20181123 数据结构作业

1800022769　　张靖昆　　20181123

我承诺诚实作业，没有抄袭他人！

1．**第 2 题**：**请查看可执行代码项目，代码使用 JetBrain 开发的 Clion 书写。**

2．**第 3 题**：试给出采用增量序列$\{2^k-1, 2^{k-1}-1, \ldots, 7, 3, 1\}$的 Shell 排序的时间代价为$\Theta(n^{3/2})$的推理过程。

解：

说明：以下证明中以实数来确定时间复杂度的推算，不考虑取整的处理，因为无论是否取整对结果都没有影响。

1）**引入引理**：如果一个数组以增量 g 已经有序，以增量 h 已经有序，并且 g 和 h 互质，都是 d 的常熟倍(Θ(d))，则以增量 d 对 a 排序的时间复杂度为Θ(nd)。

2）我们从增量序列$\{2^k-1, 2^{k-1}-1, \ldots, 7, 3, 1\}$中找到一个最接近$n^{1/2}$的$d_t$，来考察该增量序列下时间复杂度的上下界，即对 k≤t 和 k＞t 的情况下的时间复杂度。k 为序列的数量。

3）k≤t：对于$d_k = 2^k - 1$，容易看到d_{k-1}与d_{k-2}互质且规模均为$\Theta(d_k)$，因此对于增量d_k，其排序的时间复杂度为$\Theta(nd_k)$，那么对于所有的 1≤k≤t，其总时间复杂度为$\Theta(n) + \Theta(n \times (2^2-1)) + \cdots \Theta(n \times (2^t-1)) =$

$\Theta(n \times (2^t - 2 - t))$，而$2^t - 2 - t$和$2^t - 1$一个数量级，因此总是时间复杂度为$\Theta(n \times (2^t - 1)) = \Theta(n^{3/2})$。

4）k>t：对于d_k，需要对d_k个长度为$\frac{n}{d_k}$的子序列分别排序，由于对于长度为 n 的序列进行直接插入排序的时间代价是$\Theta(n^2)$，因此对d_k个长度为$\frac{n}{d_k}$排序的时间复杂度为$\Theta\left(d_k * (\frac{n}{d_k})^2\right)$即$\Theta\left(\frac{n^2}{d_k}\right)$，那么对于所有 k>t，其总时间复杂度为$\Theta\left(\frac{n^2}{d_{t+1}}\right) + \Theta\left(\frac{n^2}{d_{t+2}}\right) + \cdots$，根据等比数列性质，他与$\Theta\left(\frac{n^2}{d_k}\right)$一个数量级，所以总时间复杂度为$\Theta\left(\frac{n^2}{d_k}\right)$，即$\Theta\left(\frac{n^2}{n^{1/2}}\right) = \Theta(n^{3/2})$。

综上，采用增量序列$\{2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1\}$的 Shell 排序的时间代价为$\Theta(n^{3/2})$。

3．**第 7 题**：在堆排序中采用的堆是基于二叉树的，因此时间代价是Θ(n log n)。如果基于三叉树的堆来实现堆排序，时间代价是否会变为$\Theta(n\ log_3 n)$？是编写出基于三叉树的堆排序算法，并分析时间代价。

解：算法请查看可执行代码项目，代码使用 JetBrain 开发的 Clion 书写

分析时间代价：

算法与二叉树相差不大，核心部分首先是建立初始大顶堆，然后进行 n 次大顶堆的调整。建立大顶堆所需时间为Θ(n)，调整 n 次大顶堆所花费的时间代价都是$\Theta(n\ log_3 n)$，所以总的时间代价是$\Theta(n\ log_3 n)$。因为三叉树的树高只在$log_3 n$级别，共 n 次操作，从而时间代价就是$\Theta(n\ log_3 n)$。但要注意的是其本质上和Θ(n log n)等

价，因为$log_3 n = log_3 2 * log\ \ n$。

4．**第 25 题**：在下列情况下，你会选择那种排序算法？请说明理由。

1）需要对 1000 个数字进行排序，程序只需运行一次

解：采用归并排序或堆排序。1000 个数字是一个规模较大的数据集，因此适用 O(n log n)的算法。不采用快速排序是因为数字的初始状态不知，快速排序有可能达到最坏的情况，这两种排序虽然比平均上比快拍慢，但是速度稳定，最好和最差都是 O(n log n)级别。

