Hashtable

ArrayblockingQueue

LinkedBlockingQueue

Arrays.sort()

**Collections.sort()源码分析**

## **简介**

1. 在Java 6中Arrays.sort()和Collections.sort()使用的是MergeSort，而在Java 7中，默认实现换成了TimSort。
2. Collections.sort调用容器的toArray函数，将容器转换成Object数组后，再通过Arrays.sort()进行排序；排序完成后再遍历容器，将更新容器中的每个元素值；

## **TimSort原理**

如果待排序数组长度<32，则使用binarySort；   
否则，使用TimSort：

1. 计算minRun（子序列的长度）：   
   1.1. 若待排数组总长度为2^n，则minRun选取为16；   
   1.2. 否则，通过将数组长度右移的方式（每次除以2），计算出一个16-32之间的值，作为minRun；
2. 使用binarySort排序每个子序列；
3. 最后用mergeSort对所有的子序列排序；

## **MergeSort原理**

mergeSort是一个将两个有序序列合并成一个有序序列的算法。   
它用两个指针i、j，分别指向两个有序子序列的头部，并用一个额外的数组，存储合并后的序列；   
若a[i]<=a[j]：将a[i]存入那个额外的数组，并i++;   
若a[i]>a[j]：将a[j]存入那个额外的数组，并j++;   
直到某一个子序列为空，最后把另一个非空子序列输出至额外数组中即可；

### **时间复杂度**

速度仅次于快速排序，为稳定排序算法，一般用于对总体无序，但是各子项相对有序的数列。   
不管是什么情况下，时间复杂度均为O(nlogn)，空间复杂度为O(n)。

### **MergeSort时间复杂度证明**

MergeSort一般采用递归，不停的将子序列一分为二，直到子序列只包含一个元素为止；然后开始回溯，将相邻的两个子序列两两合并，直到合并成一个序列为止。   
这个过程中，一共要进行logN趟合并操作，每一趟合并都要遍历当前子序列所有的元素，把它们输出到一个新的数组中去，所以总体时间复杂度是O(nlogn)。

### **快排的时间复杂度证明**

快排需要使用递归将序列一分为二，那么一共要进行logn趟操作；   
每一趟都要执行一次partition函数，因为partition中要扫描所有的元素，时间复杂度为O(n)；   
所以快排总的时间复杂度是O

## **BinarySort原理**

binarySort是一种插入排序，直接插入排序通过遍历的方式向前搜索插入点，而binarySort通过二分搜索的方式搜索插入点。   
过程如下：

1. 将待排序列分成两段，前半段为有序序列，后半段为无序序列；
2. 每趟在有序序列中为无序序列的第一个元素寻找插入位置，并使用二分搜索的方式寻找；
3. 当无序序列长度为0的时候整个排序完成。

### **时间复杂度**

插入排序不管在什么情况下，都要为n-1个无序元素寻找插入点；   
当使用二分搜索寻找插入点时，查找的时间复杂度为O(logn)，所以经过n-1躺查找后，时间复杂度为O(n\*logn)。   
而直接插入排序，由于采用向前遍历的方式查找插入点，因此时间复杂度为O(n^2)。

## **策略模式**

Collection.sort通过Comparator作为排序的规则，这是策略模式。   
在实现一个功能的时候，如果这个功能的某个步骤可能会有多种实现，或者在编码的时候还没办法将具体的实现确定下来，这时候就可以使用策略模式，将这个功能封装成一个接口，具体的实现由调用者自己去定义，这就是策略模式的思想。