work simulation 90min(120min) 21Q
不是deadline和requirement权衡
排序题
从高到低打分 按照某原则排序 然后打分
比如 选择方案组合
给各种方案组合 打分 有的requirement满足好一些 有的deadline满足好一些 逻辑不能乱 有的方案一定是由于另一种方案 如果把这种分打的比不好的方案低 就是逻辑有问题
衡量逻辑
排序才是关键
a 推荐程度5星 需要4周完成
b 推荐程度4星 需要3周完成
c 推荐程度3星 需要3周完成
结论 方案ab一定比ac分高
一定先排序 再打分

客户是上帝，dealine你拖不起

每个选项可以评1~5分，most effective是5，然后1是least effective

刚开始让你看一些介绍amazon工作环境的视频

1 上来给一段video，场景是项目的晨会，就是把team正在推进的项目描述一下，期间会有多个项目和你有关系，后面会遇到。
2 进入工作界面。在这里可以看到接收到邮件，接收到的instant message。
3 进入工作状态。会有同事给你发邮件，发信息。需要你对他们提出的问题做一些判断，也就是给解决问题的选项评分。
4 一共有21道题。中间会穿插video。有让你看log分析bug原因的，有让你看分析报告给出问题结论的，当然更多的还是那种让你判断怎么推进项目走向的。

第一个情境是给图书馆写图书推荐系统，关于book api的题目
第一问让两个人继续说
大约是两个人在对一个项目发表不同的观点。感觉选择tell me more的选项是最正确的
因为一开始基本上只是那两个人在强调自己是对的。你选了tell me more之后他会具体告诉你他们在说什么。我当时选了其中的一个人得观点，结果后面接的就是“等等他还不知道全部的信息” 然后他就继续把其他你需要知道的信息告诉你

第二问选图书馆的服务器有没有开放关于实体书的api
一是两个小哥讨论图书推荐的API应该自己做还是用现成的。自己做的API涵盖面广但是due赶不上了，别人做的只有电子书，但是可以赶上due，就是明年才能涵盖纸质书。
book api那道选的deadline

俩个年轻老白讨论客人要强烈要求有硬皮书的推荐，但服务器里只有digital版本的，到底要不要加这个功能，感觉后面的视频是根据你的选择来的（有待考证）

经理让我去调差bug, 去问一个烙印拿bug的报告
后面有会议说系统出现bug，该做出什么反应，选看internal bug 记录
里面有个会议室白人，亚裔，烙印在讨论服务器最近好多complaints,然后我选则的要看Internal test，结果后面会议结束烙印站起来义正言辞跟我说，我已经写了20年服务器了，不可能有错误的，而且我刚刚才调试过机器，绝对不可能是内部错误。呵呵，里面有个选项问，烙印 is not helpful

这个问题是关于Amazon recommendation system 的，给你推荐一些你可能感兴趣的item。但是第一个issue是总失败， 第二个issue是显示Germany。
第一个问题是因为user name太长了，所以一直报错。 第二个问题是因为他用proxy的name来决定是不是语言了。
德国amazon出了什么问题，让你看log回答问题的。问你大概是哪里出了问题
亚马逊推荐广告给客户，给英国人推荐的德文的广告，还有些广告不是amazon的产品，给你log文件，问你这个bug的问题可能在哪儿 找bug in error log
各个员工讨论case media network 服务器最近好多complaints,
有德国的，有invalid recommendation的，给了个列表好多国家的服务器返回什么404/ german recommendation/ invalid recom/问是什么原因。
找相同出错原因问那几条的相同点就可以
看log得题就找相同错误的规律，我记得有道我选了地点都在德国，有个是因为username太长没存全
看log file的， 一个德语因为服务器， 一个用户名太长
一个是有些book recommendation是invalid,一个是有些用户的语言变成德语.
用户名长度影响啊，以及服务器所在地影响

现在距离客户deadline只有2周，但是两周只能完成一部分。有两个人在讨论怎么办，一个人说我们应该延到4周，做完整。另一个说我们应该先实现一部分功能，做个demo，然后再慢慢做。让你选你的看法，支持谁。

有一个项目，deadline是两周，现在解决方案有两种，一种是自己开发，一种是重其他组的服务；自己开发，会比较慢，但是requirement能全部满足，会超过deadline；用其他组的服务，requirement有些不能满足，但是可以在deadline前做完。这时候会给你几个选项，让你去评分。

两个人对话，一个人说要写新的API， 一个人说要用已经存在的，但是功能不全。 Deadline 和满足requirement的区别
有个项目，用已有的api可以更快的做出来，自己写api满足更多requirement，问你选哪个

deadline比预计的更早，需要cross team work但其他team的人没空
要开会，但不同组的人员要么没空，要么不在国内，问你该咋办

customer要求加两个feature，但是这样会超过deadline，问你该怎么做

你负责的东西遇到bug了，给你log，选择可能的原因，这个也简单，可以 排除法

每做完一道题，中间那个任务栏的前面会打上勾. 只要点击那个任务栏的最上面的那个就是新出来的题
3为中评 1-2 为差评 4-5为好评

15 Customer Complaints on Recommendation Service
Priya Senior SDE
We have been receiving a lot of customer complaints over the past three days about problems on pages related to your new Recommendation feature. It is likely being caused by one of the three services you're using. There have been no recent updates to the services and we have ruled out all external factors. This issue is affecting customers, resulting in our product order rate dropping. We really need to solve this as quickly as possible. The reporting team generated an error report(see attached), let me know what you think.
Thanks,
Priya
Based on the data in the Error Rate over Time report, how would you respond? Please select only one response option.
Identify Service 1 as the problem
Identify Service 2 as the problem
Identify Service 3 as the problem
I think Service 1 is the problem, but I would like to see another report to confirm ??
I think Service 2 is the problem, but I would like to see another report to confirm
I think Service 3 is the problem, but I would like to see another report to confirm ?
I dont know which service is the problem. I want to request another report to get additional information.

