一、数组的概述

- 数组(Array)，是多个相同类型数据按一定顺序排列的集合，并使用一个名字命名，并通过编号的方式对这些数据进行统一管理。
- 数组的常见概念
  - 数组名
  - 下标(或索引)
  - 元素
  - 数组的长度

定价	菜名
20.32	拌海蜇
43.21	龙须菜
43.22	炝冬笋
54.21	玉兰片
64.21	烧鱼头
33.87	炸豆腐
65.33	拔丝山药
78.32	杏仁酪
56.98	烧萝卜
22.65	麻婆豆腐
62.32	红扒鱼翅
71.99	鱼香肉丝
44.55	叫花鸡
39.21	酸菜鱼
29.43	茄汁鱼卷
37.24	粉蒸鸡
48.21	酱茄子
50.62	佛跳墙
71.42	西湖醋鱼
89.49	太极虾
50.26	烤方

- 数组本身是引用数据类型，而数组中的元素可以是任何数据类型，包括基本数据类型和引用数据类型。
- 创建数组对象会在内存中开辟一整块连续的空间，而数组名中引用的是这块连续空间的首地址。
- 数组的长度一旦确定，就不能修改。
- 我们可以直接通过下标(或索引)的方式调用指定位置的元素，速度很快。
- 数组的分类：
  - 按照维度：一维数组、二维数组、三维数组、…
  - 按照元素的数据类型分：基本数据类型元素的数组、引用数据类型元素的数组(即对象数组)

二、一维数组的使用

1、声明

- 一维数组的声明方式：

  type var[] 或 type[] var;

  - 例如：

```
int a[];
int[] a1;
double b[];
String[] c; //引用类型变量数组
```

- Java语言中声明数组时不能指定其长度(数组中元素的数)， 例如： int a[5]; //非法

2、初始化

●**动态初始化：**数组声明且为数组元素分配空间与赋值的操作分开进行

```
int[] arr = new int[3];
arr[0] = 3;
arr[1] = 9;
arr[2] = 8;
```

```
String names[];
names = new String[3];
names[0] = “钱学森”;
names[1] = “邓稼先”;
names[2] = “袁隆平”;
```

●**静态初始化：**在定义数组的同时就为数组元素分配空间并赋值。

```
int arr[] = new int[]{ 3, 9, 8};
或
int[] arr = {3,9,8};
```

```
String names[] = {
 “李四光”，“茅以升”，“华罗庚”
}
```

3、数组元素的引用

●定义并用运算符**new**为之分配空间后，才可以引用数组中的每个元素；

●数组元素的引用方式：数组名[数组元素下标]

- 数组元素下标可以是整型常量或整型表达式。如a[3] , b[i] , c[6*i];
- 数组元素下标从0开始；长度为n的数组合法下标取值范围: 0 —>n-1；如int a[]=new int[3]；可引用的数组元素为a[0]、a[1]、a[2]

●每个数组都有一个属性**length**指明它的长度，例如：**a.length** 指明数组**a**的长度(元素个数)

- 数组一旦初始化，其长度是不可变的

4、数组元素的默认初始化值

●数组是引用类型，它的元素相当于类的成员变量，因此数组一经分配空间，其中的每个元素也被按照成员变量同样的方式被隐式初始化。例如：

```
public class Test {
    public static void main(String argv[]){
        int a[]= new int[5];
        System.out.println(a[3]);          //a[3]的默认值为0
    }
}
```

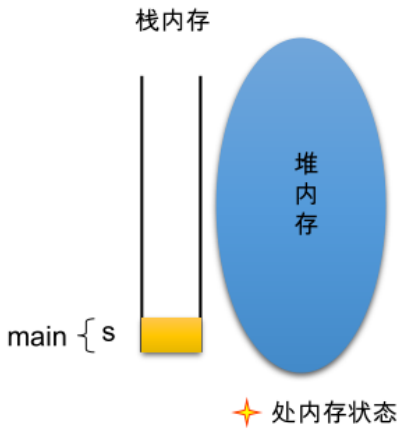
- **对于基本数据类型而言，默认初始化值各有不同**
- **对于引用数据类型而言，默认初始化值为null(注意与0不同！)**

数组元素类型	元素默认初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0
char	0 或写为:'\u0000'(表现为空)
boolean	false
引用类型	null

5、创建基本数据类型组

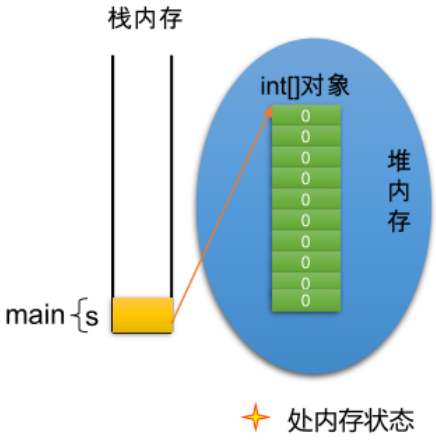
- Java中使用关键字new来创建数组
- 如下是创建基本数据类型元素的一维数组

```
public class Test{
    public static void main(String args[]){
        int[] s;
        s = new int[10];
        for ( int i=0; i<10; i++ ) {
            s[i] =2*i+1;
            System.out.println(s[i]);
        }
    }
}
```



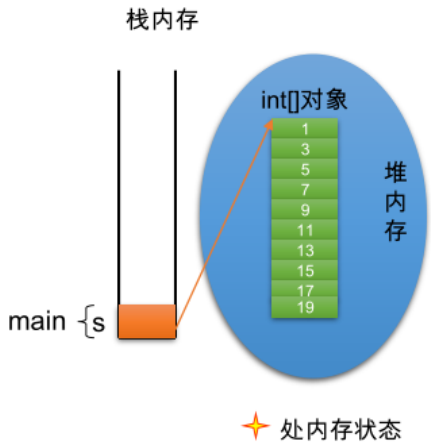
处内存状态


```
public class Test{
    public static void main(String args[]){
        int[] s;
        s = new int[10];
        //int[] s=new int[10];
        //基本数据类型数组在显式赋值之前，
        //Java会自动给他们赋默认值。
        for ( int i=0; i<10; i++ ) {
            s[i] =2*i+1;
            System.out.println(s[i]);
        }
    }
}
```

