```java
/**
 * @author qisande
 * @date 2024-12-05 15:30:03
 * @description: 桶排序 - out-place、稳定
 *
 * 平均时间复杂度：O(n * k)
 * 空间复杂度：O(n + k)
 * k：“桶”的个数
 */
public class BucketSort {

    public static List<Integer> sort(List<Integer> array, int bucketSize) {
        if (array == null || array.size() < 2 || bucketSize <= 0) {
            return array;
        }
        int max = array.get(0), min = array.get(0);
        for (Integer element : array) {
            if (element > max) {
                max = element;
            }
            if (element < min) {
                min = element;
            }
        }
        int bucketCount = (max - min) / bucketSize + 1;
        List<List<Integer>> bucketArr = new ArrayList<>(bucketCount);
```

```
        List<Integer> resultArr = new ArrayList<>();
        for (int i = 0; i < bucketCount; i++) {
            bucketArr.add(new ArrayList<>());
        }
        for (Integer element : array) {
            bucketArr.get((element - min) / bucketSize).add(element);
        }
        for (int i = 0; i < bucketCount; i++) {
            if (bucketSize == 1) {
                for (List<Integer> bucket : bucketArr) {
                    for (Integer element : bucket) {

 resultArr.add(element);
                    }
                }
            } else {
                if (bucketCount == 1) {
                    bucketSize--;
                }
                List<Integer> temp = sort(bucketArr.get(i), bucketSize);
                for (Integer element : temp) {
                    resultArr.add(element);
                }
            }
```

```
        }
        System.out.println("Sorting: " + resultArr);
        return resultArr;
    }

    public static void main(String[] args) {
        List<Integer> array = Arrays.asList(705, 499, 161, 837, 522, 253, 988, 674, 946, 510);
        sort(array, 3);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-05 15:30:18
 * @description: 计数排序 - out-place、稳定
 * <p>
 * 平均时间复杂度：O(n + k)
 * 空间复杂度：O(k)
 * k：“桶”的个数
 * 只能对整数进行排序
 */
public class CountingSort {

    public static void sort(int[] array) {
        if (array == null || array.length < 2) {
            return;
        }
        int bias, min = array[0], max = array[0];
```

```
        for (int i = 0; i < array.length; i++) {
            if (array[i] < min) {
                min = array[i];
            }
            if (array[i] > max) {
                max = array[i];
            }
        }
        bias = 0 - min;
        int[] bucket = new int[max - min + 1];
        Arrays.fill(bucket, 0);
        for (int i = 0; i < array.length; i++) {
            bucket[array[i] + bias]++;
        }
        int index = 0, i = 0;
        while (index < array.length) {
            if (bucket[i] != 0) {
                array[index++] = i - bias;
                bucket[i]--;
            } else {
                i++;
            }
        }
        System.out.println("Sorting: " + Arrays.toString(array));
    }

    public static void main(String[] args) {
        int[] array = new int[]{705, 499, 161, 837, 522, 253, 988, 674, 946, 510};
```

```
46          sort(array);
47      }
48  }
```

```
1  /**
2   * @author qisande
3   * @date 2024-12-03 11:37:37
4   * @description: 希尔排序 - in-place、不稳定
5   *
6   * 平均时间复杂度：O(nlog2n)
7   * 空间复杂度：O(1)
8   * 是简单插入排序经过改进之后的一个更高效的版本，也称为缩小增量排序
9   */
10 public class ShellSort {
11
12     public static void sort(int[] array) {
13         if (array == null || array.length <= 0) {
14             return;
15         }
16         int temp, gap = array.length / 2;
17         while (gap > 0) {
18             for (int i = gap; i < array.length; i++) {
19                 temp = array[i];
20                 int preIndex = i - gap;
21                 while (preIndex >= 0 && temp < array[preIndex]) {
22                     array[preIndex + gap] = array[preIndex];
23                     preIndex -= gap;
24                 }
```

```
                array[preIndex + gap] = temp;
            }
            gap /= 2;
            System.out.println("Sorting: " + Arrays.toString(array));
        }
    }

    public static void main(String[] args) {
        int[] array = new int[]{8, 9, 1, 7, 2, 3, 5, 4, 6, 0};
        sort(array);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-03 11:37:10
 * @description: 直接插入排序 - 不稳定
 * 平均时间复杂度：O(n^2)
 * 空间复杂度：O(1)
 */
public class StraightInsertionSort {

    /**
     * 交换法
     *
     * @param array
     */
    public static void sort(int[] array) {
        if (array == null || array.length <= 0) {
```

```
            return;
        }
        int temp;
        for (int i = 1; i < array.length; i++) {
            temp = array[i];
            int preIndex = i - 1;
            while (preIndex >= 0 && temp < array[preIndex]) {
                array[preIndex + 1] = array[preIndex];
                preIndex--;
            }
