一、查找

1、二分查找（折半查找）

【例题】适用于折半查找的表的存储方式及元素排列要求为( )

链接方式存储，元素无序

链接方式存储，元素有序

顺序方式存储，元素无序

顺序方式存储，元素有序

【例题】二分查找要求节点 ()

有序、顺序存储

有序、链接存储

无序、顺序存储

无序、链接存储

【例题】对无序表用折半法查找比顺序查找快()

对

错

对于无序的顺序表，根本不能用二分查找

【例题】下面关于二分查找的叙述中正确的是：

表必须有序,表可以顺序方式存储,也可以链表方式存储

表必须有序且表中数据必须是整型,实型或字符型

表必须有序,而且只能从小到大排列

表必须有序,且表只能以顺序方式存储

【例题】用向量和单链表示的有序表均可使用折半查找方法来提高查找速度()

对

错

折半查找属于随机访问特性 链表不行

堆排序也不能用链表 因为调整堆时没法随机访问底层孩子节点

快速排序可以链表

归并排序可用链表

基数排序可用链表

插入排序链表比数组要快一些 减少移动次数

【例题】若有序表的关键字序列为（b,c,d,e,f,g,q,r,s,t），则在二分查找关键字b的过程中，先后进行的关键字依次为：

f,c,b

f,d,b

g,c,b

g,d,b

第一次下标是（0 + 9）/2 = 4

第二次下标是（0+3）/2 = 1

最后一次下标是0

【例题】对有序数组{2、11、15、19、30、32、61、72、88、90、96}进行二分查找，则成功找到15需比较（）次

3

4

2

5

0 1 2 3 4 5 6 7 8 9 10

2 11 15 19 30 32 61 72 88 90 96

第一次 index=（0+10）/2=5 对应32 ； 比15大 所以下次范围是 0到4

第二次 index=（0+4）/2=2 对应15 找到

【例题】若有一个顺序有序表A[1:18] 中有18个元素，现进行二分查找，则查找 A［3］的比较序列的下标依次为（ ）。

1，2，3

9，5，2，3

9，5，3

9，4，2，3

【例题】折半查找有序表（4,6,10,12,20,30,50,70,88,100）。若查找表中元素58，则它将依次与表中哪些元素比较大小，查找结果失败（）

20，50

30，88，70，50

30，88，50

20，70，30，50

【例题】考虑以下二分查找的代码：

int bsearch(int array[], int n, int v)

{

int left, right, middle;

left = 0, right = n - 1;

while (left <= right) {

middle = left + (right - left) / 2;

if (array[middle] > v ) {

right = middle;

} else if (array[middle] < v) {

left = middle;

} else {

return middle;

}

}

return -1;

}

对于输入array为：{2, 6, 8, 10, 13, 25, 36, 45, 53, 76, 88, 100, 127}， n = 13, v = 127时,

运行bsearch函数，while循环调用的次数为\_\_\_\_。

1

3

4

5

6

无数次

本题是一个坑，这是不正确的二分查找法，当中间的没有找到的时候，left和right下标应该左移或者右移（left++,right--），实质是（ right = middle-1，left=middle+1），才不会出现除以二向下取值的时候出现无限循环。

【例题】在长度为n的有序线性表中进行二分查找，最坏情况下需要比较的次数是（ ）。

http://115.28.168.160/CourseFile/MSBGB/ms2-3-4.png

2、顺序查找

【例题】对于静态表的顺序查找法，若在表头设置监视哨，则正确的查找方式为（）。

从第0个元素往后查找该数据元素

从第1个元素往后查找该数据元素

从第n个元素往开始前查找该数据元素

与查找顺序无关

常把第一个或最后一个元素作为哨兵。

表头设置监视哨，就是将空出来的下标为0的这个元素的值设为Key。

这样我们就不用多次判断 i 是否越界，因为就算静态表中找不到，也会在0位置上配对成功，return 0

n个元素都要比较一次，但都不成功，最后监视哨也要比较一次，比较成功，一共比较n+1次。

例子：有5个元素，分别是1，2，3，4，5，要找的元素是8。那么8就是监视哨，数列如下：

8，1，2，3，4，5。

从5开始向前查找，一共要比较6次，比较到监视哨成功，监视哨所在的下标是0，所以返回值为0.