2）只需要对 50 个数字进行排序，程序只需运行一次。

解：选择直接插入排序。50 数字属于小规模数据，因此 O(n log n)的算法没必要采用。不采用选择排序和冒泡排序是因为他们的速度没有直接插入排序快，他们有过多的时间浪费在了比较上面。

3）在编写一个很庞大的程序中需要编写一个对 5 个记录进行排序的函数，而且该函数将被反复调用很多次。

解：采用直接插入排序或归并排序。从教材 P231 页表 8.4 可知，对于这种小规模数据，这两种算法的效率最佳。

4）需要对 1000 个大型的记录进行排序，记录本身存储在外存中，在内存中只保存了所有记录的排序码。排序码之间的比较非常快，但是移动代价大，因为一旦移动一个排序码，相应的外存中的记录也要移动，将涉及上百个磁盘块的移动。

解：采用直接选择排序。1000 个数字是大数据，但是由于 O(n log n)算法的移动次数无法估量，因此风险保守考虑不采用。三个简单排序算法中移动次数只有直接选择排序的平均情况是 O(n)的代价，其他两个平均状况都是 $O(n^2)$，因此采用直接选择排序。

5）在图书馆中里计算机类书籍区，一共有 12 列书架，书架上的书本都是按照编目号排列好的，但其中有些书被读者放错了地方，但通常不会偏离超过一个书架。试设计一个算法将这些数重新放回正确的位置。

解：根据题目开始要求（选择那种排序算法并说明理由），这里不考虑算法的具体实现。**应当选择插入排序**。这里类似索引排序中每个小分组选择的排序，在分组内选择插入排序就可以，因为数据规模比较小。

