

Chapter 9 High Frequency

Count Complete Tree Nodes

Given a complete binary tree, count the number of nodes.

Definition of a complete binary tree from Wikipedia:

In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h .

Answer:

/*

满二叉树(Full Binary Tree):

除最后一层无任何子节点外, 每一层上的所有节点都有两个子节点(最后一层上的无子节点的节点为叶子节点)

若深度为 h ,

第 h 层的结点数 = $2^{(h-1)}$

总结点数 = $2^h - 1$

完全二叉树(Complete Binary Tree):

若设二叉树的深度为 h , 除第 h 层外, 其它各层($1 \sim h-1$)的结点数都达到最大个数, 第 h 层所有的结点都连续集中在最左边

总结点数 $[2^{(h-1)}, 2^h - 1]$

e.g. 一棵完全二叉树有770个结点, 则它的叶子结点是259个

*/

/*

SOL 1

找到最底层, 然后二分搜索, 找到第一个空节点, 效率是 $O(\log n * \log n)$, 其中 n 是节点总数

$(1 << (dep - 1)) - 1 + \text{start means}$ 倒数第二层以上的满二叉树总结点数 + 末层的叶子结点数

SOL 2

判断左子树最右结点的深度和右子树最右结点的深度,

如果相等, 右子树必为满二叉树, 直接公式算出右子树节点数, 递归左子树

如果不等, 左子树必为满二叉树, 直接公式算出左子树节点数, 递归右子树

time $O(h^2)$

Let n be the total number of the tree. It is likely that you will get a child tree as a perfect binary tree and a non-perfect binary tree ($T(n/2)$) at each level.

$T(n) = T(n/2) + c_1 \log n$

$= T(n/4) + c_1 \log n + c_2 (\log n - 1)$

$= \dots$

$= T(1) + c [\log n + (\log n - 1) + (\log n - 2) + \dots + 1]$

$= O(\log n * \log n)$

*/

```
public class CompleteTreeSol {
    public static int countNodes1(TreeNode root) {
        if (root == null) {
            return 0;
        }
        if (root.left == null && root.right == null) {
            return 1;
        }
    }
}
```

TreeNode pos = root;

```
        int dep = 0;
        while (pos != null) {
            pos = pos.left;
            dep++;
        }

        int start = 1;
        int end = (1 << (dep - 1));
        if (isFull(root, dep, end)) {
            return (1 << dep) - 1;
        }

        while (start + 1 < end) {
            int mid = start + (end - start) / 2;
            if (isFull(root, dep, mid)) {
                start = mid;
            } else {
                end = mid;
            }
        }
        return (1 << (dep - 1)) - 1 + start;
    }

private static boolean isFull(TreeNode root, int dep, int start) {
    TreeNode p = root;
    int d = 0;
    int half = 1 << (dep - 2);
    while (p != null) {
        if (start > half) {
            p = p.right;
            start = start - half;
        } else {
            p = p.left;
        }
        half = half >> 1;
        d++;
    }
    return d == dep;
}

public static int countNodes2(TreeNode root) {
    if (root == null) {
        return 0;
    }

    TreeNode leftNode = root.left;
    int leftHeight = 0;
    while (leftNode != null) {
        leftHeight++;
    }
}
```

```
        leftNode = leftNode.right;
    }

    TreeNode rightNode = root.right;
    int rightHeight = 0;
    while (rightNode != null) {
        rightHeight++;
        rightNode = rightNode.right;
    }

    if (leftHeight == rightHeight) {
        return 1 + countNodes2(root.left) + ((int)Math.pow(2, rightHeight) - 1);
    } else {
        return 1 + countNodes2(root.right) + ((int)Math.pow(2, leftHeight) - 1);
    }
}

public static class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) {
        val = x;
    }
}

public static void main(String[] args) {
    TreeNode n1 = new TreeNode(1);
    TreeNode n2 = new TreeNode(2);
    TreeNode n3 = new TreeNode(3);
    TreeNode n4 = new TreeNode(4);
    TreeNode n5 = new TreeNode(5);
    TreeNode n6 = new TreeNode(6);
    TreeNode n7 = new TreeNode(7);
    TreeNode n8 = new TreeNode(8);
    TreeNode n9 = new TreeNode(9);
    TreeNode n10 = new TreeNode(10);
    n1.left = n2;
    n1.right = n3;
    n2.left = n4;
    n2.right = n5;
    n3.left = n6;
    n3.right = n7;
    n4.left = n8;
    n4.right = n9;
    n5.left = n10;
    System.out.print(countNodes2(n1));
}
}
```

Add Two Numbers

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Answer:

```
public class AddTwoSol {
    public static ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        if (l1 == null || l2 == null) {
            return null;
        }

        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        int carry = 0;
        while (l1 != null || l2 != null || carry == 1) {
            int sum = carry;
            if (l1 != null) {
                sum += l1.val;
                l1 = l1.next;
            }

            if (l2 != null) {
                sum += l2.val;
                l2 = l2.next;
            }

            carry = sum / 10;
            ListNode cur = new ListNode(sum % 10);
            tail.next = cur;
            tail = tail.next;
        }
        return dummy.next;
    }

    public static class ListNode {
        int val;
        ListNode next;
        ListNode(int x) {
            val = x;
        }
    }

    public static void main(String[] args) {
        ListNode n1 = new ListNode(2);
        ListNode n2 = new ListNode(4);
        ListNode n3 = new ListNode(3);
        ListNode n4 = new ListNode(5);
    }
}
```

```
        ListNode n5 = new ListNode(6);
        ListNode n6 = new ListNode(4);
        n1.next = n2;
        n2.next = n3;
        n3.next = null;
        n4.next = n5;
        n5.next = n6;
        n6.next = null;
        ListNode rst = addTwoNumbers(n1, n4);
        while (rst != null) {
            System.out.print(rst.val);
            rst = rst.next;
        }
    }
}
```

Single Number

Given an array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Answer:

```
public class Solution {
    public int singleNumber(int[] nums) {
        if (nums == null || nums.length == 0) {
            return -1;
        }

        int cur = 0;
        for (int i = 0; i < nums.length; i++) {
            cur ^= nums[i];
        }
        return cur;
        /*
        for (int i = 0; i < nums.length; i++) {
            int cur = nums[i];
            int cnt = 0;
            for (int j = 0; j < nums.length; j++) {
                if (cur == nums[j]) {
                    cnt++;
                }
            }
            if (cnt == 1) {
                return cur;
            }
        }
        return -1;
        */
    }
}
```

Single Number II

Given an array of integers, every element appears three times except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Answer:

/*

SOL 1 time O(n), space O(n)

SOL 2 time O(n), space O(1)

*/

```
public class SingleNum2Sol {
    public static int singleNumber1(int[] nums) {
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < nums.length; i++) {
            Integer cnt = map.get(nums[i]);
            if (cnt != null) {
                map.put(nums[i], cnt + 1);
            } else {
                map.put(nums[i], 1);
            }
        }

        Set<Integer> set = map.keySet();
        for (Integer i : set) {
            if (map.get(i) != 3) {
                return i;
            }
        }
        return 0;
    }

    public static int singleNumber2(int[] nums) {
        if (nums == null) {
            return 0;
        }

        int cur = 0;
        for (int i = 0; i < 32; i++) {
            int sum = 0;
            for (int j = 0; j < nums.length; j++) {
                if (((nums[j] >> i) & 1) == 1) {
                    sum++;
                    sum = sum % 3;
                }
            }
            cur |= sum << i;
        }
        return cur;
    }
}
```

by eamon 06/09/2015

```
    }

    public static void main(String[] args) {
        int[] nums = {7, 5, 4, 4, 4, 5, 5};
        System.out.print(singleNumber2(nums));
    }
}
```

Single Number III

Given $2 * n + 2$ numbers, every numbers occurs twice except two, find them.

Example

Given [1, 2, 2, 3, 4, 4, 5, 3] return 1 and 5

Challenge

$O(n)$ time, $O(1)$ extra space.