16 New Product Design
Hi
For the new social network integration product, Ravi is the senior engineer and Jane is the product manager. To help us decide what features to include in the time we have. I asked Ravi to estimate the time to develop each feature and I asked Jane to provide a rough prioritization of each feature. I put their responses in a chart on the Wiki. The file is named "Social Network Integration Design". I'm really swamped today so can you take a look at the chart and give me your recommendation for what features we should include? We have 8 weeks to get something up and running.
Rate the effectiveness of each of the following options
F, G
A, B, D
A, C, D, G
A, D, F
A, C, F
A, C, D, G, H

Feature	time to develop	benefits
A,	1 to 2 weeks	M
B	4 to 6 weeks	H
C	2 to 4 weeks	L
D	2 to 4 weeks	M
E	5 to 9 weeks	H
F	3 to 5 weeks	H
G	2 to 4 weeks	M
H	1 to 2 weeks	L

F G	3-5 H 2-4 M	4
A B D	1-2 M 4-6 H 2-4 M	2
A C D G	1-2 M 2-4 L 2-4 M 2-4 M	6
A D F	1-2 M 2-4 M 3-5 H	1
A C F	1-2 M 2-4 L 3-5 H	3
A C D G H	1-2 M 2-4 L 2-4 M 2-4 M 1-2 L	5

17 Last part of code
Aaron
Here's the last bit of code for the social network project. The code snippet is attached to this email. Sorry this took so long to finish, but it turned out to be more difficult than I thought. Can you just give it a quick check to make sure everything looks okay? I'm leaving for the day, so I'll see you at the demonstration in the morning! Thanks!
What is the problem with the Product.wasPurchasedByUser() method?
Please select only one response option.
It will run slowly because algorithm complexity is O(n^2)
It will run slowly on large datasets because more than one user could have purchased the same product.
It will run slowly because algorithm complexity is O(n^3)
It has performance issues.
有两个关于时间复杂度的不选。剩下两个我不是很确定

What is the most effective way of improving the ShoppingCart() class for the long term? Please select only one response option. ?
ShoppingCart should not override the user property every time new product is added to it.
Make ShoppingCart a property of Product class to improve performance.
Change the design of ShoppingCart by removing ShoppingCart.user and making shopping cart a property of User instead.

Product should have getUser() method so that we know what user has added it to his/her shopping cart.

Please indicate whether the five tests within the ShoppingCartTest() class would pass or fail. Assume each unit test is independent of the others. Please provide a response for each test.
Test1
fail
Assert.assertTrue user1.getDefaultPaymentMethod().getIsDefault();
getDefaultPaymentMethod() 返回null

Test2
pass
PaymentMethod method1 = new PaymentMethod(PaymentMethodType.CREDIT_CARD, "myPersonalCard", true);
Assert.assertSame method1, user1.getDefaultPaymentMethod()

Test3
pass
Assert.assertEquals Bob user2.getName()
fail
Assert.assertEquals alice@foo.com user1.getEmail()
构造函数有四个参数但是函数里只给其中三个赋值了email没有赋值

Test4
pass
Assert.assertSame(thumbnail, myProduct.getImages().get(0))

Test5
fail
Assert.assertTrue(10.75 == product.getPrice());
有一个 setPrice 方法有一个 overload，一个参数是 int，一个参数是 double，所以传 double 会被转型为 int
setPrice 可以是 double 或者integer， 但是 getPrice 是返回 integer

20 Problem with the website
Jacob
Are you still here? I just learned there might be an issue with the website. This could be a real problem but I'm not sure what's going on. No one else is here right now so I could use your help.
If you were to ask Jacob a question about the issue, how would you rate the importance of each of the following questions?
Do any other projects depend on fixing this problem? 4
How long will it take to solve this problem? 5
How does this affect customers? 3 1
Are we receiving complaints from customers? 1 2
How many customers is this affecting? 2
If I help you with this problem, will you help me finish my work today? 6
到下班时间了，有个同事遇到一个bug想让你帮他，和他一起加班。会给你几个选项，选项是你如果要做判断所需要问那个同事的问题，比如，这个bug会不会影响客户？这个bug有没有客户投

