



In [7]:

```
1 # 455 饼干
2 def findContentChildren(g, s):
3     # 大饼干喂饱大胃口先
4     g.sort() # 胃口
5     s.sort() # 饼干
6     gpoint = len(g)-1
7     spoint = len(s)-1
8     res = 0
9     while gpoint >=0 and spoint>=0:
10         # 不能while gpoint and spoint 这样会忽略gpoint=0的情况
11         if g[gpoint] <= s[spoint]:
12             res += 1
13             spoint -= 1
14         gpoint -= 1
15     return res
16
17 # 376 摆动序列
18 def wiggleMaxLength(nums):
19     curDiff = 0 # 右-中
20     preDiff = 0 # 左-中
21     res = 1
22     for i in range(len(nums)-1):
23         curDiff = nums[i+1]-nums[i]
24         if curDiff * preDiff <= 0 and curDiff != 0:
25             res += 1
26             preDiff = curDiff
27     return res
28
29 # 53 最大子序和
30 def maxSubArray(nums):
31     res = float('-inf')
32     cur = 0
33     for i in range(len(nums)):
34         cur += nums[i]
35         if cur <= nums[i]: cur = nums[i] # 加进去反而变小了, 重新开始
36         if cur > res: res = cur
37     return res
38
39 # 122 买卖股票II
40 def maxProfit(prices):
41     # 捕获上坡那一段即可
42     res = 0
43     for i in range(1, len(prices)):
44         res += max(prices[i]-prices[i-1], 0)
45     return res
46
47 # 55 跳跃游戏I
48 def canJump(nums):
49     cover = 0
50     for i in range(len(nums)):
51         if i > cover : return False # 出现断层
52         cover = max(nums[i]+i, cover)
53         if cover > len(nums): return True # 早停
54     return True
55
56 # 45 跳跃游戏II
57 def jump(nums):
```

```

58     end = 0
59     cover = 0
60     res = 0
61     for i in range(len(nums)-1):
62         # 记得减1, 实际上, cover会比end快到达终点, end会比i先到达终点
63         cover = max(nums[i]+i, cover)
64         # if end == len(nums)-1: return res
65         if cover >= len(nums)-1: return res+1 # 少走了一步
66         if i == end:
67             end = cover
68             res += 1
69     return res
70
71 # 1005 k次反后最大化数组
72 def largestSumAfterKNegations(nums, k):
73     nums.sort()
74     for i in range(len(nums)):
75         # 处理完所有负数
76         if nums[i] < 0 and k > 0:
77             nums[i] *= -1
78             k -= 1
79         else:
80             break
81     if k > 0:
82         nums.sort()
83         nums[0] *= (-1)**k
84     return sum(nums)
85
86 # 134 加油站
87 def canCompleteCircuit(gas, cost):
88     # 暴利求解 index = (index + 1) % n 适合环形遍历
89     n = len(gas)
90     for i in range(n):
91         res = gas[i]-cost[i]
92         index = (i+1)%n
93         while res > 0 and index != i:
94             res += gas[index]-cost[index]
95             index = (index+1) % n
96         if res >= 0 and index == i:
97             return i
98     return -1
99
100 # 贪心算法
101 if sum(gas) < sum(cost) : return -1 # 不可能跑一圈
102 n = len(gas)
103 curSum = 0
104 start = 0
105 for i in range(n):
106     curSum += gas[i]-cost[i]
107     if curSum < 0:
108         curSum = 0 # 突然不够油了
109         start = i+1 # 重新开始
110     return start
111
112 # 435 无重叠子区间
113 def eraseOverlapIntervals(intervals):
114     n = len(intervals)
115     intervals.sort(key=lambda x:x[1])
116     count = 1 # 记录没有交集的区间数
117     end = intervals[0][1]
118     for i in intervals:

```

```

119         if i[0] >= end:
120             count += 1
121             end = i[1]
122     return n-count # 去掉没有交集的就是有重合的
123
124 # 986 区间列表的交集
125 def intervalIntersection(firstList, secondList):
126     i, j = 0, 0
127     n, m = len(firstList), len(secondList)
128     res = []
129     while i < n and j < m:
130         # 无交集的情况取反 a2 < b1 or a1 > b2
131         if firstList[i][1] >= secondList[j][0] and firstList[i][0] <= secondList[j][1]:
132             res.append([max(firstList[i][0], secondList[j][0]), min(firstList[i][1], secondList[j][1])])
133             if secondList[j][1] > firstList[i][1]: i += 1
134             else: j += 1
135     return res
136
137 # 135 分发糖果
138 def candy(ratings):
139     # 需要一个存放糖果nums
140     n = len(ratings)
141     nums = [1]*n
142     # 从前往后确定i值大于i-1
143     for i in range(1, n):
144         if ratings[i] > ratings[i-1]: nums[i] = nums[i-1]+1 # 比前面的大1
145     # 从后往前确定i-1大于i的值
146     for i in range(n-2, -1, -1):
147         if ratings[i] > ratings[i+1]: nums[i] = max(nums[i], nums[i+1] + 1) # 取两边都满足的最
148     return sum(nums)
149
150 # 860 柠檬水找零
151 def lemonadeChange(bills):
152     five, ten, twenty = 0, 0, 0
153     for bill in bills:
154         if bill == 5: five += 1
155         if bill == 10:
156             if five <= 0: return False
157             ten += 1
158             five -= 1
159         if bill == 20:
160             if ten >= 1 and five > 0:
161                 twenty += 1
162                 ten -= 1
163                 five -= 1
164             elif five >= 3:
165                 twenty += 1
166                 five -= 3
167             else: return False
168     return True
169
170 # 406 身高排序
171 def reconstructQueue(people):
172     # 优先按身高高的people的k来插入，后序插入节点也不会影响前面已经插入的节点
173     queue = []
174     people.sort(key=lambda x: (-x[0], x[1])) # 一维降序，二维升序，（）中谁先排谁
175     for i in people:
176         if i[1] > len(queue): queue.append(i)
177         else:
178             queue.insert(i[1], i)
179     return queue

```

```
180
181 # 452 最少的气球——按end排序
182 def findMinArrowShots(points):
183     # 求区间的交集, [1, 2], [2, 3]不相交但是count += 1
184     # 重叠的情况很多种, 求不重叠情况! start > end
185     points.sort(key=lambda x:x[1])
186     count = 1
187     end = points[0][1]
188     for i in points:
189         start = i[0]
190         if start > end :
191             count += 1
192             end = i[1]
193     return count
194
195 # 763 划分字母区间
196 def partitionLabels(s):
197     curfarther = 0
198     s = list(s)
199     res = []
200     j = -1
201     dic = {s[i]:s.index(s[i],i) for i in range(len(s))} # 记录最远的位置
202     for i in range(len(s)):
203         curfarther = max(curfarther, dic[s[i]])
204         if i == curfarther:
205             res.append(i-j)
206             j = i
207     return res
208
209 # 56 区间合并——按start排序
210 def merge(intervals):
211     # 融合区间
212     intervals.sort(key=lambda x:x[0])
213     res = []
214     res.append(intervals[0])
215     for interval in intervals:
216         start = interval[0]
217         if start <= res[-1][1]: # 衡量标准, 最大end
218             res[-1][1] = max(interval[1], res[-1][1])
219         else:
220             res.append(interval)
221     return res
222
223 # 738 单调递增数字
224 def monotoneIncreasingDigits(n):
225     # 单调递增的数字
226     # 从后向前遍历
227     nums = [int(i) for i in str(n)]
228     length = len(nums)
229     for i in range(length-2, -1, -1):
230         if nums[i] > nums[i+1]:
231             nums[i] -= 1
232             nums[i+1:] = '9'*len(nums[i+1:]) # 后面赋值为9
233     nums = [str(i) for i in nums]
234     return int(''.join(nums))
235
236 # 968 监控二叉树
237 def minCameraCover(root):
238     # 0 表示无摄像头+有覆盖
239     # 1 表示摄像头
240     # 2 表示有覆盖
```

```
241 self.res = 0
242 def traversal(cur):
243     if not cur: return 2 # 标记为2
244     left = traversal(cur.left)
245     right = traversal(cur.right)
246     if left == 2 and right == 2:
247         # 叶子节点,左右节点都覆盖,那么这个应该什么都没有或者是叶子
248         return 0
249     elif left == 0 or right == 0:
250         # 有一个孩子没有覆盖,父节点就应该放摄像头
251         self.res += 1
252         return 1
253     elif left == 1 or right == 1:
254         # 有一个孩子有摄像头了,父节点就应该被覆盖了
255         return 2
256     else: return
257 if traversal(root) == 0: self.res += 1 # 这里已经进行了方法修改,特殊情况
258 return self.res
```