【例题】若查找每个记录的概率均等,则在具有n个记录的连续顺序文件中采用顺序查找法查找一个记录,其平均查找长度ASL为()

(n-1)/2

n/2

(n+1)/2

n

【例题】顺序查找的平均时间是多少？

n/2

n

n \* n

log n

【例题】对大小均为n的有序表和无序表分别进行顺序查找,在等概率查找的情况下,对于查找成功,它们的平均查找长度是相同的,而对于查找失败,它们的平均查找长度是不同的()

对

错

因为所有位置等概率查找，顺序查找的话，都是一个一个查找过去。

如果成功，查找长度就相同。

但是如果失败，那么有序表可以尽早退出，而无序表必须全部遍历完，查找长度就有差异了。

3、分块查找

【例题】设顺序线性表的长度为30，分成5块，每块6个元素，如果采用分块查找并且索引表和块内均采用顺序查找，则其平均查找长度为( )。

6

11

5

6.5

分块查找会分两部分进行,第一步先进行索引表查找判断其在那个字表中,第二步然后进行在字表中的查找

索引表有5个元素 所以平均查找长度为:(1+5)/2=3

字表中有6个元素,所以平均查找长度为:(1+6)/2=3.5

所以总的平均查找长度为3+3.5=6.5

【例题】如果要求一个线性表既能较快的查找，又能适应动态变化的要求，则可采用( )查找法。

分快查找

顺序查找

折半查找

基于属性

【例题】适于对动态查找表进行高效率查找的组织结构是分块有序表()

对

错

静态查找表只进行以下2个操作：

1.查找某个“特定”数据元素是否在查找表中

2.查找某个“特定”数据元素的各种属性

有序表、分块有序表、线性链表都是静态查找表

性能分析：平均查找长度：（当查找关键字等概率时）ASL = 1/(n+1)

动态查找表:表结构是在查找过程中动态生成的，通俗解释，对于给定key,若表中存在某关键字与key相等则查找成功返回，若未找到则插入关键字等于key的记录。

二叉排序树、平衡二叉树、B树、B+树都是动态查找。（对查找表进行插入和删除操作---即为动态的）

4、三种查找的比较

【例题】就平均查找长度而言,分块查找最小,折半查找次之,顺序查找最大()

对

错

分块查找：

1.将顺序表分为若干块，除最后一块，前面每块元素相等，块间有序，块内无序

2.索引表内元素有序，用二分折半查找，每块内元素无序，用顺序查找

3.所以分块查找介于折半查找和顺序查找之间

5、其他查找算法

【例题】给定一个整数sum,从有N个有序元素的数组中寻找元素a,b,使得a+b的结果最接近sum,最快的平均时间复杂度是：

O(n)

O(nlogn)

O(n^2)

O(logn)

思想类似于两端向中间扫描

1、设定两个指针P1、P2，分别指向数组开始和结尾，即P1指向最小值，P2指向最大值；

2、计算 \*P1+\*P2 的值为 SUM，与 sum 比较，记录它们的差值 DIF 和 SUM，若 SUM<sum，P1++，若SUM>sum，P2--；

3、重复以上过程直到DIF最小。（保持住DIF最小的那一对，新DIF大于旧DIF则不更新）

【例题】在一个有8个int数据的数组中，随机给出数组的数据，找出最大和第二大元素一定需要进行（）次比较：

8

9

10

11

比如 A B C D E F G H，通过8进4的方式，A与B比较， C与D比较.....然后再4进2，A与C比较（假设A,C比B,D大），E与G比较。再2进1，比如A与E比较（假设A，E比C,G大）选出最大的A，总共7次。

然后次大的数一定是被最大数PK下去的，所以再选B C E三个比较2次得到次大的数

A

A E

A C E G （7次）

A B C D E F G H

再选 BCE中最大的（2次），共9次，不过可以这个方法比较次数是少一点，但是所需要的空间大，要记下与沿途的最大值比较的数。

分析：这是一个考最优算法的下界问题。

选择问题的复杂度下界，已经有证明，可参考算法导论或屈婉玲的算法设计与分析技术这本书。

对于选择问题，找最大问题的下界是：n-1

找第二大问题的下界是：n+logn-2

因此，本题，n=8，代入下界公式：8+3-2 = 9，所以选择B.