诉过？解决这个bug要多长时间？这个bug会不会影响到其他组的工作进度？ 你需要这几个问题评分，哪个最优。

```
coding 60min(70min)
we value fully working code more than partially working but efficient
code
```
1 Round Robin
```
public static float waitingTimeRobin(int[] arrival,int[] run, int q)
import java.util.LinkedList;
import java.util.Queue;
// e.g.
// arrival_time = [0, 1, 4], execution_time = [5, 2, 3], q = 3
// average wait time = (7 - 5) + (5 - 3) + (10 - 7) / 3 = 2.3333333
// q is quantum
public class RoundRobin {
      // SOL 1
      public static float waitingTimeRobin1(int[] arrival, int[] run,
int q) {
            // corner case
            if (arrival == null || run == null || arrival.length !=
run.length) {
                  return 0;
            }
            // use Queue to store Process type elements
            Queue<Process> queue = new LinkedList<Process>();
            int curTime = 0;
            int waitTime = 0;
            // use a int type variable to track the next Process index
            int nextProIdx = 0;
            while (!queue.isEmpty() || nextProIdx < arrival.length) {
                  if (!queue.isEmpty()) {
                        Process cur = queue.poll();
                        // continue suming up the waitTime
                        waitTime += curTime - cur.arriveTime;
                        // based on Round Robin's principle, every round
time is limited within the certain quantum q
                  // if exceed time q, the not finished process should
be forced to be interrupted, and switch to the next process waiting in
the Queue
                        curTime += Math.min(cur.excuteTime, q);
                        // if arrival time of the next process is smaller
than current time
                        // means the next process should be pushed into
the Queue
                        for (int i = nextProIdx; i < arrival.length; i++)
{
                              if (arrival[i] <= curTime) {
                                    queue.offer(new Process(arrival[i],
run[i]));
                                    nextProIdx = i + 1;
                              } else {
                                    break;
                              }
```

```
                    }
                    // push the interrupted process into the tail of
the Queue
                    if (cur.excuteTime > q) {
                         queue.offer(new Process(curTime,
cur.excuteTime - q));
                    }
               } else {
                    // push element in arrival time array and
corresponding element in run time array into Queue
                    queue.offer(new Process(arrival[nextProIdx],
run[nextProIdx]));
                    // at the same update the current time point
                    curTime = arrival[nextProIdx++];
               }
          }
          return (float)waitTime / arrival.length;
     }
     // declare a Process type to store arrival time array and run
time array
     public static class Process {
          int arriveTime;
          int excuteTime;
          Process(int arr, int exc) {
               arriveTime = arr;
               excuteTime = exc;
          }
     }

     public static void main(String[] args) {
          int[] arrival1 = {0, 1, 4};
          int[] run1 = {5, 2, 3};
          int q1 = 3;
          int[] arrival2 = {0, 1, 3, 9};
          int[] run2 = {2, 1, 7, 5};
          int q2 = 2;

          System.out.print(waitingTimeRobin1(arrival2, run2, q2));
     }
}
```

2 Rotate Matrix

```
public class RotateMatrix {
     public static int[][] rotate1(int[][] matrix, int flag) {
          if (matrix == null || matrix.length == 0 ||
matrix[0].length == 0) {
               return null;
          }

          int rows = matrix.length;
          int cols = matrix[0].length;

          int[][] right = new int[cols][rows];
```

```
		int[][] left = new int[cols][rows];

		if (flag == 1) {
			for (int i = 0; i < rows; i++) {
				for (int j = 0; j < cols; j++) {
					right[j][rows - 1 - i] = matrix[i][j];
				}
			}
			return right;
		} else if (flag == 0) {
			for (int i = 0; i < rows; i++) {
				for (int j = 0; j < cols; j++) {
					left[cols - 1 -j][i] = matrix[i][j];
				}
			}
			return left;
		}
		return null;
	}

	public static void printMatrix(int[][] test) {
		for (int i = 0; i < test.length; i++) {
			for (int j = 0; j < test[i].length; j++) {
				System.out.print(" " + test[i][j]);
			}
			System.out.println();
		}
	}

	public static void main(String[] args) {
		int[][] test = {{1, 2, 3, 4, 5},
				{6, 7, 8, 9, 10},
				{11, 12, 13, 14, 15}};
		printMatrix(rotate1(test, 0));
	}
}
```

3 Find minimum path sum of a binary tree

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

```
 10
 /\
5 15

7
/\
3 10
/ /\
1 8 12
public class MinPathSum {
	// time O(n) space O(logn)
	public static int minPath4(TreeNode root) {
		if (root == null) {
```

```
                return 0;
            }

            if (root.left == null && root.right == null) {
                return root.val;
            }
            int minSum = Integer.MAX_VALUE;
            Stack<TreeNode> pathNode = new Stack<TreeNode>();
            Stack<Integer> pathSum = new Stack<Integer>();
            pathNode.push(root);
            pathSum.push(root.val);
            while (!pathNode.isEmpty()) {
                TreeNode curNode = pathNode.pop();
                int curSum = pathSum.pop();
                if (curNode.left == null && curNode.right == null) {
                    if (curSum < minSum) {
                        minSum = curSum;
                    } else {
                        minSum = minSum;
                    }
                }
                if (curNode.right != null) {
                    pathNode.push(curNode.right);
                    pathSum.push(curSum + curNode.right.val);
                }
                if (curNode.left != null) {
                    pathNode.push(curNode.left);
                    pathSum.push(curSum + curNode.left.val);
                }
            }
            return minSum;
        }

        //
        public static int minPath2(TreeNode root) {
            if (root == null) {
                return 0;
            }
            if (root.left != null && root.right == null) {
                return minPath2(root.left) + root.val;
            }
            if (root.left == null && root.right != null) {
                return minPath2(root.right) + root.val;
            }
            return Math.min(minPath2(root.left), minPath2(root.right))
+ root.val;
        }
        //
        public static int minPath1(TreeNode root) {
            if (root == null) {
                return 0;
            }
            if (root.left == null && root.right == null) {
                return root.val;
            }
            if (root.left == null) {
                  return root.val + minPath1(root.right);
            }
            if (root.right == null) {
                  return root.val + minPath1(root.left);
            }
            return root.val + Math.min(minPath1(root.left),
minPath1(root.right));
      }

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

4 insert value into a circle linked-list
给你一个已排序的环形链表（最大节点的下一个节点是最小的节点）的某一个节点（注意不一定是最小的节点）和一个int类型的数，要求把这个数插入到链表里，插入后仍然是一个升序排列的环形链表
思路是先找到最大和最小的节点，然后比较给的数是否比最小的节点还小，如果是，那么插到最大节点后，最小节点前；如果不是，那么继续向后查找直到找到刚好比给的数小的一个节点，把它插到这个节点后面
插入一个新的节点到一个sorted cycle linkedlist，返回新的节点。给的list节点不一定是最小节点
考虑好两个case，一个是正常的，一个是在首位连接处insert
有两点注意一下：
一个是给你的CNode start不一定是最小值的CNode，所以要先找到最小的点；
第二是list里面可能含有duplicate，我一开始看题目说是CNode按ascending order排列，以为没有dup，结果test case过了22/23，然后硬生生想了40min想最后一个是什么case

