Right Rotation

```
public static int rightRotate(String word1, String word2) {
    if (word1 == null || word2 == null || word1.length() == 0 ||
word2.length() == 0 || word1.length() != word2.length()) {
        return -1;
    }
    String str = word1 + word1;
    return str.indexOf(word2) != -1 ? 1 : -1;
}
```

Grey code

```
//term1和term2是题目给的两个BYTE
byte x = (byte)(term1 ^ term2);
int total = 0;
while(x != 0){
    x = (byte) (x & (x - 1));
    total++;
}
if(total == 1) return 1; else return 0;
```

remove vowel

```
StringBuffer sb = new StringBuffer();
String v = "aeiouAEIOU";
for(int i = 0; i < string.length(); i++){
    if(v.indexOf(string.charAt(i)) > -1) continue;
    sb.append(string.charAt(i));
}
return sb.toStirng();
```

检验括号

```
public boolean isValidParentheses(String s) {
        if (s == null || s.length() == 0)   return true;
        Stack<Character> stack = new Stack<Character>();

        for (int i = 0; i < s.length(); i++) {
            if (stack.empty())  stack.push(s.charAt(i));
            else if (s.charAt(i) - stack.peek() == 1 || s.charAt(i) -
stack.peek() == 2)    stack.pop();
            else    stack.push(s.charAt(i));
        }

        return stack.empty();
    }
```

longest palindromic substring

```
public String longestPalindrome(String s) {
   char[] chars = s.toCharArray();
   int len = s.length();
   while (len >= 0) {
       for (int i = 0; i + len - 1 < chars.length; i++) {
           int left = i;
           int right = i + len - 1;
           boolean good = true;
           while (left < right) {
               if (chars[left] != chars[right]) {
                   good = false;
                   break;
               }
               left++;
               right--;
           }
           if (good)
               return s.substring(i, i + len);
       }
       --len;
   }
   return "";
}
```

```
public String longestPalindrome(String s) {
      int len = s.length(), l = 0, r = 0, max = 0;
      for (int i = 0; i < len-1; i++) {
         int v1 = search(s, i, i);
         int v2 = search(s, i, i+1);
         int v = Math.max(v1, v2);

         if (v > max) {
            max = v;
            l = i - (v - 1) / 2;
            r = i + v / 2;
         }

      }

      return s.substring(l, r+1);
   }

   public int search(String s, int left, int right) {
      while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
         left--; right++;
      }
      return right - left - 1;
   }
```

Merge Two list

```java
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode head = new ListNode(0);
        ListNode cur = head;
        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                cur.next = l1;
                l1 = l1.next;
            }
            else {
                cur.next = l2;
                l2 = l2.next;
            }
            cur = cur.next;
        }
        cur.next = (l1 != null) ? l1 : l2;
        return head.next;
    }
}
```

reverse second half of linked list

```java
public static ListNode reverseSecondHalfList(ListNode head) {
    if (head == null || head.next == null)      return head;
    ListNode fast = head;
    ListNode slow = head;
    while (fast.next != null && fast.next.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    ListNode pre = slow.next;
    ListNode cur = pre.next;
    while (cur != null) {
        pre.next = cur.next;
        cur.next = slow.next;
        slow.next = cur;
        cur = pre.next;
    }
    return head;
}
```

Subtree

```
public boolean isSubTree(TreeNode T1, TreeNode T2) {
        if (T2 == null) return true;
        if (T1 == null) return false;
        return (isSameTree(T1,T2) || isSubTree(T1.left, T2) ||
isSubTree(T1.right, T2));
    }
    public boolean isSameTree(TreeNode T1, TreeNode T2) {
        if (T1 == null && T2 == null)
            return true;
        if (T1 == null || T2 == null)
            return false;
        if (T1.val != T2.val)
            return false;
        return (isSameTree(T1.left, T2.left) && isSameTree(T1.right,
T2.right));
    }
```

