男科一梦（再续一集）-TimSort的实现

原创：育树霖疯 育树霖疯 2017-09-07

> 书接上回，小育通过冯诺依曼了解到了模板模式，插入排序和归并排序参见男科一梦（一）：模板模式，男科一梦（续集）：归并排序。也从乔西那里得知JDK1.7之前Arrays.sort()几乎采用归并排序。这个时候一个不速之客来到。他就是TimSort算法的发明者Tim Peters.

乔西：在JDK1.7以前我们几乎就是按照MergeSort来实现Arrays.sort()的。只是细节上略有差别。当集合小于7的时候我们使用插入排序。当集合大于7的时候，先通过递归分解，把集合分解成若干等于7的子集合。我们再采用插入排序来进行子集合的排序。这是因为插入排序在小集合排序上最坏时间复杂度和MergeSort的运算量差不多,但最好情况优于MergeSort。具体的实现你可以通过Arrays里legacyMergeSort()看到，不过你动作要快，因为我们准备把它移除掉。

小育：为什么要移除掉？是有了更好的方法了吗？

不速之客-TimSort

乔西正准备开口，一个骑着大蟒蛇的金发碧眼男子从天而降。

乔西：可能这个问题，由Tim Peters来解释比较合适。

小育：Tim Peters，那个骑着乌梢蛇的男人？

乔西：对。

Tim：人生苦短，我用Python。Timsort是我最早用在python上的排序方法。后来java觉得这方法还不错，从JDK从1.7开始，采用了Timsort代替了传统的MergeSort。Timsort基于实际情况出发，对于待排序集合原本就部分有序，进行了工业级改进。这是一个稳定的具有*自适应性*的MergeSort算法。尽量避免出现MergeSort中nlog2n最坏时间复杂度的情况。实际运行起来效果会比传统的MergeSort好很多。乔西不是对小于7的集合用插入排序代替MergeSort吗。我和他的出发点都是一样的，只是Timsort稍稍复杂一点，可以应对各种情况，这就是我所提到的自适应。

Tim：TimSort根据集合的规模分为两种情况。如果小于32那么我们用mini-timsort实现。

1. 我们先找到这个序列中从第一个位置开始的非严格升序（可以接受等于的情况，文章中后面简称升序）或者严格降序（不能接受等于的情况，文章中后面简称降序）。比如对集合{7,4,2,1,1,3,5}进行排序。我发现从第一个元素开始是一个降序。那么我把集合分割成下面两个字集合。{7,4,2,1}{1,3,5}。因为严格降序，第5个元素1，被划入后一个集合。

(图3.1找出升序或降序)

2. 反转集合{7,4,2,1}->{1,2,4,7}

（图3.2如果为降序，反转集合）

3. 把剩下的部分{1,3,5}用二分插入的方式插入已经排好序列的{1,2,4,7}中。

（图3.3用二分插入的方式将白色元素插入升序集合中，完成排序）

小育：我们用MergeSort的时候，遇见{9，8，7，6，5，4，3，2，1}这样的降序集合，是MergeSort时间复杂度最坏的情况。而Timsort一旦发现是降序，他直接反转集合就行。确实帮计算机省了不少电。那谈谈集合大于32的情况吧。

Tim：这个要稍微复杂一些。我先给你一些结构性的描述，具体的细节我待会展开讲：
1. **找出升序**：和上面<32的步骤1一样，找出一个升序，或者一个经过反转的降序。命名为initRun。用{1,2,3,4,5,6,7,8,9,5,3,1,4,…n}做个例子吧。

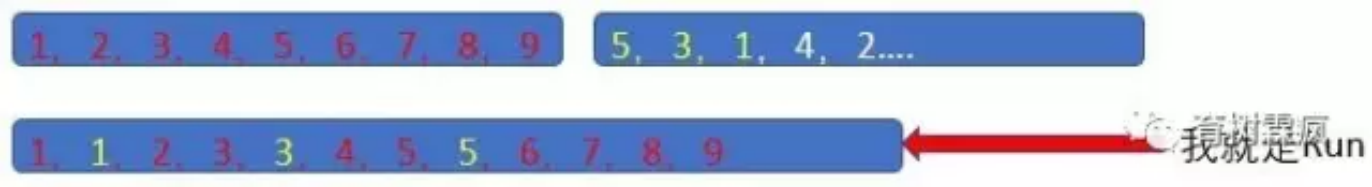
（图3.4找出自然升序）

2. **计算最小合并序列长度minRun:**根据剩余待排序元素的个数nRemaining（第一次计算时：nRemaining为整个待排序元素的总数），计算出一个minRun，在步骤3中用到（后面会详细讲解计算细节）。