```
    public static CNode insert7(CNode myList, int n) {
            // corner case if it is null
            if (myList == null) {
                  return new CNode(n);
            } else if (myList.next == myList) { // only one element
                  // add this node
                  myList.next = new CNode(n);
                  // the next's next reference points back to the head
so forms a cycle
                  myList.next.next = myList;
                  // check value and return the smaller one as head
                  if (myList.val < n) {
                        return myList;
                  } else {
                        return myList.next;
```

```
                }
            } else if (n < myList.val) {
                // if it is the smallest element
                // find tail and append
                CNode cur = myList;
                while (cur.next != myList) {
                    cur = cur.next;
                }
                // add value after the tail
                cur.next = new CNode(n);
                // set the appended node's next to original header
                cur.next.next = myList;
                // because this is smallest value! And that's another
reason we return List other than void
                return cur.next;
            }
            // find a position when node.val < n and node.next.val > n
            // or node.next == head (largest)
            CNode cur = myList;
            while (cur.next != myList && cur.next.val <= n) {
                cur = cur.next;
            }
            CNode curNext = cur.next;
            cur.next = new CNode(n);
            // made a copy of original next and assign it here
            cur.next.next = curNext;
            // return header position is unchanged
            return myList;
        }

    // SOL 2
    //  case 1 : pre.val ≤ x ≤ cur.val:
    //  Insert between pre and cur.
    //  case 2 : x is the maximum or minimum value in the list:
    //  Insert before the head. e.g. the head has the smallest value
and its pre.val > head.val.
    //  case 3 : Traverses back to the starting point:
    //  Insert before the starting point.
        public static CNode insert3(CNode head, int x) {
            if (head == null) {
                head = new CNode(x);
                head.next = head;
                return head;
            }
            CNode cur = head;
            CNode pre = null;

            do {
                pre = cur;
                cur = cur.next;
                // case 1
                if (x <= cur.val && x >= pre.val) {
                    break;
                }
                // case 2
                if (pre.val > cur.val && (x < cur.val || x > pre.val))
{
                      break;
                }
          } while (cur != head); // when back to starting point, then
stop. For case 3

          CNode newNode = new CNode(x);
          newNode.next = cur;
          pre.next = newNode;
          return newNode;
     }

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

5 greatest common divisor

```
// Euclid algorithm
// 命题: 对任意 m, n ∊ N, 证明gcd(m,n) = gcd(n, m mod n)
     // SOL 1 recursion
     public static int getGCD1(int[] arr) {
          if (arr == null || arr.length == 0) {
               return 0;
          }
          if (arr.length < 2) {
               return arr[0];
          }

          int rst = arr[0];
          for (int i = 1; i < arr.length; i++) {
               if (arr[i] > 0 && rst > 0) {
                    rst = helper(rst, arr[i]);
               } else {
                    return 0;
               }
          }
          return rst;
     }

     private static int helper(int m, int n) {
          if (m % n == 0) {
               return n;
          } else {
               return helper(n, m % n);
          }
     }
     // time O(n), space O(1)
```

```
	public static int getGCD3(int[] nums) {
		if (nums == null || nums.length == 0 || nums[0] == 0) {
			return 0;
		}

		if (nums.length == 1) {
			return nums[0];
		}
		int rst = nums[0];
		for (int i = 1; i < nums.length; i++) {
			if (nums[i] == 0) {
				return 0;
			}
			// m is the divident
			// rst is the divisor
			int m = nums[i];
			while (m % rst != 0) {
				int tmp = rst;
				rst = m % rst;
				m = tmp;
			}
		}
		return rst;
	}
```

