Leetcode: Record

Start from Tuesday, February 23, 2016

Johnny Liu

Contents

Problem 1

Problem 234 Palindrome Linked List

```
Given a singly linked list, determine if it is a palindrome.

Follow up:
Could you do it in O(n) time and O(1) space?

https://leetcode.com/problems/palindrome-linked-list/
```

Solution 1

Listing 1 shows solution with $O(n)runtime, O(n)space$.

The key idea is to use a stack storing elements from List, where popping elements out from a stack is exactly reversing the List.(The stack may use $O(n)space$.)

Listing 1: Thought 1

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isPalindrome(ListNode head) {
        int len = getLength(head);
        if (len == 0 || len == 1){
            return true;
        }
        Stack<ListNode> stack = new Stack<ListNode>();
        int idx = 0;
        ListNode cur = head;
        boolean flag = true;
        while (cur != null){
            stack.push(cur);
            if (len % 2 == 0 && idx == len / 2 - 1){
                break;
            }
            if (len % 2 == 1 && idx == len / 2){
                stack.pop();
                break;
            }
            idx ++;
            cur = cur.next;
        }
        cur = cur.next;
        while (cur != null){
            ListNode compare = stack.pop();
            if (compare.val != cur.val){
                return false;
```

Problem 1 continued on next page...

```
                }
                cur = cur.next;
            }
            return true;
        }

        private int getLength(ListNode head){
            int len = 0;
            ListNode cur = head;
            while (cur != null){
                len ++;
                cur = cur.next;
            }
            return len;
        }

}
```

Solution 2

Listing 2 shows Solution 2 with $\boldsymbol{O(n)runtime, O(n)space}$.

Instead of using stack, this solution finds element in the middle of whole List, reverse the second half and compare the elements with those in the first half.
Some key ideas:

1. The key idea of getting the middle element is to keep two iteration pointers, one with 1 step while another with 2 steps every time they iterate.
2. Take care of how to reverse a Linked-list in-place.

Listing 2: Thought2

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
        public boolean isPalindrome(ListNode head) {
            if (head == null || head.next == null){
                return true;
            }
            ListNode cur = head;
            // get middle
            ListNode mid = getMid(head);
            // reverse from mid to end
            mid = reverse(mid);
            // check equations
            while (mid != null){
                if (mid.val != cur.val){
                    return false;
```

```
                }
                mid = mid.next;
                cur = cur.next;
            }
            return true;
        }

        public ListNode getMid(ListNode head){
            ListNode slow = head;
            ListNode fast = head;
            while (fast.next != null && fast.next.next != null){
                slow = slow.next;
                fast = fast.next.next;
            }
            return slow.next;
        }

        public ListNode reverse(ListNode head){
            if (head.next == null){
                return head;
            }
            ListNode p,q,r;
            p = head;
            q = p.next;
            r = q.next;
            p.next = null;
            while (q.next != null){
                q.next = p;
                p = q;
                q = r;
                r = q.next;
            }
            q.next = p;
            return q;
        }

}
```

Problem 2

Problem 35 Search Insert Position

```
Given a sorted array and a target value, return the index if the target is found.

If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.
[1,3,5,6], 5 -> 2
[1,3,5,6], 2 -> 1
[1,3,5,6], 7 -> 4
[1,3,5,6], 0 -> 0

https://leetcode.com/problems/search-insert-position/
```

Solution

Basic Binary Search algorithm

Listing 3: Solution

```
public class Solution {
     public int searchInsert(int[] nums, int target) {
     if (target <= nums[0]){
          return 0;
     }
     if (target > nums[nums.length - 1]){
          return nums.length;
     }
        int l = 0 , r = nums.length - 1;
        while (l < r){
          int mid = (l + r) / 2;
          if (target == nums[mid]){
               return mid;
            }
          if (mid == l && target != nums[mid]){
                return mid + 1;
          }
          if (target > nums[mid]){
               l = mid;
          }
          else{
               r = mid;
          }
        }
        return -1;
    }
}
```

Problem 3

Problem 121 Best Time to Buy and Sell Stock

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

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

https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
```

Solution 1

Brute force algorithm, list all possibilities, $O(n^2)runtime$, undoubtedly Time Limit Exceed.

Listing 4: Solution 1

```
public class Solution {
     // TLE
    public int maxProfit(int[] prices) {
     int max = 0;
        for (int i = 0 ; i < prices.length ; i++){
          for (int j = i + 1 ; j < prices.length ; j ++){
               if (prices[j] - prices[i] > max){
                    max = prices[j] - prices[i];
               }
          }
        }
        return max;
    }
}
```

Solution 2

Maintain a minimum value until just before current index, so that we may get a better result when selling current stock while minimum price bought, $O(n)runtime$.

Listing 5: Solution 2

```
public class Solution {
    public int maxProfit(int[] prices) {
     int max = 0;
     if (prices.length == 0){
          return 0;
     }
     int curMin = prices[0];
        for (int i = 1 ; i < prices.length ; i++){
          int profit = prices[i] - curMin;
          if (profit > max){
               max = profit;
          }
          if (prices[i] < curMin){
               curMin = prices[i];
          }
```

```
            }
            return max;
        }
}
```

Problem 4

Problem 122 Best Time to Buy and Sell Stock II

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

Design an algorithm to find the maximum profit.
 You may complete as many transactions as you like
  (ie, buy one and sell one share of the stock multiple times).
   However, you may not engage in multiple transactions
    at the same time (ie, you must sell the stock before you buy again).

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/
```

Solution

Don't hesitate to use Greedy algorithm to solve the problem.

Listing 6: Solution

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

Problem 5

Problem 123 Best Time to Buy and Sell Stock III

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

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

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

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/
```

Solution

This solution here is a variant of Problem "Best Time to Buy and Sell Stock". We use each day as split point, respectively calculating maximum profits before and after the i^{th} day, then compare all of them to get the real maximum profit.

BTW, when we look at the 4th problem of this series, the count of transaction is extended to k, so that we can just substitute 2 into k to get the result of this problem. The solution will be demonstrated in next Problem.

However, simply apply such process will get TLE. We need to do some pre-processing. We may calculate maximum profit of [0,1],[0,2].... and [n-1,n],[n-2,n] so that we can reduce the complexity to $O(n)$

Listing 7: Solution

```
public class Solution {
    public int maxProfit(int[] prices) {
        if (prices.length == 0){
            return 0;
        }
        int maxSum = 0;
        int[] maxProfit = new int[prices.length];
        int[] maxProfitRev = new int[prices.length];

        int max = 0;
        int min = prices[0];
        for (int i = 1 ; i < prices.length ; i ++){
            int diff = prices[i] - min;
            max = Math.max(diff, max);
            if (prices[i] < min){
                min = prices[i];
            }
            maxProfit[i] = max;
        }
        max = 0;
        int high = prices[prices.length - 1];
        for (int i = prices.length - 2 ; i >= 0 ; i--){
            int diff = high - prices[i];
            max = Math.max(max, diff);
            if (prices[i] > high){
                high = prices[i];
```

```
                }
                maxProfitRev[i] = max;
            }
            for (int i = 0 ; i < prices.length ; i ++){
                maxSum = Math.max(maxSum, maxProfit[i] + maxProfitRev[i]);
            }
            return maxSum;
        }
}
```

Problem 6

Problem 188 Best Time to Buy and Sell Stock IV

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

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

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

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/
```

Solution

It's not a conventional way of using DP. Detailed Description can be seen below.

题目的关键是要写出下面的动态转移方程:
local[i][j]=max(global[i-1][j-1]+max(diff,0),local[i-1][j]+diff),
global[i][j]=max(local[i][j],global[i-1][j]),
而这个方程的具体理解是这样的。
首先 global 比较简单，不过是不断地和已经计算出的 local 进行比较，把大的保存在 global 中。
然后看 local.关键是要理解 local 的定义，local[i][j]表示，前 i 天进行了 j 次交易，并且第 i 天进行了第 j 次交易的最大利润，所以 local[i][j]中必然有一次交易，也就是当近一次交易，发生在第 i 天。 local 由两个部分的比较完成。

第一部分是，global[i-1][j-1]+max(diff,0), 表示的就是，前面把之前的 j - 1 次交易，放在之前的 i - 1 天，然后让第 i 天来进行第 j 次交易，那么加入此时 diff(price[i] - price[i - 1])大于零，那么正好可以可借助这次交易的机会增长里利润(利润= diff)，否则的话，如果 diff 小于零，那就在第 i 天当天进行一次买卖，凑一次交易的次数，但是产生利润为 0.
第二部分是, local[i-1][j]+diff， 这里的 local[i-1][j]表示的是，前面 j 次交易在第 i -1 天就已经完成了，可是因为说了 local[a][b]一定要表达在第 a 天完成了 b 次交易的最大利润，所以就需要强制使得交易在第 i 天发生，为了实现这一点，只需要在 local[i - 1][j]的基础上，加上 diff (= price[i] - price[i - 1])就可以了。如果 diff < 0 那也没有办法，因为必须满足 local 的定义。接下来算 global 的时候，总会保证取得一个更大的值。

关于正道题目的全文的分析，Code Ganker 的还挺不错。 下面全文摘抄作者的分析:
"这道题是 Best Time to Buy and Sell Stock 的扩展，现在我们最多可以进行两次交易。我们仍然使用动态规划来完成，事实上可以解决非常通用的情况，也就是最多进行 k 次交易的情况。

这里我们先解释最多可以进行 k 次交易的算法，然后最多进行两次我们只需要把 k 取成 2 即可。我们还是使用"局部最优和全局最优解法"。我们维护两种量，一个是当前到达第 i 天可以最多进行 j 次交易，最好的利润是多少（global[i][j]），另一个是当前到达第 i 天，最多可进行 j 次交易，并且最后一次交易在当天卖出的最好的利润是多少（local[i][j]）。下面我们来看递推式。

全局的比较简单，
global[i][j]=max(local[i][j],global[i-1][j]),
也就是去当前局部最好的，和过往全局最好的中大的那个（因为最后一次交易如果包含当前天一定在局部最好的里面，否则一定在过往全局最优的里面）。
对于局部变量的维护，递推式是
local[i][j]=max(global[i-1][j-1]+max(diff,0),local[i-1][j]+diff),
也就是看两个量，第一个是全局到 i-1 天进行 j-1 次交易，然后加上今天的交易，如果今天是赚钱的话（也就是前面只要 j-1 次交易，最后一次交易取当前天），第二个量则是取 local 第 i-1 天 j 次交易，然后加上今天的差值（这里因为 local[i-1][j]比如包含第 i-1 天卖出的交易，所以现在变成第 i 天卖出，并不会增加交易次数，而且这里无论 diff 是不是大于 0 都一定要加上，因为否则就不满足 local[i][j]必须在最后一天卖出的条件了）。

上面的算法中对于天数需要一次扫描，而每次要对交易次数进行递推式求解，所以时间复杂度是 O(n*k)，如果是最多进行两次交易，那么复杂度还是 O(n)。空间上只需要维护当天数据皆可以，所以是 O(k)，当 k=2，则是 O(1)。"

(Honestly, I don't quite understand the solution here.

