/\*\*  
 \* 快速排序  
 \* 挖坑填坑式  
 \*/  
 public void quickSort(int start,int end){  
 if(start<0||start>=end){  
 return ;  
 }  
 int i = start,j = end;  
 //第一个元素作为基数，并将其位置空出来  
 int baseVal = arr[start];  
 while(i<j){  
 //从后往前找小于基数的  
 while(i<j&&baseVal<=arr[j]){  
 j--;  
 }  
 //将找到的数填到上一个空位，同时j位空出  
 if(i<j){  
 arr[i] = arr[j];  
 i++;  
 }  
 //从前往后找大于基数的  
 while(i<j&&baseVal>arr[i]){  
 i++;  
 }  
 //将找到的数填到上一个空位，同时i位空出  
 if(i<j){  
 arr[j] = arr[i];  
 j--;  
 }  
 }  
 arr[i] = baseVal;  
 //整理基数前半数组,排除基数  
 quickSort(start,i-1);  
 //整理基数后半数组，排除基数  
 quickSort(j+1,end);  
 }  
  
 /\*\*  
 \* 插入排序  
 \*/  
 public void insertionSort(){  
 int j,key;  
 for(int i = 1;i<arr.length;i++){  
 //指向有序区最后一个元素  
 j = i-1;  
 //取出无序区第一个元素  
 key = arr[i];  
 //把有序区数据挨个后移，直到找到新元素的位置  
 while(j>0&&key<arr[j]){  
 arr[j+1] = arr[j];  
 j--;  
 }  
 arr[j+1] = key;  
 }  
 }  
  
 //冒泡排序  
 public void bubbleSort(){  
 for( int i = 0;i<arr.length;i++){  
 for(int j = 0;j<arr.length-i-1;j++){  
 if(arr[j]>arr[j+1]){  
 exchange(arr,j,j+1);  
 }  
 }  
 }  
 }  
  
  
 //堆排序  
 public void heapSort(){  
 //建堆  
 int i = arr.length/2-1;  
 while(i>=0){  
  
// adjustHeapRecursion(arr,i,arr.length-1);  
 adjustHeapLoop(arr,i,arr.length-1);  
 i--;  
 }  
 //排序  
 for(int j = arr.length-1;j>=0;j--){  
 exchange(arr,0,j);  
 //调整堆  
// adjustHeapRecursion(arr,0,j-1);  
 adjustHeapLoop(arr,0,j-1);  
  
 }  
  
  
 }  
  
 //循环调整堆  
 //调整树结构找到合适的位置填充arr[start]的值  
 private void adjustHeapLoop(int[] arr,int start,int end){  
  
 //保存初始节点的值  
 int temp = arr[start];  
 //从左子树开始，循环结束时指向子节点的左子树  
 for(int i = start\*2+1;i<=end;i = i\*2+1){  
 //比较左右节点，i<end保证有右节点  
 if(i<end&&arr[i]<arr[i+1]){  
 //右节点大，指向右节点  
 i++;  
 }  
 //当前子节点与初始节点比较  
 if(temp<arr[i]){  
 //子节点大，用子节点的值填充父节点  
 arr[start] = arr[i];  
 //将当前子节点作为下次循环的父节点  
 start = i;  
 }else{  
 //左右节点均小于初始节点，无需再往下调整  
 break;  
 }   
 }  
 //找到合适位置填充初始节点的值  
 arr[start] = temp;  
  
 }  
  
 //递归调整堆  
 private void adjustHeapRecursion(int[] arr,int start,int end){  
  
 int index = start\*2+1;  
 //左右子节点均存在，且左小于右  
 if(start\*2+1<end&&arr[start\*2+1]<arr[start\*2+2]){  
 //指向右节点  
 index++;  
 }  
  
 //如果子节点大于父节点，交换值，并调整子树  
 if(index<=end&&arr[start]<arr[index]){  
 exchange(arr,index,start);  
 adjustHeapRecursion(arr,index,end);  
  
 }  
  
 }  
  
 private void exchange(int[] arr,int i,int j){  
 if(i>=arr.length||j>=arr.length){  
 System.out.println("数据交换失败，数组越界");  
 return;  
 }  
 int temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }