如何理解并掌握 Java 数据结构

(https://gitbook.cn/gitchat/author/59ee9f5e1b0bc73189b0cd68)

张振华 (https://gitbook....

向作者提问 (https://gitbook.cn/m/mazi/author/59ee9f5e1b0bc73189b0cd68/question)

曾经先后在驴妈妈，携程，要买车公司担任过Java高级工程师、架构师、开发主管、技术经理等职务。在电商公司工作期间，负责过PC站和后端服务的平台架构、实现和升级。 目前在做一些Java架构工作。前后从业10几年没有离开Java，2015年出版《Java并发编程从入门到精通》。2018年出版《Spring Data Jpa从入门到精通》。 网名：张振华.Jack

查看本场Chat

(https://gitbook.cn/gitchat/activity/5a391b122edf834ef46c6296)

Jack和大家一起来重温《Java数据结构》经典之作。

第一部分：Java数据结构

要理解Java数据结构，必须能清楚何为数据结构？

> 数据结构:
>
> 1. Data_Structure，它是储存数据的一种结构体，在此结构中储存一些数据，而这些数据之间有一定的关系。
> 2. 而各数据元素之间的相互关系，又包括三个组成成分，数据的逻辑结构，数据的存储结构和数据运算结构。
> 3. 而一个数据结构的设计过程分成抽象层、数据结构层和实现层。

数据结构在Java的语言体系中按逻辑结构可以分为两大类：线性数据结构和非线性数据结构。

一、Java数据结构之：线性数据结构

> 线性数据结构：常见的有一维数组，线性表，栈，队列，双队列，串。

1：一维数组

在Java里面常用的util有：String [],int [],ArrayList,Vector,CopyOnWriteArrayList等。及可以同过一维数组[]自己实现不同逻辑结构的Util类。而ArrayList封装了一些[]的基本操作方法。ArrayList和Vector的区别是:Vector是线程安全的，方法同步。CopyOnWriteArrayList也是线程安全的但效率要比Vector高很多。(PS:如果不懂出门右拐看另一篇chat)。

数组这种数据结构典型的操作方法，是根据下标进行操作的，所以insert的的时候可以根据下标插入到具体的某个位置，但是这个时候它后面的元素都得往后面移动一位。所以插入效率比较低,更新，删除效率也比较低，而查询效率非常高,查询效率时间复杂度是1。

2： 线性表

线性表是有序的储存结构、链式的储存结构。链表的物理储存空间是不连续的，链表的每一个节点都知道上一个节点、或者下一个节点是谁，通常用Node表示。常见的有顺序链表(LinkedList、Linked***)，单项链表（里面只有Node类），双向链表(两个Node类)，循环链表(多个Node类)等。

操作方法：插入效率比较高，插入的时候只需要改变节点的前后节点的连接即可。而查询效率就比较低了，如果实现的不好，需要整个链路找下去才能找到应该找的元素。所以常见的方法有：add(index,element),addFirst(element),addLast(element)。getFirst(),getLast(),get(element)等。

常见的Uitil有：LinkedList，LinkedMap等，而这两个JDK底层也做了N多优化，可以有效避免查询效率低的问题。当自己实现的时候需要注意。其实树形结构可以说是非线性的链式储存结构。

3: 栈Stack

栈,最主要的是要实现先进后出，后进先出的逻辑结构。来实现一些场景对逻辑顺序的要求。所以常用的方法有push(element)压栈，pop()出栈。

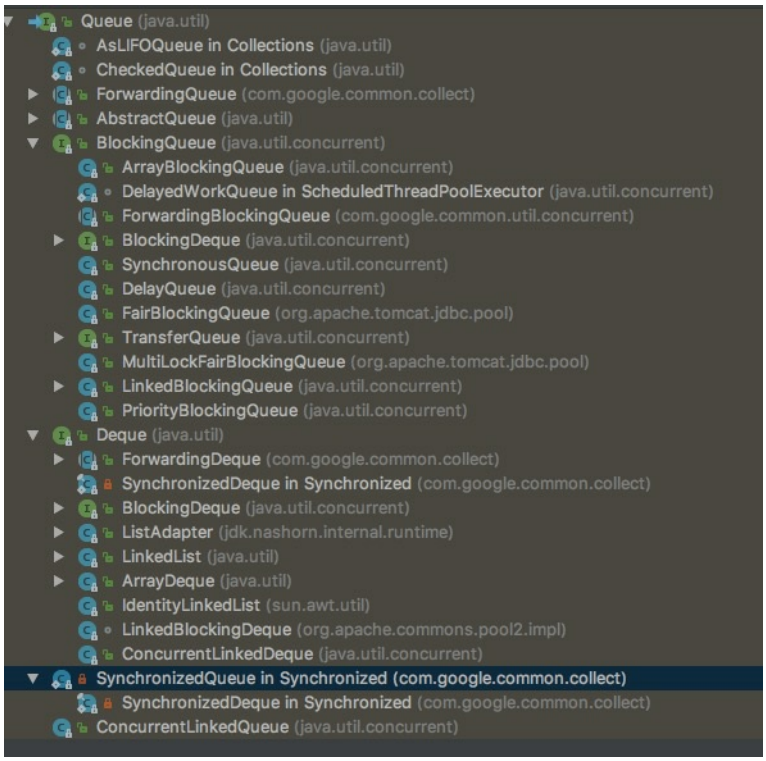
java.util.Stack。就实现了这用逻辑。而Java的Jvm里面也用的到了此种数据结构，就是线程栈，来保证当前线程的执行顺序。

4：队列

队列，队列是一种特殊的线性数据结构，队列只能允许在队头，队尾进行添加和查询等相关操作。队列又有单项有序队列，双向队列，阻塞队列等。

Queue这种数据结构注定了基本操作方法有：add(E e)加入队列，remove(),poll()等方法。

队列在Java语言环境中是使用频率相当高的数据结构，所有其实现的类也很多来满足不同场景。



> 使用场景也非常多，如线程池，mq，连接池等。

5：串

串：也称字符串，是由N个字符组成的优先序列。在Java里面就是指String,而String里面是由chat[]来进行储存。

> KMP算法： 这个算法一定要牢记，Java数据结构这本书里面针对字符串的查找匹配算法也只介绍了一种。关键点就是：在字符串比对的时候，主串的比较位置不需要回退的问题。

二、Java数据结构之：非线性数据结构

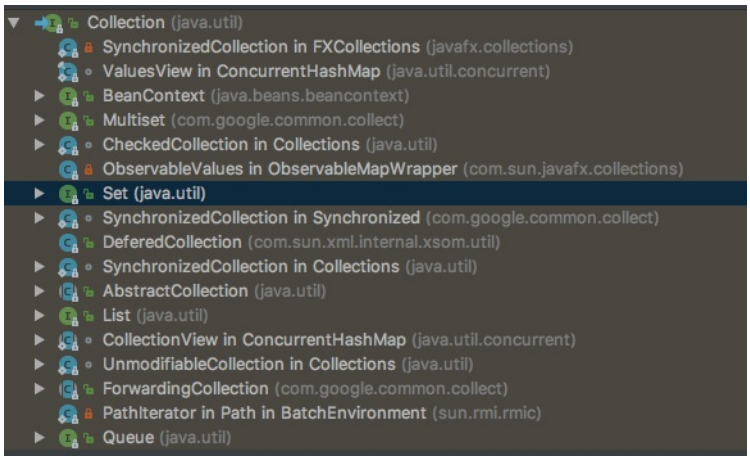
> 非线性数据结构：常见的有：多维数组，集合，树，图，散列表(hash).

