//插入排序：插入排序，希尔排序

//选择排序：简单选择排序，堆排序(二叉树的方法)

//交换排序：冒泡排序，快速排序

//基数排序，归并排序

//快速排序、堆排序或归并排序序时间复杂都为nlog2n

public class SumSortDemo{

public static void main(String args[]){

int[] arr = {10,8,1,5,10,7,9,3,6,4};

arr = quickSort(arr,0,9);

for(int i=0;i<arr.length;i++){

System.out.print(arr[i]+" ");

}

}

//冒泡排序，最简单的排序

public static void bubble(int[] a){

for(int i=1;i<a.length;i++){

for(int j=0;j<a.length;j++){

if(a[i]<a[j]){

int temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

for(int i=0;i<a.length;i++){

System.out.print(a[i]+" ");

}

}

//插入排序，是稳定的排序，时间复杂度n^2

public static void insertSort(int[] a){

for(int i=1;i<a.length;i++){

if(a[i]<a[i-1]){

int x = a[i];

int j = i-1;

a[i] = a[i-1];

while(j>=0&&x<a[j]){

a[j+1] = a[j];

j--;

}

a[j+1] = x;

}

}

for(int i=0;i<a.length;i++){

System.out.print(a[i]+" ");

}

}

//希尔排序，不稳定排序，其实就是在插入排序的基础上增加dk的概念

public static void shellSort(int[] a){

for(int dk = a.length/2;dk>=1;dk=dk/2){

for(int i= dk; i<a.length; i++){

if(a[i] < a[i-dk]){ //若第i个元素大于i-1元素，直接插入。小于的话，移动有序表后插入

int j = i-dk;

int x = a[i]; //复制为哨兵，即存储待排序元素

a[i] = a[i-dk]; //首先后移一个元素

while(j>=0&&x < a[j]){ //查找在有序表的插入位置

a[j+dk] = a[j];

j = j-dk; //元素后移

}

a[j+dk] = x; //插入到正确位置

}

}

}

for(int i=0;i<a.length;i++){

System.out.print(a[i]+" ");

}

}

//每次从后面的选择一个最小的到前面

public static void selectSort(int a[]){

for(int i=0;i<a.length;i++){ //一定要从0开始

int low = i;

for(int j=i;j<a.length;j++){ //注意这里找的是下标,不是元素

if(a[j]<a[low]){

low = j;

}

}

int temp = a[i];

a[i] = a[low];

a[low] = temp;

}

for(int i=0;i<a.length;i++){

System.out.print(a[i]+" ");

}

}

//快速排序,每次选出一个基准元素，把其他元素放在左右两边

public static int[] quickSort(int[] a,int left,int right){

if(left>=right){

return a;

}

int i = left;

int j = right;

int key = a[left];

while(i<j){

while(i<j&&key<=a[j]){

j--;

}

a[i] = a[j];

while(i<j&&key>=a[i]){

i++;

}

a[j] = a[i];

}

a[i] = key;

quickSort(a,left,i-1);

quickSort(a,i+1,right);

return a;

}

}