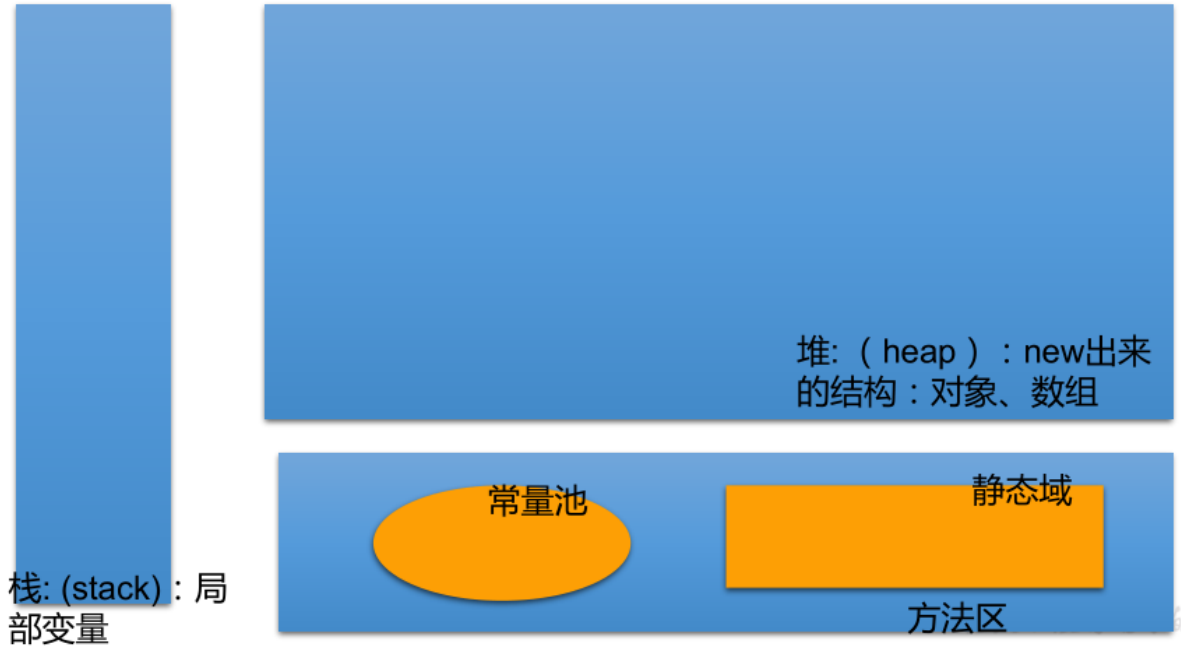


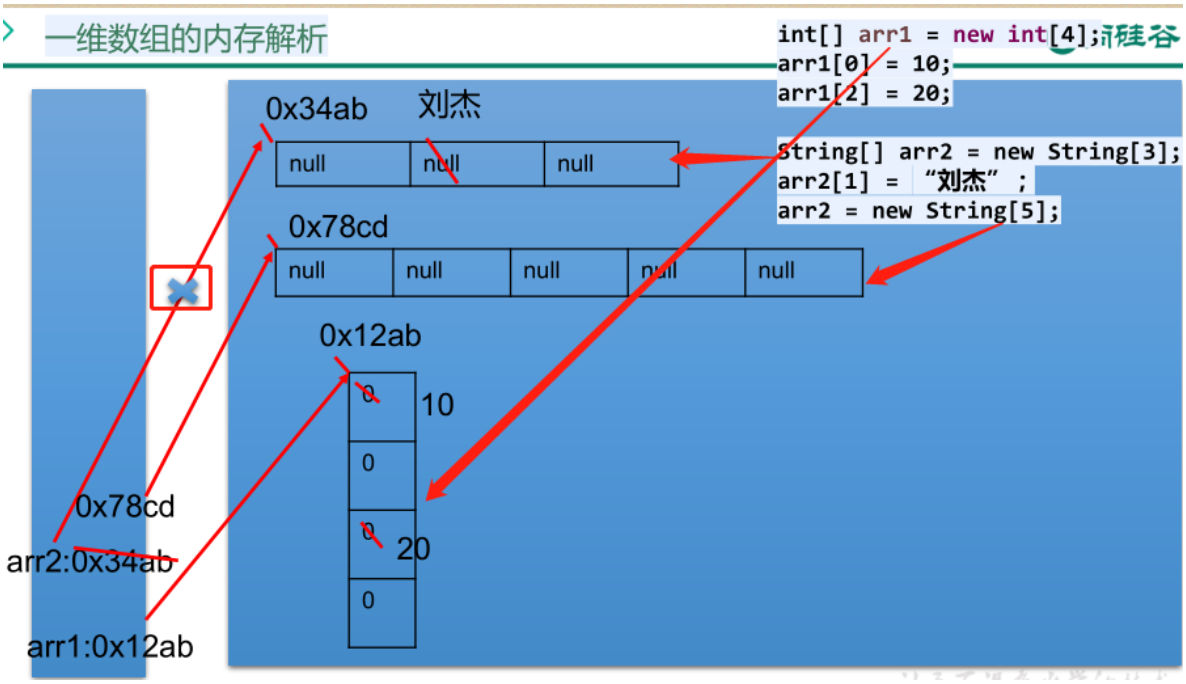
处内存状态

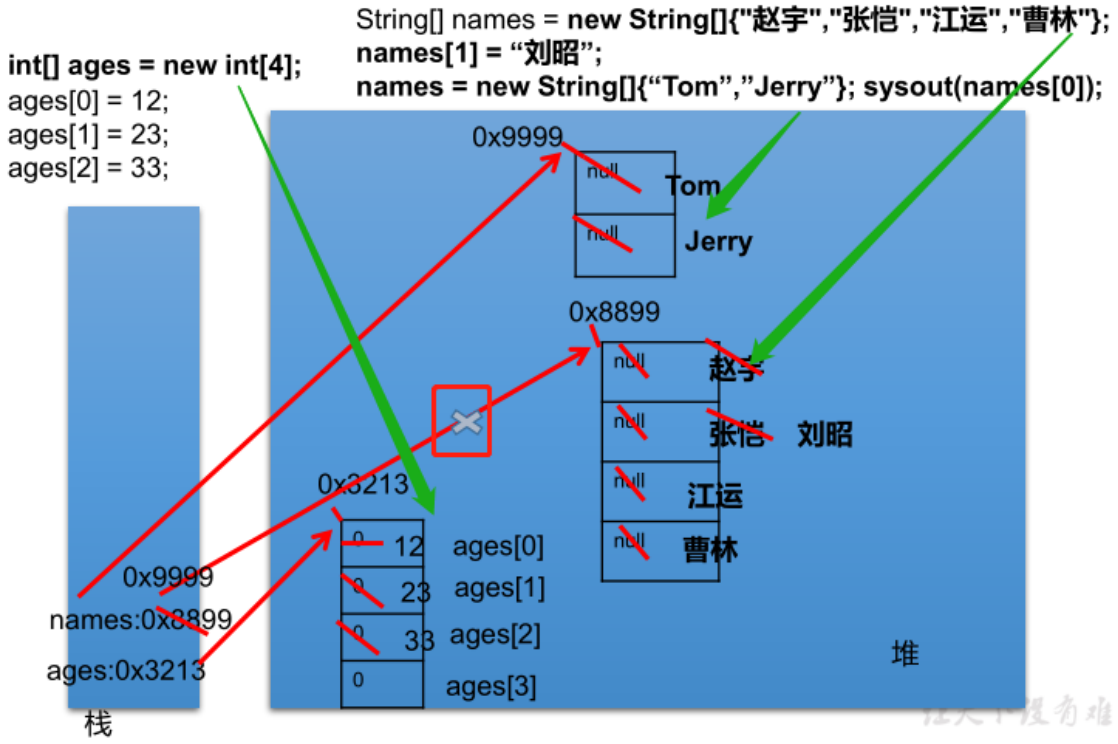
```
public class Test{
    public static void main(String args[]){
        int[] s;
        s = new int[10];
        for ( int i=0; i<10; i++ ) {
            s[i] =2*i+1;
            System.out.println(s[i]);
        }
    }
}
```



