**实验一**

**一、实验目的**

1.理解分治法的思想，并可以将理论转化为实践，运用到普通的数学问题当中。

2.能够使用递归或迭代的方法编写分治算法，使得结果正确并且过程简洁易懂。

3.降低代码的时间空间复杂度，做好算法优化。

**二、实验内容**

实验(1)：设X[ 0 : n - 1]和Y[ 0 : n – 1 ]为两个数组，每个数组中含有n个已排好序的

数。找出X和Y的2n个数的中位数。

实验(3)：定义一个Advertisement类，该类中至少包含该物品的数量，名称，联系人

e-mail，最好有开拍时间及关闭时间，根据用户输入的关键字比如名称，

mail，时间等，利用非递归的归并排序对所有的广告进行排序，并列出所有

排好序的广告。

**三、实验环境**

操作系统：Windows 7

调试软件名称：Microsoft Visual C++ 6.0

**四、问题分析**

实验(1)：由于输入的是是两个相等大小的数组，因而数字的个数为偶数，中位数应当为中间的两个数，根据题意，在这里设大小较小的那个作为中位数。

当输入的两个数组(设为x，y)个数都为偶数(n)时，首先找出两个数组的中位数(设

下标为mid)，然后比较这两个数组中位数的大小，若x[mid]==y[mid]，则返回

x[mid]；若x[mid]<y[mid]，则重新建立数组xTemp和yTemp，大小为n/2，并将

x[mid+1]~x[n-1]赋值给xTemp，将y[0]~y[mid]赋值给yTemp，然后递归调用该算

法；若x[mid]>y[mid]，只是将x[mid]<y[mid]情况中的x与y互换即可。

当输入的两个数组(设为x，y)个数都为奇数(n)时，若x[mid]<y[mid]，则重新建立数组xTemp和yTemp，大小为(n+1)/2，并将x[mid]~x[n-1]赋值给xTemp，将y[0]~y[mid]赋值给yTemp，然后递归调用该算法。其它与过程相似。

由于算法中使用递归，因而需要一个终止条件，否则会陷入一个死循环中。该算法的终止条件为数组长度length==1时，返回大小较小的那个数即可。

利用该方法的时间复杂度为O(logn)。

实验(3)：该问题中明确提出了使用非递归的归并算法对该问题进行求解，因而解决该

问题的核心 算法可以确定下来：即为非递归归并算法。但是除此之外，要求解该问

题仍需注意以下几点：

由于是对广告进行排序，并且排序关键字根据广告的属性而定。因而必定需要有

Advertisement类用来存储广告信息，并且该类提供获取其属性值和设置其属性值的

方法。

由于在广告中有日期存储，并且日期需要比较先后顺序，因而建立类Date，其中有

年、月、日的属性，值得注意的是，在该类中需要重写<，以便日期之间进行比

较。

主要函数为mergeSort方法，该方法主要根据用户输入的关键字对广告进行排序，

并且输出时为广告编号并写入文件。该方法需要有三个参数，分别为：输入广告数

组，数组长度，关键字。

**五、问题解决**

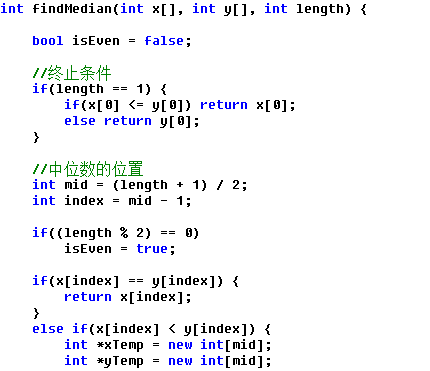
实验(1):

