☆排序

1. 给出内排序、外排序的定义。

解答：内排序是排序过程中参与排序的数据全部在内存中所做的排序，排序过程中无需进行内外存数据传送，决定排序方法时间性能的主要是数据排序码的比较次数和数据对象的移动次数。

外排序是在排序的过程中参与排序的数据太多，在内存中容纳不下，因此在排序过程中需要不断进行内外存的信息传送的排序。

2. 设待排关键字序列为：

{12, 2, 16, 30, 28, 10, 16, 20, 6, 18},

试给出以下排序方法每趟排序后的结果。

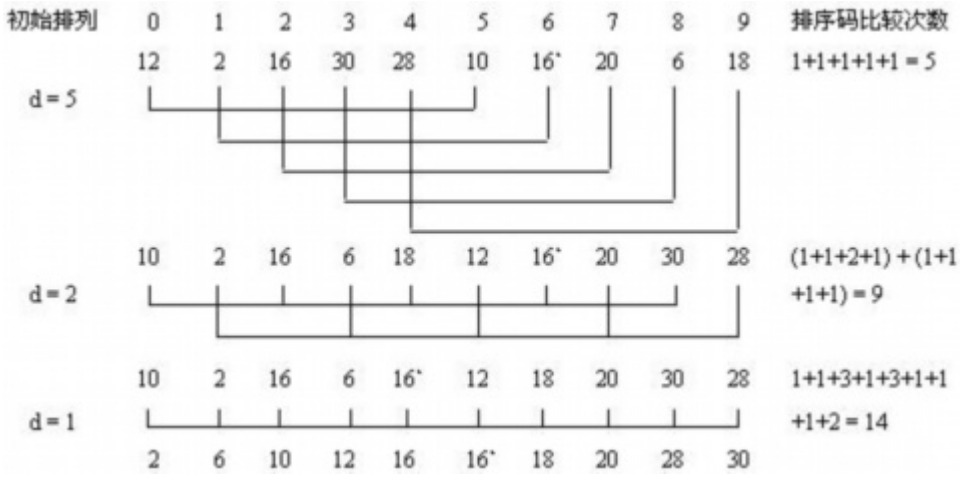
(1)直接插入排序

(2)希尔排序（增量为 5, 2, 1）

（1）直接插入排序

初始排列	0	1	2	3	4	5	6	7	8	9	排序码比较次数
i=1	[12]	2	16	30	28	10	16`	20	6	18	1
i=2	[2	12]	16	30	28	10	16`	20	6	18	1
i=3	[2	12	16]	30	28	10	16`	20	6	18	1
i=4	[2	12	16	30]	28	10	16`	20	6	18	2
i=5	[2	12	16	28	30]	10	16`	20	6	18	5
i=6	[2	10	12	16	28	30]	16`	20	6	18	3
i=7	[2	10	12	16	16`	28	30]	20	6	18	3
i=8	[2	10	12	16	16`	20	28	30]	6	18	3
i=9	[2	6	10	12	16	16`	20	28	30]	18	8
	[2	6	10	12	16	16`	18	20	28	30]	

（2） 希尔排序(增量为 5, 2, 1)



3.举例说明在起泡排序过程中，什么情况下关键字会朝与排序相反的方向移动。在快速排序过程中有这种现象吗？

解答：如果在待排序序列的后面的若干排序码比前面的排序码小，则在起泡排序的过程中，排序码可能向与最终它应移向的位置相反的方向移动。例如：

初始关键字：59　45　10　90

第一趟排序：45　10　59　90

第二趟排序：10　45　59　90

其中 45 在第一趟排序中移向了与最终位置相反的方向。但在快速排序中不会出现这种情况，因为在每趟排序中，比基准元素大的都交换到右边，而比基准元素小的都交换到左边。

4.如果只想在一个 n 个元素的任意序列中得到其中最小的第 k（k<n）个元素之前的部分排序序列，问最好采用什么排序方法？为什么？

解答：一般来讲，当 n 比较大且要选的数据 k<<n 时，采用堆排序方法中的调整算法 FilterDown( )最好。但当 n 比较小时，采用锦标赛排序方法更好。
例如，对于序列{57, 40, 38, 11, 13, 34, 48, 75, 6, 19, 9, 7 }，选最小的数据 6，需形成初始堆，进行 18 次数据比较；选次小数据 7 时，需进行 4 次数据比较；再选数据 9 时，需进行 6 次数据比较;选数据 11 时,需进行 4 次数据比较。但如果选用锦标赛排序,对于有 n(n>0)个数据的序列，选最小数据需进行 n-1 次数据比较，以后每选一个数据，进行数据比较的次数，均需 （log2n） -1 次。