Answer:

/*

$2n + 2 \Rightarrow 2n' + 1, 2n'' + 1$

$s \wedge x \rightarrow s = 001 \wedge 101 = 100$

从某一位上分两拨, 把a和b分开, 剩下的数, 仍然成双成对在某个组出现

*/

/*

00000000000000000000000000000000100 -> s 32bits

11111111111111111111111111111111011

00000000000000000000000000000000100 -> -s ($\sim s + 1$) 取反加1

00000000000000000000000000000000100 -> y

*/

```
public class SingleNum3Sol {
    public static List<Integer> singleNumber(int[] nums) {
        List<Integer> rst = new ArrayList<Integer>();
        int s = 0;
        for (int x : nums) {
            s ^= x;
        }
        // int y = s & (-s);
        int[] ans = new int[2];
        for (int x : nums) {
            if ((x & s) != 0) {
                ans[0] ^= x;
            } else {
                ans[1] ^= x;
            }
        }
        for (int i : ans) {
            rst.add(i);
        }
        return rst;
    }
}
```

by eamon 06/09/2015

```
        public static void main(String[] args) {
            int[] nums = {1, 2, 2, 3, 4, 4, 5, 3};
            System.out.print(singleNumber(nums));
        }
    }
```

Majority Element

Given an array of size n , find the majority element. The majority element is the element that appears more than $\lfloor n/2 \rfloor$ times.

You may assume that the array is non-empty and the majority element always exist in the array.

Answer:

/*

两个数不一样就抵消掉

要对majority number进行检查, 以排除不存在majority number的情况. 如1, 2, 3, 4这样的数列, 是没有majority number的.

*/

```
public class MajorNumSol {
    public static int majorityElement(int[] nums) {
        if (nums == null || nums.length == 0) {
            return -1;
        }

        int maj = nums[0];
        int len = nums.length;
        // SOL 1 Voting
        int cnt = 0;
        for (int i = 0; i < len; i++) {
            if (cnt == 0) {
                maj = nums[i];
                cnt++;
            } else if (nums[i] == maj) {
                cnt++;
            } else {
                cnt--;
            }
        }
        // SOL 2 Examining
        /*
        cnt = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == maj) {
                cnt++;
            }
        }
        */
        if (cnt <= nums.length / 2) {
            return -1;
        }
    }
}
```



```
        return maj;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4};
        System.out.print(majorityElement(nums));
    }
}
```

Majority Number II

Given an array of integers, the majority number is the number that occurs more than 1/3 of the size of the array.

Find it.

Note

There is only one majority number in the array

Example

For [1, 2, 1, 2, 1, 3, 3] return 1

Challenge

O(n) time and O(1) space

Answer:

/*

对cnt1, cnt2减数时, 相当于丢弃了3个数字(当前数字, n1, n2). 也就是说, 每一次丢弃数字, 我们是丢弃3个不同的数字.

而Majority number超过了1/3所以它最后一定会留下来.

*/

```
public class MajorNum2Sol {
    public static int majorityNumber(int[] nums) {
        if (nums == null || nums.length <= 2) {
            return -1;
        }

        int n1 = 0;
        int n2 = 0;
        int cnt1 = 0;
        int cnt2 = 0;
        int len = nums.length;
        for (int i = 0; i < len; i++) {
            if (cnt1 != 0 && nums[i] == n1) {
                cnt1++;
            } else if (cnt2 != 0 && nums[i] == n2) {
                cnt2++;
            } else if (cnt1 == 0) {
                cnt1 = 1;
                n1 = nums[i];
            } else if (cnt2 == 0) {
                cnt2 = 1;
                n2 = nums[i];
            } else {
                cnt1--;
            }
        }
    }
}
```

```
                cnt2--;
            }
        }

        cnt1 = 0;
        cnt2 = 0;
        for (int num : nums) {
            if (num == n1) {
                cnt1++;
            } else if (num == n2) {
                cnt2++;
            }
        }

        if (cnt1 < cnt2) {
            return n2;
        }
        return n1;
    }

    public static void main(String[] args) {
        int[] nums = {1, 1, 1, 1, 2, 3, 2, 4, 4, 4};
        System.out.print(majorityNumber(nums));
    }
}
```

Majority Number III

Given an array of integers and a number k, the majority number is the number that occurs more than 1/k of the size of the array. Find it.

Note

There is only one majority number in the array.

Example

For [3,1,2,3,2,3,3,4,4,4] and k = 3, return 3

Challenge

O(n) time and O(k) extra space

Answer:

```
public class MajorNum3Sol {
    public static int majorityNumber(int k, int[] nums) {
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(nums[i])) {
                map.put(nums[i], map.get(nums[i]) + 1);
            } else if (map.size() < k) {
                map.put(nums[i], 1);
            } else {
                ArrayList<Integer> tmpList = new ArrayList<Integer>();
                for (Integer n : map.keySet()) {
                    tmpList.add(n);
                }
            }
        }
    }
}
```

```
    }
    for (Integer m : tmpList) {
        map.put(m, map.get(m) - 1);
        if (map.get(m) == 0) {
            map.remove(m);
        }
    }
}

int rst = 0;
int num = 0;
for (Integer node : map.keySet()) {
    if (num < map.get(node)) {
        num = map.get(node);
        rst = node;
    }
}
return rst;
}

public static void main(String[] args) {
    int[] nums = {3, 1, 2, 3, 2, 3, 3, 4, 4, 4};
    int k = 3;
    System.out.print(majorityNumber(k, nums));
}
}
```

Best Time to Buy and Sell Stock

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Answer:

/*

变型题:

Given an array *arr[]* of integers, find out the difference between any two elements such that larger element appears after the smaller number in *arr[]*.

Examples: If array is [2, 3, 10, 6, 4, 8, 1] then returned value should be 8 (Diff between 10 and 2). If array is [7, 9, 5, 6, 3, 2] then returned value should be 2 (Diff between 7 and 9)

*/

/*

注意本题目里限定的先后顺序

min记录最小买入价

maxProfit记录最大利润

遍历array, 不断更新最小买入价, 计算更新最大利润

*/

```
public class Solution {
    public int maxProfit(int[] prices) {
        if (prices == null) {
```

by eamon 06/09/2015

```
        return 0;
    }

    int minValue = Integer.MAX_VALUE;
    int maxProfit = 0;
    for (int i = 0; i < prices.length; i++) {
        minValue = Math.min(minValue, prices[i]);
        maxProfit = Math.max(maxProfit, prices[i] - minValue);
    }
    return maxProfit;
}
}
```

Best Time to Buy and Sell Stock II

Say you have an array for which the i th element is the price of a given stock on day i . Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Answer:

```
/*
每次比较当天和前一天的股票值, 如果是上升, 就加上这个差值即可
*/
public class Solution {
    public int maxProfit(int[] prices) {
        if (prices == null) {
            return 0;
        }

        int maxProfit = 0;
        for (int i = 1; i < prices.length; i++) {
            int dif = prices[i] - prices[i - 1];
            if (dif > 0) {
                maxProfit += dif;
            }
        }
        return maxProfit;
    }
}
```

Best Time to Buy and Sell Stock III

Say you have an array for which the i th element is the price of a given stock on day i . Design an algorithm to find the maximum profit. You may complete at most two transactions.

Note:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Answer:

```
/*
```

SOL 1

1. 从左往右扫描, 计算0-i的这个区间的最大利润. 方法可以参见股票第一题.

2. 从右往左扫描, 计算 $i-len$ 这个区间的最大利润.

3. 再从头至尾扫一次, 每个节点加上左边和右边的利润. 记录最大值.

$O(n^2)$ 的算法很容易想到:

找寻一个点 j , 将原来的 $prices[0..n-1]$ 分割为 $prices[0..j]$ 和 $prices[j..n-1]$, 分别求两段的最大profit.