6）需要将 500 张随机排列的图书卡片按照字母顺序排好序。

解：选择快速排序。对于随机排列且数据规模较大的情况下，快速排序有着最佳的优势。

7）已知一个包含了 5000 个单词的列表已按照字母顺序排好列，需要再进行一次检查，确保所有单词已经排好序。

解：选择冒泡排序。由于列表中单词已经基本有序，只需要进行一次检查，那么冒泡排序可以用于检查，即在有序的情况下，冒泡排序只需检查一次就可确定是否有序。

5．**第 26 题**：某整型数组 A 的 10 个元素值依次为{6,2,9,7,3,8,4,5,0,1}用下列各排序方法，将 A 中的元素有小到大排序。

1）取第一个元素值为 6 作为分割数，试写出快速排序第一次分隔后 A 中的结果。

解：使用表格来看一下第一次快速排序的操作过程。其中 6 为分割数，将它从位置 0 暂时拿出来，并从数组末端即指针 j 开始查找。

a）**初始状态**

下标	0	1	2	3	4	5	6	7	8	9
指针 i	↓									

数组		2	9	7	3	8	4	5	0	1
指针 j										↑

b）第一次查找，发现 1 小于 6，将它放置在下标 0，并将。

下标	0	1	2	3	4	5	6	7	8	9
指针 i		↓								
数组	1	2	9	7	3	8	4	5	0	
指针 j										↑

c）第二次查找，发现 9 大于 6，将它放在下标 9，并将指针 j--。

下标	0	1	2	3	4	5	6	7	8	9
指针 i			↓							
数组	1	2		7	3	8	4	5	0	9

指针 j								↑	

d）第三次查找，发现 0 小于 6，将它放置在指针 i 所对位置下标 2，并将 i++。

下标	0	1	2	3	4	5	6	7	8	9
指针 i				↓						
数组	1	2	0	7	3	8	4	5		9
指针 j									↑	

e）第四次查找，发现 7 大于 6，将它放置在 j 所指位置下标 8，并将 j--。

下标	0	1	2	3	4	5	6	7	8	9
指针 i				↓						
数组	1	2	0		3	8	4	5	7	9
指针 j								↑		

f）第五次查找，发现 5 小于 6，将其放在 i 所指位置下标 3，并将 i++。

下标	0	1	2	3	4	5	6	7	8	9
指针 i					↓					
数组	1	2	0	5	3	8	4		7	9
指针 j								↑		

g）第六次查找，发现 8 大于 6，将其放在 j 所指位置下标 7，并将 j--。

下标	0	1	2	3	4	5	6	7	8	9
指针 i						↓				
数组	1	2	0	5	3		4	8	7	9
指针 j							↑			

h）第七次查找，发现 4 小于 6，将其防止在 i 所指位置下标 5，并将 i++。

下标	0	1	2	3	4	5	6	7	8	9
指针 i							↓			
数组	1	2	0	5	3	4		8	7	9
指针 j							↑			

i）第八次查找，发现 i 与 j 相遇，代表第一次的分隔结束，将 6 放置在 i 所指位置下标 6 结束。所得到最终序列如表中数组所示{1,2,0,5,3,4,6,8,7,9}

下标	0	1	2	3	4	5	6	7	8	9
指针 i							↓			
数组	1	2	0	5	3	4	6	8	7	9
指针 j							↑			

2）用堆排序，试写出将第一个选出的数据放在 A 的最后位置上，将 A 调整成堆后的 A 中结果。

解：题目中是“写出”，并没有要求过程，因此直接给出堆排序的答案。

建立初始大顶堆：{9,7,8,5,3,6,4,2,0,1}

将第一个放置最后位置：{7,8,5,3,6,4,2,0,1,9}

再次调整：{8,7,6,5,3,1,4,2,0,1,9}，此为最终答案

3）用基数为 3 的基数排序法，试写出第一次分配和收集后 A 中的结果

解：由于本题的元素非常特殊，因此第一次分配和收集后 A 中结果为{0,1,2,3,4,5,6,7,8,9}

6．**第 27 题**：已知一组元素的排序码为{46,74,16,53,14,26,40,53’,86,65,27,34}

1）利用直接插入排序的方法写出每次向前面有序表插入一个元素后的排序结果

解：已经就位的元素以浅绿色底色标记，每次被操作的元素以边框标记

a）**第 1 次**：46 是第一个元素，不需要动：

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	74	16	53	14	26	40	53’	86	65	27	34

b）**第 2 次**：74 大于 46，不需要动：

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	74	16	53	14	26	40	53’	86	65	27	34

c）**第 3 次**：将 16 插入到下标 0，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	16	46	74	53	14	26	40	53’	86	65	27	34

d）**第 4 次**：将 53 插入在下标 2，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	16	46	53	74	14	26	40	53’	86	65	27	34

e）**第 5 次**：将 14 插入下标 0，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	46	53	74	26	40	53’	86	65	27	34

f）**第 6 次**：将 26 插入到下标 2，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	46	53	74	40	53’	86	65	27	34

g）**第 7 次**：将 40 插入到下标 3，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	40	46	53	74	53’	86	65	27	34

h）**第 8 次**：将 53’插入到下标 6，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	40	46	53	53’	74	86	65	27	34

i）**第 9 次**：86 大于 74，因而不需要挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11

元素	14	16	26	40	46	53	53’	74	86	65	27	34

j）**第 10 次**：将 65 插入到下标 7，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	40	46	53	53’	65	74	86	27	34

k）**第 11 次**：将 27 插入到下标 3，并将其他已就位元素向后挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	40	46	53	53’	65	74	86	34

l）**第 12 次**：将 34 插入到下标 4，并将其他已就位元素向后挪动。

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	34	40	46	53	53’	65	74	86

第 12 次所得序列即为最终的使用插入排序所得序列。

2）利用直接选择排序方法写出每次选择和交换后的排列结果

解：每次就位的最大元素以浅绿色底色标记。采用每次选择最大的方法

a）**第 1 次**：从下标 0-11 中选出最大的元素即 86 与下标 11 的元素 34 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	74	16	53	14	26	40	53’	34	65	27	86

b）**第 2 次**：从下标 0-10 中选出最大的元素即 74 与下标 10 的元素 27 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	27	16	53	14	26	40	53’	34	65	74	86

c）**第 3 次**：从下标 0-9 中选出最大的元素即 65，发现不需要挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	27	16	53	14	26	40	53’	34	65	74	86

d）**第 4 次**：从下标 0-8 中选出最大的元素即 53（注意是从下标 0 开始查找）与下标 8 的元素 34 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	27	16	34	14	26	40	53’	53	65	74	86

e）**第 5 次**：从下标 0-7 中选出最大的元素即 53’，发现不需要挪动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	27	16	34	14	26	40	53’	53	65	74	86

f）**第 6 次**：从下标 0-6 中选出最大的元素即 46 与下标 6 的元素 40 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	40	27	16	34	14	26	46	53’	53	65	74	86

g）**第 7 次**：从下标 0-5 中选出最大的元素即 40 与下标 5 的元素 26 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11

