/\* Note:Your choice is C IDE \*/  
#include "stdio.h"  
#include"stdlib.h"  
#define MAX 4  
void SequenceSearch(int \*fp,int Length);  
void Search(int \*fp,int length);  
void Sort(int \*fp,int length);  
/\*  
================================================  
功能:选择排序  
输入:数组名称(也就是数组首地址)、数组中元素个数  
================================================  
\*/  
void select\_sort(int \*x, int n)  
{  
          int i, j, min, t;  
      for (i=0; i<n-1; i++) /\*要选择的次数:0~n-2共n-1次\*/  
      {  
           min = i; /\*假设当前下标为i的数最小,比较后再调整\*/  
           for (j=i+1; j<n; j++)/\*循环找出最小的数的下标是哪个\*/  
           {  
                if (\*(x+j) < \*(x+min))  
                {     
                     min = j; /\*如果后面的数比前面的小,则记下它的下标\*/  
                }  
       }          
       if (min != i) /\*如果min在循环中改变了,就需要交换数据\*/  
       {  
            t = \*(x+i);  
            \*(x+i) = \*(x+min);  
            \*(x+min) = t;  
       }  
      }  
}  
/\*  
================================================  
功能:直接插入排序  
输入:数组名称(也就是数组首地址)、数组中元素个数  
================================================  
\*/  
  
void insert\_sort(int \*x, int n)  
{  
int i, j, t;  
for (i=1; i<n; i++) /\*要选择的次数:1~n-1共n-1次\*/  
{  
   /\*  
    暂存下标为i的数。注意:下标从1开始,原因就是开始时  
    第一个数即下标为0的数,前面没有任何数,单单一个,认为  
    它是排好顺序的。  
   \*/  
   t=\*(x+i);  
     for (j=i-1; j>=0 && t<\*(x+j); j--) /\*注意:j=i-1,j--,这里就是下标为i的数,在它前面有序列中找插入位置。\*/  
       {  
         \*(x+j+1) = \*(x+j); /\*如果满足条件就往后挪。最坏的情况就是t比下标为0的数都小,它要放在最前面,j==-1,退出循环\*/  
       }  
   \*(x+j+1) = t; /\*找到下标为i的数的放置位置\*/  
}  
}  
/\*  
================================================  
功能:冒泡排序  
输入:数组名称(也就是数组首地址)、数组中元素个数  
================================================  
\*/  
  
void bubble\_sort(int \*x, int n)  
{  
int j, k, h, t;    
for (h=n-1; h>0; h=k) /\*循环到没有比较范围\*/  
{  
   for (j=0, k=0; j<h; j++) /\*每次预置k=0,循环扫描后更新k\*/  
   {  
     if (\*(x+j) > \*(x+j+1)) /\*大的放在后面,小的放到前面\*/  
       {  
         t = \*(x+j);  
         \*(x+j) = \*(x+j+1);  
         \*(x+j+1) = t; /\*完成交换\*/  
         k = j; /\*保存最后下沉的位置。这样k后面的都是排序排好了的。\*/  
       }  
    }  
}  
}  
/\*  
================================================  
功能:希尔排序  
输入:数组名称(也就是数组首地址)、数组中元素个数  
================================================  
\*/  
  
void shell\_sort(int \*x, int n)  
{  
int h, j, k, t;  
for (h=n/2; h>0; h=h/2) /\*控制增量\*/  
{  
   for (j=h; j<n; j++) /\*这个实际上就是上面的直接插入排序\*/  
   {  
    t = \*(x+j);  
     for (k=j-h; (k>=0 && t<\*(x+k)); k-=h)  
      {  
       \*(x+k+h) = \*(x+k);  
      }  
    \*(x+k+h) = t;  
   }  
}  
}  
/\*  
================================================  
功能:快速排序  
输入:数组名称(也就是数组首地址)、数组中起止元素的下标  
================================================  
\*/  
  
