```java
public int removeDuplicates(int[] nums) {
    if(nums == null || nums.length == 0) return 0;
    int p = 0;
    int q = 1;
    while (q < nums.length) {
        if (nums[p] != nums[q]) {
            nums[p + 1] = nums[q];
            ++ p;
        }
        ++ q;
    }
    return p + 1;
}
```

Remove Duplicates from Sorted Array

因为是已排序数组 所以 当前元素 比 前一位元素 不一样 则代表不重复。否则代表重复

双下标 p = 0, q = 1 从第二位元素开始遍历
a[p + 1] = a[q] when a[q] != a[p], then ++p (update p)

```java
//  一维数组的坐标转换
//  1.枚举: right bar x, left bar y, (x-y)*height_diff
//  O(n^2)
//  2. O(n), 左右边界 i, j , 向中间收敛, 谁高度更小谁就往里面挪, 去看里面还有没有更高的棒子
class Solution {
    public int maxArea(int[] a) {
        int max = 0;
        for (int i = 0, j = a.length - 1; i < j; ) {
            int minHeight = a[i] < a[j] ? a[i++] : a[j--];
            max = Math.max(max, (j - i + 1) * minHeight);
        }
        return max;
    }
}
```

Two Sum II - Input array is sorted

双指针法 — 左右收敛法:
O(n) 左右边界 i, j , 向中间收敛, 谁高度小谁就往里挪，去看里面还有没有更高的棒子

装水最多的水桶

Intersection of two arrays

使用 HashSet, 当 add 失败 则这个元素曾经出现过

检测两数组交集:
双 HashSet, 第一个 hs 存放数组1
遍历数组2时, 如果 hs1 包含了此元素的元素，则此元素为交集元素，插入 hs2
最后 hs2 转数组

Two Sum

使用 hashMap 缓存曾经遍历过的数字
Key = int, Value = index

If map.containsKey(target - current)
则代表另外一个数字已被找到

Three Sum

- 先排序
- 再遍历数组 从 0 - 到倒数第三位
- 双指针 从当前数字的右边一位，跟数组最右边开始往两边会和，来寻找这三个指针的和是否为 0

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res = new ArrayList<>();
        for(int i = 0; i < nums.length - 2; i++) {
            if(nums[i] > 0) break;
            if(i > 0 && nums[i] == nums[i - 1]) continue;
            int j = i + 1, k = nums.length - 1;
            while(j < k){
                int sum = nums[i] + nums[j] + nums[k];
                if(sum < 0)
                    while(j < k && nums[j] == nums[++j]);
                else if (sum > 0)
                    while(j < k && nums[k] == nums[--k]);
                else {
                    res.add(new ArrayList<Integer>(Arrays.asList(nums[i], nums[j], nums[k])));
                    while(j < k && nums[j] == nums[++j]);
                    while(j < k && nums[k] == nums[--k]);
                }
            }
        }
        return res;
    }
}
```

移动0到数组最后

```java
// 1.记录非0数插入点, 遇到非0的数就互换位置,
// 将原位置设置为0(需要非0数插入点不等于当前index), 递增非0数插入点
class Solution {
    public void moveZeroes(int[] a) {
        int j = 0;
        for (int i = 0; i < a.length; ++i) {
            if (a[i] != 0) {
                a[j] = a[i];
                if (i != j) {
                    a[i] = 0;
                }
                j++;
            }
        }
    }
}
```

移动0, 一个变量记录非0下标, 遇到非0则交换, 默认 j 跟着 i 走, 一旦遇到 0, j 就会停在 0 所在的索引, 一旦遇到非 0 则会互换位置, 然后 ++j

```java
//  背下来
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        while(n>0) nums1[m+n-1] = (m==0 ||nums2[n-1] > nums1[m-1]) ? nums2[--n] : nums1[--m];
    }
}
```

Merge Sorted Array

此题是 merge sort 算法最后一个步骤，合并左右两侧已合并数组

数组操作核心

操作快慢指针

双下标数据互换

双下标默认同步, 遇到条件则其中一个下标停留

双下标左右相遇可以翻转数组