元素	26	27	16	34	14	40	46	53’	53	65	74	86

h）**第 8 次**：从下标 0-4 中选出最大的元素即 34 与下标 4 的元素 14 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	26	27	16	14	34	40	46	53’	53	65	74	86

i）**第 9 次**：从下标 0-3 中选出最大的元素即 27 与下标 3 的元素 14 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	26	14	16	27	34	40	46	53’	53	65	74	86

j）**第 10 次**：从下标 0-2 中选出最大的元素即 26 与下标 2 的元素 16 交换

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	16	14	26	27	34	40	46	53’	53	65	74	86

k）**第 11 次**：从下标 0-1 中选出最大的元素 16，发现不需要挪动，而此时就是最后的操作，剩余一个元素

不要再比较。**第 11 次所得序列即为最终的排序序列**。

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	40	46	53	53’	65	74	86	34

3）利用最大堆排序写出在构成初始堆和利用堆排序的过程中，每次过筛(代码 5.1 中的函数 ShiftDown)运算后的排列结果，并画出初始堆所对应的完全二叉树

a）**第 1 次**：初始堆

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	86	74	40	53’	65	34	16	46	53	14	27	26

b）**第 2 次**：第 1 次 ShiftDown，86 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	74	65	40	53’	27	34	16	46	53	14	26	86

c）**第 3 次**：第 2 次 ShiftDown，86、74 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	65	53’	40	53	27	34	16	46	26	14	74	86

d）**第 4 次**：第 3 次 ShiftDown，86、74、65 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	53’	53	40	46	27	34	16	14	26	65	74	86

e）**第 5 次**：第 4 次 ShiftDown，86、74、65、53’已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	53	46	40	26	27	34	16	14	53’	65	74	86

f）**第 6 次**：第 5 次 ShiftDown，86、74、65、53’、53 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11

元素	46	27	40	26	14	34	16	53	53’	65	74	86

g）**第 7 次**：第 6 次 ShiftDown，86、74、65、53’、53、46 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	40	27	34	26	14	16	46	53	53’	65	74	86

h）**第 8 次**：第 7 次 ShiftDown，86、74、65、53’、53、46、40 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	34	27	16	26	14	40	46	53	53’	65	74	86

i）**第 9 次**：第 8 次 ShiftDown，86、74、65、53’、53、46、40、34 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	27	26	16	14	34	40	46	53	53’	65	74	86

j）**第 10 次**：第 9 次 ShiftDown，86、74、65、53’、53、46、40、34、27 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	26	14	16	27	34	40	46	53	53’	65	74	86

k）**第 11 次**：第 10 次 ShiftDown，86、74、65、53’、53、46、40、34、27、26 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	16	14	26	27	34	40	46	53	53’	65	74	86