5.试设计一个算法，使得在O(n)时间内重排数组，将所有负值关键字排在所有正值(非负值)关键字之前。

算法思想：因为只需将负数关键字排在前面而无需进行精确排列顺序，因此本算法采用两端扫描的方法，左边扫描到正数时停止，再扫描右边，遇到负数时与左边的当前记录交换，如此交替进行，一趟下来就可以完成排序。

```
void ReSort(SeqList R)
{//重排数组，使负值关键字在前
int i=1,j=n; //数组存放在R[1..n]中
while (i<j) //i<j表示尚未扫描完毕
{ while(i<j&&R[i].key<0) //遇到负数则继续扫描
i++;
R[0]=R[i]; //R[0]为辅助空间
while(i<j&&R[j].key>=0)// 遇到正数则继续向左扫描
j--;
R[i++]=R[j];R[j--]=R[0];//交换当前两个元素并移动指针
}//endwhile
}//ReSort
```

本算法在任何情况下的比较次数均为n(每个元素和0)相比，交换次数少于n/2，总的来说，时间复杂度为O(n).

6.对第2题中的待排关键字序列，试给出以下排序方法每趟排序后的结果。

(1)起泡排序

(2)快速排序

(3)直接选择排序

(4)堆排序

(5)二路归并排序

(6）基数排序

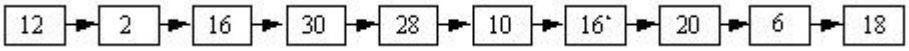
（1）起泡排序

初始排列	0	1	2	3	4	5	6	7	8	9	排序码比较次数
i=0	[12	2	16	30	28	10	16`	20	6	18]	9
i=1	2	[12	6	16	30	28	10	16`	20	18]	8
i=2	2	6	[12	10	16	30	28	16`	18	20]	7
i=3	2	6	10	[12	16	16`	30	28	18	20]	6
i=4	2	6	10	12	[16	16`	18	30	28	20]	5
i=5	2	6	10	12	16	[16`	18	20	30	28]	4
i=6	2	6	10	12	16	16`	[18	20	28	30]	3
	2	6	10	12	16	16`	18	20	28	30	

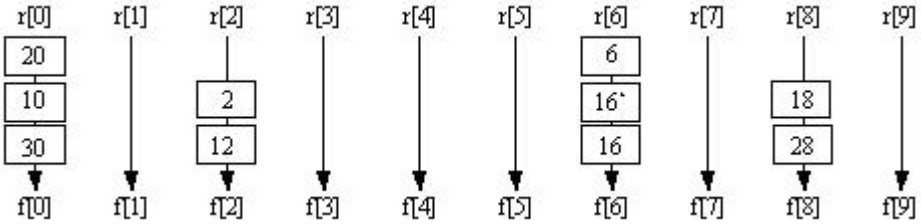
（2） 快速排序

Pivot	Pvtpos	0	1	2	3	4	5	6	7	8	9	排序码比较次数
12	0,1,2,3	**[12** ↑pos	**2** ↑pos	**16** ↑pos	**30** ↑pos	**28**	**10**	**16`**	**20**	**6**	**18]**	9
6	0,1	**[6** ↑pos	**2** ↑pos	**10]**	12	[28	16	16`	20	30	18]	2
28	4,5,6,7,8	[2]	6	[10]	12	**[28** ↑pos	**16** ↑pos	**16`** ↑pos	**20** ↑pos	**30** ↑pos	**18]**	5
18	4,5,6	2	6	10	12	**[18** ↑pos	**16** ↑pos	**16`** ↑pos	**20]**	28	[30]	3
16`	4	2	6	10	12	**[16** · ↑pos	**16]**	18	[20]	28	30	1
		2	6	10	12	16`	[16]	18	20	28	30	

