高频题的陨落篇

**该篇主要讲解面试中的高频题目：**

**1. Single Number I, II, III**

**2. Majority Number I, II, III**

**3. Best Time to Buy and Sale Stock I, II, III**

**4. Subarray I, II, III, IV, V, VI**

**5. 2-Sum, 3-Sum, 3-Sum Closest, 4-Sum, k-Sum, k-Sum II**

**6. Quick Questions**

**7. Partition Array**

**1.1 Single Number I，利用异或a ^ a = 0的性质，扫描一遍即可。**

public int singleNumber(int[] A) {

if(A == null || A.length == 0){

return -1;

}

int res = A[0];

for(int i = 1; i < A.length; i++){

res ^= A[i];

}

return res;

}

**1.2 Single Number II，提供2种解法。第一种，为nk + 1类型题目的通解；第二种，该方法比较巧妙也比较难以理解，在位运算篇有详细解释，有能力者可以参考。**

public int singleNumber(int[] A) {

if(A == null || A.length == 0){

return -1;

}

int[] digits = new int[32];

for(int i = 0; i < 32; i++){

for(int j = 0; j < A.length; j++){

digits[i] += A[j] >> i & 1;

}

}

int res = 0;

for(int i = 0; i < 32; i++){

res += digits[i] % 3 << i;

}

return res;

}

public int singleNumber(int[] A) {

int ones = 0;

int twos = 0;

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

ones = (A[i] ^ ones) & (~twos);

twos = (A[i] ^ twos) & (~ones);

}

return ones;

}

**1.3 Single Number III，这道题是2n + 2的问题。将所有数异或后得到的结果c是a异或b的结果，那么问题来了，我们该如何将a和b区别开至两组。由于c一定不为0的，所以c的某一位一定不为0，所以我们可以根据这一位将所有数分成2组，即可得到结果。**

public List<Integer> singleNumberIII(int[] A) {

List<Integer> res = new ArrayList<Integer>();

if(A == null || A.length == 0){

return res;

}

int c = 0;

for(int num : A){

c ^= num;

}

int digit = 0;

for(int i = 0; i < 32; i++){

if((c >> i & 1) == 1){

digit = i;

}

}

res.add(0);

res.add(0);

for(int num : A){

if((num >> digit & 1) == 1){

res.set(0, res.get(0) ^ num);

} else {

res.set(1, res.get(1) ^ num);

}

}

return res;

}

**2.1 Majority Number I，又到了一年一度的Majority Number选举，每个数字都为自己投票，Majority Number获得的票数最多。不同的数字会抵消Majority Number的投票。**

public int majorityNumber(ArrayList<Integer> nums) {

if(nums == null || nums.size() == 0){

return -1;

}

int vote = nums.get(0);

int count = 1;

for(int i = 1; i < nums.size(); i++){

if(nums.get(i) == vote){

count++;

} else if(count == 0){

vote = nums.get(i);

count++;

} else {

count--;

}

}

return vote;

}

**2.2 Majority Number II，使用2个候选变量储存结果即可。注意，第一遍的count1和count2并不是2个候选数字在数组中出现的次数，需要重新扫描一遍数组，找出最终结果。**

public int majorityNumber(ArrayList<Integer> nums) {

if(nums == null || nums.size() == 0){

return -1;

}

if(nums.size() < 3){

return nums.get(0);

}

int vote1 = nums.get(0);

int vote2 = 0;

int count1 = 1;

int count2 = 0;

for(int i = 1; i < nums.size(); i++){

if(nums.get(i) == vote1){

count1++;

} else if(nums.get(i) == vote2){

count2++;

} else if(count1 == 0){

vote1 = nums.get(i);

count1++;

} else if(count2 == 0){

vote2 = nums.get(i);

count2++;

} else {

count1--;

count2--;

}

}

count1 = 0;

count2 = 0;

for(int num : nums){

if(num == vote1){

count1++;

} else if(num == vote2){

count2++;

}

}

return count1 > count2 ? vote1 : vote2;