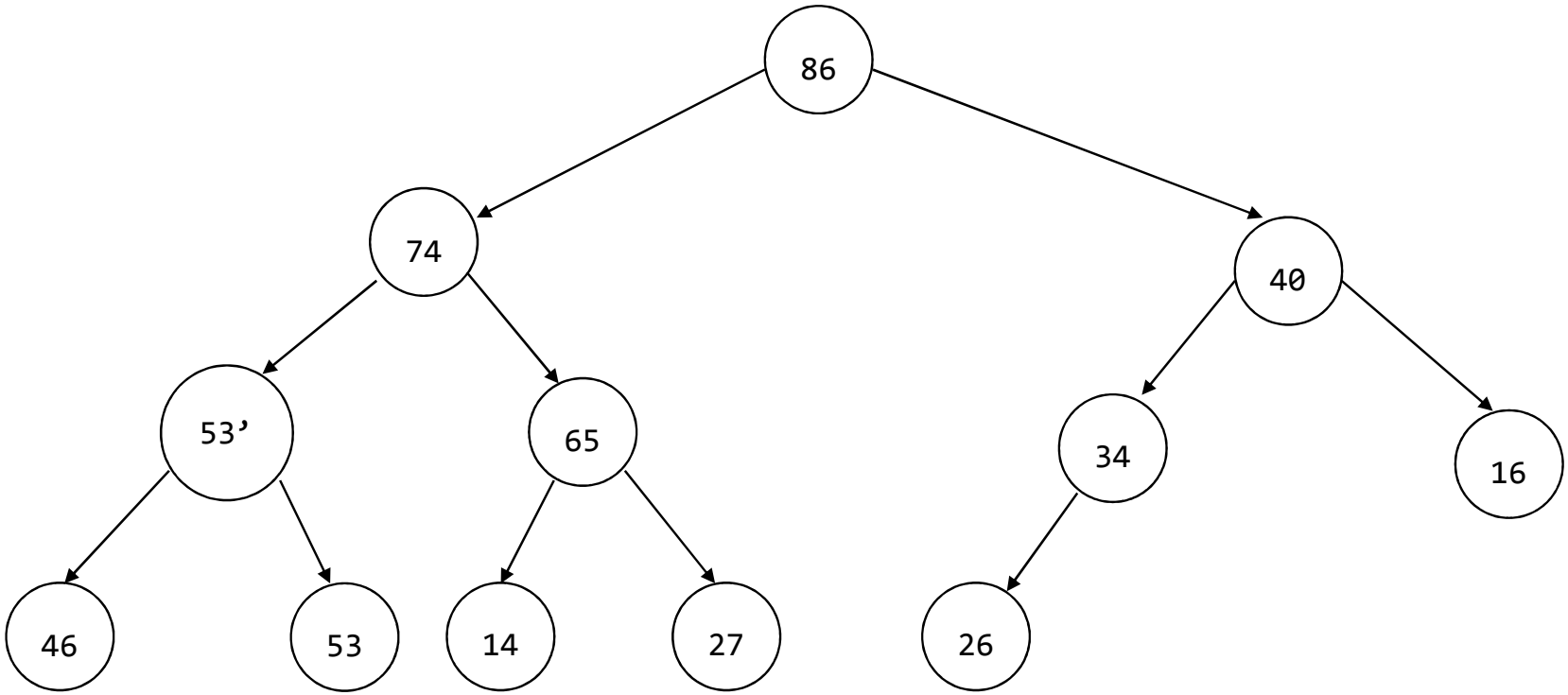
l）**第 12 次**：第 11 次 ShiftDown，86、74、65、53’、53、46、40、34、27、26、16 已经就位

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	34	40	46	53	53’	65	74	86

此时堆排序结束，所得序列就是最终的序列

初始堆对应的二叉树如下：

在下一页


86
74
40
53’
65
34
16
46
53
14
27
26

4）采用快速排序，每次都取子序列的最左元素为轴值，写出每一层划分后的排列结果，并画出由此快速排序

注，**这里的快速排序每次从数组最末端开始**。

a）**第 1 次划分结果**：轴值为 46

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	34	27	16	40	14	26	46	53’	86	65	53	74

b）**第 2 次划分结果**：轴值为 34 和 53’

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	26	27	16	14	34	40	46	53’	86	65	53	74

c）**第 3 次划分结果**：轴值为 26、86，其中 40 由于元素只有一个，因此不需要再动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	34	40	46	53’	74	65	53	86

