数组、链表、树代码题总结——一休

如果文档中有什么问题可以找王道班主任或者直接找我。

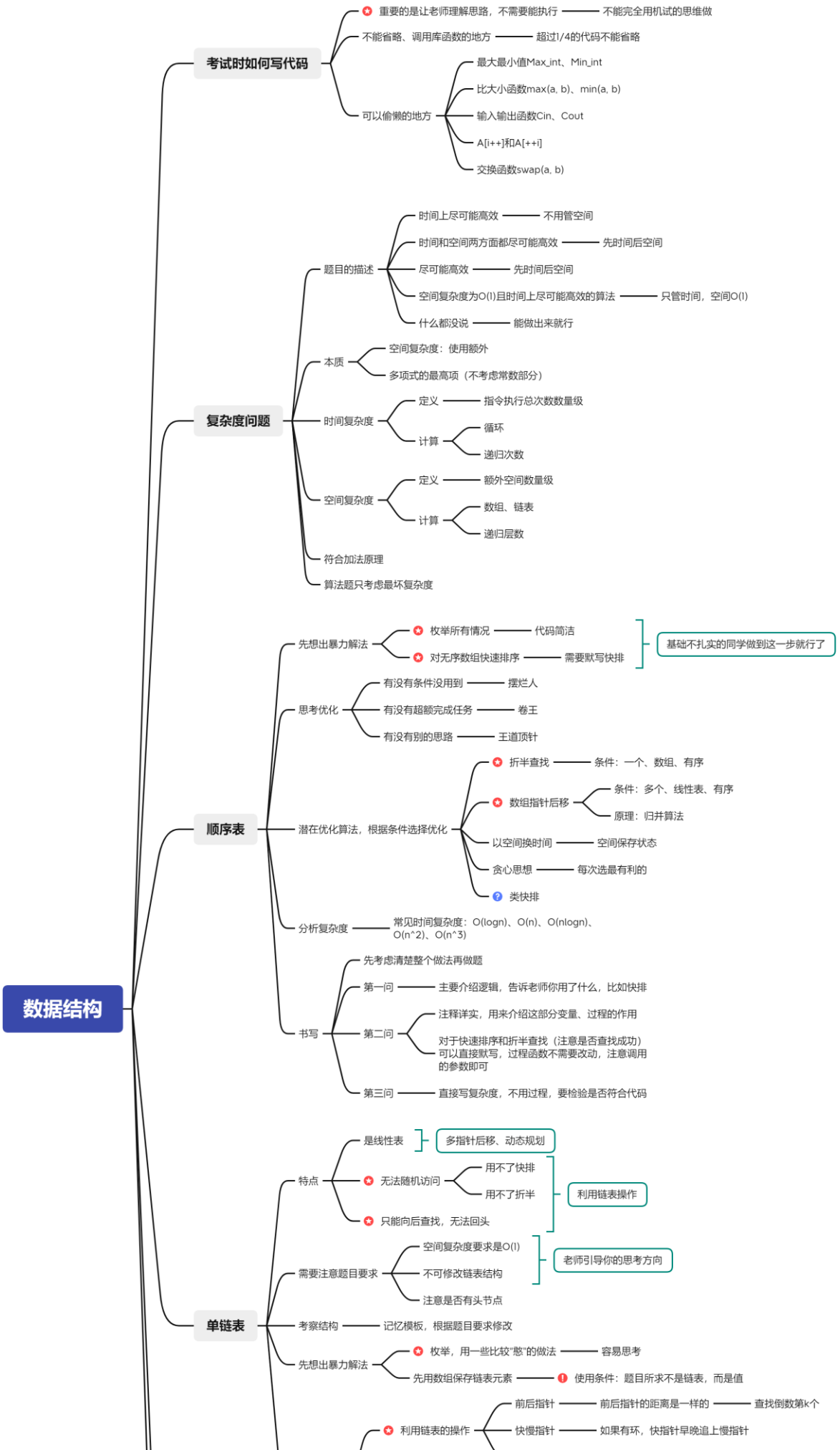
本来我想讲的东西很多，希望能够把所有可能用到的东西都展示给大家（比如引用，如果理解不了，考试中怎么通过其他方法避免使用引用），但是时间紧张，之后会考虑制作对应课程包，不同的同学想听的东西不同，大家有什么建议以及这几次课程的评价都可

以填问卷或者直接告诉我或班主任，感谢大家的支持，希望你们能学会这种分析做题的方法，那代码题拿分甚至拿满分都是没问题。数据结构其他部分每一小块我也在做总结，但是这个很难做，你们可以去看一下习题视频里面的红黑树和并查集部分，希望可以帮助大家复习。

【腾讯文档】算法大题课程反馈及建议

https://docs.qq.com/form/page/DRXBpY0xJa0RtZUtT

0 思维导图


数据结构
考试时如何写代码
重要的是让老师理解思路，不需要能执行
不能完全用机试的思维做
不能省略、调用库函数的地方
超过1/4的代码不能省略
可以偷懒的地方
最大最小值Max_int、Min_int
比大小函数max(a, b)、min(a, b)
输入输出函数Cin、Cout
A[i++]和A[++i]
交换函数swap(a, b)
复杂度问题
题目的描述
时间上尽可能高效
不用管空间
时间和空间两方面都尽可能高效
先时间后空间
尽可能高效
先时间后空间
空间复杂度为O(1)且时间上尽可能高效的算法
只管时间，空间O(1)
什么都没说
能做出来就行
本质
空间复杂度：使用额外
多项式的最高项（不考虑常数部分）
时间复杂度
定义
指令执行总次数数量级
计算
循环
递归次数
空间复杂度
定义
额外空间数量级
计算
数组、链表
递归层数
符合加法原理
算法题只考虑最坏复杂度
顺序表
先想出暴力解法
枚举所有情况
代码简洁
对无序数组快速排序
需要默写快排
基础不扎实的同学做到这一步就行了
思考优化
有没有条件没用到
摆烂人
有没有超额完成任务
卷王
有没有别的思路
王道顶针
潜在优化算法，根据条件选择优化
折半查找
条件：一个、数组、有序
数组指针后移
条件：多个、线性表、有序
原理：归并算法
以空间换时间
空间保存状态
贪心思想
每次选最有利的
类快排
分析复杂度
常见时间复杂度：O(logn)、O(n)、O(nlogn)、O(n^2)、O(n^3)
书写
先考虑清楚整个做法再做题
第一问
主要介绍逻辑，告诉老师你用了什么，比如快排
第二问
注释详实，用来介绍这部分变量、过程的作用
对于快速排序和折半查找（注意是否查找成功）可以直接默写，过程函数不需要改动，注意调用的参数即可
第三问
直接写复杂度，不用过程，要检验是否符合代码
单链表
特点
是线性表
多指针后移、动态规划
无法随机访问
用不了快排
用不了折半
利用链表操作
只能向后查找，无法回头
需要注意题目要求
空间复杂度要求是O(1)
不可修改链表结构
老师引导你的思考方向
注意是否有头节点
考察结构
记忆模板，根据题目要求修改
先想出暴力解法
枚举，用一些比较"憨"的做法
容易思考
先用数组保存链表元素
使用条件：题目所求不是链表，而是值
利用链表的操作
前后指针
前后指针的距离是一样的
查找倒数第k个
快慢指针
如果有环，快指针早晚追上慢指针

1 考试时如何写代码

考试中的注释也是必不可少的，因为重要的是让老师理解我们的思路，所以在关键位置处写注释有助于老师阅读。①在变量的定义处告知该变量是做什么的；②在使用伪代码处和函数调用处告知这部分实现哪些功能；③代码逻辑复杂的地方告知是做什么的。

很多简单的代码、库函数写起来浪费时间，考试时简写或者写个系统调用即可，只要让老师知道你想干嘛即可，但重要的代码一定不能这样，否则是 0 分。比如说本题就是考察快速排序，你直接调用 Qsort 不把快排过程写出来，当然拿不到分。那如何判断？只要这个东西的代码有可能超过你的所有代码 1/4（其实就是他很重要），就一定要写。

给大家总结了一些可以偷懒节省时间的地方，考试时简写即可，使用的时候写清楚他的功能即可。

1.1 最大最小值 Max_int、Min_int

从名字可以看出来，Max_int 是最大的整数值，Min_int 是最小的整数值（同理，float、double、unsigned 也可以这样定义），他常用于比较选择最大最小值时，比如初值设为 Max_int，那遇到任意值都会比他小或者相等，这样只要遇到比他小的就更新为更小的值。

```
int D_min=Max_int;              //D_min 初始设为最大的整数
for (int i=0; i<n; i++)         //遍历 D 循环选出最小的 D
   if (D[i]<D_min)
       D_min=D[i];              //D[i]比 D_min 小，D_min 更新
```

1.2 比大小函数 max(a, b)、min(a, b)

从名字可以看出来，这两个函数是为了得到 a 和 b 中的最小值和最大值

函数定义如下（不用掌握）：

```
int max(int a, b){
   if(a>b)
      return a;
   else return b;
}
int min(int a, int b){
   if(a<b)
      return a;
   else return b;
}
```

使用这两个函数能让我们的代码变得简洁、清晰：

```
int D_min=Max_int;              //D_min 初始设为最大的整数
for (int i=0; i<n; i++)         //遍历 D 循环选出最小的 D
   D_min=min(D_min, D[i]);      //D_min=D_min 和 D[i]中的较小值
```

1.3 输入输出函数 cin、cout

C 语言中的输入、输出函数分别是 scanf 和 printf 函数，因为和变量的类型有关，写起来比较麻烦，在考试中可以用 C++，那我们可以使用 C++的 cin、cout 函数来输入、输出，而不用管变量的类型，会简洁很多。

```
cin>>A[0];                      //读入 A[0]，cin 是左箭头
cin>>b>>c;                      //读入 b、c
cout<<A[0];                     //输出 a
cout<<b<<c<<endl;               //输出 b、c，并输出回车
scanf("%d", A);                //读入 A[0]
scanf("%d", &b);                //读入 b
```

使用 cin、cout 函数不用管是不是要加&，也不用管他是什么类型的变量，可以节省考试时间，也减少出错可能：

```
int D_min=Max_int;              //D_min 初始设为最大的整数
for (int i=0; i<n; i++)         //遍历 D 循环选出最小的 D
   D_min=min(D_min, D[i]);      //D_min=D_min 和 D[i]中的较小值
for (int i=0; i<n; i++)
   cout<<A[i]<<" ";             //循环输出数组 A 以及空格
```

当然还有一种更简单的写法，写中文。比如直接写输出数组 A、输入数组 A：

```
int D_min=Max_int;              //D_min 初始设为最大的整数
for (int i=0; i<n; i++)         //遍历 D 循环选出最小的 D
   D_min=min(D_min, D[i]);      //D_min=D_min 和 D[i]中的较小值
for (int i=0; i<n; i++)
   输出数组 A
```

1.4 A[i++]和 A[++i]

这两个写法是不一样的，A[i++]中 i 在前表示先使用 i，再 i++；A[++i]中先 i++，再使用 A[i]（此时 i 是 i++后的结果）。这两种写法相当于两步缩写成了一步，如果记不住还是拆开写成两步稳妥。

比如对于 cout<<A[i++]，相当于先用再加：

```
cout<<A[i];                    //输出 A[i]
i++;                           //i 自增
```

比如对于 cout<<A[++i]，相当于先加再用：

```
i++;                           //i 自增
cout<<A[i];                    //输出 A[i]
```

1.5 交换函数 swap(a, b)

函数定义如下（不用掌握）：

```
int swap(int *a, *b){
    int temp=a;
    a=b;
    b=temp;
}
```

让我们看看这两种不同的写法，①不使用 swap 函数：

```
for(int i=n-1; i>1; i--)
   for (int j=0; j<i; j++)          //冒泡排序
      if (A[i]<A[j]){               //交换 A[i]和 A[j]
          temp=A[i];
          A[i]=A[j];
          A[j]=temp;
       }
```

②使用 swap 函数：

```
for(int i=n-1; i>1; i--)
   for (int j=0; j<i; j++)          //冒泡排序
if (A[i]<A[j]){                 //交换 A[i]和 A[j]
          swap(A[i], A[j]);          //交换 A[i]和 A[j]
```