1：多维数组

一维数组前面咱也提到了，多维数组无非就是String [][],int[][]等。Java里面很少提供这样的工具类，而java里面tree和图底层的native方法用了多维数组来储存。

2：集合

由一个或多个确定的元素所构成的整体叫做集合。在Java里面可以去广义的去理解为实现了Collection接口的类都叫集合。



3：树

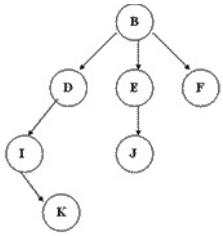
> 树形结构，作者觉得它是一种特殊的链形数据结构。最少有一个根节点组成，可以有多个子节点。树，显然是由递归算法组成。

树的特点：

1. 在一个树结构中，有且仅有一个结点没有直接父节点，它就是根节点。
2. 除了根节点，其他结点有且只有一个直接父节点
3. 每个结点可以有任意多个直接子节点。

树的数据结构又分如下几种：

- 1) 自由树/普通树：对子节点没有任何约束。

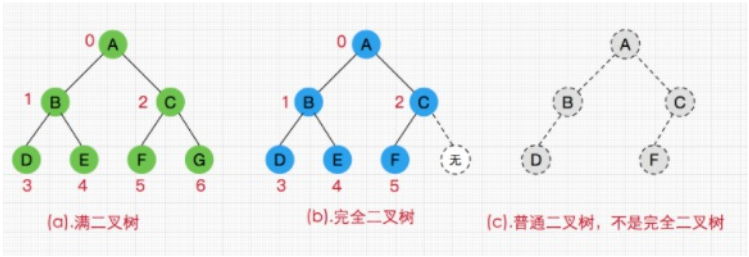


- 2) 二叉树：每个节点最多含有两个子节点的树称为二叉树。

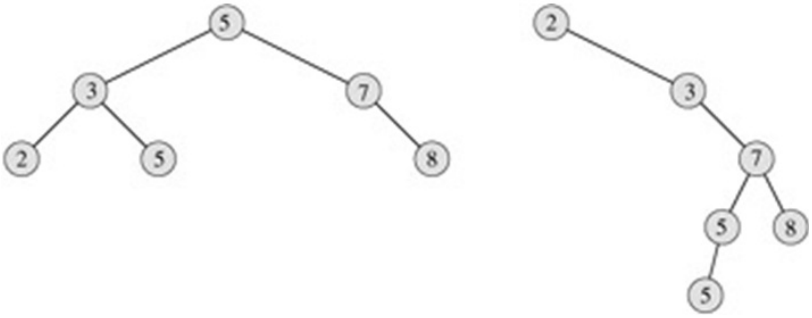
  2.1) 一般二叉树：每个子节点的父亲节点不一定有两个子节点的二叉树成为一般二叉树。

  2.2) 完全二叉树：对于一颗二叉树，假设其深度为d（d>1）。除了第d层外，其它各层的节点数目均已达最大值，且第d层所有节点从左向右连续地紧密排列，这样的二叉树被称为完全二叉树；

  2.3) 满二叉树：所有的节点都是二叉的二叉树成为满二叉树。



- 3) 二叉搜索树/BST：binary search tree,又称二叉排序树、二叉查找树。是有序的。要点：如果不为空，那么其左子树节点的值都小于根节点的值；右子树节点的值都大于根节点的值。