6、一维数组的内存解析







三、多维数组的使用

1、多维数组概述

- Java 语言里提供了支持多维数组的语法。
- 如果说可以把一维数组当成几何中的线性图形，那么二维数组就相当于是一个表格，像右图Excel中的表格一样。
- 对于二维数组的理解，我们可以看成是一维数组array1又作为另一个一维数组array2的元素而存在。其实，从数组底层的运行机制来看，其实没有多维数组。

姓名	联系电话
陈伟霆	13387654384
刘诗诗	13845678765
周笔畅	13012393458
鹿晗	13623490545
张艺兴	13334505689
杨幂	13623495439
宋茜	13723490897
李敏镐	13789459034
苍井空	13323409435
赵丽颖	13945893324
迪丽热巴	18623495489
周杰伦	18523489094
胡歌	13723409895
郑爽	13423409548
唐嫣	13721348984
戚薇	13909234895
王俊凯	13012390435
华晨宇	18723490438
佟丽娅	18823467128
刘亦菲	13834589533
邓紫棋	17623485739

2、多维数组初始化

二维数组[][]：数组中的数组

格式1（动态初始化）：int[][] arr = new int[3][2];

定义了名称为arr的二维数组
二维数组中有3个一维数组
每一个一维数组中有2个元素
一维数组的名称分别为arr[0], arr[1], arr[2]
给第一个一维数组1脚标位赋值为78写法是：arr[0][1] = 78;

格式2（动态初始化）：int[][] arr = new int[3][];

二维数组中有3个一维数组。
每个一维数组都是默认初始化值null (注意：区别于格式1）
可以对这个三个一维数组分别进行初始化
arr[0] = new int[3]; arr[1] = new int[1]; arr[2] = new int[2];
注：
int[][]arr = new int[][3]; //非法

格式3（静态初始化）：int[][] arr = new int[][]{{3,8,2},{2,7},{9,0,1,6}};

定义一个名称为arr的二维数组，二维数组中有三个一维数组
每一个一维数组中具体元素也都已初始化
第一个一维数组 arr[0] = {3,8,2};
第二个一维数组 arr[1] = {2,7};
第三个一维数组 arr[2] = {9,0,1,6};
第三个一维数组的长度表示方式：arr[2].length;

- 注意特殊写法情况：int[] x,y[]; x是一维数组，y是二维数组。
- Java中多维数组不必都是规则矩阵形式

```
        //1.二维数组的声明和初始化
        int[] arr = new int[]{1,2,3};//一维数组
        //静态初始化
        int[][] arr1 = new int[][]{{1,2,3},{4,5},{6,7,8}};
        //动态初始化1
        String[][] arr2 = new String[3][2];
        //动态初始化2
        String[][] arr3 = new String[3][];
        //错误的情况
//      String[][] arr4 = new String[][4];
//      String[4][3] arr5 = new String[][];
//      int[][] arr6 = new int[4][3]{{1,2,3},{4,5},{6,7,8}};

        //也是正确的写法：
        int[] arr4[] = new int[][]{{1,2,3},{4,5,9,10},{6,7,8}};
        int[] arr5[] = {{1,2,3},{4,5},{6,7,8}};
```

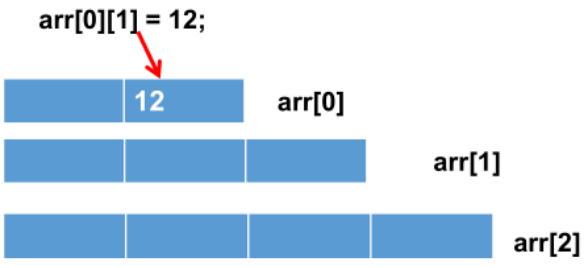
```
/*
 * 二维数组的使用：
 *  规定：二维数组分为外层数组的元素，内层数组的元素
 *      int[][] arr = new int[4][3];
 *      外层元素：arr[0],arr[1]等
 *      内层元素：arr[0][0],arr[1][2]等
 *
```

```
 *    ⑤ 数组元素的默认初始化值
 *    针对于初始化方式一：比如：int[][] arr = new int[4][3];
 *        外层元素的初始化值为：地址值
 *        内层元素的初始化值为：与一维数组初始化情况相同
 *
 *    针对于初始化方式二：比如：int[][] arr = new int[4][];
 *        外层元素的初始化值为：null
 *        内层元素的初始化值为：不能调用，否则报错。
 *
 *    ⑥ 数组的内存解析
 *
 */
public class ArrayTest3 {
    public static void main(String[] args) {

        int[][] arr = new int[4][3];
        System.out.println(arr[0]);//[I@15db9742
        System.out.println(arr[0][0]);//0

//      System.out.println(arr);//[[I@6d06d69c

        System.out.println("******************");
        float[][] arr1 = new float[4][3];
        System.out.println(arr1[0]);//地址值
        System.out.println(arr1[0][0]);//0.0

        System.out.println("******************");

        String[][] arr2 = new String[4][2];
        System.out.println(arr2[1]);//地址值
        System.out.println(arr2[1][1]);//null

        System.out.println("******************");
        double[][] arr3 = new double[4][];
        System.out.println(arr3[1]);//null
//      System.out.println(arr3[1][0]);//报错

    }
}
```

int[][] arr = new int[3][2];