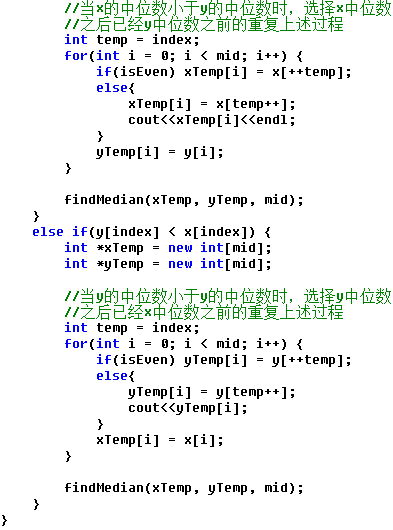
解决办法：首先判断两个数组的中位数是否相等，若相等，则直接返回结果；否则，

判断数组长度的奇偶性，然后根据奇偶性的不同，进入不同的条件语句，

反复递归调用，遇到终止条件后不断跳出递归，最终即可得出结果。

主要算法：





在该算法中，特别强调对于奇偶长度数组的不同处理，因为对于奇数长度的数组

而言，两个数组的中位数都有可能是所有数的中位数，因而在新建立的数组中仍然保

留；而对于偶数长度的数组而言，较小的中位数不可能成为所有数的中位数，因为其

比(n/2+1)个数都要小，因而该中位数不作保留，所以对于不同类型长度的数组，新数

组的赋值方式有所不同，因而分情况来处理。

需要注意的是，最终，两个数组都会不断减小问题规模直至数组长度为1，因而

该递归的终止条件是两个数组的长度为1。并且根据题中给出的例子，应当选取两个

数中较小的数作为中位数。

调试过程：

在第一次思考选取中位数的问题时，我的想法与上述过程大致一致，但是没有加

入判断奇偶的思想，属于不太成熟的想法。对应于上述代码，即是去掉所有有关布尔

变量isEven的语句即可。虽然代码可运行，但是不是最终我们想要的结果，于是我

认真思考了一下，取了其他的一些可能输入，然后发现若按照之前的想法，当数组为

偶数个时，重新建立的两个数组从数学角度来说长度是不一致的，就说明代码内部的

逻辑出现了问题。

于是我将方法参数中的length去掉，取而代之的是lengthX和lengthY，但是在后

续编代码的过程中发现这样使得代码的复杂性大大提高，而且加入了很多冗余工作。

最后通过分别对奇数和偶数个数组的仔细分析，发现各自的特性后，发现上一版

本的复杂性来源于我对多余数据的考虑，因而将不可能作为中位数的数据排除掉，发

现两个数组剩余的数据个数仍然相同，因而仍然采用length参数，但是在方法中加入

了布尔变量用于判断length的奇偶，在建立新数组的过程中也根据奇偶的不同来进行

不同的赋值，运行后成功。

实验(3)：

解决办法：该问题能否解决，主要取决于归并算法，因而在此我主要分析解决该问题

的归并算法的主要思想：

使用归并算法，是将问题从规模(设为s)为1一直合并直至问题规模为全部，因

而需要使用一个记录问题规模的变量，然后每次问题规模扩大为原来的二倍进行循

环，直至问题规模大于数组长度时循环结束。

在该循环内部，即为进行合并的函数。但是当两两合并时，会出现数组最后的剩

余个数不足两个等规模的s的大小或者不足一个规模s的大小。当出现第一种情况

时，仍然调用归并函数merge但是传入的参数需要有所变化；当出现第二种情况时，

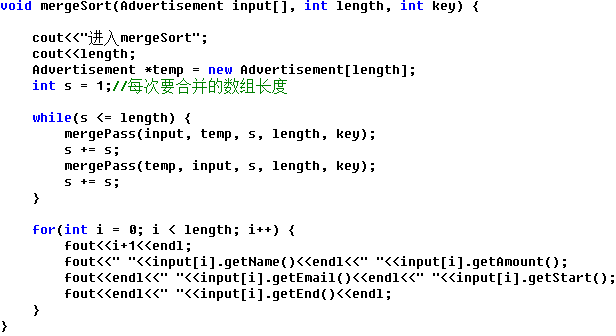
直接将剩余的数赋值到结果数组的相应位置。在归并函数merge内部，主要是对要合

并的两组数进行两两比较，假设有两个指针分别指向这两组数的起始位置，然后比较

该指针指向数的大小，将小的那个赋值给结果数组，并将指针后移，以此类推，直到

某一组数的结尾，然后将该组数中剩余的数赋值给结果数组。

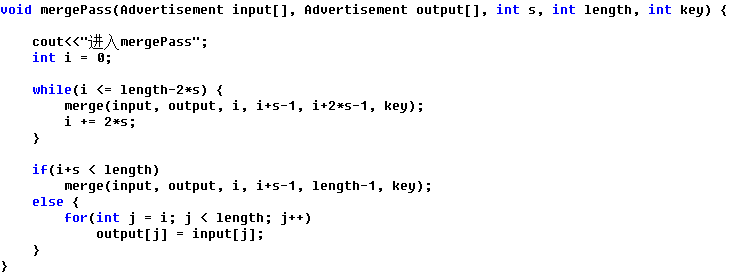
主要算法：



该函数的主要作用是控制问题规模的大小不断增大直至问题规模为全部，其中主

要调用控制合并的函数，最终结果会被存入仍会被存入input数组中，然后通过for

循环将排序后的每个广告信息编号输出到文件中。



该函数主要是为合并函数进行准备，面对不同情况，通过传入不同的参数来调用

合并函数。其中I变量用来存储每次要合并的两组数的第一个下标，然后在调用合并

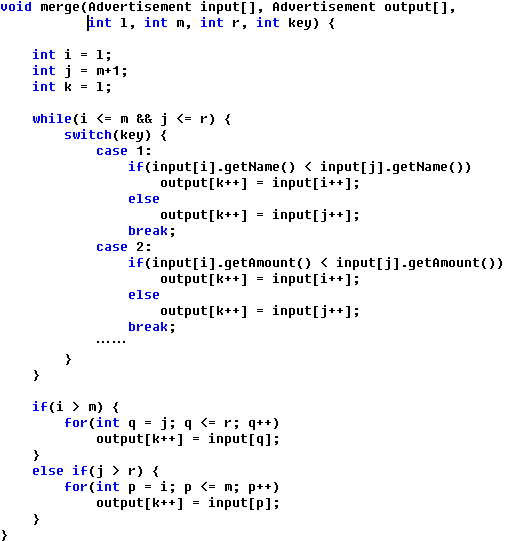
函数merge后该变量便增大两倍的问题规模s，直到i>length-2\*s，此时便退出该

while循环，此时i指向的是length-2\*s之后的某个数，那么如果i+s<length，说明当

前剩余的数的个数在s到2s之间，调用归并函数，传入的参数为left=i，Mid=i+s-

1,right=length-1；如果i+s>length，说明剩余个数小于s，那么直接将剩余的数拷贝到

结果数组中。



该函数的参数有输入数组，输出数组，合并的两组数的左下标，中间下标，右下

标，排序关键字。在该函数中，定义了三个变量，分别是i,j,k。i初始用来指向第一

组数的第一个位置，j初始用来指向第二组数的第一个位置，k初始用来指向结果数

组的第一个位置。进入while循环，匹配关键字，然后根据不同的关键字调用对广告

的不同属性比较大小，将小的放入结果数组后，指针右移，直至某一个指针到达最该

组数的右端，退出循环。退出循环后，判断是哪个指针越界，然后将另外一个指针指

向的数组中剩余的数直接全部赋值给结果数组，函数结束。

**六、实验结果总结**

回答以下问题：

1. 对不同的输入，该算法都存在哪几类可能出现的情况，你的测试数据完全覆盖了你所想到的这些情况，测试结果如何？

实验(1)：以下输入与结果相对应：

输入——>结果

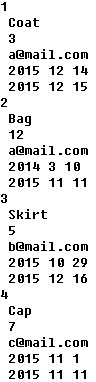
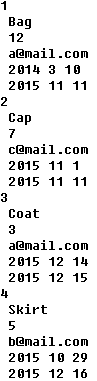
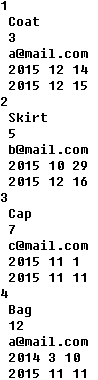
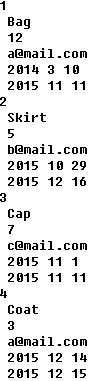
输入——>结果

