# 冒泡排序（Bubble sort）：

1. 反复对比相邻的两个元素，如果与预期的顺序（升序）不符，则换位。
2. 每一轮对比都能保证“把最大的数字移动到最右侧”。
3. 因为每轮循环可以确保把最大的数字移动到最右侧，而且最后剩余的1个数字肯定是最小的，且已经在最左侧了，没必要再进行一次循环，所以，循环轮次=数组长度-1。
4. 因为每轮都能把最大的数字移动到最右侧，所以后续的轮次就不用对这些右侧的数字进行换位了，所以，每轮的循环次数=数组长度-当前轮次。（在写代码时，当前轮次一般设为初始值0，所以，要对应减1）
5. 需要进行多轮循环，可以使用嵌套循环

   外层循环表示循环轮次

   - 初始条件：int i = 0;
   - 循环条件：i < array.length - 1;

   内层循环用于对比和换位

   - 初始条件：int j = 0;
   - 循环条件：j < array.length - i - 1;
6. 冒泡排序（升序与降序）：

```
import java.util.Arrays;

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

        for (int i = 0; i < array.length - 1; i++) { //升序
            for (int j = 0; j < array.length - i - 1; j++) {
                if (array[j] > array[j + 1]) {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }

        for (int i = 0; i < array.length - 1; i++) { //降序
            for (int j = 0; j < array.length - i - 1; j++) {
                if (array[j] < array[j + 1]) {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }

        System.out.println(Arrays.toString(array)); //输出数组
    }
}
```

7. 冒泡排序的效率：

```
import java.util.Arrays;
import java.util.Random;

public class BubbleSort {
    public static void main(String[] args) {
        Random random = new Random();
        int numberBound = 100; //numberBound表示生成0到numberBound-1之间的随机数
        int numbersCount = 10000; //生成多少个随机数
        int[] array = new int[numbersCount];
        for (int i = 0; i < array.length; i++) {
            array[i] = random.nextInt(numberBound);
        }
        System.out.println(Arrays.toString(array)); //输出数组

        int temp;
        int compareCount = 0, swapCount = 0; //对比次数，换位次数

        long startTime = System.currentTimeMillis(); //获取当前系统时间，单位毫秒

        for (int i = 0; i < array.length - 1; i++) { //升序
            for (int j = 0; j < array.length - i - 1; j++) {
                compareCount++; //对比次数
                if (array[j] > array[j + 1]) {
                    swapCount++; //换位次数
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }

        long endTime = System.currentTimeMillis();

        for (int i = 0; i < array.length - 1; i++) { //降序
            for (int j = 0; j < array.length - i - 1; j++) {
                compareCount++; //对比次数
                if (array[j] < array[j + 1]) {
                    swapCount++; //换位次数
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }

        System.out.println(Arrays.toString(array)); //输出数组
        System.out.println("对比次数：" + compareCount);
        System.out.println("换位次数：" + swapCount);
        System.out.println("耗时：" + (endTime - startTime) + "毫秒");
    }
}
```