进行优化:

对于点 $j+1$, 求 $prices[0..j+1]$ 的最大profit时, 很多重复的工作在求 $prices[0..j]$ 的最大profit中已经做过了,

类似于Best Time to Buy and Sell Stock, 可以在 $O(1)$ 的时间从 $prices[0..j]$ 推出 $prices[0..j+1]$ 的最大profit,

但是如何从 $prices[j..n-1]$ 推出 $prices[j+1..n-1]$? 反过来思考, 我们可以用 $O(1)$ 的时间由 $prices[j+1..n-1]$ 推出 $prices[j..n-1]$

数组 $left[i]$ 记录了 $prices[0..i]$ 的最大profit,

数组 $right[i]$ 记录了 $prices[i..n]$ 的最大profit.

已知 $left[i]$, 求 $left[i+1]$. 已知 $right[i]$, 求 $right[i-1]$.

用 $O(n)$ 的时间找出最大的 $left[i] + right[i]$

*/

```
public class BestTime3Sol {
    // SOL 1
    public static int maxProfit1(int[] prices) {
        if (prices == null || prices.length == 0) {
            return 0;
        }

        int len = prices.length;
        int[] left = new int[len];
        int[] right = new int[len];

        int min = prices[0];
        left[0] = 0;
        for (int i = 1; i < len; i++) {
            min = Math.min(min, prices[i]);
            left[i] = Math.max(left[i-1], prices[i] - min);
        }

        int max = prices[len-1];
        right[len-1] = 0;
        for (int i = len-2; i >= 0; i--) {
            max = Math.max(max, prices[i]);
            right[i] = Math.max(right[i+1], max - prices[i]);
        }

        int rst = 0;
        for (int i = 0; i < len; i++) {
            rst = Math.max(rst, left[i] + right[i]);
        }
        return rst;
    }

    // SOL 2 DP
    public static int maxProfit2(int[] prices) {
```

by eamon 06/09/2015

```
        if (prices == null || prices.length == 0) {
            return 0;
        }

        int rst = 0;
        int len = prices.length;
        int[] left = new int[len];

        int min = prices[0];
        left[0] = 0;
        for (int i = 1; i < len; i++) {
            min = Math.min(min, prices[i]);
            left[i] = Math.max(left[i - 1], prices[i] - min);
        }

        int max = Integer.MIN_VALUE;
        int profit = 0;
        for (int i = len - 1; i >= 0; i--) {
            max = Math.max(max, prices[i]);
            profit = Math.max(profit, max - prices[i]);
            rst = Math.max(rst, profit + left[i]);
        }
        return rst;
    }

    public static void main(String[] args) {
        int[] prices = {2, 3, 10, 6, 4, 8, 1};
        System.out.print(maxProfit2(prices));
    }
}
```

Best Time to Buy and Sell Stock IV

Say you have an array for which the i th element is the price of a given stock on day i .

Design an algorithm to find the maximum profit. You may complete at most k transactions.

Note:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Answer:

/*

$f[i][j]$ 表示前 i 天, 至多进行 j 次交易(也可以不到 j 次交易), 第 i 天必须sell的最大获益

$g[i][j]$ 表示前 i 天, 至多进行 j 次交易, 第 i 天sell或者不sell的最大获益

前 i 天, 至多交易 j 次, 第 i 天必须sell = 前 $i - 1$ 天, 至多交易 j 次, 第 $i - 1$ 天必须sell + increase; 前 $i - 1$ 天, 至多交易 $j - 1$ 次, 第 $i - 1$ 天可不sell + increase

前 i 天, 至多交易 j 次, 第 i 天可不交易 = 前 $i - 1$ 天, 至多交易 j 次, 第 $i - 1$ 天可不sell; 前 i 天, 至多交易 j 次, 第 i 天必须sell

*/

/*

56778435

len = 8

by eamon 06/09/2015

$k = 5 > \text{len} / 2$

profit $6 - 5 + 7 - 6 + 8 - 7 + 5 - 3 = 5$

$k = 3$

profit $6 - 5 + 7 - 6 + 8 - 7 = 3$ in fact even not larger than $8 - 5 + 5 - 3 = 5$

*/

```
public class Solution {
    public int maxProfit(int k, int[] prices) {
        if (k == 0) {
            return 0;
        }

        if (k >= prices.length / 2) {
            int profit = 0;
            for (int i = 1; i < prices.length; i++) {
                if (prices[i] > prices[i - 1]) {
                    profit += prices[i] - prices[i - 1];
                }
            }
            return profit;
        }

        int[][] f = new int[prices.length][k + 1];
        int[][] g = new int[prices.length][k + 1];

        f[0][0] = g[0][0] = 0;
        for (int i = 1; i <= k; i++) {
            f[0][i] = g[0][i] = 0;
        }

        for (int i = 1; i < prices.length; i++) {
            int increase = prices[i] - prices[i - 1];
            f[i][0] = 0;
            for (int j = 1; j <= k; j++) {
                f[i][j] = Math.max(g[i - 1][j - 1] + increase, f[i - 1][j] + increase);
                g[i][j] = Math.max(g[i - 1][j], f[i][j]);
            }
        }
        return g[prices.length - 1][k];
    }
}
```

Maximum Subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$,

the contiguous subarray $[4, -1, 2, 1]$ has the largest sum = 6.

More practice:

If you have figured out the $O(n)$ solution, try coding another solution using the divide and conquer approach, which is more subtle.

by eamon 06/09/2015

Answer:

```
/*
滑动窗口
*/
public class Solution {
    public int maxSubArray(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        int max = Integer.MIN_VALUE;
        int sum = 0;
        for (int i = 0; i < nums.length; i++) {
            if (sum < 0) {
                sum = 0;
            }

            sum += nums[i];
            max = Math.max(max, sum);
        }
        return max;
    }
}
```

Minimum Subarray

Given an array of integers, find the subarray with smallest sum.

Return the sum of the subarray.

Note

The subarray should contain at least one integer.

Example

For [1, -1, -2, 1], return -3

Answer:

```
public class MinSubSol {
    public static int minSubArray1(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        int min = Integer.MAX_VALUE;
        int sum = 0;
        int maxSum = 0;
        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            min = Math.min(min, sum - maxSum);
            maxSum = Math.max(sum, maxSum);
            /*
            if (sum > 0) {
                sum = 0;
            }
            sum += nums[i];
            */
        }
    }
}
```



```
        min = Math.min(min, sum);
        */
    }
    return min;
}
// 取反
public static int minSubArray2(int[] nums) {
    int len = nums.length;
    int max = Integer.MIN_VALUE;
    int sum = 0;
    for (int i = 0; i < len; i++) {
        if (sum < 0) {
            sum = -nums[i];
        } else {
            sum += -nums[i];
        }
        max = Math.max(max, sum);
    }
    return -max;
}

public static void main(String[] args) {
    int[] nums = {1, -1, -2, 1};
    System.out.print(minSubArray2(nums));
}
}
```

Maximum Subarray II

Given an array of integers, find two non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Note

The subarray should contain at least one number

Example

For given [1, 3, -1, 2, -1, 2], the two subarrays are [1, 3] and [2, -1, 2] or [1, 3, -1, 2] and [2], they both have the largest sum 7.

Answer:

```
public class MaxSubarray2Sol {
    public static int maxTwoSubArrays(int[] nums) {
        int len = nums.length;
        int[] left = new int[len];
        int[] right = new int[len];
        int sum = 0;
        int max = Integer.MIN_VALUE;
        int minSum = 0;
        for (int i = 0; i < len; i++) {
            if (sum < 0) {
                sum = 0;
            }
        }
    }
}
```

by eamon 06/09/2015

```
        sum += nums[i];
        max = Math.max(max, sum);
        left[i] = max;
        /*
            sum += nums[i];
            max = Math.max(max, sum - minSum);
            minSum = Math.min(sum, minSum);
            left[i] = max;
        */
    }
    sum = 0;
    max = Integer.MIN_VALUE;
    minSum = 0;
    for (int i = len - 1; i >= 0; i--) {
        sum += nums[i];
        max = Math.max(max, sum - minSum);
        minSum = Math.min(sum, minSum);
        right[i] = max;
    }

    max = Integer.MIN_VALUE;
    for (int i = 0; i < len - 1; i++) {
        max = Math.max(max, left[i] + right[i + 1]);
    }
    return max;
}

public static void main(String[] args) {
    int[] nums = {1, 3, -1, 2, -1, 2};
    System.out.print(maxTwoSubArrays(nums));
}
}
```

Maximum Subarray III

Given an array of integers and a number k, find k non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Note

The subarray should contain at least one number

Example

Given [-1,4,-2,3,-2,3],k=2, return 8

Answer:

/*

SOL 1

d[i][j] means the maximum sum we can get by selecting j subarrays from the first i elements.