            array[preIndex + 1] = temp;
            System.out.println("Sorting: " + Arrays.toString(array));
        }
    }

    public static void main(String[] args) {
        int[] array = new int[]{8, 9, 1, 7, 2, 3, 5, 4, 6, 0};
        sort(array);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-05 15:31:41
 * @description: 归并排序 - out-place、稳定
 *
 * 平均时间复杂度：O(nlog(n))
 * 空间复杂度：O(n)
```

```java
 */
public class MergeSort {

    public static int[] sort(int[] array) {
        if (array == null || array.length < 2) {
            return array;
        }
        int mid = array.length / 2;
        int[] left = Arrays.copyOfRange(array, 0, mid);
        int[] right = Arrays.copyOfRange(array, mid, array.length);
        return merge(sort(left), sort(right));
    }

    /**
     * 将两段排序好的数组结合成一个排序数组
     *
     * @param left
     * @param right
     * @return
     */
    public static int[] merge(int[] left, int[] right) {
        int[] result = new int[left.length + right.length];
        for (int index = 0, i = 0, j = 0; index < result.length; index++) {
            if (i >= left.length) {
                result[index] = right[j++];
```

```
            } else if (j >= right.length) {
                result[index] = left[i++];
            } else if (left[i] > right[j]) {
                result[index] = right[j++];
            } else {
                result[index] = left[i++];
            }
        }
        System.out.println("Sorting: " + Arrays.toString(result));
        return result;
    }

    public static void main(String[] args) {
        int[] array = new int[]{8, 9, 1, 7, 2, 3, 5, 4, 6, 0};
        sort(array);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-05 15:32:30
 * @description: 基数排序 - out-place、稳定
 *
 * 平均时间复杂度：O(n * k)
 * 空间复杂度：O(n + k)
 * k：“桶”的个数
 */
public class RadixSort {

    public static void sort(int[] array) {
```

```
        if (array == null || array.length < 2) {
            return;
        }
        int max = array[0];
        for (int i = 1; i < array.length; i++) {
            max = Math.max(max, array[i]);
        }
        int maxDigit = 0;
        while (max != 0) {
            max /= 10;
            maxDigit++;
        }
        int mod = 10, div = 1;
        ArrayList<ArrayList<Integer>> bucketList = new ArrayList<>();

        for (int i = 0; i < 10; i++) {
            bucketList.add(new ArrayList<>());
        }
        for (int i = 0; i < maxDigit; i++, mod *= 10, div *= 10) {
            for (int j = 0; j < array.length; j++) {
                int num = (array[j] % mod) / div;

 bucketList.get(num).add(array[j]);
            }
            int index = 0;
```

```
            for (ArrayList<Integer> arrayList : bucketList) {
                for (Integer num : arrayList) {
                    array[index++] = num;
                }
                arrayList.clear();
            }
            System.out.println("Sorting: " + Arrays.toString(array));
        }
    }

    public static void main(String[] args) {
        int[] array = new int[]{705, 499, 161, 837, 522, 253, 988, 674, 946, 510};
        sort(array);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-03 11:38:50
 * @description: 堆排序 - in-place、不稳定
 *
 * 平均时间复杂度：O(nlog(n))
 * 空间复杂度：O(1)
 */
public class Heapsort {

    private static int len;

    public static void sort(int[] array) {
```

```
14          len = array.length;
15          if (array == null || len <= 0) {
16              return;
17          }
18          buildMaxHeap(array);
19          while (len > 0) {
20              swap(array, 0, len - 1);
21              len--;
22              adjustHeap(array, 0);
23              System.out.println("Sorting: " +
Arrays.toString(array));
24          }
25      }
26
27      /**
28       * 调整使之成为最大堆
29       *
30       * @param array
31       * @param i
32       */
33      public static void adjustHeap(int[]
array, int i) {
34          int maxIndex = i;
35          if (i * 2 < len && array[i * 2] >
array[maxIndex]) {
36              maxIndex = i * 2;
37          }
38          if (i * 2 + 1 < len && array[i * 2 +
1] > array[maxIndex]) {
39              maxIndex = i * 2 + 1;
40          }
41          if (maxIndex != i) {
42              swap(array, i, maxIndex);
```

```
43              adjustHeap(array, maxIndex);
44          }
45      }
46
47      /**
48       * 建立最大堆
49       *
50       * @param array
51       */
52      public static void buildMaxHeap(int[]
array) {
53          for (int i = (len / 2 - 1); i >= 0;
i--) {
54              adjustHeap(array, i);
55          }
56      }
57
58      /**
59       * 临时变量法
60       * 交换数组 array 的 i 和 j 位置的数据
61       *
62       * @param array
63       * @param i
64       * @param j
65       */
66      public static void swap(int[] array, int
i, int j) {
67          int temp = array[i];
68          array[i] = array[j];
69          array[j] = temp;
70      }
71
