**11. Container With Most Water**

Given *n* non-negative integers *a1*, *a2*, ..., *an*, where each represents a point at coordinate (*i*, *ai*). *n* vertical lines are drawn such that the two endpoints of line *i* is at (*i*, *ai*) and (*i*, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

**Note:**You may not slant the container and *n* is at least 2.



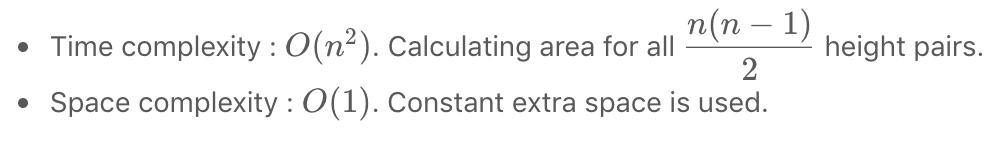
The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

思路：

1. **暴力破解法：**

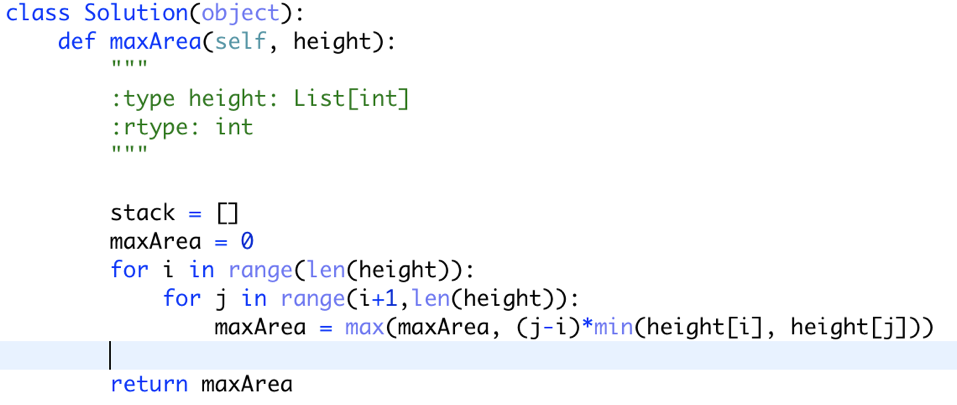
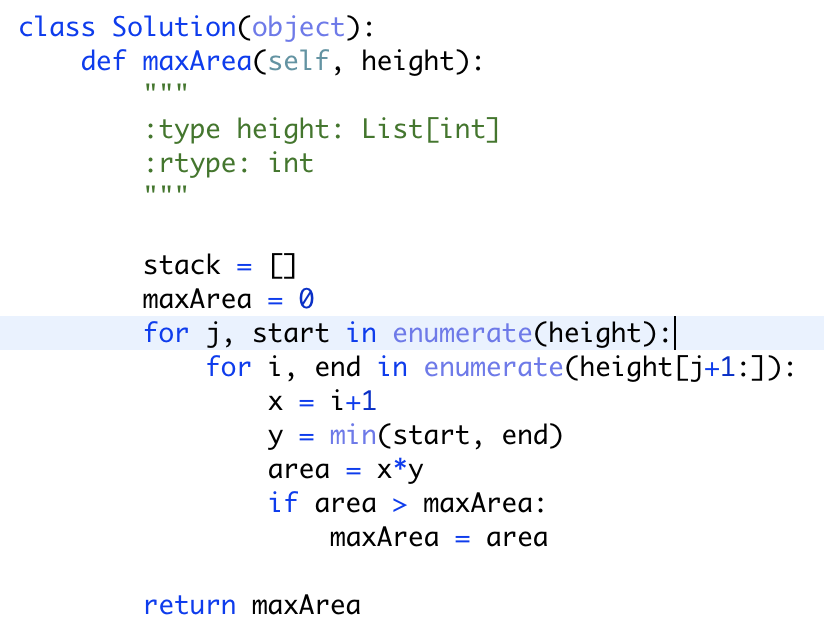
用两层循环遍历，记录面积最大值。