void quick\_sort(int \*x, int low, int high)  
{  
int i, j, t;  
if (low < high) /\*要排序的元素起止下标,保证小的放在左边,大的放在右边。这里以下标为low的元素为基准点\*/  
{  
   i = low;  
   j = high;  
   t = \*(x+low); /\*暂存基准点的数\*/  
   while (i<j) /\*循环扫描\*/  
   {  
      while (i<j && \*(x+j)>t) /\*在右边的只要比基准点大仍放在右边\*/  
        {  
         j--; /\*前移一个位置\*/  
        }  
    if (i<j)   
      {  
         \*(x+i) = \*(x+j); /\*上面的循环退出:即出现比基准点小的数,替换基准点的数\*/  
         i++; /\*后移一个位置,并以此为基准点\*/  
      }  
    while (i<j && \*(x+i)<=t) /\*在左边的只要小于等于基准点仍放在左边\*/  
    {  
     i++; /\*后移一个位置\*/  
    }  
    if (i<j)  
    {  
     \*(x+j) = \*(x+i); /\*上面的循环退出:即出现比基准点大的数,放到右边\*/  
     j--; /\*前移一个位置\*/  
    }  
   }  
   \*(x+i) = t; /\*一遍扫描完后,放到适当位置\*/  
   quick\_sort(x,low,i-1);   /\*对基准点左边的数再执行快速排序\*/  
   quick\_sort(x,i+1,high);   /\*对基准点右边的数再执行快速排序\*/  
}  
}  
/\*  
================================================  
功能:堆排序  
输入:数组名称(也就是数组首地址)、数组中元素个数  
================================================  
\*/  
/\*  
功能:渗透建堆  
输入:数组名称(也就是数组首地址)、参与建堆元素的个数、从第几个元素开始  
\*/  
void sift(int \*x, int n, int s)  
{  
int t, k, j;  
t = \*(x+s); /\*暂存开始元素\*/  
k = s;   /\*开始元素下标\*/  
j = 2\*k + 1; /\*右子树元素下标\*/  
while (j<n)  
{  
   if (j<n-1 && \*(x+j) < \*(x+j+1))/\*判断是否满足堆的条件:满足就继续下一轮比较,否则调整。\*/  
   {  
    j++;  
   }  
   if (t<\*(x+j)) /\*调整\*/  
   {  
    \*(x+k) = \*(x+j);  
    k = j; /\*调整后,开始元素也随之调整\*/  
    j = 2\*k + 1;  
   }  
   else /\*没有需要调整了,已经是个堆了,退出循环。\*/  
   {  
    break;  
   }  
}   
\*(x+k) = t; /\*开始元素放到它正确位置\*/  
}  
/\*  
功能:堆排序  
输入:数组名称(也就是数组首地址)、数组中元素个数  
\*/  
void heap\_sort(int \*x, int n)  
{  
int i, k, t;  
//int \*p;  
for (i=n/2-1; i>=0; i--)  
{  
   sift(x,n,i); /\*初始建堆\*/  
}   
for (k=n-1; k>=1; k--)  
{  
   t = \*(x+0); /\*堆顶放到最后\*/  
   \*(x+0) = \*(x+k);  
   \*(x+k) = t;  
   sift(x,k,0); /\*剩下的数再建堆\*/   
}  
}  
/\*构造随机输出函数类\*/  
void input(int a[]){  
int i;  
srand( (unsigned int)time(NULL) );  
for (i = 0; i < 4; i++)  
{  
   a[i] = rand() % 100;  
}  
printf("\n");  
}  
/\*构造键盘输入函数类\*/      
/\*void input(int \*p)  
{  
     int i;  
     printf("请输入 %d 个数据 :\n",MAX);  
      for (i=0; i<MAX; i++)  
      {  
       scanf("%d",p++);   
      }  
      printf("\n");  
}\*/  
/\*构造输出函数类\*/  
void output(int \*p)  
{  
     int i;  
     for ( i=0; i<MAX; i++)  
        {  
           printf("%d ",\*p++);  
        }      
}  
  
  
// 归并排序中的合并算法  
void Merge(int a[], int start, int mid, int end)  
{  
     int i,k,j, temp1[10], temp2[10];  
     int n1, n2;  
     n1 = mid - start + 1;  
     n2 = end - mid;  
  
     // 拷贝前半部分数组  
     for ( i = 0; i < n1; i++)  
     {  
         temp1[i] = a[start + i];  
     }  
     // 拷贝后半部分数组  
     for (i = 0; i < n2; i++)  
     {  
         temp2[i] = a[mid + i + 1];  
     }  
     // 把后面的元素设置的很大  
     temp1[n1] = temp2[n2] = 1000;  
     // 逐个扫描两部分数组然后放到相应的位置去  
     for ( k = start, i = 0, j = 0; k <= end; k++)  
     {  
         if (temp1[i] <= temp2[j])  
         {  
             a[k] = temp1[i];  
             i++;  
         }  
         else  
         {  
             a[k] = temp2[j];  
             j++;  
         }  
     }  
}  
  