3. **构造合并序列run:**比较initRun与minRun的长度，如果:

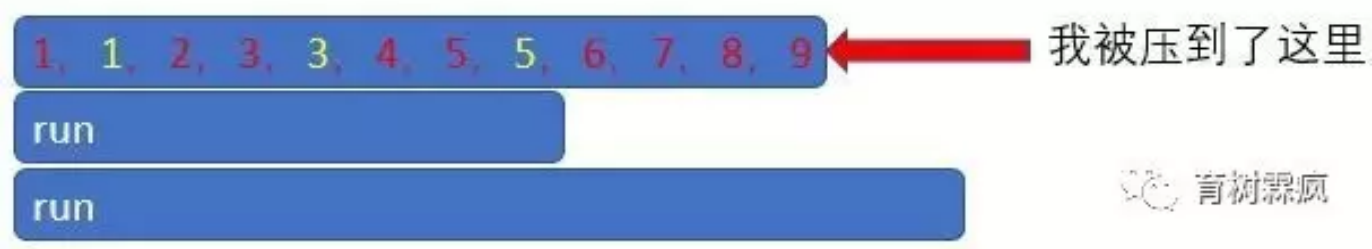
a. initRun < minRun，用二分插入排序，把原集合中initRun后面的元素补足minRun个元素。构造出一个集合命名为run。

b. initRun >= minRun，run取initRun。



（图3.5假设步骤2计算出本次的minRun是12，这确实是一个很假的假设，默认minRange=32，因为实际情况minRun会在16到32之间，initRun为9，那么取后面3个元素补足12位，形成run）

4. **压栈等待合并：**将步骤3中，构建的run压入名runBase的栈中。



(图3.6栈runBase)

5. **构造runBase的阶梯结构：**将runBase中的run进行合并，根据run中的元素个数栈构造出上小下大的阶梯结构。这个阶梯要求：阶梯内任意3级从下向上的run的元素个数（run0, run1, run2）满足run0>run1+run2 && run1>run2。不满足条件的run就会被合并，合并的原则如是：每一次有新的run被压入栈，那么我们比较栈顶3个run的元素个数（栈顶的run是新压入栈的run）。

a. 如果run0<=run1+run2,那么run1与run0和run2中元素较少的那个run合并。b. 如果run0>run1+run2,但run1<=run2,那么run1与run2合并。

c. 重复步骤a和b，直到满足条件run0>run1+run2 && run1>run2或者baseRun中不足3个run。

（图3.7构造结束后稳定结构的栈runBase）

6. 重复步骤1-5，直到所有的元素都被压入栈中（nRemaining==0）。

7. 栈合并：将稳定结构的栈runBase进行合并，直到所有的run都被合并，合并后的run就是最后的排序结果。当然在合并的过程中，通常情况下用栈顶的run与下一个run进行合并，比如下图中的run4与run3合并。如果出现了run4>run2的情况（假如run4=8,run2=7），那么我们合并run3与run2(合并的原则是合并相邻两个短的run)。