最简单，最容易想到，但是时间复杂度太高

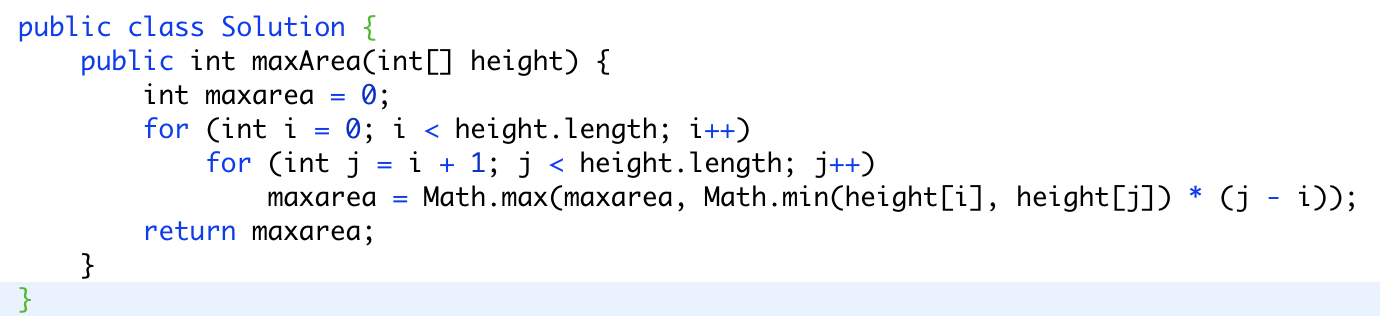


**然后这个暴力方法用python写是会超时不能通过的，但同样的思路改成java就可以了。似乎是因为python本身速度就比较慢。**

不能通过的python：



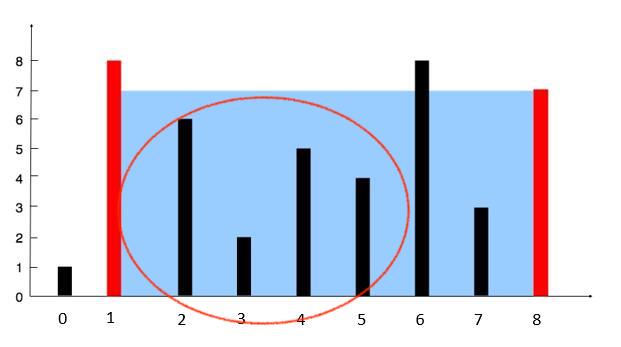
可以通过的java：



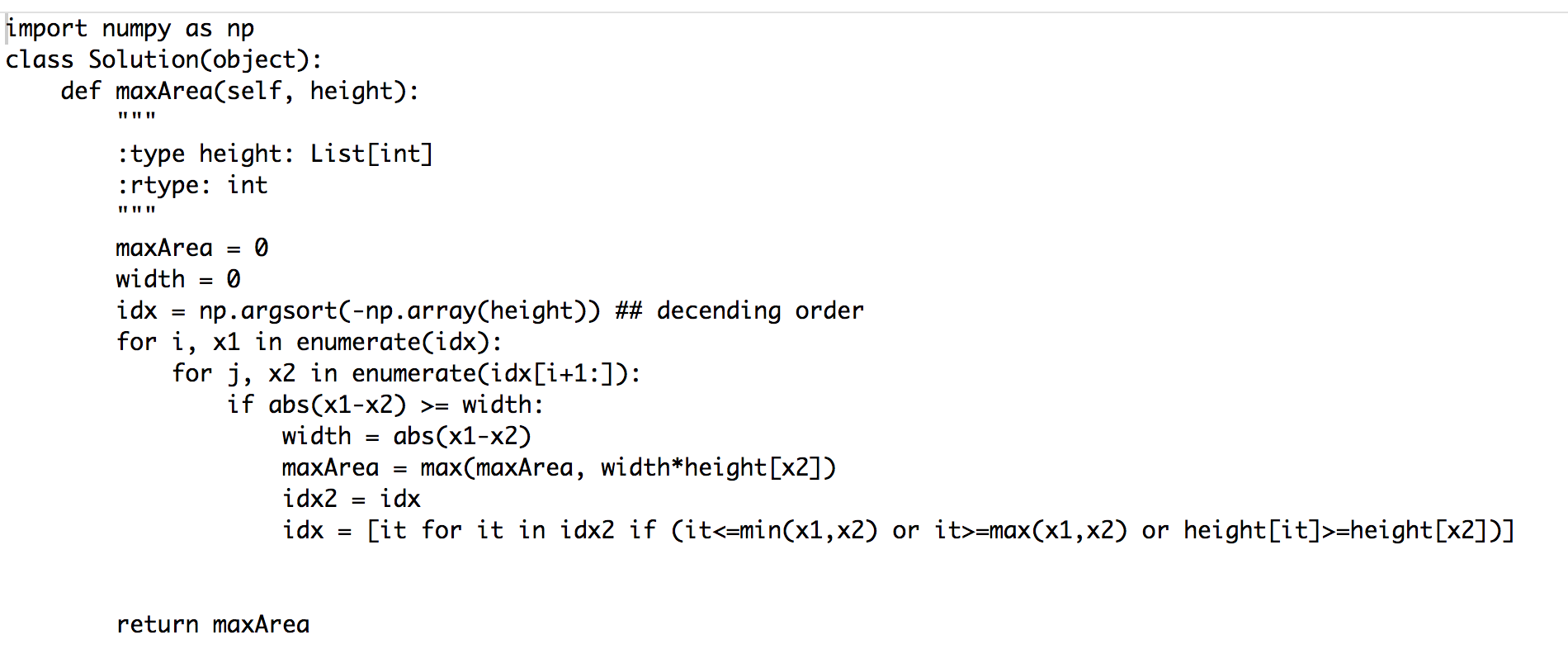
综上，光暴力破解肯定是不行的，我们要优雅的解决这道题。首先，我们知道面积是有两个变量决定的：长度（坐标差）和 高度（最小y值）。那么我们就可以从这两方面下手来减少计算复杂度