6、字符串匹配算法KMP算法

【例题】字符串'ababaabab'的nextval为()

(0,1,0,1,0,4,1,0,1)

(0,1,0,1,0,2,1,0,1)

(0,1,0,1,0,0,0,1,1)

(0,1,0,1,0,1,0,1,1)

i 0 1 2 3 4 5 6 7 8

s a b a b a a b a b

next[i] -1 0 0 1 2 3 1 2 3

先计算前缀next[i]的值：

next[i]的值主要是看s[i]之前的字符串中重复的子串长度。next[0] = -1，定值。

next[1]是看s[1]之前的字符串“a”中重复的子串长度为0，故next[1] = 0。

next[2]是看s[2]之前的字符串“ab”中重复的子串长度为0，故next[2] = 0。

next[3]是看s[3]之前的字符串"aba"中重复的子串长度，s[0]与s[2]重复，长度为1，故next[3] = 1。

next[4]是看s[4]之前的字符串"abab"中重复的子串长度，s[01]与s[23]重复，长度为2，故next[4] = 2。

next[5]是看s[5]之前的字符串"ababa"中重复的子串长度，s[012]与s[234]重复，长度为3，故next[5] = 3。

next[6]是看s[6]之前的字符串"ababaa"中重复的子串长度，s[0]与s[5]重复(因为多了一个a，无法找到长度为3的重复字符串，这只能是s[0]和s[5]重复)，长度为1，故next[6] = 1。

同样的，求next[7]和next[8]分别为2和3。

接下来计算nextval[i]的值：

nextval[i]的求解需要比较s中next[i]所在位置的字符是否与s[i]的字符一致，如果一致则用s[next[i]]的nextval的值作为nextval[i]，如果不一致，则用next[i]做为nextval[i]。

nextval[0] = -1,和next[0]的值一样。

nextval[1]，比较s[next[1]] ?= s[1]，next[1] = 0，s[0] = a，而s[1] = b，二者不一致，则nextval[1] = next[1] = 0。

nextval[2]，比较s[next[2]] ?= s[2]，next[2] = 0，s[0] = a，而s[2] = a，二者一致，则nextval[2] = nextval[s[next[2]]] = nextval[s[0]] = -1(严谨来看这么表述是有问题的，因为nextval[2]表示nextval数组中 第3个数值，而nextval[s[0]]表示的是s[0]对应的字母‘a’所对应的nextval值 -1，这里nextval[]的用法并不严谨，只是为了表述方便 )。

nextval[3]，比较s[next[3]] ?= s[3]，next[3] = 1，s[1] = b，而s[3] = b，二者一致，则nextval[3] = nextval[s[next[3]]] = nextval[s[1]] = 0。

nextval[4]，比较s[next[4]] ?= s[4]，next[4] = 2，s[2] = a，而s[4] = a，二者一致，则nextval[4] = nextval[s[next[4]]] = nextval[s[2]] = -1。

nextval[5]，比较s[next[5]] ?= s[5]，next[5] = 3，s[3] = b，而s[5] = a，二者不一致，则nextval[5] = next[5] = 3。