2 复杂度问题

如无特殊说明，数据结构大题中问的时间和空间复杂度都是最坏情况下的复杂度，只有一个例外：快速排序（考虑平均复杂度，或者给快速排序增加一种优化）。

在计算机学科中，$\log_2 n$ 常常写作 logn，他们都是以 2 为底的对数运算。

2.1 时间复杂度

时间复杂度是对时间增长速度的一个估计。时间复杂度为 O(f(n))表示该算法执行的指令数的上界是 O(f(n))级别的，可以简单的用极限理解，若所有指令总执行次数 T，时间复杂度为 O(f(n))的算

法表示$\lim_{x\to\infty}\frac{T}{f(n)}$=K（K 为常数）。

例如 T1=n^2+n，T2=2n^2-n，通过极限的思想得到这两个算法的 f(n)=n 即时间复杂度都是 O(n)，常数 K1=1，K2=2，K1≠K2，但复杂度是一样的，所以常数系数不影响复杂度；本例中时间复杂度只有最高次决定，所以较低次项不影响复杂度。总之我们就是需要查看算法的指令执行次数 T 的最高次项（而不考虑他的系数）。

O($\log_2 n$)和 O($\log_2 n$)其实是相等的，因为 $\log_2 n=\log_2 3*\log_3 n$，而 $\log_2 3$ 是常数，所以 O($\log_2 n$)=O($\log_2 n$)，考虑复杂度时，所有的 log 都尽量写成 $\log_2 n$ 或者 logn 的形式。

对于考试时写的代码，时间复杂度只需要考虑**循环**和**递归次数**。

循环次数

建议尽量写 for 循环，所有循环都可以写成 for 循环

1) 对于多层 for 循环，如果每一层的变量都是自增 1，则直接把每层执行次数相乘

```
for (i=0;i<n;i++)
    for (j=0;j<n;j++)
        for (k=0;k<n;k++)
            if (A[i][j]>A[i][k]+A[k][j])
                    A[i][j]>A[i][k]+A[k][j];
```

他的每一层都是变量自增，且变量都是从 0~n-1 共 n 次，所以总次数是 n^3，时间复杂度是 O(n^3)。

2) 对于多层 for 循环，如果每一层的变量都是自增 1，但第二三层变量取值范围和第一层变量有关

```
for (i=0;i<n;i++)
    for (j=0;j<i;j++)
        for (k=0;k<j;k++)
            if (A[i][j]>A[i][k]+A[k][j])
                    A[i][j]>A[i][k]+A[k][j];
```

时间复杂度和第①种一样，但不用考虑具体次数，时间复杂度是 O(n^3)。

3) 对于两层循环，变量不是自增 1（这种情况主要针对小题）

```
for (i=1;i<n;i*=2)                    //22 考研真题
    for (j=0;j<i;j++)
        sum++;
```

需要讨论第一层变量 i 取不同值时，j 的取值有 x 种，然后对 x 求和。比如本题，i=1, 2, 4, 8…2^k(2^k<n≤2^{k+1})时，第二层 j 有 i 个取值，所以总次数是对 i 求和，即 T=1+2+4+2^k=2^{k+1}-1，n<T<2n，时间复杂度就是 O(n)。

递归次数

递归总次数和时间复杂度有关，而最大递归层数和空间复杂度有关。

对于树中序遍历的递归写法，每个节点都会调用一次所以总共调用次数是 O(n)级别的，时间复杂度是 O(n)，而递归使用的层数最多是树高 h，空间复杂度是 O(h)。

```
void inorder(BTNode *p){
    if (p==NULL) return;
    inorder(p->lchild);
    visit(p);                                    //对 p 节点的访问等
    inorder(p->rchild);
}
```

2.2 空间复杂度

只算额外空间开销，不考虑题目中已经给出的数据所占的空间，比如题目中给出了一个数组，这个数组是不计入空间复杂度的。

定义的数组、链表

单个定义的变量都不需要考虑，但定义的数组中元素个数以及链表中节点个数都要计入空间复杂度中。

递归层数

是在递归栈中最多出现多少个过程函数，层数和总次数比较见时间复杂度的递归次数。

2.3 注意题目关于复杂度的要求

共有五种情况：

1) 时间上尽可能高效：表示只要求时间复杂度，对空间复杂度不做要求
2) 时间和空间两方面都尽可能高效：时间和空间复杂度都要求尽量小，时间复杂度小优先。
3) 尽可能高效：要求同（2)。
4) 空间复杂度为 O(1)且时间上尽可能高效的算法：要求算法的空间复杂度是 O(1)，时间复杂度尽可能小。
5) 没有描述：只要能做出来就行，不要求复杂度是多少，常见于树中

3 顺序表

推荐流程：想出暴力解法->分析暴力解法复杂度->在暴力解法基础上思考优化

暴力解法是最容易想到的做法，思维难度小，复杂度高（主要是时间复杂度），通常情况下都不会是最优解，比如这题满分是 15 分，下面是比较：

	所花时间	得分
暴力解法	5 分钟	11 分
最优解	15 分钟，不一定能做出来	15 分

如果我们孤注一掷的求最优解可能最后连暴力解的分都拿不到，做过真题的同学会发现有一些年份的最优解是比较难的，在考试中很难做出来，所以不管怎么样至少都要拿到暴力解法的分数。

①对于基础不够好的同学目标就是做出暴力，如果正好自己能够做出来更优解就写，以节省时间为主。②对于基础牢固的同学，目标应该是先想出暴力解放有个保底分，然后在暴力的基础上做出改进优化后的算法，即保暴力分，冲满分。③对于经常刷机试题、算法扎实的同学目标就应该是最优解，但如果实在想不到，还是应该写暴力解。

如果我们不满足于暴力解法，就应该想，暴力解法浪费了什么，我们在什么地方可以对他优化，优化一般要使时间或者空间复杂度减小（主要是时间复杂度），比如说题目给的是有序数组，但是我们没有用到有序性，这是有条件没用；或者题目不需要排序，只要求中位数，但是我们对他排序了，超额完成任务，本来可以不做这么多的，这就是浪费。最好的情况就是我们完美的利用了题目的条件，而且又不多做一丁点事，这样的算法一般都更优秀。

3.1 暴力解法

枚举法

枚举法是最容易想到的方法，把所有可能的情况都考虑到，然后从中选出符合题目要求的情况，通常使用 for 循环遍历。

对无序数组排序

对于一个无序的数组可以先通过排序把他变成有序再处理，排序使用快速排序。考试中直接默写快速排序（注意这是升序！），然后注意调用快速排序时的参数即可。

快速排序是一种分治思想，每一轮快排分为两个步骤：①选择枢值 key。②枢值移动到他的最终位置，左右划分为两个区间，左边区间的所有元素都小于枢值，右边区间的所有元素都大于枢值。然后对左右区间分别进行快排，不断重复直到当前处理区间元素个数小于等于 1（即 L>=R）。

快速排序的平均时间复杂度是 O(nlogn)，平均空间复杂度是 O(logn)，是考试中最快的不稳定排序算法，一般要用到排序时都使用快速排序。快速排序的最坏时间复杂度是 $O(n^2)$，最坏空间复杂度都是 O(n)，但我们只需要加一个小优化就能避免最坏情况：即随机选择一个元素作为枢值。优化后最坏时间复杂度 O(nlogn)，最坏空间复杂度 O(logn)。

（注：快速排序有很多写法，交换式和挖坑式，不同写法中间过程不一样但只要能实现排序即可，本代码适用于算法大题，方便记忆）

代码如下：

```
void Qsort(int A[], L, R){                    //a 数组保存数据，L 和 R 是边界
    if (L>=R) return;                        //当前区间元素个数<=1 则退出
    int pivot, i=L, j=R;                     //i 和 j 是左右两个数组下标移动
    把 A[L~R]中随机一个元素和 A[L]交换          //快排优化，使得基准值的选取随机
    pivot=A[L];                              //pivot 作为基准值参与比较
    while (i<j){
        while (i<j && A[j]>pivot)
            j--;
        while (i<j && A[i]<=pivot)
            i++;R
        if (i<j)
            swap(A[i], A[j]);                //交换 A[i]和 A[j]
    }
    swap(A[L], A[i]);                        //将基准值放入他的最终位置
     /*此时 A[L~i-1]<=A[i]<=A[i+1~R]*/
    Qsort(A, L, i-1);                        //递归处理左区间
    Qsort(A, i+1, R);                        //递归处理右区间
}
```

3.2 思考优化的方向

主要是三个方向：①有没有条件没有使用到，如有序性，那如何利用这个条件？②有没有超额完成题目中没有要求的任务，比如题目要求求最小值，但我们却将他排序了，这是没有必要的，那如何不做这个任务？③除了暴力的思考方向，还有没有别的方向？

3.3 进阶需要掌握的算法

利用有序性——折半查找

在线性表中如果需要查找权值为 x 的元素，时间复杂度是 O(n)，也就是说不管什么线性表，查找元素的复杂度下界是 O(n)。如果要实现查找的时间复杂度是 O(logn)，因为待查找元素可能出现在任意位置，那需要每次查找都能够排除一半情况，这就是折半查找的逻辑。折半查找前提：①数列必须是有序的，升序或者降序；②只能是顺序表（数组），不能是链表。

考试中当我们看到出现了一个有序数组（如果是链表一定不能使用）的时候，首先想想是否需要查找，能不能使用折半查找，这是经常考察的点，2011、2018、2020 年都可以使用折半查找。

折半查找，也叫二分查找，假设我们在升序数组 A[L~R]中查找 x，L 和 R 是上下界（即 Left 和 Right），mid=(L+R)/2，每次把 x 与 A[mid]比较：如果 x>A[mid]，说明 x 一定不会出现在 A[L~mid]，只可能出现在 A[mid+1~R]，更新 L=mid+1；而如果 x<=A[mid]，说明 x 一定不会出现在 A[mid+1~R]，只可能出现在 A[L~mid]，更新 R=mid。重复比较——更新过程，直到 L==R，此时 A[L]就是我们要找的元素，如果 A[L]不等于 x 说明 x 不在数组 A 中。

（注意：①书写折半查找，最怕当只有两个元素的出现死循环，书写完后一定要带入只有两、三个元素的时候试一下是否有问题；②如果是题目中返回查找成功时的下标可以直接使用这段代码，否则需要调整最后一个 if 语句。）

代码如下：

```
int Binary_Search(int A[], L, R, x){ //A[]和 x 不一定是 int 型
    int mid;
    while (L<R){                          //如果 L>R 则范围错误
        mid=(L+R)/2;                      //mid 取中间数，向下取整
        if (x<=A[mid])
            R=mid;
        else L=mid+1;                  //更新查找范围
    }
    if (A[L]==x) return L;                //查找成功，返回数组下标 L
        else return -1;                          //查找失败
}
```

时间复杂度为 O(logn)，空间复杂度为 O(1)。

设置多指针后移

多指针后移常用于有序线性表（顺序表和链表都可以），如果考试中给出有序的线性表，优先考虑这种方式，这种方式可以用于合并多个有序线性表、查找第 k 个元素等等，归并排序就是用到了这种思想。

归并两个升序数组（理解+应用），代码如下：

```
int C[n+m];                          /*我习惯将新数组设为全局变量*/
void Merge(int A[], n, B[], m){  //数组 A、B 长度分别为 n、m
  int i=j=k=0;                  //i、j 作为遍历 A、B 的下标
  while (i<n && j<m)             //直到有一个数组遍历完
    if (A[i]<B[j])              //将小的那个数存入 C 数组
       C[k++]=A[i++];
    else C[k++]=B[j++];
  for (; i<n; i++)
    C[k++]=A[i];                //将 A 中剩余的数存入 C 数组
  for (; j<m; j++)
    C[k++]=B[j];                //将 B 中剩余的数存入 C 数组
}
```

例题：

定义距离 d = |x – y|，给定 2 个长度为 n 的升序数组 A、B，x 是数组 A 中的某个元素，y 是数组 B 中的某个元素。请设计一个尽可能高效的算法，计算并输出所有可能的的最小距离 D。例如 数组 A = {–16, –8, 5, 8, 13}，B ={-2, 0, 2, 6, 10}，则最小距离为 1，相应的 x=5, y=6, d= |5 – 6|=1。