Listing 8: Solution

```
public class Solution {
    public int maxProfit(int k, int[] prices) {
        if (prices.length == 0){
            return 0;
        }
        // for those special input data
        if (k > prices.length){
            int sum = 0;
            for (int i = 1 ; i < prices.length ; i++){
                if (prices[i] > prices[i - 1]){
                    sum += prices[i] - prices[i - 1];
                }
            }
```

```
            return sum;
        }
        else{
            int[][] total = new int[prices.length][k + 1];
            int[][] part = new int[prices.length][k + 1];
            for (int i = 1 ; i < prices.length ; i++){
                int diff = prices[i] - prices[i - 1];
                for (int j = k ; j >= 1 ; j--){
                    part[i][j] = Math.max(total[i - 1][j - 1]
                     + Math.max(0, diff), part[i - 1][j] + diff);
                    total[i][j] = Math.max(part[i][j], total[i - 1][j]);
                }
            }

            return total[prices.length - 1][k];
        }
    }
}
```

Problem 7

Problem 309 Best Time to Buy and Sell Stock with Cooldown

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

Design an algorithm to find the maximum profit.
 You may complete as many transactions as you like
  (ie, buy one and sell one share of the stock multiple times)
   with the following restrictions:

You may not engage in multiple transactions at the same time
 (ie, you must sell the stock before you buy again).
After you sell your stock, you cannot buy stock on next day.
 (ie, cooldown 1 day)
Example:

prices = [1, 2, 3, 0, 2]
maxProfit = 3
transactions = [buy, sell, cooldown, buy, sell]

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/
```

Solution

Another problem I didn't understand.....

分析

因为当前日期买卖股票会受到之前日期买卖股票行为的影响，首先考虑到用DP解决。

这道题比较麻烦的是有个cooldown的限制，其实本质也就是买与卖之间的限制。对于某一天，股票有三种状态: buy, sell, cooldown, sell与cooldown我们可以合并成一种状态，因为手里最终都没股票，最终需要的结果是sell，即手里股票卖了获得最大利润。所以我们可以用两个DP数组分别记录当前持股跟未持股的状态。然后根据题目中的限制条件，理清两个DP数组的表达式。

对于当天最终未持股的状态，最终最大利润有两种可能，一是今天没动作跟昨天未持股状态一样，二是昨天持股了，今天卖了。所以我们只要取这两者之间最大值即可，表达式如下：

```
sellDp[i] = Math.max(sellDp[i - 1], buyDp[i - 1] + prices[i]);
```

对于当天最终持股的状态，最终最大利润有两种可能，一是今天没动作跟昨天持股状态一样，二是前天还没持股，今天买了股票，这里是因为cooldown的原因，所以今天买股要追溯到前天的状态。我们只要取这两者之间最大值即可，表达式如下：

```
buyDp[i] = Math.max(buyDp[i - 1], sellDp[i - 2] - prices[i]);
```

最终我们要求的结果是

```
sellDp[n - 1] 表示最后一天结束时手里没股票时的累积最大利润
```

当然，这里空间复杂度是可以降到O(1)的，具体见第二种代码实现。

Listing 9: Solution

```java
public class Solution {
    public int maxProfit(int[] prices) {
        if (prices.length == 0){
            return 0;
```

```
        }
        int sell[] = new int[prices.length];
        int buy[] = new int[prices.length];

        sell[0] = 0;
        buy[0] = -prices[0];

        for (int i = 1 ; i < prices.length ; i++){
            sell[i] = Math.max(sell[i - 1], buy[i - 1] + prices[i]);
            if (i == 1){
                buy[i] = Math.max(buy[i - 1], -prices[i]);
            }
            else{
                buy[i] = Math.max(buy[i - 1], sell[i - 2] - prices[i]);
            }
        }

        return sell[prices.length - 1];
    }
}
```

Problem 8

Problem 9 Palindrome Number

```
Given a string, determine if it is a palindrome, considering
 only alphanumeric characters and ignoring cases.

For example,
"A man, a plan, a canal: Panama" is a palindrome.
"race a car" is not a palindrome.

Note:
Have you consider that the string might be empty? This is a good question to ask
 during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

https://leetcode.com/problems/valid-palindrome/
```

Solution

Though it's just a easy string-handling problem with not many edge cases, I made several mistakes:

- Ignoring the case problems of letters, should lowercase characters in sentence.
- Forgot to check boundaries, especially when all characters are non-alphanumeric, there will be an IndexOutOfBoundException.

Listing 10: Solution

```
public class Solution {
    public boolean isPalindrome(String s) {
        if (s == null || s.length() == 0){
            return true;
        }
        int i = 0, j = s.length() - 1;
        while (i < j){
            while (i < s.length() && !Character.isLetter(s.charAt(i))
                && !Character.isDigit(s.charAt(i))){
                i ++;
            }
            while (j > 0 && !Character.isLetter(s.charAt(j))
                && !Character.isDigit(s.charAt(j))){
                j --;
            }
            if (i >= j){
                return true;
            }

            if (Character.toLowerCase(s.charAt(i)) !=
                 Character.toLowerCase(s.charAt(j))){
                return false;
            }
            i ++;
            j --;
        }
```

```
            return true;
        }
    }
```

Problem 9

Problem 206 Reverse Linked List

```
Reverse a singly linked list.

Hint:
A linked list can be reversed either iteratively or recursively.
 Could you implement both?


https://leetcode.com/problems/reverse-linked-list/
```

Solution 1

Non-recursive solution, basically maintains 3 pointers, 'cur', 'next', next node of 'next', see details in Listing 11

Listing 11: Solution 1

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseList(ListNode head) {
        if (head == null || head.next == null){
            return head;
        }
        ListNode cur = head;
        ListNode next = cur.next;
        cur.next = null;

        while (next.next != null){
            ListNode tmp = next.next;
            next.next = cur;
            cur = next;
            next = tmp;
        }
        next.next = cur;
        return next;
    }
}
```

Solution 2

Recursive solution.

Listing 12: Solution 2

```
/**
```

```java
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseList(ListNode head) {
        if (head == null || head.next == null){
            return head;
        }
        ListNode nextNode = head.next;
        head.next = null;
        ListNode retNode = reverseList(nextNode);
        nextNode.next = head;
        return retNode;
    }
}
```

Problem 10

92 Reverse Linked List II

```
Reverse a linked list from position m to n.
 Do it in-place and in one-pass.

For example:
Given 1->2->3->4->5->NULL, m = 2 and n = 4,

return 1->4->3->2->5->NULL.

Note:
Given m, n satisfy the following condition:
1 $<=$ m $<=$ n $<=$ length of list.

https://leetcode.com/problems/reverse-linked-list-ii/
```

Solution

Dummy node is a excellent aid when dealing with problems involving Linked-list.
The main idea of my solution is to firstly use dummy node to avoid talking about conditions when the reversion starts from the 1st element, which may need a lot of code to deal with empty head nodes; secondly iterate to the element where the reversion starts, record its previous node as the tail of 1st part of the whole 3 parts; reverse until the index reaches n; reconnect the tails of each parts by assigning .next attribute.

Listing 13: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseBetween(ListNode head, int m, int n) {
        if (n == m){
            return head;
        }
        int idx = 0;
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode cur = dummy;
        ListNode prev = cur;
        while (cur.next != null && idx < m) {
            prev = cur;
            cur = cur.next;
            idx++;
        }
        ListNode tail1 = prev;
        ListNode tail2 = cur;
        ListNode next = cur.next;
```

```
        while (cur.next != null && idx < n){
            ListNode tmp = next.next;
            next.next = cur;
            cur = next;
            next = tmp;
            idx++;
        }
        tail2.next = next;
        tail1.next = cur;
        return dummy.next;
    }
}
```

Problem 11

19 Remove Nth Node From End of List

```
Given a linked list, remove the nth node from the end of list and return its head.

For example,

   Given linked list: 1->2->3->4->5, and n = 2.

   After removing the second node from the end, the linked list becomes 1->2->3->5.
Note:
Given n will always be valid.
Try to do this in one pass.

https://leetcode.com/problems/remove-nth-node-from-end-of-list/
```

Solution

We can solve this problem by calculating its length first and get index of deleted node by length-n. However, this is not a solution in one-pass;

A tactful idea is that we first iterate one pointer n times, then iterate pointer describe above together with a pointer from head of the list, when the 1st point reaches the end of list, the 2nd point stays just at the node we want to remove. See 14

Listing 14: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode p1 = head;
        ListNode p2 = head;
        while (n-- > 0) {
            p1 = p1.next;
        }
        if (p1 == null){
            return head.next;
        }
        ListNode prev = p2;
        while (p1 != null) {
            p1 = p1.next;
            prev = p2;
            p2 = p2.next;
        }
        prev.next = p2.next;
        return head;
    }
}
```

Problem 12

203 Remove Linked List Elements

```
Remove all elements from a linked list of integers that have value val.

Example
Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6
Return: 1 --> 2 --> 3 --> 4 --> 5

https://leetcode.com/problems/remove-linked-list-elements/
```

Solution

An easy problem, just be aware of some edge cases such as inputting an empty list or a list full of values to be removed.