1. **高度减小，找到长的（淹没法）：**

我们从自身出发，如果我们遇到这一题，人脑是如何解决的呢？首先肯定先挑选最高的两条线计算面积，然后再依次挑选次高的，且能使长度变宽的线进行面积计算，然后比较大小。从这我们就能发现减少复杂度的方法：就是去掉那些既不能使长度变宽，也不能使高度变高的线。拿下图举例子，当我们挑选1和6时，我们肯定就不会去考虑2，3，4，5这些线了（红圈里），他们也就没有遍历的必要了。而当我们挑选1和8时，7肯定也就没有遍历的必要了。简单来说，就是那些被水淹没的线不用考虑。

****

**这是我自己想到的一个算法，尝试写了这种思路，然后还是发现某些testcase过不了，但我感觉逻辑没有特别大问题。如果有时间大家可以研究一下，附上代码做参考：**

****

1. **长度减小，找到高的（两点法）：**

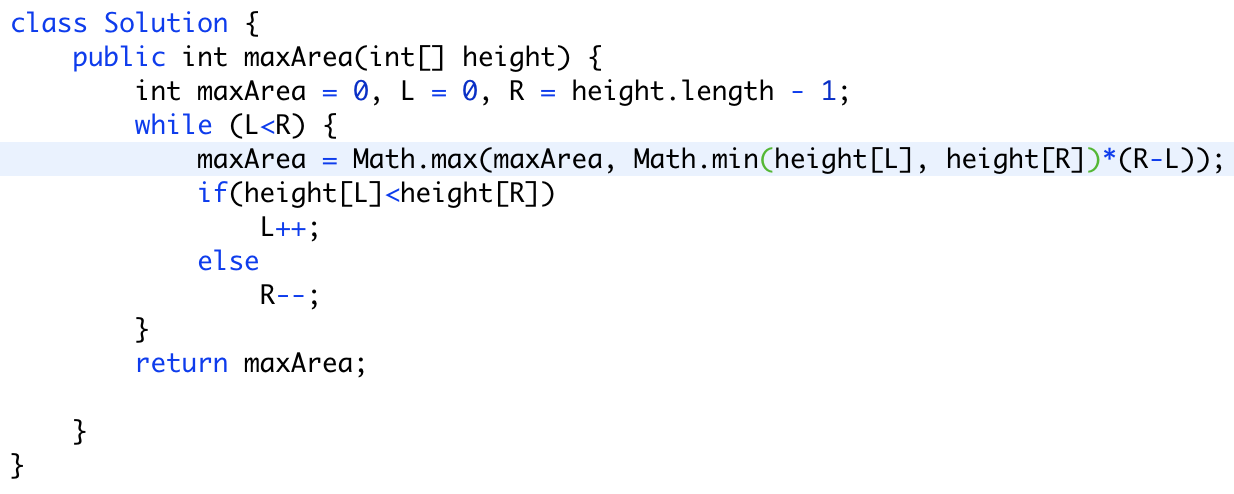
上一个解法是从高度逐渐减小出发，同样我们也可以从长度逐渐减小出发，排除那些矮的。如果选 0 到 8 就保证了长度最长，此时大小是 0 号柱子的高度 1 乘以长度 8 。我们如果想面积更大怎么做呢，只能减小长度，增加高度。是左边的柱子向右移动变成 1 号柱子呢？还是右边的柱子向左移动变成 7 号柱子呢？当然是哪边的柱子短就改哪边的！只有这样，高度才有可能增加。

例如我们如果把 8 号柱子变成 7 号柱子，此时长度减少了，然而高度还是 0 号柱子没有变化，所以面积就会减少。把 1 号柱子变成 2 号柱子就很好了，因为此时高度就变成了 8 号柱子的高度，面积就有可能会增加。

如果左右两边柱子相等该怎么办呢？随意！

我们假设 1 号 和 8 号 柱子高度是相等的。如果他们之间的柱子只有 1 根比它俩高或者没有比它俩高的，那么最大面积就一定选取是 1 号和 8 号了，所以 1 号接着变大，或者 8 号接着减小都是无所谓的，因为答案已经确定了。

java



Python:





事实证明同样写法，java的确比python快

**可以从数列两头分别开始遍历降低复杂度**