同样的求nextval[6]，nextval[7]，nextval[8]分别为 0 ，-1 ， 0。

这里是nextval的下标从-1开始，如果从1开始，则其余各位均＋1，nextval为0,1,0,1,0,4,1,0,1

【例题】KMP算法下，长为n的字符串中匹配长度为m的子串的复杂度为（）

O（N）

O（M+N）

O（M+LOGM）

O（N+LOGM）

kmp算法完成的任务是

给定两个字符串O和f，长度分别为n和 m，判断f是否在O中出现，如果出现则返回出现的位置。常规方法是遍历O的每一个位置，然后从该位置开始和f进行匹配，但是这种方法的复杂度是 O(nm)。kmp算法通过一个O(m)的预处理，使匹配的复杂度降为O(n+m)。

kmp算法思想

我们首先用一个图来描述kmp算法的思想。在字符串O中寻找f，当匹配到位置i时两个字符串不相等，这时我们需要将字符串f向前移动。常规方法是每次向前移动一位，但是它没有考虑前i-1位已经比较过这个事实，所以效率不高。事实上，如果我们提前计算某些信息，就有可能一次前移多位。假设我们根据已经获得的信息知道可以前移k位，我们分析移位前后的f有什么特点。我们可以得到如下的结论：

A段字符串是f的一个前缀。

B段字符串是f的一个后缀。

A段字符串和B段字符串相等。

所以前移k位之后，可以继续比较位置i的前提是f的前i-1个位置满足：长度为i-k-1的前缀A和后缀B相同。只有这样，我们才可以前移k位后从新的位置继续比较。



所以kmp算法的核心即是计算字符串f每一个位置之前的字符串的前缀和后缀公共部分的最大长度（不包括字符串本身，否则最大长度始终是字符串本身）。获得f每一个位置的最大公共长度之后，就可以利用该最大公共长 度快速和字符串O比较。当每次比较到两个字符串的字符不同时，我们就可以根据最大公共长度将字符串f向前移动(已匹配长度-最大公共长度)位，接着继续比较下一个位置。事实上，字符串f的前移只是概念上的前移，只要我们在比较的时候从最大公共长度之后比较f和O即可达到字符串f前移的目的。



7、二元查找树

【例题】二元查找树的任何结点的左右子树都是二元查找树()

对

错

二元查找树：它首先要是一棵二元树，在这基础上它或者是一棵空树；或者是具有下列性质的二元树：（1）若左子树不空，则左子树上所有结点的值均小于它的根结点的值； （2）若右子树不空，则右子树上所有结点的值均大于它的 根结点的值；（3）左、右子树也分别为二元查找树。

8、几种查找算法的比较

【例题】既希望较快的查找又便于线性表动态变化的查找方法是()

顺序查找

折半查找

索引顺序查找

哈希法查找

索引查找是在索引表和主表(即线性表的索引存储结构)上进行的查找。索引查找的过程是：首先根据给定的索引值K1，在索引表上查找出索引值等于K1的索引项，以确定K1对应的子表在主表中的开始位置和长度，然后再根据给定的关键字K2，在对应的子表中查找出关键字等于K2的元素(结点)。

对索引表或子表进行查找时，若表是顺序存储的有序表，则既可进行顺序查找，也可进行二分查找。否则只能进行顺序查找。

9、判断是否有环

【例题】下面哪一方法可以判断出一个有向图是否有环(回路)()

深度优先遍历

拓扑排序

Dijkstra求最短路径

求关键路径

判断是否有环方法：1.拓扑排序

2.深度优先遍历（碰到重复结点）

3.广度优先遍历

10、二叉树遍历

【例题】已知二叉树后序遍历序列是bfegcda，中序遍历序列是badefcg，它的前序遍历序列是：

abcdefg

abdcefg

adbcfeg

abecdfg

二、排序

1、拓扑排序

【例题】任何有向图的结点都可以排成拓扑排序，而且拓扑序列不唯一。（ ）

正确

错误

【例题】有环图也能进行拓扑排序()

对

错

2、排序算法的复杂度

【例题】以下排序中时间复杂度最差的是

归并排序

选择排序

希尔排序

堆排序

|  |  |  |  |
| --- | --- | --- | --- |
| 排序算法 | 最好时间 | 平均时间 | 最坏时间 |
| 桶排序(不稳定) | O(n) | O(n) | O(n) |
| 基数排序(稳定) | O(n) | O(n) | O(n) |
| 归并排序(稳定) | O(nlogn) | O(nlogn) | O(nlogn) |
| 快速排序(不稳定) | O(nlogn) | O(nlogn) | O(n^2) |
| 堆排序(不稳定) | O(nlogn) | O(nlogn) | O(nlogn) |
| 希尔排序(不稳定) |  | O(n^1.25) |  |
| 冒泡排序(稳定) | O(n) | O(n^2) | O(n^2) |
| 选择排序(不稳定) | O(n^2) | O(n^2) | O(n^2) |
| 直接插入排序(稳定) | O(n) | O(n^2) | O(n^2) |

