Window Sum

```
public static ArrayList<Integer> getWindowSum(ArrayList<Integer> list, int k){
        if(list == null || list.size() == 0)
                return new ArrayList<Integer>();
        if(k < 1 || k > list.size())            return null;
        ArrayList<Integer> ans = new ArrayList<Integer>();
        int len = list.size();
        int sum = 0;
        for(int i = 0; i < len; i++){
                sum += list.get(i);
                if(i-k+1 >= 0){
                        ans.add(sum);
                        sum -= list.get(i-k+1);
                }
        }
        return ans;
}
```

```
/*
 * window sum就是给一个包含整数的arraylist和一个window size k，
 * 返回所有长度为k的窗口的数的和。
 * 比如数组[1,2,3,4,5],window size 2，
 * 那么长度为2的窗口就是[1,2],[2,3],[3,4],[4,5],和就依次是3,5,7,9.
 */
```

Rotate Matrix

```
public static int[][] rotate(int[][] matrix, int flag){
        int m = matrix.length, n = matrix[0].length;
        int[][] buf = new int[n][m];
        //shit matrix
        for(int i = 0; i < m; i++)
                for(int j = 0; j < n; j++)
                        buf[j][i] = matrix[i][j];
        if(flag == 1){ // rotate clockwise
                for(int i = 0; i < n; i++){
                        for(int j = 0; j < m/2; j++){
                                int tmp = buf[i][j];
                                buf[i][j] = buf[i][m-1-j];
                                buf[i][m-1-j] = tmp;
                        }
                }
        }
        else{   // rotate counter-clockwise
                for(int i = 0; i < n/2; i++){
                        for(int j = 0; j < m; j++){
                                int tmp = buf[i][j];
                                buf[i][j] = buf[n-i-1][j];
                                buf[n-i-1][j] = tmp;
                        }
                }
        }
        return buf;
}
```

GetKClosestPoint

```
public static Point[] getKClosest(Point[] points, int k) {
        if(points == null || points.length == 0)
                return points;
        if(k < 0)  return null;
        Point o = new Point(0, 0);
        Arrays.sort(points, new Comparator<Point>(){
                @Override
                public int compare(Point a, Point b){
                        return distance(a, o) - distance(b, o);
                }
        });
        if(k >= points.length)        return points;
        Point[] ans = new Point[k];
        for(int i = 0; i < k; i++)
                ans[i] = points[i];
        return ans;
}
public static int distance(Point a, Point b){
        return (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y);
}
```

```
public static Point[] getKClosest(Point[] points, Point origin, int k) {
        if(points == null || points.length < k) return points;
        PriorityQueue<Point> pq = new PriorityQueue<Point>(new
Comparator<Point>(){
                @Override
                public int compare(Point a, Point b){
                        return Double.compare(distance(b, origin),
                                distance(a, origin));
                }
        });
        for(Point p : points) {
                pq.offer(p);
                if(pq.size() > k)
                        pq.poll();
        }
        Point[] ans = new Point[k];
        while(!pq.isEmpty())
                ans[--k] = pq.poll();
        return ans;
}
```

```
/*
 * 一个组织发现了外星人，要给他们通信。
 * 我们的任务是给太空中的一些有可能有外星人的点发射信号。
 * 但是由于天线质量差（真是奇怪的理由），只能给太空中的 k 个点发射信号。
 * 现在又已知一个点P，它的坐标是(0,0)，这个点周围最有可能有外星人。
 * 好了，给你N个点，  找到这个N个点中离原点P最近的k个。
 */
```