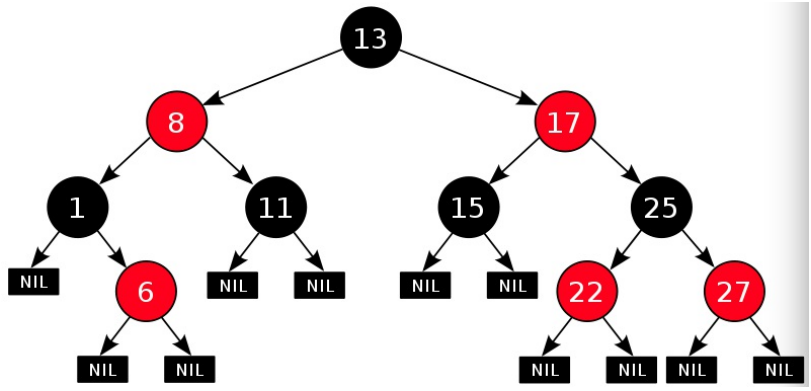
3.1) 二叉平衡树：二叉搜索树，是有序的排序树，但左右两边包括子节点不一定平衡，而二叉平衡树是排序树的一种，并且加点条件，就是任意一个节点的两个叉的深度差不多（比如差值的绝对值小于某个常数，或者一个不能比另一个深出去一倍之类的）。这样的树可以保证二分搜索任意元素都是O(log n)的，一般还附带带有插入或者删除某个元素也是O(log n)的的性质。

为了实现，二叉平衡树又延伸出来了一些算法，业界常见的有AVL、和红黑算法，所以又有以下两种树：

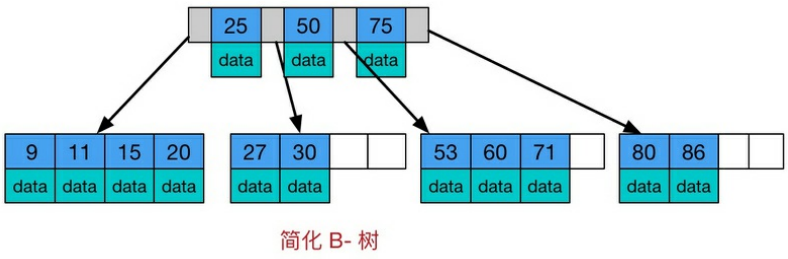
3.1.1) AVL树：最早的平衡二叉树之一。应用相对其他数据结构比较少。windows对进程地址空间的管理用到了AVL树。

3.1.2) 红黑树：通过制定了一些红黑标记和左右旋转规则来保证二叉树平衡。

> 红黑树的5条性质：
>
> 1. 每个结点要么是红的，要么是黑的。
> 2. 根结点是黑的。
> 3. 每个叶结点（叶结点即指树尾端NIL指针或NULL结点）是黑的。
> 4. 如果一个结点是红的，那么它的俩个儿子都是黑的。
> 5. 对于任一结点而言，其到叶结点树尾端NIL指针的每一条路径都包含相同数目的黑结点。

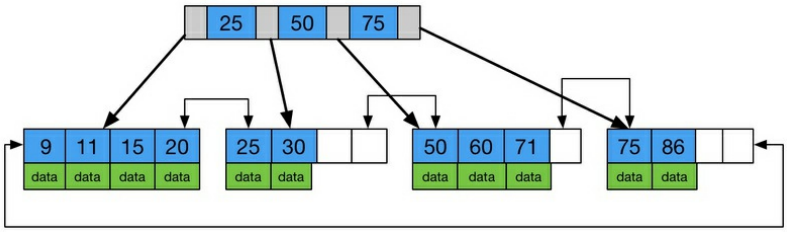


- 4) B-tree：又称B树、B-树。又叫平衡(balance)多路查找树。树中每个结点最多含有m个孩子（m>=2）。它类似普通的平衡二叉树，不同的一点是B-树允许每个节点有更多的子节点。



简化 B- 树

- 4) B+tree：又称B+。是B-树的变体，也是一种多路搜索树。



简化 B+ 树

> 树总结：
> 树在Java里面应用的也比较多。非排序树，主要用来做数据储存和展示。而排序树，主要用来做算法和运算，HashMap里面的TreeNode就用到了红黑树算法。而B+树在数据库的索引原理里面有典型的应用。