这是 2 个升序数组，查找一种最小的情况，可以考虑使用指针后移，设置两个指针（实际上是两个 int 变量）i 和 j 保存此时正在处理的数组 A、B 元素的下标，每次只比较 a[i]和 b[j]，数组 A 和 B 都是升序，每次 i 或 j 后移都会导致 x 或 y 变大，最终要求的是最小的 d，所以我们的每次选择 i 还是 j 后移的时候原则就是尽量不让 d 变大，即要让 a[i]和 b[j]中较小的那个后移。初始时 i=j=0, x=-16, y=-2, d=14，a[i]小，应该 i 后移即 i++，这样会缩小 x 和 y 的差距，而如果 j 后移会继续拉大 x 和 y 的差距。①如果 i++，则 x=-8, y=-2，d=6；②如果 j++，则 x=-16, y=-0，d=16。我们的目的是求出最小的 d，所以②这种情况是没意义的，选①情况，每次比较完 x 和 y 后把较小的那个值指针后移。最终每一个数组都只会遍历一次，不会回头，所以时间复杂度是 O(n)。

3.4 分析复杂度

分析所用算法的时间和空间复杂度，常见时间复杂度：O(logn)、O(n)、O(nlogn)、$O(n^2)$、$O(n^3)$。到这一步我们仍然只是思考还未进行书写，下一步开始书写。

3.5 书写

先思考清楚整个算法需要做哪些事情已经对应流程，分析完复杂度再开始写。

第一小问：设计思想

主要介绍逻辑，告诉老师你用了什么，比如快速排序，重要的是让老师知道用的方法是什么。

第二小问：描述算法

注释要详实，用来介绍变量、过程的作用，提示老师。对于快速排序和折半查找（注意是否查找成功）可以直接默写，过程函数不需要改动，注意调用的参数即可。

第三小问：求时间和空间复杂度

直接写复杂度，不用过程，要检验是否符合代码。

4 链表

代码题中的链表一般都是单链表，我们复习时就按照单链表复习即可，如果万一考到双链表，只需要知道他和单链表最大的不同就是双链表可以向前和向后查找，而单链表只能向后查找。

做题推荐流程：和顺序表一样，想出暴力解法->分析暴力解法复杂度->在暴力解法基础上思考优化，但在暴力解和优化的方向上也有不同。

4.1 需要注意的地方

4.1.1 特点

1) 是线性表。线性表能用的方法他都能用，比如归并两个有序序列，所以数组指针后移的方法也可以用到链表中。
2) 无法随机访问。如果一个方法需要利用随机访问的特性，那就不能用在链表上，比如说折半查找和快速排序。
3) 只能向后查找，无法回头。比如要求倒数第 k 个结点，无法从最后一个结点往前查，得转化成正数第 x 个结点，从前往后查，或者采用前后指针固定间距。

4.1.2 题目的要求

1) 如果题目要求**空间复杂度是 O(1)**，这就意味着不能额外设置数组，不能将链表先用数组保存然后再进行处理。
2) 如果题目要求**不可修改链表结构**，意思是我们不能随意的对链表元素顺序进行修改以及随意的插入和删除。

4.1.3 考察结构

题目中经常会有如下描述：

2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释。

我们可以提前记忆模板，然后考试中根据题目要求进行改动（**仔细审题**）：

单链表（data 是该节点的权值，一般可以定义成 int；next 是指向下一个节点的指针，保存的是下一个节点的位置）：

```
typedef struct LNode{
    Elemtype data;                  //该节点的权值
    struct LNode *next;             //next 指向下一个节点
}LNode;
```

双链表（不需要特别记忆，只是比单链表多一个 prior 指针而已）：

```
typedef struct LNode{
    Elemtype data;                  //该节点的权值
    struct LNode *prior,*next;      //prior 指向上一个节点
}LNode;
```

4.2 暴力解法

枚举法，直接处理

链表的一些题目可以先采用最直接易想的方法，比如要改变链表元素顺序的话，可以将需要重排的元素一个一个拆下来重新插入。这些方法一般看起来比较憨，但是容易想到。

将链表用数组保存，再处理

可以使用数组保存链表中的结点地址或者直接保存数据，然后再按照数组的做题方法操作即可。注意：①如果题目中求的是处理之后的链表；②如果题目要求空间复杂度为 O(1)；这两种情况都不能使用数组保存链表的结点。

4.3 思考优化的方向

主要是三个方向：①有没有条件没有使用到，如有序性，那如何利用这个条件？②有没有超额完成题目中没有要求的任务，比如题目要求求最小值，但我们却将他排序了，这是没有必要的，那如何不做这个任务？③除了暴力的思考方向，还有没有别的方向？

4.4 适用于链表的优化算法

设置多指针后移

类似于前面的数组多指针后移，在链表中也可以使用这种方法，在数组中每次后移是 i++，而链表中则是 p=p->next，本质上都是一样的。

归并两个带头节点的升序链表（理解+应用），代码如下：

```
void Merge(LNode *L1, *L2){          //数组 A、B 长度分别为 n、m
  LNode *p=L1->next, *q=L2->next;   //p、q 指向 L1、L2 第一个元素
  LNode *r=L1;                      //新链表头节点为 L1，r 指向末尾
  LNode *pn, *qn;                   //用来暂存 p->next 和 q->next
  while (p!=null && q!=null)        //直到有一个链表遍历完
    if (p->data<q->data){           //将小的那个数存入新链
       pn=p->next;                  //pn 为 p 下一个元素
       r->next=p;                   //p 插入到 r 后面
       p->next=null;                //这是新链最后一个元素
       r=p;                         //尾指针 r 指向最后一个元素
       p=pn;                        //p 指向 p 下一个元素
    }
    else{
       qn=q->next;                  //qn 为 q 下一个元素
       r->next=q;                   //q 插入到 r 后面
       q->next=null;                //这是新链最后一个元素
       r=q;                         //尾指针 r 指向最后一个元素
       q=qn;                        //q 指向 q 下一个元素
    }
  if (p!=null)
    r->next=p;                      //将剩余部分连到 r 后面
  if (p!=null)
    r->next=q;                      //将剩余部分连到 r 后面
  /*L1 是合并后的升序链表，注意此时 r 已经不是指向尾元素的指针了*/
}
```

例题：

已知两个长度分别为 m 和 n 带头节点的升序链表 L1 和 L2，若将它们合并为一个长度为 m + n 的升序链表，头节点为 L3。请设计一个时间和空间上尽可能高效的算法，返回新的头节点 L1。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

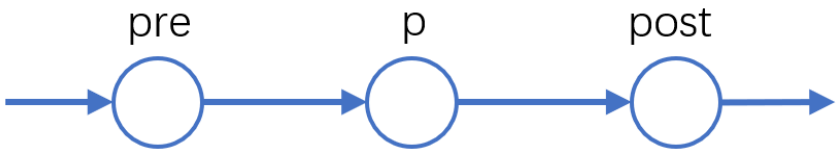
这是 2 个升序链表，要合并这两个链表得到新的升序链表，可以考虑使用指针后移，设置两个指针 p 和 q 分别保存此时正在处理的链表 L1 和 L2 中的结点，每次比较 p 和 q 所指结点的权值，把权值较小的那个结点取下来尾插入新的链中，然后对应指针后移一个结点，重复进行直到一条链遍历完，然后把另一条链连在 L3 末尾，得到的 L3 就是升序序列。最终每个结点都只会遍历一次，不会回头，所以时间复杂度是 O(n+m)。

4.5 分析复杂度

分析所用算法的时间和空间复杂度，常见时间复杂度：O(logn)、O(n)、O(nlogn)、O(n^2)、O(n^3)。书写部分类似于数组，有的题目会要求写出对应结点结构。

4.6 一些基本操作

有很多同学经常会不知道插入、删除等操作该怎么写，特别是步骤应该按什么顺序，怕断链，其实有一种很简单的做法（熟悉的同学不需要掌握）：**只要把所有涉及到的结点都事先用指针保存下来就不可能断链。**如下图，我们可以实现自己约定好当前删除结点就用指针 p 指向，前一个结点就用 pre 指向，后一个就用 post 指向。代码的顺序可以改变（注意，下面的代码在操作之前就已经用对应的指针如 pre、p、post 指向对应结点了，在写之前请确认）。



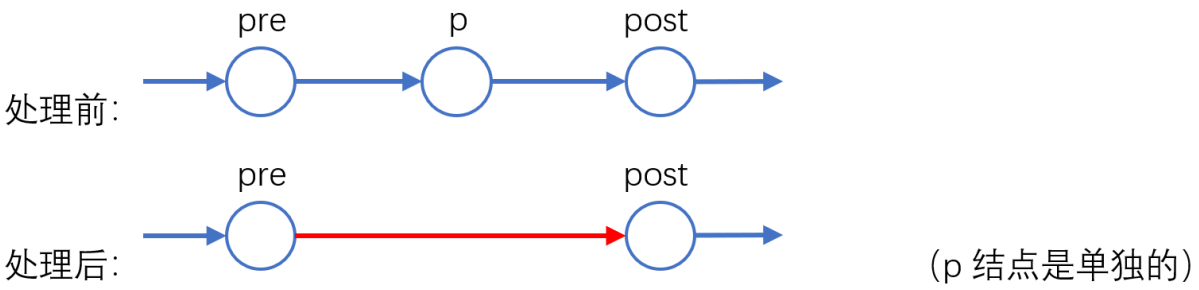
4.6.1 取出/删除 p 指向的结点

取出 p 指向的结点：

```
pre->next=post;
p->next=null;                    //不写也没关系，之后会改变 p->next;
```

删除 p 指向的结点：

```
pre->next=post;
free(p);                         //不写也没关系，考试不会扣分的
```

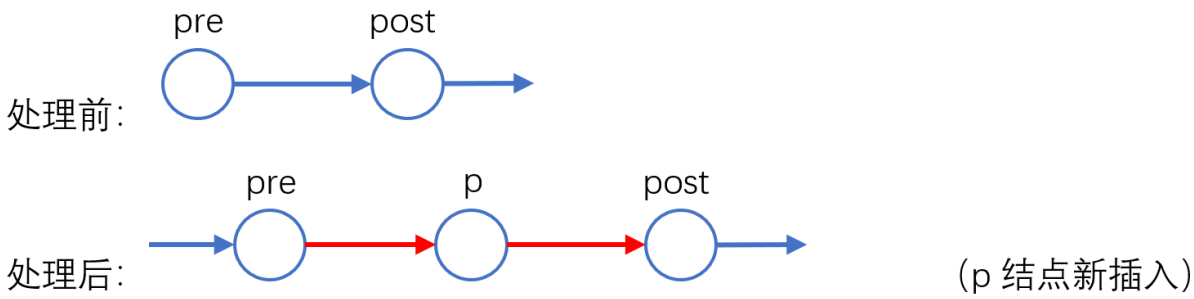


（p 结点是单独的）

4.6.2 将 p 结点插入到 pre 结点后面

p 插入 pre 后面的结点：

```
p->next=post;
pre->next=p;
```

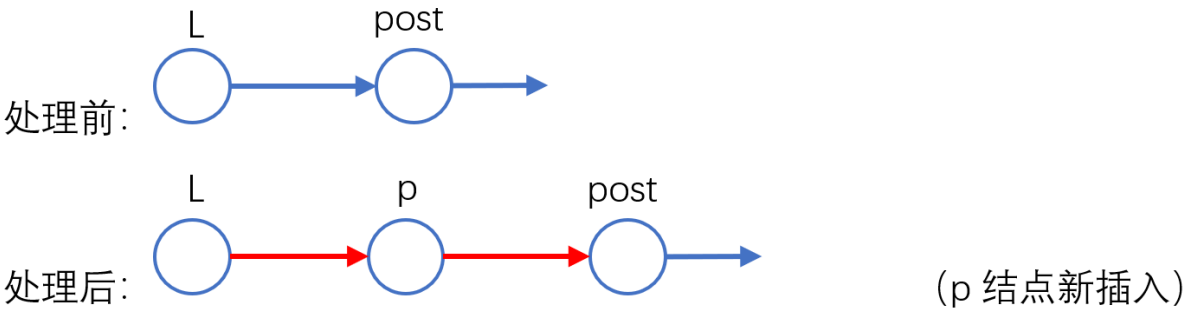


（p 结点新插入）

4.6.3 头插法插入 p

需要将 p 插入到 L 和 post=L->next 之间

```
p->next=post;
L->next=p;
```



（p 结点新插入）

4.6.4 遍历链表

对于线性表（数组+链表）的遍历，如果表中元素个数确定就用 for 循环，否则用 while 循环。在遍历过程中访问指针 p 每次都需要后移即 p=p->next，不要忘记。