（图3.7稳定结构的栈runBase）

（图3.8 run4与run3合并后）

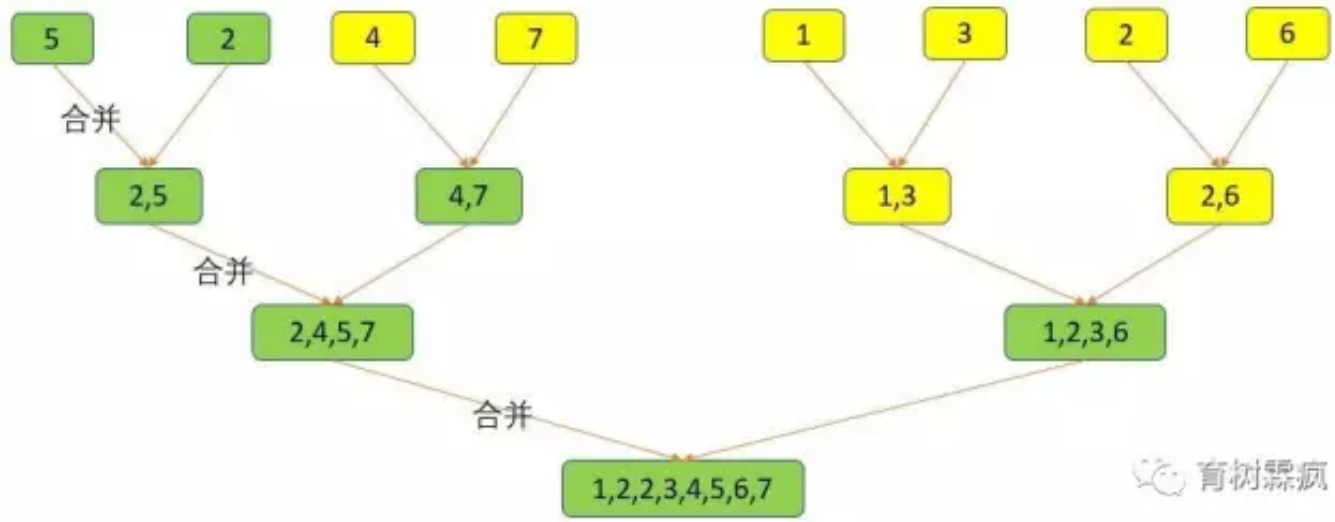
（图3.9 run3与run2合并后）

（图3.10 run2与run1合并后）

（图3.11 run1与run0合并后,合并完成，排序结束，run0排序结果）

小育：第5步构造runBase的时候，我有个疑问。为什么这个阶梯一定要满足run0>run1+run2&& run1>run2这样的条件呢？

Tim:还记得冯老爷子讲的MergeSort吗？MergeSort合并的时候栈里元素个数是什么样的情况？下面是MergeSort那张图，我只保留了合并的部分，然后用绿色标注了一组末端元素的合并过程。你想想如何我们根据计算顺序把绿色的部分压栈，会是怎样？



(图3.12)

小育：情况是这样。

(图3.13)

我明白了，你是在模拟MergeSort的合并的部分。你用run0>run1+run2&& run1>run2这条件构造出与MergeSort类似的楼梯结构。

Tim:你说的很不错。TimSort是MergeSort的变种而已。还有些细节我需要传授给你。

如何计算minRun:首先你要知道minRun的长度是动态的，他是根据每次nRemaining计算出来的。计算方式如下：

a. 如果nRemaining<32那么，minRun就是nRemaining。

b. 如果nRemaining>=32,那么minRun的值会在16到32之间。为了实现这一点，我们对nRemaining不断的除以2，直到得到一个k,16<=k<=32。

- 如果除以2的过程中，出现过不能整除的情况，那么最终minRun=k+1。
- 如果除以2的过程中，一直能够整除，那么最终minRun=k。

这里有一个很简单的例子：分别是nRemaining为100和102的例子。

nRemaining=100：不断除以2；100->50->25;k=25;16<=25<=32;除以2过程中都能除尽，k=25。

nRemaining=102：不断除以2；102->51->25;k=25;16<=25<=32;51/2不能整除。minRun=k+1=26。

小育：这样是为了避免run过小，合并过于频繁的情况？

Tim:对。最后一个优化放在了两个集合之间的合并上。

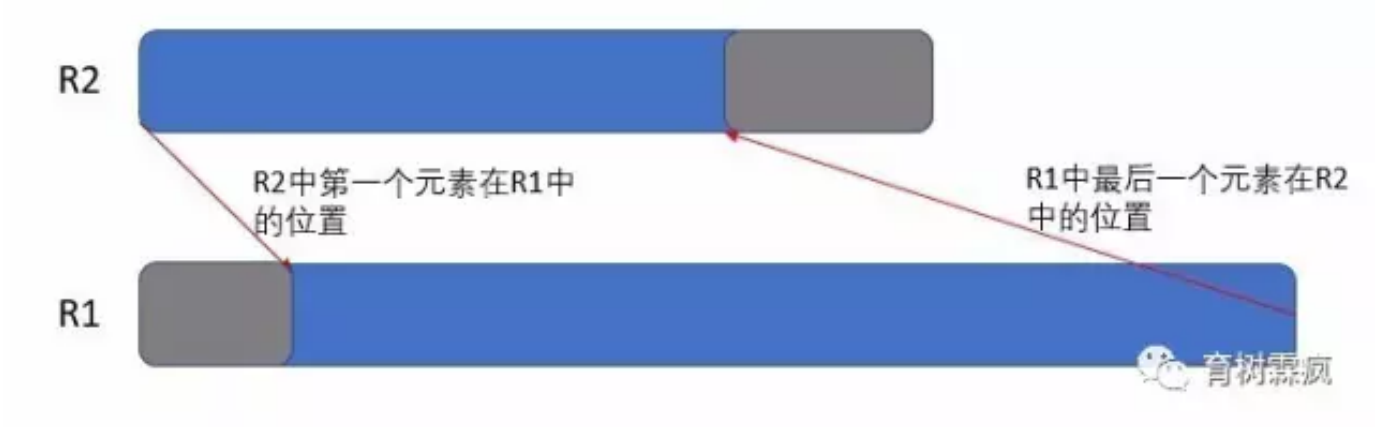
1. 我们现在有两个进行合并的栈，我们从上到下把他们称为R2和R1。R2和R1**已经是有序的集合**，记住这一点，这很重要。