Listing 15: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode removeElements(ListNode head, int val) {
        while (head != null && head.val == val){
            head = head.next;
        }
        if (head == null || head.next == null)
            return head;
        ListNode cur = head.next;
        ListNode prev = head;
        while (cur != null){
            while (cur != null && cur.val == val){
                prev.next = cur.next;
                cur = cur.next;
            }
            if (cur == null){
                return head;
            }
            prev = cur;
            cur = cur.next;
        }
        return head;
    }
}
```

Problem 13

83 Remove Duplicates from Sorted List

```
Given a sorted linked list,
 delete all duplicates such that each element appear only once.

For example,
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

https://leetcode.com/problems/remove-duplicates-from-sorted-list/
```

Solution

An easy problem, since list is sorted, just check if value of neighbor nodes are equal.

Listing 16: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null){
            return head;
        }
        ListNode cur = head;
        ListNode next = cur.next;
        while (next != null){
            while (next != null && cur.val == next.val){
                cur.next = next.next;
                next = next.next;
            }
            if (next == null)
                return head;
            cur = next;
            next = next.next;
        }
        return head;
    }
}
```

Problem 14

82 Remove Duplicates from Sorted List II

```
Given a sorted linked list, delete all nodes that have duplicate numbers,
 leaving only distinct numbers from the original list.

For example,
Given 1->2->3->3->4->4->5, return 1->2->5.
Given 1->1->1->2->3, return 2->3.

https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/
```

Solution

My solution is to use 'prev' and 'cur' pointer to deal with such problem. 'prev' maintains lists which already be processed to have only distinct values; 'cur' is used to skip all adjacent elements whose values are equal.

There is another interesting train of thought, where distinct value means the value of element is not equal to its neighbors(both left side and right side.).

Listing 17: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null){
            return head;
        }
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode former = dummy;
        ListNode cur = dummy.next;
        ListNode next = cur.next;
        while(next != null){
            if (cur.val == next.val){
                int val = cur.val;

                while (cur != null && cur.val == val){
                    cur = cur.next;
                }
                if (cur == null){
                    former.next = null;
                    return dummy.next;
                }
                else {
                    next = cur.next;
                }
            }
```

Problem 14 continued on next page...

```
                else{
                    former.next = cur;
                    former = cur;

                    cur = cur.next;
                    next = cur.next;
                }
            }
            former.next = cur;
            return dummy.next;
        }
}
```

Problem 15

328 Odd Even Linked List

```
Given a singly linked list,
 group all odd nodes together followed by the even nodes.
Please note here we are talking about
 the node number and not the value in the nodes.

You should try to do it in place. The program should run in
 O(1) space complexity and O(nodes) time complexity.

Example:
Given 1->2->3->4->5->NULL,
return 1->3->5->2->4->NULL.
Note:
The relative order inside both the even and odd groups should remain as it was
 in the input.
The first node is considered odd, the second node even and so on ...

https://leetcode.com/problems/odd-even-linked-list/
```

Solution

An easy problem, just to be aware of conditions when length of list is odd and even.

Listing 18: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode oddEvenList(ListNode head) {
        if (head == null || head.next == null){
            return head;
        }
        ListNode cur = head;
        ListNode next = cur.next;
        ListNode evenHead = next;
        while (next != null){
            cur.next = next.next;
            if (cur.next == null){
                cur.next = evenHead;
                return head;
            }
            next.next = cur.next.next;
            cur = cur.next;
            next = next.next;
        }
        cur.next = evenHead;
        return head;
```

Problem 15 continued on next page...

```
        }
30  }
```

Problem 16

237 Delete Node in a Linked List

```
Write a function to delete a node (except the tail) in a singly linked list,
 given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node
 with value 3, the linked list should become 1 -> 2 -> 4 after calling
  your function.

https://leetcode.com/problems/delete-node-in-a-linked-list/
```

Solution

An easy but weird problem, must use deep copy to finish the task, since OJ System will check the value of node with current index.

Listing 19: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void deleteNode(ListNode node) {
        //input check
        if(node==null) return;
        node.val = node.next.val;
        node.next = node.next.next;
    }
}
```

Problem 17

160 Intersection of Two Linked Lists

```
Write a program to find the node at which
 the intersection of two singly linked lists begins.


For example, the following two linked lists:

A:          a1 -> a2
                   ->
                     c1 -> c2 -> c3
                   ->
B:     b1 -> b2 -> b3
begin to intersect at node c1.


Notes:

If the two linked lists have no intersection at all, return null.
The linked lists must retain their original structure after the function returns.
You may assume there are no cycles anywhere in the entire linked structure.
Your code should preferably run in O(n) time and use only O(1) memory.

https://leetcode.com/problems/intersection-of-two-linked-lists/
```

Solution

Solution of this problem needs tactics.

Brute force solution is definitely not good enough. The idea of $\boldsymbol{O(n)runtime}$ and $\boldsymbol{O(1)space}$ solution is similar to the idea in *Problem 19 Remove Nth Node From End of List*, which needs to iterate the longer list several times(equals to the difference between two length), then iterate both lists since they are of same length currently.

Listing 20: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
     public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        if (headA == null || headB == null){
            return null;
        }
        int lenA = 1 ; int lenB = 1;
        ListNode curA = headA;
```

```
        ListNode curB = headB;
        while (curA.next != null){
            lenA ++;
            curA = curA.next;
        }
        while (curB.next != null){
            lenB ++;
            curB = curB.next;
        }
        if (curA.val != curB.val){
            return null;
        }
        ListNode longer, shorter;
        int diff = Math.abs(lenA - lenB);
        if (lenA > lenB){
            longer = headA;
            shorter = headB;
        }
        else{
            longer = headB;
            shorter = headA;
        }
        boolean found = false;
        ListNode inter = null;
        while (diff --> 0){
            longer = longer.next;
        }
        while(longer != null){
            if (!found && longer.val == shorter.val){
                found = true;
                inter = longer;
            }
            if (longer.val != shorter.val){
                found = false;
                inter = null;
            }
            longer = longer.next;
            shorter = shorter.next;
        }
        return inter;
    }
}
```

Problem 18

21 Merge Two Sorted Lists

```
Merge two sorted linked lists and return it as a new list.
 The new list should be made by splicing together the nodes of the first two lists.

https://leetcode.com/problems/merge-two-sorted-lists/
```

Solution

This problem indicates the importance of dummy node in solving linked list problems.

Listing 21: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(-1);
        ListNode cur = dummy;
        if (l1 == null){
            return l2;
        }
        if (l2 == null){
            return l1;
        }
        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;
        }
        if (l1 == null){
            cur.next = l2;
        }
        if (l2 == null){
            cur.next = l1;
        }
        return dummy.next;
    }
}
```

Problem 19

23 Merge k Sorted Lists

```
Merge k sorted linked lists and return it as one sorted list.
 Analyze and describe its complexity.

https://leetcode.com/problems/merge-k-sorted-lists/
```

Solution 1 & 2

Solution 1 is Brute force. With $O(nk^2)runtimeandO(1)space$. ($nk^2$ comes from 2n + 3n + ... + kn). We may skip Solution 1 here.

Solution 2 makes use of heap. Keep a heap with min root, pop one , link it into result, push its next node. Since Heap has a complexity of log(k) runtime with nk elements, the whole process has a $O(knlogk)runtimeandO(k)space$ complexity.

Listing 22: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    private static final Comparator<ListNode> listComparator
     = new Comparator<ListNode>(){
      @Override
      public int compare(ListNode x,  ListNode y){
          return x.val - y.val;
      }
    };

    public ListNode mergeKLists(ListNode[] lists) {
        if (lists.length == 0){
            return null;
        }
        Queue<ListNode> queue
        = new PriorityQueue<ListNode>(lists.length, listComparator);
        for (ListNode list : lists){
            if (list != null){
                queue.offer(list);
            }
        }

        ListNode dummy = new ListNode(-1);
        ListNode head = dummy;
        while (!queue.isEmpty()){
            ListNode curNode = queue.poll();
            head.next = curNode;
            head = head.next;
```

Problem 19 continued on next page...

```
            if (curNode.next != null){
                queue.offer(curNode.next);
            }
        }

        return dummy.next;
    }
}
```

Solution 3

Solution 3 is still of $O(knlogk) runtime$ but with $O(1) space$.

The number of lists reduces from:

$k- > k/2- > k/4- > ...- > 2- > 1$

When the size of lists increases from :

$n- > 2n- > 4n- > ...- > 2^{logk}n$

So the runtime complexity is:

$kn + k/2 * 2n + ... + 2^{logk}n = nklogk$

Listing 23: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2){
        ListNode dummy = new ListNode(-1);
        ListNode head = dummy;
        ListNode cur1 = list1;
        ListNode cur2 = list2;

        while (cur1 != null && cur2 != null){
            if (cur1.val < cur2.val){
                head.next = cur1;
                cur1 = cur1.next;
                head = head.next;
            }
            else{
                head.next = cur2;
                cur2 = cur2.next;
                head = head.next;
            }
        }

        while (cur1 != null){
            head.next = cur1;
            cur1 = cur1.next;
            head = head.next;
        }
```

Problem 19 continued on next page...

```
        while (cur2 != null){
            head.next = cur2;
            cur2 = cur2.next;
            head = head.next;
        }

        return dummy.next;
    }

    public ListNode mergeKLists(ListNode[] lists) {
        if (lists.length == 0){
            return null;
        }
        int begin = 0 , end = lists.length - 1;

        while (end > 0){
            begin = 0;
            while (begin < end){
                // two pointers
                lists[begin] = mergeTwoLists(lists[begin], lists[end]);
                begin ++;
                end --;
            }
        }

        return lists[0];
    }
}
```

Problem 20

143 Reorder List

```
Given a singly linked list L: L0->L1->...->Ln-1->Ln,
reorder it to: L0->Ln->L1->Ln-1->L2->Ln-2->...

You must do this in-place without altering the nodes' values.

