33. Search in Rotated Sorted Array

almost one pass

```
class Solution:
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """

        l = 0
        right = len(nums) - 1

        while l <= right :
            mid = (l+right)//2

            if nums[mid] >= nums[l] and target >= nums[l]:
                num_mid = nums[mid]
            elif nums[mid] < nums[l] and target < nums[l]:
                num_mid = nums[mid]
            elif target < nums[l]:
                num_mid = -9999
            else:
                num_mid = 9999

            if target == num_mid:
                return mid
            elif target > num_mid:
                l = mid+1
            else:
                right = mid
        return -1
```

35. Search Insert Position

almost one pass

```
class Solution:
    def searchInsert(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """

        left = 0
        right = len(nums)

        while left < right:

            mid = (left + right ) //2
            if nums[mid] < target:
                left = mid +1
            else:
                right = mid
        return left
```

---

74. Search a 2D Matrix

almost one pass:

```
class Solution:
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """

        if matrix is None or len(matrix) == 0 or len(matrix[0]) == 0:
            return False

        num_row = len(matrix)
        num_col = len(matrix[0])

        i = 0
        j = num_col - 1

        while i < num_row and j >= 0:
            if matrix[i][j] > target:
                j -= 1
            elif matrix[i][j] < target:
                i += 1
            else:
                return True

        return False
```

75. Sort Colors

a new swap method in python

```
nums[p1], nums[p2] = nums[p2], nums[p1]
```

```
class Solution:
    def sortColors(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-place instead.
        """

        p0 = 0
        p1 = 0
        p2 = len(nums) - 1

        while p1 <= p2:
            if nums[p1] == 0:
                # swap with p0
                nums[p0], nums[p1] = nums[p1], nums[p0]
                p0 +=1
                p1 +=1
            elif nums[p1] == 1:
                p1+=1
            else:
                nums[p1], nums[p2] = nums[p2], nums[p1]
                p2 -= 1
```

---

153. Find Minimum in Rotated Sorted Array

almost one pass

```
class Solution:
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        l = 0
        r = len(nums)-1

        while l < r:
            if nums[l] < nums[r]:
                return nums[l]

            mid = (l+r) //2

            if nums[mid] >= nums[l]:
                l = mid+1
            else:
                r = mid

        return nums[l]
```

209. Minimum Size Subarray Sum

```
class Solution:
    def minSubArrayLen(self, s, nums):
        """
        :type s: int
        :type nums: List[int]
        :rtype: int
        """
        start = 0
        end = 0
        curSum = 0
        res = 9999

        for end in range(len(nums)):
            curSum += nums[end]
            while curSum >= s:
                res = min(res, end-start + 1)
                curSum -= nums[start]
                start+=1

        if res == 9999:
            return 0
        return res
```

560. Subarray Sum Equals K

one pass

611. Valid Triangle Number

time exceed, so there must be a faster solution.

```
class Solution:
    def triangleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """

        c = 0
        n = len(nums)
        nums.sort()

        for i in range(n-1, 1, -1):
            lo = 0
            hi = i - 1
            while lo < hi:
                if nums[hi] + nums[lo] > nums[i]:
                    c += hi-lo# since lo can be replaced by all the number between t
hem, for this certain hi,
                    hi -= 1
                else:
                    lo +=1
        return c
```

```
class Solution:
    def searchInsert(self, nums, target, i_start):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """

        left = i_start
        right = len(nums)

        while left < right:

            mid = (left + right ) //2
            if nums[mid] < target:
                left = mid +1
            else:
                right = mid
        return left


    def triangleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        nums.sort()
        res = 0
        for i in range(len(nums) - 2):
            for j in range(i+1, len(nums)-1):

                k = self.searchInsert(nums, nums[i] + nums[j] - 0.5, j+1)
                if k == j+1:
                    continue
                else:
                    res += (k-j-1)
        return res
```

713. Subarray Product Less Than K

one stupid idea is to iterate every combination and test if it's okay.

sometimes we can skip some combinations. or we can start from 0 and put each number as the first number of the subarray.

```
class Solution:
    def numSubarrayProductLessThanK(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int"""
        product = 1
        j = -1
        count = 0

        for i, n in enumerate(nums):
            product *= n
            while product >= k and j < i:
                j += 1
                product //= nums[j]
            count += i - j
        return count
```