【例题】已知数组元素基本有序的情况下，下面采用那个算法对数组排序时间复杂度最低()

直接选择排序

堆排序

快速排序

插入排序

因为数组元素，基本有序，所以快速排序是最慢的，因为会退化成冒泡排序；选择排序时间复杂度都是O(n^2)，堆排序都是O(nlogn)，但是基本有序对插入排序是最好的，因为这样只需要比较大小，不需要移动，时间复杂度趋近于O(n)。

【例题】以下排序算法时间复杂度为 O(nlogn) 的是（）

冒泡排序

快速排序

堆排序

插入排序

【例题】下列排序算法的常规实现中，哪些空间复杂度是O(1)

冒泡

选择

归并

快排

堆排序

3、排序算法的比较

【例题】下列排序算法中()排序在一趟结束后不一定能选出一个元素放在其最终位置上

选择

起泡

归并

堆

A，选择排序每次选择一个元素放在其最终位置，直道序列有序为止

B，冒泡排序每一趟都将子序列中最小的元素放在子序列的首位，也就是元素的最终位置

D，堆排序，以最小堆为例，堆顶元素是所有元素中最小的，可以一次性放在最终位置

【例题】下面的排序方法中，关键字比较次数与记录的初始排列无关的是\_\_\_\_\_\_

希尔排序

冒泡排序

直接插入排序

直接选择排序

插入排序，冒泡排序和快速排序的排序趟数与序列的初始状态有关

堆排序和选择排序的排序次数与初始状态无关，即最好情况和最坏情况都一样

【例题】下列的排序算法中，初始数据集的排列顺序对算法的性能无影响的是（）

插入排序

堆排序

冒泡排序

快速排序

【例题】下列排序算法中元素的移动次数和关键字的初始排列次序无关的是()

直接插入排序

起泡排序

基数排序

快速排序

选堆归基不变

【例题】下列排序算法不稳定的有?

插入排序

希尔排序

冒泡排序

堆排序

归并排序

快速排序

选择排序

不稳定：快选堆希

稳 定：插冒归基

【例题】下面的排序算法中，稳定是

直接插入排序法

快速排序法

直接选择排序法

堆排序法

【例题】请问对一个排好序的数组进行查找，时间复杂度为（）

O(n)

O(lgn)

O(nlgn)

O(1)



二分查找:n,n/2，n/4…令n/2\*k=1. 得k=log2n（以2为为底的对数）

【例题】基于比较的排序算法是（）

基数排序

冒泡排序

桶排序

希尔排序

四个选项当中，基数排序和桶排序是用运算来确定排序顺序的。冒泡排序和希尔排序是基于比较的排序算法。

选择B、D

4、快速排序

【例题】为实现快速排序算法，待排序序列宜采用的存储方式是（）。

顺序存储

散列存储

链式存储

索引存储

快速排序中查询操作用的较多，而顺序存储适用于频繁查询时使用；链式存储适用于频繁地插入、删除、更新元素时使用。

【例题】下列排序法中，每经过一次元素的交换会产生新的逆序的是（ ）

快速排序

冒泡排序

简单插入排序

简单选择排序

5、外排序

【例题】外排中使用置换选择排序的目的,是为了增加初始归并段的长度()