6 Shortest Job First
input：requestTimes[]，每个request到达处理器的时间; durations[] 每个request要处理的持续时间。 两个数组是一一对应的，并已按requestTimes[] 从小到大排序过
example1：
requestTime: [0, 2, 4, 5]
duration: [7, 4, 1, 4]
题目要求是short task first。也就是说先处理p1，处理之后的时间是7，那么之后就处理p3,因为它的处理时间为1，最短。p3之后是p2，最后是p5。这个例子的average waiting time 是4，test case里给的答案.
example2:
requestTime: [0, 1, 3, 9]
duration: [2, 1, 7, 5]
average waiting time 是0.5

```
// O(nlogn)
	public static float SJFaverage2(int[] request, int[] duration) {
		if (request == null || duration == null || request.length !
= duration.length) {
			return 0;
		}
		int index = 0;
		int len = request.length;
		int waitTime = 0;
		int curTime = 0;
		PriorityQueue<process> heap = new
PriorityQueue<process>(new Comparator<process>() {
			public int compare(process p1, process p2) {
```

```
                    if (p1.excTime == p2.excTime) {
                         return p1.arrTime - p2.arrTime;
                    }
                    return p1.excTime - p2.excTime;
               }
          });
          while (!heap.isEmpty() || index < len) {
               if (!heap.isEmpty()) {
                    process cur = heap.poll();
                    waitTime += curTime - cur.arrTime;
                    curTime += cur.excTime;
                    while (index < len && curTime >= request[index])
{
                         heap.offer(new process(request[index],
duration[index++]));
                    }
               } else {
                    heap.offer(new process(request[index],
duration[index]));
                    curTime = request[index++];
               }
          }
          return (float) waitTime / len;
     }

// O(nlogn)
     public static float SJFaverage6(int[] arrive, int[] execute) {
          if (arrive.length == 0) {
               return 0;
          }

          PriorityQueue<process> heap = new
PriorityQueue<process>(arrive.length, new Comparator<process>() {
               public int compare(process p1, process p2) {
                    if (p1.excTime == p2.excTime) {
                         return p1.arrTime - p2.arrTime;
                    } else {
                         return p1.excTime - p2.excTime;
                    }
               }
          });

          int curTime = 0;
          int waitTime = 0;
          int i = 0;
          while (!heap.isEmpty() || i < arrive.length) {
               if (heap.isEmpty()) {
                    int at = arrive[i];
                    while (i < arrive.length && arrive[i] == at) {
                         heap.offer(new process(arrive[i],
execute[i]));
                         curTime = arrive[i];
                         i++;
                    }
```

```
                } else {
                    process p = heap.poll();
                    waitTime += curTime - p.arrTime;
                    curTime += p.excTime;
                    for (; i < arrive.length; i++) {
                        if (arrive[i] <= curTime) {
                            heap.offer(new process(arrive[i],
execute[i]));
                        } else {
                            break;
                        }
                    }
                }
            }
            return (float) waitTime / arrive.length;
        }

        public static class process {
            int arrTime;
            int excTime;
            process(int x, int y) {
                arrTime = x;
                excTime = y;
            }
        }

// O(n^2)
public static float SJFaverage3(int[] request, int[] duration) {
            if (request == null || duration == null || request.length
== 0 || duration.length == 0) {
                return 0;
            }

            int len = request.length;
            int[] end = new int[len];
            int curTime = 0;
            for (int i = 0; i < len; i++) {
                if (i == 0) {
                    curTime = duration[0];
                    end[0] = duration[0];
                } else {
                    int minDuration = Integer.MAX_VALUE;
                    int curProcess = 0;
                    for (int j = 0; j < len; j++) {
                        if (end[j] != 0) {
                            continue;
                        }

                        if (request[j] <= curTime) {
                            if (duration[j] < minDuration) {
                                minDuration = duration[j];
                                curProcess = j;
                            }
                        } else {
```

```
                                        break;
                                }
                        }

                        if (curProcess == 0) {
                                curProcess = i;
                                curTime = request[curProcess];
                        }

                        curTime = curTime + duration[curProcess];
                        end[curProcess] = curTime;
                }
        }

        int waitSum = 0;
        for (int i = 0; i < len; i++) {
                waitSum += end[i] - request[i] - duration[i];
        }

        return (float)waitSum / (float)len;
}
```

7 count miss
output miss count
e.g.
size = 4, input array {1, 2, 3, 4, 5, 4, 1}
1 miss
2 miss
3 miss
4 miss
5 miss replace 1
4 hit 把4提前到第一位
1 miss replace 2

```
// SOL 3
public static int count3(int[] arr, int size) {
        if (arr == null || arr.length < 1) {
                return 0;
        }

        List<Integer> cache = new ArrayList<Integer>();
        int cnt = 0;
        for (int i = 0; i < arr.length; i++) {
                if (cache.contains(arr[i])) {
                        int tmp = arr[i];
                        cache.remove(cache.indexOf(arr[i]));
                        cache.add(tmp);
                } else {
                        cache.add(arr[i]);
                        cnt++;
                }
                if (cache.size() > size) {
                        cache.remove(0);
                }
            }
            return cnt;
        }
        // SOL 1 ?
        public static int count1(int[] arr, int size) {
            if (arr == null) {
                return 0;
            }

            List<Integer> cache = new ArrayList<Integer>();
            int cnt = 0;
            for (int i = 0; i < arr.length; i++) {
                if (cache.contains(arr[i])) {
                    cache.remove(new Integer(arr[i]));
                } else {
                    cnt++;
                    if (size == cache.size()) {
                        cache.remove(0);
                    }
                }
                cache.add(arr[i]);
            }
            return cnt;
        }
        // SOL 2
        public static int count2(int[] arr, int size) {
            if (arr == null || arr.length < 1) {
                return 0;
            }
            List<Integer> list = new ArrayList<Integer>();
            int cnt = 0;
            for (int num : arr) {
                int index = list.indexOf(num);
                if (index == -1) {
                    cnt++;
                    list.add(num);
                    if (list.size() > size) {
                        list.remove(0);
                    }
                } else {
                    list.remove(index);
                    list.add(num);
                }
            }
            return cnt;
        }
```

