*/\*\**

*\* 二分查找  
 \* 非递归代码**\*/*public static int biSearch(int[] array, int a) {  
 int lo = 0;  
 int hi = array.length - 1;  
 int mid;  
 while (lo <= hi) {  
 mid = (lo + hi) / 2;  
 if (array[mid] == a) {  
 return mid + 1;  
 } else if (array[mid] < a) {  
 lo = mid + 1;  
 } else {  
 hi = mid - 1;  
 }  
 }  
 return -1;  
}  
  
*/\*\**

*\* 二分查找  
 \* 递归实现**\*/*public static int sort(int[] array, int a, int lo, int hi) {  
 if (lo <= hi) {  
 int mid = (lo + hi) / 2;  
 if (a == array[mid]) {  
 return mid + 1;  
 } else if (a > array[mid]) {  
 return *sort*(array, a, mid + 1, hi);  
 } else {  
 return *sort*(array, a, lo, mid - 1);  
 }  
 }  
 return -1;  
}  
  
  
*/\*\*  
 \* 查找第一个元素出现的位置（元素允许重复）**\*/*public static int biSearchFirst(int[] array, int a) {  
 int n = array.length;  
 int low = 0;  
 int hi = n - 1;  
 int mid = 0;  
 while (low < hi) {  
 mid = (low + hi) / 2;  
 if (array[mid] < a) {  
 low = mid + 1;  
 } else {  
 hi = mid;  
 }  
 }  
 if (array[low] != a) {  
 return -1;  
 } else {  
 return low;  
 }  
}  
  
*/\*\*  
 \* 查询元素最后一次出现的位置**\*/*public static int biSearchLast(int[] array, int a) {  
 int n = array.length;  
 int low = 0;  
 int hi = n - 1;  
 int mid = 0;  
 while (low < hi) {  
 mid = (low + hi + 1) / 2;  
 if (array[mid] <= a) {  
 low = mid;  
 } else {  
 hi = mid - 1;  
 }  
 }  
  
 if (array[low] != a) {  
 return -1;  
 } else {  
 return hi;  
 }  
}

*/\*\*  
 \*冒泡排序*

*\* 稳定  
\* 平均情况/最坏情况：O(n^2)  
\* 最好情况：O(n)  
\* 空间复杂度：O(1)**\*/*private static int[] bubbleSort(int[] a) {  
 int temp = 0;  
 for (int i = a.length - 1; i > 0; --i) {  
 for (int j = 0; j < i; ++j) {  
 if (a[j + 1] < a[j]) {  
 temp = a[j];  
 a[j] = a[j + 1];  
 a[j + 1] = temp;  
 }  
 }  
 }  
 return a;  
}

*/\*\*  
 \* 构建大顶堆  
 \*/*public static void adjustHeap(int[] a, int i, int len) {  
 int temp, j;  
 temp = a[i];  
 for (j = 2 \* i; j < len; j \*= 2) {// 沿关键字较大的孩子结点向下筛选  
 if (j < len && a[j] < a[j + 1])  
 ++j; // j为关键字中较大记录的下标  
 if (temp >= a[j])  
 break;  
 a[i] = a[j];  
 i = j;  
 }  
 a[i] = temp;  
}

*/\*\*  
 \* 堆排序  
 \*/*  
public static void heapSort(int[] a) {  
 int i;  
 for (i = a.length / 2 - 1; i >= 0; i--) {// 构建一个大顶堆  
 *adjustHeap*(a, i, a.length - 1);  
 }  
 for (i = a.length - 1; i >= 0; i--) {// 将堆顶记录和当前未经排序子序列的最后一个记录交换  
 int temp = a[0];  
 a[0] = a[i];  
 a[i] = temp;  
 *adjustHeap*(a, 0, i - 1);// 将a中前i-1个记录重新调整为大顶堆  
 }  
}

*/\*\*  
 \* 快速排序  
 \* 介绍：快速排序  
 \* 不稳定  
 \* 平均情况/最好情况/空间复杂度：O(n\*log2n)  
 \* 最坏情况：O(n^2)  
 \** ***@param*** *a  
 \** ***@param*** *o  
 \** ***@param*** *e  
 \*/*private static void quickSort(int[] a, int o, int e){  
 int left = o;  
 int right = e;  
 int temp;  
 if(o<e){  
 while(left<right){  
 while(left<e&&a[left]<=a[o]){  
 left++;  
 }  
 while(right>o&&a[right]>a[o]){  
 right--;  
 }  
 if(left<right){  
 temp = a[left];  
 a[left] = a[right];  
 a[right] = temp;  
 }  
 }  
 temp = a[o];  
 a[o] = a[right];  
 a[right] = temp;  
  
 *quickSort*(a, o, right-1);  
 *quickSort*(a, right+1, e);  
 }  
}

*/\*\*  
 \* 插入排序  
 \*  
 \* 从第一个元素开始，该元素可以认为已经被排序  
 \* 取出下一个元素，在已经排序的元素序列中从后向前扫描  
 \* 如果该元素（已排序）大于新元素，将该元素移到下一位置  
 \* 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置  
 \* 将新元素插入到该位置中  
 \* 重复步骤2  
 \*/*public static void insertSort(int[] numbers)  
{  
 int size = numbers.length;  
 int temp = 0 ;  
 int j = 0;  
  
 for(int i = 0 ; i < size ; i++)  
 {  
 temp = numbers[i];  
 //假如temp比前面的值小，则将前面的值后移  
 for(j = i ; j > 0 && temp < numbers[j-1] ; j --)  
 {  
 numbers[j] = numbers[j-1];  
 }  
 numbers[j] = temp;  
 }  
}