已知元素个数 n：

```
Node* p=L->next;
for (int i=0; i<n; i++){
   visit(p);                                    //访问 p 结点
   p=p->next;
}
```

不知道元素个数：

```
Node* p=L->next;
while (p!=null){
   visit(p);                                    //访问 p 结点
   p=p->next;
}
```

5 树

5.1 需要注意的地方

5.1.1 题目的要求

树的题目一般不会特别要求复杂度，如果没有特别要求的话，考试中只要做出来就可以直接写，不用考虑优化问题。一般都会考察树的遍历，除非特别说明，否则遍历就用递归写，不要写非递归。

5.1.2 考察结构

树和森林（孩子兄弟表示法）

firstchild 是指向该节点第一个儿子节点的指针，nextbro 是指向下一个兄弟节点的指针

```
typedef struct TNode{
    Elemtype  data;                          //每个节点的权值
    struct TNode *firstchild, *nextbro;
}TNode;                                      //树节点
```

树和森林（节点用双亲表示法存储，常用于并查集）

一般采用数组表示，data 数组保存每个节点的权值，father 数组保存每个节点的父亲节点在数组中的下标：

```
int father[maxsize];        //保存每个节点的父亲节点在数组中的下标
Elemtype data[maxsize];     //保存每个节点的权值
```

也可以

```
typedef struct TNode{
    Elemtype  data;          //每个节点的权值
    struct TNode *father;    //father 是指向该节点父亲节点的指针
}TNode;                      //树节点
```

二叉树（左右孩子表示）

```
typedef struct BTNode{
    Elemtype  data;                    //每个节点的权值
    struct BTNode *lchild,*rchild;     //指向左右孩子节点的指针
}BTNode;                               //二叉树节点
```

5.2 树的遍历（递归）

树的遍历是 408 代码题的常考点，包括 2014、2017、2022 年，只要考察树必然需要用到遍历，一般是考察树的遍历+某种树的定义（如二叉搜索树），而二叉树的遍历包括前序中序后序遍历，他们又分为递归和非递归两种方式，每一种遍历的非递归逻辑都完全不一样，复习起来很痛苦也没有必要，而递归的代码逻辑非常简单，所以考试中除了特殊情况，树的遍历请一定使用递归（递归真的非常简单好用）。

前中后序的递归遍历时间复杂度为 O(n)，空间复杂度为 O(h)，n 是总节点数，h 是树高。

前序（根左右）：

```
void preorder(BTNode *p){
    if (p==NULL) return;
    visit(p);                          //对 p 节点的访问等
    preorder(p->lchild);
    preorder(p->rchild);
    /*这三条语句，visit 在先就是先序，在中就是中序，在后就是后序*/
}
```

中序（左根右）：

```
void inorder(BTNode *p){
    if (p==NULL) return;
    inorder(p->lchild);
    visit(p);                          //对 p 节点的访问等
    inorder(p->rchild);
}
```

后序（左右根）：

```
void postorder(BTNode *p){
    if (p==NULL) return;
    postorder(p->lchild);
    postorder(p->rchild);
    visit(p);                                    //对 p 节点的访问等
}
```

5.3 可能考察的点

对平衡二叉树、二叉排序树的判断

哈夫曼树的遍历、计算和判断

6 用真题检验上述做题方法

6.1 真题

2009 年

42.（15 分）已知一个带有表头结点的单链表，结点结构为

data	link

假设该链表只给出了头指针 list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第 k 个位置上的结点（k 为正整数）。若查找成功，算法输出该结点的 data 域的值，并返回 1；否则，只返回 0。要求：

1）描述算法的基本设计思想。

2）描述算法的详细实现步骤。

3）根据设计思想和实现步骤，采用程序设计语言描述算法（使用 C、C++或 Java 语言实现），关键之处请给出简要注释。

链表

暴力解：枚举　　时间：$O(n^2)$　　空间：O(1)

暴力解：递归+链表遍历　　时间：O(n)　　空间：O(n)

暴力解：数组保存所有链表结点　　时间：O(n)　　空间：O(n)

较优解（也是最优）：遍历链表两次　　时间：O(n)　　空间：O(1)

最优解：链表操作技巧——前后双指针　　时间：O(n)　　空间：O(1)

暴力解：枚举

1）两层循环，外层是遍历 i，内层是找到倒数第 i 个元素，比如上一轮查找到倒数第 i-1 个结点是 q，那下一轮当 p->link==q 时 p 就是倒数第 i 个结点。

2）算法如下：

```
int ans(node* list, int k){
   node* p, q=null;                    //q=null 是倒数第 0 个结点
   for (int i=1; i<=k; i++){
      p=list->link;
      if (q==p)
          return 0;                    //找不到倒数第 k 个结点
      while (p->link!=q)
          p=p->link;                   //p 最终是倒数第 i 个结点
      q=p;
   }
   cout<<p->data;                      //输出倒数第 k 个结点的值
   return 1;
}
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

暴力解：递归+链表遍历

1）递归调用一个一个访问节点，全局变量 t 表示此时访问的结点是倒数第几个结点，如果 t==k 说明是倒数第 k 个结点，输出该结点。

2）算法如下：

```
int t=0;                               //t 表示当前访问的是倒数第 k 个节点
void find(node* p, int k){
   if (p==null)
      return;
   find(p->link, k);
   t++;
   if (t==k)
      cout<<p->data;                   //输出倒数第 k 个元素的值
}
int ans(node* list, int k){
```

```
        find(list->link, k);
        if (t<k)
            return 0;
        return 1;
    }
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

暴力解：数组保存所有链表结点

1）遍历链表，将所有结点都存放在数组中，最后输出数组中倒数第 k 个元素即可。

2）算法如下：

```
int ans(node* list, int k){
    node* A[maxn];                          //结点数组
    node* p=list->link;                     //正在遍历的结点
    int n=0;                                //数组中结点个数
    while (p!=null){
        A[n++]=p;
        p=p->link;
    }
    if (n<k)
        return 0;
    cout<<A[n-k]->data;                     //输出倒数第 k 个元素的值
    return 1;
}
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

暴力解：数组保存所有链表结点

1）要找到倒数第 k 个结点，但链表不能回退，所以需要知道他是正数第 t 个结点（不算头结点），t=n-k+1，n 是总结点数，那我们只需要先遍历一次链表求出 n，然后再遍历一次链表找到第 t 个结点。

2）算法如下：

```
int t=0;                                    //t 表示当前访问的是倒数第 k 个节点
int find(node* p, int k){
    if (p==null)
        return 0;                           //查找失败
    t=1+find(p->link);
```

```
    if (t==k)
        cout<<p->data;                    //输出倒数第 k 个元素的值
    return 1;                             //查找成功
}
int ans(node* list, int k){
    return find(list->link, k);
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

较优解（也是最优）：遍历链表两次

1）要找到倒数第 k 个结点，但链表不能回退，所以需要知道他是从前往后数第 t 个结点（不算头结点），t=n-k+1，n 是总结点数，那我们只需要先遍历一次链表求出 n，然后再遍历一次链表找到第 t 个结点。

2）算法如下：

```
int ans(node* list, int k){
    node* p=list->link;
    int t, n=0;
    while (p!=null){
        n++;
        p=p->link;
    }
    t=n-k+1;
    if (t<0)
        return 0;
    p=list->link;
    for (int i=0; i<t; i++)
        p=p->link;
    cout<<p->data;                        //输出倒数第 k 个元素的值
    return 1;
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

最优解：链表操作技巧——前后双指针

1）初始时设置两个指针 p 和 q 都指向头结点，q 指针先后移 k 次，若此过程中 q 指向了 null

则查找失败，返回 0。然后将 p 和 q 同时后移，直到 q 指向 null 为止，因为 p 和 q 直接相差 k 次移动，所以此时 p 一定指向倒数第 k 个结点，输出 p，返回 1。

2）算法如下：

```
int ans(node* list, int k){
    node* p=q=list;
    for (int i=0; i<k; i++){
        q=q->link;
        if (q==null)
            return 0;
    }
    while (q!=null){
        p=p->link;
        q=q->link;
    }
    cout<<p->data;                    //输出倒数第 k 个元素的值
    return 1;
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

2010 年

42．（13 分）设将 n（n > 1）个整数存放到一维数组 R 中。试设计一个在时间和空间两方面都尽可能高效的算法。将 R 中保存的序列循环左移 p（0 < p < n）个位置，即将 R 中的数据由$(X_0, X_1, \cdots, X_{n-1})$变换为$(X_p, X_{p+1}, \cdots, X_{n-1}, X_0, X_1, \cdots, X_{p-1})$。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C、C++或 Java 语言描述算法，关键之处给出注释。

3）说明你所设计算法的时间复杂度和空间复杂度。

顺序表

暴力解：另设一个数组，再移动　　时间：O(n)　　空间：O(n)

最优解：技巧，数组翻转　　时间：O(n)　　空间：O(1)

暴力解：另设一个数组，再移动

1）原序列保存在数组 A 中，设另一个数组 B，将 A[0], A[1],…, A[p-1]放到 B[n-p], B[n-p+1],…, B[n-1]中，将 A[p], A[p+1],…, A[n-1]放到 B[0], B[1],…, B[n-p-1]中，数组 B 即是所求。

2）算法如下：

```
void ans(int A[], n){
   int B[n];
   for (int i=0; i<p; i++)
      B[i+n-p]=A[i];
   for (int i=p; i<n; i++)
      B[i-p]=A[i];
   输出 B 数组
}
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

最优解：技巧，数组翻转　　时间：O(n)　　空间：O(1)

1）原序列保存在数组 A 中，将数组 A[0]~A[p-1]翻转，然后将数组 A[p]~A[n-1]翻转，最后将数组 A[0]~A[n-1]翻转。

2）算法如下：

```
void Reverse(int A[], L, R){
   num=(R-L)/2;
   for (int i=0; i<=num; i++)
      swap(A[L+i],A[R-i]);              //交换这两个数
}
void ans(int A[], n){
   Reverse(A, 0, p-1);
   Reverse(A, p, n-1);
   Reverse(A, 0, n-1);
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

2011 年

42．(15 分）一个长度为 L（$L\geqslant 1$）的升序序列 S，处在第$\lceil L/2\rceil$个位置的数称为 S 的中位数。例如，若序列 S1 = (11, 13, 15, 17, 19)，则 S1 的中位数是 15，两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若 S2 = (2, 4, 6, 8, 20)，则 S1 和 S2 的中位数是 11。现在有两个等长升序序列 A 和 B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列 A 和 B 的中位数。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C、C++或 Java 语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

顺序表，找中位数

暴力解：都放入新数组中快速排序　　时间：O(nlogn)　空间：O(n)

较优解：数组指针后移把数存入第三个数组　　时间：O(n)　　空间：O(n)

较优解：数组指针后移　　时间：O(n)　　空间：O(1)

最优解：多有序数组折半查找　　时间：O(logn)　空间：O(1)

暴力解：都放入新数组中快速排序

1）将数组 A 和数组 B 中的数据都存入数组 C 中，C 数组长度为 2n，对数组 C 进行快速排序，输出排序后数组中的 C[n-1]。

2）算法如下：

```
void Qsort(int A[], L, R){        //a 数组保存数据，L 和 R 是边界
    if (L>=R) return;              //当前区间元素个数<=1 则退出
    int key, i=L, j=R;             //i 和 j 是左右两个数组下标移动
    把 A[L~R]中随机一个元素和 A[L]交换      //快排优化，使得基准值的选取随机
    key=A[L];                      //key 作为枢值参与比较
    while (i<j){
        while (i<j && A[j]>key)
            j--;
        while (i<j && A[i]<=key)
            i++;
        if (i<j)
            swap(A[i], A[j]);      //交换 A[i]和 A[j]
    }
    swap(A[L], A[i]);
    Qsort(A, L, i-1);             //递归处理左区间
    Qsort(A, i+1, R);             //递归处理右区间
}
void ans(int A[], B[], n){
    int C[2n];
    for (int i=0; i<n; i++){
        C[i]=A[i];
        C[n+i]=B[i];
```

```
    }
    Qsort(C, 0, 2n-1);                    //快速排序处理 C[0]~C[2n-1]
    cout<<C[n-1];                         //输出 C[n-1]
}
```