72      public static void main(String[] args) {
```

```
        int[] array = new int[]{8, 9, 1, 7, 2, 3, 5, 4, 6, 0};
        sort(array);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-03 11:38:32
 * @description: 选择排序 - in-place、不稳定
 *
 * 平均时间复杂度：O(n^2)
 * 空间复杂度：O(1)
 */
public class SelectSort {

    public static void sort(int[] array) {
        if (array == null || array.length <= 0) {
            return;
        }
        for (int i = 0; i < array.length; i++) {
            int minIndex = i;
            for (int j = i; j < array.length; j++) {
                if (array[j] < array[minIndex]) {
                    minIndex = j;
                }
            }
            swap(array, i, minIndex);
```

```
            System.out.println("Sorting: " + Arrays.toString(array));
        }
    }

    /**
     * 临时变量法
     * 交换数组 array 的 i 和 j 位置的数据
     *
     * @param array
     * @param i
     * @param j
     */
    public static void swap(int[] array, int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String[] args) {
        int[] array = new int[]{8, 9, 1, 7, 2, 3, 5, 4, 6, 0};
        sort(array);
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-03 11:36:07
 * @description: 冒泡排序 - 稳定
 * 平均时间复杂度：O(n^2)
 * 空间复杂度：O(1)
```

```java
 */
public class BubbleSort {

    public static void sort(int[] array) {
        if (array == null || array.length == 0) {
            return;
        }

        int length = array.length;
        // 外层：需要作 length-1 次循环比较
        for (int i = 0; i < length - 1; i++) {
            // 内层：每次循环需要两两比较的次数，每次比较后，都会将当前最大的数放到最后位置，
            // 所以每次比较次数递减一次
            for (int j = 0; j < length - 1 - i; j++) {
                if (array[j] > array[j + 1]) {
                    // 交换数组 array 的 j 和 j+1 位置的数据
                    swapByTemp(array, j, j + 1);
                }
            }
        }
    }

    /**
     * 临时变量法
     * 交换数组 array 的 i 和 j 位置的数据
     *
```

```
33          * @param array
34          * @param i
35          * @param j
36          */
37         public static void swapByTemp(int[]
array, int i, int j) {
38             int temp = array[i];
39             array[i] = array[j];
40             array[j] = temp;
41         }
42
43         /**
44          * 算术法
45          * 交换数组 array 的 i 和 j 位置的数据
46          *
47          * @param array
48          * @param i
49          * @param j
50          */
51         public static void
swapByArithmetic(int[] array, int i, int j)
{
52             array[i] = array[i] + array[j];
53             // array[i] + array[j] - array[j] =
array[i]
54             array[j] = array[i] - array[j];
55             // array[i] + array[i] - array[i] =
array[i]
56             array[i] = array[i] - array[i];
57         }
58
59         /**
60          * 位运算法
```

```
61          * 交换数组 array 的 i 和 j 位置的数据
62          * 不好用，有问题，后面再测
63          * @param array
64          * @param i
65          * @param j
66          */
67         public static void
swapByBitOperation(int[] array, int i, int
j) {
68             array[i] = array[i] ^ array[j];
69             //
array[i]^array[j]^array[j]=array[i]
70             array[j] = array[i] ^ array[j];
71             //
array[i]^array[j]^array[j]=array[i]
72             array[i] = array[i] ^ array[j];
73         }
74
75         public static void main(String[] args) {
76             int[] array = new int[]{8, 9, 1, 7,
2, 3, 5, 4, 6, 0};
77             sort(array);
78             for (int i : array) {
79                 System.out.println(i);
80             }
81         }
82 }
```

```
1 /**
2  * @author qisande
3  * @date 2024-12-03 11:36:32
4  * @description: 快速排序 - 不稳定
5  * 平均时间复杂度：O(nlogn)
```

```
 * 空间复杂度：O(1)
 */
public class QuickSort {

    /**
     * 左右指针法
     *
     * @param array 待排序数组
     * @param low 左边界
     * @param high 右边界
     */
    public static void sort(int array[], int low, int high) {
        if (array == null || array.length <= 0) {
            return;
        }
        if (low >= high) {
            return;
        }

        int left = low;
        int right = high;
        int key = array[left];

        while (left < right) {
            while (left < right && array[right] >= key) {
                right--;
            }
            while (left < right && array[left] <= key) {
                left++;
```

```
35              }
36              if (left < right) {
37                  swap(array, left, right);
38              }
39          }
40          swap(array, low, left);
41          System.out.println("Sorting: " +
Arrays.toString(array));
42          sort(array, low, left - 1);
43          sort(array, left + 1, high);
44      }
45
46      /**
47       * 临时变量法
48       * 交换数组 array 的 i 和 j 位置的数据
49       *
50       * @param array
51       * @param i
52       * @param j
53       */
54      public static void swap(int[] array, int
i, int j) {
55          int temp = array[i];