SOL 2

DP d[i][j] means the maximum sum we can get by selecting j subarrays from the first i elements.

time $O(\text{len}^2 * k)$

```
*/
public class MaxSubarray3Sol {
    public static int maxSubArray1(int[] nums, int k) {
        int len = nums.length;
        if (len < k) {
            return 0;
        }

        int[][] d = new int[len + 1][k + 1];
        for (int i = 1; i <= len; i++) {
            for (int j = 1; j <= k; j++) {
                if (i < j) {
                    d[i][j] = 0;
                    continue;
                }
                d[i][j] = Integer.MIN_VALUE;
                for (int p = j - 1; p <= i - 1; p++) {
                    int local = nums[p];
                    int global = local;
                    for (int t = p + 1; t <= i - 1; t++) {
                        local = Math.max(local + nums[t], nums[t]);
                        global = Math.max(local, global);
                    }
                    if (d[i][j] < d[i][j] - 1 + global) {
                        d[i][j] = d[p][j] - 1 + global;
                    }
                }
            }
        }
        return d[len][k];
    }

    public static int maxSubArray2(int[] nums, int k) {
        if (nums.length < k) {
            return 0;
        }

        int len = nums.length;
        int[][] d = new int[len + 1][k + 1];
        for (int i = 0; i <= len; i++) {
            d[i][0] = 0;
        }

        for (int j = 1; j <= k; j++) {
            for (int i = j; i <= len; i++) {
                d[i][j] = Integer.MIN_VALUE;
                int endMax = 0;
                int max = Integer.MIN_VALUE;
                for (int p = i - 1; p >= j - 1; p--) {
```

```
        endMax = Math.max(nums[p], endMax + nums[p]);
        max = Math.max(endMax, max);
        d[i][j] = Math.max(d[i][j], d[p][j - 1] + max);
    }
}
return d[len][k];
}

public static void main(String[] args) {
    int[] nums = {-1, 4, -2, 3, -2, 3};
    int k = 2;
    System.out.print(maxSubArray2(nums, k));
}
}
```

Maximum Product Subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array [2,3,-2,4],
the contiguous subarray [2,3] has the largest product = 6.

Answer:

/*

DP

因为有正负值好几种情况, 所以计算当前节点最大值, 最小值时, 应该考虑前一位置的最大值, 最小值几种情况

e.g. 如果当前为-2, 前一个位置最小值为-6, 最大值为8, 那么当前最大值应该是-2 * -6 = 12

对于以index位置结尾的连续子串来说, 计算最大, 最小值可以三种选择:

1. 当前值 * 前一位置的最大值
2. 当前值 * 前一位置的最小值(若当前值负, 则最大)
3. 丢弃前一位置的所有的值

对这三项取最大值, 最小值, 就可以得到当前的最大乘积, 最小乘积

*/

```
public class Solution {
    public int maxProduct(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        int max = 1;
        int min = 1;
        int rst = Integer.MIN_VALUE;
        for (int i = 0; i < nums.length; i++) {
            int n1 = max * nums[i];
            int n2 = min * nums[i];
            max = Math.max(nums[i], Math.max(n1, n2));
            min = Math.min(nums[i], Math.min(n1, n2));
            rst = Math.max(max, rst);
        }
    }
}
```

by eamon 06/09/2015

```
        return rst;
    }
}
```

Maximum Subarray Difference

Given an array with integers.

Find two non-overlapping subarrays A and B, which $|SUM(A) - SUM(B)|$ is the largest.

Return the largest difference.

Note

The subarray should contain at least one number

Example

For [1, 2, -3, 1], return 6

Challenge

$O(n)$ time and $O(n)$ space.

Answer:

```
public class MaxDiffSol {
    public static int maxDiffSubArrays(int[] nums) {
        int len = nums.length;
        if (len == 0) {
            return 0;
        }

        int[] leftMin = new int[len];
        int[] leftMax = new int[len];
        int endMin = nums[0];
        int endMax = nums[0];
        leftMin[0] = endMin;
        leftMax[0] = endMax;
        for (int i = 1; i < len; i++) {
            endMax = Math.max(nums[i], nums[i] + endMax);
            leftMax[i] = Math.max(leftMax[i - 1], endMax);
            endMin = Math.min(nums[i], nums[i] + endMin);
            leftMin[i] = Math.min(leftMin[i - 1], endMin);
        }

        int[] rightMin = new int[len];
        int[] rightMax = new int[len];
        endMin = nums[len - 1];
        endMax = nums[len - 1];
        rightMin[len - 1] = endMin;
        rightMax[len - 1] = endMax;
        for (int i = len - 2; i >= 0; i--) {
            endMax = Math.max(nums[i], nums[i] + endMax);
            rightMax[i] = Math.max(rightMax[i + 1], endMax);
            endMin = Math.min(nums[i], nums[i] + endMin);
            rightMin[i] = Math.min(rightMin[i + 1], endMin);
        }

        int maxDiff = 0;
```

by eamon 06/09/2015

```
        for (int i = 0; i < len - 1; i++) {
            if (maxDiff < Math.abs(leftMin[i] - rightMax[i + 1])) {
                maxDiff = Math.abs(leftMin[i] - rightMax[i + 1]);
            }
            if (maxDiff < Math.abs(leftMax[i] - rightMin[i + 1])) {
                maxDiff = Math.abs(leftMax[i] - rightMin[i + 1]);
            }
        }
        return maxDiff;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, -3, 1};
        System.out.print(maxDiffSubArrays(nums));
    }
}
```

Minimum Size Subarray Sum

Given an array of n positive integers and a positive integer s , find the minimal length of a subarray of which the sum $\geq s$. If there isn't one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and $s = 7$, the subarray [4,3] has the minimal length under the problem constraint.

More practice:

If you have figured out the $O(n)$ solution, try coding another solution of which the time complexity is $O(n \log n)$.

Answer:

/*

SOL 1 Sliding Window

滑动窗口, 若小于 s , 把右边界往右滑动扩大窗口, 否则把左边界往右滑, 缩小窗口, 所有的数都是正数

SOL 2 Binary Search

$\text{sums}[i] - \text{target} = \text{sums}[i] - \text{sums}[i] + s - 1 < \text{sums}[i] - \text{sums}[\text{end}] = s$

*/

```
public class Solution {
    public int minSubArrayLen1(int s, int[] nums) {
        if (nums.length < 1) {
            return 0;
        }

        int left = 0;
        int right = 0;
        int sum = 0;
        int rst = nums.length + 1;
        while (right < nums.length) {
            sum += nums[right];
            while (sum >= s) {
                if (right - left + 1 == 1) {
                    return 1;
                }
            }
        }
    }
}
```

```
        rst = Math.min(rst, right - left + 1);
        sum -= nums[left];
        left++;
    }
    right++;
}
if (rst > nums.length) {
    return 0;
} else {
    return rst;
}
}

public int minSubArrayLen2(int s, int[] nums) {
    int len = nums.length;
    int[] sums = new int[len + 1];
    sums[0] = 0;
    for (int i = 1; i < len + 1; i++) {
        sums[i] = sums[i - 1] + nums[i - 1];
    }

    int rst = Integer.MAX_VALUE;
    for (int i = 1; i < len + 1; i++) {
        if (sums[i] >= s) {
            int end = binarySearch(0, i, sums[i] - s + 1, sums);
            if (end != -1 && i - end < rst) {
                rst = i - end;
            }
        }
    }

    if (rst == Integer.MAX_VALUE) {
        return 0;
    } else {
        return rst;
    }
}

private int binarySearch(int start, int end, int target, int[] sums) {
    if (start >= end) {
        return -1;
    }

    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (sums[mid] == target) {
            end = mid - 1;
        } else if (sums[mid] > target) {
            end = mid - 1;
        }
    }
}
```

by eamon 06/09/2015

```
        } else if (sums[mid] < target) {
            start = mid;
        }
    }
    if (sums[end] < target) {
        return end;
    } else if (sums[start] < target) {
        return start;
    } else return -1;
}
}
```

Subarray Sum

Given an integer array, find a subarray where the sum of numbers is zero. Your code should return the index of the first number and the index of the last number.