For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.

https://leetcode.com/problems/reorder-list/
```

Solution

The train of thought is somewhat straightforward, just find the middle element, reverse the latter half, and reconnect elements in first half and second half. But not coding in IDE to solve a problem of Medium difficulty is impossible to me currently...

Listing 24: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
public void reorderList(ListNode head) {
        if (head == null || head.next == null|| head.next.next == null){
            return;
        }
        ListNode slow = head;
        ListNode fast = head;
        ListNode prev = slow;
        while (fast != null && fast.next != null){
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }
        prev.next = null;
        // reverse latter half
        ListNode cur = slow;
        ListNode next = cur.next;
        cur.next = null;
        while (next != null){
            ListNode tmp = next.next;
            next.next = cur;
            cur = next;
            next = tmp;
        }
        // cur is the reversed list
        ListNode p = head;
```

Problem 20 continued on next page...

```
        ListNode q = cur;
        while (p.next != null){
            ListNode r = p.next;
            ListNode s = q.next;
            p.next = q;
            q.next = r;
            p = r;
            q = s;
        }
        p.next = q;
    }
}
```

Problem 21

326 Power of Three

```
Given an integer, write a function to determine if it is a power of three.

Follow up:
Could you do it without using any loop / recursion?

https://leetcode.com/problems/power-of-three/
```

Solution

Using recursion or loop is extremely easy, but not using both is a challenge. Note that $log_3(n) = \frac{log(n)}{log(3)}$ and it will return a integer if n is power of 3. Then we should just check if div and $Math.round(div)$ has a difference less than 0.000000001(If we just check equation we may find 243 returns false because of precision problem in Java).

Listing 25: Solution

```
public class Solution {
    public boolean isPowerOfThree(int n) {
        double div = Math.log(n) / Math.log(3);
        double diff = Math.abs(div - Math.round(div));
        if (diff <= 0.0000000001){
            return true;
        }else{
            return false;
        }
    }
}
```

Problem 22

231 Power of Two

```
Given an integer, write a function to determine if it is a power of two.

Follow up:
Could you do it without using any loop / recursion?

https://leetcode.com/problems/power-of-two/
```

Solution 1

Solution 1 is nearly the same as solution in Pro 326, *Power of Two*, just change 3 into 2.

Listing 26: Solution 1

```
public class Solution {
    public boolean isPowerOfTwo(int n) {
        double div = Math.log(n) / Math.log(2);
        double diff = Math.abs(div -  Math.round(div));
        if (diff <= 0.0000000001)
            return true;
        else
            return false;
    }
}
```

Solution 2

This solution is interesting. Speaking of 2(or 4,16, etc, we can do sqrt calculation first to make problem familiar), we may use bit operators to solve it. Both checking n-1 is all 1 in Binary or in form of 1000000 in Binary are fine.

Listing 27: Solution 2

```
public class Solution {
    public boolean isPowerOfTwo(int n) {
     if (n <= 0)
          return false;
        int cur = (int)n - 1;
        while (cur != 0){
          if ((cur & 1) != 1)
               return false;
          cur = (cur >> 1);
        }
        return true;
    }
}
// OR JUST!!!
public class Solution {
    public boolean isPowerOfTwo(int n) {
        return (n != 0
         && n != Integer.MIN_VALUE
          && (n & (n - 1)) == 0);
    }
}
```

Problem 23

342 Power of Four

```
Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example:
Given num = 16, return true. Given num = 5, return false.

Follow up: Could you solve it without loops/recursion?

https://leetcode.com/problems/power-of-four/
```

Solution 1

First identify whether it is power of 2(simply using n&(n-1)==0), then if it is , the form of the number must be like 10....0, check whether there are even number of 0 behind the heading 1.

However, I'm still using a loop.

Listing 28: Solution 1

```
public class Solution {
    public boolean isPowerOfFour(int num) {
        if (((num & (num - 1)) != 0) || num == 0){
            return false;
        }
        while ((num & 3) == 0){
            num >>= 2;
        }
        return ((num & 3) == 1);
    }
}
```

Solution 2

Still, check if it is power of 2. Then use 0x55555555(10101010101010101010101010101010101 in binary) to check if the single 1 only appears at odd position.

Listing 29: Solution 2

```
public class Solution {
    public boolean isPowerOfFour(int num) {
        return num > 0
         && (num&(num-1)) == 0 && (num & 0x55555555) != 0;
    }
}
```

Problem 24

191 Number of 1 Bits

```
Write a function that takes an unsigned integer and returns
 the number of '1' bits it has (also known as the Hamming weight).

For example, the 32-bit integer '11' has
 binary representation 00000000000000000000000000001011,
  so the function should return 3.

https://leetcode.com/problems/number-of-1-bits/
```

Solution 1

To be accurate, the 2 solutions are 2 ways of thinking rather than ways of solving problems. The first one uses >>> (unsigned bit operator) and check if n is still larger than 0. *DO NOT USE '>>'! It will overflow for* 2^{32}*!*

Listing 30: Solution 1

```
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int count = 0;
        while(n != 0){
            count += (n & 1);
            n >>>= 1;
        }
        return count;
    }
}
```

Solution 2

The 2nd solution make use of the fact that the integer is at most 32 bits. It iterates the number for 32 bits to see whether 1 exists.

Listing 31: Solution 2

```
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int count = 0;
        for(int i = 0; i < 32; i++){
            count += (n & 1);
            n >> = 1;
        }
        return count;
    }
}
```

What's more, using n & (n-1) also does excellent work. Mind those boundaries.

Problem 25

202 Happy Number

```
Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process:
 Starting with any positive integer,
  replace the number by the sum of the squares of its digits,
   and repeat the process until the number equals 1 (where it will stay),
    or it loops endlessly in a cycle which does not include 1.
Those numbers for which this process ends in 1 are happy numbers.

Example: 19 is a happy number

1*1 + 9*9 = 82
8*8 + 2*2 = 68
6*6 + 8*8 = 100
1*1 + 0*0 + 0*0 = 1

https://leetcode.com/problems/happy-number/
```

Solution

A set is enough for this problem.

Listing 32: Solution

```
public class Solution {
    public boolean isHappy(int n){
        Set<Integer> set = new HashSet<Integer>();
        int sum = 0;
        while (true){
            while (n != 0){
                sum += (n % 10) * (n % 10);
                n /= 10;
            }
            if (sum == 1){
                break;
            }
            if (set.contains(sum)){
                return false;
            }
            else{
                set.add(sum);
            }
            n = sum;
            sum = 0;
        }
        return true;
    }
}
```

Problem 26

61 Rotate List

```
Given a list, rotate the list to the right by k places, where k is non-negative.

For example:
Given 1->2->3->4->5->NULL and k = 2,
return 4->5->1->2->3->NULL.

https://leetcode.com/problems/rotate-list/
```

Solution

Though this problem seems not that difficult, there are still some edge cases which needs to be dealt with.

The main idea of my solution is to use 2 pointers to get the 'Nth element from the end'(sounds familiar), one goes faster for k steps.

Here comes the **TRAP**! When k is (much) larger than length of the list, we will get Time Limit Exceed error. So mod operator is in need here.

Also we need to be aware of the condition when k==0(or k==n where n%n==0)

Listing 33: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if (head == null || head.next == null || k == 0){
            return head;
        }
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        int len = 0;
        ListNode cur = dummy;
        while (cur.next != null){
            len++;
            cur = cur.next;
        }
        k = k % len;
        ListNode fast = head;
        while (k--- > 0){
            fast = fast.next;
            if (k > 0 && fast == null){
                fast = head;
            }
        }
        ListNode slow = head;
        if (fast == null){
```

```
            return head;
        }
        while (fast.next != null){
            fast = fast.next;
            slow = slow.next;
        }
        fast.next = dummy.next;
        dummy.next = slow.next;
        slow.next = null;
        return dummy.next;
    }
}
```

Problem 27

24 Swap Nodes in Pairs

```
Given a linked list, swap every two adjacent nodes and return its head.

For example,
Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space.
 You may not modify the values in the list, only nodes itself can be changed.

https://leetcode.com/problems/swap-nodes-in-pairs/
```

Solution

Not many tactics, just maintain 2 pointers to reverse cur and next.

Listing 34: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode swapPairs(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode dummy = new ListNode(-1);
        ListNode tail = head;
        dummy.next = head.next;
        ListNode cur = head;
        ListNode next = cur.next;
        while (cur != null && next != null){
            ListNode tmp = next.next;
            tail.next = next;
            next.next = cur;
            cur.next = null;
            tail = cur;
            cur = tmp;
            if (cur != null){
                next = cur.next;
            }
            if (next == null){
                tail.next = cur;
            }
        }
        return dummy.next;
    }
}
```

Problem 27 continued on next page...

Problem 28

66 Plus One

```
Given a non-negative number represented as an array of digits,
 plus one to the number.

The digits are stored such that
 the most significant digit is at the head of the list.

https://leetcode.com/problems/plus-one/
```

Solution

Though it is correct to use carry like we usually do when doing sum action, plus one is not as complicated to use carry digit. See listing 35, it's a pretty lovely solution.