3）时间复杂度：O(nlogn)　　空间复杂度：O(n)

较优解：数组指针后移把数存入第三个数组

1）采用归并的方式合并数组 A 和 B 到新的数组 C 中，给 A、B 数组设置变量 i 和 j 保存数组下标，初始时都为 0，每次比较 A[i]和 B[j]，数组 C 的下一个空位保存小的那个数，并且对应指针后移，直到一个数组遍历完，将另一个数组中剩下的元素依次保存到 C 空位中。

2）算法如下：

```
void ans(int A[], B[], n){
    int C[2n];
    int i=j=k=0;
    while (i<n && j<n)
        if (A[i]<B[j])
            C[k++]=A[i++];
        else C[k++]=B[j++];
    while (i<n)
        C[k++]=A[i++];
    while (j<n)
        C[k++]=B[j++];
    cout<<C[n-1];                         //输出 C[n-1]
}
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

较优解：数组指针后移

1）给 A、B 数组设置变量 i 和 j 保存数组下标，计数 k 表示此时是第几小的元素，初始时都为 0，每次比较 A[i]和 B[j]，取较小的那个数对应指针后移且 k++，直到 k 等于 n，输出这个数。

2）算法如下：

```
void ans(int A[], B[], n){
    int i=j=0, k;
    for (k=1; k<n; k++)                   //共移动 n-1 次
        if (A[i]<B[j]) i++;
        else j++;
```

```
        cout<<min(A[i], B[j]);              //输出A[i]和B[j]中的小值
    }
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

最优解：多有序数组折半查找

1）对 A、B 数组进行折半查找，设 L1、R1 为数组 A 的左右查找边界，L2、R2 为 B 的左右查找边界。数组 A[L1~R1]和 B[L2~R2]的中位数，设为 a 和 b，求序列 A、B 的中位数过程如下：

①若 a<b 则舍弃序列 A 中较小的一半 A[L1~mid-1]，同时舍弃序列 B 中较大的一半 B[mid+1~R1]。

②若 a>=b 则舍弃序列 A 中较大的一半 A[mid+1~R1]，同时舍弃序列 B 中较小的一半 B[L1~mid-1]。

重复过程①、②，且需要保证数组 A、B 中的元素个数相同，如果不同则从长的数组中舍弃一个（因为我们取 mid 时都是向下取整，所以这里舍弃最小的那个值），直到两个序列中均只含一个元素时为止，较小者即为所求的中位数。

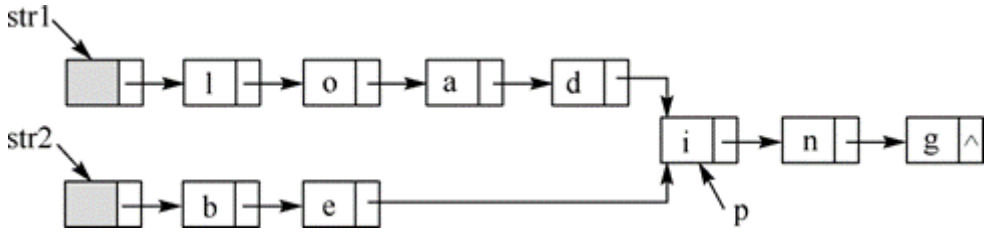
2）算法如下：

```
    void ans(int A[], B[], n){
        int mid1, mid2, L1=L2=0, R1=R2=n-1;
        while (L1<=R1){                     //如果L>=R则退出循环
            mid1=(L1+R1)/2;
            mid2=(L2+R2)/2;                 //取中间数，向下取整
            if (A[mid1]<B[mid2]){
                L1=mid1;
                R2=mid2;                    //更新查找范围
                if (R1-L1!=R2-L2)
                    L1++;                   //保证R1-L1=R2-L2
            }
            else{
                R1=mid1;
                L2=mid2;                    //更新查找范围
                if (R1-L1!=R2-L2)
                    L2++;                   //保证R1-L1=R2-L2
            }
        }
        cout<<min(A[L1],B[L2]);             //输出A[L1]和B[L2]较小值
    }
```

3）时间复杂度：O(logn)　　空间复杂度：O(1)

2012 年

42．假定采用带头结点的单链表保存单词，当两个单词有相同的后缀时，则可共享相同的后缀存储空间，例如，“loading”和“being”的存储映像如下图所示。



设 str1 和 str2 分别指向两个单词所在单链表的头结点，链表结点结构为，请设计一个时间上尽可能高效的算法，找出由 str1 和 str2 所指向两个链表共同后缀的起始位置（如图中字符 i 所在结点的位置 p）。要求：

1）给出算法的基本设计思想。

2）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释。

3）说明你所设计算法的时间复杂度。

链表

暴力解：枚举 str1 和 str2 共同后缀起始位置　　时间：$O(n^2)$　　空间：O(1)

暴力解：链表转数组，数组保存结点地址　　时间：O(n)　　空间：O(n)

最优解：链表操作，双指针　　时间：O(n)　　空间：O(1)

暴力解：枚举 str1 和 str2 起始位置

1）设 str1 和 str2 的长度较大的值是 n，两层循环枚举 str1 和 str2 相同后缀的起始位置。

2）算法如下：

```
void ans(Node* str1, str2){
    Node* p=str1->next;                    //str1 的后缀起始位置
    Node* q;                               //之后循环中会赋初值
    while (p!=null){
        q=str2->next;                      //每次进入内层循环前要给 q 赋初值
        while (q!=null){
            if (p==q){
                cout<<p;                   //后缀完全匹配
                return;
```

```
                }
                q=q->next;
            }
            p=p->next;
        }
        cout<<null;                          //无相同后缀，可不写
    }
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

暴力解：链表转数组，数组保存结点地址

1）设 str1 和 str2 的长度较大的值是 n，设置两个新数组 S1 和 S2 分别保存 str1 和 str2 链中的每个结点的地址（这里用为了方便，从 S1[1]和 S2[1]开始存，S1[0]和 S2[0]作为哨兵，设置不相同）。

2）算法如下：

```
void ans(Node* str1, str2){
    Node* S1[n+1];
     Node* S2[n+1];          //保存 str1 和 str2 各结点地址
    Node* p=str1->next;     //str1 的后缀起始位置
    Node* q=str2->next;             //str2 的后缀起始位置
    Len1=Len2=0;
    while (p!=null){
        S1[++Len1]=p;
        p=p->next;
    }
    while (q!=null){
        S2[++Len2]=q;
        q=q->next;
    }
    if (S1[Len1]!=S2[Len2]){
        cout<<null;                     //无相同后缀，可不写
        return;
    }
S1[0]=-1;
S2[0]=-2;                           //S1[0]和 S2[0]作为哨兵判断链是否访问完
```

```
        for (int i=1; i<min(Len1, Len2); i++)
            if (S1[Len1-i]!=S2[Len2-i]){
                cout<<S1[Len1-i+1];          //输出起始地址
                return;
            }
    }
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

最优解：链表操作，双指针

1）每个链表设置一个指针，先求出两个链的长度，然后长的链指针往后移动 x 次（x 是长度差的绝对值），使得两链剩余未访问结点数相同，然后两链表的指针同步往后移动比较，直到指向的是同一个结点。

2）算法如下：

```
    void ans(Node* str1, str2){
        Node* p=str1->next;              //指针 p 用来遍历 str1
        Node* q=str2->next;              //指针 q 用来遍历 str2
        int Len1=Len2=0;
        while (p!=null){
            Len1++;
            p=p->next;
        }
        while (q!=null){
            Len2++;
            q=q->next;
        }
        if (Len1>Len2)
            for (int i=0; i<Len1-Len2; i++)
                str1=str1->next;
        else
            for (int i=0; i<Len2-Len1; i++)
                str2=str2->next;
        int Len=min(Len1, Len2);
        for (int i=0; i<Len; i++){
            if (str1==str2){
                cout<<str1;              //输出相同后缀起始地址
```

```
            return;
        }
        str1=str1->next;
        str2=str2->next;
    }
    cout<<null;                          //无相同后缀起始地址
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

2013 年

41．（13 分）已知一个整数序列 A = (a_0, a_1,…, a_{n-1})，其中 0≤a_i＜n（0≤i < n)。若存在 a_{p1} = a_{p2} =…= a_{pm}= x 且 m > n/2（0≤p_k< n, 1≤k≤m)，则称 x 为 A 的主元素。例如 A= (0, 5, 5, 3, 5, 7, 5, 5)，则 5 为主元素；又如 A = (0, 5, 5, 3, 5, 1, 5, 7)，则 A 中没有主元素。假设 A 中的 n 个元素保存在一个一维数组中，请设计一个尽可能高效的算法，找出 A 的主元素。若存在主元素，则输出该元素；否则输出-1。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C、C++或 Java 语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

顺序表

暴力解：分别判断每个数是不是主元素　　时间：O(n^2)　　空间：O(1)

暴力解：快速排序，然后查找主元素　　时间：O(nlogn)　空间：O(logn)

较优解：以空间换时间　　时间：O(n)　空间：O(n)

最优解：技巧　　时间：O(n)　　空间：O(1)

暴力解：枚举，分别判断每个数是不是主元素

1）双重循环枚举判断每个元素是否是主元素

2）算法如下：

```
void ans(int A[], n){
    int m;                                  //m 统计当前枚举元素出现次数
    for (int i=0; i<n; i++){
        m=0;                  //每次选择元素的时候 m 要清零
```

```
        for (int j=0; j<n; j++)
            if (A[i]==A[j])
                m++;
        if (m>n/2){
            cout<<A[i];                    //找到了主元素
            return;
        }
    }
    cout<<-1;                              //未找到主元素
    return;
}
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

暴力解：快速排序，然后查找主元素

1）先快速排序得到升序序列，权值相同的元素都会相邻，主元素如果存在则一定是 A[(n-1)/2]扫描一趟数组判断 A[(n-1)/2]是否是主元素。

2）算法如下：

```
void Qsort(int A[], L, R){        //a 数组保存数据，L 和 R 是边界
    if (L>=R) return;              //当前区间元素个数<=1 则退出
    int key, i=L, j=R;             //i 和 j 是左右两个数组下标移动
    把 A[L~R]中随机一个元素和 A[L]交换   //快排优化，使得基准值的选取随机
    key=A[L];                      //key 作为枢值参与比较
    while (i<j){
        while (i<j && A[j]>key)
            j--;
        while (i<j && A[i]<=key)
            i++;
        if (i<j)
            swap(A[i], A[j]);      //交换 A[i]和 A[j]
    }
    swap(A[L], A[i]);
    Qsort(A, L, i-1);              //递归处理左区间
    Qsort(A, i+1, R);              //递归处理右区间
}
void ans(int A[], n){
```

```
    Qsort(A, 0, n-1);
    int t=A[(n-1)/2];
   int sum=0;
    for (int i=1; i<n; i++)
       if (A[i]==t)
          sum++;
    if (sum>n/2)
        cout<<t;                          //找到主元素
    else cout<<-1;                        //未找到主元素
    return;
}
```

3）时间复杂度：O(nlogn)　　空间复杂度：O(logn)

较优解：以空间换时间

1）设置一个数组 count[0~n-1]统计每个数字出现的次数，比如 count[i]=1 表示数字 i 出现了 1 次，遍历一次 A 数组统计所有数字出现的次数，处理完后扫描一次 count 数组找到是否有主元素，如果有则输出主元素，否则输出-1。

2）算法如下：

```
void ans(int A[], n){
    int count[n];                     //count 统计数字出现频率
    for (int i=0; i<n; i++)
       count[A[i]]++;        /*0<=A[i]<=n，不会越界*/
    for (int j=0; j<n; j++)
       if (count[j]>n/2){
           cout<<j;                   //找到了主元素，输出主元素
           return;
       }
    cout<<-1;                         //未找到主元素，输出-1
    return;
}
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

最优解：技巧

1）变量 j 保存可能的主元素，初始时 j=A[0]，t=1，从 A[1]开始扫描一次数组 A，如果 j==A[i]

则 t++，否则 t--，如果 t<0 则更新 j=A[i]且 t 变成 1。之后在扫描一次数组 A，判断 j 是否是主元素（m 统计该元素出现的个数），如果是则输出 j，否则输出-1。

2）算法如下：

