Binary Search Practice:

- Lintcode 462. Total Occurrence of Target
  Hint: 1. Two times Binary Search for the first and the last positions of target
  2. What happen if the array is empty
  3. What if you can not find the position in that array
- Lincode 459. Closest Number in Sorted Array
  Hint: 1. Find the first number >= target
  2. Compare this number with the last one

1.三步翻转法
Mainly focus on how to rotate an array in O(n) time complexity and O(1) space complexity.

Practice:

- Lintcode 39. Recover Rotated Sorted Array

```
public class Solution {
    /**
     * @param nums: An integer array
     * @return: nothing
     */
    public void recoverRotatedSortedArray(List<Integer> nums) {
        // write your code here
        for(int i = 0; i < nums.size() - 1; i++){
            if(nums.get(i) > nums.get(i + 1)){
                reverse(nums, 0, i);
                reverse(nums, i + 1, nums.size() - 1);
                reverse(nums, 0, nums.size() - 1);
            }
        }
    }

    private void reverse(List<Integer> nums, int start, int end){
        for (int i = start, j = end; i < j; i++, j--) {
            int temp = nums.get(i);
            nums.set(i, nums.get(j));
            nums.set(j, temp);
        }
    }
}
```

- Lintcode 8. Rotate String

2.二维矩阵查找问题(更像智力问题)
从左下角或者右上角开始搜索

3.快速幂算法

Recursion

```
int power(int x, int n) {
    if (n == 0) return 1;
    if (n % 2 == 0) {
        int tmp = power(x, n / 2);
        return tmp * tmp;
    } else {
        int tmp = power(x, n / 2);
        return tmp * tmp * x;
    }
}
```

No Recursion

```
int power(int x, int n) {
    int ans = 1, base = x;
    while (n != 0) {
        if (n % 2 == 1) {
            ans *= base;
        }
        base *= base;
        n = n / 2;
    }
    return ans;
}
```

Section 3. Two Pointers

1.双向双指针
e.g. 三步翻转法

经典问题：

- 判断回文串
  Follow up 1: 不区分大小写，忽略非英文字母
  Follow up 2: 允许删掉一个字母（类似的，允许插入一个字母）

1. 依然用相向双指针的方式从两头出发，两根指针设为 L 和 R。
2. 如果 s[L] 和 s[R] 相同的话，L++, R--
3. 如果 s[L] 和 s[R] 不同的话，停下来，此时可以证明，如果能够通过删除一个字符使得整个字符串变成回文串的话，那么一定要么是 s[L]，要么是 s[R]。

- Two Sum
  1. 使用HashSet 记录diff T: O(n) S:O(n)
  2. 排序，双指针 T:O(nlgn) S:O(1)

```
public class Solution {
    public int[] twoSum(int[] numbers, int target) {
        Arrays.sort(numbers);

        int L = 0, R = numbers.length - 1;
        while (L < R) {
            if (numbers[L] + numbers[R] == target) {
                int[] pair = new int[2];
                pair[0] = numbers[L];
                pair[1] = numbers[R];
                return pair;
            }
            if (numbers[L] + numbers[R] < target) {
                L++;
            } else {
                R--;
            }
        }
        return null;
    }
}
```

2.同向双指针

1. 数组去重问题 Remove duplicates in an array
2. 滑动窗口问题 Window Sum
3. 两数之差问题 Two Difference
4. 链表中点问题 Middle of Linked List
5. 带环链表问题 Linked List Cycle