开篇：

一个程序员曾经问我一个简单的问题：“怎样给一个磁盘文件排序？”

而我却直接犯了一个错误。

现在，在我开始讲这个故事之前，我想请大家思考下，如何回答这个问题呢？

### 一次友好的对话：

我最大的错误就是，马上回答了这个问题。我告诉他一些在磁盘上实现归并排序的简要思路，并建议他深入研究算法教材。不过他并不感冒。他更关心如何解决具体问题，而非深入学习。

于是我告诉他一本流行的程序设计书里有磁盘排序的程序。大约有200行代码和十几个函数，我估计最多一周的时间应该足以让他完成实现与测试的工作。

我以为我已经解决了他的问题。但事实并非如初，于是有了下面的对话

为什么非要自己编写排序程序呢？为什么不用系统提供的排序功能呢？

我需要在一个大系统中排序。由于一些技术原因，我不能使用系统中的排序功能。

需要排序的内容是什么？多少记录？格式？

最多一千万条记录，每条记录都是7位的整数。

既然文件如此之小，为什么要在磁盘上而不是内存里排序呢？

排序功能只是大系统中的一部分，所以估计我实际上只有1MB左右的内存可用。

记录还有其他特征么？

每条记录都是七位的正整数，没有其他相关数据，且每个整数都只出现一次。

这段对话让问题更明确了，在美国，电话号码是由3位“区号”后再跟7位数字组成。拨打含“免费”区号800（当时只有这么一个号码）的电话是不收费的。实际的免费电话号码数据库包含大量信息：免费的电话号码，呼叫实际中转接的一个或几个号码，用户姓名地址等。

这位程序员正在开发这类数据库的处理系统的一小部分，需要排序的整数就是免费电话号码，输入文件是电话号码的列表，并已经删除其他信息，号码重复出现算出错。期望的输文件是以升序排列的电话号码列表。由于其应用背景，用户大约每小时请求一次有序文件，并且在排序完成之前什么也干不了。因此，排序最多只允许执行几分钟，10秒则是一个比较理想的运行时间。

### 准确的问题描述

对这位程序员来说，这些需求加起来就是：“如何给磁盘文件排序？”在试图解决这个问题之前，先将已知条件组织成一种更客观、更易用的形式。

输入：一个最多包含n个正整数的文件，每个数都小于n，其中n=10^7。如果在输入文件中有任何重复出现的整数就是致命错误，没有其他数据与该整数相关联。

输出：按升序排列的输入整数的列表。

约束：最多有大约1MB的内存空间可用，有充足的磁盘空间可用。运行时间最多几分钟，而运行时间为10秒就不需要进一步优化了。

请花一点时间思考一下该问题的规范说明。现在你打算给该程序员什么样的建议呢？

### 程序设计

显而易见的方法是以一般的基于磁盘的归并排序程序为起点，对其进行调整，因为我们是对整数进行排序。这样就可以将原来的200行程序减少为几十行，同时也运行的更快，但是完成程序并使之运行可能依然需要几天的时间。

另外一种方法更多的利用了该排序问题的特殊性。如果每个号码都使用了7个字节来存储，那么在可用的1MB存储空间里大约可以存143,000个号码。如果每个号码都用32位整数来表示的话，在1MB存储空间里就可以存储250,000个号码。因此，可以使用遍历输入文件40次的程序来完成排序。在第一次遍历中，将0到249999之间的任何整数都读入内存，并对这最多250,000个整数进行排序，然后写到输出文件中。第二次遍历排序250,000到499,999之间的整数，以此类推。到第40次遍历时对9,750,000到9,999,999之间的整数进行排序。对内存中排序来说，快速排序会相当高效，而且仅仅需要20行代码。于是，整个程序就可以通过一两页纸的代码实现。该程序拥有所期望的特性—不必考虑使用中间磁盘文件；但是，为此所付出的代价是要读取输入文件40次。

归并排序读入输入文件一次，然后在工作文件的帮助下完成排序并写入输出文件一次。工作文件需要多次读写。

40趟算法读入输入文件多次，写输入文件仅一次，不使用中间文件。

下图所示的方案更好。我们结合上述两种方法的优点，读输入文件仅一次，且不使用中间文件。

只有在输入文件中的所有整数都可以在可用的1MB内存中表示的时候才能够实现该方案。于是问题就归结为是否能够用大约800万个可用位来表示最多1000万个互异的整数。

### 实现概要

我们想到了用 位图 来表示集合。可以用一个20位长的字符串来表示一个所有元素都小于20的简单的非负整数集合。例如：用如下字符串表示集合（1, 2, 3, 5, 8, 13）:

0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0

代表集合中数值的位都置为1，其他位置为0.

