# 从零开始

## 评判结果

* Accepted（答案正确）
* Wrong Answer（答案错误）：

出现该种错误后，一般有两种解决方向：

1. 如果对设计的算法正确性有较大的把握，那么你可以重点考虑是否存在某些特殊数据使程序出现错误，比如边界数据，比如程序中变量出现溢出。
2. 另一种方向，即怀疑算法本身的正确性，那么你就需要重新考虑你的算法设计了。

* Time Limit Exceeded（超出时间限制）：

若这种结果出现在你的评判结果里，依然有两种方向可供参考：

1. 假如你确定算法时间复杂度能够符合题目的要求，那么依旧可以检查是否程序可能在某种情况下出现死循环，是否有边界数据可能会让你的代码不按照预想的工作，从而使程序不能正常的结束。
2. 你设计的算法时间复杂度是否已经高于题目对复杂度的要求，如果是这样，那么你需要重新设计更加高效的算法或者对你现行的算法进行一定的优化。

* Runtime Error（运行时错误）：

你可以考虑以下几个要点来排除该错误：

1. 程序是否访问了不该访问的内存地址，比如访问数组下标越界。
2. 程序是否出现了除以整数 0，从而使程序异常。
3. 程序是否调用了评判系统禁止调用的函数。
4. 程序是否会出现因为递归过深或其他原因造成的栈溢出。

* Compile Error（编译错误）
* Memory Limit Exceeded （使用内存超出限制）

造成这种错误的原因主要有两个方面：

1. 你的程序申请过多的内存来完成所要求的工作，即算法空间复杂度过高。
2. 因为程序本身的某种错误使得程序不断的申请内存，例如因为某种原因出现了死循环，使得队列中不断的被放入元素。当然也千万别忽略自己的低级错误，比如在声明数组大小时多打了一个 0。

## 复杂度的估计

* 一般题目中会给我们的程序 1 秒的运行时限，这也是最常见的时间限制（或最常见的时间限制数量级）。对于该时限，通常，我们所设计的算法复杂度不能超过百万级别，即不能超过一千万。即若算法的时间复杂度是 O(n^2)，则该 n（往往在题目中会给出数据范围）不应大于 3000，否则将会达到我们所说的千万数量级复杂度，从而程序运行时间超出题目中给出的用时限定。
* 我们在很多情况下都应该有“空间换时间”的思想。

# 经典入门

## 排序

在《数据结构》中接触的排序包括：直接插入排序、折半插入排序、希尔排序、冒泡排序、快速排序、简单选择排序、堆排序、归并排序、基数排序

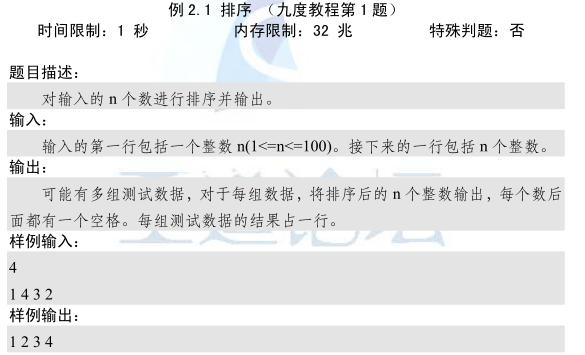
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 排序方法 | | 平均时间复杂度 | 最坏时间复杂度 | 空间复杂度 | 稳定性 |
| 简  单  排  序 | 直接插入排序 | O(n2) | O(n2) | O(1) | 稳定 |
| 折半插入排序 | O(n2) | O(n2) | O(1) | 稳定 |
| 起泡排序 | O(n2) | O(n2) | O(1) | 稳定 |
| 简单选择排序 | O(n2) | O(n2) | O(1) | 不稳定 |
| 高  级  排  序 | 快速排序 | O(nlogn) | O(n2) | O(logn) | 不稳定 |
| 堆排序 | O(nlogn) | O(nlogn) | O(1) | 不稳定 |
| 希尔排序 | O(n1+k)(0<k<1) | O(n1+k)(0<k<1) | O(1) | 不稳定 |
| 归并排序 | O(nlogn) | O(nlogn) | O(n) | 稳定 |
| 基数排序 | O(d(n+rd)) | O(d(n+rd)) | O(rd) | 稳定 |

选择排序算法准则：

一般而言，需要考虑的因素有以下四点：设待排序元素的个数为n.

* 当n较大，则应采用时间复杂度为O(nlog2n)的排序方法：快速排序、堆排序或归并排序
* 当n较大，内存空间允许，且要求稳定性：归并排序
* 当n较小，可采用直接插入或直接选择排序。
  1. 直接插入排序：当元素分布有序，直接插入排序将大大减少比较次数和移动记录的次数。
  2. 直接选择排序 ：元素分布有序，如果不要求稳定性，选择直接选择排序
* 一般不使用或不直接使用传统的冒泡排序。

**经典例题：2006 年华中科技大学计算机保研机试真题**



**使用冒泡排序：**

#include <iostream>

using namespace std;

int main()

{

int n;

int buf[105]; // 根据题目中的n的取值范围定义buf数组的长度

while(cin >> n)// 如果在C语言中实现需要使用 while(scanf("%d",&n)!=EOF)