8 day change(cell growth)

int[] dayChange(int[] cells, int days), cells 数组，有8个元素

变化的具体要求是：一个cell, 如果它左右两边的数一样，那么就将这个数设置为0， 不一样则为1(题目中用inactive和active来描述，后来给出coding的时候用0和1来代替). 因为第一个数和最后一个数只有一个相邻的数，所以默认这个cells[0]的左边及cells[len-1]的右边都为0
e.g.
cells: [1, 0, 0, 0, 0, 1, 0, 0]
days: 1
output: [0, 1, 0, 0, 1, 0, 1, 0]

```
// time O(n*days) space O(n)
public static int[] change5(int[] arr, int days) {
           if (arr == null || arr.length <= 1 || days <= 0) {
                return arr;
           }
           int len = arr.length;
           // preNum represents previous day's list
           int[] preNum = new int[len];
           preNum = arr;
           for (int cnt = 0; cnt < days; cnt++) {
                int[] curNum = new int[len];
                curNum[0] = preNum[1];
                curNum[len - 1] = preNum[len - 2];
                for (int i = 1; i < len - 1; i++) {
                     curNum[i] = preNum[i - 1] ^ preNum[i + 1];
                }
                preNum = curNum;
           }
           return preNum;
      }

public static int[] change3(int[] arr, int days) {
           if (days <= 0) {
                return arr;
           }

           int cnt = 0;
           while (cnt < days) {
                int[] tmp = new int[arr.length];
                for (int i = 0; i < arr.length; i++) {
                     int left;
                     int right;
                     if (i - 1 < 0) {
                          left = 0;
                     } else {
                          left = arr[i - 1];
                     }

                     if (i + 1 > arr.length - 1) {
                          right = 0;
                     } else {
                          right = arr[i + 1];
                     }

                     if (left == right) {
```

```
                    tmp[i] = 0;
                } else {
                    tmp[i] = 1;
                }
            }

            arr = tmp;
            cnt++;
        }
        return arr;
    }
```

9 Find path in maze
输入一个2D array, m*n（0代表墙，1代表路），一只老鼠站在（0，0）处，在迷宫的某处有cheese的（标记为9），问是否存在路径通向吃的。return的1为能达到，0为不能

```
// backtracking
// 1 path 0 wall
    public static int maze3(int[][] grid) {
        int rows = grid.length;
        int cols = grid[0].length;
        int[][] visited = new int[rows][cols];

        if (!dfsHelper(0, 0, grid, visited)) {
            return 0;
        }
        return 1;
    }

    public static boolean dfsHelper(int i, int j, int[][] grid, int[]
[] visited) {
        if (i >= 0 && i < grid.length && j >= 0 && j <
grid[0].length && grid[i][j] == 9) {
            visited[i][j] = 1;
            return true;
        }

        if (isSafe(grid, i, j) == true && visited[i][j] == 0) {
            visited[i][j] = 1;
            if (dfsHelper(i - 1, j, grid, visited)) {
                return true;
            }
            if (dfsHelper(i, j + 1, grid, visited)) {
                return true;
            }
            if (dfsHelper(i + 1, j, grid, visited)) {
                return true;
            }
            if (dfsHelper(i, j - 1, grid, visited)) {
                return true;
            }
            visited[i][j] = 0;
            return false;
        }
```

```
            return false;
      }

      public static boolean isSafe(int[][] grid, int i, int j) {
            if (i >= 0 && i < grid.length && j >= 0 && j <
grid[0].length && grid[i][j] == 1) {
                  return true;
            }
            return false;
      }
// 0 path 1 wall
      public static boolean isSafe2(int[][] grid, int i, int j) {
            if (i >= 0 && i < grid.length && j >= 0 && j <
grid[0].length && grid[i][j] == 0) {
                  return true;
            }
            return false;
      }

      public static void main(String[] args) {
            int[][] grid = {
                        {9, 0, 1, 1, 1},
                        {1, 1, 0, 0, 1},
                        {0, 1, 0, 0, 1},
                        {0, 9, 1, 1, 1}};

            System.out.print(maze3(grid));
      }
}

// iteration bfs
      public static boolean maze2(int[][] grid) {
            if (grid == null) {
                  return false;
            }
            Queue<point> queue = new LinkedList<point>();
            int[][] mark = new int[grid.length][grid[0].length];
            int[] dx = {0, -1, 1, 0};
            int[] dy = {1, 0, 0, -1};
            point start = new point(0, 0);
            queue.offer(start);
            while (!queue.isEmpty()) {
                  point tmp = queue.poll();
                  mark[tmp.x][tmp.y] = 1;
                  if (grid[tmp.x][tmp.y] == 9) {
                        return true;
                  } else if (grid[tmp.x][tmp.y] == 0) {
                        continue;
                  } else {
                        for (int i = 0; i < 4; i++) {
                              if (tmp.x + dx[i] >= 0 && tmp.x + dx[i] <
grid.length && tmp.y + dy[i] >= 0 && tmp.y + dy[i] < grid[0].length) {
                                    if (mark[tmp.x +dx[i]][tmp.y + dy[i]]
== 0) {
```

```
                                    queue.offer(new point(tmp.x + 
dx[i], tmp.y + dy[i]));
                              }
                        }
                  }
            }
        }
        return false;
    }

    public static class point {
        int x;
        int y;
        point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
```