LRU Cache Miss

```
public static int countMissLL(int[] arr, int size){
        if(arr == null || arr.length == 0)      return 0;
        if(size < 1)        return arr.length;
        LinkedList<Integer> cache = new LinkedList<Integer>();
        int missed = 0;
        for(int x : arr){
                if(cache.contains(x))
                        cache.remove(x);
                else
                        missed++;
                cache.addFirst(x);
                if(cache.size() > size)
                        cache.removeLast();
        }
        return missed;
}
```

```
public static int countMiss(int[] arr, int size){
        if(arr == null || arr.length == 0)      return 0;
        if(size < 1)        return arr.length;
        int missed = 0;
        LinkedHashMap<Integer, Boolean> cache = new LinkedHashMap<Integer,
                        Boolean>(size, 0.75f, true){
                @Override
                public boolean removeEldestEntry(Map.Entry<Integer, Boolean> eldest){
                        return this.size() > size;
                }
        };
        for(int x : arr){
                if(cache.get(x) == null){
                        missed++;
                        cache.put(x, true);
                }
        }
        return missed;
}
```

BST Min Path Sum

```
// for binary search tree from root to a leaf
public static int minPathSum(TreeNode  root){
          if(root == null)    return 0;
          if(root.left != null && root.right != null)
                    return Math.min(minPathSum(root.left),
                              minPathSum(root.right))  + root.val;
          if(root.left != null)
                    return minPathSum(root.left)  + root.val;
          return minPathSum(root.right)  + root.val;
}
```

跟BST没啥关系，不要看到BST就以为是最左边的路径之和（左边路径可以很长，右边路径可以很短），用递归做很简单。

```
// for general binary tree with arbitrary start and end nodes
public static int minPathSumAny(TreeNode root){
          Result r = new Result(Integer.MAX_VALUE);
          minPath(root, r);
          return r.val;
}
public static int minPath(TreeNode root, Result res){
          if(root == null)      return 0;
          int l = minPath(root.left, res);
          int r = minPath(root.right, res);
          int min = Math.min(l, r) + root.val;
          int min2 = Math.min(min, l + r + root.val);
          res.val = Math.min(res.val, min2);
          return min;
}
```

Insert Cycle List

```
public ListNode insert(ListNode arb, int val){
          ListNode newNode = new ListNode(val);
          if(arb == null){
                    newNode.next  = newNode;
                    return newNode;
          }
          ListNode ptr = arb;
          do{
                    // val is between two nodes, stop search
                    if(val >= ptr.val && val <= ptr.next.val)
                              break;
                    if(ptr.val > ptr.next.val && (val > ptr.val
                                        || val < ptr.next.val))
                              break;
                    ptr = ptr.next;
          }while(ptr != arb);
          newNode.next  = ptr.next;
          ptr.next  = newNode;
          return newNode;

}
```

Company Tree

```java
public static Node getMaxAvgSubtree(Node root){
        if(root == null)    return root;
        Node[] ans = new Node[1];
        helper(root, ans);
        return ans[0];
}
public static ResultWrapper helper(Node root, Node[] ans) {
        int sum = root.val, num = 1;
        double maxAvg = Integer.MIN_VALUE;
        if(root.children == null || root.children.isEmpty())
                return new ResultWrapper(sum, num, maxAvg);
        for(Node child : root.children){
                ResultWrapper rw = helper(child, ans);
                sum += rw.sum;
                num += rw.num;
                maxAvg = Math.max(maxAvg, rw.maxAvg);
        }
        double curAvg = (double) sum / num;
        if(curAvg > maxAvg){
                ans[0] = root;
                maxAvg = curAvg;
        }
        return new ResultWrapper(sum, num, maxAvg);
}
```

Longest Palindrome

```java
public static String longestPalindrome(String s) {
        int[] pos = new int[2];
        for(int i = 0; i < s.length(); i++){
                expand(s, i, i,pos);
                expand(s, i-1, i, pos);
        }
        return s.substring(pos[0], pos[0] + pos[1]);
}

public static void expand (String s, int i, int j, int[] pos){
        while(i >= 0 && j < s.length() && s.charAt(i) == s.charAt(j)){
                if(j-i+1 > pos[1]){
                        pos[0] = i;
                        pos[1] = j-i+1;
                }
                i--;
                j++;
        }
}
```