4：Hash

Hash概念：

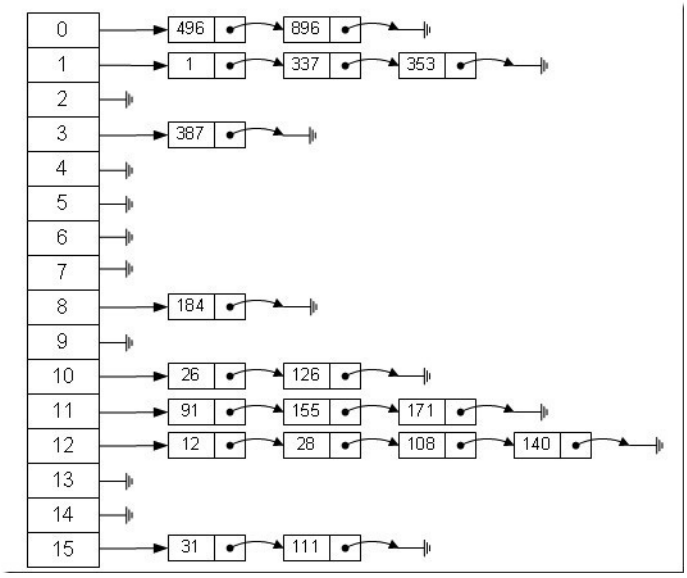
- Hash，一般翻译做“散列”，也有直接音译为“哈希”的，就是把任意长度的输入（又叫做预映射， pre-image），变换成固定长度的输出，该输出就是散列值。一般通过Hash算法实现。
- 所谓的Hash算法都是散列算法，把任意长度的输入，变换成固定长度的输出，该输出就是散列值.（如：MD5,SHA1,加解密算法等）
- 简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。

Java中的hashCode：

- 我们都知道所有的class都是Object的子类，既所有的class都会有默认Object.java里面的hashCode的方法，如果自己没有重写，默认情况就是native方法通过对象的内存的+对象的值然后通过hash散列算法计算出来个int的数字。
- 最大的特性是：不同的对象，不同的值有可能计算出来的hashCode可能是一样的。

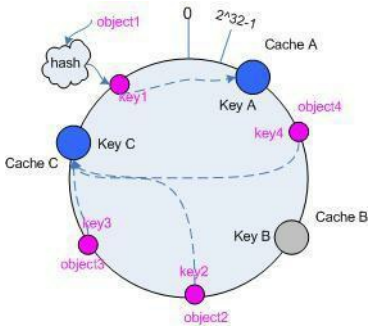
Hash表：

- Java中数据存储方式最底层的两种结构，一种是数组，另一种就是链表。而Hash表就是综合了这两种数据结构。
- 如：HashTable,HashMap。这个时候就得提一下HashMap的原理了，默认16个数组储存，通过Hash值取模放到不同的桶里面去。（注意：JDK1.8此处算法又做了改进，数组里面的值会演变成树形结构。）
- 哈希表具有较快（常量级）的查询速度，及相对较快的增删速度，所以很适合在海量数据的环境中使用。一般实现哈希表的方法采用“拉链法”，我们可以理解为“链表的数组”。



一致性Hash：

- 我们查看一下HashMap的原理，其实发现Hash很好的解决了单体应用情况下的数据查找和插入的速度问题。但是毕竟单体应用的储存空间是有限的，所有在分布式环境下，应运而生了一致性Hash算法。
- 用意和hashCode的用意一样，只不过它是取模放在不同的IP机器上而已。具体算法可以找一下相关资料。
- 而一致性Hash需要注意的就是默认分配的桶比较多些，而当其中一台机器挂了，影响的面比较小一些。
- 需要注意的是，相同的内容算出来的hash一定是一样的。既：幂等性。
-