10 Check Subtree
Determine whether T2 is subtree of T1

```
// SOL 1
// time O(nm) or O(n + km) k is the number of occurrences of T2's root 
in T1
// space O(log n + log m) recursion
public static int check1(TreeNode t1, TreeNode t2) {
        if (t2 == null) {
            return 1;
        }
        if (subTree(t1, t2)) {
            return 1;
        } else return -1;
    }

    public static boolean subTree(TreeNode r1, TreeNode r2) {
        if (r1 == null) {
            return false;
        } else if (r1.val == r2.val && matchTree(r1, r2)) {
            return true;
        }
        return (subTree(r1.left, r2) || subTree(r1.right, r2));
    }

    public static boolean matchTree(TreeNode r1, TreeNode r2) {
        if (r2 == null && r1 == null) {
            return true;
        } else if (r1 == null || r2 == null) {
            return false;
        } else if (r1.val != r2.val) {
            return false;
        } else {
            return (matchTree(r1.left, r2.left) && 
matchTree(r1.right, r2.right));
```

```
            }
        }

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

11 Valid Parenthesis

判断是不是valid的parenthesis string，不是的话返回-1，是的话返回valid pair个数

```
import java.util.Stack;
// time O(n), space O(n / 2)
public static int isValid(String s) {
            if (s == null || s.length() == 0) {
                return 0;
            }

            Stack<Character> stack = new Stack<Character>();
            for (int i = 0; i < s.length(); i++) {
                if (stack.isEmpty()) {
                    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));
                }
            }
            if (stack.isEmpty()) {
                return s.length() / 2;
            } else {
                return -1;
            }
        }

public class Solution {
    public boolean isValid(String s) {
        if (s == null || s.length() == 0) {
            return true;
        }

        Stack<Character> stack = new Stack<Character>();
        int len = s.length();
        for (int i = 0; i < len; i++) {
            char c = s.charAt(i);
            if (stack.isEmpty()) {
                if (c == ')' || c == ']' || c == '}') {
                    return false;
                }
                stack.push(c);
                continue;
            }

            if (c == ')' && stack.peek() == '('
            || c == ']' && stack.peek() == '['
            || c == '}' && stack.peek() == '{') {
                stack.pop();
            } else if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            } else {
                return false;
            }
        }
        return stack.isEmpty();
    }
}

public class Solution {
    public boolean isValid(String s) {
        if (s == null || s.length() % 2 == 1) {
            return false;
        }

        HashMap<Character, Character> map = new HashMap<Character,
Character>();
        map.put('(', ')');
        map.put('[', ']');
        map.put('{', '}');
        /*
        map.put('a', '1');
        map.put('b', '2');
        */
        Stack<Character> stack = new Stack<Character>();
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (map.containsKey(c)) {
                stack.push(c);
            } else if (map.containsValue(c)) {
                if (!stack.isEmpty() && map.get(stack.peek()) == c) {
                    stack.pop();
                } else {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    };
}
```

12 reverse second half of linked list

如果mid在正中间，从mid开始翻，不是正中间，从mid+1开始翻

```
public class ReverseList {
	public static JNode reorder2(JNode head) {
		if (head == null || head.next == null || head.next.next == 
null) {
			return head;
		}

		JNode fast = head.next;
		JNode slow = head;
		while (fast.next != null && fast.next.next != null) {
			fast = fast.next.next;
			slow = slow.next;
		}
		JNode pre = slow.next;
		JNode cur = pre.next;
		while (cur != null) {
			pre.next = cur.next;
			cur.next = slow.next;
			slow.next = cur;
			cur = pre.next;
		}
		return head;
	}

	public static JNode reorder3(JNode head) {
		if (head == null || head.next == null) {
			return head;
		}
		JNode pre = findMidPre(head);
		JNode right = pre.next;
		pre.next = null;
		right = reverse(right);
		pre.next = right;
		return head;
	}

	public static JNode reverse(JNode node) {
		if (node == null) {
			return node;
		}
		JNode dummy = new JNode(0);
		while (node != null) {
			JNode tmp = node.next;
			node.next = dummy.next;
			dummy.next = node;
			node = tmp;
		}
		return dummy.next;
	}

	public static JNode findMidPre(JNode node) {
		if (node == null) {
			return node;
		}
		JNode fast = node.next;
```

```
            JNode slow = node;
            while (fast != null && fast.next != null &&
fast.next.next != null) {
                  fast = fast.next.next;
                  slow = slow.next;
            }
            return slow;
      }