Listing 35: Solution

```
public class Solution {
    public int[] plusOne(int[] digits){
        for (int idx = digits.length - 1 ; idx >= 0 ; idx--){
            int digit = digits[idx];
            if (digit < 9){
                digits[idx] = digit + 1;
                return digits;
            }
            else{
                digits[idx] = 0;
            }
        }
        int[] newDigits = new int[digits.length + 1];
        newDigits[0] = 1;
        for (int idx = 0 ; idx < digits.length ; idx ++){
            newDigits[idx + 1] = 0;
        }
        return newDigits;
    }
}
```

Problem 29

2 Add Two Numbers

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

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8

https://leetcode.com/problems/add-two-numbers/
```

Solution

Some sort of addition calculator with high precision. Some edge cases to be considered. However, my solution is far from 'Clean Code'.

Listing 36: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode addTwoNumbers(ListNode head1, ListNode head2) {
        ListNode l1 = head1, l2 = head2;
        ListNode dummy = new ListNode(-1);
        ListNode cur = dummy;
        int carry = 0;
        while (l1 != null && l2 != null){
            int digit1 = l1.val;
            int digit2 = l2.val;
            int sum = digit1 + digit2 + carry;
            int digit = sum % 10;
            carry = sum / 10;
            ListNode newNode = new ListNode(digit);
            cur.next = newNode;
            cur = cur.next;
            l1 = l1.next;
            l2 = l2.next;
        }
        if (l1 == null){
            while (l2 != null){
                int digit = (l2.val + carry) % 10;
                carry = (l2.val + carry) / 10;
                ListNode newNode = new ListNode(digit);
                cur.next = newNode;
                cur = cur.next;
                l2 = l2.next;
```

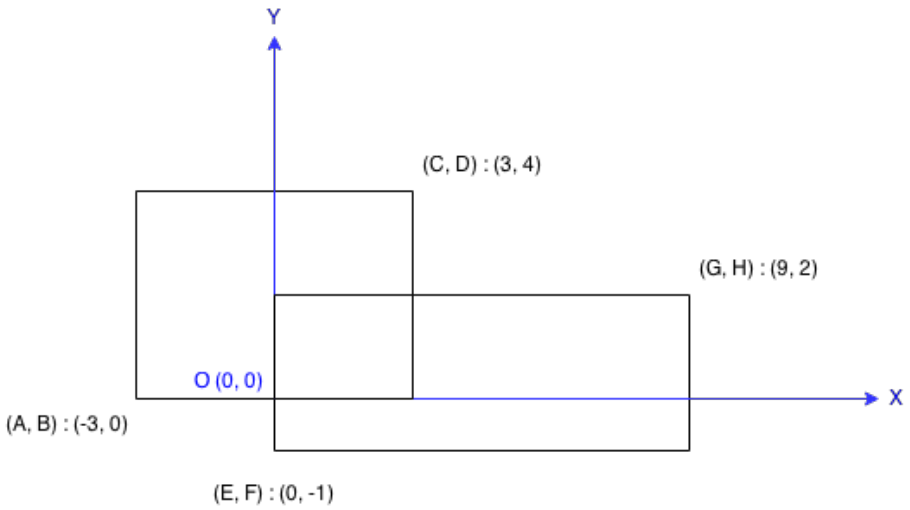
Problem 29 continued on next page...

```
            }
        }
        else if (l2 == null){
            while (l1 != null){
                int digit = (l1.val + carry) % 10;
                carry = (l1.val + carry) / 10;
                ListNode newNode = new ListNode(digit);
                cur.next = newNode;
                cur = cur.next;
                l1 = l1.next;

            }
        }
        if (l1 == null && l2 == null){
            if (carry > 0){
                ListNode newNode = new ListNode(carry);
                cur.next = newNode;
            }

        }
        return dummy.next;
    }
}
```

Problem 30

223 Rectangle Area

```
Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and
 top right corner as shown in the figure.

For Rectangle Area, see image below.
Assume that the total area is never beyond the maximum possible value of int.

https://leetcode.com/problems/rectangle-area/
```



Solution

Quite an easy problem, just read the description of problem more carefully.

Listing 37: Solution

```
public class Solution {
    public int computeArea(int A, int B, int C, int D,
     int E, int F, int G, int H) {
        int newXMin = Math.max(A, E);
        int newYMin = Math.max(B, F);
        int newXMax = Math.min(C, G);
        int newYMax = Math.min(D, H);
        int area1 = (D - B) * (C - A);
        int area2 = (H - F) * (G - E);
        if (newXMin < newXMax && newYMin < newYMax){
            int common = (newYMax - newYMin) * (newXMax - newXMin);
            return area1 + area2 - common;
        }
        else{
            return area1 + area2;
        }
    }
}
```

Problem 31

7 Reverse Integer

```
Reverse digits of an integer.

Example1: x = 123, return 321
Example2: x = -123, return -321

click to show spoilers.

Have you thought about this?
Here are some good questions to ask before coding.
 Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be?
 ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow?
 Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows.
  How should you handle such cases?

For the purpose of this problem,
 assume that your function returns 0 when the reversed integer overflows.

https://leetcode.com/problems/reverse-integer/
```

Solution

Two points which may attract interviewers if you have mentioned,

- What will happen if I input minus number?
- What if the reversed number overflows?

And solution in 'Clean Code Handbook' seems mistaken since it only consider the condition when sum > MAX_VALUE/10 and there still remains unresolved number. What about 2147483649?

Honestly, I don't think it is a problem as easy as I imagined at first sight.

Listing 38: Solution

```
public class Solution {
    public int reverse(int x) {
        int maxHead = Integer.MAX_VALUE / 10;
        int maxTail = Integer.MAX_VALUE % 10;
        int minTail = Integer.MIN_VALUE % 10;
        int origin = x;
        int sum = 0;
        while (origin != 0){
            int digit = origin % 10;
            origin /= 10;
            if (Math.abs(sum) > maxHead && digit != 0){
                return 0;
            }
```

Problem 31 continued on next page...

```
            if (Math.abs(sum) == maxHead){
                if ((digit > 0 && digit > maxTail)
                 || (digit < 0 && digit < minTail))
                return 0;
            }
            sum = sum * 10 + digit;
        }
        return sum;
    }
}
```

Problem 32

190 Reverse Bits

```
Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596
 (represented in binary as 00000010100101000001111010011100),
  return 964176192 (represented in binary as 00111001011110000010100101000000).

Follow up:
If this function is called many times, how would you optimize it?

https://leetcode.com/problems/reverse-bits/
```

Solution

Basically it is an easy bit manipulation problem. But when we need to call it several times, the method of optimization can be found `http://articles.leetcode.com/reverse-bits`.

Listing 39: Solution

```
public class Solution {
    // you need treat n as an unsigned value
    public int reverseBits(int n) {
        int newNum = 0;
        for (int i = 0 ; i < 32 ; i++){
            newNum = (newNum << 1);
            newNum += (n & 1);
            n = n >>> 1;
        }
        return newNum;
    }
}
```

Problem 33

141 Linked List Cycle

```
Given a linked list, determine if it has a cycle in it.

Follow up:
Can you solve it without using extra space?

https://leetcode.com/problems/linked-list-cycle/
```

Solution

Still, the slow and fast pointer, if the list has cycle inside, the slow pointer will be caught sooner or later.

Listing 40: Solution

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while (fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast){
                return true;
            }
        }
        return false;
    }
}
```

Problem 34

142 Linked List Cycle II

```
Given a linked list, return the node where the cycle begins.
 If there is no cycle, return null.

Note: Do not modify the linked list.

Follow up:
Can you solve it without using extra space?

https://leetcode.com/problems/linked-list-cycle-ii/
```

Solution

If no restrictions in that 'Note', we can easily handle this by HashSet.

Now we can only use mathematical method to solve this problem.

Forget about this problem for a moment. How can we get the length of cycle? When slow and fast pointers continues iterating so as to meet for the 2nd time, the slow pointer passes exactly the length of the whole cycle(or an integral multiple of cycle length)!

Coming back to this problem, we assign 'a' as length from head of list to where the cycle begins, 'b' as length from where cycle begins to where 2 pointers met, 'c' as length passed by fast pointer after it first passed the node where they met.

Then?

$2(a+b) = a+b+c$, here c may be l*R, then we get a = c - b!!!

Which means, if we re-assign 2 pointers, let one starts from head of list, the other starts from where they met, they will meet at where the cycle begins!

For more details, refer to `http://blog.sina.com.cn/s/blog\_6f611c300101fs1l.html`

Listing 41: Solution

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode detectCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        ListNode l1 = null, l2 = null;
        ListNode cycleBegins = null;
        while (fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
```

```
            if (slow == fast){
                l1 = head;
                l2 = slow;

                break;
            }
        }
        if (fast != null && fast.next != null && l1 != null && l2 != null){
            while (l1 != l2){
                l1 = l1.next;
                l2 = l2.next;
            }

            cycleBegins = l1;
        }
        return cycleBegins;
    }
}
```

Problem 35

13 Roman to Integer

```
Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

https://leetcode.com/problems/roman-to-integer/
```

Solution

Main idea is simple, just to check each character and sum up those values.

However, those IX(for 9, 11 is XI) and Xl are special(they are designed for the purpose that the same notation won't appear for more than 3 times in a row.), honestly we can deal with all kinds of patterns, we still have a simplified algorithm: when cur digit is larger than prev(IX, X is larger), then we should add cur to sum and remove prev for 2 times. sum = sum + cur - 2 * prev.

Listing 42: Solution

```
public class Solution {
    private static final HashMap<Character, Integer> values
     = new HashMap<Character, Integer>(){
        {
            put('I', 1);
            put('V', 5);
            put('X', 10);
            put('L', 50);
            put('C', 100);
            put('D', 500);
            put('M', 1000);
        }
    };
    public int romanToInt(String s) {
        int sum = 0;
        int prev = -1;
        char[] chars = s.toCharArray();
        for (char c : chars){
            int cur = values.get(c);
            if (prev != -1 && prev < cur){
                sum += cur - 2 * prev;
            }
            else{
                sum += cur;
            }
            prev = cur;
        }
        return sum;
    }
}
```

Problem 36

12 Integer to Roman

```
Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

https://leetcode.com/problems/integer-to-roman/
```

Solution

Find biggest value, remove it, and append corresponding Roman notation to String.

Difficult thing is that there exists IX or XV to indicate reduction. Luckily, there is a given information that integer is 1∼3999, so that we can list all possible notation patterns since IX always appear after X(9 < 10, 10 comes first).

