第 7 章

1. 选择题

（1）A　（2）C　（3）C　（4）D　（5）D　（6）C　（7）B　（8）C　（9）D　（10）C

2. 判断题

（1）√　（2）X　（3）√　（4）X　（5）X　（6）√　（7）X　（8）√　（9）√　（10）√

3. 简答题

（1）以关键字序列（tim,kay,eva,roy,dot,jon,kim,ann,tom,jim,guy,amy）为例，手工执行以下排序算法（按字典序比较关键字的大小），写出每一趟排序结束时的关键字状态：

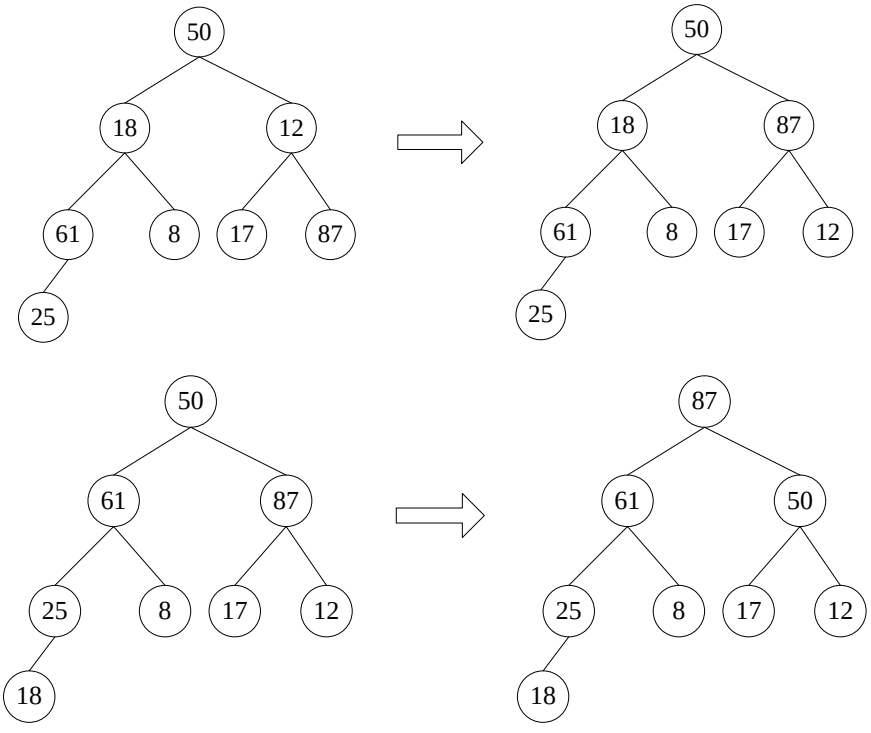
1）直接插入排序；　2）冒泡排序；　3）直接选择排序；

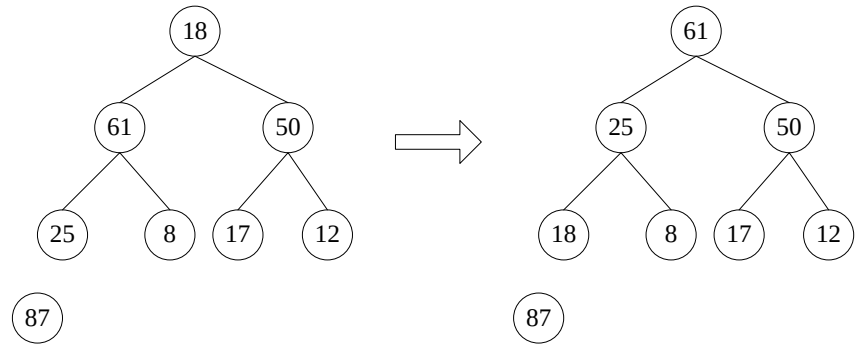
4）快速排序；　5）归并排序；　6）基数排序。

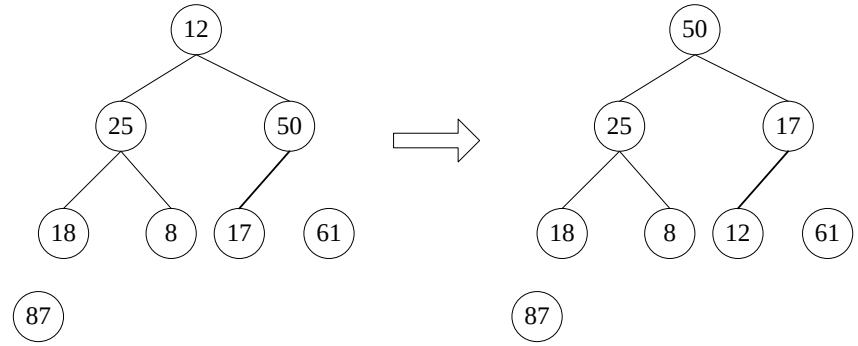
【解答】 略。

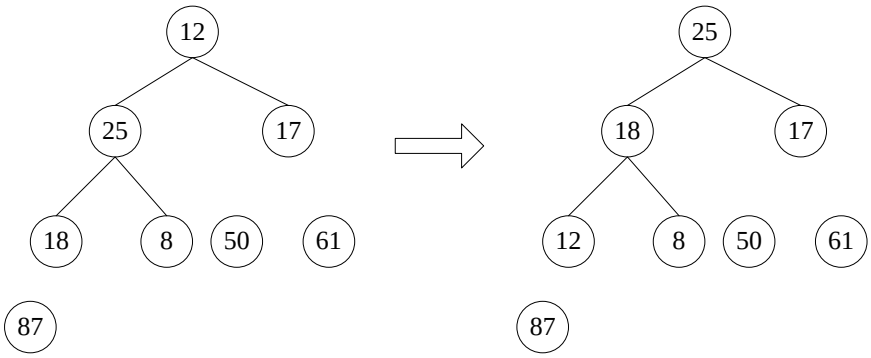
（2）已知序列{50,18,12,61,8,17,87,25}，请给出采用堆排序对该序列做升序排序时的每一趟结果。