输入——>结果

以上测试数据分别对应于数组大小为奇数、偶数以及较大数据的情况。

实验(3)：

当输入关键字为Start，Amount，Title，Email时，结果分别为：



1. 算法实现的复杂度在问题规模很大时可以接受吗？

可以。

1. 如果不用分治方法还能想到其他的解决方式吗？和分治相比会有更好的效率吗？

实验(1)：

对于找两个数组的中位数，可以间接通过排序算法来获得，具体方法

是：可以将两个数组合并为一个数组，然后对该数组进行排序，排序过程平

均需要O(nlogn)的时间，而该算法的时间复杂度为O(logn)，因而从效率上讲

还是使用分治法更为合适。

实验(3)：

排序方法千差万别，可以通过快排、插入排序、基数排序等很多中方法

进行排序，但是归并排序的算法复杂度无疑是最低的，只有O(nlogn)，因而

使用分治方法是最好的。

1. 所选用的数据结构合适吗？

实验(1)：

在本实验中对于两个数组的存储主要通过使用动态数组，因为数组的大

小在一开始无法确定，而是通过从输入文件读取获得的，因而需要动态的为

数组分配空间，这样避免了使用数组时长度的过小和过大问题，避免了不必

要的错误或者浪费不必要的空间，因而使用动态数组是合适的。

实验(3)：

使用vector作为存储广告的数据结构，因为vector的长度不固定，如果

使用数组的话对存储广告的个数就会有所限制，因而该数据结构是合适的。

对于日期数据专门建立了一个类Date，由于在本实验中涉及到了大量的

比较操作，因而对于日期类我重载了<运算符重载，使得日期之间具有可比

性，方便了题目中涉及到的比较操作，因而该数据结构是合适的。

1. 叙述通过实验你对分治方法的理解及你认为的分治法的优缺点。

分治方法能够将一个大的问题分解成为若干个互相独立的子问题，并且这些子问题与原问题具有相似的性质，最终将子问题的解合并即可得到原问题的解。

分治法能够将问题简单化，使用一种让人容易理解的方法来对问题进行求解，可读性强。但是实现分治的一般方法为递归，这就导致分治法的运行效率比较低，耗费的时间复杂度和空间复杂度都会比较大。

**实验二**

**一、实验目的**

1.能够理解动态规划的思想，并且能够判断出使用动态规划的条件(最有子结构和重叠

子问题)。

2.根据题意能够写出递归式，并能根据递归式进行编程。

**二、实验内容与实验步骤**

设有n种不同面值的硬币，各硬币的面值存于数组T[1:n]中。现要用这些面值的硬币来找钱，可以实用的各种面值的硬币个数不限。当只用硬币面值T[1],T[2],…,T[i]时，可找出钱数j的最少硬币个数记为C(i,j)。若只用这些硬币面值，找不出钱数j时，记C(i,j)=∞。

**三、实验环境**

操作系统：Windows 7

调试软件名称：Microsoft Visual C++ 6.0

**四、问题分析**

使用动态规划解决问题，是将问题从最小规模不断扩大直到得出最终解，因而首先

要确定问题应该如何划分。由题可得，问题即是对于n种面值的硬币找出待找零数

j所需要最少的硬币个数，而该问题的子问题则是j=1,2,…不断递增直到j=j-1为

止，由于具有最优子结构性质，因而每个子问题得出的结果必然是该子问题的最优

解，然后父问题由子问题叠加选出最优而来，以此类推，即可得出最终的最优结

果。

由上述分析，可以确定该问题的递归式：

设c(i,j)表示从第1个到第i个硬币可选，要找的钱数为j



使用该想法，时间复杂度应为O(n2)。

**五、问题解决**

解决办法：

由于题目的要求，在该算法中只能使用一个长度为L的数组，因而设数组

int\*result=new int[L]用来存储从1~L的待找零数分别对应的最少硬币数。首先对该数

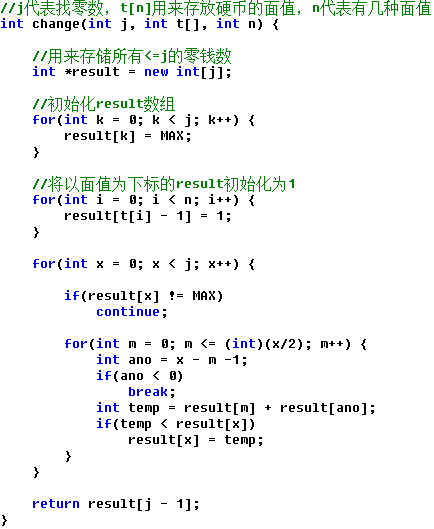
组进行初始化，对于数组下标为t[0~n-1]的数，初始化为1，说明对于该找零数，有

直接对应的面值数，因而只需要一个硬币即可。对于其它数组下标的数，初始化为宏

定义常量MAX以表示找不出待找零数。

初始化之后，通过迭代对数组进行填充。(x为数组下标)设置一个条件语句if(result[x] != MAX)判断是否进行此次迭代，意为若result[x]已被更改过，则说明其值已为当前子问题的最优解，那么跳过此次迭代，否则继续进行此次迭代。若要继续此次迭代，需要设一个临时变量temp用于存储result[m]+result[n]的值，其中m和n都满足0<m,n<x并且m+n+1=x，若temp< result[x]，则result[x]=temp，等到内部循环终止，result[x]中的解为最优。以此类推，最终得出result[L-1]即为该问题的最优解。

主要算法：



该算法中最多使用了两层for循环，因而时间复杂度为O(n2)。

调试过程：

由于上述算法比较简单，因而一次成型，没有涉及到调试。但是在此之前，由于没有看清题意，我使用了多个二维数组实现该算法，虽然也实现成功，但是却提高了复杂性。不过在算法编写过程中，后者使我对该问题的认识更加清晰，也对我的第二次编写有很大的帮助作用。