}

**2.3 Majority Number III，我们使用一个HashMap来存储候选元素。这道题可以用来练习如何对HashMap进行遍历。同样注意，最后要重新遍历一遍找出最终结果。**

public int majorityNumber(ArrayList<Integer> nums, int k) {

if(nums == null || nums.size() == 0){

return -1;

}

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

map.put(nums.get(0), 1);

for(int i = 1; i < nums.size(); i++){

if(map.containsKey(nums.get(i))){

int value = map.get(nums.get(i)) + 1;

if(value == 0){

map.remove(nums.get(i));

} else {

map.put(nums.get(i), value);

}

} else {

if(map.size() < k - 1){

map.put(nums.get(i), 1);

} else {

ArrayList<Integer> keyList = new ArrayList<Integer>();

for(Map.Entry entry : map.entrySet()){

int key = (int)entry.getKey();

int value = (int)entry.getValue();

value--;

if(value == 0){

keyList.add(key);

}

map.put(key, value);

}

for(int key : keyList){

map.remove(key);

}

}

}

}

for(Map.Entry entry : map.entrySet()){

entry.setValue(0);

}

int vote = 0;

int count = 0;

for(int i = 0; i < nums.size(); i++){

if(map.containsKey(nums.get(i))){

int value = map.get(nums.get(i)) + 1;

if(value > count){

vote = nums.get(i);

count = value;

}

map.put(nums.get(i), value);

}

}

return vote;

}

**3.1 Best Time to Buy and Sale Stock I，局部最优和全局最优解法。**

public int maxProfit(int[] prices) {

if(prices == null || prices.length == 0){

return 0;

}

int local = 0;

int global = 0;

for(int i = 0; i < prices.length - 1; i++){

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

global = Math.max(local, global);

}

return global;

}

**3.2 Best Time to Buy and Sale Stock II，贪心算法。**

public int maxProfit(int[] prices) {

if(prices == null || prices.length == 0){

return 0;

}

int res = 0;

for(int i = 0; i < prices.length - 1; i++){

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

}

return res;

}

**3.3 Best Time to Buy and Sale Stock III，提供2种解法。稳定版和通用版。**

public int maxProfit(int[] prices) {

if(prices == null || prices.length == 0){

return 0;

}

int local = 0;

int global = 0;

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

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

for(int i = 0; i < prices.length - 1; i++){

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

global = Math.max(local, global);

left[i] = global;

}

local = 0;

global = 0;

for(int i = prices.length - 2; i >= 0; i--){

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

global = Math.max(local, global);

right[i] = global;

}

int max = 0;

for(int i = 0; i < prices.length - 1; i++){

max = Math.max(max, left[i] + right[i + 1]);

}

return max;

}

public int maxProfit(int[] prices) {

if(prices == null || prices.length == 0){

return 0;

}

int[] local = new int[3];

int[] global = new int[3];

for(int i = 0; i < prices.length - 1; i++){

int diff = prices[i + 1] - prices[i];

for(int j = 2; j >= 1; j--){

local[j] = Math.max(global[j - 1] + Math.max(diff, 0), local[j] + diff);

global[j] = Math.max(local[j], global[j]);

}

}

return global[2];

}

**4.1 Subarray I – Maximum Subarray，局部最优和全局最优以及Divide & Conquer解法。**

// Solution 1 - Dynamic Programming

public int maxSubArray(int[] A) {

if(A == null || A.length == 0){

return 0;

}

int local = A[0];

int global = A[0];

for(int i = 1; i < A.length; i++){

local = Math.max(0, local) + A[i];

global = Math.max(local, global);

}

return global;

}

// Solution 2 - Divide and Conquer

public int maxSubArray(int[] A) {

if(A == null || A.length == 0){

return 0;

}

return helper(A, 0, A.length - 1);

}

private int helper(int[] A, int left, int right){

if(left == right){

return A[left];