// 归并排序  
void MergeSort(int a[], int start, int end)  
{  
     if (start < end)  
     {  
         int i;  
         i = (end + start) / 2;  
         // 对前半部分进行排序  
         MergeSort(a, start, i);  
         // 对后半部分进行排序  
         MergeSort(a, i + 1, end);  
         // 合并前后两部分  
         Merge(a, start, i, end);  
     }  
}  
/\*顺序查找\*/  
void SequenceSearch(int \*fp,int Length)  
{  
int i;  
int data;  
printf("开始使用顺序查询.\n请输入你想要查找的数据.\n");  
scanf("%d",&data);  
for(i=0;i<Length;i++)  
   if(fp[i]==data)  
   {  
    printf("经过%d次查找,查找到数据%d,表中位置为%d.\n",i+1,data,i);  
    return ;  
   }   
printf("经过%d次查找,未能查找到数据%d.\n",i,data);  
}  
/\*二分查找\*/  
void Search(int \*fp,int Length)  
{  
int data;  
   int bottom,top,middle;  
    int i=0;  
printf("开始使用二分查询.\n请输入你想要查找的数据.\n");  
scanf("%d",&data);  
printf("由于二分查找法要求数据是有序的,现在开始为数组排序.\n");  
Sort(fp,Length);  
printf("数组现在已经是从小到大排列,下面将开始查找.\n");  
bottom=0;  
top=Length;  
while (bottom<=top)  
{  
   middle=(bottom+top)/2;  
   i++;  
   if(fp[middle]<data)  
   {  
    bottom=middle+1;  
   }  
   else if(fp[middle]>data)  
   {  
    top=middle-1;  
   }  
   else  
   {  
    printf("经过%d次查找,查找到数据%d,在排序后的表中的位置为%d.\n",i,data,middle);  
    return;  
   }  
}  
printf("经过%d次查找,未能查找到数据%d.\n",i,data);  
}  
  
void Sort(int \*fp,int Length)  
{  
int temp;  
int i,j,k;      
printf("现在开始为数组排序,排列结果将是从小到大.\n");  
for(i=0;i<Length;i++)  
   for(j=0;j<Length-i-1;j++)  
    if(fp[j]>fp[j+1])  
    {  
     temp=fp[j];  
     fp[j]=fp[j+1];  
     fp[j+1]=temp;  
    }  
    printf("排序完成!\n下面输出排序后的数组:\n");  
    for(k=0;k<Length;k++)  
    {  
     printf("%5d",fp[k]);  
    }  
    printf("\n");  
     
}  
void main()  
{   
int start=0,end=3;  
int \*p, i, a[MAX];  
int count=MAX;  
int arr[MAX];  
int choise=0;  
/\*printf("请输入你的数据的个数:\n");  
scanf("%d",&count);\*/  
/\* printf("请输入%d个数据\n",count);  
for(i=0;i<count;i++)  
{  
   scanf("%d",&arr[i]);  
}\*/  
/\*录入测试数据\*/  
      input(a);  
      printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
do   
{   
   printf("1.使用顺序查询.\n2.使用二分查找法查找.\n3.退出\n");  
   scanf("%d",&choise);  
   if(choise==1)  
    SequenceSearch(a,count);  
   else if(choise==2)  
    Search(a,count);  
   else if(choise==3)  
    break;   
} while (choise==1||choise==2||choise==3);  
  
  
/\*录入测试数据\*/  
      input(a);  
      printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
/\*测试选择排序\*/  
p = a;  
printf("选择排序之后的数据:\n");  
select\_sort(p,MAX);  
output(a);  
printf("\n");  
system("pause");  
/\*\*/  
/\*录入测试数据\*/  
input(a);  
printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
/\*测试直接插入排序\*/  
printf("直接插入排序之后的数据:\n");  
p = a;  
insert\_sort(p,MAX);  
output(a);  
printf("\n");  
system("pause");  
/\*录入测试数据\*/  
input(a);  
printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
/\*测试冒泡排序\*/  
printf("冒泡排序之后的数据:\n");  
p = a;  
insert\_sort(p,MAX);  
output(a);  
printf("\n");  
system("pause");  
/\*录入测试数据\*/  
input(a);  
printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
/\*测试快速排序\*/  
printf("快速排序之后的数据:\n");  
p = a;  
quick\_sort(p,0,MAX-1);  
output(a);  
printf("\n");  
system("pause");  
/\*录入测试数据\*/  
input(a);  
printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
/\*测试堆排序\*/  
printf("堆排序之后的数据:\n");  
p = a;  
heap\_sort(p,MAX);  
output(a);  
printf("\n");  
system("pause");  
/\*录入测试数据\*/  
input(a);  
printf("随机初始数组为：\n");  
      output(a);  
      printf("\n");  
/\*测试归并排序\*/  
printf("归并排序之后的数据:\n");  
p = a;  
MergeSort(a,start,end);  
output(a);  
printf("\n");  
system("pause");  
}