**六、实验结果总结**

回答以下问题：

1. 该方法实现的复杂度在问题规模很大时可以接受吗？

该算法的时间复杂度为n2，当规模很大时也可以接受。

1. 如果不用动态规划方法还能想到其他的解决方式吗？和动态规划相比会有更好的效率吗？

对于找零问题，可以使用贪心算法来进行求解，但是求得的最终解不一定是最优解，但是一定接近最优解，而且使用贪心的话效率不会太差，但是考虑到求取最优解的最好解法，我认为还是使用动态规划更为恰当。

1. 所选用的数据结构合适吗？

在该算法中，我使用动态数组来存储硬币面值，是因为硬币的种类在程序起始时无法确定，而是通过从文件中读取来获得的，因而需要动态分配数组大小。但是使用定长数组也是可以的，只要设一个足够大的长度就能满足，但是这样会造成空间浪费。因而我认为使用动态数组是合适的。

1. 该算法都存在哪几类可能出现的情况，你的测试完全覆盖了你所想到的这些情况吗，测试结果如何？

输入——>输出

输入——>输出

1. 叙述通过实验你对动态规划方法的理解及其优缺点

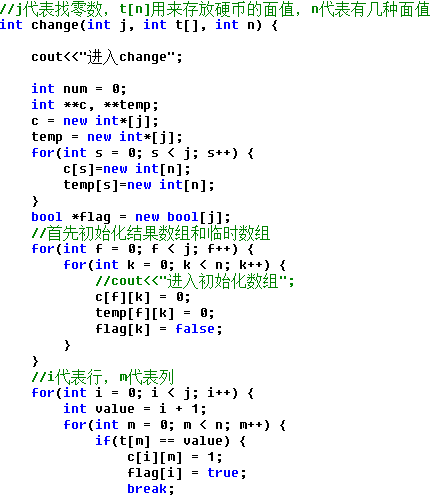
动态规划与分治法类似但又不完全相同，它们的基本思想都是将原问题分解成若干子问题，对子问题进行求解，最后将子问题的解合并成原问题的解。但是分治法中的子问题是相互独立并且问题类型相同的，而动态规划中的子问题是具有重叠性质的。

动态规划适用于对最优解进行求解，写出递归式之后会对整个问题的思路更加清晰，但是其时间复杂度很高，是一种低效率的算法。

**六、附录**

对于动态规划求解找零钱问题，我还使用了一种较为复杂的办法进行求解：利用二维数组进行求解。对于这种方法，是对于动态规划的二维数组解法的一种重现，该算法的时间复杂度和空间复杂度都不如使用一维数组的情况。

该算法代码如下：(正确性已经得到验证)





**实验三**

**实验目的：**

1.理解回溯法的思想，能够理清思路，使用递归或迭代方式进行回溯求解。

2.掌握回溯函数的编写技巧，务必在函数起始位置加上终止条件。

3.了解回溯法的使用范围，当需要求出最优子集或者最优排序等时考虑回溯。

**实验内容：**

装载问题：有一批共n个集装箱要装上2艘载重量分别为c1和c2的轮船，其中集装箱i的重量为wi ，且 ，要求确定是否有一个合理的装载方案可将这n个集装箱装上这2艘轮船。如果有，请给出该方案。

**实验步骤：**

1.分析要解决的问题，给出你的思路，可以借助图表等辅助表达。

虽说该装载问题最终的结果是求出如何装上两艘船，但实际上可以等价于：如何尽可能多的将物品放在c1船上，这样整个问题便可解决。因而该装载问题属于求出最优子集的问题，所以可以通过使用回溯法可以得出最优解。

首先可以画出一颗递归树，每个节点代表当前的状态，然后左分支代表选，右分支代表不选。到这一步为止，可以勾勒出一颗满二叉树，然后通过比较至根节点时c1船上的重量得出最优解。但是该过程冗余复杂，导致时空复杂度很高，而且有很多的分支没有必要进行，因而需要通过剪枝函数和上界函数来控制算法的时空复杂度，去掉一些不必要的计算。

剪枝函数设计为：在放入第i个物品之前，首先判断当前载重量+i的重量<船的容量，如果成立，则将左支剪掉。该方法仅在搜索左子树时用到，因为前提是放入物品。

上界函数设计为：在选择不放入第i个物品之前，首先判断，当前载重量+剩余载重量>当前最优载重量，如果成立，才继续搜索右支，否则是在右支不可能找到比当前最优解更好的解的。