Listing 43: Solution

```
public class Solution {
    private static final String[] romans = { "M", "CM", "D", "CD", "C", "XC",
            "L", "XL", "X", "IX", "V", "IV", "I" };
    private static final int[] values = { 1000, 900, 500, 400, 100, 90, 50, 40,
            10, 9, 5, 4, 1 };
    public String intToRoman(int num) {
        StringBuilder sb = new StringBuilder();
        while (num != 0){
            int idx = 0;
            while (true){
                while (num < values[idx]){
                    idx ++;
                }
                sb.append(romans[idx]);
                num -= values[idx];
                if (num == 0){
                    return sb.toString();
                }
            }
        }
        return sb.toString();
    }
}
```

Problem 37

168 Excel Sheet Column Title

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

For example:

    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB

https://leetcode.com/problems/excel-sheet-column-title/
```

Solution

We can regard the integer as an number with base-26(binary is base-2 and decimal is base-10), but there is another thing we need to be aware of - base-2 consists only 0 and 1, where A-Z usually represents 1-26. So we should normalize our number, by reducing 1 to the integer we can normalize A-Z to 0-25.

Listing 44: Solution

```
public class Solution {
    public String convertToTitle(int n) {
        char[] titleDigits = new char[26];
        char a = 'A';
        for (int idx = 0 ; idx < 26 ; idx ++){
            titleDigits[idx] = (char)(a + idx);
        }
        String result = "";
        int cur = n;
        int rem = -1;

        while (cur != 0){
            rem = (cur - 1) % 26;
            result = titleDigits[rem] + result;
            cur = (cur - 1) / 26;
        }
        return result;
    }
}
```

Problem 38

171 Excel Sheet Column Number

```
Related to question Excel Sheet Column Title

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

For example:

    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28

https://leetcode.com/problems/excel-sheet-column-number/
```

Solution

It's just a reversion of its related problem.

Listing 45: Solution

```
public class Solution {
    public int titleToNumber(String s) {
        char[] title = s.toCharArray();
        int sum = 0;

        for (int idx = 0 ; idx < title.length ; idx++){
            char c = title[idx];
            int digit = c - 'A' + 1;

            sum = sum * 26 + digit;
        }

        return sum;
    }
}
```

Problem 39

60 Permutation Sequence

```
The set [1,2,3,  ,n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order,
We get the following sequence (ie, for n = 3):

"123"
"132"
"213"
"231"
"312"
"321"
Given n and k, return the kth permutation sequence.

Note: Given n will be between 1 and 9 inclusive.

https://leetcode.com/problems/permutation-sequence/
```

Solution 1

This is maybe the 1st backtracking problem solved by myself!!

In fact, the problem can be solved without recursion in Listing 47

The idea of solving this problem is to justify each digit from the most significant digit. Using (k - 1) / (n - 1)! we can get the index of current digit and remaining numbers.

For example, for n = 3 and k = 4, the 1st digit is decided by (4 - 1) / (3 - 1)! = 1 remains 1, so the 1st digit is the (1+1), 2nd biggest element in remaining set, which is 2; recursively, the whole number is 231.

Listing 46: Solution 1

```
public class Solution {
    public String getPermutation(int n, int k) {
        Set<Integer> numbers = new HashSet<Integer>();
        for (int i = 1;  i <= n ; i++){
            numbers.add(i);
        }
        return getPermutation(numbers , n , k);
    }
    private String getPermutation(Set<Integer> numbers, int n, int k) {
        if (n == 0){
            return "";
        }
        int qou = (k - 1) / getFact(n - 1);
        int rem = (k - 1) % getFact(n - 1);
        Iterator<Integer> ite = numbers.iterator();
        int num = ite.next();
        int count = qou;

        while (ite.hasNext() && count-- > 0){
            num = ite.next();
        }
        numbers.remove(num);
```

```
        return String.valueOf(num) + getPermutation(numbers, n - 1, rem + 1);
    }

    private int getFact(int n){
        for (int i = 1 ; i <= n ; i++){
            power *= i;
        }
        return power;
    }
}
```

Solution 2

Listing 47: Solution 2

```
class Solution {
    int getPermutationNumber(int n) {
        int result = 1;
        for(int i=1;i<=n;++i) {
            result *=i;
        }

        return result;
    }
public:
    string getPermutation(int n, int k) {
        vector<int> result;
        vector<int> nums;
        for(int i=1;i<=n;++i) {
            nums.push_back(i);
        }
        k=k-1;
        for(int i=0;i<n;++i) {
            int perms = getPermutationNumber(n-1-i);
            int index = k/perms;
            result.push_back(nums[index]);
            k%=perms;
            nums.erase(nums.begin()+index);
        }
        string s="";
        for(int i=0;i<result.size();++i) {
            s += std::to_string(result[i]);
        }
        return s;
    }
};
```

Solution 3 There is a more interesting solution, given the fact that the numbers only contains 1-9.

Listing 48: Solution 3

```
class Solution {
public:
    string getPermutation(int n, int k) {
```

```
        string res;
        string nums = "123456789";
        int f[10] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};
        --k;
        for (int i = n; i >= 1; --i) {
            int j = k / f[i - 1];
            k %= f[i - 1];
            res.push_back(nums[j]);
            nums.erase(nums.begin() + j);
        }
        return res;
    }
};
```

Problem 40

46 Permutations

```
Given a collection of distinct numbers, return all possible permutations.

For example,
[1,2,3] have the following permutations:
[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].

https://leetcode.com/problems/permutations/
```

Solution

It is a representative problem of backtracking or recursion.

In fact, there are all kinds of algorithms to generate all-permutations.

See http://blog.csdn.net/ljiabin/article/details/40151393

Listing 49: Solution

```
public class Solution {
    Set<Integer> used = new HashSet<Integer>();
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    List<Integer> array = new ArrayList<Integer>();
    public List<List<Integer>> permute(int[] nums) {
        for (int idx = 0; idx < nums.length; idx++) {
            int num = nums[idx];
            if (used.contains(num)) {
                continue;
            } else {
                used.add(num);
                array.add(num);
                if (array.size() == nums.length) {
                    ArrayList<Integer> newList = new ArrayList<Integer>();
                    for (int ele : array) {
                        newList.add(ele);
                    }
                    result.add(newList);
                }

                permute(nums);
                array.remove(array.size() - 1);
                used.remove(num);
            }
        }
        return result;
    }
}
```

Problem 41

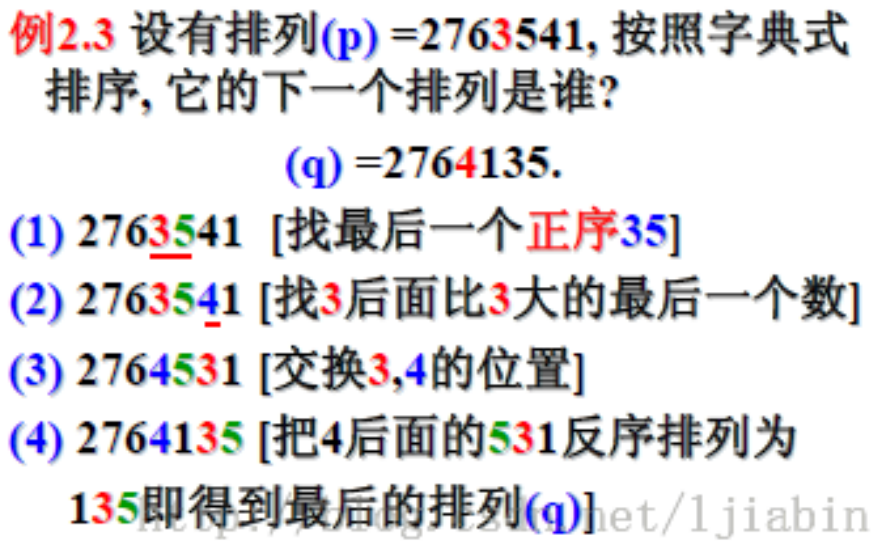
31 Next Permutation

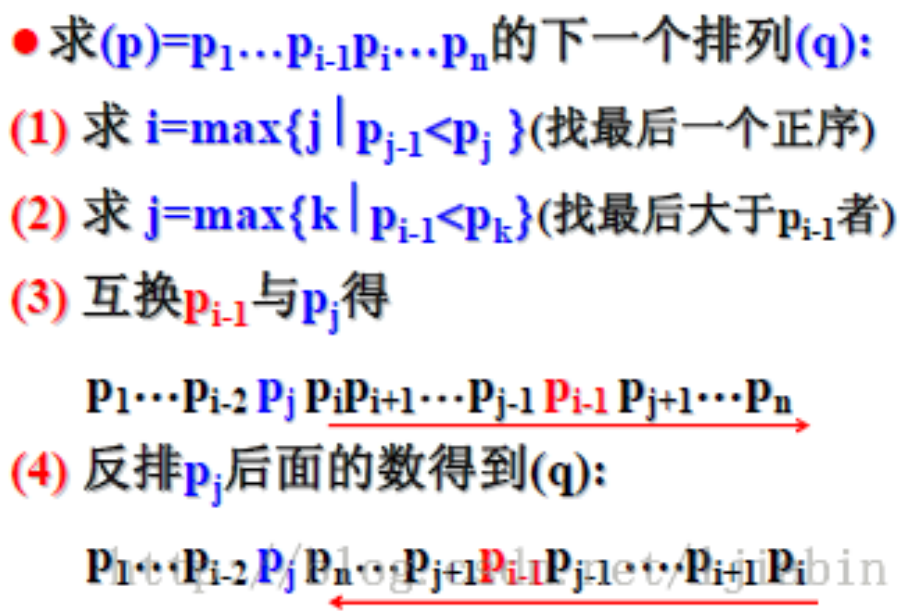
```
Implement next permutation, which rearranges numbers into the lexicographically
 next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as
 the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column
 and its corresponding outputs are in the right-hand column.
    1,2,3 -> 1,3,2
    3,2,1 -> 1,2,3
    1,1,5 -> 1,5,1

https://leetcode.com/problems/next-permutation/
```

Solution

Utilizes a lexicographical order Algorithm.

The algorithm is explained as below.





Listing 50: Solution

```java
public class Solution {
    public void nextPermutation(int[] nums) {
        int preIdx = -1 , postIdx = -1;
        for (int i = 0 ; i < nums.length - 1 ; i ++){
            if (nums[i] < nums[i + 1])
                preIdx = i;
        }
        if (preIdx < 0){
            reverse(nums , 0 , nums.length - 1);
            return;
        }
        for (int i = preIdx + 1; i < nums.length ; i ++){
            if (nums[preIdx] < nums[i])
                postIdx = i;
        }
        int tmp = nums[preIdx];
        nums[preIdx] = nums[postIdx];
        nums[postIdx] = tmp;
        reverse(nums, preIdx + 1, nums.length - 1);
    }
    private void reverse(int[] nums, int i, int j) {
        while (i < j){
            int tmp = nums[i];
            nums[i] = nums[j];
            nums[j] = tmp;
            i ++;
            j --;
        }
    }
}
```

Problem 42

47 Permutations II

```
Given a collection of numbers that might contain duplicates, return
 all possible unique permutations.