2. 找出R2中第一个元素在R1中的位置。在这个位置之前的R1中的元素，不需要参加比较（图3.14 R1中灰色部分），因为他们一定比R2中任意元素小。

3. 找出R1中最后一个元素在R2中的位置。在这个位置之后的R2中的元素也不需要参与比较（图3,14 R2中灰色部分），因为他们一定比R1中的任意元素大。

4. 划分完成后，蓝色部分用MergeSort方式合并（参见男科一梦（续集）：归并排序图2.2）。在合并过程中，发现R1连续7次比R2小（或者大），那么我们再次重复步骤2和3，划分出新蓝色部分。直到不满足连续7次小（或者大），则重新采用MergeSort的方式进行合并。

小育：按照你的优化，参与比较的元素是R1和R2中蓝色的部分，灰色的部分不再需要参与比较了。然后再用MergeSort一一比较的过程中发现一个集合中的元素连续7次比另外一个元素大，我们就会猜想，那个集合可能还有很多元素都比另外一个集合大，所以我们需要再次划分出灰色的不比较空间。从而减少比较次数。



(图3.14步骤2和3的划分)

Tim:正是如此。

小育：果然是工业级的优化。增加了排序的自适应性，避免排序向MergeSort的最坏时间复杂度nlog2n靠拢。

1. 他们能识别降序进行翻转，获取待比较序列中连续的升序列。

2. 动态计算minRun以减少合并次数。

3. 把要合并的run模拟压栈成mergeSort的样子。

4. 自动排除一些不需要比较的头部和尾部。

5. 在一个序列总比另外一个序列小的时候，他会猜测会有更多的数据满足这个情况，再次划出那些不用去比较的头部和尾部。

Tim:乔西在JDK1.7的时候把TimSort引入了java，有兴趣你可以去看看Java版的实现。

突然间，老冯家的房子上落了一匹瓦下来。

乔西：时间差不多了，这一层梦境要坍塌了，你得赶紧回到现实世界去了。这是一个梦境手指陀螺，你拿着。我担心不能一次把你送出梦境，这样你就会迷失在梦境空间。如果你醒来发现手指陀螺还在，你旋转它，如果它像吃了炫迈一样根本停不下来。说明你依然在梦境中。你得想办法跳个楼啥的脱离那层梦境。是时候和老冯和Tim告别了。

小育：多谢你们。真是一个好梦。

老冯：不用谢我们，梦由心生。

Tim:有空学学Python。

回归现实

房子彻底坍塌了，小育也醒来了，醒来之后一桌子的口水，还放着一本乔西的《Effect Java》。书上面放着一个小包裹，里面还真有一个手指陀螺，不过是小育前几天在淘宝买的。小育打开包裹，一个黄铜做的手指陀螺和一个写着好评立即返5元的宣传单。他拿起陀螺转了起来。德国工艺就是好，陀螺飞快的转着。必须好评，还能返立返5元呢。好评完后，返钱到帐，陀螺依旧转着，一点都没比刚才慢，根本停不下来。小育的背心有点发凉了，想起醒来之前乔西告诉他梦境手指陀螺的事情。1小时过去了，陀螺还转着，根本停不下来，像吃了一箱炫迈一样。恐怕这不是德国工艺能达到的境界。要么是外星科技，要么我还在做梦，难道我和小燕分手也只是一场梦？小育不自觉的走到阳台上。纠结着跳还是不跳。突然被人从后面踹了一脚。

小育从睡梦中惊醒过来，抬头一看床边是小燕的脚。。。

(全文完)