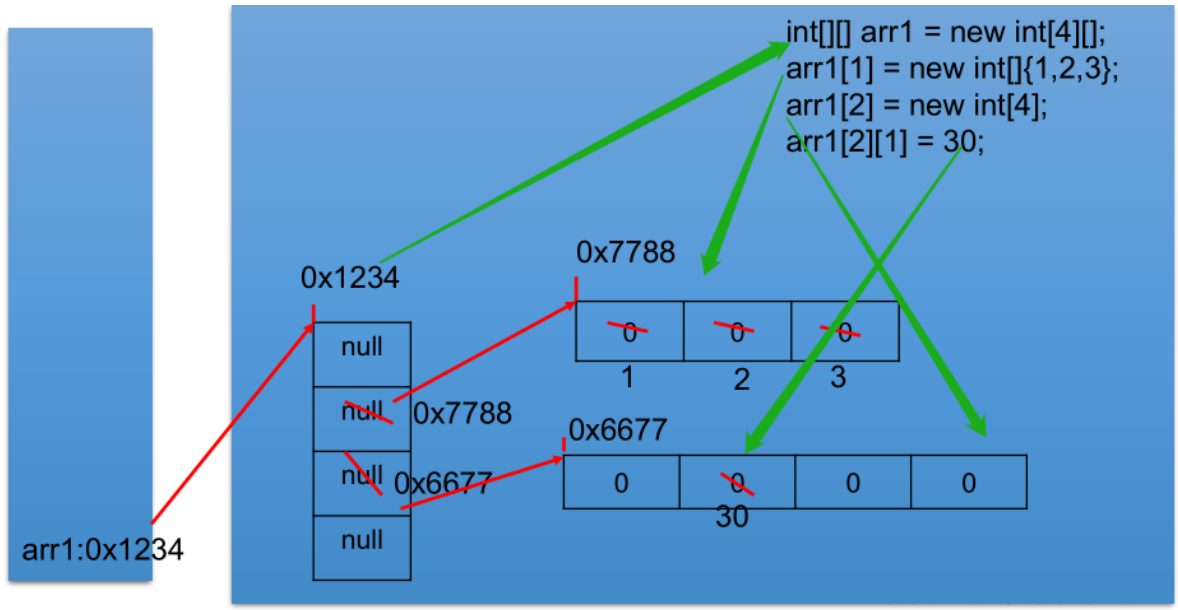
或者

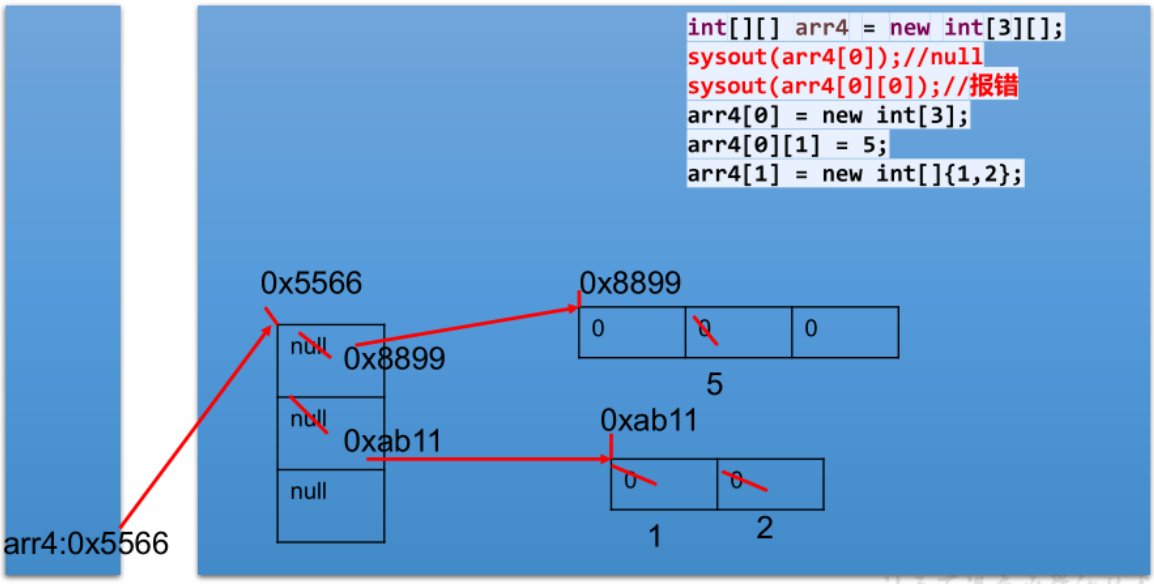
int[][] arr = new int[3][];
arr[0] = new int[2];
arr[1] = new int[2];
arr[2] = new int[2];

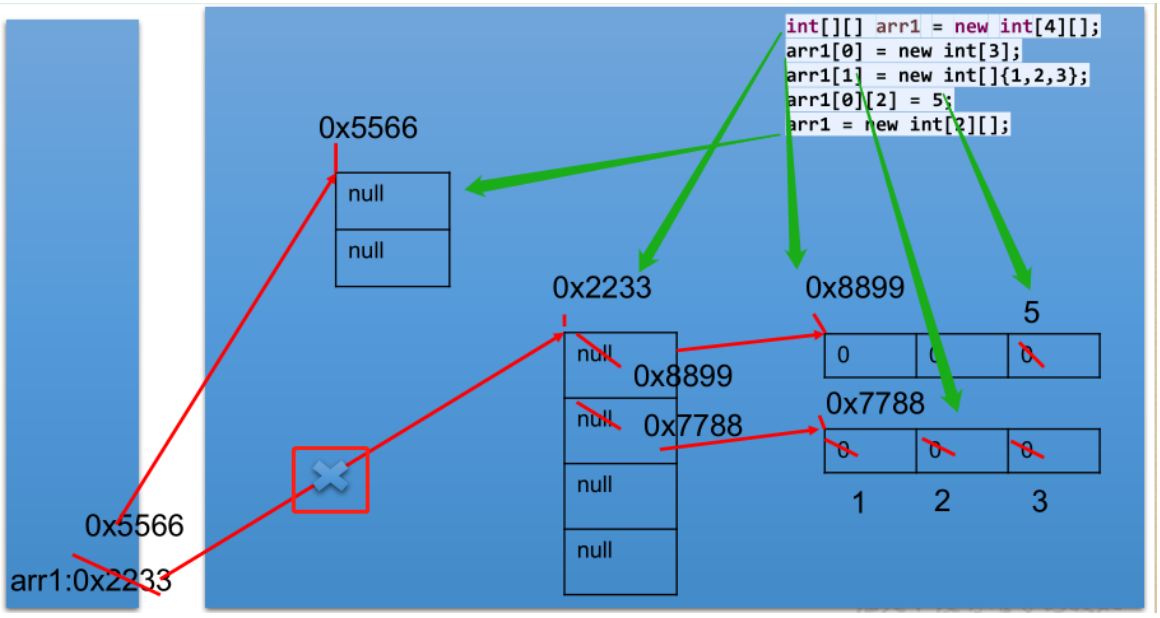


int[][] arr = new int[3][];
arr[0] = new int[2];
arr[1] = new int[3];
arr[2] = new int[4];