}

int mid = left + (right - left) / 2;

int leftSub = helper(A, left, mid);

int rightSub = helper(A, mid + 1, right);

int leftMax = A[mid];

int temp = 0;

for(int i = mid; i >= left; i--){

temp += A[i];

leftMax = Math.max(temp, leftMax);

}

int rightMax = A[mid + 1];

temp = 0;

for(int j = mid + 1; j <= right; j++){

temp += A[j];

rightMax = Math.max(temp, rightMax);

}

return Math.max(Math.max(leftSub, rightSub), leftMax + rightMax);

}

**4.2 Subarray II – Maximum Product Subarray，变形的局部最优和全局最优解法。**

public int maxProduct(int[] A) {

if(A == null || A.length == 0){

return 0;

}

int localMax = A[0];

int localMin = A[0];

int global = A[0];

for(int i = 1; i < A.length; i++){

int tempLocalMax = localMax;

localMax = Math.max(A[i], Math.max(A[i] \* localMin, A[i] \* localMax));

localMin = Math.min(A[i], Math.min(A[i] \* localMin, A[i] \* tempLocalMax));

global = Math.max(global, localMax);

}

return global;

}

**4.3 Subarray III – Minimum Subarray，将nums数组元素取相反数，再按Maximum Subarray的方法求最大，最后返回-global即可。**

public int minSubArray(ArrayList<Integer> nums) {

if(nums == null || nums.size() == 0){

return 0;

}

int local = -nums.get(0);

int global = -nums.get(0);

for(int i = 1; i < nums.size(); i++){

local = Math.max(local, 0) - nums.get(i);

global = Math.max(local, global);

}

return -global;

}

**4.4 Subarray IV – Maximum Subarray II，同Best Time to Buy and Sale Stock III稳定版的原理一样，两次扫描得到从左至右和从右至左的最优解，再最终扫描一次得到结果。**

public int maxTwoSubArrays(ArrayList<Integer> nums) {

if(nums == null || nums.size() < 2){

return 0;

}

int[] left = new int[nums.size()];

left[0] = nums.get(0);

int local = nums.get(0);

int global = nums.get(0);

for(int i = 1; i < nums.size(); i++){

local = Math.max(local, 0) + nums.get(i);

global = Math.max(local, global);

left[i] = global;

}

int[] right = new int[nums.size()];

right[nums.size() - 1] = nums.get(nums.size() - 1);

local = nums.get(nums.size() - 1);

global = nums.get(nums.size() - 1);

for(int i = nums.size() - 2; i >= 0; i--){

local = Math.max(local, 0) + nums.get(i);

global = Math.max(local, global);

right[i] = global;

}

int max = Integer.MIN\_VALUE;

for(int i = 0; i < nums.size() - 1; i++){

max = Math.max(max, left[i] + right[i + 1]);

}

return max;

}

**4.5 Subarray V – Maximum Subarray Difference，与上一题思路一样，只是这次要同时保存左右两边的最大和最小值，最终结果可能为左边最大-右边最小或者右边最大-左边最小。**

public int maxDiffSubArrays(ArrayList<Integer> nums) {

if(nums == null || nums.size() < 2){

return 0;

}

int len = nums.size();

int[] leftMax = new int[len];

leftMax[0] = nums.get(0);

int[] leftMin = new int[len];

leftMin[0] = nums.get(0);

int localMax = nums.get(0);

int globalMax = nums.get(0);

int localMin = nums.get(0);

int globalMin = nums.get(0);

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

localMax = Math.max(localMax, 0) + nums.get(i);

globalMax = Math.max(localMax, globalMax);

leftMax[i] = globalMax;

localMin = Math.min(localMin, 0) + nums.get(i);

globalMin = Math.min(localMin, globalMin);

leftMin[i] = globalMin;