对

错

6、冒泡排序

【例题】已知用某种排序方法对关键字序列（51，35，93，24，13，68，56，42，77）进行排序时，前两趟排序的结果为

（35，51，24，13，68，56，42，77，93）

（35，24，13，51，56，42，68，77，93）

所采用的排序方法是\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

插入排序

冒泡排序

快速排序

归并排序

7、堆排序

【例题】设有5000个元素，希望用最快的速度挑选出前10个最大的，采用（ ）方法最好。

快速排序

堆排序

希尔排序

归并排序

一般来说：(Top K问题)找出N个数据中前K大(小)的K个数，选堆排

因为堆排序来解决 Top K 问题并不需要全部排序， 只需要维护一个大小为K的最大(小)堆。它的时间复杂度为O(nlogK)

【例题】将整数数组（7-6-3-5-4-1-2）按照堆排序的方式原地进行升序排列，请问在整个排序过程中，元素3的数组下标发生过\_\_\_\_次改变。

0

1

2

3

4

5

8、归并排序

【例题】归并排序中,归并的趟数是()

O(n)

O(logn)

O(nlogn)

O(n\*n)

归并排序是建立在归并操作上的一种有效的排序算法,该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

归并过程为：比较a[i]和a[j]的大小，若a[i]≤a[j]，则将第一个有序表中的元素a[i]复制到r[k]中，并令i和k分别加上1；否则将第二个有序表中的元素a[j]复制到r[k]中，并令j和k分别加上1，如此循环下去，直到其中一个有序表取完，然后再将另一个有序表中剩余的元素复制到r中从下标k到下标t的单元。归并排序的算法我们通常用递归实现，先把待排序区间[s,t]以中点二分，接着把左边子区间排序，再把右边子区间排序，最后把左区间和右区间用一次归并操作合并成有序的区间[s,t]。

故选B

9、基数排序

【例题】对给定的关键字序列110， 119， 007， 911， 114， 120， 122 进行基数排序， 则第 2 趟分配收集后得到的关键字序列是（ ）。

007， 110， 119， 114， 911， 120， 122

007， 110， 119， 114， 911， 122， 120

007， 110， 911， 114， 119， 120， 122

110， 120， 911， 122， 114， 007， 119

基数排序是通过“分配”和“收集”过程来实现排序。

1） 首先根据个位数值（只看个位）来排序：

110 120 911 122 114 007 119

2） 再看十位（只看十位数值大小）来排序：

007 110 911 114 119 120 122

3） 最后看百位：

007 110 114 119 120 122 911

所以 第 2 趟分配收集后得到的关键字序列是 { 007 110 911 114 119 120 122 }。

10、插入排序

【例题】若要求排序是稳定的,且关键字为实数,则在下列排序方法中应选()排序为宜

直接插入

直接选择

堆

快速

基数

直接选择，堆，快速都是不稳定的，只剩下基数和直接插入，而数据是实数，基数排序是分别按照个位，十位，百位...进行的排序，适用于整数，这里关键字是实数，不符合要求。

【例题】输入若已经是排好序的，下列排序算法最快的是（）

插入排序

Shell排序

合并排序

快速排序

【例题】在下列排序方法中,()方法可能出现这种情况:在最后一趟开始之前,所有的元素都不在其最终应在的正确位置上

快速排序

起泡排序

堆排序

插入排序

【例题】对N个数进行排序,在各自最优条件下以下算法复杂度最低的是()

快速排序

堆排序

冒泡排序

插入排序

选择排序

归并排序

11、快速排序

【例题】有字符序列（Q,H,C,Y,P,A,M,S,R,D,F,X）,新序列（F,H,C,D,A,M,P,S,R,Y,Q,X）是下列（ ）排序算法一趟扫描结果。

堆排序

快速排序

希尔排序

冒泡排序

一趟快速排序意思是：寻找一个支点，将该序列位置整个调整一边，可以看到M是支撑点，左边都是比M小的，右边都是比M大的，

注意：支点不一定是左边第一个数，可以任意选的。

【例题】对下列关键字序列用快速排序法进行排序时,速度最快的情形是()