3、二维数组的内存解析







四、数组中涉及到的常见算法

1、数组元素的赋值(杨辉三角、回形数等)

(1) 杨辉三角

```
package com.atguigu.exer;
/*
 * 使用二维数组打印一个 10 行杨辉三角。

【提示】
 1. 第一行有 1 个元素, 第 n 行有 n 个元素
 2. 每一行的第一个元素和最后一个元素都是 1
 3. 从第三行开始, 对于非第一个元素和最后一个元素的元素。即:
yanghui[i][j] = yanghui[i-1][j-1] + yanghui[i-1][j];
 *
 */
public class YangHuiTest {

    public static void main(String[] args) {
        //1.声明并初始化二维数组
        int[][] yangHui = new int[10][];

        //2.给数组的元素赋值
        for(int i = 0;i < yangHui.length;i++){
            yangHui[i] = new int[i + 1];

            //2.1 给首末元素赋值
            yangHui[i][0] = yangHui[i][i] = 1;
            //2.2 给每行的非首末元素赋值
            //if(i > 1){
            for(int j = 1;j < yangHui[i].length - 1;j++){
                yangHui[i][j] = yangHui[i-1][j-1] + yangHui[i-1][j];
            }
            //}
        }

        //3.遍历二维数组
        for(int i = 0;i < yangHui.length;i++){
            for(int j = 0;j < yangHui[i].length;j++){
                System.out.print(yangHui[i][j] + "  ");
            }
            System.out.println();
        }

    }

}
```

(2) 回形数

```
/**
 * 从键盘输入一个整数（1~20）
 * 则以该数字为矩阵的大小，把1,2,3…n*n 的数字按照顺时针螺旋的形式填入其中。例如： 输入数字2，则程序输出： 1 2
```

```
 * 4 3
 * 输入数字3，则程序输出： 1 2 3
 * 8 9 4
 * 7 6 5
 * 输入数字4， 则程序输出：
 * 1   2   3   4
 * 12  13  14  5
 * 11  16  15  6
 * 10   9  8    7
 */

//方式一：
class RectangleTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("输入一个数字");
        int len = scanner.nextInt();
        int[][] arr = new int[len][len];

        int s = len * len;
        /*
         * k = 1:向右 k = 2:向下 k = 3:向左 k = 4:向上
         */
        int k = 1;
        int i = 0, j = 0;
        for (int m = 1; m <= s; m++) {
            if (k == 1) {
                if (j < len && arr[i][j] == 0) {
                    arr[i][j++] = m;
                } else {
                    k = 2;
                    i++;
                    j--;
                    m--;
                }
            } else if (k == 2) {
                if (i < len && arr[i][j] == 0) {
                    arr[i++][j] = m;
                } else {
                    k = 3;
                    i--;
                    j--;
                    m--;
                }
            } else if (k == 3) {
                if (j >= 0 && arr[i][j] == 0) {
                    arr[i][j--] = m;
                } else {
                    k = 4;
                    i--;
                    j++;
                    m--;
                }
            } else if (k == 4) {
                if (i >= 0 && arr[i][j] == 0) {
                    arr[i--][j] = m;
                } else {
                    k = 1;
```

```
                    i++;
                    j++;
                    m--;
                }
            }
        }

        // 遍历
        for (int m = 0; m < arr.length; m++) {
            for (int n = 0; n < arr[m].length; n++) {
                System.out.print(arr[m][n] + "\t");
            }
            System.out.println();
        }
    }
}

// 方式二：
class RectangleTest1 {

    public static void main(String[] args) {
        int n = 7;
        int[][] arr = new int[n][n];

        int count = 0; // 要显示的数据
        int maxX = n - 1; // x轴的最大下标
        int maxY = n - 1; // Y轴的最大下标
        int minX = 0; // x轴的最小下标
        int minY = 0; // Y轴的最小下标
        while (minX <= maxX) {
            for (int x = minX; x <= maxX; x++) {
                arr[minY][x] = ++count;
            }
            minY++;
            for (int y = minY; y <= maxY; y++) {
                arr[y][maxX] = ++count;
            }
            maxX--;
            for (int x = maxX; x >= minX; x--) {
                arr[maxY][x] = ++count;
            }
            maxY--;
            for (int y = maxY; y >= minY; y--) {
                arr[y][minX] = ++count;
            }
            minX++;
        }

        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr.length; j++) {
                String space = (arr[i][j] + "").length() == 1 ? "0" : "";
                System.out.print(space + arr[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

2、求数值型数组中元素的最大值、最小值、平均数、总和等

```
package com.atguigu.java;
/*
 * 算法的考查：求数值型数组中元素的最大值、最小值、平均数、总和等
 *
 * 定义一个int型的一维数组，包含10个元素，分别赋一些随机整数，
 * 然后求出所有元素的最大值，最小值，和值，平均值，并输出出来。
 * 要求：所有随机数都是两位数。
 *
 * [10,99]
 * 公式：(int)(Math.random() * (99 - 10 + 1) + 10)
 *
 */
public class ArrayTest1 {
    public static void main(String[] args) {
        int[] arr = new int[10];

        for(int i = 0;i < arr.length;i++){
            arr[i] = (int)(Math.random() * (99 - 10 + 1) + 10);
        }

        //遍历
        for(int i = 0;i < arr.length;i++){
            System.out.print(arr[i] + "\t");
        }
        System.out.println();

        //求数组元素的最大值
        int maxValue = arr[0];
        for(int i = 1;i < arr.length;i++){
            if(maxValue < arr[i]){
                maxValue = arr[i];
            }
        }
        System.out.println("最大值为：" + maxValue);

        //求数组元素的最小值
        int minValue = arr[0];
        for(int i = 1;i < arr.length;i++){
            if(minValue > arr[i]){
                minValue = arr[i];
            }
        }
        System.out.println("最小值为：" + minValue);
        //求数组元素的总和
        int sum = 0;
        for(int i = 0;i < arr.length;i++){
            sum += arr[i];
```

```
        }
        System.out.println("总和为：" + sum);
        //求数组元素的平均数
        int avgValue = sum / arr.length;
        System.out.println("平均数为：" + avgValue);
    }
}
```

3、数组的复制、反转、查找(线性查找、二分法查找)

（1）数组的复制