}

int[] rightMax = new int[len];

rightMax[len - 1] = nums.get(len - 1);

int[] rightMin = new int[len];

rightMin[len - 1] = nums.get(len - 1);

localMax = nums.get(len - 1);

globalMax = nums.get(len - 1);

localMin = nums.get(len - 1);

globalMin = nums.get(len - 1);

for(int i = len - 2; i >= 0; i--){

localMax = Math.max(localMax, 0) + nums.get(i);

globalMax = Math.max(localMax, globalMax);

rightMax[i] = globalMax;

localMin = Math.min(localMin, 0) + nums.get(i);

globalMin = Math.min(localMin, globalMin);

rightMin[i] = globalMin;

}

int max = 0;

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

max = Math.max(Math.max(Math.abs(leftMax[i] - rightMin[i + 1]), Math.abs(rightMax[i + 1] - leftMin[i])), max);

}

return max;

}

**4.6 Subarray VI – Maximum Subarray III，该题就是Best Time to Buy and Sale Stock III中使用的通用解法。由于本人能力有限，理解的不是很透彻。仅在此提供一个解法，希望有朝一人能被有缘人发现，为其所用。**

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

d[i][j] = max{d[p][j-1]+maxSubArray(p+1,i)

we iterate p from i-1 to j-1, so we can record the max subarray we get at current p, this value can be used to calculate the max subarray from p-1 to i when p becomes p-1.

public int maxSubArray(ArrayList<Integer> nums, int k) {

if (nums.size() < k){

return 0;

}

int len = nums.size();

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.get(p), endMax + nums.get(p));

max = Math.max(endMax, max);

if (d[i][j] < d[p][j - 1] + max)

d[i][j] = d[p][j - 1] + max;

}

}

return d[len][k];

}

**5.1 2-Sum，巧妙利用HashMap和补集原理来寻找要求的2个数字。**

public int[] twoSum(int[] numbers, int target) {

if(numbers == null || numbers.length < 2){

return null;

}

int[] res = new int[2];

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

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

if(map.containsKey(target - numbers[i])){

res[0] = map.get(target - numbers[i]) + 1;

res[1] = i + 1;

return res;

}

map.put(numbers[i], i);

}

return res;

}

**5.2 3-Sum，遍历数组num一遍，使用排序+夹逼的方式找出2Sum等于-num[i]的结果即可。**

public ArrayList<ArrayList<Integer>> threeSum(int[] num) {

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

if(num == null || num.length < 3){

return res;

}

Arrays.sort(num);

for(int i = num.length - 1; i >= 2; i--){

if(i != num.length - 1 && num[i] == num[i + 1]){

continue;

}

ArrayList<ArrayList<Integer>> item = findTwoSum(num, i - 1, -num[i]);

for(int j = 0; j < item.size(); j++){

item.get(j).add(num[i]);

}

res.addAll(item);

}

return res;

}

private ArrayList<ArrayList<Integer>> findTwoSum(int[] num, int end, int target){

int left = 0;

int right = end;

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

while(left < right){

if(num[left] + num[right] == target){

ArrayList<Integer> item = new ArrayList<Integer>();

item.add(num[left]);

item.add(num[right]);

res.add(item);

left++;

right--;

while(left < right && num[left] == num[left - 1]){

left++;

}

while(left < right && num[right] == num[right + 1]){

right--;

}

} else if(num[left] + num[right] < target){

left++;

} else {

right--;

}

}

return res;

}

**5.3 3-Sum Closest，与3-Sum基本一致，只是找closest的时候细节需要稍作变动即可。**

public int threeSumClosest(int[] num, int target) {

if(num == null || num.length < 3){

return -1;

}

Arrays.sort(num);

int closest = Integer.MAX\_VALUE;

for(int i = 0; i < num.length - 2; i++){

int value = twoSumClosest(num, target - num[i], i + 1);

if(Math.abs(value) < Math.abs(closest)){

closest = value;

}

}

return target + closest;

}

private int twoSumClosest(int[] num, int target, int start){

int left = start;

int right = num.length - 1;

int closest = Integer.MAX\_VALUE;

while(left < right){

if(num[left] + num[right] == target){

return 0;