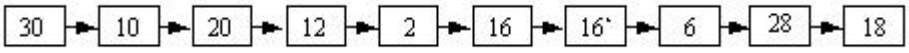
（3）基数排序



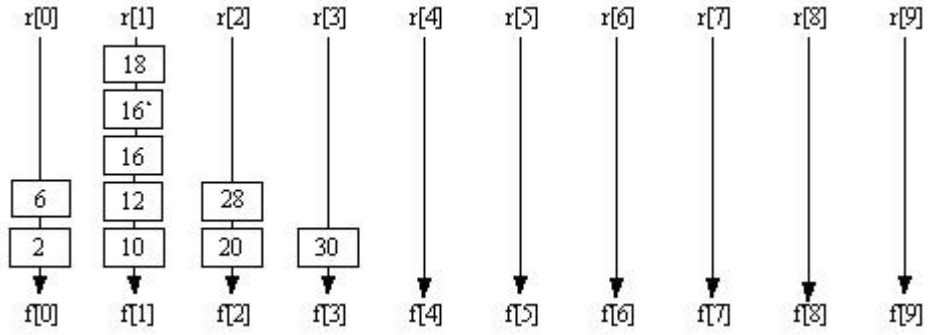
按最低位分配



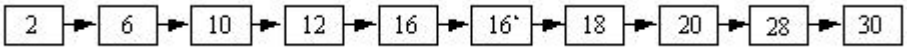
收集



按最高位分配

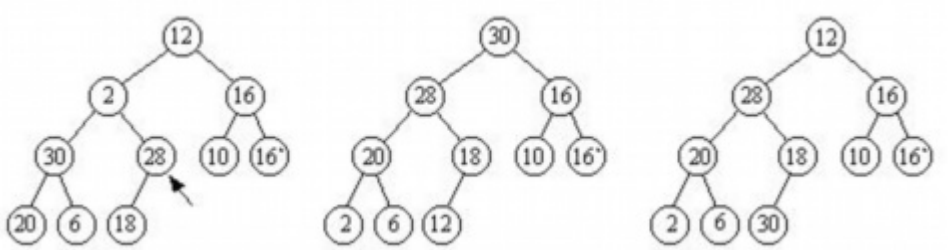


收集

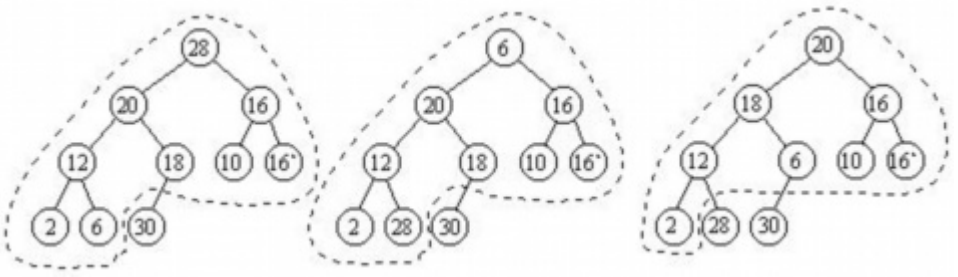


（4） 堆排序

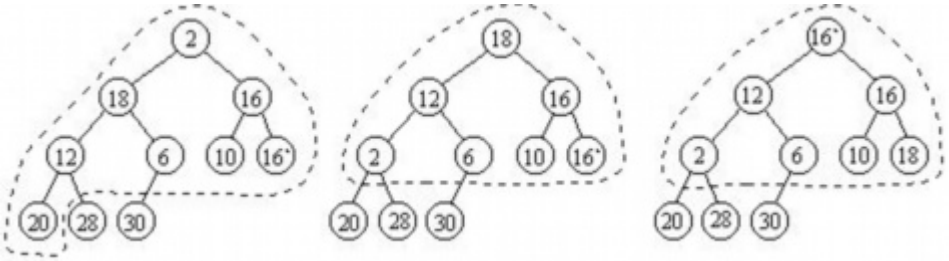
第一步，形成初始的最大堆 (略)，第二步，做堆排序。



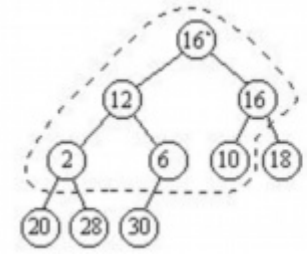
初始排列，不是最大堆　　形成初始最大堆　　交换 $0^{\#}$ 与 $9^{\#}$ 对象



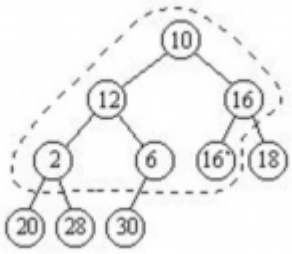
从 0# 到 8# 重新形成堆　　交换 0# 与 8# 对象　　从 0# 到 7# 重新形成堆