```
package com.atguigu.exer;
/*
 * 使用简单数组
(1)创建一个名为ArrayExer2的类，在main()方法中声明array1和array2两个变量，他们是int[]类型的数组。
(2)使用大括号{}，把array1初始化为8个素数：2,3,5,7,11,13,17,19。
(3)显示array1的内容。
(4)赋值array2变量等于array1，修改array2中的偶索引元素，使其等于索引值(如array[0]=0,array[2]=2)。打印出array1。
 *
 * 思考：array1和array2是什么关系？array1和array2地址值相同，都指向了堆空间的唯一的一个数组实体。
 * 拓展：修改题目，实现array2对array1数组的复制
 */
public class ArrayExer2 {
    public static void main(String[] args) {  //alt + /
        int[] array1,array2;

        array1 = new int[]{2,3,5,7,11,13,17,19};

        //显示array1的内容
        for(int i = 0;i < array1.length;i++){
            System.out.print(array1[i] + "\t");
        }

        //赋值array2变量等于array1
        //不能称作数组的复制。
        array2 = array1;

        //修改array2中的偶索引元素，使其等于索引值(如array[0]=0,array[2]=2)
        for(int i = 0;i < array2.length;i++){
            if(i % 2 == 0){
                array2[i] = i;
            }

        }
        System.out.println();
        //打印出array1
```

```
        for(int i = 0;i < array1.length;i++){
            System.out.print(array1[i] + "\t");
        }
    }
}
```

```
package com.atguigu.exer;
/*
 * 使用简单数组
 * 拓展：修改题目，实现array2对array1数组的复制
 */
public class ArrayExer3 {
    public static void main(String[] args) {  //alt + /
        int[] array1,array2;

        array1 = new int[]{2,3,5,7,11,13,17,19};

        //显示array1的内容
        for(int i = 0;i < array1.length;i++){
            System.out.print(array1[i] + "\t");
        }

        //数组的复制:
        array2 = new int[array1.length];
        for(int i = 0;i < array2.length;i++){
            array2[i] = array1[i];
        }


        //修改array2中的偶索引元素，使其等于索引值(如array[0]=0,array[2]=2)
        for(int i = 0;i < array2.length;i++){
            if(i % 2 == 0){
                array2[i] = i;
            }

        }
        System.out.println();
        //打印出array1
        for(int i = 0;i < array1.length;i++){
            System.out.print(array1[i] + "\t");
        }
    }
}
```

(2) 数组的反转

```
        String[] arr = new String[]{"JJ","DD","MM","BB","GG","AA"};

        //数组的反转
```

```
//方法一：
for(int i = 0;i < arr.length / 2;i++){
    String temp = arr[i];
    arr[i] = arr[arr.length - i -1];
    arr[arr.length - i -1] = temp;
}

//方法二：
for(int i = 0,j = arr.length - 1;i < j;i++,j--){
    String temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}
```

(3) 线性查找

```
String[] arr = new String[]{"JJ","DD","MM","BB","GG","AA"};
//查找（或搜索）
//线性查找：
String dest = "BB";
dest = "CC";

boolean isFlag = true;

for(int i = 0;i < arr.length;i++){

    if(dest.equals(arr[i])){
        System.out.println("找到了指定的元素，位置为：" + i);
        isFlag = false;
        break;
    }

}
if(isFlag){
    System.out.println("很遗憾，没有找到的啦！");

}
```

(4) 二分法查找

```
//二分法查找：(熟悉)
//前提：所要查找的数组必须有序。
int[] arr2 = new int[]{-98,-34,2,34,54,66,79,105,210,333};

int dest1 = -34;
dest1 = 35;
int head = 0;//初始的首索引
int end = arr2.length - 1;//初始的末索引
```

```
        boolean isFlag1 = true;
        while(head <= end){

            int middle = (head + end)/2;

            if(dest1 == arr2[middle]){
                System.out.println("找到了指定的元素，位置为：" + middle);
                isFlag1 = false;
                break;
            }else if(arr2[middle] > dest1){
                end = middle - 1;
            }else{//arr2[middle] < dest1
                head = middle + 1;
            }


        }

        if(isFlag1){
            System.out.println("很遗憾，没有找到的啦！");
        }
```

4、数组元素的排序算法

(1) 排序算法概述

排序：假设含有n个记录的序列为{R1，R2，...,Rn},其相应的关键字序列为{K1，K2，...,Kn}。将这些记录重新排序为{Ri1,Ri2,...,Rin},使得相应的关键字值满足条Ki1<=Ki2<=...<=Kin,这样的一种操作称为排序。

- 通常来说，排序的目的是快速查找。

衡量排序算法的优劣：

1.时间复杂度：分析关键字的比较次数和记录的移动次数

2.空间复杂度：分析排序算法中需要多少辅助内存

3.稳定性：若两个记录A和B的关键字值相等，但排序后A、B的先后次序保持不变，则称这种排序算法是稳定的。

排序算法分类：内部排序和外部排序。

- 内部排序：整个排序过程不需要借助于外部存储器（如磁盘等），所有排序操作都在内存中完成。
- 外部排序：参与排序的数据非常多，数据量非常大，计算机无法把整个排序过程放在内存中完成，必须借助于外部存储器（如磁盘）。外部排序最常见的是多路归并排序。可以认为外部排序是由多次内部排序组成。

十大内部排序算法

- 选择排序
  - 直接选择排序、堆排序
- 交换排序
  - 冒泡排序、快速排序
- 插入排序
  - 直接插入排序、折半插入排序、Shell排序
- 归并排序
- 桶式排序
- 基数排序

详细操作，见《附录：尚硅谷_宋红康_排序算法.pdf》

算法的5大特征

输入（Input）	有0个或多个输入数据，这些输入必须有清楚的描述和定义
输出（Output）	至少有1个或多个输出结果，不可以没有输出结果
有穷性（有限性，Finiteness）	算法在有限的步骤之后会自动结束而不会无限循环，并且每一个步骤可以在可接受的时间内完成
确定性（明确性，Definiteness）	算法中的每一步都有确定的含义，不会出现二义性
可行性（有效性，Effectiveness）	算法的每一步都是清楚且可行的，能让用户用纸笔计算而求出答案

说明：满足确定性的算法也称为：确定性算法。现在人们也关注更广泛的概念，例如考虑各种非确定性的算法，如并行算法、概率算法等。另外，人们也关注并不要求终止的计算描述，这种描述有时被称为过程（procedure）。

(2) 冒泡排序

介绍：

冒泡排序的原理非常简单，它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。

排序思想：

1. 比较相邻的元素。如果第一个比第二个大（升序），就交换他们两个。
2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。
3. 针对所有的元素重复以上的步骤，除了最后一个。
4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较为止。

