交网址：

http://t.cn/E9ehDw4

习题 3.7　找位置（华中科技大学复试上机题）

对给定的一个字符串，找出有重复的字符，并给出其位置。例如，对于字符串 abcaaAB12ab12，输出如下：a,1; a,4; a,5; a,10,b,2; b,11,1,8; 1,12,2,9; 2,13。

提交网址：

http://t.cn/E9eh4jY

小结

本章介绍了排序和查找的基本概念，其中排序是机试中出现概率很大的一类题型，希望读者能把握其中的规律。查找部分介绍了两种简单的查找方法，即线性查找和二分查找，其中二分查找需要读者重点把握，因此它在机试中出现的概率更大。更为高效的查找方式将在关于数据结构章节的散列表部分介绍。