```
void ans(int A[], n){
    int j=A[0], m, t=1;                    //取 A[0]作为可能主元素
    for (int i=1; i<n; i++)
        if (A[i]==j) t++;
        else{
            t--;
            if (t<0){
                j=A[i];
                t=1;
            }
        }
    m=0;
    for (int i=0; i<n; i++)
        if  (A[i]==j)
            m++;
    if (m>n/2) cout<<j;                    //j 是主元素，输出 j
    else cout<<-1;                         //j 不是主元素，输出-1
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

2014 年

41．(13 分）二叉树的带权路径长度（WPL）是二叉树中所有叶结点的带权路径长度之和。给定一棵二叉树 T，采用二叉链表存储，结点结构如下：

left	weight	right

其中叶结点的 weight 域保存该结点的非负权值。设 root 为指向 T 的根结点的指针，请设计求 T 的 WPL 的算法，要求：

1）给出算法的基本设计思想。

2）使用 C 或 C++语言，给出二叉树结点的数据类型定义。

3）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

带权路径长度（哈夫曼树中的内容）

WPL 有两种计算方法：

1 **WPL 的定义**：WPL=所有（叶结点的权值 W*该结点深度 D）求和

2 在哈夫曼树中，WPL=所有非叶结点的权值和。本题不是哈夫曼树，不能这样计算。

（如果本题给的是哈夫曼树，可以使用方法 2，这样在遍历的时候，只需要判断这个节点是不是叶节点，可以不用考虑深度）

解法：二叉树遍历（建议递归）　　时间：O(n)　　空间：O(h)

解法：二叉树遍历（建议递归）

1）先序遍历二叉树，设置全局变量 WPL，每次访问到叶子结点时，将他的权值与深度相乘，再累加到 WPL 中。

2）类型定义：

```
typedef struct BTNode{
   int weight;
   struct BTNode *left,*right;
}BTNode;
```

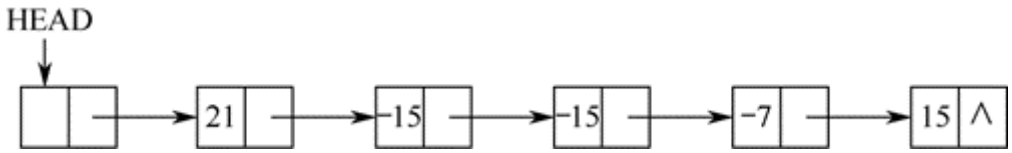
3）算法如下：

```
int WPL=0;                                    //WPL 是全局变量
void preorder(BTree p, int d){          //当前结点 p，深度 d
   if (p==null)                               //p 是空结点
      return;
   if (p->left==null && p->right==null)     //p 是叶结点
      WPL+=d*p->weight;
   preorder(p->left, d+1);                  //递归调用左孩子
   preorder(p->right, d+1);                  //递归调用右孩子
}
void ans(BTNode* T){
   preorder(T, 0);                           //根结点深度为 0
}
```

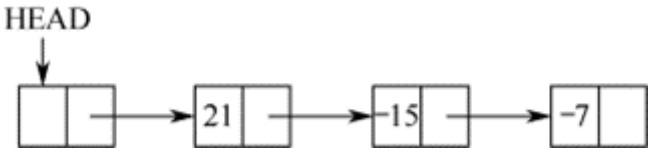
2015 年

41．（15 分）用单链表保存 m 个整数，结点的结构为[data][link]，且|data|≤n（n 为正整数）。现要求设计一个时间复杂度尽可能高效的算法，对于链表中 data 的绝对值相等的结点，仅保留第一

次出现的结点而删除其余绝对值相等的结点。例如，若给定的单链表 HEAD 如下：



则删除结点后的 HEAD 为



要求：

1）给出算法的基本设计思想。

2）使用 C 或 C++语言，给出单链表结点的数据类型定义。

3）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

4）说明你所设计算法的时间复杂度和空间复杂度。

链表

暴力解：枚举，每一个结点都和其他结点比较　　时间：$O(m^2)$　　空间：O(1)

最优解：空间换时间　　时间：O(m)　　空间：O(n)

暴力解：枚举，每一个结点都和其他结点比较

1）两层循环枚举所有结点的两两比较，外层遍历链表中的结点 p，内层遍历 p 之后的结点将和 p 绝对值相等的结点删除（只保留第一个出现的 p）。

2）结点数据结构定义：

```
typedef struct Node{
    int data;                          //该节点权值
    struct Node *link;                 //下一个结点
}Node;
```

3）算法如下：

```
void ans(Node* HEAD){
    Node* p=HEAD->link;                //外层遍历结点 p
    Node* q, r;                        //q 是 r 前一个结点
    while (p!=null){
        q=p;                           //q 从 p 开始
        while (q->link!=null){
            r=q->link                  //r 表示待比较结点
```

```
            if (abs(r->data)==abs(p->data)){
                q->link=r->link;
                free(r);
            }
            else q=q->link;              //不相同时才修改 q
        }
        p=p->link;
    }
}
```

3）时间复杂度：$O(m^2)$　　空间复杂度：O(1)

最优解：空间换时间

1）data 的绝对值≤n，那可以用数组 count[0~n]来保存对于每一个绝对值是否出现过，比如 count[1]>0 表示该绝对值已经出现过了，之后再出现就可以删除当前访问的结点。遍历一次链表，更新 count 数组以及删除后面重复出现的结点。

2）结点数据结构定义：

```
typedef struct Node{
    int data;                            //该节点权值
    struct Node *link;                   //下一个结点
}Node;
```

3）算法如下：

```
void ans(Node* HEAD){
    Node* p=HEAD->link;                  //当前访问结点 p
    Node* pre=HEAD;                      //p 的前一个结点 pre
    bool count[n+1];                     //可以加上数组初始化
    while (p!=null){
        if (count[abs(p->data)]==false)
            count[abs(p->data)]=true;    //p 的权值第一次出现
        else{                            //p 的权值前面已经出现过
            pre->link=p->link;
            free(p);
            p=pre;
        }
        pre=p;                           //更新 pre
        p=pre->link;                     //更新 p
```

```
        }
    }
```

3）时间复杂度：O(m)　　空间复杂度：O(n)

2016 年

43．已知由 n（$n\geqslant 2$）个正整数构成的一个集合 $A=\{a_i|0\leqslant k<n\}$，将其划分为两个不相交的子集 A_1和 A_2，元素个数分别是 n_1和 n_2，A_1和 A_2中元素之和分别为 S_1和 S_2。设计一个尽可能高效的划分算法，满足$|n_1-n_2|$最小且$|S_1-S_2|$最大。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（3）说明你所设计算法的平均时间复杂度和空间复杂度。

数组

暴力解：快速排序　　时间：O(nlogn)　　空间：O(logn)

最优解：类快排思想找中位数枢值　　时间：O(n)　　空间：O(n)

暴力解：快速排序

1）对数组 A[0~n-1]进行快速排序得到升序，排序后集合 S1 是 A[0~n/2-1]（向下取整），集合 S2 是 A[n/2~n-1]（向下取整）。

2）算法如下：

```
void Qsort(int A[], L, R){        //a 数组保存数据，L 和 R 是边界
   if (L>=R) return;               //当前区间元素个数<=1 则退出
   int key, i=L, j=R;              //i 和 j 是左右两个数组下标移动
   把 A[L~R]中随机一个元素和 A[L]交换   //快排优化，使得基准值的选取随机
   key=A[L];                       //key 作为枢值参与比较
   while (i<j){
      while (i<j && A[j]>key)
          j--;
      while (i<j && A[i]<=key)
          i++;
      if (i<j)
          swap(A[i], A[j]);        //交换 A[i]和 A[j]
   }
   swap(A[L], A[i]);
```

```
        Qsort(A, L, i-1);                    //递归处理左区间
        Qsort(A, i+1, R);                    //递归处理右区间
    }
    void ans(int A[], n){
        Qsort(A, 0, n-1);
        输出 S1：A[0~n/2-1]
        输出 S2：A[n/2~n-1]                  //奇数时中间数发给 S2
    }
```

3）时间复杂度：O(nlogn)　　空间复杂度：O(logn)

最优解：类快排思想找中位数枢值

1）对数组 A[0~n-1]进行类似快速排序的做法，在处理左右区间时只处理可能包含中位数的区间，即如果区间的范围是[l, r]，则只有 l<=n/2-1<=r 才会处理该区间。对快排的递归调用稍作修改即可。

2）算法如下：

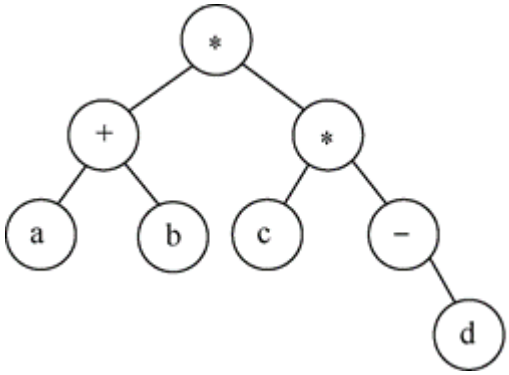
```
    void Qsort(int A[], L, R){          //a 数组保存数据，L 和 R 是边界
        if (L>=R) return;                //当前区间元素个数<=1 则退出
        int key, i=L, j=R;               //i 和 j 是左右两个数组下标移动
        把 A[L~R]中随机一个元素和 A[L]交换   //快排优化，使得基准值的选取随机
        key=A[L];                        //key 作为枢值参与比较
        while (i<j){
            while (i<j && A[j]>key)
                j--;
            while (i<j && A[i]<=key)
                i++;
            if (i<j)
                swap(A[i], A[j]);        //交换 A[i]和 A[j]
        }
        swap(A[L], A[i]);
        if (n/2-1>=L && n/2-1<=i-1)      //n/2-1 在左区间范围中
            Qsort(A, L, i-1);            //递归处理左区间
        if (n/2-1>=i+1 && n/2-1<=R)      //n/2-1 在右区间范围中
            Qsort(A, i+1, R);            //递归处理右区间
    }
    void ans(int A[], n){
```

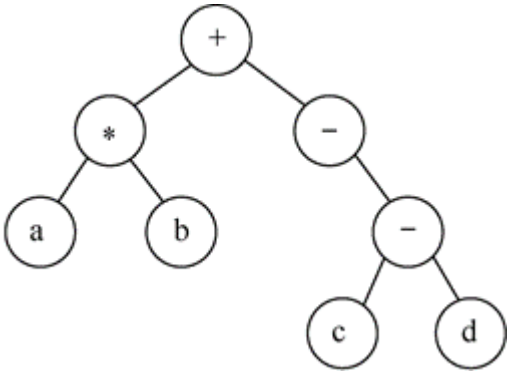
```
        Qsort(A, 0, n-1);
        输出 S1：A[0~n/2-1]
        输出 S2：A[n/2~n-1]
    }
```

3）时间复杂度：O(n)　　空间复杂度：O(logn)

2017 年

41．(15 分）请设计一个算法，将给定的表达式树（二叉树）转换为等价的中缀表达式（通过括号反映操作符的计算次序）并输出。例如，当下列两棵表达式树作为算法的输入时，输出的等价中缀表达式分别为(a+b)*(c*(-d))和(a*b)+(-(c-d))。





二叉树结点定义如下：

```
typedef struct node{
    char data[10];              //存储操作数或操作符
    struct node *left, *right;
}BTree;
```

要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

二叉树

解法：中序遍历，递归　　时间：O(n)　　空间：O(h)

1）中序遍历二叉树，对于当前访问的非空结点 p，则先输出”（“，然后递归调用左子树，输出 p 的权值，递归调用右子树，输出”)”，如果 p 是根或者叶结点则不需要输出”（”或”)”。