City Connection

```java
public static ArrayList<Connection> getLowCost(ArrayList<Connection> connections) {
    if(connections == null || connections.isEmpty()) return connections;
    Collections.sort(connections, new Comparator<Connection>(){
        @Override
        public int compare(Connection c1, Connection c2){     return c1.cost - c2.cost; }
    });
    Map<String, String> map = new HashMap<String, String>();// pre-processing to make city connect to itself
    for(Connection con : connections){
        map.put(con.node1, con.node1);
        map.put(con.node2, con.node2);
    }
    ArrayList<Connection> ans = new ArrayList<Connection>(); // traverse connections to build MST
    for(Connection con : connections){
        String root1 = root(con.node1, map);
        String root2 = root(con.node2, map);
        if(root1.equals(root2)) continue;          // if they are already connected
        map.put(root2, root1);                     // union them
        ans.add(con);
    }
    if(map.size() - 1 != ans.size())return null;   //检查是否联通，不连通的话边更少
    Collections.sort(ans, new Comparator<Connection>(){
        @Override
        public int compare(Connection c1, Connection c2){
            if(c1.node1.equals(c2.node1))
                return c1.node2.compareTo(c2.node2);
            return c1.node1.compareTo(c2.node1);
        }
    });
    return ans;
}
```

Order Dependency

```java
public static List<Order> getOrderList(List<Order_Dependency> orderDependencies){
        Map<String, Order> orderMap = new HashMap<String, Order>();
        Map<String, Integer> in_degree = new HashMap<String, Integer>();
        Map<String, Set<String>> graph = new HashMap<String, Set<String>>();
        for(Order_Dependency od : orderDependencies) {
                String order = od.order.name;
                String dept = od.dependent.name;
                orderMap.putIfAbsent(order, od.order);
                orderMap.putIfAbsent(dept, od.dependent);
                in_degree.putIfAbsent(order, 0);
                in_degree.putIfAbsent(dept, 0);
                if(!graph.containsKey(order) || !graph.get(order).contains(dept))
                        in_degree.put(dept, in_degree.get(dept) + 1);         // duplicate dependencies would be ignored.
                graph.putIfAbsent(order, new HashSet<String>());
                graph.get(order).add(dept);
        }
        Queue<String> que = new LinkedList<String>();
        for(String key : in_degree.keySet())
                if(in_degree.get(key) == 0)
                        que.offer(key);
        List<Order> ans = new ArrayList<Order>();
        while(!que.isEmpty()) {
                String s = que.poll();
                ans.add(orderMap.get(s));
                Set<String> adjs = graph.get(s);
                if(adjs == null) continue;
                for(String adj : adjs)
                        if(in_degree.put(adj, in_degree.get(adj) - 1) == 1)
                                que.offer(adj);
        }
        if(in_degree.size() != ans.size())          return null;
        return ans;
}
```

High Five

```
public static Map<Integer, Double> getHighFive(List<Node> scores) {
        Map<Integer, PriorityQueue<Integer>> scoreMap = new HashMap<Integer, PriorityQueue<Integer>>();
        for(Node s : scores){
                scoreMap.putIfAbsent(s.id, new PriorityQueue<Integer>(5));
                PriorityQueue<Integer> ss = scoreMap.get(s.id);
                ss.offer(s.score);
                if(ss.size()>5)     ss.poll();
        }
        Map<Integer, Double> avgHighScore = new HashMap<Integer, Double>();
        for(int id : scoreMap.keySet()){
                PriorityQueue<Integer> pq = scoreMap.get(id);
                double sum = 0;
                for(double s : pq)
                        sum+=s;
                avgHighScore.put(id, sum / 5);
        }
        return avgHighScore;
}
```

CopyListWithRandomPointer

```
public static RandomListNode copy(RandomListNode head){
        if(head == null)        return head;
        RandomListNode ptr = head;
        // make copies for all nodes
        while(ptr != null) {
                RandomListNode copy = new RandomListNode(ptr.label);
                copy.next = ptr.next;
                ptr.next = copy;
                ptr = copy.next;
        }
        // setup random node links
        ptr = head;
        while(ptr != null){
                RandomListNode copy = ptr.next;
                if(ptr.random != null)
                        copy.random = ptr.random.next;
                ptr = copy.next;
        }
        // separate nodes
        RandomListNode dummy = new RandomListNode(-1);
        ptr = dummy;
        while(head != null) {
                ptr.next = head.next;
                ptr = ptr.next;
                head.next = ptr.next;
                head = head.next;
                ptr.next = null;
        }
        return dummy.next;
}
```