Example

Given [-3, 1, 2, -3, 4], return [0, 2] or [1, 3].

Answer:

/*

使用Map来记录index, sum的值. 当遇到两个index的sum相同时, 表示从index1+1到index2是一个解

注意: 添加一个index = -1作为虚拟节点. 这样我们才可以记录index1 = 0的解

time O(n) space O(n)

*/

```
public class SubSumSol {
    public static List<Integer> subarraySum(int[] nums) {
        int len = nums.length;
        List<Integer> rst = new ArrayList<Integer>();
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        map.put(0, -1);
        int sum = 0;
        for (int i = 0; i < len; i++) {
            sum += nums[i];
            if (map.containsKey(sum)) {
                rst.add(map.get(sum) + 1);
                rst.add(i);
                return rst;
            }
            map.put(sum, i);
        }
        return rst;
    }

    public static void main(String[] args) {
        int[] nums = {-3, 1, 2, -3, 4};
        System.out.print(subarraySum(nums));
    }
}
```


by eamon 06/09/2015

```
/*  
*outputs:  
*[0, 2]  
*/
```

Subarray Sum Closest

Given an integer array, find a subarray with sum closest to zero. Return the indexes of the first number and last number.

Example

Given [-3, 1, 1, -3, 5], return [0, 2], [1, 3], [1, 1], [2, 2] or [0, 4]

Challenge

$O(n \log n)$ time

Answer:

```
/*
```

```
Array.sort() time  $O(n \log n)$ 
```

```
*/
```

```
public class SubSumCloSol {  
    public static List<Integer> subarraySumClosest(int[] nums) {  
        List<Integer> rst = new ArrayList<Integer>();  
        if (nums.length == 0) {  
            return rst;  
        }  
  
        int len = nums.length;  
        Element[] sums = new Element[len + 1];  
        sums[0] = new Element(0, -1);  
        int sum = 0;  
        for (int i = 0; i < len; i++) {  
            sum += nums[i];  
            sums[i + 1] = new Element(sum, i);  
        }  
  
        Arrays.sort(sums);  
        int min = Math.abs(sums[0].getValue() - sums[1].getValue());  
        int start = Math.min(sums[0].getIndex(), sums[1].getIndex()) + 1;  
        int end = Math.max(sums[0].getIndex(), sums[1].getIndex());  
        for (int i = 1; i < len; i++) {  
            int diff = Math.abs(sums[i].getValue() - sums[i + 1].getValue());  
            if (diff < min) {  
                min = diff;  
                start = Math.min(sums[i].getIndex(), sums[i + 1].getIndex()) + 1;  
                end = Math.max(sums[i].getIndex(), sums[i + 1].getIndex());  
            }  
        }  
        rst.add(start);  
        rst.add(end);  
        return rst;  
    }  
}
```

```
public static class Element implements Comparable<Element> {
    int val;
    int index;
    public Element(int v, int i) {
        val = v;
        index = i;
    }

    public int compareTo(Element other) {
        return this.val - other.val;
    }

    public int getIndex() {
        return index;
    }

    public int getValue() {
        return val;
    }
}

public static void main(String[] args) {
    int[] nums = {-3, 1, 1, -3, 5};
    System.out.print(subarraySumClosest(nums));
}

/*
 * outputs:
 * [1, 3]
 */
```

Two Sum

Given an array of integers, find two numbers such that they add up to a specific target number. The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9

Output: index1=1, index2=2

Answer:

```
/*
consider {3, 2, 4} target = 6
{3, 3} {2, 4}
*/
```

```
public class TwoSumSol {
    public static int[] twoSum1(int[] nums, int target) {
        int[] index = new int[2];
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < nums.length; i++) {
```

```
        map.put(nums[i], i + 1);
    }

    for (int i = 0; i < nums.length; i++) {
        if (map.containsKey(target - nums[i])) {
            index[0] = i + 1;
            index[1] = map.get(target - nums[i]);
            if (index[0] == index[1]) {
                continue;
            }
            return index;
        }
    }
    return index;
}

// Can't use the sort method here, since the question asks for indexes.
public static int[] twoSum2(int[] nums, int target) {
    if (nums == null || nums.length < 2) {
        return null;
    }

    Arrays.sort(nums);
    int left = 0;
    int right = nums.length - 1;
    int[] index = new int[2];
    while (left < right) {
        int sum = nums[left] + nums[right];
        if (sum == target) {
            index[0] = left + 1;
            index[1] = right + 1;
            break;
        } else if (sum < target) {
            left++;
        } else {
            right--;
        }
    }
    return index;
}

public static void main(String[] args) {
    int[] nums = {2, 7, 11, 15};
    int[] rst = twoSum2(nums, 9);
    for (int i = 0; i < rst.length; i++) {
        System.out.print(rst[i] + " ");
    }
}
```

by eamon 06/09/2015

3Sum

Given an array S of n integers, are there elements a, b, c in S such that $a + b + c = 0$? Find all unique triplets in the array which gives the sum of zero.

Note:

Elements in a triplet (a,b,c) must be in non-descending order. (ie, $a \leq b \leq c$)

The solution set must not contain duplicate triplets.

For example, given array S = {-1 0 1 2 -1 -4},

A solution set is:

(-1, 0, 1)

(-1, -1, 2)

Answer:

/*

two pointer problem: time -> sort $O(n \log n)$ + pointer move $O(n^2)$

notice: skip duplicate numbers, e.g. [0,0,0,0]

*/

```
public class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> rst = new ArrayList<List<Integer>>();

        if (nums == null || nums.length < 3) {
            return rst;
        }

        Arrays.sort(nums);
        for (int i = 0; i < nums.length - 2; i++) {
            if (i != 0 && nums[i] == nums[i - 1]) {
                continue;
            }
            int left = i + 1;
            int right = nums.length - 1;
            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    List<Integer> list = new ArrayList<Integer>();
                    list.add(nums[i]);
                    list.add(nums[left]);
                    list.add(nums[right]);
                    rst.add(list);
                    left++;
                    right--;
                    while (left < right && nums[left] == nums[left - 1]) {
                        left++;
                    }
                    while (left < right && nums[right] == nums[right + 1]) {
                        right--;
                    }
                }
                else if (sum < 0) {
                    left++;
                }
                else {
                    right--;
                }
            }
        }
    }
}
```

by eamon 06/09/2015

```
        right--;  
    }  
}  
}  
return rst;  
}  
}
```

3Sum Closest

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

Answer:

```
/*  
time O(nlogn (sort) + n^2)  
notice: otherwise it will overflow for operation (closet - target)  
*/  
public class Solution {  
    public int threeSumClosest(int[] nums, int target) {  
        if (nums == null || nums.length < 3) {  
            return Integer.MAX_VALUE;  
        }  
  
        Arrays.sort(nums);  
        int closet = Integer.MAX_VALUE / 2;  
        for (int i = 0; i < nums.length - 2; i++) {  
            int left = i + 1;  
            int right = nums.length - 1;  
            while (left < right) {  
                int sum = nums[left] + nums[right] + nums[i];  
                if (sum == target) {  
                    return sum;  
                } else if (sum < target) {  
                    left++;  
                } else {  
                    right--;  
                }  
                if (Math.abs(sum - target) < Math.abs(closet - target)) {  
                    closet = sum;  
                } else {  
                    closet = closet;  
                }  
            }  
        }  
        return closet;  
    }  
}
```

4Sum

Given an array S of n integers, are there elements a, b, c, and d in S such that $a + b + c + d = \text{target}$? Find all unique quadruplets in the array which gives the sum of target.