```
package com.atguigu.java;
/*
 * 数组的冒泡排序的实现
 *
 */
public class BubbleSortTest {
```

```
    public static void main(String[] args) {

        int[] arr = new int[]{43,32,76,-98,0,64,33,-21,32,99};

        //冒泡排序
        for(int i = 0;i < arr.length - 1;i++){

            for(int j = 0;j < arr.length - 1 - i;j++){

                if(arr[j] > arr[j + 1]){
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }

            }

        }

        for(int i = 0;i < arr.length;i++){
            System.out.print(arr[i] + "\t");
        }

    }
}
```

(3) 快速排序

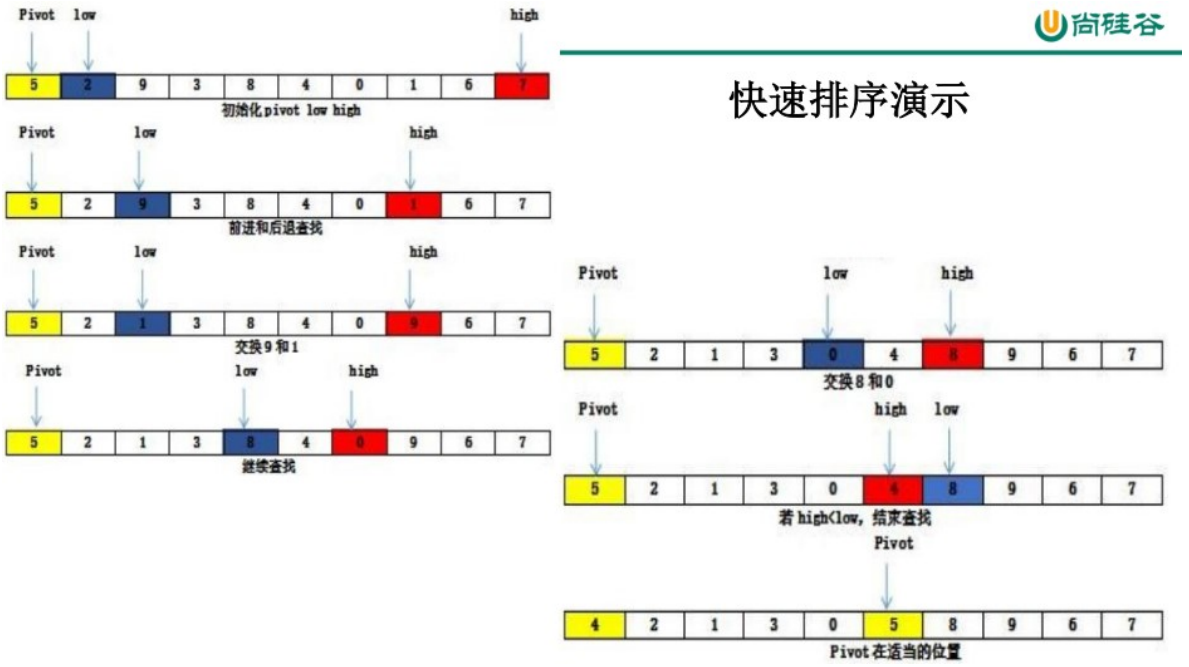
介绍：

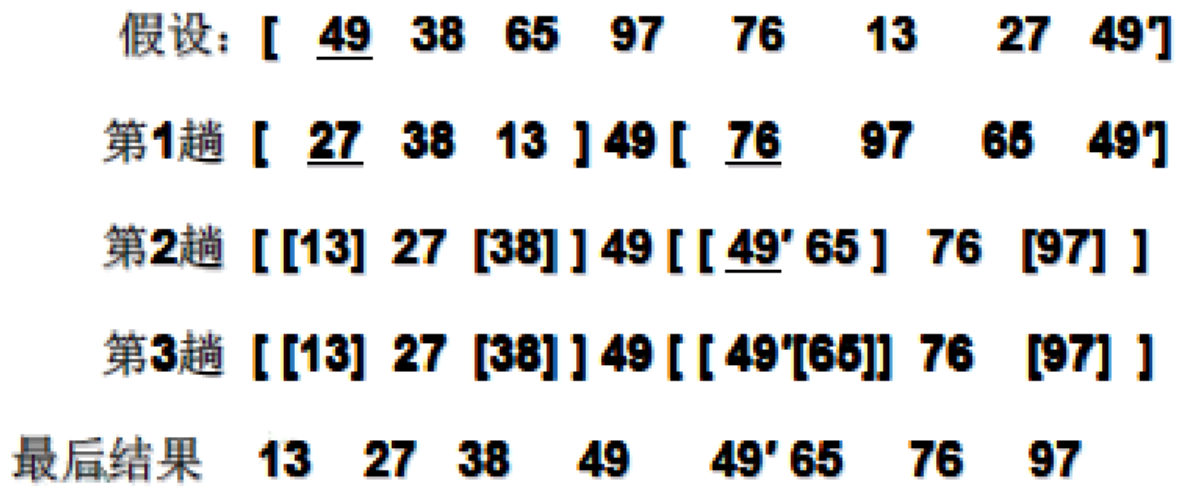
快速排序通常明显比同为O(nlogn)的其他算法更快，因此常被采用，而且快排采用了分治法的思想，所以在很多笔试面试中能经常看到快排的影子。可见掌握快排的重要性。

快速排序（Quick Sort）由图灵奖获得者Tony Hoare发明，被列为20世纪十大算法之一，是迄今为止所有内排序算法中速度最快的一种。冒泡排序的升级版，交换排序的一种。快速排序的时间复杂度为O(nlog(n))。

排序思想：

1. 从数列中挑出一个元素，称为"基准"（pivot），
2. 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区结束之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。
3. 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。
4. 递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会结束，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去。






```java
package com.atguigu.java;

/**
 * 快速排序
 * 通过一趟排序将待排序记录分割成独立的两部分，其中一部分记录的关键字均比另一部分关键字小，
 * 则分别对这两部分继续进行排序，直到整个序列有序。
 * @author shkstart
 * 2018-12-17
 */
public class QuickSort {
    private static void swap(int[] data, int i, int j) {
        int temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    }

    private static void subSort(int[] data, int start, int end) {
        if (start < end) {
            int base = data[start];
            int low = start;
            int high = end + 1;
            while (true) {
```