For example,
[1,1,2] have the following unique permutations:
[1,1,2], [1,2,1], and [2,1,1].

https://leetcode.com/problems/permutations-ii/
```

Solution

We can use the same algorithm from Pro-31 to get the next permutation, but remember to sort the array first, or it may lose some results.

It can also be solved by backtracking method.

Listing 51: Solution

```
public class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> list = new ArrayList<List<Integer>>();
        Arrays.sort(nums);
        List<Integer> newPerm = new ArrayList<Integer>();
        for (int idx = 0 ; idx < nums.length ; idx++){
            newPerm.add(nums[idx]);
        }
        list.add(newPerm);
        while (nextPermutation(nums)){
            newPerm = new ArrayList<Integer>();
            for (int idx = 0 ; idx < nums.length ; idx++){
                newPerm.add(nums[idx]);
            }
            list.add(newPerm);
        }
        return list;
    }
    public boolean nextPermutation(int[] nums) {
        int preIdx = -1 , postIdx = -1;
        for (int i = 0 ; i < nums.length - 1 ; i ++){
            if (nums[i] < nums[i + 1]){
                preIdx = i;
            }
        }
        if (preIdx < 0){
            return false;
        }
        for (int i = preIdx + 1; i < nums.length ; i ++){
            if (nums[preIdx] < nums[i]){
                postIdx = i;
            }
        }
```

Problem 42 continued on next page...

```
            int tmp = nums[preIdx];
            nums[preIdx] = nums[postIdx];
            nums[postIdx] = tmp;
            reverse(nums, preIdx + 1, nums.length - 1);
            return true;
        }

        private void reverse(int[] nums, int i, int j) {
            while (i < j){
                int tmp = nums[i];
                nums[i] = nums[j];
                nums[j] = tmp;

                i ++;
                j --;
            }
        }
    }
```

Problem 43

77 Combinations

```
Given a collection of numbers that might contain duplicates,
 return all possible unique permutations.

For example,
[1,1,2] have the following unique permutations:
[1,1,2], [1,2,1], and [2,1,1].Given two integers n and k,
 return all possible combinations of k numbers out of 1 ... n.

For example,
If n = 4 and k = 2, a solution is:

[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]

https://leetcode.com/problems/combinations/
```

Solution

A dfs problem, similar but not quite same as Problem "Permutations", Permutations aim at swapping elements, this one is more like "N-Queens".

Listing 52: Solution

```
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();

    public void dfsCombine(List<Integer> curList, int totalDigit, int curDigit, int totalNumb
        if (curDigit == totalDigit){
            result.add(new ArrayList(curList));
            return;
        }
        for (int number = curNumber + 1 ; number <= totalNumber; number ++){
            if (curList.size() <= curDigit){
                curList.add(number);
            }
            else{
                curList.set(curDigit, number);
            }
            dfsCombine(curList, totalDigit, curDigit + 1, totalNumber, number);
        }

    }

    public List<List<Integer>> combine(int n, int k) {
        List<Integer> list = new ArrayList<Integer>();
```

Problem 43 continued on next page...

```
        dfsCombine(list, k, 0, n, 0);
        return result;
    }
}
```

Problem 44

172 Factorial Trailing Zeroes

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

Note: Your solution should be in logarithmic time complexity.

https://leetcode.com/problems/factorial-trailing-zeroes/
```

Solution

What decides the number of trailing 0 is how many numbers has the factor of 5(since 5*2=10 and count of 2 is much more than 5). Note that not only 5 should be considered, 25 provides two 5s, which also needs to be take into consideration.

Listing 53: Solution

```
public class Solution {
    public int trailingZeroes(int n) {
        int sum = 0;
        while (n != 0){
            n = n / 5;
            sum += n;
        }
        return sum;
    }
}
```

Problem 45

8 String to Integer (atoi)

```
Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge,
 please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely
 (ie, no given input specs).
 You are responsible to gather all the input requirements up front.


Requirements for atoi:
The function first discards as many whitespace characters as necessary
 until the first non-whitespace character is found.
  Then, starting from this character, takes an optional initial plus or minus sign
   followed by as many numerical digits as possible, and interprets them
    as a numerical value.

The string can contain additional characters after those that
 form the integral number,
 which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str
 is not a valid integral number, or if no such sequence exists
  because either str is empty or it contains only whitespace characters,
   no conversion is performed.

If no valid conversion could be performed, a zero value is returned.
 If the correct value is out of the range of representable values,
  INT_MAX (2147483647) or INT_MIN (-2147483648) is returned.

https://leetcode.com/problems/string-to-integer-atoi/
```

Solution

The biggest difficulty of this problem is that there are so many edge cases(though less than those in valid number).

Listing 54: Solution

```
public class Solution {
    public int myAtoi(String str) {
     if (str == null || str.length() == 0){
          return 0;
     }
        int sign = 1;
        char[] ch = str.toCharArray();
        int idx = 0;
        while (ch[idx] == ' '){
          idx ++;
        }
        if (ch[idx] == '-'){
```

```
            sign = -1;
            idx++;
        }
        else if (ch[idx] == '+'){
            idx++;
        }
        int sum = 0;
        while (idx < str.length()){
            int temp = ch[idx] - '0';
            if (temp > 9 || temp < 0){
                return sum * sign;
            }
            if (sum > Integer.MAX_VALUE / 10 ||
                (sum == Integer.MAX_VALUE / 10 && temp > Integer.MAX_VALUE % 10)){
                return (sign == 1) ? Integer.MAX_VALUE : Integer.MIN_VALUE;
            }
            sum = sum * 10 + temp;
            idx ++;
        }
        return sum * sign;
    }
}
```

Problem 46

28 Implement strStr()

```
Implement strStr().

Returns the index of the first occurrence of needle in haystack,
 or -1 if needle is not part of haystack.

https://leetcode.com/problems/implement-strstr/
```

Solution

An easy string manipulating problem. Since I'm using a $O(mn)$ solution, and there's another famous algorithm named KMP which can solve this problem with lower complexity, the solution has the space for improvement.

And also don't forget the solution on "Clean Code Handbook".

Listing 55: Solution

```java
public class Solution {
    public int strStr(String haystack, String needle) {
        if (needle.length() == 0){
            return 0;
        }
        for (int hIdx = 0 ; hIdx < haystack.length() ; hIdx ++){
            if (haystack.length() - hIdx < needle.length()){
                return -1;
            }

            if (haystack.charAt(hIdx) == needle.charAt(0)){
                int hStart = hIdx;
                int nStart = 0;
                while (nStart < needle.length()
                    && haystack.charAt(hStart) == needle.charAt(nStart)){
                    hStart ++;
                    nStart ++;
                }
                if (nStart == needle.length()){
                    return hIdx;
                }
            }
        }

        return -1;
    }
}
```

Problem 47

169 Majority Element

```
Given an array of size n, find the majority element.
 The majority element is the element that appears more than floor(n/2) times.

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

https://leetcode.com/problems/majority-element/
```

Solution 1

This is my solution, luckily write out **Moore voting algorithm**, though not 'Clean Code':

We maintain a current candidate and a counter initialized to 0. As we iterate the array, we look at the current element x:

If the counter is 0, we set the current candidate to x and the counter to 1.

If the counter is not 0, we increment or decrement the counter based on whether x is the current candidate.

After one pass, the current candidate is the majority element. Runtime complexity = O(n).

Listing 56: Solution 1

```
public class Solution {
    public int majorityElement(int[] nums) {
        int curNum = -1;
        int curCount = 0;
        int idx1 = 0, idx2 = 1;
        while (idx1 < nums.length && idx2 < nums.length) {
            if (nums[idx1] == nums[idx2]) {
                curCount++;
                curNum = nums[idx1];
                idx2++;
            } else {
                if (nums[idx1] == curNum) {
                    if (curCount > 0) {
                        curCount--;
                        idx2++;

                        continue;
                    }
                    else{
                        curNum = -1;
                    }

                }
                idx1 = idx2 + 1;
                idx2 = idx2 + 2;
            }
        }
        if (idx1 == nums.length - 1 && curNum == -1) {
            curNum = nums[idx1];
        }
```

```
        return curNum;
    }
}
```

Solution 2

The reason why I record Solution 2 in such an excitement is that the algorithm is FXXXING excellent!!

We would need 32 iterations, each calculating the number of 1's for the ith bit of all n numbers. Since a majority must exist, therefore, either count of 1's > count of 0's or vice versa (but can never be equal). The majority numbers ith bit must be the one bit that has the greater count.(Words below are copied from Solutions of this problem in Leetcode)

Listing 57: Solution 2

```
public int majorityElement(int[] num) {
    int ret = 0;
    for (int i = 0; i < 32; i++) {
        int ones = 0, zeros = 0;
        for (int j = 0; j < num.length; j++) {
            if ((num[j] & (1 << i)) != 0) {
                ++ones;
            }
            else
                ++zeros;
        }

        if (ones > zeros)
            ret |= (1 << i);
    }
    return ret;
}
```

Problem 48

229 Majority Element II

```
Given an integer array of size n,
 find all elements that appear more than floor(n/3) times.
  The algorithm should run in linear time and in O(1) space.

Hint:

How many majority elements could it possibly have?

https://leetcode.com/problems/majority-element-ii/
```

Solution

Also, use Moore's Voting Algorithm, this time we have 2 candidates comparing to other nodes. Remember to check finally whether the two candidates are all majority elements we want since there may be only one majority element.