Sliding Window Max

```
public static int[] maxSlidingWindow(int[] nums, int k) {
          if(nums == null || nums.length == 0 || k > nums.length)      return new int[0];
          if(k < 1) return nums;
          int len = nums.length;
          Deque<Integer> dq = new LinkedList<Integer>();
          int[] ans = new int[len - k + 1];
          for(int i = 0; i < len; i++) {
                    while(!dq.isEmpty() && nums[dq.peekLast()] <= nums[i])
                              dq.pollLast();
                    dq.offerLast(i);
                    if(dq.peekFirst() + k == i)
                              dq.pollFirst();
                    if(i + 1 - k >= 0)
                              ans[i+1-k] = nums[dq.peekFirst()];
          }
          return ans;
}
```

Gray Code

```
public static int check(byte a, byte b){
        byte x = (byte) (a ^ b);
        int count = 0;
        while(x != 0){
                count++;
                x = (byte) (x & (x-1));
        }
        return count == 1 ? 1 : 0;
}
```

```
Given two hexadecimal numbers find if they can be
consecutive in gray code
For example: 10001000, 10001001
return 1
since they are successive in gray code
Example2: 10001000, 10011001
return -1
since they are not successive in gray code.
```

Four Integer

```
public static int[] makeLargest(int a, int b, int c, int d){
        int[] ans = new int[]{a, b, c, d};
        Arrays.sort(ans);
        swap(ans, 0, 1);
        swap(ans, 2, 3);
        swap(ans, 0, 3);
        return ans;
}

public static void swap(int[] arr, int i, int j){
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
}
```

```
/*
 * Given four integers, make F(S) = abs(S[0]-S[1])+abs(S[1]-S[2])+abs(S[2]-S[3])  to be largest.
 */
```

Rotate String

```
public static boolean isRoundRotated(String s1, String s2){
          if(s1 == null || s2 == null)  return false;
          if(s1.length() != s2.length()) return false;
          return (s1 + s1).indexOf(s2) >= 0;
}
```

Given two words, find if second word is the round rotation of first word.

For example: abc, cab

return 1

since cab is round rotation of abc

Example2: ab, aa

return -1

since ab is not round rotation for aa

Remove Vowels

```
public static String removeVowel(String s) {
          if(s == null || s.isEmpty())  return s;
          String vowels = "aeiouAEIOU";
          StringBuilder sb = new StringBuilder();
          for(int i = 0; i < s.length(); i++){
                    if(vowels.indexOf(s.charAt(i))  == -1)
                              sb.append(s.charAt(i));
          }
          return sb.toString();
}
```

Closest Two Sum

```
public static double[] find(double[] weights, double target){
    if(weights == null || weights.length < 2) return null;
    Arrays.sort(weights);
    int i = 0, j = weights.length - 1;
    double[] ans = new double[2];
    while(i < j){
        if(weights[i] + weights[j] == target){
            ans[0] = weights[i];
            ans[1] = weights[j];
            return ans;
        }
        else if(weights[i] + weights[j] < target){
            ans[0] = weights[i];
            ans[1] = weights[j];
            i++;
        }
        else
            j--;
    }
    return j == 0? null : ans;
}
```

要求在array中选出两个weights總总和小于等于capacity但最接近capacity 然後指定到一個Container object並且return

Reverse Half Linked List

```
public static ListNode reverseHalf(ListNode head){
    ListNode dummy = new ListNode(-1);
    dummy.next = head;
    ListNode slow = dummy, fast = dummy.next;
    // find out middle node
    while(fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    dummy.next = null;
    fast = slow.next;
    slow.next = null;
    while(fast != null){
        ListNode cur = fast;
        fast = fast.next;
        cur.next = slow.next;
        slow.next = cur;
    }
    return head;
}
```