}

int value = num[left] + num[right] - target;

if(Math.abs(value) < Math.abs(closest)){

closest = value;

}

if(value < 0){

left++;

} else {

right--;

}

}

return closest;

}

**5.4 4-Sum，这道题提供2种解法。第一种，即是3-Sum再套一层循环，容易理解且代码容易写，但时间复杂度为O(n^3); 第二种，二分思想很好，合理利用了TreeMap数据结构，但代码较为复杂冗长，时间复杂度可以降至O(n^2logn)。**

public ArrayList<ArrayList<Integer>> fourSum(int[] num, int target) {

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

if(num == null || num.length < 4){

return res;

}

Arrays.sort(num);

for(int i = num.length - 1; i >= 3; i--){

if(i == num.length - 1 || num[i] != num[i + 1]){

ArrayList<ArrayList<Integer>> curRes = threeSum(num, 0, i - 1, target - num[i]);

for(int j = 0; j < curRes.size(); j++){

curRes.get(j).add(num[i]);

}

res.addAll(curRes);

}

}

return res;

}

public ArrayList<ArrayList<Integer>> threeSum(int[] num, int l, int r, int target) {

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

if(num == null || num.length < 3){

return res;

}

for(int i = r; i >= 2; i--){

if(i == r || num[i] != num[i + 1]){

ArrayList<ArrayList<Integer>> curRes = twoSum(num, 0, i - 1, target - num[i]);

for(int j = 0; j < curRes.size(); j++){

curRes.get(j).add(num[i]);

}

res.addAll(curRes);

}

}

return res;

}

public ArrayList<ArrayList<Integer>> twoSum(int[] num, int l, int r, int target) {

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

if(num == null || num.length < 2){

return res;

}

while(l < r){

if(num[l] + num[r] == target){

ArrayList<Integer> item = new ArrayList<Integer>();

item.add(num[l]);

item.add(num[r]);

res.add(item);

l++;

r--;

while(l < r && num[l] == num[l - 1]){

l++;

}

while(l < r && num[r] == num[r + 1]){

r--;

}

} else if(num[l] + num[r] < target){

l++;

} else {

r--;

}

}

return res;

}

import java.util.TreeMap;

public class Solution {

class Pair {

int a;

int ai;

int b;

int bi;

public Pair(int a, int ai, int b, int bi){

this.a = a;

this.ai = ai;

this.b = b;

this.bi = bi;

}

boolean same(Pair p){

return p != null && p.a == a && p.b == b;

}

}

public List<List<Integer>> fourSum(int[] num, int target) {

List<List<Integer>> res = new ArrayList<>();

if(num.length < 4){

return res;

}

Arrays.sort(num);

TreeMap<Integer, List<Pair>> map = new TreeMap<>();

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

for(int j = i + 1; j < num.length; j++){

Pair pair = new Pair(num[i], i, num[j], j);

int sum = num[i] + num[j];

List<Pair> list;

if(map.containsKey(sum)){

list = map.get(sum);

} else {

list = new ArrayList<>();

map.put(sum, list);

}

list.add(pair);

}

}

Integer first = map.firstKey();

Integer last = map.lastKey();

while(first != null && last != null && first <= last){

if(first + last > target){

last = map.lowerKey(last);

} else if(first + last < target){

first = map.higherKey(first);

} else {

Pair lastA = null;

for(Pair a : map.get(first)){

if(a.same(lastA)){

continue;

}

lastA = a;

Pair lastB = null;

for(Pair b: map.get(last)){

if(a.bi < b.ai){

if(b.same(lastB)){

continue;

}

lastB = b;

res.add(Arrays.asList(new Integer[]{a.a, a.b, b.a, b.b}));

}

}

}

last = map.lowerKey(last);

first = map.higherKey(first);

}

}

return res;

}

}

**5.5 k-Sum，**res[i][j][v] means the way of selecting i elements from the first j elements so that their sum equals to v. Then we have:

res[i][j][v] = res[i - 1][j - 1][v - A[j - 1]] + d[i][j - 1][v]