第二部分：Java基本算法

理解了Java数据结构，还必须要掌握一些常见的基本算法。 理解算法之前必须要先理解的几个算法的概念：

> 空间复杂度：一句来理解就是，此算法在规模为n的情况下额外消耗的储存空间。
>
> 时间复杂度：一句来理解就是，此算法在规模为n的情况下，一个算法中的语句执行次数称为语句频度或时间频度。
>
> 稳定性：主要是来描述算法，每次执行完，得到的结果都是一样的，但是可以不同的顺序输入，可能消耗的时间复杂度和空间复杂度不一样。

一、二分查找算法

二分查找又称折半查找，优点是比较次数少，查找速度快，平均性能好，占用系统内存较少；其缺点是要求待查表为有序表，且插入删除困难。这个是基础，最简单的查找算法了。

```
    public static void main(String[] args) {
        int srcArray[] = {3,5,11,17,21,23,28,30,32,50,64,78,81,95,101};
        System.out.println(binSearch(srcArray, 28));
    }
    /**
     * 二分查找普通循环实现
     *
     * @param srcArray 有序数组
     * @param key 查找元素
     * @return
     */
    public static int binSearch(int srcArray[], int key) {
        int mid = srcArray.length / 2;
//        System.out.println("=:"+mid);
        if (key == srcArray[mid]) {
            return mid;
        }

//二分核心逻辑
        int start = 0;
        int end = srcArray.length - 1;
        while (start <= end) {
//            System.out.println(start+"="+end);
            mid = (end - start) / 2 + start;
            if (key < srcArray[mid]) {
                end = mid - 1;
            } else if (key > srcArray[mid]) {
                start = mid + 1;
            } else {
                return mid;
            }
        }
        return -1;
    }
```

二分查找算法如果没有用到递归方法的话，只会影响CPU。对内存模型来说影响不大。时间复杂度log2n，2的开方。空间复杂度是2。一定要牢记这个算法。应用的地方也是非常广泛，平衡树里面大量采用。

二、递归算法

递归简单理解就是方法自身调用自身。

```
    public static void main(String[] args) {
        int srcArray[] = {3,5,11,17,21,23,28,30,32,50,64,78,81,95,101};
        System.out.println(binSearch(srcArray, 0,15,28));
    }
    /**
     * 二分查找递归实现
     *
     * @param srcArray  有序数组
     * @param start 数组低地址下标
     * @param end   数组高地址下标
     * @param key  查找元素
     * @return 查找元素不存在返回-1
     */
    public static int binSearch(int srcArray[], int start, int end, int key)
{
        int mid = (end - start) / 2 + start;
        if (srcArray[mid] == key) {
            return mid;
        }
        if (start >= end) {
            return -1;
        } else if (key > srcArray[mid]) {
            return binSearch(srcArray, mid + 1, end, key);
        } else if (key < srcArray[mid]) {
            return binSearch(srcArray, start, mid - 1, key);
        }
        return -1;
    }
```

递归几乎会经常用到，需要注意的一点是：递归不光影响的CPU。JVM里面的线程栈空间也会变大。所以当递归的调用链长的时候需要-Xss设置线程栈的大小。

三、八大排序算法

- 一、直接插入排序（Insertion Sort）
- 二、希尔排序（Shell Sort）
- 三、选择排序（Selection Sort）
- 四、堆排序（Heap Sort）
- 五、冒泡排序（Bubble Sort）
- 六、快速排序（Quick Sort）
- 七、归并排序（Merging Sort）
- 八、基数排序（Radix Sort）

