```java
package sort;

import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

import javax.security.auth.kerberos.KerberosKey;

public class Sort {

    public static void main(String[] args) {
        int[] arr = { 12, 3, 5, 15, 9, 8, 6, 2, 7 };
        // bubbleSort(arr);
//        quickSort(arr, 0, arr.length-1);
        // selectSort1(arr);
//      insertSort1(arr);
        // mergeSort(arr, 0, arr.length - 1);
        // heapSort(arr);
//        shellSort(arr,arr.length);
         nonRecrutQuickSort(arr);
        printArr(arr);
        // System.out.println(robot(arr,0));
    }
    //冒泡排序
    public static void bubbleSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    swap(arr, j, j + 1);
                }
            }
        }
    }

    public static void mergeArray(int[] arr, int left, int mid, int
right) {
        if (arr == null || arr.length == 0)
            return;
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1;
        int k = 0;
        // 二路归并
        while (i < mid && j <= right) {
            if (arr[i] <= arr[j]) {
                temp[k++] = arr[i++];
            } else {
```

```
                temp[k++] = arr[j++];
            }
        }
        // 处理子数组中剩余元素
        while (i <= mid) {
            temp[k++] = arr[i++];
        }
        while (j <= right) {
            temp[k++] = arr[j++];
        }
        // 从临时数组中拷贝到目标数组
        for (i = 0; i < temp.length; i++) {
            arr[left + i] = temp[i];
        }
    }
    //归并排序
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            // 归并排序使得左边序列有序
            mergeSort(arr, left, mid);
            // 归并排序使得右边序列有序
            mergeSort(arr, mid + 1, right);
            // 合并两个有序序列
            mergeArray(arr, left, mid, right);
        }
    }

    //选择排序
    public static void selectSort1(int[] arr) {
        if (arr == null || arr.length == 0)
            return;
        for (int i = 0; i < arr.length - 1; i++) {// 比较n-1次
            for (int j = i + 1; j < arr.length; j++) {// 从i+1开始比较,
minIndex默认为i
                if (arr[j] < arr[i]) {
                    swap(arr, i, j);
                }
            }
        }
    }
    //快速排序
    public static void quickSort(int[] arr, int left, int right) {
        if (left >= right) {
            return;
        }
        int pivot = partition(arr, left, right);
        quickSort(arr, left, pivot - 1);
        quickSort(arr, pivot + 1, right);
```

```
94      }
95      //非递归快速排序
96      public static void nonRecrutQuickSort(int a[]) {
97          if (a == null || a.length <= 0)
98              return;
99          Stack<Integer> index = new Stack<Integer>();
100         int start = 0;
101         int end = a.length - 1;
102
103         int pivotPos;
104
105         index.push(start);
106         index.push(end);
107
108         while (!index.isEmpty()) {
109             end = index.pop();
110             start = index.pop();
111
112             pivotPos = partition(a, start, end);
113             if (start < pivotPos - 1) {
114                 index.push(start);
115                 index.push(pivotPos - 1);
116             }
117             if (end > pivotPos + 1) {
118                 index.push(pivotPos + 1);
119                 index.push(end);
120             }
121         }
122     }
123
124     public static int partition(int[] arr, int low, int high) {
125         int pivot = arr[low];
126         while (low < high) {
127             while (low < high && arr[high] >= pivot)
128                 high--;
129             arr[low] = arr[high];
130             while (low < high && arr[low] <= pivot)
131                 low++;
132             arr[high] = arr[low];
133         }
134         arr[low] = pivot;
135         return low;
136     }
137     //堆排序
138     public static void heapSort(int[] arr) {
139         int len = arr.length - 1;
140         int beginIndex = (len - 1) / 2;// 第一个非叶子节点
141         // 将数组堆化
142         for (int i = beginIndex; i >= 0; i--) {
```

```
143             maxHeapify(arr, i, len);
144         }
145         // 对堆化数组排序，每次都移出最顶层节点arr[0]，与尾部节点位置调换，同时遍历长度-1，
146         // 然后从新调整被换到根节点末尾都元素，使其符合堆堆特性，直至未排序堆堆长度未0
147         for (int j = len; j >= 0; j--) {
148             swap(arr, 0, j);
149             maxHeapify(arr, 0, j - 1);
150         }
151     }
152 
153     private static void maxHeapify(int[] arr, int index, int len) {
154         int li = (index * 2) + 1;// 左子节点索引
155         int ri = 2 * (index + 1);// 右子节点索引
156         int cMax = li;// 子节点值最大索引，默认左子节点
157         if (li > len)
158             return;// 左子节点索引超出计算范围，直接返回
159         if (ri <= len && arr[ri] > arr[li])
160             cMax = ri;// 先判断左右子节点哪个大
161         if (arr[cMax] > arr[index]) {
162             swap(arr, cMax, index);// 如果父节点被子节点调换
163             maxHeapify(arr, cMax, len);// 则需要继续判断换下后堆父节点是否符合堆堆性质
164         }
165     }
166     //希尔排序
167     public static void shellSort(int[] arr, int len) {
168         int i, j, gap;
169         for (gap = len / 2; gap > 0; gap /= 2) {
170             for (i = gap; i < len; i++) {
171                 for (j = i - gap; j >= 0 && arr[j] > arr[j + gap]; j -= gap)
172                     swap(arr, j, j + gap);
173             }
174         }
175     }
176     //插入排序
177     public static void insertSort1(int[] arr) {
178         if (arr == null || arr.length == 0)
179             return;
180         for (int i = 1; i < arr.length; i++) {// 假设第一个位置正确，要往后移，必须假设第一个
181             for (int j = i - 1; j >= 0 && arr[j] > arr[j + 1]; j--) {
182                 swap(arr, j, j+1);
183             }
184         }
185     }
186 
```

```
    public static void printArr(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }

    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

```