在我们的实际问题中，每个7位十进制整数表示一个小于1千万的整数。我们可以使用一个具有1千万个位的字符串来表示这个文件。其中，当且仅当整数i在文件中存在时，第i位为1.

这种表示利用了该问题的三个在排序问题中不常见的属性：

a.输入数据限制在相对较小的范围内；

b.数据没有重复;

c.没有其他数据关联。

若给定表示文件中整数集合的位图数据结构，则可以分三个阶段来编写程序。

第一阶段：将所有的位都置为0

第二阶段：通过读入文件中的每个整数来建立集合，将每个对应位置为1

第三阶段：遍历每一位，如果为1，就输出对应的整数

如果n为位图中的位数，即n=10,000,000，程序可以使用伪代码表示如下：

// phase 1: initialize set to empty

for i = [0, n]

bit[i] = 0

// phase 2: insert present elements into the set

for each i in the input file

bit[i] = 1

// phase 3: write sorted output

for i = [0, n]

if bit[i] == 1

write i on this output file

### 原理

那个程序员打电话把她的问题告诉我，然后我们花了大约一刻钟时间明确了问题所在，并找到了位图解决方案。他花了几个小时来实现并测试这个十几行代码的程序。该程序远远优于我们在电话刚开始时所担心的需要花费一周时间编写的几百行代码的那个程序。而且程序执行的很快：磁盘上的归并排序可能需要许多分钟的时间，该程序所需时间只比读取输入和写入输入所需的时间多一点点—大约10秒钟。

从这些事实中可以总结出该实例的第一个结论：对小问题的仔细分析有时可以得到明显的实际益处。在该实例中，几分钟的仔细研究可以大幅削减代码的长度、程序员的工作时间和程序的运行时间。正应了第一个超音速飞行的人对一架飞机的赞美之词“结构简单、部件很少、易于维护、非常坚固”。

然而，当规范说明的某些因素发生改变时，该程序的特殊结构将很难修改。除了精巧的编程之外，该实例阐明了如下一般原理。

正确的问题：

明确问题，这场战役就成功了90%，我很庆幸该程序员没有满足于第一个程序。

位图数据结构：

该数据结构描述了一个有限定义域内的稠密集合，其中的每一个元素最多出现一次并

且没有其他的数据与该元素相关联。即使这些条件没有完全满足，比如存在重复元素或额外

的数据，也可以用有限定义域内的键作为一个表项更复杂的表格的索引。

多趟算法：

这些算法多趟读入其输入数据，每次完成一步。我们已经见过了一个40趟算法。

时间—空间折中与双赢：

编程文献和理论中充斥着时间—空间的折中：通过使用更多的时间，可以减少程序所

需的空间。但很多时候，减少程序运行的空间也会减少其运行时间。空间上高效的位图结构

显著的减少了排序的运行时间。空间需求的减少之所以会导致运行时间的减少，有两个原因：

1. 需要处理的数据变少了，意味着所需时间表少

2. 数据保存在内存而非磁盘中，避免了磁盘访问

当然，只有在原始的设计远非最佳方案时，才有可能时空双赢。

简单的设计：

一位著名的法国作家兼飞机设计师说过：“设计者确定其设计已经达到完美的标准不

是不能再增加任何东西，而是不能再减少任何东西。”程序员应该使用该标准来检验自己完成

的程序。简单的程序通常比具有相同功能的复杂程序更可靠、安全、健壮、高效，而且易于

实现和维护。

思考：

1. 该程序员说他有1MB左右的内存空间，我们的代码实际上需要1.25MB的空间。他可以不费力气的索取到额外的空间。如果1MB是严格的边界，你会推荐如何处理呢？你的算法的运行时间又是多少呢？

2. 如果那个程序员说的不是每个整数最多出现一次，而是每个整数最多出现10次，你又如何建议他呢？你的解决方案如何随着可用存储空间总量的变化而变化？

3. 我们的程序假定在输入中没有出现两次的整数。如果某个数出现超过一次会发生什么？如何修改程序来调用错误处理函数？当输入的整数小于0或大于n时又会发生什么？如果某个输入不是整数呢？在这些情况下，程序该如何处理？程序还应该包括哪些检查？描述一些用以测试程序的小型数据集合，并说明如何正确处理上诉以及其他的不良情况。

4. 当时美国所有的免费电话区号都是800.现在却包括800，877和888，而且还在增多。如何在1MB空间内完成对所有这些免费号码的排序？如何将免费电话号码存储在一个集合中，要求可以实现非常快速的查找以判定一个给定的免费电话号码是否可用或者已经存在呢？

5. 在成本低廉的隔日送达时代之前，商店允许顾客通过电话订购商品，并在几天后上门自取。商店的数据库使用客户的电话号码作为检索的主关键字。你如何组织商店的数据库，以允许高效的插入和检索操作？