```
                while (low < end && data[++low] - base <= 0)
                    ;
                while (high > start && data[--high] - base >= 0)
                    ;
                if (low < high) {
                    swap(data, low, high);
                } else {
                    break;
                }
            }
            swap(data, start, high);

            subSort(data, start, high - 1);//递归调用
            subSort(data, high + 1, end);
        }
    }
    public static void quickSort(int[] data){
        subSort(data,0,data.length-1);
    }


    public static void main(String[] args) {
        int[] data = { 9, -16, 30, 23, -30, -49, 25, 21, 30 };
        System.out.println("排序之前：\n" + java.util.Arrays.toString(data));
        quickSort(data);
        System.out.println("排序之后：\n" + java.util.Arrays.toString(data));
    }
}
```

(4) 排序算法性能对比及选择

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

各种内部排序方法性能比较

1.**从平均时间而言**：快速排序最佳。但在最坏情况下时间性能不如堆排序和归并排序。

2.**从算法简单性看**：由于直接选择排序、直接插入排序和冒泡排序的算法比较简单，将其认为是简单算法。对于Shell排序、堆排序、快速排序和归并排序算法，其算法比较复杂，认为是复杂排序。

3.**从稳定性看**：直接插入排序、冒泡排序和归并排序时稳定的；而直接选择排序、快速排序、 Shell排序和堆排序是不稳定排序

4.**从待排序的记录数n的大小看**，n较小时，宜采用简单排序；而n较大时宜采用改进排序。

排序算法的选择

(1)若n较小(如n≤50)，可采用**直接插入**或**直接选择排序**。

当记录规模较小时，直接插入排序较好；否则因为直接选择移动的记录数少于直接插入，应选直接选择排序为宜。

(2)若文件初始状态基本有序(指正序)，则应选用**直接插入**、**冒泡**或随机的**快速排序**为宜；

(3)若n较大，则应采用时间复杂度为O(nlgn)的排序方法：**快速排序**、**堆排序**或**归并排序**。

(5) 排序算法扩展

为知笔记地址：排序算法

GitHub地址：
https://github.com/wangliu1102/StudyNotes/tree/master/%E5%B0%9A%E7%A1%85%E8%B0%B7Java/%E4%B8%80%E3%80%81%E5%9F%BA%E7%A1%80%E9%98%B6%E6%AE%B5/1%E3%80%81Java%E5%9F%BA%E7%A1%80_2019%E7%89%88/%E7%AC%AC1%E9%83%A8%E5%88%86%EF%BC%9AJava%E5%9F%BA%E7%A1%80%E7%BC%96%E7%A8%8B/%E7%AC%AC3%E7%AB%A0_%E6%95%B0%E7%BB%84

五、Arrays工具类的使用

java.util.Arrays类即为操作数组的工具类，包含了用来操作数组（比如排序和搜索）的各种方法。

1	**boolean equals(int[] a,int[] b)**	判断两个数组是否相等。
2	**String toString(int[] a)**	输出数组信息。
3	**void fill(int[] a,int val)**	将指定值填充到数组之中。
4	**void sort(int[] a)**	对数组进行排序。
5	**int binarySearch(int[] a,int key)**	对排序后的数组进行二分法检索指定的值。

数组排序

- java.util.Arrays类的sort()方法提供了数组元素排序功能：

```
import java.util.Arrays;
public class SortTest {
	public static void main(String[] args) {
		int [] numbers = {5,900,1,5,77,30,64,700};
		Arrays.sort(numbers);
		for(int i = 0; i < numbers.length; i++){
			System.out.println(numbers[i]);
		}
	}
}
```

```
package com.atguigu.java;

import java.util.Arrays;

/*
 * java.util.Arrays:操作数组的工具类，里面定义了很多操作数组的方法
 *
 *
 */
public class ArraysTest {
    public static void main(String[] args) {
```

```
        //1.boolean equals(int[] a,int[] b):判断两个数组是否相等。
        int[] arr1 = new int[]{1,2,3,4};
        int[] arr2 = new int[]{1,3,2,4};
        boolean isEquals = Arrays.equals(arr1, arr2);
        System.out.println(isEquals);

        //2.String toString(int[] a):输出数组信息。
        System.out.println(Arrays.toString(arr1));


        //3.void fill(int[] a,int val):将指定值填充到数组之中。
        Arrays.fill(arr1,10);
        System.out.println(Arrays.toString(arr1));


        //4.void sort(int[] a):对数组进行排序。
        Arrays.sort(arr2);
        System.out.println(Arrays.toString(arr2));

        //5.int binarySearch(int[] a,int key)
        int[] arr3 = new int[]{-98,-34,2,34,54,66,79,105,210,333};
        int index = Arrays.binarySearch(arr3, 210);
        if(index >= 0){
            System.out.println(index);
        }else{
            System.out.println("未找到");
        }


    }
}

false
[1, 2, 3, 4]
[10, 10, 10, 10]
[1, 2, 3, 4]
8
```

六、数组使用中的常见异常

编译时，不报错！！

数组脚标越界异常(ArrayIndexOutOfBoundsException)
int[] arr = new int[2]; System.out.println(arr[2]); System.out.println(arr[-1]); 访问到了数组中的不存在的脚标时发生。
空指针异常(NullPointerException)
int[] arr = null; System.out.println(arr[0]); arr引用没有指向实体，却在操作实体中的元素时。

七、每日练习

1、写出一维数组初始化的两种方式?

int[] arr = new int[5];//动态初始化

String[] arr1 = new String[]{"Tom","Jerry","Jim"};//静态初始化

数组一旦初始化，其长度就是确定的。arr.length

数组长度一旦确定，就不可修改。

2、写出二维数组初始化的两种方式?

int[][] arr = new int[4][3];//动态初始化1

int[][] arr1 = new int[4][];//动态初始化2

int[][] arr2 = new int[][]{{1,2,3},{4,5,6},{7,8}};//静态初始化

3、不同类型的一维数组元素的默认初始化值各是多少?

数组元素类型	元素默认初始值
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0
char	0 或写为:'\u0000'(表现为空)
boolean	false
引用类型	null

4、使用冒泡排序，实现如下的数组从小到大排序?

快排时间复杂度：O(nlogn)

冒泡时间复杂度：O(n^2)

```
 int[] arr = new int[]{34,5,22,-98,6,-76,0,-3};

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

5、java能动态分配数组吗?

可以。

```
int i = 12;
int[] myInt = new int[i];
```

6、数组有没有length()这个方法? String有没有length()这个方法?

数组没有length()这个方法，有length的属性。String有length()这个方法。

7、Java中的任何数据类型都可以使用System.out.pritln方法显示?

对基本数据类型而言，输出的往往是变量的值;

对于像数组这一类复杂的数据类型，输出的是其堆空间中存储位置的hashCode值 。

8、操作二维数组的注意点?

操作二维数组不应使用常数来控制维数。具体方法是array.length表示行数，array[row].length来表示row行的列数。这样当数组行数和列数不相等时，代码可以自动调整为正确的值。