144. Binary Tree Preorder Traversal

Morris traversal

无论是先序还是中序遍历，我们只要做到能判断左子树访问过了，然后访问右子树即可。所以我们先找左子树的最右节点，如果这个节点没有右节点，那么将这个节点的右节点设置为当前的节点。然后访问左子树。如果这个节点最终访问到了当前节点，说明当前节点访问过了，则访问右节点，同时把这个环去掉。如果当前节点没有左子树，则直接访问右子树时间复杂度2n，空间复杂度1

<https://leetcode.com/problems/binary-tree-preorder-traversal/solution/>

<https://zhuanlan.zhihu.com/p/101321696>

<https://blog.csdn.net/wdq347/article/details/8853371>

145. Binary Tree Postorder Traversal

方法1：递归

方法2：迭代。我的实现是用用一个额外指针指向当前节点。如果当前节点也是栈的最后一个，则表示当前节点第一次访问，如果不是，则表示这是上一个要回退的节点，利用它和栈最后一个节点的关系可以判断最后一个节点的访问状态，例如，如果这个节点是右节点的话那么就说明最后一个节点都访问过了，把最后一个节点拉出来即可。时空间复杂度n。同样可实现中序遍历

方法3：先修改前序遍历，使得其为根-右-左，然后把这个顺序反序就是后序遍历

方法4：Morris traversal。比较复杂，暂不讨论

<https://leetcode.com/problems/binary-tree-postorder-traversal/discuss/45559/My-Accepted-code-with-explaination.-Does-anyone-have-a-better-idea>

<https://blog.csdn.net/wdq347/article/details/8853371>

148. Sort List

首先考虑排序算法发现适用链表的有快速排序和归并排序。根据题目要求，归并排序符合。

方法1：自上而下的归并排序。找到中点可以使用快慢指针的方法。时间复杂度是n，所以不会影响总体的时间复杂度。重点在于自上而下的话递归栈会使用logn空间。所以时间复杂度nlogn，空间复杂度logn

方法2：自下而上的归并排序。为了解决空间复杂度的问题，和数组的类似。每次我们从1,2,4,8…n的顺序依次迭代（相当于是步长）。然后把当前size大小的部分每两个合成一个。为了找到上一个，需要一个tail指针指向之前已经合并好的。同时需要一个middle指针指向两个部分的中间，一个end指针指向后面还没合并的部分这样就可以了。每一趟时间复杂度是n，middle指针之类的其实只增不减。一共有logn趟，所以时间复杂度nlogn。空间复杂度1。与数组区别的地方在于一是需要额外的指针记录一些信息，因为链表不能前移，二是每次找middle指针需要时间，但是这不影响总体时间复杂度。

<https://leetcode.com/problems/sort-list/solution/>

<https://leetcode-cn.com/problems/sort-list/solution/leetcode-148-pai-xu-lian-biao-zi-xia-er-o5i24/>

<https://blog.csdn.net/dugudaibo/article/details/79508198>

149. Max Points on a Line

枚举法。把问题转换为过每一个点的每条线上最多有几个点，那就是把剩下的点遍历，看看有几个是在一条线上的。根据数学知识，当我们有了一个点的时候，只需要斜率就可以唯一确定一条直线。所以分为两层遍历，第一层枚举每个点，第二层看看有哪几个点和这个点在一条线上（与这个点形成的斜率一样）。这里需要的注意的是存斜率时如果用浮点数会有精度问题。继续用数学知识，用分数存储斜率，并同时除以最大公约数来最简化分数。最大公约数求法见leetcode3-365题。时间复杂度n^2logn，因为最大公约数求取需要logn时间，空间复杂度n

<https://leetcode.com/problems/max-points-on-a-line/solution/>

150. Evaluate Reverse Polish Notation

用栈，每遇到一个运算符，就把栈头的两个元素运算然后入栈，时间复杂度n，空间复杂度n

python的话利用输入直接做栈可以实现空间复杂度1

<https://leetcode.com/problems/evaluate-reverse-polish-notation/solution/>

<https://blog.csdn.net/qq_45252661/article/details/96433359>

151. Reverse Words in a String

直接split，然后reverse，然后join。O(1)方法见leetcode1-151

<https://leetcode.com/problems/reverse-words-in-a-string/solution/>

152. Maximum Product Subarray

方法1：分析法。见leetcode1-152

方法2：动态规划。注意到负数会使得最大变最小，最小变最大。所以每个状态我们保留两个值，一个是算到（包含）当前的最大值，一个是算到（包含）当前的最小值。然后以最大值为例，n+1的最大值有三种可能，自己，自己乘以之前的最大值或者自己乘以之前的最小值。最小值同理。注意这样整体的最大值并不在最后一个数，而是中间都有可能。同时算法遇到0会自己重置，不需要特殊处理。时间复杂度n，空间复杂度可以优化到1

<https://leetcode.com/problems/maximum-product-subarray/solution/>

153. Find Minimum in Rotated Sorted Array

常规二分法，见leetcode1-153。也可以设计让middle索引最终指向最小值，每次拿它和end索引上的数比较即可，注意每次end不能是middle减一，因为middle有可能提前到达最小值索引。具体见代码

用start和end表示开始和结束的索引时，把这个设置成左闭右开更符合一般规范，也更好处理。

但是二分法除外，用二分法时，需要将end设置为可以取到的，也就是初始值是长度减一。

链表问题，设置一个哨兵node，也就是head的head，这样它永远是head，会使得处理变简单。

动态规划时，对于长度为n的输入，动态规划的数组一般长度为n+1，这样不用判断i-1是不是越界，也会简化代码。