It means two operations, select the jth element and not select the jth element.

public int kSum(int A[], int k, int target) {

if(A == null || A.length < k){

return 0;

}

int[][][] res = new int[k + 1][A.length + 1][target + 1];

for(int i = 1; i <= A.length; i++){

if(A[i - 1] <= target){

for(int j = i; j <= A.length; j++){

res[1][j][A[i - 1]] = 1;

}

}

}

for(int i = 2; i <= k; i++){

for(int j = i; j <= A.length; j++){

for(int v = 1; v <= target; v++){

res[i][j][v] = 0;

if(i < j){

res[i][j][v] += res[i][j - 1][v];

}

if(A[j - 1] <= v){

res[i][j][v] += res[i - 1][j - 1][v - A[j - 1]];

}

}

}

}

return res[k][A.length][target];

}

**5.6 k-Sum II，回溯黄金模板，类似Combination Sum，只是加了要求k个数的限定条件。**

public ArrayList<ArrayList<Integer>> kSumII(int A[], int k, int target) {

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

if(A == null || A.length < k){

return res;

}

Arrays.sort(A);

ArrayList<Integer> item = new ArrayList<Integer>();

helper(A, k, 0, target, 0, item, res);

return res;

}

private void helper(int[] A, int k, int num, int target, int start, ArrayList<Integer> item, ArrayList<ArrayList<Integer>> res){

if(target < 0){

return;

}

if(num == k && target == 0){

res.add(new ArrayList<Integer>(item));

return;

}

for(int i = start; i < A.length; i++){

if(i > start && A[i] == A[i - 1]){

continue;

}

item.add(A[i]);

helper(A, k, num + 1, target - A[i], i + 1, item, res);

item.remove(item.size() - 1);

}

}

**6.1 Pow(x, n)，提供2种解法。第一种，位运算；第二种，二分法。**

// Solution 1 - Bit Operation

public double pow(double x, int n) {

if(n == 0){

return 1.0;

}

double res = 1.0;

if(n < 0){

if(x >= 1.0 / Double.MAX\_VALUE || x <= 1.0 / Double.MIN\_VALUE){

x = 1.0 / x;

} else {

return Double.MAX\_VALUE;

}

if(n == Integer.MIN\_VALUE){

res \*= x;

n++;

}

}

n = Math.abs(n);

boolean isNeg = false;

if(x < 0 && n % 2 == 1){

isNeg = true;

}

x = Math.abs(x);

while(n > 0){

if((n & 1) == 1){

if(res > Double.MAX\_VALUE / x){

return Double.MAX\_VALUE;

}

res \*= x;

}

x \*= x;

n >>= 1;

}

return isNeg ? -res : res;

}

// Solution 2 - Binary Store

public double pow(double x, int n) {

if(n == 0){

return 1.0;

}

double half = pow(x, n / 2);

if((n & 1) == 0){

return half \* half;

} else if(n > 0){

return half \* half \* x;

} else {

return half \* half / x;

}

}

**6.2 Sqrt，提供3种做法。第一种，二分查找；第二种，公式法；第三种，位运算。**

// Solution 1 - Binary Search

public int sqrt(int x) {

if(x < 0){

return -1;

}

if(x == 0 || x == 1){

return x;

}

int left = 1;

int right = x / 2 + 1;

while(left <= right){

int mid = left + (right - left) / 2;

if(mid <= x / mid && x / (mid + 1) < (mid + 1)){

return mid;

}

if(mid > x / mid){

right = mid - 1;

} else {

left = mid + 1;

}

}

return -1;

}

// Solution 2 - Formula

public int sqrt(int x) {

if(x < 0){

return -1;

}

if(x == 0 || x == 1){

return x;

}

double lastY = 0;

double y = 1;

while(y != lastY){

lastY = y;

y = (y + x / y) / 2;

}

return (int)y;

}

// Solution 3 - Bit Manipulation

public int sqrt(int x) {

int res = 0;

int bit = 1 << 16;

while(bit > 0){

res |= bit;

if(res == x / res){

break;