Two Sum

```
public static int TwoSumCount(int[] nums, int target) {
        if (nums == null || nums.length < 2)
            return 0;
        Map<Integer, Integer> map = new HashMap<Integer, Integer>();
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(target - nums[i]))
                count += map.get(target - nums[i]);
            if (!map.containsKey(nums[i]))
                map.put(nums[i], 1);
            else map.put(nums[i], map.get(nums[i]) + 1);
        }
        return count;
    }

    public static void main(String[] args) {
        int rvalue = TwoSumCount(new int[] {1, 1, 2, 3, 4}, 5);
        System.out.println(rvalue);
        return;
    }
```

Find K Nearest Point

```
public Point[] Solution(Point[] array, Point origin, int k) {
        Point[] rvalue = new Point[k];
        int index = 0;
        PriorityQueue<Point> pq = new PriorityQueue<Point> (k, new
Comparator<Point> () {
            @Override
            public int compare(Point a, Point b) {
                return (int) (getDistance(a, origin) - getDistance(b,
origin));
            }
        });

        for (int i = 0; i < array.length; i++) {
            pq.offer(array[i]);
            if (pq.size() > k)
                pq.poll();
        }
        while (!pq.isEmpty())
            rvalue[index++] = pq.poll();
        return rvalue;
    }
    private double getDistance(Point a, Point b) {
        return Math.sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y -
b.y));
    }
```

Overlap Rectangle

```
bool doOverlap(Point l1, Point r1, Point l2, Point r2)
{
    // If one rectangle is on left side of other
    if (l1.x > r2.x || l2.x > r1.x)
        return false;

    // If one rectangle is above other
    if (l1.y < r2.y || l2.y < r1.y)
        return false;

    return true;
}
// Overlap Rectangle
// Rect 1: top-left(A, B), bottom-right(C, D)
// Rect 2: top-left(E, F), bottom-right(G, H)
public int computeArea(int A, int B, int C, int D, int E, int F, int G, int
H) {
   int innerL = Math.max(A, E);
   int innerR = Math.max(innerL, Math.min(C, G));
   int innerT = Math.max(B, F);
   int innerB = Math.max(innerT, Math.min(D, H));
```

```
   return (C - A) * (B - D) - (innerR - innerL) * (innerT - innerB) + (G -E)
* (F - H);
}
```

window sum

(arraylist == null || arraylist.size() == 0)要return一个已经初始化的arrayList而不是null

```
public List<Integer> GetSum(List<Integer> A, int k) {
   ArrayList<Integer> result  = new ArrayList<>();
   if (A == null || A.size() == 0 || k <= 0) return result;
   int count = 0;
   for (int i = 0; i < A.size(); i++) {
       count++;
       if (count >= k) {
           int sum = 0;
           for (int j = i; j >= i - k + 1; j--) {
               sum += A.get(j);
           }
           result.add(sum);
       }
   }
   return result;
}
public int[] Solution(int[] array, int k) {
        if (array == null || array.length < k || k <= 0)    return null;
        int[] rvalue = new int[array.length - k + 1];
        for (int i = 0; i < k; i++)
            rvalue[0] += array[i];
        for (int i = 1; i < rvalue.length; i++) {
            rvalue[i] = rvalue[i-1] - array[i-1] + array[i+k-1];
        }
        return rvalue;
    }
```

GCD Greatest Common Divisor

```
public int Solution(int[] array) {
        if (array == null || array.length == 1) return 0;
        int gcd = array[0];
        for (int i = 1; i < array.length; i++) {
            gcd = gcd(gcd, array[i]);
        }
        return gcd;
    }
    private int gcd(int num1, int num2) {
        if (num1 == 0 || num2 == 0) return 0;
        while (num1 != 0 && num2 != 0) {
            if (num2 > num1) {
                num1 ^= num2;
                num2 ^= num1;
```

```
                num1 ^= num2;
            }
            int temp = num1 % num2;
            num1 = num2;
            num2 = temp;
        }
        return num1 + num2;
    }
```