56          array[i] = array[j];
57          array[j] = temp;
58      }
59
60      public static void main(String[] args) {
61          int[] array = new int[]{8, 9, 1, 7,
2, 3, 5, 4, 6, 0};
62          sort(array, 0, array.length - 1);
63      }
64  }
```

```java
/**
 * @author qisande
 * @date 2024-12-11 15:56:05
 * @description: 广度优先搜索，又叫层序遍历
 */
public class BreadthFirst {

    /**
     * 非递归算法
     *
     * @param root
     */
    public static void nonRecursive(TreeNode root) {
        if (root == null) {
            return;
        }
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            System.out.println(node.val + "");
            if (node.left != null) {
                queue.offer(node.left);
            }
            if (node.right != null) {
                queue.offer(node.right);
            }
        }
    }
```

```
}
```

```java
/**
 * @author qisande
 * @date 2024-12-11 16:41:18
 * @description: 广度优先搜索，又是前序遍历
 */
public class DepthFirst {

    /**
     * 非递归算法
     *
     * @param root
     */
    public static void nonRecursive(TreeNode root) {
        if (root == null) {
            return;
        }
        Deque<TreeNode> stack = new LinkedList<>();
        stack.push(root);
        while (!stack.isEmpty()) {
            TreeNode node = stack.pop();
            System.out.println(node.val + "");
            if (node.right != null) {
                stack.push(node.right);
            }
            if (node.left != null) {
                stack.push(node.left);
            }
        }
```

```
    }
}
```

```
/**
 * @author qisande
 * @date 2024-12-11 15:48:24
 * @description: 中序遍历
 */
public class InOrder {

    /**
     * 递归算法
     *
     * @param root
     */
    public static void recursive(TreeNode root) {
        if (root != null) {
            recursive(root.left);
            System.out.println(root.val + "");
            recursive(root.right);
        }
    }

    /**
     * 非递归算法
     *
     * @param root
     */
    public static void nonRecursive(TreeNode root) {
        if (root == null) {
```

```
            return;
        }
        LinkedList<TreeNode> stack = new LinkedList<>();
        TreeNode pNode = root;
        while (pNode != null || !stack.isEmpty()) {
            if (pNode != null) {
                stack.push(pNode);
                pNode = pNode.left;
            } else {
                TreeNode node = stack.pop();
                System.out.println(pNode.val + " ");
                pNode = node.right;
            }
        }
    }

}
```

```
/**
 * @author qisande
 * @date 2024-12-11 15:48:38
 * @description: 后序遍历
 */
public class PostOrder {

    /**
     * 递归算法
     *
     * @param root
     */
```

```
    public static void recursive(TreeNode root) {
        if (root != null) {
            recursive(root.left);
            recursive(root.right);
            System.out.println(root.val + "");
        }
    }

}
```

```
/**
 * @author qisande
 * @date 2024-12-11 15:48:03
 * @description: 前序遍历
 */
public class PreOrder {

    /**
     * 递归算法
     *
     * @param root
     */
    public static void recursive(TreeNode root) {
        if (root != null) {
            System.out.println(root.val + "");
            recursive(root.left);
            recursive(root.right);
        }
    }
```

```
20
21      /**
22       * 非递归算法
23       *
24       * @param root
25       */
26      public static void nonRecursive(TreeNode
root) {
27          if (root == null) {
28              return;
29          }
30          LinkedList<TreeNode> stack = new
LinkedList<>();
31          TreeNode pNode = root;
32          while (pNode != null ||
!stack.isEmpty()) {
33              if (pNode != null) {
34                  System.out.println(pNode.val
+ " ");
35                  stack.push(pNode);
36                  pNode = pNode.left;
37              } else {
38                  TreeNode node = stack.pop();
39                  pNode = node.right;
40              }
41          }
42      }
43
44  }
```

```
1  /**
2   * @author qisande
3   * @date 2024-12-11 16:03:27
```

```
 * @description: 树的节点
 */
public class TreeNode {

    Object val;
    TreeNode left;
    TreeNode right;

    public TreeNode() {}

    public TreeNode(Object value) {
        this.val = value;
    }

    public TreeNode(Object value, TreeNode left, TreeNode right) {
        this.val = value;
        this.left = left;
        this.right = right;
    }

    public Object getVal() {
        return val;
    }

    public void setVal(Object val) {
        this.val = val;
    }

    public TreeNode getLeft() {
        return left;
    }

```

```
36      public void setLeft(TreeNode left) {
37          this.left = left;
38      }
39
40      public TreeNode getRight() {
41          return right;
42      }
43
44      public void setRight(TreeNode right) {
45          this.right = right;
46      }
47  }
```