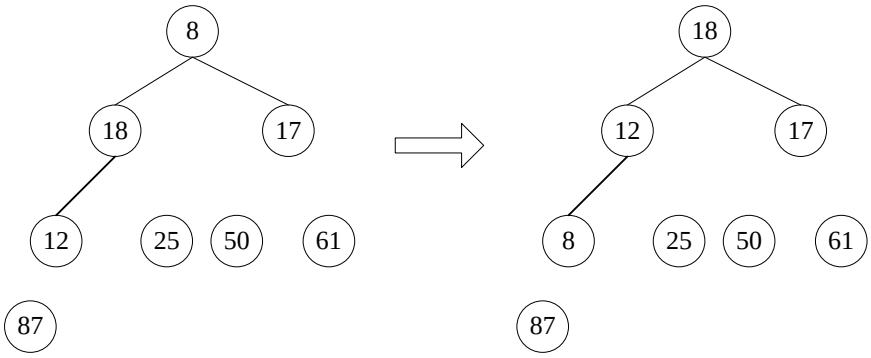
【解答】堆排序过程如下图示：

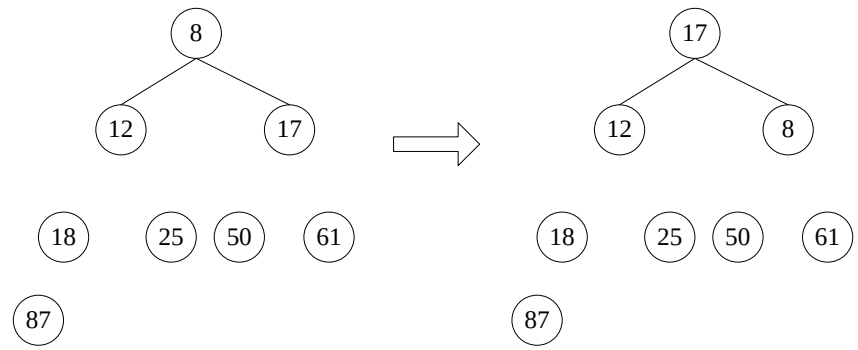


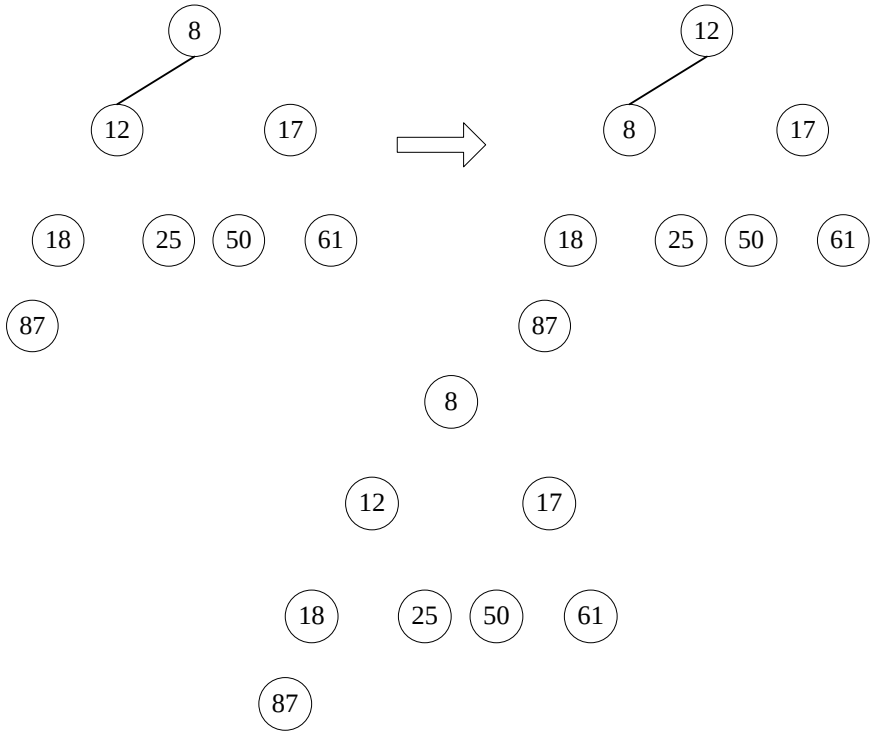

18
61
50
25
8
17
12
87
61
25
50
18
8
17
12
87


12
25
50
18
8
17
61
87
50
25
17
18
8
12
61
87


12
25
17
18
8
50
61
87
25
18
17
12
8
50
61
87


8
18
17
12
25
50
61
87
18
12
17
8
25
50
61
87


8
12
17
18
25
50
61
87
17
12
8
18
25
50
61
87



（3）有 n 个不同的英文单词，它们的长度相等，均为 m，若 n>>50，m<5，试问采用什么排序方法时间复杂度最小？为什么？

【提示】采用基数排序。基数排序是一种借助多关键码排序思想对单关键码进行排序的方法，它适合 n 很大，而关键码较小的序列。本题中英文单词数目 n>>50，而单词长度 m<5，因此采用基数排序方法最佳。

（4）如果只想得到一个含有 n 个元素的序列中第 k（k<<n）小元素之前的部分排序序列，最好采用什么排序方法？为什么？如有这样一个序列：{57,11,25,36,18,80,22}得到其第 3 个最小元素之前的部分序列{11,18,22}，使用所选择的算法实现时，要执行多少次比较？

【解答】采用堆排序。简单选择排序和冒泡排序可以在一趟排序后选出一个最大（或最小）元素，要比较 n-1 次，选次大元素要再比较 n-2 次,…其时间复杂度是 $O(n^2)$。当 k<<n 时，从 n 个元素中选 k 个元素不能使用这种方法。而快速排序、插入排序、归并排序、基数排序等要等到最后才能确定各元素位置。只有堆排序，在未结束全部排序前，可以有部分排序结果。建立堆后，堆顶元素就是最大（或最小）元素，然后，调整堆又选出次大（小）元素。凡要求在 n 个元素中选出 k(k<<n,k>2)个最大（或最小）元素，一般均使用堆排序。因为堆排序建堆比较次数至多不超过 4n,对深度为 k 的堆，在调堆算法中进行的关键字的比较次数至多为 2(k-1)次，且辅助空间为 O(1)。