{21,25,5,17,9,23,30}

{25,23,30,17,21,5,9}

{21,9,17,30,25,23,5}

{5,9,17,21,23,25,30}

pivotkey的选择越靠近中央，即左右两个子序列长度越接近，排序速度越快。

21正好是序列的正中，所以排除B，D。

A经过一次排序后结果为9，17，5，（21），25，23，30

C经过一次排序后结果为5，9，17，（21），25，23，30

对于子序列9，17，5和5，9，17，后者在有序状态下用快速排序方法的速度没有前者快，答案为A。

【例题】若一组记录的排序码为（46, 79, 56, 38, 40, 84），则利用快速排序，以第一个记录为基准得到的一次划分结果是

38，40，46，56，79，84

40，38，46，79，56，84

40，38，46，56，79，84

40，38，46，84，56，79

对关键码序列28,16,32,12,60,2,5,72快速排序：

一趟快速排序的算法是：

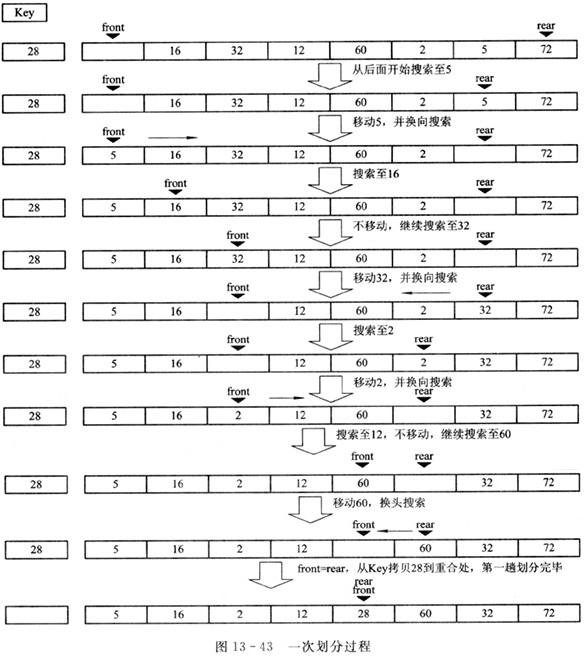
1）设置两个变量i、j，排序开始的时候：i=0，j=N-1；

2）以第一个数组元素作为关键数据，赋值给key，即key=A[0]；

3）从j开始向前搜索，即由后开始向前搜索(j--)，找到第一个小于key的值A[j]，将A[j]和A[i]互换；

4）从i开始向后搜索，即由前开始向后搜索(i++)，找到第一个大于key的A[i]，将A[i]和A[j]互换；

5）重复第3、4步，直到i=j； (3,4步中，没找到符合条件的值，即3中A[j]不小于key,4中A[i]不大于key的时候改变j、i的值，使得j=j-1，i=i+1，直至找到为止。找到符合条件的值，进行交换的时候i， j指针位置不变。另外，i==j这一过程一定正好是i+或j-完成的时候，此时令循环结束）。



【例题】快速排序在已经有序的情况下效率最差，复杂度为（）

O(nlogn)

O(n^2)

O(n^1.5)

【例题】下列哪个算法是对一个list排序的最快方法（）

快速排序

冒泡排序

二分插入排序

线性排序

list采用链式结构存储，在C++ STL中的list采用双向链表存储，比较适合用快速排序进行排序，这是由快速排序不需要随机访问元素的特点决定的。

冒泡排序适合list，但是算法复杂度为O(n^2)，没有快速排序快。

二分插入排序算法适合顺序存储情况，不适合链式存储。

12、选择排序

【例题】排序方法中，从未排序序列中挑选元素，并将其一次插入已排序序列（初始时为空）的一端的方法，称为（）。

希尔排序

归并排序

插入排序

选择排序

关键在于并将其一次插入已排序序列（初始时为空）的一端的方法，这个一端说明了不是插入排序，因为插入位置不一定在端点。

13、枚举排序

【例题】在排序算法中每一项都与其他各项进行比较,计算出小于该项的项的个数,以确定该项的位置叫()

插入排序

枚举排序

选择排序

交换排序

枚举排序，通常也被叫做秩排序，算法基本思想是：对每一个要排序的元素，统计小于它的所有元素的个数，从而得到该元素在整个序列中的位置，时间复杂度为O（n^2）