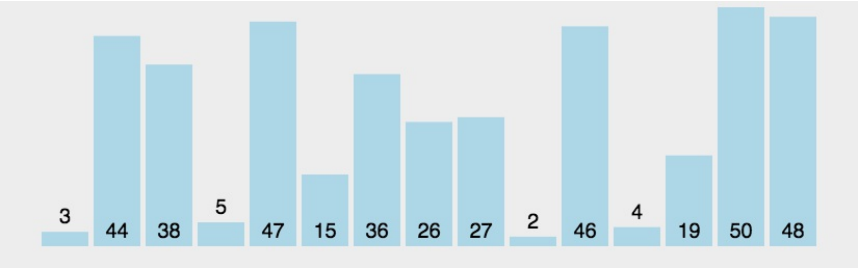
八大算法，网上的资料就比较多了。

吐血推荐参考资料：git hub 八大排序算法详解(https://itimetraveler.github.io/2017/07/18/%E5%85%AB%E5%A4%A7%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95%E6%80%BB%E7%BB%93%E4%B8%8Ejava%E5%AE%9E%E7%8E%B0/)。此大神比作者讲解的还详细，作者就不在这里，描述重复的东西了，作者带领大家把重点的两个强调一下,此两个是必须要掌握的。

1：冒泡排序

基本思想：

冒泡排序（Bubble Sort）是一种简单的排序算法。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。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端。



以下是冒泡排序算法复杂度:

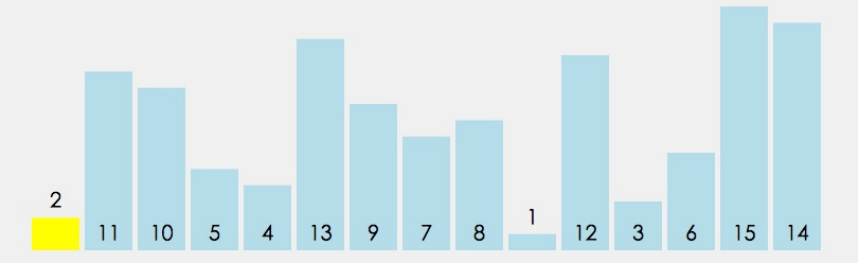
平均时间复杂度	最好情况	最坏情况	空间复杂度
O(n²)	O(n)	O(n²)	O(1)

> 冒泡排序是最容易实现的排序, 最坏的情况是每次都需要交换, 共需遍历并交换将近n²/2次, 时间复杂度为O(n²). 最佳的情况是内循环遍历一次后发现排序是对的, 因此退出循环, 时间复杂度为O(n). 平均来讲, 时间复杂度为O(n²). 由于冒泡排序中只有缓存的temp变量需要内存空间, 因此空间复杂度为常量O(1).

Tips: 由于冒泡排序只在相邻元素大小不符合要求时才调换他们的位置, 它并不改变相同元素之间的相对顺序, 因此它是稳定的排序算法.

```
/**
 * 冒泡排序
 *
 * ①. 比较相邻的元素。如果第一个比第二个大，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①~③，直到没有任何一对数字需要比较。
 * @param arr  待排序数组
 */
public static void bubbleSort(int[] arr){
    for (int i = arr.length; i > 0; i--) {       //外层循环移动游标
        for(int j = 0; j < i && (j+1) < i; j++){     //内层循环遍历游标及之后(或之前)的元素
            if(arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                System.out.println("Sorting: " + Arrays.toString(arr));
            }
        }
    }
}
```

2：快速排序



快速排序使用分治策略来把一个序列（list）分为两个子序列（sub-lists）。步骤为：

①. 从数列中挑出一个元素，称为”基准”（pivot）。

②. 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（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（recursively）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。

递归到最底部时，数列的大小是零或一，也就是已经排序好了。这个算法一定会结束，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。