LRU Cache count miss

```
public int Solution(int[] array, int size) {
        if (array == null)  return 0;
        List<Integer> cache = new LinkedList<Integer>();
        int count = 0;
        for (int i = 0; i < array.length; i++) {
            if (cache.contains(array[i])) {
                cache.remove(new Integer(array[i]));
            }
            else {
                count++;
                if (size == cache.size())
                    cache.remove(0);
            }
            cache.add(array[i]);
        }
        return count;
    }
```

Round Robin

```
class process {
    int arrTime;
    int exeTime;
    process(int arr, int exe) {
        arrTime = arr; exeTime = exe;
    }
}
public class RoundRobinScheduling {
    public float Solution(int[] Atime, int[] Etime, int q) {
        if (Atime == null || Etime == null || Atime.length != Etime.length)
            return 0;
        int length = Atime.length;
        Queue<process> queue = new LinkedList<process>();
        int curTime = 0, waitTime = 0;
        int index = 0;
        while (!queue.isEmpty() || index < length) {
            if (!queue.isEmpty()) {
                process cur = queue.poll();
                waitTime += curTime - cur.arrTime;
                curTime += Math.min(cur.exeTime, q);
                for (; index < length && Atime[index] <= curTime; index++)
                    queue.offer(new process(Atime[index], Etime[index]));
                if (cur.exeTime > q)
```

```
                    queue.offer(new process(curTime, cur.exeTime - q));
            }
            else {
                queue.offer(new process(Atime[index], Etime[index]));
                curTime = Atime[index++];
            }
        }
        return (float) waitTime / length;
    }
}
```

Rotate Matrix

```
public int[][] Solution(int[][] matrix, int flag) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
return matrix;
        //int m = matrix.length, n = matrix[0].length;
        int[][] rvalue;
        rvalue = transpose(matrix);
        flip(rvalue, flag);
        return rvalue;
    }

    private int[][] transpose(int[][] matrix) {
        int m = matrix.length, n = matrix[0].length;
        int[][] rvalue = new int[n][m];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                rvalue[i][j] = matrix[j][i];
        return rvalue;
    }

    private void flip(int[][] matrix, int flag) {
        int m = matrix.length, n = matrix[0].length;
        if (flag == 1) {    //clock wise
            for (int i = 0; i < m; i++)
                for (int j = 0; j < n / 2; j++) {
                    matrix[i][j] ^= matrix[i][n-j-1];
                    matrix[i][n-j-1] ^= matrix[i][j];
                    matrix[i][j] ^= matrix[i][n-j-1];
                }
        }
        else {
            for (int i = 0; i < m / 2; i++)
                for (int j = 0; j < n; j++) {
                    matrix[i][j] ^= matrix[m-i-1][j];
                    matrix[m-i-1][j] ^= matrix[i][j];
                    matrix[i][j] ^= matrix[m-i-1][j];
                }
        }
    }
```

Binary search tree minimum sum from root to leaf

```
public int Solution(TreeNode root) {
        if (root == null)   return 0;
        if (root.left != null && root.right == null)
            return Solution(root.left) + root.val;
        if (root.left == null && root.right != null)
            return Solution(root.right) + root.val;
        return Math.min(Solution(root.left), Solution(root.right)) +
root.val;
    }
```

Shorted job first

```
public float Solution(int[] req, int[] dur) {
        if (req == null || dur == null || req.length != dur.length)
            return 0;
        int index = 0, length = req.length;
        int waitTime = 0, curTime = 0;
        PriorityQueue<process> pq = new PriorityQueue<process>(new
Comparator<process>() {
            @Override
            public int compare(process p1, process p2) {
                if (p1.exeTime == p2.exeTime)
                    return p1.arrTime - p2.arrTime;
                return p1.exeTime - p2.exeTime;
            }
        });
        while (!pq.isEmpty() || index < length) {
            if (!pq.isEmpty()) {
                process cur = pq.poll();
                waitTime += curTime - cur.arrTime;
                curTime += cur.exeTime;
                while (index < length && curTime >= req[index])
                    pq.offer(new process(req[index], dur[index++]));
            }
            else {
                pq.offer(new process(req[index], dur[index]));
                curTime = req[index++];
            }
        }
        return (float) waitTime / length;
    }
```