2.分析利用你的想法解决该问题可能会有怎样的时空复杂度。

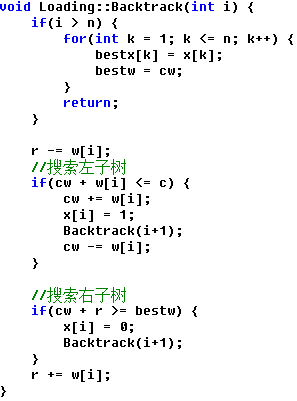
最差时间复杂度为O(n2n)

3.根据对问题的分析，写出解决办法。

我使用递归的方法进行求解。递归算法的优点是比较简单易懂，易编写，其缺点是时空复杂度很大。

由于整个结构可以画为一颗二叉递归树，然后每一步判断选与不选，如果选入后满足条件，则继续向下深度搜索；否则回到上一步走另外一条路。在判断选与不选的过程中，需要利用上述剪枝函数与上界函数。

4.描述你在进行实现时，主要的函数或操作内部的主要算法；分析这个算法的时、空复杂度，并说明你设计的巧妙之处，如有创新，将其清晰的表述。



(设传入参数为i)该方法为解决该问题的回溯算法。当进入树的第i+1前，剩余重量要减去该物品的重量(无论是搜索左树还是搜索右树)。然后判断当前重量+该物品的重量<船的容量，若是，则更新当前承重，然后调用递归函数搜索左树，当递归结束时，应该回到上一层，即当前承重-放入物品的重量以便递归其右子树。在搜索右子树之前，应先判断是否有必要对右子树进行搜索，即判断当前承重+剩余重量<当前最优承重，若是则说明搜索右子树得不到比当前最优解更好的解，否则，继续调用回溯算法。最后，将剩余重量加上第i个物品的重量，一次递归结束，返回调用该递归函数的函数。

需要注意的是，在函数的起始位置，必须设有递归结束条件。在上述描述的算法中，若能到达叶子节点，则叶子节点就是当前最优解，因而只要i>n，即说明已到达叶子节点，那么此时更新最优解和最优值，然后return返回上级调用。

**实验结果：**

1. 对不同的输入，该算法都存在哪几类可能出现的情况，你的测试数据完全覆盖了你所想到的这些情况，测试结果如何？

输入——>输出

输入——>输出

以上两种结果分别为测试较小规模数据和较大规模数据时的情况。

1. 算法实现的复杂度在问题规模很大时可以接受吗？

不可以。因为回溯算法是一种穷举式的搜索，虽然会通过上界函数和约束函数对回溯法的搜索范围进行约束，但是时间复杂度仍然会达到指数级，不适合规模太大的输入。

1. 如果使用动态规划会比回溯法好吗？

我认为动态规划和回溯法各有利弊。

回溯法相对于动态规划来说，程序的可读性更强，更容易理解。但是从复杂度的角度来说，回溯法的复杂度往往都是指数级，相对于动态规划来说，其复杂度更大。

1. 所选用的数据结构合适吗？

在该算法中，我使用动态数组对货物的重量进行存储，是因为程序在编译时并不能确定数组的大小，而是在文件中读取数组的大小，因而需要在运行的时候动态地为该数组分配空间，并且使用数组时随机存取很快，因而使用该数据结构是合适的。

**实验总结：**

叙述通过实验你对回溯方法的理解及你认为的回溯法的优缺点。

回溯法的基本思想是：进行能避免不必要搜索的穷举式搜索法。它主要是使用深

度优先的方式对解空间树进行系统地搜索，是用于解一些组合数相当大的问题。

回溯法的优点是其算法容易理解，它可以系统地搜索一个问题的所有或任意解，

既有系统性，又有跳跃性。而且由于往往是对解空间树进行搜索，因而可以通过一些

剪枝函数剪掉那些不满足约束或者得不到最优解的子树，同时在搜索过程中是动态地

产生问题的解空间，在任何时刻，算法只保存从根节点到当前扩展节点的路径，因而

回溯法所需的计算空间通常为树的高度。

但是回溯法的复杂度很高。其空间复杂度的复杂性主要体现在回溯法对堆栈空间

的利用很大，是因为它不断的进入回溯函数而不返回，可能导致堆栈空间溢出。其时

间复杂度的的复杂性主要体现在它的穷举式搜索，搜索的时间复杂度往往能达到指数

级。