内功修炼

- 在栈中实现函数找到栈中的最小值

```python
class Stack:
    def __init__(self):
        self.min = []
        self.stack = []

    def add(self, x):
        self.stack.append(x)
        if len(self.min) == 0 or x < self.stack[self.min[-1]]:
            self.min.append(len(self.stack) - 1)

    def top(self):
        if len(self.stack) == 0:
            raise Exception("栈为空")
        else:
            return self.stack[-1]

    def pop(self):
        if len(self.stack) == 0:
            raise Exception("栈为空")
        else:
            a = self.stack.pop()
            if len(self.stack) == self.min[-1]:
                self.min.pop()
            return a

    def min(self):
        if len(self.min) == 0:
            raise Exception("栈为空")
        else:
            return self.stack[self.min[-1]]
```

  - 用空间换时间，引入一个记录最小值**索引**的min_stack，并维护这个栈
  - 增加的时候，如果值比栈中的数还小，那么就加入这个值在主栈中的索引
  - 删除的时候，如果这个值的索引和min_stack中的最小值的索引相等，则pop出min_stack中的栈顶元素
  - 如果操作的时候遇到空栈，就需要返回一个异常
- 三数之和

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()
        for i in range(len(nums) - 2):
            if nums[i] > 0: return res  # !!!
            if i > 0 and nums[i] == nums[i - 1]: continue  # !!!
            l, r = i + 1, len(nums) - 1
            target = -nums[i]
            while l < r:
                sum = nums[l] + nums[r]
                if sum == target:
                    res.append([nums[i], nums[l], nums[r]])
                    l += 1
                    r -= 1
                    while l < r and nums[l] == nums[l - 1]: l += 1
                    while l < r and nums[r] == nums[r + 1]: r -= 1
                elif sum < target:
                    l += 1
                else:
                    r -= 1
        return res
```

- 优化速度，对于第一个数字i，遇到重复的数字，第一次要先做，第二次才能略过，即nums[i] ==nums[i-1] 而不是nums[i] ==nums[i+1]，不然会导致漏可能性
- 尽量不要在if判断的时候，采取加减法，效率很慢

- 柱状图中最大的矩形

```python
class Solution:
    def largestRectangleArea(self, heights):
        heights.append(0)
        stack = [-1]
        maxarea = 0
        for i in range(len(heights)):
            while heights[i]<heights[stack[-1]]:
                h = heights[stack.pop()]
                w = i - stack[-1] -1
                maxarea = max(maxarea,h*w)
            stack.append(i)
        return maxarea
```

- 用了一个哨兵思想，在stack开头放了一个-1 在heights结尾放了一个0
- stack中有一个-1的作用，在单调栈中找前一个小于自己的索引时，可以完美的处理边界问题
- 结尾放0，在最后的时候，用0来表示栈中元素的右边界，可以清空栈，直到剩下最后的-1

- 设计循环双端队列
  - head端是先放值，再++，tail端是先--，再放值
  - 要多设计一位容积k+1，用来区分是空还是满队列

```python
self.head,self.tail = 0,0
self.capacity = k+1
self.arr = [0 for _ in range(k+1)]
```