Day change(cell growth)

```
public int[] Solution(int[] days, int n) {
        if (days == null || n <= 0) return days;
        int length = days.length;
        int[] rvalue = new int[length + 2];
        rvalue[0] = rvalue[length+1] = 0;
        int pre = rvalue[0];
```

```
        for (int i = 1; i <= length; i++)
            rvalue[i] = days[i-1];
        for (int i = 0; i < n; i++) {
            for (int j = 1; j <= length; j++) {
                int temp = rvalue[j];
                rvalue[j] = pre ^ rvalue[j+1];
                pre = temp;
            }
        }
        return Arrays.copyOfRange(rvalue, 1, length+1);
    }
```

Maze

```
public class Maze {
    private final static int[] dx = {-1, 0, 0, 1};
    private final static int[] dy = {0, 1, -1, 0};
    public int Solution(int[][] matrix) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0)
            return 0;
        if (matrix[0][0] == 9)
            return 1;
        int m = matrix.length, n = matrix[0].length;
        Queue<int[]> queue = new LinkedList<int[]>();
        queue.offer(new int[]{0, 0});
        matrix[0][0] = 1;
        while (!queue.isEmpty()) {
            int[] cur = queue.poll();
            for (int i = 0; i < 4; i++) {
                int[] next = {cur[0] + dx[i], cur[1] + dy[i]};
                if (next[0] >= 0 && next[0] < m && next[1] >= 0 && next[1] <
n) {
                    if (matrix[next[0]][next[1]] == 9)
                            return 1;
                    else if (matrix[next[0]][next[1]] == 0) {
                        queue.offer(next);
                        matrix[next[0]][next[1]] = 1;
                    }
                }
            }
        }
        return 0;
    }
}
```

Maximum Sliding Window

```
public int[] maxSlidingWindow(int[] nums, int k) {
        if (nums == null || k <= 0) return new int[0];
        int n = nums.length; int ri = 0;;
        int[] r = new int[n-k+1];
```

```
        Deque<Integer> q = new ArrayDeque<>();
        for (int i = 0; i < nums.length; i++) {
            while (!q.isEmpty() && q.peek() < i - k + 1) {
                q.poll();
            }
            while (!q.isEmpty() && nums[q.peekLast()] > nums[i]) {
                q.pollLast();
            }
            q.offer(i);
            if (i >= k - 1) {
                r[ri++] = nums[q.peek()];
            }
        }
        return r;

    }
```

Four Integer

```
public int[] Solution(int A, int B, int C, int D) {
        int[] rvalue = new int[4];
        rvalue[0] = A; rvalue[1] = B; rvalue[2] = C; rvalue[3] = D;
        Arrays.sort(rvalue);
        swap(rvalue, 0, 1); (rvalue, 2, 3); swap(rvalue, 0, 3);
        return rvalue;
    }
    private void swap(int[] array, int i, int j) {
        array[i] ^= array[j];
        array[j] ^= array[i];
        array[i] ^= array[j];
    }
```

Arithmetic sequence

```
public static int getLAS(int[] A){
    // Time: O(n)
    // Space: O(1)
    if (A.length < 3) return 0;

    int res = 0;
    int diff = Integer.MIN_VALUE;
    int count = 0;
    int start = 0;
    for (int i = 1; i < A.length; i++){
        int currDiff = A[i] - A[i - 1];
        if (diff == curDiff){
            count += i - start - 1 > 0 ? i - start - 1 : 0;
        } else {
            start = i - 1;
            diff = currDiff;
            res += count;
            count = 0;
        }
    }
```

```
    res += count;
    return res;
}
```

