排序算法

1. Timsort

1.1 原理

现实中的大多数据通常是有部分已经排好序的，Timsort利用了这一特点达到高效的目的。

1. `Timsort` 将排序数组拆分成许多个分块（这些分块叫 `run` ）
   - 这些 `run` 不是随便拆分的，而是有顺序的，拆分后的每个 `run` 都是有序的（降序/升序）， `run` 的最小长度是2；
   - 如果 `run` 是升序则 `run` 保持原样，如果 `run` 是**严格降序**，则进行翻转，变成升序；

     > `严格降序` ：不存在相等的情况；
     > 遵守 `严格降序` 的目的是为了保证排序的稳定性；

2. 合并 `run`
   - 为了提高合并效率， `Timsort` 采用 `二分插入排序算法` 进行合并；

     > `二分插入排序算法` 是直接插入排序算法的优化版，是在插入的时候使用 `二分查找` 找到插入位置；然后再插入；

     > *我们要将A和B这2个run 合并，且A是较小的run。因为A和B已经分别是排好序的，二分查找会找到B的第一个元素在A中何处插入（图4）。同样，A的最后一个元素找到在B的何处插入，找到以后，B在这个元素之后的元素就不需要比较了（图5）。这种查找可能在随机数中效率不会很高，但是在其他情况下有很高的效率。*
     > 理解：似乎二分查找只是用于查找上述中的边界位置；

1.2 Java中

假设：排序的数组为a，a的长度为aLength

1.2.1 aLength == 1

表示待排序元素数量为1，所以没必要排序，直接返回；

1.2.2 aLength < 32

1. 先从a的第一个元素开始，查找连续的数（无论升序/降序），如果是降序就转成升序；

   > 如：数组 `7,8,9,10,13,4,1,2, ...` ，此步骤就会得到 `7,8,9,10,13` ；数组 `10,9,7,4,2,20,30,33,34 ...` ，此步骤就会得到 `2,4,7,9,10` ；
   > 方法：countRunAndMakeAscending

2. 然后再通过 `二分插入排序` 对剩下的元素进行排序；

1.2.3 aLength >= 32

1. 计算 `run` 的最小长度
   这个算法特别巧妙：直接上代码：

```
// 代码位置：Arrays.sort(Object[] a) --> ComparableTimSort.sort
// --> minRunLength(nRemaining)
private static final int MIN_MERGE = 32;
```

```
private static int minRunLength(int n) {
    assert n >= 0;
    int r = 0;      // Becomes 1 if any 1 bits are shifted off
    while (n >= MIN_MERGE) {
        r |= (n & 1);
        n >>= 1;
    }
    return n + r;
}
```

> 上述代码：如果 `n` 是奇数， `r` 始终为1；如果 `n >>= 1` 为奇数且大于等于 `MIN_MERGE` ， `r` 也为1；其他情况下， `r` 均为0；此算法的目的是为了使用 `minRunLength` 对 `aLength` 均分时，每个片段长度尽量接近；

2. 拆分数组
   2.1 同 `1.2.2` 的步骤1：

```
int runLen = countRunAndMakeAscending(a, lo, hi);
```

   2.2 如果 `runLen < minRunLength`
   采用 二分插入算法 从后续的元素中获取元素将runLen扩展到 `minRunLength` ；

3. 最后就是合并所有的run；