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

      public static void main(String[] args) {
            JNode n1 = new JNode(1);
            JNode n2 = new JNode(2);
            JNode n3 = new JNode(3);
            JNode n4 = new JNode(4);
            JNode n5 = new JNode(5);
            JNode n6 = new JNode(6);
            n1.next = n2;
            n2.next = n3;
            n3.next = n4;
            n4.next = n5;
            n5.next = n6;
            n6.next = null;
            JNode x = reorder2(n1);
            while (x != null) {
                  System.out.print(x.val);
                  x = x.next;
            }
      }
}
```

13 same tree

判断两棵树是否相同（结构和值）

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isSameTree1(TreeNode p, TreeNode q) {
        if (p == null && q == null) {
            return true;
        } else if (p == null || q == null) {
            return false;
        } else if (p.val == q.val) {
            if (isSameTree(p.left, q.left) && isSameTree(p.right,
q.right)) {
                return true;
            } else {
                return false;
            }
        }
        return false;
    }

    public boolean isSameTree2(TreeNode p, TreeNode q) {
        if (p == null && q == null) {
            return true;
        }
        if (p == null || q == null) {
            return false;
        }
        Stack<TreeNode> s1 = new Stack<TreeNode>();
        Stack<TreeNode> s2 = new Stack<TreeNode>();
        TreeNode cur1 = p;
        TreeNode cur2 = q;
        while (true) {
            while (cur1 != null && cur2 != null) {
                s1.push(cur1);
                s2.push(cur2);
                cur1 = cur1.left;
                cur2 = cur2.left;
            }
            if (cur1 != null || cur2 != null) {
                return false;
            }
            if (s1.isEmpty() && s2.isEmpty()) {
                break;
            }
            cur1 = s1.pop();
            cur2 = s2.pop();
            if (cur1.val != cur2.val) {
                return false;
            }
            cur1 = cur1.right;
            cur2 = cur2.right;
        }
        return true;
    }
}
```

14 find out number of arithmetic sequence in array

```
public class ArithmeticSeq {
     public static int find(int[] nums) {
          if (nums == null || nums.length < 3) {
               return 0;
```

```
            }
            int left = 0;
            int right = 1;
            int diff = nums[1] - nums[0];
            int cnt = 0;
            while (right < nums.length - 1) {
                  if (diff != nums[right + 1] - nums[right]) {
                        cnt += (right - left - 1) * (right - left) / 2;
                        if (cnt > 1000000000) {
                              return -1;
                        }
                        diff = nums[right + 1] - nums[right];
                        left = right;
                  }
                  right++;
            }
            cnt += (right - left - 1) * (right - left) / 2;
            return cnt > 1000000000 ? -1 : cnt;
      }

      public static void main(String[] args) {
            int[] arr = {2,5,2,3,4,6,8,10,12,9,8,7,6,2,4,8};
            System.out.println(find(arr));
      }
}

15 Amplitude
public class Amplitude {
      // 4
      public static int tmp = 0;

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

            int rst = dfs(root, root.val, root.val);
            return rst;
      }

      public static int dfs(TreeNode root, int min, int max) {
            if (root == null) {
                  return 0;
            }
            min = Math.min(min, root.val);
            max = Math.max(max, root.val);
            if (root.left == null && root.right == null) {
                  tmp = Math.max(tmp, max - min);
            }
            dfs(root.left, min, max);
            dfs(root.right, min, max);
            return tmp;
      }
      // bug 2
```

```
	public static int getAmplitude2(TreeNode root) {
		if (root == null)
			return 0;
		if (root.left == null && root.right == null)
			return root.val;
		Stack<TreeNode> pathNode = new Stack<TreeNode>();
		Stack<int[]> pathMinMax = new Stack<int[]>();
		pathNode.push(root);
		pathMinMax.push(new int[] { root.val, root.val });
		int amplitude = Integer.MIN_VALUE;
		while (!pathNode.empty()) {
			TreeNode crtNode = pathNode.pop();
			if (crtNode.left == null && crtNode.right == null) {
				int[] crtMinMax = pathMinMax.pop();
				amplitude = amplitude > (crtMinMax[1] -
crtMinMax[0]) ? amplitude : (crtMinMax[1] - crtMinMax[0]);
				continue;
			}
			if (crtNode.right != null) {
				int[] crtMinMax = new int[2];
				crtMinMax[0] = crtNode.right.val <
pathMinMax.peek()[0] ? crtNode.right.val : pathMinMax.peek()[0];
				crtMinMax[1] = crtNode.right.val >
pathMinMax.peek()[1] ? crtNode.right.val : pathMinMax.peek()[1];
				pathNode.push(crtNode.right);
				pathMinMax.push(crtMinMax);
			}
			if (crtNode.left != null) {
				int[] crtMinMax = new int[2];
				crtMinMax[0] = crtNode.left.val <
pathMinMax.peek()[0] ? crtNode.left.val : pathMinMax.peek()[0];
				crtMinMax[1] = crtNode.left.val >
pathMinMax.peek()[1] ? crtNode.left.val : pathMinMax.peek()[1];
				pathNode.push(crtNode.left);
				pathMinMax.push(crtMinMax);
			}
			pathMinMax.pop();
		}
		return amplitude;
	}
	// 4
	public static int getAmplitude3(TreeNode root) {
		if (root == null)
			return 0;
		return helper2(root, root.val, root.val);
	}

	private static int helper2(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(helper2(root.left, min, max),
helper2(root.right, min, max));
      }

      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(6);
            TreeNode n2 = new TreeNode(4);
            TreeNode n3 = new TreeNode(8);
            TreeNode n4 = new TreeNode(2);
            n1.left = n2;
            n1.right = n3;
            n2.left = n4;

            System.out.println(getAmplitude2(n1));
      }
 }
```