Note:

Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, $a \leq b \leq c \leq d$)

The solution set must not contain duplicate quadruplets.

For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:

(-1, 0, 0, 1)

(-2, -1, 1, 2)

(-2, 0, 0, 2)

Answer:

/*

先排序, 这后两重for循环, 然后对最后的一个数组设两个指针遍历
time $O(n \log n) + O(n^2) * O(n) = O(n^3)$

*/

```
public class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> rst = new ArrayList<List<Integer>>();
        Arrays.sort(nums);

        for (int i = 0; i < nums.length - 3; i++) {
            if (i != 0 && nums[i] == nums[i - 1]) {
                continue;
            }

            for (int j = i + 1; j < nums.length - 2; j++) {
                if (j != i + 1 && nums[j] == nums[j - 1]) {
                    continue;
                }
                int left = j + 1;
                int right = nums.length - 1;
                while (left < right) {
                    int sum = nums[i] + nums[j] + nums[left] + nums[right];
                    if (sum < target) {
                        left++;
                    } else if (sum > target) {
                        right--;
                    } else {
                        List<Integer> list = new ArrayList<Integer>();
                        list.add(nums[i]);
                        list.add(nums[j]);
                        list.add(nums[left]);
                        list.add(nums[right]);
                        rst.add(list);
                        left++;
                        right--;
                        while (left < right && nums[left] == nums[left - 1]) {
                            left++;
                        }
                    }
                }
            }
        }
        return rst;
    }
}
```

```
    }
    while (left < right && nums[right] == nums[right + 1]) {
        right--;
    }
}
}
}
}
return rst;
}
}
```

k Sum

Given n distinct positive integers, integer k ($k \leq n$) and a number target.

Find k numbers where sum is target. Calculate how many solutions there are?

Example

Given [1,2,3,4], k=2, target=5. There are 2 solutions:

[1,4] and [2,3], return 2.

Answer:

/*

D[i][j][t]前i个数中, 挑出j个数, 组成和为t有多少方案

D[0][0][0]表示在一个空集中找出0个数, target为0, 则有1个解, 就是什么也不挑

(1) 我们可以把当前A[i - 1]这个值包括进来, 所以需要加上D[i - 1][j - 1][t - A[i - 1]] (前提是t - A[i - 1]要大于0)

(2) 我们可以不选择A[i - 1]这个值, 这种情况就是D[i - 1][j][t], 也就是说直接在前i - 1个值里选择一些值加到target.

*/

```
public class KSumSol {
    public static int kSum(int nums[], int k, int target) {
        if (target < 0) {
            return 0;
        }

        int len = nums.length;
        int[][][] d = new int[len + 1][k + 1][target + 1];

        for (int i = 0; i < len + 1; i++) {
            for (int j = 0; j < k + 1 && j <= i; j++) {
                for (int t = 0; t <= target; t++) {
                    if (j == 0 && t == 0) {
                        d[i][j][t] = 1;
                    } else if (!(i == 0 || j == 0 || t == 0)) {
                        d[i][j][t] = d[i - 1][j][t];
                        if (t - nums[i - 1] >= 0) {
                            d[i][j][t] += d[i - 1][j - 1][t - nums[i - 1]];
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
        return d[len][k][target];
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4};
        System.out.print(kSum(nums, 2, 5));
    }
}
```

Partition Array

Given an array nums of integers and an int k, partition the array (i.e move the elements in "nums") such that:

All elements < k are moved to the left

All elements >= k are moved to the right

Return the partitioning index, i.e the first index i nums[i] >= k.

Example

If nums=[3,2,2,1] and k=2, a valid answer is 1.

Note

You should do really partition in array nums instead of just counting the numbers of integers smaller than k.

If all elements in nums are smaller than k, then return nums.length

Challenge

Can you partition the array in-place and in O(n)?

Answer:

```
public class partitionArraySol {
    public static int partitionArray1(int[] nums, int k) {
        ArrayList<Integer> l = new ArrayList<Integer>();
        ArrayList<Integer> r = new ArrayList<Integer>();
        ArrayList<Integer> rst = new ArrayList<Integer>();
        int len = nums.length;
        int ans = 0;
        for (int i = 0; i < len; i++) {
            if (nums[i] < k) {
                ans++;
                l.add(nums[i]);
            } else {
                r.add(nums[i]);
            }
        }
        len = l.size();
        for (int i = 0; i < len; i++) {
            rst.add(l.get(i));
        }
        len = r.size();
        for (int i = 0; i < len; i++) {
            rst.add(r.get(i));
        }
        return ans;
    }
}
```



```
    }

    public static int partitionArray2(int[] nums, int k) {
        int i = 0;
        int j = nums.length - 1;
        while (i <= j) {
            while (i <= j && nums[i] < k) {
                i++;
            }
            while (i <= j && nums[j] >= k) {
                j--;
            }
            if (i <= j) {
                int tmp = nums[i];
                nums[i] = nums[j];
                nums[j] = tmp;
                i++;
                j--;
            }
        }
        return i;
    }

    public static void main(String[] args) {
        int[] test = {3, 2, 2, 1};
        System.out.print(partitionArray1(test, 2));
    }
}

/*
 * outputs:
 * 1
 */
```

Sort Colors

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note:

You are not suppose to use the library's sort function for this problem.

Follow up:

A rather straight forward solution is a two-pass algorithm using counting sort.

First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

Answer:

/*

SOL 1

类似radix sort, 先扫描一次得知所有的值出现的次数, 再依次setup它们即可

SOL 2

使用双指针指向左边排好的0和右边排好的2, 再加一个指针cur扫描整个数组. 一趟排序下来就完成了.

注意: 与右边交换之后, cur不能移动, 因为你有可能交换过来是1或是0, 还需要与左边交换. 而与左边交换后, cur就可以向右边移动了.

非常要注意的是, 我们要使用cur <= right作为边界值. 因为right指向的是未判断的值. 所以当cur == right时, 此值仍然需要继续判断

```
*/
public class SortColorsSol {
    // SOL 1 two pass
    public static void sortColors1(int[] nums) {
        if (nums == null || nums.length == 0) {
            return;
        }

        int len = nums.length;
        int red = 0;
        int white = 0;
        int blue = 0;
        for (int i = 0; i < len; i++) {
            if (nums[i] == 0) {
                red++;
            } else if (nums[i] == 1) {
                white++;
            } else {
                blue++;
            }
        }

        for (int i = 0; i < len; i++) {
            if (red > 0) {
                nums[i] = 0;
                red--;
            } else if (white > 0) {
                nums[i] = 1;
                white--;
            } else {
                nums[i] = 2;
            }
        }
    }

    // SOL 2 one pass
    public static void sortColors2(int[] nums) {
        if (nums == null || nums.length == 0) {
            return;
        }

        int len = nums.length - 1;
        int left = 0;
```

```
        int right = len;
        int cur = 0;
        while (cur <= right) {
            if (nums[cur] == 2) {
                swap(nums, cur, right);
                right--;
            } else if (nums[cur] == 0) {
                swap(nums, cur, left);
                left++;
                cur++;
            } else {
                cur++;
            }
        }
    }

    public static void swap(int[] nums, int n1, int n2) {
        int tmp = nums[n1];
        nums[n1] = nums[n2];
        nums[n2] = tmp;
    }
    // SOL 3
    public static void sortColors3(int[] nums) {
        if (nums == null || nums.length == 0) {
            return;
        }

        int left = 0;
        int right = nums.length - 1;
        int cur = 0;
        while (cur <= right) {
            switch (nums[cur]) {
                case 0:
                    swap(nums, left, cur);
                    left++;
                    cur++;
                    break;
                case 1:
                    cur++;
                    break;
                case 2:
                    swap(nums, cur, right);
                    right--;
                    break;
                default:
                    cur++;
                    break;
            }
        }
    }
```

```
    }

    public static void main(String[] args) {
        int[] nums = {2, 0, 1, 2};
        sortColors3(nums);
        for (int i = 0; i < nums.length; i++) {
            System.out.print(nums[i] + " ");
        }
    }
}
```

Sort Colors II

Given an array of n objects with k different colors (numbered from 1 to k), sort them so that objects of the same color are adjacent, with the colors in the order 1, 2, ... k .