得到序列{57,11,25,36,18,80,22}的第 3 个最小元素之前的部分序列{11,18,22}共需 14 次比较：其中建堆输出 11 比较 8 次，调堆输出 17 和 25 各需要比较 4 次和 2 次。

（5）阅读下列排序算法，并与已学的算法比较，讨论算法中基本操作的执行次数。

```
void sort(datatype R[], int n)
    {i=1;
     while(i<n-i+1)
        {min=max=1;
         for(j=i+1;j<=n-i+1;++j)
             if(r[j].key<a[min].key) min=j;
             else if(r[j].key>r[max].key)    max=j;
         if(min!=i) {w=r[min];r[min]=r[i];r[i]=w;}
         if(max!=n-i+1)
```

```
            {if (max=i) {w=r[min];r[min]=r[n-i+1];r[N-i+1]=w;}
             else    {w=r[max];r[max]=r[n-i+1];r[N-i+1]=w;}
            }
        i++;
       }
    }
```

【解答】这是一个双向选择排序算法，每次选择关键码最小的记录放在前面，同时选择关键码最大的记录放在后面。比较 n*(n-1)/2 次。最好情况移动记录 0 次，最坏情况大约移动记录 3n 次。

（6）请回答以下关于堆的问题：

1）堆的存储结构是顺序的，还是链式的?

2）设有一个大顶堆，即堆中任意结点的关键码均大于它的左孩子和右孩子的关键码。其具有最大值的元素可能在什么地方?

3）对 n 个元素进行初始建堆的过程中，最多做多少次数据比较?

【解答】1）堆的存储结构是顺序的。

2）堆顶。

3）不超过 4n。

2. 算法设计题

1．请以单链表为存储结构实现简单选择排序的算法。

【提示】每趟从单链表表头开始，顺序查找当前链值最小的结点。找到后，插入到当前的有序表区的最后。

```
void SelectSort (LinkList L)                    /*设链表 L 带头结点*/
   {q=L;                                        /*指向第一数据前驱*/
    while(q->next!=NULL)
       {pl=q->next;
        minp=pl;                                /*minp 指向当前已知的最小数*/
        while(pl->next!=NULL)
           {if(pl->next->data<minp->data)
                minp=pl->next;                  /*找到了更小数*/
            pl=pl->next; }                      /*继续往下找*/
        if(minp!=q->next)                       /*将当前最小数交换到第一个位置上*/
           {rl=minp->next;
            minp->next=rl->next;
            r2=q->next;
            q->next=r2->next;
            r1->next=q->next;
            q->next=r1;
            r2->next=minp->next;
            minp->next=r2;
           }
        q=q>next;
       }
   }
```

2．请以单链表为存储结构实现直接插入排序的算法。

【提示】注意在链式结构上插入元素的实现方式。

```
LinkList Sort (LinkList L)                      /* L 为带头结点的单链表*/
```

```
    {p=L->next->next;                                   /*p 指向第二个结点*/
     L->next->next=NULL;                                /*设置 L 为只含有一个结点的单链表/
     while(p)  /*当结点尚未插入结束时，将结点 p 逐个插入到单链表 L 中去，同时要保持其有序性*/
         {s=p->next;
          pre=L;
          q=L->next;
          while (q&&p->data.key > q->data.key)
              {pre=q;
               q=q->next;}                              /*寻找插入位置*/
               p->next=q;
          pre->next=p;                                  /*插入元素*/
          p=s;
         }
      return(L);
    }
```

3．编写一个双向冒泡的算法，即相邻两遍向相反方向冒泡。

【提示】注意算法的结束条件。

```
typedef struct
   {KeyType key;
    …
   }datatype;
typedef struct
   {datatype elem[MAXSIZE];
     int    length;
   }Stable;                                             /*排序表类型定义*/
void Sort(Stable *L)
   {flag=1;
    j=1;
    while(flag)
        {flag=0 ;
         for( i=j ; i<=L->length-j; i++)                /*正向冒泡*/
             if(L->elem[i].key>L->elem[i+1].key )
               {L->elem [i]<=>L->elem[i+1];
                flag=1;}
         for (i=L->length-j; i>=j+1; i--)               /*反向冒泡*/
             if(L->elem[i].key<L->elem[i-1].key)
               {L->elem[i]<=> L->elem[i-1] ;
                flag=1; }
          j++;
        }
   }
```