{

for(int i = 0;i<n;i++)

cin >> buf[i]; // 初始化

for(int i = 0;i<n;i++)

for(int j = 0;j<n-i-1;j++)

if(buf[j]>buf[j+1]){} // 交换 冒泡排序的核心内容

for(int i = 0;i<n;i++)

cout << buf[i] << ' '; // 输出

cout << endl;

}

return 0;

}

说明：

（1）while(scanf(“%d”,&n)!=EOF) scanf函数是有返回值的，它将返回被输入函数成功赋值的变量个数。在此例中，若成功完成输入并对n赋值，scanf函数的返回值即为1

（2）由于改题目中说明由多组测试用例，因此需要用循环输入，当输入一个值，不等于EOF(-1)，循环条件成立，程序进入循环体，执行程序；如果输入已经到达结尾（输入文件到达末尾或在命令台输入中输入Ctrl+z），scanf函数无法再为变量n赋值，于是scanf函数返回EOF(end of file)

（3）若输入为字符串而程序采用gets()的方法读入，则相同功能的循环判断语句为while(gets(字符串变量))

更多的字符串输入处理看文章：

<https://www.cnblogs.com/mist2019/p/10327764.html>

推荐使用getline() 和 fgets()方法

**使用快速排序：**

当n的级别越来越大的时候，已经超过了百万级别，于是我们需要使用诸如快速排序、归并排序等具有更优复杂度的排序算法。

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

int n;

int buf[10005];

while(cin >> n)

{

for(int i = 0;i<n;i++)

cin >> buf[i]; // 初始化

sort(buf,buf+n);

for(int i = 0;i<n;i++)

cout << buf[i] << ' '; // 输出

cout << endl;

}

return 0;

}

引用了sort函数，该函数是在头文件algorithm中，起始地址为 buf，结束地址为 buf + n。该函数调用完成后，buf 数组中的数字就已经通过快速排序升序排列。

补充：algorithm库<https://blog.csdn.net/wuce_bai/article/details/51027166>

默认的sort的排序是升序排序，如果要完成降序排序，那么需要：

#include <algorithm>

using namespace std;

bool cmp(int x,int y)

{

return x>y;

}

int main()

{

while(cin >> n)

{

// 相关操作

sort(buf,buf+n,cmp);

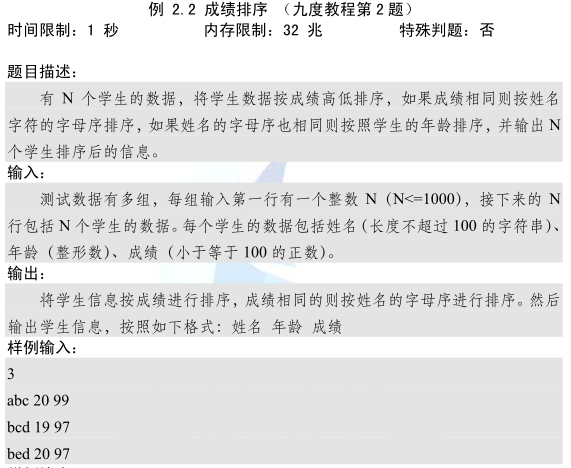
// 相关操作

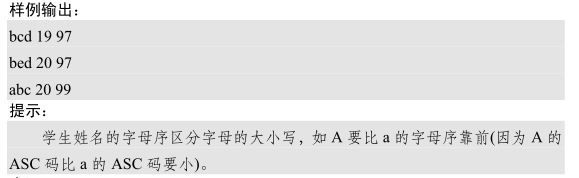
}

}

关于 cmp 函数的定义规则我们只需简单的记得，当 cmp 的返回值为 true 时，即表示 cmp 函数的第一个参数将会排在第二个参数之前（即在我们定义的规则中，cmp 表示第一个参数是否比第二个参数大，若是，则排在前面）。

**经典例题：2000 年清华大学计算机研究生机试真题**





#include <iostream>

#include <algorithm>

using namespace std;

typedef struct Student

{

char name[105];// 姓名

int age;// 年龄

int score;// 成绩

}Student;

bool cmp(Student a,Student b)

{

if(a.score != b.score)

return a.score < b.score; // 若分数不相同则分数低者在前

int tmp = strcmp(a.name,b.name);

if(tmp != 0)

return tmp < 0;// 若分数相同则名字字典序小者在前

else

return a.age < b.age; // 若名字也相同则年龄小者在前

}

int main()

{

int N;

Student buf[105];

while(cin >> N)

{

for(int i = 0;i<N;i++)

cin >> buf[i].name >> buf[i].age >> buf[i].score;

sort(buf,buf+N,cmp);

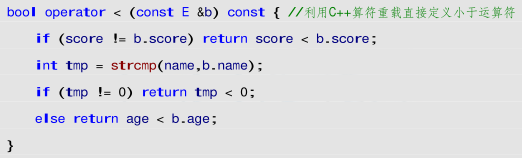
for(int i = 0;i<N;i++)

cout << buf[i].name << ' ' << buf[i].age << ' ' << buf[i].score << endl;

}

return 0;

}



以上是排序的小于运算符重载内容，非常的方便！调用sort函数的时候，会自动的调用运算符重载函数（这一部分需要重新学习一下C++）