注意

You are not suppose to use the library's sort function for this problem.

样例

Given colors=[3, 2, 2, 1, 4], $k=4$, your code should sort colors in-place to [1, 2, 2, 3, 4].

挑战

A rather straight forward solution is a two-pass algorithm using counting sort. That will cost $O(k)$ extra memory.

Can you do it without using extra memory?

Answer:

/*

SOL 1

使用快排, 时间复杂度是 $O(n\log n)$, 空间复杂度是 $O(1)$

SOL 2

inplace, $O(N)$ 时间复杂度的算法

使用类似桶排序的思想, 对所有的数进行计数

1. 从左扫描到右边, 遇到一个数字, 先找到对应的bucket, 第一个3对应的bucket是index = 2(bucket从0开始计算)
2. Bucket如果有数字, 则把这个数字移动到i的position(就是存放起来), 然后把bucket记为-1(表示该位置是一个计数器, 计1)
3. Bucket存的是负数, 表示这个bucket已经是计数器, 直接减1. 并把color[i]设置为0(表示此处已经计算过)
4. Bucket存的是0, 与3一样处理, 将bucket设置为-1, 并把color[i]设置为0(表示此处已经计算过)
5. 回到position i, 再判断此处是否为0(只要不是为0, 就一直重复2-4的步骤)
6. 完成1-5的步骤后, 从尾部到头部将数组置结果. (从尾至头是为了避免开头的计数器被覆盖)

e.g.

i = 0 3 2 2 1 4

2 2 -1 1 4

2 -1 -1 1 4

0 -2 -1 1 4

i = 3 -1 -2 -1 0 4

i = 4 -1 -2 -1 -1 0

*/

```
public class SortColors2Sol {
    public static void sortKColors1(int[] nums, int k) {
        if (nums == null) {
            return;
        }
    }
}
```

```
    }

    quickSort(nums, 0, nums.length - 1);
}

public static void quickSort(int[] nums, int left, int right) {
    if (left >= right) {
        return;
    }

    int pivot = nums[right];
    int pos = partition(nums, left, right, pivot);
    quickSort(nums, left, pos - 1);
    quickSort(nums, pos + 1, right);
}

public static int partition(int[] nums, int left, int right, int pivot) {
    while (left <= right) {
        while (left <= right && nums[left] < pivot) {
            left++;
        }
        while (left <= right && nums[right] >= pivot) {
            right--;
        }
        if (left <= right) {
            swap(nums, left, right);
            left++;
            right--;
        }
    }
    return left;
}

public static void swap(int[] nums, int left, int right) {
    int tmp = nums[left];
    nums[left] = nums[right];
    nums[right] = tmp;
}

public static void sortKColors2(int[] nums, int k) {
    if (nums == null) {
        return;
    }

    int len = nums.length;
    for (int i = 0; i < len; i++) {
        while (nums[i] > 0) {
            int val = nums[i];
            if (nums[val - 1] > 0) {
```

```
        nums[i] = nums[val - 1];
        nums[val - 1] = -1;
    } else if (nums[val - 1] <= 0) {
        nums[val - 1]--;
        nums[i] = 0;
    }
}

int index = len - 1;
for (int i = k - 1; i >= 0; i--) {
    int cnt = -nums[i];
    if (cnt == 0) {
        continue;
    }
    while (cnt > 0) {
        nums[index] = i + 1;
        index--;
        cnt--;
    }
}

public static void main(String[] args) {
    int[] nums = {3, 2, 2, 1, 4};
    sortKColors2(nums, 4);
    for (int i = 0; i < nums.length; i++) {
        System.out.print(nums[i] + " ");
    }
}
```

Sort Letters by Case

Given a string which contains only letters. Sort it by lower case first and upper case second.

Note

It's not necessary to keep the original order of lower-case letters and upper case letters.

Example

For "abAcD", a reasonable answer is "acbAD"

*/

```
public class SortLettersSol {
    public static void sortLetters(char[] nums) {
        int low = 0;
        int cap = nums.length - 1;
        while (low < cap) {
            while (low < cap && nums[low] >= 'a' && nums[low] <= 'z') {
                low++;
            }
            while (low < cap && nums[cap] >= 'A' && nums[cap] <= 'Z') {
                cap--;
            }
        }
    }
}
```

```
        }

        char tmp = nums[low];
        nums[low] = nums[cap];
        nums[cap] = tmp;
    }
}

public static void main(String[] args) {
    char[] nums = {'a', 'b', 'A', 'c', 'D'};
    sortLetters(nums);
    for (int i = 0; i < nums.length; i++) {
        System.out.print(nums[i] + " ");
    }
}
}
```

Interleaving Positive and Negative Numbers

Given an array with positive and negative integers. Re-range it to interleaving with positive and negative integers.

Note

You are not necessary to keep the original order of positive integers or negative integers.

Example

Given [-1, -2, -3, 4, 5, 6], after re-range, it will be [-1, 5, -2, 4, -3, 6] or any other legal answer.

Challenge

Do it in-place and without extra memory.

Answer:

/*

1. Put all the positive numbers at in the left part.
2. Have more Positive numbers. Reverse the array.
3. Reorder the negative and the positive numbers.

*/

```
public class InterleaveNumSol {
    public static int[] rerange(int[] nums) {
        if (nums == null || nums.length <= 2) {
            return nums;
        }

        int len = nums.length;
        int cntPositive = 0;

        for (int num : nums) {
            if (num > 0) {
                cntPositive++;
            }
        }

        int i1 = 0;
        for (int i2 = 0; i2 < len; i2++) {
```

```
        if (nums[i2] > 0) {
            swap(nums, i1, i2);
            i1++;
        }
    }

    int posPointer = 1;
    int negPointer = 0;
    if (cntPositive > nums.length / 2) {
        posPointer = 0;
        negPointer = 1;
        int left = 0;
        int right = len - 1;
        while (left < right) {
            swap(nums, left, right);
            left++;
            right--;
        }
    }

    while (true) {
        while (posPointer < len && nums[posPointer] > 0) {
            posPointer += 2;
        }
        while (negPointer < len && nums[negPointer] < 0) {
            negPointer += 2;
        }
        if (posPointer >= len || negPointer >= len) {
            break;
        }
        swap(nums, posPointer, negPointer);
    }
    return nums;
}

private static void swap(int[] nums, int left, int right) {
    int tmp = nums[left];
    nums[left] = nums[right];
    nums[right] = tmp;
}

public static void main(String[] args) {
    int[] nums = {-1, -2, -3, 4, 5, 6};
    rerange(nums);
    for (int i = 0; i < nums.length; i++) {
        System.out.print(nums[i] + " ");
    }
}
```


by eamon 06/09/2015

```
/*  
 * outputs:  
 * -1 5 -3 4 -2 6  
 */
```

Factorial Trailing Zeroes

Given an integer n , return the number of trailing zeroes in $n!$.

Note: Your solution should be in logarithmic time complexity.