2）算法如下：

```
void inorder(BTree* p){                    //当前结点 p
```

```
    if (p==null)                                    //p 是空结点
        return;
    if (p->left!=null || p->right!=null)
        cout<<"(";
    inorder(p->left);                               //递归调用左子树
    输出 p->data                                    //输出操作数（符）
    inorder(p->right);                              //递归调用右子树
    if (p->left!=null || p->right!=null)
        cout<<")";
}
void ans(BTNode* T){
    inorder(T->left);                               //调用根左子树
    输出 T->data                                    //输出根操作数（符）
    inorder(T->right);                              //调用根右子树
}
```

2018 年

41.（13 分）给定一个含 n（$n\geq 1$）个整数的数组，请设计一个在时间上尽可能高效的算法，找出数组中未出现的最小正整数。例如，数组{−5, 3, 2, 3}中未出现的最小正整数是 1；数组{1, 2, 3}中未出现的最小正整数是 4。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

顺序表

暴力解：枚举最小正整数　　时间：$O(n^2)$　　空间：O(1)

暴力解：快速排序，然后扫描一遍数组　　时间：O(nlogn)　　空间：O(logn)

最优解：空间换时间　　时间：O(n)　　空间：O(n)

暴力解：枚举

1）可以证明最小未出现的正整数 t 一定≤n+1（因为如果 t>n+1，则 1~t-1 都会出现一次，但只有 n 个整数，所以 t-1≤n，即 t≤n+1），枚举所有可能的未出现最小正整数，扫描一次数组判断是否符合。

2）算法如下：

```
void ans(int A[], n){
   bool flag;                    //flag 表示是否出现
         for (int i=1; i<n+1; i++){
       flag=false;              //假设未出现
      for (int j=0; j<n; j++)
           if (i==A[j]){
              flag=true;          //此时 i 出现了，flag 变成 true
                    break;              //此时 i 已经出现，跳出内层循环
                 }
       if (flag==false){          //若 i 未出现过则输出 i
                 cout<<i;
                 return;
              }
           }
}
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

暴力解：快速排序，然后扫描一遍数组

1）先对数组 A 快速排序得到升序序列，然后遍历数组找到第一个未出现的最小正整数。

2）算法如下：

```
void Qsort(int A[], L, R){       //a 数组保存数据，L 和 R 是边界
   if (L>=R) return;              //当前区间元素个数<=1 则退出
   int key, i=L, j=R;             //i 和 j 是左右两个数组下标移动
   把 A[L~R]中随机一个元素和 A[L]交换   //快排优化，使得基准值的选取随机
   key=A[L];                      //key 作为枢值参与比较
   while (i<j){
      while (i<j && A[j]>key)
          j--;
      while (i<j && A[i]<=key)
          i++;
      if (i<j)
          swap(A[i], A[j]);       //交换 A[i]和 A[j]
   }
   swap(A[L], A[i]);
   Qsort(A, L, i-1);              //递归处理左区间
```

```
        Qsort(A, i+1, R);                    //递归处理右区间
    }
    void ans(int A[], n){                    //算法代码
        Qsort(A, 0, n-1);
        int t=1;                             //假设初始最小未出现正整数 t=1
        for (int i=0; i<n; i++){
            if (A[j]>t){
                cout<<t;                     //此时 t 是最小未出现正整数，输出 t
              return;
            }
          if (A[j]==t)
                t++;                         //t 已出现，最小未出现正整数是 t+1
        }
        cout<<t;                             //输出最小未出现正整数
        return;
    }
```

3）时间复杂度：O(nlogn)　　空间复杂度：O(logn)

最优解：空间换时间

1）可以证明最小未出现的正整数 t 一定≤n+1（因为如果 t>n+1，则 1~t-1 都会出现一次，但只有 n 个整数，所以 t-1≤n，即 t≤n+1），设置一个数组 count，count[i]表示 i 是否在原数组 A 中出现过，扫描一遍数组 A，更新 count 数组，之后再扫描一遍 count 数组输出未出现的最小正整数。

2）算法如下：

```
    void ans(int A[], n){
        int count[n+2];                      //0~n+1
        for (int i=0; i<n; i++)              //扫描数组 A
           if (A[i]>0 && A[i]<=n+1)          //当 0<A[i]<=n+1 时才修改
               count[A[i]]++;
        for (int j=1; j<=n+1; j++)           //扫描 count 数组
           if (count[j]==0){                 //j 未出现过，是最小未出现正整数
               cout<<j;                      //输出 j
               return;
           }
    }
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

2019 年

41．(13 分) 设线性表$L=(a_1,a_2,a_3,\cdots,a_{n-2},a_{n-1},a_n)$采用带头结点的单链表保存，链表中的结点定义如下：

```
typedef struct node {
        int data;
        struct node* next;
} NODE;
```

请设计一个空间复杂度为 O(1)且时间上尽可能高效的算法，重新排列 L 中的各结点，得到线性表$L'=(a_1,a_n,a_2,a_{n-1},a_3,a_{n-2},\cdots)$。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（3）说明你所设计的算法的时间复杂度。

链表（这题如果用数组保存链表结点绝对是零分）

暴力解：将后一半链表取下来每个结点从头插入　　时间：$O(n^2)$　　空间：O(1)

最优解：将后半段逆置（头插法），再重新插入　　时间：O(n)　　空间：O(1)

暴力解：将后一半链表取下来每个结点从头插入

1）从 $a_{\lceil n/2\rceil-1}$到从 a_n一个一个取下来从头开始插入，对于 a_{n-i}从 a_1开始往后移动 i 次即可找到插入位置，然后插入该位置即可。

2）算法如下：

```
void ans(Node* L, int n){
    int t=(n+1)/2;              //即 n/2 向上取整
    Node* pre=L, p, q, qq;      //q 为指向后半段链的指针
    for (int i=0; i<t; i++)
        pre=pre->next;          //pre 指向 a⌈ n/2⌉
    q=pre->next;                //q 指向 a⌈ n/2⌉ +1
    pre->next=null;             //a⌈ n/2⌉ 的下一个结点为空
    len=n-t;                    //后一半链长度
    for (int i=len; i>0; i--){  //一个一个重新插入
        pre=L;
        for (int j=0; j<i; j++) //找到插入位置
            pre=pre->next;
        p=pre->next;            //pre 是插入位置
        pre->next=q;            //插入 q
        qq=q->next;             //qq 暂存 q 的下一个结点
        q->next=p;              //q 下一个结点是插入位置后的点
        q=qq;                   //q 指向 qq 所指结点
    }
}
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

最优解：将后半段逆置（头插法），再重新插入

1）从 $a_{\lceil n/2\rceil+1}$到从 a_n这一段链逆置，再一个一个插入（只需要在上一次插入结点后两位插入即可）。

2）算法如下：

```
void ans(Node* L, int n){
    int t=(n+1)/2;              //即 n/2 向上取整
    Node* pre=L, p, q, qq;      //q 为指向后半段链的指针
    for (int i=0; i<t; i++)
        pre=pre->next;          //pre 指向 a⌈ n/2⌉
    q=pre->next;                //q 指向 a⌈ n/2⌉ +1
    pre->next=null;             //a⌈ n/2⌉ 的下一个结点为空
    Node* L1=malloc(Node);      //给后半段链设置头节点，L1 指向头节点
    L1.next=null;               //链 L1 初始没有其他结点
    Node* q1;
    while (q!=null){            //逆置，q 是待处理结点
        q1=q->next;             //q1 指向结点 p 的下一个结点
        q->next=L1->next;       //q 的下一个结点变成 L->next
        L1->next=q;             //L 的下一个结点是 q，q 完成头插
        q=q1;                   //更新 q，准备处理下一个结点
    }
    len=n-t;                    //后一半链长度
    pre=L->next;
    q=L1->next;
    for (int i=len; i>0; i--){  //一个一个重新插入
        p=pre->next;            //pre 是插入位置
        pre->next=q;            //插入 q
        qq=q->next;             //qq 暂存 q 的下一个结点
        q->next=p;              //q 下一个结点是插入位置后的点
        q=qq;                   //q 指向 qq 所指结点
        pre=p;                  //下一个插入位置
    }
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

2020 年

41．（13 分）定义三元组(a，b，c)（a，b，c 均为正数）的距离 D = |a – b| + |b – c| + |c – a|。给定 3 个非空整数集合 S1、S2 和 S3，按升序分别存储在 3 个数组中。请设计一个尽可能高效的算法，计算并输出所有可能的三元组(a，b，c)（a∈S1，b∈S2，c∈S3）中的最小距离。例如 S1 = {–1，0，9}，S2 = {–25，–10，10，11}，S3 ={2，9，17，30，41}，则最小距离为 2，相应的三元组为(9，10，9)。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

顺序表

暴力解：三重循环将 a、b、c 都扫描一次　　时间：O(nml)　　空间：O(1)

较优解：两重循环扫描 a、b，而 c 数组指针后移　　时间：O(nm)　　空间：O(1)

较优解：一重循环扫描 a，两轮折半查找 b、c　　时间：O(n*log(m*l))　　空间：O(1)

最优解：贪心，a、b、c 数组指针后移　　时间：O(n+m+l)　　空间：O(1)

暴力解：三重循环将 abc 都扫描一次

1）数组 A、B、C 的长度分别为 n、m、l，三重循环枚举 abc，对于每一组 abc 求出对应的距离 D，并从中选出最小的距离。

2）算法如下：

```
void ans(int A[], n, B[], m, C[], l){   //n、m、l 为三个数组长度
  int D_min=Max_int, D;                 //Dmax 赋值为最大整数
  for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
      for (int k=0; k<l; k++){      //三重循环枚举 abc
        D=abs(A[i]-B[j])+abs(B[j]-C[k])
          +abs(C[k]-A[i]);
        if (D<D_min) D_min=D;     //更新最小 D 值
      }
  cout<<D_min;                      //输出最小 D 值
}
```

3）时间复杂度：O(nml)或 $O(n^3)$　　空间复杂度：O(1)

较优解：两重循环扫描 ab，而 c 数组指针后移

1）数组 A、B、C 的长度分别为 n、m、l，两重循环枚举 a、b，对数组 C 设置变量 k 存储数组下标不断后移，初始 k=0，只要移动后的 D 更小，就选择往后移动，然后对于每一组 abc 求出对应的距离 D，并从中选出最小的距离。

2）算法如下：

```
int DD(int a, b, c){
    return abs(a-b)+abs(b-c)+abs(c-a);
}
void ans(int A[], n, B[], m, C[], l){ //n、m、l 为三个数组长度
    int D_min=Max_int, D;                    //Dmax 赋值为最大整数
    /*D 是本轮 a、b 确定的情况下最小的距离，D_min 是所有情况最小距离*/
    int k=0;                                 //数组 C 的下标初始为 0
    for (int i=0; i<n; i++)
        for (int j=0; j<m; j++){             //两重循环枚举 a、b
            D=DD(A[i], B[j], C[k]);          //枚举 a、b 得到的初始 D
            while (k<l-1 && DD(A[i], B[j], C[k+1])<D)
                D=DD(A[i], B[j], C[++k]);    //更新下标 k 和距离 D
            if (D<D_min) D_min=D;            //更新 D_min 值
        }
    cout<<D_min;                             //输出最小 D 值
}
```

3）时间复杂度：O(nm) 或 $O(n^2)$　　空间复杂度：O(1)

较优解：一重循环扫描 a，两轮折半查找 b、c

1）本解法可以证明当 a 确定时，b 只能是数组 B 中<a 且最接近 a 的那个数或>=a 且最接近 a 的那个数，c 也只能是数组 C 中<a 且最接近 a 的那个数或>=a 且最接近 a 的那个数，所以只需要枚举 a 并折半查找确定 b 和 c 可能的值然后选择最小的 D。

数组 A、B、C 的长度分别为 n、m、l，一重循环枚举 a=A[i]，通过两次折半查找可以查找到插入位点 j 和 k 保证 B[j]<A[i]<B[j+1]，C[k]<A[i]<C[k+1]（如果插入位点是最左端和最右端另外判断），然后判断<A[i], B[j], C[k]>、<A[i], B[j+1], C[k]>、<A[i], B[j], C[k+1]>、<A[i], B[j+1], C[k+1]>哪种情况 D 最小即可，就能得到 a 确定的情况下最小可能的 D，枚举 a 更新 D_min 即可。最后输出最小值 D_min。