GCD

```java
public static int gcd(int[] arr){
        if(arr == null || arr.length == 0)
                return -1;
        int ans = arr[0];
        for(int i = 1; i < arr.length; i++)
                ans = gcd(ans, arr[i]);
        return ans;
}

public static int gcd(int a, int b){
        if(b == 0)   return a;
        return gcd(b, a % b);
}
```

Subtree Check

```java
public static boolean checkSubtree(TreeNode a, TreeNode b){
        if(b == null)        return true;
        if(a == null)        return false;
        return sameTree(a, b) || checkSubtree(a.left, b)
                || checkSubtree(a.right, b);
}

public static boolean sameTree(TreeNode a, TreeNode b){
        if(a == null && b == null)
                return true;
        if(a == null || b == null)
                return false;
        if(a.val != b.val)
                return false;
        return sameTree(a.left, b.left) && sameTree(a.right, b.right);
}
```

Tree Amplitude

```
public static int maxDiff(TreeNode root){
        if(root == null)    return 0;
        return maxDiff(root, root.val, root.val);
}

public static int maxDiff(TreeNode root, int min, int max){
        if(root == null)    return max - min;
        min = Math.min(min, root.val);
        max = Math.max(max, root.val);
        return Math.max(maxDiff(root.left, min, max),
                maxDiff(root.right, min, max));
}
```

Given a tree of N nodes, return the amplitude of the tree
就是从 root 到 leaf max - min 的差

Arithmetic Sequence

```
public static int count(int[] arr){
        if(arr == null || arr.length < 3)
                return 0;
        int sum = 0, count = 0;
        for(int i = 2; i < arr.length; i++){
                if(arr[i] - arr[i-1] == arr[i-1] - arr[i-2]){
                        count++;
                        sum += count;
                }
                else
                        count = 0;
        }
        return sum;
}
```

Given an array, return the number of possible arithmetic sequence.
给一个数组，返回可能的等差数列个数。

Round Robin

```
public static float roundRobin(int[] aTime, int[] eTime, int q){
        if(aTime == null || aTime.length == 0) return 0;
        Queue<Process> que = new LinkedList<Process>();
        que.offer(new Process(aTime[0], eTime[0]));
        int waitTime = 0, curTime = 0;
        int len = aTime.length;
        int idx = 1;
        while(!que.isEmpty() || idx < len){
        if(que.isEmpty()){
                        que.offer(new Process(aTime[idx], eTime[idx]));
                        curTime = aTime[idx++];
                        continue;
                }
                Process p = que.poll();
                waitTime += curTime - p.arrTime;
                curTime += p.duration >= q ? q : p.duration;
                while(idx < len && aTime[idx] <= curTime){
                        que.offer(new Process(aTime[idx], eTime[idx++]));
                }
                if(p.duration > q)
                        que.offer(new Process(curTime, p.duration - q));
        }
        return (float) waitTime / len;
}
```

Shortest Job First

```
public static double calWaitingTime(int[] aTime, int[] eTime){
    // aTime is already sorted.
    if(aTime == null || aTime.length == 0)   return 0;
    PriorityQueue<Process>  pq = new PriorityQueue<Process>(new  Comparator<Process>(){
        @Override
        public int compare(Process p1, Process p2){
            if(p1.execTime  == p2.execTime)
                return p1.arrTime - p2.arrTime;
            return p1.execTime - p2.execTime;
        }
    });
    int idx = 0, len = aTime.length;
    int curTime = aTime[0], waitTime = 0;
    while(!pq.isEmpty()  || idx < len){
        if(pq.isEmpty()){
            curTime = aTime[idx];
            while(idx < len && aTime[idx]  <= curTime)
                pq.offer(new  Process(aTime[idx],  eTime[idx++]));
            continue;
        }
        Process p = pq.poll();
        waitTime += curTime - p.arrTime;
        curTime += p.execTime;
        while(idx < len && aTime[idx]  <= curTime)
            pq.offer(new  Process(aTime[idx],  eTime[idx++]));
    }
    return (double) waitTime / len;
}
```