Tree Amplitude

```
public int Solution(TreeNode root) {
        if (root == null)   return 0;
        return helper(root, root.val, root.val);
    }
    private int helper(TreeNode root, int min, int max) {
        if (root == null)   return max - min;

        if (root.val < min) min = root.val;
        if (root.val > max) max = root.val;

        return Math.max(helper(root.left, min, max), helper(root.right, min,
max));
    }
```

Search in 2D matrix

```
class Point {
    int x;
    int y;
}
Point isInMatrix(int[][] matrix, int target){
    int row = matrix.length;
    int column = matrix[0].length;
    int r = 0;
    int c = column - 1;
    while (r < row && c >= 0){
        if (matrix[r][c] == target){
            return new Point(r,c);
        }
        if (matrix[r][c] > target){
            c--;
        } else {
            r++;
        }
    }
    return new Point(-1,-1);
}
```

Maximum Minimum Path

```
int helper(int[][] matrix){
    int[] result = new int[matrix[0].length];
    result[0] = matrix[0][0];
    for(int i=1; i<matrix[0].length; i++){
        result[i] = Math.min(result[i-1], matrix[0][i]);
    }
    for(int i=1; i<matrix.length; i++){
        result[0] = Math.min(matrix[i][0], result[0]);
        for(int j=1; j<matrix[0].length; j++){
           result[j] = (result[j]<matrix[i][j] && result[j-1]<matrix[i][j])?
               Math.max(result[j-1], result[j]):matrix[i][j];
            }
        }
        return result[result.length-1];
}
public class MaximumMinimumPath {
    private int min, max, row, col;
    public int maxMinPath(int[][] matrix) {
        row = matrix.length;
        col = matrix[0].length;
        min = Integer.MAX_VALUE;
        max = Integer.MIN_VALUE;
        dfsHelper(matrix, min, 0, 0);
     return max;
    }

    public void dfsHelper(int[][] matrix, int min, int i, int j ){
        if (i >= row || j >= col) return;
        if (i == row - 1 && j == col - 1) {
            min = Math.min(min, matrix[i][j]);
            max = Math.max(max, min);
            return;
```

```
        }
        min = Math.min(min, matrix[i][j]);
        dfsHelper(matrix, min, i, j + 1);
        dfsHelper(matrix, min, i + 1, j);
    }
}
```

若是用Java，用到queue, list啥的记得前面手动import java.util.* 2.所有函数都是static的，所以自己写其他helper函数的时候记得加上static

```
import java.util.ArrayDeque;
import java.util.Deque;
```

Close Two Sum

```
public static void findOptimalWeights(double capacity, double[] weights, int numOfContainers){
        double min = 0.0; minPos = 0; int maxPos = weights.length - 1;
        int i =0, j = weights.length-1;
        Arrays.sort(weights);

        while(i < j){
            double sum = weights[i] + weights[j];

            if(sum > min && sum <= capacity){
                min = sum;
                minPos = i;
                maxPos = j;
            }
            if(sum > capacity){
                j--;
            }else {
                i++;
            }
        }
        System.out.println("The two numbers for which sum is closest to target are "
                + weights[minPos] + " and " + weights[maxPos]);
    }
```

1. right rotation.

Code:

```
public static int rightRotate(String word1, String word2) {
    if (word1 == null || word2 == null || word1.length() == 0 || word2.length()
== 0 || word1.length() != word2.length()) {
        return -1;
    }
    String str = word1 + word1;
    return str.indexOf(word2) != -1 ? 1 : -1;
}
```

18/18 passed

2. grey code. 没用地里大神的x & (x - 1)这个做法。

```
public static int greyCode(byte element1, byte element2) {
    byte res = (byte) (element1 ^ element2);
    for (int i = 0; i <= 7; i++) {
        byte temp = (byte)(1 << i);
        if (temp == res) {
            return 1;
        }
    }
    System.out.println("No");
    return 0;
}
```

16/16 passed.