} else if(res > x / res){

res ^= bit;

}

bit >>= 1;

}

return res;

}

**6.3 Factorial Trailing Zero，由于尾数中的0只可能由是2 \* 5形成，而2的数量肯定是充足的，所以只要计算5的数量即可得到最终尾数中0的数量。**

public int trailingZeroes(int n) {

int res = 0;

while(n > 0){

n /= 5;

res += n;

}

return res;

}

**6.4 Check Power of 2 – O(1)，知之为知之，不知为不知。**

public boolean checkPowerOf2(int n) {

if(n == 0 || n == Integer.MIN\_VALUE){

return false;

}

return (n & (n - 1)) == 0;

}

**7.1 Partition Array，这道题是Quick Sort的一个Subroutine，应该好好练习。**

public int partitionArray(ArrayList<Integer> nums, int k) {

if(nums == null || nums.size() == 0){

return 0;

}

int left = 0;

int right = nums.size() - 1;

while(true){

while(nums.get(left) < k){

left++;

if(left == nums.size()){

break;

}

}

while(nums.get(right) >= k){

right--;

if(right == 0){

break;

}

}

if(left >= right){

break;

}

int temp = nums.get(left);

nums.set(left, nums.get(right));

nums.set(right, temp);

}

if(left == right && right == 0){

return 0;

}

if(left == right && left == nums.size() - 1){

return nums.size();

}

return right + 1;

}

**7.2 Sort Letters by Case，思路与Partition Array一样。**

public void sortLetters(char[] chars) {

if(chars == null || chars.length <= 1){

return;

}

int left = 0;

int right = chars.length - 1;

while(true){

while('a' <= chars[left] && chars[left] <= 'z'){

left++;

if(left == chars.length - 1){

break;

}

}

while('A' <= chars[right] && chars[right] <= 'Z'){

right--;

if(right == 0){

break;

}

}

if(left >= right){

break;

}

char temp = chars[left];

chars[left] = chars[right];

chars[right] = temp;

}

}

**7.3 Sort Colors，提供2种解法。第一种，Counting Sort思想；第二种，Two Pointers思想。**

// Solution 1 - Counting Sort

public void sortColors(int[] A) {

if(A == null || A.length == 0){

return;

}

int[] num = new int[3];

int[] res = new int[A.length];

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

num[A[i]]++;

}

for(int i = 1; i < num.length; i++){

num[i] += num[i - 1];

}

for(int i = A.length - 1; i >= 0; i--){

res[--num[A[i]]] = A[i];

}

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

A[i] = res[i];

}

}

// Solution 2 - Double Pointers

public void sortColors(int[] A) {

if(A == null || A.length == 0){

return;

}

int idx0 = 0;

int idx1 = 0;

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

if(A[i] == 0){

A[i] = 2;

A[idx1++] = 1;

A[idx0++] = 0;

} else if(A[i] == 1){

A[i] = 2;

A[idx1++] = 1;

}

}

}

**7.4 Interleaving Positive and Negative Numbers，一定要完美的岔开正数和负数。如果正数比负数多，那么开头第一个数一定要为正数，反之为负数。**

public int[] rerange(int[] A) {

if(A == null || A.length <= 1){

return A;

}

int posNum = 0;

for(int num : A){

posNum = num > 0 ? posNum + 1 : posNum;

}

int posIdx = posNum > (A.length - posNum) ? 0 : 1;

int negIdx = posNum > (A.length - posNum) ? 1 : 0;

while(true){

while(A[posIdx] > 0){

posIdx += 2;

if(posIdx > A.length - 1){

break;

}

}

while(A[negIdx] < 0){

negIdx += 2;

if(negIdx > A.length - 1){

break;

}

}

if(posIdx > A.length - 1 || negIdx > A.length - 1){

break;

}

int temp = A[posIdx];

A[posIdx] = A[negIdx];

A[negIdx] = temp;

}

return A;

}