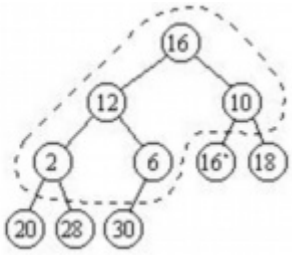
交换 0# 与 7# 对象　　从 0# 到 6# 重新形成堆　　交换 0# 与 6# 对象



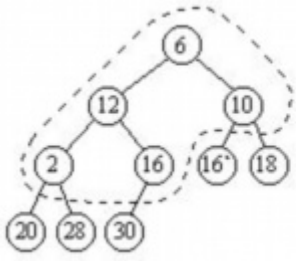
从 0# 到 5# 重新形成堆



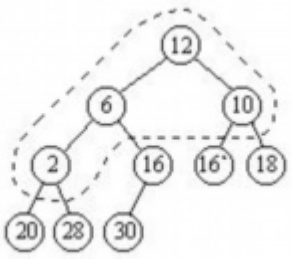
交换 0# 与 5# 对象



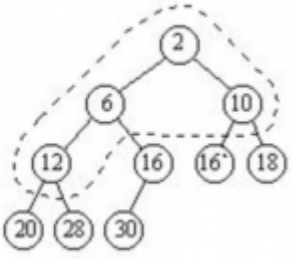
从 0# 到 4# 重新形成堆



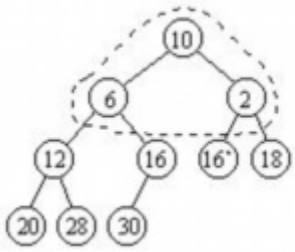
交换 0# 与 4# 对象



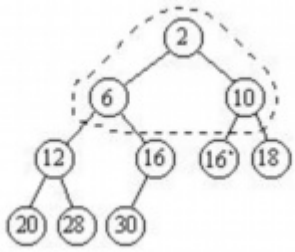
从 0# 到 3# 重新形成堆



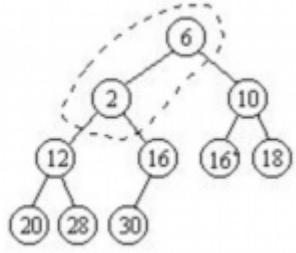
交换 0# 与 3# 对象



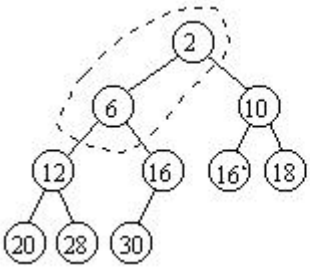
从 0# 到 2# 重新形成堆



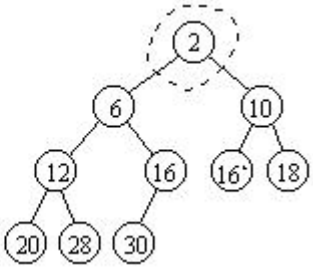
交换 0# 与 2# 对象



从 0# 到 1# 重新形成堆



交换 0# 与 1# 对象



从 0# 到 1# 重新形成堆，得到结果

7.请编写一个算法，在基于单链表表示的待排序关键字序列上进行简单选择排序。

典型错误提示：有同学在以下红色标注处先交换的后继然后再交换的前驱，是不是有问题？画图分析一下显然看出先交换后继的话指针全指乱了，改为先交换前驱后交换后继，OK。

```
lnode * sort(lnode * head)
{
 lnode *p,*q,*max,*s;//p 为一轮比较的头指针
                //max 指向比较过程中最大数的前驱,s 为交换用指针
 p=(lnode *)malloc(LEN);          //建立一个哨兵
 p->next =head;
 head=p;
 while(p->next->next   != NULL)
 {
  max=p ;
  q=p->next ;
     while(q ->next != NULL)
  {
       if(max->next ->exp < q->next ->exp )
          max=q ;
       q=q->next ;
  }
     if (max !=p )
  {
  //交换 p->next,max->next
  s=p->next ;
  p->next = max->next ;
  max->next = s;
        s=p->next ->next ;
  p->next ->next = max->next ->next ;
```

```
 max->next ->next = s;
 //要先交换前驱后交换后继
 }
p=p->next ;
}
s=head;
head=s->next ;
free(s);
return(head);
}
```

8. 在什么条件下，MSD 基数排序比 LSD 基数排序效率更高。

解答：由于 MSD 基数排序是递归的高位优先的排序方法，一般来说，实现的效率较 LSD 基数排序低。但如果待排序码的大小只取决于高位的少数几位并且与大多数低位无关时，采用 MSD 方法比 LSD 方法的效率要高。