2）算法如下：

```
int DD(int a, b, c){
    return abs(a-b)+abs(b-c)+abs(c-a);
}
int Binary_Search(int A[], L, R, x){
    int mid;
    while (L<R){                    //如果 L>R 则范围错误
        mid=(L+R)/2;                //mid 取中间数，向下取整
        if (x<=A[mid])
            R=mid;
        else L=mid+1;               //更新查找范围
    }
    return L;                       //查找到位置，返回数组下标 L
}
void ans(int A[], n, B[], m, C[], l){     //n、m、l 为三个数组长度
    int D_min=Maxint, D;            //Dmax 赋值为最大整数
    /*D 是本轮 a、b 确定的情况下最小的距离，D_min 是所有情况最小距离*/
    int j, k;
    for (int i=0; i<n; i++){
        j=Binary_Search(B[], 0, m-1, A[i]);    //根据 A[i]折半查找
        k=Binary_Search(C[], 0, l-1, A[i]);    //根据 A[i]折半查找
    对<A[i],B[j-1]或 B[j]或 B[j+1],C[k-1]或 C[k]或 C[k+1]>排列组合求出最小的 D 并与
D_min 比较更新                      //更新 D_min 值
    }
    cout<<D_min;                    //输出最小 D 值
}
```

3）时间复杂度：O(n*log(ml))或 O(nlogn)

空间复杂度：O(1)

最优解：贪心，a、b、c 数组指针后移

1）

数组 A、B、C 的长度分别为 n、m、l，对数组 A、B、C 分别设置变量 i、j、k 存储数组下标不断后移，初始 i=j=k=0，可以证明如果是往后移动，只有当移动 A[i]、B[j]、C[k]最小的那个时，D=2L。才有可能减小，所以每次都只后移动所指元素值最小的对应数组下标变量，一直重复进行，并更新 D 的最小值 D_min，直到所指元素值最小的对应数组无法在后移，最后输出最小值 D_min。

2）算法如下：

```
int DD(int a, b, c){
    return abs(a-b)+abs(b-c)+abs(c-a);
}
void ans(int A[], n, B[], m, C[], l){   //n、m、l 为三个数组长度
    int D_min=Maxint, D;                //Dmax 赋值为最大整数
    /*D 是本轮 a、b 确定的情况下最小的距离，D_min 是所有情况最小距离*/
    int i=j=k=0;                        //数组 C 的下标初始为 0
    while(i<n && j<m && k<l){
        D_min=min(DD(A[i], B[j], C[k]), D_min);   //更新 D_min
        if (A[i]<B[j] && A[i]<C[k])
            i++;
        else
            if (B[j]<C[k])
                j++;
            else k++;
    }
    cout<<D_min;                        //输出最小 D 值
}
```

3）时间复杂度：O(n+m+l)或 O(n)　　空间复杂度：O(1)

6.2 补充习题

1．已知一个整数序列 A = (a_0, a_1,…, a_{n-1})，其中 0≤a_i < n（0≤i < n）。从数组中选择一些数，要求选择的数中任意两数差的绝对值<=k。设计一个在时间上尽可能高效的算法输出最多可能选择的元素个数。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C、C++或 Java 语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

顺序表

暴力解：枚举，枚举最小元素　　时间：$O(n^2)$　　空间：O(1)

最优次优解：快速排序，然后数组指针后移　　时间：O(nlogn)　　空间：O(logn)

最优解：空间换时间　　时间：O(n)　　空间：O(n)

暴力解：枚举，枚举最小元素

1）假设我们选择的数中最小的那个为 A[i]，那么我们可以选择的数范围就是 A[i]~ A[i]+k，只要确定了 A[i]再访问一遍数组就能够算出有多少个可选元素 m，所以使用两层循环，外层 i 遍历数组，内层 j 再遍历一次整个数组判断可选元素个数，选择最多可能的可选元素，输出 m。

2）算法如下：

```
void ans(int A[], n, k){
int m;                          //m 是本次遍历选择元素数
int m_max=0;                    //m_max 初值为 0
for (int i=0; i<n; i++){        //A[i]为本次枚举最小元素
   m=0;                         //每次枚举时都需要清空 m
   for (int j=0; j<n; j++)
   if (A[j]>=A[i] && A[j]<=A[i]+k)
      m++;                      //A[j]处于选择范围 m++
   m_max=max(m_max, m);         //如果 m 大则更新 m_max
}
   cout<<m_max;                 //输出最大 m 值
}
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

次优解：快速排序，然后数组指针后移

1）先快速排序将 A 数组变成升序，设置数组指针 i 和 j（初始为 0），枚举 i 从 0~n-1，j 从上一轮 i 的 j 开始往后移动，如果 A[j+1]<=A[i]+k 说明 A[j+1]仍然处于 A[i]~A[i]+k 这个范围，此时 j 后移，直到 j 移动到最后一个元素说明已经不会有更大的 m=j-i+1。

2）算法如下：

```
void Qsort(int A[], L, R){       //a 数组保存数据，L 和 R 是边界
    if (L>=R)
       return;                    //当前区间元素个数<=1 则退出
    int pivot, i=L, j=R;          //i 和 j 是左右两个数组下标移动
    swap(A[L], A[m]);             //m 是 L~R 之间随机数（快排优化）
    pivot=A[L];                   //pivot 作为基准值参与比较
    while (i<j){
        while (i<j && A[j]>pivot)
            j--;
        while (i<j && A[i]<=pivot)
            i++;
        if (i<j)
            swap(A[i], A[j]);     //交换 A[i]和 A[j]
    }
    swap(A[L], A[i]);             //将 A[L]放入最终位置
    Qsort(A, L, i-1);             //递归处理左区间
    Qsort(A, i+1, R);             //递归处理右区间
}
void ans(int A[], n, k){
    int m;                        //m 是本次遍历选择元素数
    int m_max=0;                  //m_max 初值为 0
    int i=j=0;                    //A[i~j]是选择的元素
    Qsort(A, 0, n-1);             //快速排序
    for (i=0; i<n; i++){
        while (j+1<n && A[j+1]<=A[i]+k)
            j++;                  //j+1 也在范围内，R 后移
        m=j-i+1;                  //求出 m
        m_max=max(m_max, m);      //如果 m 大则更新 m_max
        if (j==n-1)
            break;                //j 指到最后一个元素，不会有更大的 m
    }
    cout<<m_max;                  //输出最大 m 值
}
```

3）时间复杂度：O(nlogn)　　空间复杂度：O(logn)

（快排的时间复杂度为 O(nlogn)，指针后移的时间复杂度为 O(n)，所以总时间复杂度是 O(nlogn)。）

最优解：空间换时间

1）数组中每个元素大小都在 0~n-1，可以使用数组 B[n]表示每个值出现的次数，扫描一次整个数组统计每个数出现的次数，然后 R 从 0 扫描到 n-1，维持一个队列，队头 L，队尾 R，保证这个队列长度不超过 k，m 表示队列中元素出现次数的总和，此时可取数的范围就是[L~R]，一个新元素 R 入队后（m+=B[R]），如果队列长度超过 k 了则将 L 出队（m-=B[L]），统计最大的 m 即最大队列中元素出现的次数和。

2）算法如下：

```
void ans(int A[], n, k){
    int m;                      //m 是本次取值范围为 L~R 选择元素数
    int m_max=0;                //m_max 初值为 0
    for (int i=0; i<n; i++)
        B[A[i]]++;              //A[i]出现次数+1
    int L=R=0;
    for (R=0; R<n; R++){
        m+=B[R];                //m 加上 B[R]
        if (R-L>k){             //如果范围差大于 k 则 L 右移
            m-=B[L];            //此时范围是[L+1,R]，m 减去 B[L]
            L++;
        }
        m_max=max(m_max, m);    //如果 m 大则更新 m_max
    }
    cout<<m_max;                //输出最大 m 值
}
```

3）时间复杂度：O(n)　　空间复杂度：O(n)

2 .在数组中，某个数字减去它右边的数字得到一个数对之差。求所有数对之差的最大值。例如，在数组{2, 4, 1, 16, 7, 5, 11, 9}中，数对之差的最大值是 11，是 16 减去 5 的结果。设计一个在时间上尽可能高效的算法输出所有数对之差的最大值。

（1）给出算法的基本设计思想。

（2）根据设计思想，采用 C 或 C++语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间和空间复杂度。

顺序表

暴力解：枚举，枚举数对中的两个元素　　时间：$O(n^2)$　　空间：O(1)

最优解：以空间换时间，保存 i 之前的最大元素　　时间：O(n)　空间：O(1)

暴力解：枚举，枚举数对中的两个元素

1）使用两层循环，外层 i 遍历数组，内层 j 再从 i+1 开始遍历一次整个数组（即我们默认 i<j），A[i]是数对中的前一个元素，A[j]即是数对中的后一个元素，求出 m=A[i]和 A[j]之差的绝对值，输出最大的 m。

2）算法如下：

```
void ans(int A[], n, k){
int m;                              //m 是数对之差的绝对值
int m_max=Min_int;                  //m_max 初值为最小 int
for (int i=0; i<n-1; i++)           //枚举前一个元素
   for (int j=i+1; j<n; j++){       //枚举后一个元素
      m=A[i]-A[j];                  //A[i]和 A[j]之差的绝对值
      m_max=max(m_max, m);          //如果 m 大则更新 m_max
   }
   cout<<m_max;                     //输出最大 m 值
}
```

3）时间复杂度：$O(n^2)$　　空间复杂度：O(1)

最优解：以空间换时间，保存 i 之前的最大元素

1）j 遍历一遍数组，t 保存 A[0~j-1]的最大元素，相当于枚举 j 作为数对后一个元素，而 t 作为数对第一个元素，数对差为 m=t-A[j]，保存最大数对差 m_max。

2）算法如下：

2）算法如下：

```
int ans(int A[], n, key){
    for (int i=0; i<n; i++)              //枚举 i
    if (A[i]==key)
        return i;                        //查找成功
    return -1;                           //查找失败
}
```

3）时间复杂度：O(n)　　空间复杂度：O(1)

最优解：折半查找两次

1）A[0~n/2]递减，所以先折半查找一次；A[n/2~n-1]递增，所以再折半查找一次。（注意因为一个是降序一个是升序，这两次写法会有不同，也可以合成一种写法。）

2）算法如下：

```
int Binary_Search_up(int A[], L, R, x){
    int mid;
    while (L<R){                        //如果 L>R 则范围错误
        mid=(L+R)/2;                    //mid 取中间数，向下取整
        if (x<=A[mid]) R=mid;           //注意是升序
        else L=mid+1;                   //更新查找范围
    }
    return L;                           //查找到位置，返回数组下标 L
}
int Binary_Search_down(int A[], L, R, x){
    int mid;
    while (L<R){                        //如果 L>R 则范围错误
        mid=(L+R)/2;                    //mid 取中间数，向下取整
        if (x>=A[mid]) R=mid;           //注意是降序
        else L=mid+1;                   //更新查找范围
    }
    return L;                           //查找到位置，返回数组下标 L
}
void ans(int A[], n, key){
    int i;
    i=Binary_Search_down(A, 0, n/2, key);   //折半查找降序序列
    if (A[i]==key)
        return i;                       //折半查找降序成功
    i=Binary_Search_up(A, n/2+1, n-1, key); //折半查找升序序列
    if (A[i]==key)
      return i;                         //折半查找升序成功
    return -1;                          //查找失败
}
```

3）时间复杂度：O(logn)　　空间复杂度：O(1)

7 问答环节

Q：如何使用模板，自己整理的模板与老师给的模板不一样怎么办？

一休：一定要记住的是快排和折半查找的模板，其他的可以自己发挥，很多算法的写法不一样，自己整理的只要保证正确当然更好，毕竟模板是为了节省考试时间，一般情况模板都可以直接作为一个过程使用，只要调整参数即可，但注意如 2016 年的类快排思想就需要在快排基础上进行修改。

Q：那些线性表里的栈，队列，循环链表会考算法吗

一休：算法题不会考，重要的是用，重要的是怎么使用，比如说队列在 BFS 中就需要使用，很多时候不是非得用某种方法才行，队列那我可以用数组表示也可以直接调用，当然这只是针对算法题，应用题有可能会考各种定义什么的。