```java
// 翻转三次
// 1.翻转全部
// -> 7,6,5,4,3,2,1
// 2.翻转 0 到 k-1
// -> 5,6,7,4,3,2,1
// 3.翻转 k 到 arr.len
public class Solution {
    public void rotate(int[] nums, int k) {
        k %= nums.length; //  分水岭 mod 后, 即使 k > nums.length 也会让 k 为 0
        reverse(nums, 0, nums.length-1);
        reverse(nums, 0, k-1);
        reverse(nums, k, nums.length-1);
    }

    public void reverse(int[] nums, int start, int end) {
        while (start < end) {
            int temp = nums[start];
            nums[start] = nums[end];
            nums[end] = temp;
            ++ start;
            -- end;
        }
    }
}
```

原数组的下标放入 (i+k) % arr.len

```java
//1. 新开辟一个数组
//  遍历 nums, a[i+k % arr.length] = nums[i]
//  a[i+k % arr.length] 相当于从 i=0, 往后挪动 k 位
//  % mod 是个循环, (i + k) 是个offset, offset 等于 arr.length 时会归零, 继续递增
//  当遇到需要被挪到前面的元素时, 元素的 i + k 刚好等于 arr.length; % 的结果 0
//  所以从不需要前挪的 arr.length-1 因为 % 后为 0, 1, ... k
//  从排好序的数组中读取数据重写原数组
public class Solution {
    public void rotate(int[] nums, int k) {
        int[] a = new int[nums.length];
        for (int i = 0; i < nums.length; ++i) {
            a[(i+k) % nums.length] = nums[i];
        }
        for (int j = 0; j < a.length; ++j) {
            nums[j] = a[j];
        }
    }
}
```

Rotate Array

O(k*n)

```java
//1. 暴力
//每次挪动最后一位到最前面, 挪动 k 次
//需要保存待交换的原位置, 并作为下一次的 previous
public class Solution {
    public void rotate(int[] nums, int k) {
        int temp, previous;
        for (int i = 0; i < k; ++i) {
            previous = nums[nums.length - 1];
            for (int j = 0; j < nums.length; ++j) {
                temp = nums[j];
                nums[j] = previous;
                previous = temp;
            }
        }
    }
}
```

Plus One

```java
//  从尾至头检查+1是否小于 10, 小于则立即 return
//  否则当前索引设置0, --当前索引
//  如果循环完了数组, 并且没有在循环中 return, 说明当前数组项都为 0
//  返回一个新开辟的数组, 数组第一位设为 1
class Solution {
    public int[] plusOne(int[] digits) {
        for (int i = digits.length - 1; i >= 0; --i) {
            if (digits[i] + 1 <= 9) {
                ++ digits[i];
                return digits;
            }
            digits[i] = 0;
        }
        int[] newDigits = new int[digits.length + 1];
        newDigits[0] = 1;
        return newDigits;
    }
}
```

Matrix

思路:
左到右 -> 上到下 -> 右到左 -> 下到上
每次走一个方向后 将这个方向移除边界条件:
rowS <= rowE && colS <= colE

左到右 celS <= celE
上到下 rowS <= rowE
右到左 rowS <= row E, colS <= colE
下到上 colS <= colE, rowS <= rowE

```java
int rowS = 0;
int rowE = matrix.length - 1;
int colS = 0;
int colE = matrix[0].length - 1;
while (rowS <= rowE && colS <= colE) {
    //  travel right
    for (int j = colS; j <= colE; ++j) result.add(matrix[rowS][j]);
    ++ rowS;

    //  travel bottom
    for (int j = rowS; j <= rowE; ++j) result.add(matrix[j][colE]);
    -- colE;

    //  travel left
    if (rowS <= rowE) {
        for (int j = colE; j >= colS; --j) result.add(matrix[rowE][j]);
    }
    -- rowE;

    //  travel top
    if (colS <= colE) {
        for (int j = rowE; j >= rowS; --j) result.add(matrix[j][colS]);
    }
    ++ colS;
}
```