代码实现：

用伪代码描述如下：

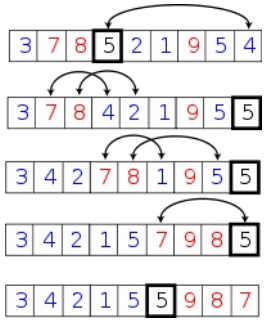
①. i = L; j = R; 将基准数挖出形成第一个坑a[i]。

②. j--，由后向前找比它小的数，找到后挖出此数填前一个坑a[i]中。

③. i++，由前向后找比它大的数，找到后也挖出此数填到前一个坑a[j]中。

④. 再重复执行②，③二步，直到i==j，将基准数填入a[i]中。

快速排序采用“分而治之、各个击破”的观念，此为原地（In-place）分区版本。



```
/**
 * 快速排序（递归）
 *
 * ①. 从数列中挑出一个元素，称为"基准"（pivot）。
 * ②. 重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（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（recursively）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。
 * @param arr   待排序数组
 * @param low   左边界
 * @param high  右边界
 */
public static void quickSort(int[] arr, int low, int high){
    if(arr.length <= 0) return;
    if(low >= high) return;
    int left = low;
    int right = high;
    int temp = arr[left];   //挖坑1：保存基准的值
    while (left < right){
        while(left < right && arr[right] >= temp){  //坑2：从后向前找到比基准小的元素，插入到基准位置坑1中
            right--;
        }
        arr[left] = arr[right];
        while(left < right && arr[left] <= temp){   //坑3：从前往后找到比基准大的元素，放到刚才挖的坑2中
            left++;
        }
        arr[right] = arr[left];
    }
    arr[left] = temp;   //基准值填补到坑3中，准备分治递归快排
    System.out.println("Sorting: " + Arrays.toString(arr));
    quickSort(arr, low, left-1);
    quickSort(arr, left+1, high);
}
```

以下是快速排序算法复杂度:

平均时间复杂度	最好情况	最坏情况	空间复杂度
$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	O(1)（原地分区递归版）

快速排序排序效率非常高。 虽然它运行最糟糕时将达到$O(n^2)$的时间复杂度, 但通常平均来看, 它的时间复杂为$O(n\log n)$, 比同样为$O(n\log n)$时间复杂度的归并排序还要快. 快速排序似乎更偏爱乱序的数列, 越是乱序的数列, 它相比其他排序而言, 相对效率更高.

最后，作者希望让大家对《Java数据结构》整体有个全面的了解,知道什么是数据结构，离我们工作中有多远，而不是一个深不可测的神秘物件。里面的细节，篇幅有限可能不能描述完，但是只要同学们的方向没有搞错，那只要针对每个点再详细的看看即可。

面试和工作，这些都是离不开的，当同学们有个完整的认识之后，一定要在工作中留心，留意每个用到的地方。

交流：欢迎大家一起留言，把自己碰到的Java数据结构问题一起讨论。
作者：张振华.Jack。
QQ交流群一：240619787
QQ交流群二：559701472

写评论

Sea

老师，连KMP算法都不讲，很多知识点一句话带过，请问写这篇文章的意义在何处？

1月7日 8 1

张振华: 好意见，篇幅有限，只讲一下精华部分，留了一点悬念在gitchat群里现场交流，给大家留了一点思考空间。也希望大家多思考，说明这位同学还是认真了，值得肯定。

张振华

如果大家觉得老师付出的值的大家肯定，帮老师点赞和打5分好评哦。这样老师更有动力。

1月6日 5 0

梅小西

我发现这个老师写的文章，基本上都是『点到为止』，『一笔带过』，请问都能随便百度到的东西，为何要花钱来看？

3月8日 3 1

任武杰: 想让你花钱买人家的课撒

Mr.Potter

很赞！感谢！

1月5日 2 0

李烨

预告的内容多一半没写啊

5月16日 2 0

yonguo

概括性的介绍，写的很清楚，感谢分享。请问有更详细的讲解Java数据结构和算法的资料或者书籍推荐吗？

1月10日 1 0

jack

写的很有深度，讲解详细，五星好评，赞！

1月10日 1 0

张曦

提纲挈领的作用吧,具体的知识没有讲到,给出了要掌握的知识地图,辛苦老师

2月14日 1 0

LeiDaGou

在实际项目中可以用到的示例呢？不是说让我们看了让我们可以在项目中使用，之后不容易忘记吗？

7月31日 0 0

张宇

内容不够深入，缺乏深度分析。

1天前 0 0