4．已知记录序列 a[1..n]中的关键字各不相同，可按如下所述实现计数排序：另设数组 c[1..n]，对每个记录 a[i]，统计序列中关键字比它小的记录个数存于 c[i]，则 c[i]=0 的记录必为关键字最小的记录，然后依 c[i]值的大小对 a 中记录进行重新排列，试编写实现上述排序的算法。

```
void Sort (datatype *A, datatype *B, int n)
```

```
{int C[n+1] ;
  for (i=1 ;i<=n; i++)    C[i]=0;
  for (i=1; i<=n ; i++)
     {for (j=1 ; j<=n ; j++)
        if(A[j].key<A[i].key)    C[i] ++;
     }
  for (i=1; i<=n; i++)
      B[C[i]+1]=A[i] ;
}
```

5．已知奇偶交换排序算法如下描述：第一趟对所有奇数的 i，将 a[i]和 a[i+1]进行比较，第二趟对所有偶数的 i，将 a[i]和 a[i+1]进行比较，每次比较时若 a[i]>a[i+1]，则将二者交换，以后重复上述二趟过程，直至整个数组有序。

（1）试问排序结束的条件是什么？

（2）编写一个实现上述排序过程的算法。

【提示】排序结束的条件是没有交换发生。

```
void Sort( dataype A[n+1])    /*0 号单元不用*/
   {flag=1;
    while(flag)
       {flag=0;
        for(i=1;i+1<=n ; i=i+2)                          /*奇数 i*/
           if(A[i].key>A[i+1].key )
              {flag=1;
               A[i]<=>A[i+1];}
        for(i=0;i+1<=n ; i=i+2)                          /*偶数 i*/
           if(A[i].key>A[i+1].key )
              {flag=1;
               A[i]<=>A[i+1];}
       }
   }
```

6．编写算法，对 n 个关键字取整数值的记录进行整理，以使得所有关键字为负值的记录排在关键字为非负值的记录之前，要求：

（1）采用顺序存储结构，至多使用一个记录的辅助存储空间；

（2）算法的时间复杂度为 O(n)。

【提示】采用类似快速排序的算法来实现。

```
void Process (int A[n])
   {low=0; high=n-1;
    while(low<high)
       {while(low<high&&A[low]<0)    low++;
        while(low<high&&A[high]>0 ) high++;
         if(low<high)
           {A[low]<=>A[high];
            low++;
            high--;}
       }
```

```
}
```

7．序列的“中值记录”指的是：如果将此序列排序后，它是第 n/2 个记录。试编写一个求中值记录的算法。

【提示】注意不要排序。采用类似快速排序的算法实现：以某个序列中第一个元素为基准分割序列为两个子序列，判断最后基准值的位置，若为 n/2，则找到中值记录；若大于 n/2，则在前半区间继续查找；若小于 n/2 则在后半区间继续查找。

```
int fun (Stable *L, int low, int high)
/*以第一个元素为基准分割下标在区间[low, high]内的元素序列*/
    {L->elem[0]=L->elem[low] ;
     K=L->elem[0].key;
     while(low<high)
        {while(low<high && L->elem[high]>=k)  high--;
          L->elem[low]=L->elem[high];
          while (low<high && L->elem[low]<=k) low++;
          L->elem[high]=L->elem[low] ;
        }
     L->elem[low]=L->elem[0];
     return(low) ;
    }
datatype Middle(Stable *L, int low , int high )
    {if (low<high)
        {s=fun(L,low,high);
         if(s==(high-low)/2 )      return (L->elem[s]) ;          /*找到中值记录*/
         if(s>(high-low)/2)       return(Middle(L,low,s-1)) ;    /*在前半区间继续查找*/
          else     return(Middle(L,s+1,high));                  /*在后半区间继续查找*/
        }
    }
```