# 初级

## 数组

### 删除排序数组中的重复项

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x2gy9m/>

**解法一**：双指针

public int removeDuplicates(int[] nums) {

int n = nums.length;

if (n == 0) {

return 0;

}

int fast = 1, slow = 1;

while (fast < n) {

if (nums[fast] != nums[fast - 1]) {

nums[slow] = nums[fast];

++slow;

}

++fast;

}

return slow;

}

### 买卖股票的最佳时机 II

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x2zsx1/>

**解法1**：动态规划

/\*\*

\* dp[i][0]表示没买或卖了，即手里没股票的利润

\* dp[i][1]表示手里持有股票的利润

\* @param prices

\* @return

\*/

public int maxProfit(int[] prices) {

int n = prices.length;

int[][] dp = new int[n][2];

// 初始化，第一天如果没有买股票，利润为0

dp[0][0] = 0;

// 初始化，第一天如果买了股票，利润为第一天股票价格的相反数

dp[0][1] = -prices[0];

for (int i = 1; i < n; ++i) {

// 手里没股票的利润(前一天就没有或前一天有，但是今天卖了)

dp[i][0] = Math.max(dp[i - 1][0], dp[i - 1][1] + prices[i]);

// 手里有股票的利润(前一天就有，今天没操作或前一天没有，今天买了)

dp[i][1] = Math.max(dp[i - 1][1], dp[i - 1][0] - prices[i]);

}

// 最后一天手里没股票的最终利润

return dp[n - 1][0];

}

**解法2**：动态规划

解法1的优化代码：

public int maxProfit(int[] prices) {

int n = prices.length;

int dp0 = 0, dp1 = -prices[0];

for (int i = 1; i < n; ++i) {

// 手里没股票的利润(前一天就没有或前一天有，但是今天卖了)

int newDp0 = Math.max(dp0, dp1 + prices[i]);

// 手里有股票的利润(前一天就有，今天没操作或前一天没有，今天买了)

int newDp1 = Math.max(dp1, dp0 - prices[i]);

dp0 = newDp0;

dp1 = newDp1;

}

return dp0;

}

**解法3**：贪心算法

public int maxProfit(int[] prices) {

int ans = 0;

int n = prices.length;

for (int i = 1; i < n; ++i) {

ans += Math.max(0, prices[i] - prices[i - 1]);

}

return ans;

}

### 旋转数组

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x2skh7/>

**解法1**：临时数组

public void rotate(int[] nums, int k) {

int length = nums.length;

int temp[] = new int[length];

for (int i = 0; i < length; i++) {

temp[(i + k) % length] = nums[i];

}

System.arraycopy(temp, 0, nums, 0, length);

}

**解法3：**数组反转

public void rotate(int[] nums, int k) {

k %= nums.length;

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 += 1;

end -= 1;

}

}

### 存在重复元素

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x248f5/>

**解法1**：排序

public boolean containsDuplicate(int[] nums) {

Arrays.sort(nums);

int n = nums.length;

for (int i = 0; i < n - 1; i++) {

if (nums[i] == nums[i + 1]) {

return true;

}

}

return false;

}

**解法2**：哈希表

public boolean containsDuplicate(int[] nums) {

Set<Integer> set = new HashSet<Integer>();

for (int x : nums) {

if (!set.add(x)) {

return true;

}

}

return false;

}

### 只出现一次的数字

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x21ib6/>

解法1：位运算

public int singleNumber(int[] nums) {

int single = 0;

for (int num : nums) {

single ^= num;

}

return single;

}

### 两个数组的交集 II

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x2y0c2/>

**解法1**：哈希表

public int[] intersect(int[] nums1, int[] nums2) {

if (nums1.length > nums2.length) {

return intersect(nums2, nums1);

}

Map<Integer, Integer> map = new HashMap<Integer, Integer>();

for (int num : nums1) {

int count = map.getOrDefault(num, 0) + 1;

map.put(num, count);

}

int[] intersection = new int[nums1.length];

int index = 0;

for (int num : nums2) {

int count = map.getOrDefault(num, 0);

if (count > 0) {

// 放入交集元素

intersection[index++] = num;

count--;

if (count > 0) {

// 同一元素，出现次数可能nums1多，可能nums2多。如果count=0,说明nums2中出现的次数多，

// 则移除该元素，因为该交集元素的数量不会增加了，取的是较少的次数。如果count>0，则需要继续往后遍历

// nums2，直到遍历结束

map.put(num, count);

} else {

map.remove(num);

}

}

}

return Arrays.copyOfRange(intersection, 0, index);

}

**解法2**：排序+双指针

public int[] intersect(int[] nums1, int[] nums2) {

Arrays.sort(nums1);

Arrays.sort(nums2);

int length1 = nums1.length, length2 = nums2.length, left = 0, right = 0, index = 0;

if (length1 > length2) {

intersect(nums2, nums1);

}

int[] intersect = new int[length1];

while (left < length1 && right < length2) {

if (nums1[left] < nums2[right]) {

left++;

} else if (nums1[left] > nums2[right]) {

right++;

} else {

intersect[index] = nums1[left];

left++;

right++;

index++;

}

}

return Arrays.copyOfRange(intersect, 0, index);

}

### 加一

题目：<https://leetcode.cn/leetbook/read/top-interview-questions-easy/x2cv1c/>

**解法1**：找出最长的后缀 99

public int[] plusOne(int[] digits) {

int n = digits.length;

for (int i = n - 1; i >= 0; --i) {

if (digits[i] != 9) {

// 找到不为 9 的数字， +1

++digits[i];

for (int j = i + 1; j < n; ++j) {

// 非9 之后的数字都置为 0

digits[j] = 0;

}

return digits;

}

}

// digits 中所有的元素均为 9

int[] ans = new int[n + 1];

ans[0] = 1;

return ans;

}