Answer:

```
/*  
idea  
1. The ZERO comes from 10.  
2. The 10 comes from  $2 \times 5$   
3. And we need to account for all the products of 5 and 2. likes  $4 \times 5 = 20 \dots$   
4. So, if we take all the numbers with 5 as a factor, we'll have way more than enough even  
numbers to pair with them to get factors of 10  
Example One  
How many multiples of 5 are between 1 and 23?  
There is 5, 10, 15, and 20, for four multiples of 5. Paired with 2's from the even factors, this  
makes for four factors of 10, so:  $23!$  has 4 zeros.  
Example Two  
How many multiples of 5 are there in the numbers from 1 to 100?  
because  $100 \div 5 = 20$ , so, there are twenty multiples of 5 between 1 and 100.  
but wait, actually 25 is  $5 \times 5$ , so each multiple of 25 has an extra factor of 5, e.g.  $25 \times 4 = 100$ ,  
which introduces extra of zero.  
So, we need know how many multiples of 25 are between 1 and 100? Since  $100 \div 25 = 4$ , there  
are four multiples of 25 between 1 and 100.  
Finally, we get  $20 + 4 = 24$  trailing zeroes in 100!  
The above example tell us, we need care about 5,  $5 \times 5$ ,  $5 \times 5 \times 5$ ,  $5 \times 5 \times 5 \times 5 \dots$   
Example Three  
By given number 4617.  
 $5^1$  :  $4617 \div 5 = 923.4$ , so we get 923 factors of 5  
 $5^2$  :  $4617 \div 25 = 184.68$ , so we get 184 additional factors of 5  
 $5^3$  :  $4617 \div 125 = 36.936$ , so we get 36 additional factors of 5  
 $5^4$  :  $4617 \div 625 = 7.3872$ , so we get 7 additional factors of 5  
 $5^5$  :  $4617 \div 3125 = 1.47744$ , so we get 1 more factor of 5  
 $5^6$  :  $4617 \div 15625 = 0.295488$ , which is less than 1, so stop here.  
Then  $4617!$  has  $923 + 184 + 36 + 7 + 1 = 1151$  trailing zeroes.  
*/  
/*  
 $O(\log_5(N))$  (base 5!) is faster than any polynomial.  
Just need to count how many times 5 appears in  $n!$  during multiplication in different forms: 5, 25,  
125, 625, ...  
*/  
public class Solution {  
    public int trailingZeroes(int n) {  
        int cnt = 0;  
        while (n > 0) {  
            n = n / 5;  
            cnt += n;  
        }  
        return cnt;  
    }  
}
```

by eamon 06/09/2015

```
        cnt += n;
    }
    return cnt;
}
}
```

Pow(x, n)
Implement pow(x, n).

Answer:

/*

使用二分法

正数的时候, 先求n/2的pow, 再两者相乘即可

如果n是负数, 所有计算完成后, 执行 $x = 1 / x$ 就行

当 $n = -2147483648$ 必须要特别处理, 因为对这个数取反会得到相同的数(已经越界), 所以当n为负时, 先加个1再说

note: int 32 bits ($-2147483648, 2147483647$) ($-2^{31}, 2^{31} - 1$) (0.后面31个1 = $2^{31} - 1$)

注意 $n \% 2$ 如果为1, 记得再乘以x

*/

```
public class Solution {
    public double myPow(double x, int n) {
        if (x == 0) {
            return 0;
        }

        if (n == 0) {
            return 1;
        }

        if (n < 0) {
            double ret = x * myPow(x, -(n + 1));
            return (double)1 / ret;
        }

        double ret = myPow(x, n / 2);
        ret = ret * ret;
        if (n % 2 != 0) {
            ret = ret * x;
        }
        return ret;
    }
}
```

Fast Power

Calculate the $a^n \% b$ where a, b and n are all 32bit integers.

Example

For $2^{31} \% 3 = 2$

For $100^{1000} \% 1000 = 0$

Challenge

$O(\log n)$

by eamon 06/09/2015

Answer:

/*

实际上这题应该是suppose $n > 0$ 的,
利用取模运算的乘法法则 $(a * b) \% p = (a \% p * b \% p) \% p$
将 $a^n \% b$ 分解

*/

```
public class FastPowSol {
    public static int fastPower1(int a, int b, int n) {
        long ret = pow(a, b, n);
        return (int) ret;
    }

    public static long pow(int a, int b, int n) {
        if (a == 0) {
            return 0;
        }

        if (n == 0) {
            return 1 % b;
        }

        if (n == 1) {
            return a % b;
        }

        long ret = 0;
        ret = pow(a, b, n / 2);
        ret *= ret;
        ret %= b;

        if (n % 2 == 1) {
            ret *= pow(a, b, 1);
        }
        ret = ret % b;
        return ret;
    }

    public static void main(String[] args) {
        System.out.print(fastPower1(2, 3, 31));
    }
}
```

O(1) Check Power of 2

Using O(1) time to check whether an integer n is a power of 2.

Example

For n=4, return true;

For n=5, return false;

Challenge

O(1) time

Answer:

/*

bit manipulation

1的个数只能有1个才是power of 2

注意Integer.MIN_VALUE, 这个只有一个1, 但是是false

note: int 32 bits (-2147483648, 2147483647) (-2^{31} , $2^{31} - 1$) (0.后面31个1 = $2^{31} - 1$)

Integer.MIN_VALUE = -2147483648 二进制表示 = 1000...0 (31个0, 1是符号位)

*/

```
public class CheckPowSol {
    public static boolean checkPowerOf21(int n) {
        if (n <= 0) {
            return false;
        }

        if ((n & (n - 1)) == 0) {
            return true;
        } else return false;
    }

    public static boolean checkPowerOf22(int n) {
        int cnt = 0;
        for (int i = 0; i < 31; i++) {
            if (((n >> i) & 1) == 1) {
                cnt++;
            }
        }
        if (cnt == 1) {
            return true;
        }
        return false;
    }

    public static void main(String[] args) {
        System.out.print(checkPowerOf21(Integer.MIN_VALUE));
    }
}
```

Sqrt(x)

Implement int sqrt(int x).

Compute and return the square root of x.

Answer:

/*

二分法 binary search

其实这里有一个非常trick地地方:

就是当循环终止的时候, l一定是偏小, r一定是偏大(实际的值是介于l和r之间的)

比如以下的例子, 90开根号是9.48按照开方向下取整的原则, 我们应该返回L.

以下展示了在循环过程中, L, R两个变量的变化过程

1. System.out.println(sqrt(90));

L R

by eamon 06/09/2015

```
1 45
1 23
1 12
6 12
9 12
9 10
9
2. System.out.println(sqrt(20));
1 10
1 5
3 5
4 5
4
3. System.out.println(sqrt(3));
1 2
*/
public class Solution {
    public int mySqrt(int x) {
        if (x == 1 || x == 0) {
            return x;
        }

        int left = 1;
        int right = x;

        while (left + 1 < right) {
            int mid = left + (right - left) / 2;
            int quo = x / mid;

            if (quo == mid) {
                return quo;
            } else if (quo < mid) {
                right = mid;
            } else {
                left = mid;
            }
        }
        return left;
    }
}
```

Excel Sheet Column Title

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
```

by eamon 06/09/2015

27 -> AA

28 -> AB

Answer:

/*

Excel序是这样的: A~Z, AA~ZZ, AAA~ZZZ, ...

本质上就是将一个10进制数转换为一个26进制数

note:

28

append 'A' + 27 % 26 = 'B' -> 0 B

append 'A' -> 0 B 1 A, that is, BA

*/

```
public class Solution {
    public String convertToTitle(int n) {
        StringBuilder sb = new StringBuilder();
        if (n < 1) {
            return "";
        } else {
            while (n > 0) {
                n--;
                char c = (char) (n % 26 + 'A');
                sb.append(c);
                n /= 26;
            }
            return sb.reverse().toString();
        }
    }
}
```

Excel Sheet Column Number

Given a column title as appear in an Excel sheet, return its corresponding column number.

For example:

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

Answer:

/*

26进制转10进制, 注意以'A'而不是0开头, 因此要“+1”

e.g.

Input: "BA"

Output: 27

i 0

num = 0 * 26 + 'B' - 'A' + 1 = 2

i 1

num = 1 * 26 + 'A' - 'A' + 1 = 27

*/

by eamon 06/09/2015

```
public class Solution {  
    public int titleToNumber(String s) {  
        int num = 0;  
        for (int i = 0; i < s.length(); i++) {  
            num = num * 26 + (s.charAt(i) - 'A' + 1);  
        }  
        return num;  
    }  
}
```