d）**第 4 次划分结果**：轴值为 14、16、74，其中 27 由于元素只有一个，因此不需要再动

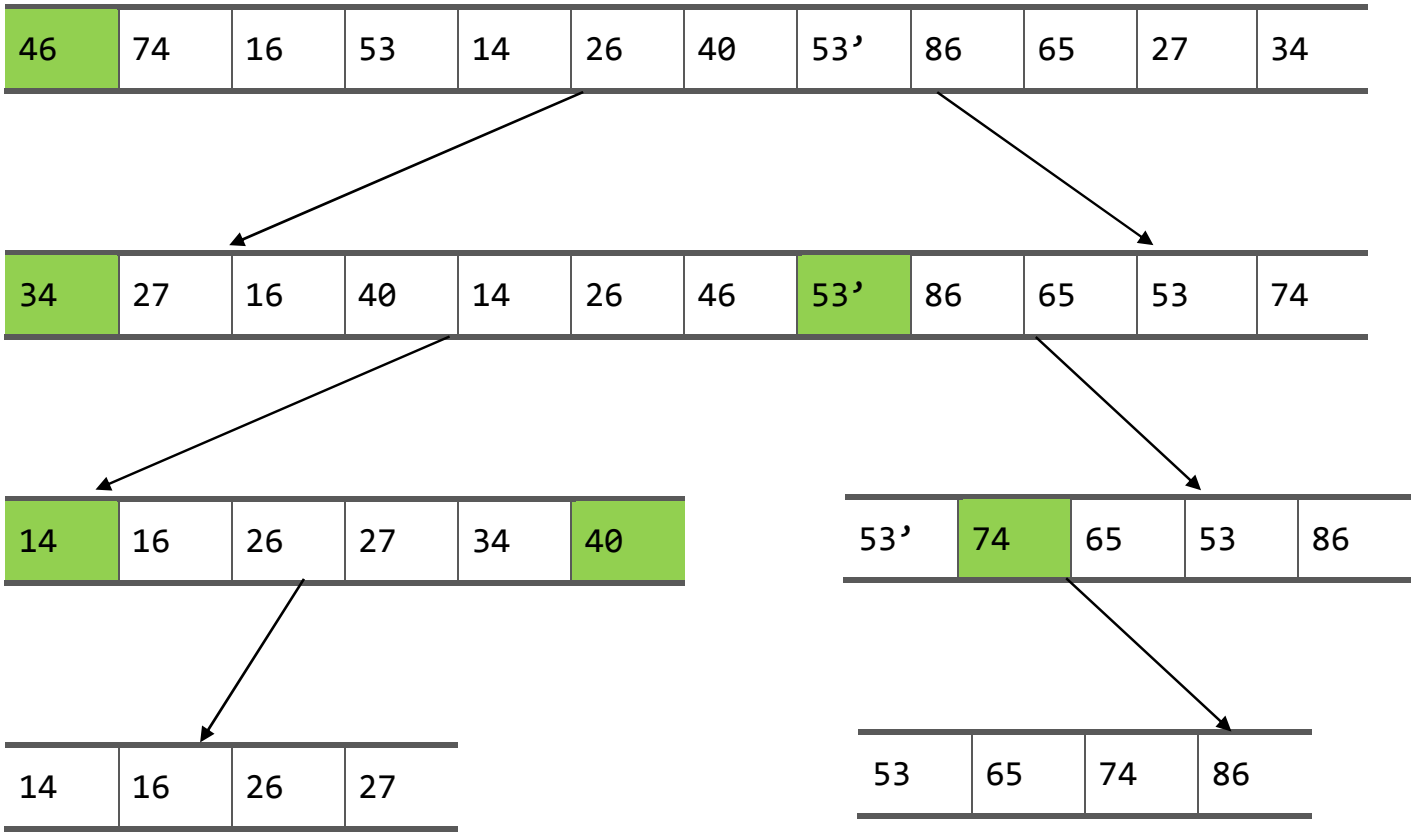
下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	34	40	46	53’	53	65	74	86

e）**第 5 次划分结果**：轴值为 53，由于 16 元素只有一个，因此不需要再动

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	34	40	46	53’	53	65	74	86

到此，快速排序到达最底层，已经全部结束。

二叉搜索树在下页，浅绿色底色为每层的轴值


46 74 16 53 14 26 40 53’ 86 65 27 34
34 27 16 40 14 26 46 53’ 86 65 53 74
14 16 26 27 34 40
53’ 74 65 53 86
14 16 26 27
53 65 74 86

5）利用归并排序的方法写出每一趟二路归并排序后的结果

a）**初始状态**:

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	74	16	53	14	26	40	53’	86	65	27	34

b）**第 1 次归并结果**:

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	46	74	16	53	14	26	40	53’	65	86	27	34

c）**第 2 次归并结果**:

下标	0	1	2	3	4	5	6	7	8	9	10	11

元素	16	46	53	74	14	26	40	53’	27	34	65	86

d）**第 3 次归并结果**：

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	40	46	53	53’	74	27	34	65	86

e）**第 4 次归并结果**：直接合并

下标	0	1	2	3	4	5	6	7	8	9	10	11
元素	14	16	26	27	34	40	46	53’	53	65	74	86