Listing 58: Solution 1

```java
public class Solution {
    public List<Integer> majorityElement(int[] nums) {
        int cand1 = -1 ,  cand2 = -1;
        int count1 = 0 , count2 = 0;

        for (int num : nums){
            if (num == cand1){
                count1 ++;
            }else if (num == cand2){
                count2 ++;
            }
            else {
                // num <> cand1 & num <> cand2
                if (count1 == 0){
                    cand1 = num;
                    count1 = 1;
                }
                else if (count2 == 0){
                    cand2 = num;
                    count2 = 1;
                }
                else{
                    count1 --;
                    count2 --;
                }
            }
        }
        count1 = 0;
        count2 = 0;
        // check if there is only one majority element
        for (int num : nums){
            if (num == cand1){
                count1 ++;
```

```
            }
            if (num == cand2){
                count2 ++;
            }
        }
        List<Integer> result = new ArrayList<Integer>();
        if (count1 > nums.length / 3){
            result.add(cand1);
        }
        if (count2 > nums.length / 3){
            result.add(cand2);
        }

        return result;
    }
}
```

Another interesting solution

From 'Clean Code Handbook', with comments served as description.

Listing 59: Solution 2

```
class Solution {
public:
    int singleNumber(int A[], int n) {
        int one,two,three;
        one=two=three=0;
        for(int i=0;i<n;i++)
        {
            // Already appeared twice
            three =  two & A[i];
            // Appeard twice add circumstances
            //  when appeared once and in A[i] there is another
            two = two | one & A[i];
            // Appeared once
            one = one | A[i];
            // When already appeared for the 3rd time,
            //  clear one and two
            one = one & ~three;
            two = two & ~three;
        }
         return one;
    }
};
```

Problem 49

67 Add Binary

```
Given two binary strings, return their sum (also a binary string).

For example,
a = "11"
b = "1"
Return "100".

https://leetcode.com/problems/add-binary/
```

Solution

An easy problem just like “Add Two Number”, easier.

Listing 60: Solution

```
public class Solution {
    public String addBinary(String a, String b) {
        StringBuilder sb = new StringBuilder();
        int lenA = a.length();
        int lenB = b.length();
        if (a.length() < b.length()){
            return addBinary(b, a);
        }
        char[] chA = a.toCharArray();
        char[] chB = b.toCharArray();
        int carry = 0;
        for (int idxA = lenA - 1 ; idxA >= 0 ; idxA--){
            int idxB = idxA - (lenA - lenB);
            int digitA = chA[idxA] - '0';
            int digitB = (idxB < 0) ? 0 : chB[idxB] - '0';
            int rem = (digitA + digitB + carry) % 2;
            carry = (digitA + digitB + carry) / 2;
            sb.append(String.valueOf(rem));
        }
        if (carry == 1){
            sb.append("1");
        }
        return sb.reverse().toString();
    }
}
```

Problem 50

258 Add Digits

```
Given a non-negative integer num, repeatedly add all its digits
 until the result has only one digit.

For example:

Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2.
 Since 2 has only one digit, return it.

Follow up:
Could you do it without any loop/recursion in O(1) runtime?

Hint:

A naive implementation of the above process is trivial.
 Could you come up with other methods?

https://leetcode.com/problems/add-digits/
```

Solution 1

First come up with the "naive" solution.

Listing 61: Solution 1

```
public class Solution {
    public int addDigits(int num) {
        int sum = 0;
        while (num != 0){
            int digit = num % 10;
            num /= 10;
            sum += digit;
            if (num == 0){
                if (sum >= 10){
                    num = sum;
                    sum = 0;
                }
            }
        }
        return sum;
    }
}
```

Solution 2

Solution 2 is interesting and simple! See figure below.

Problem 50 continued on next page...

另一个方法比较简单，可以举例说明一下。假设输入的数字是一个5位数字num，则num的各位分别为a、b、c、d、e。

有如下关系：num = a * 10000 + b * 1000 + c * 100 + d * 10 + e

即：num = (a + b + c + d + e) + (a * 9999 + b * 999 + c * 99 + d * 9)

因为 a * 9999 + b * 999 + c * 99 + d * 9 一定可以被9整除，因此num模除9的结果与 a + b + c + d + e 模除9的结果是一样的。

对数字 a + b + c + d + e 反复执行同类操作，最后的结果就是一个 1-9 的数字加上一串数字，最左边的数字是 1-9 之间的，右侧的数字永远都是可以被9整除的。

这道题最后的目标，就是不断将各位相加，相加到最后，当结果小于10时返回。因为最后结果在1-9之间，得到9之后将不会再对各位进行相加，因此不会出现结果为0的情况。因为 (x + y) % z = (x % z + y % z) % z，又因为 x % z % z = x % z，因此结果为 (num - 1) % 9 + 1，只模除9一次，并将模除后的结果加一返回。

Listing 62: Solution 2

```
public class Solution {
    public int addDigits(int num) {
        return (num - 1) % 9 + 1;
    }
}
```

Problem 51

233 Number of Digit One

```
Given an integer n, count the total number of digit 1 appearing
 in all non-negative integers less than or equal to n.

For example:
Given n = 13,
Return 6, because digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

Hint:

Beware of overflow.

https://leetcode.com/problems/number-of-digit-one/
```

Solution

Honestly, I don't think I totally understand the solution, so that I'll list my current train of thought in case of forgetting.

```
例子更容易理解，所以以例子辅助说明。
主要思路是基于每十个数（0-9）就会有1个个位数为1， 每100个数就有10个十位数为1，.....
这样的话做循环看每一位就行了。
但是需要注意的是，以十位数为例，对于205,215,255，所得到的值是不一样的（215 比205多210~215这6个；255多所有210开头的十位数为1的。）
因此这几种情况要分别考虑。
        0：就是高位数乘以现在的base，对于205就是2（2是0的高位，我们现在正在处理十位数）*10；
        1：高位数乘以base还要加上低位数（例子中是5,0的低位），还要加1， 因为0~5是6个而不是5个。|
        2：是（高位数+1）乘以base，1在例子中就表示21开头的数字们
然后累加即可
```

Listing 63: Solution

```
public class Solution {
    public int countDigitOne(int n) {
        if (n < 0){
            return 0;
        }
        int base = 1;
        int num = n;
        int count = 0;
        while (num != 0){
            int lower = n - base * num;
            int quo = num / 10;
            int rem = num % 10;
            switch(rem){
                case 0:
                    count += quo * base;
                    break;
                case 1:
                    count += quo * base + lower + 1;
                    break;
                default:
                    count += (quo + 1) * base;
            }
            base *= 10;
```

```
            num /= 10;
        }
        return count;
    }
}
```

Problem 52

204 Count Primes

```
Count the number of prime numbers less than a non-negative number, n.

https://leetcode.com/problems/count-primes/
```

Solution 1

First solution is straightforward but a bit inefficient.

Listing 64: Solution 1

```
public class Solution {
    public int countPrimes(int n) {
        int count = 0;
        for (int i = 1; i < n; i++) {
            if (isPrime(i))
                count++;
        }
        return count;
    }

    private boolean isPrime(int num) {
        if (num <= 1)
            return false;
        // Loop's ending condition is i * i <= num instead of i <= sqrt(num)
        // to avoid repeatedly calling an expensive function sqrt().
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0)
                return false;
        }
        return true;
    }
}
```

Solution 2

Solution 2 is actually extracted from the hint spoilers from problem descriptions, since it is such a litany, I put it in solution description rather than in problem.

1. Let's start with a isPrime function. To determine if a number is prime, we need to check if it is not divisible by any number less than n. The runtime complexity of isPrime function would be $O(n)$ and hence counting the total prime numbers up to n would be $O(n^2)$. Could we do better?

2. As we know the number must not be divisible by any number > n / 2, we can immediately cut the total iterations half by dividing only up to n / 2. Could we still do better?

3. Let's write down all of 12's factors:

   2 * 6 = 12

   3 * 4 = 12

   4 * 3 = 12

   6 * 2 = 12

4. As you can see, calculations of 4 * 3 and 6 * 2 are not necessary. Therefore, we only need to consider factors up to $\sqrt{n}$ because, if n is divisible by some number p, then n = p * q and since p <= q, we could derive that p <= $\sqrt{n}$.

   Our total runtime has now improved to $O(n^{1.5})$, which is slightly better. Is there a faster approach? Code see solution 1.

5. The Sieve of Eratosthenes is one of the most efficient ways to find all prime numbers up to n. But don't let that name scare you, I promise that the concept is surprisingly simple.

   For figure of the algorithm,

   see `https://leetcode.com/static/images/solutions/Sieve\_of\_Eratosthenes\_animation.gif`.

   We start off with a table of n numbers. Let's look at the first number, 2. We know all multiples of 2 must not be primes, so we mark them off as non-primes. Then we look at the next number, 3. Similarly, all multiples of 3 such as 3 * 2 = 6, 3 * 3 = 9, ... must not be primes, so we mark them off as well. Now we look at the next number, 4, which was already marked off. What does this tell you? Should you mark off all multiples of 4 as well?

6. 4 is not a prime because it is divisible by 2, which means all multiples of 4 must also be divisible by 2 and were already marked off. So we can skip 4 immediately and go to the next number, 5. Now, all multiples of 5 such as 5 * 2 = 10, 5 * 3 = 15, 5 * 4 = 20, 5 * 5 = 25, ... can be marked off. There is a slight optimization here, we do not need to start from 5 * 2 = 10. Where should we start marking off?

7. In fact, we can mark off multiples of 5 starting at 5 * 5 = 25, because 5 * 2 = 10 was already marked off by multiple of 2, similarly 5 * 3 = 15 was already marked off by multiple of 3. Therefore, if the current number is p, we can always mark off multiples of p starting at p2, then in increments of p: p2 + p, p2 + 2p, ... Now what should be the terminating loop condition?

8. It is easy to say that the terminating loop condition is p < n, which is certainly correct but not efficient. Do you still remember Hint ♯3?

9. Yes, the terminating loop condition can be p < $\sqrt{n}$, as all non-primes >= $\sqrt{n}$ must have already been marked off. When the loop terminates, all the numbers in the table that are non-marked are prime.

   The Sieve of Eratosthenes uses an extra $O(n)$ memory and its runtime complexity is $O(nloglogn)$. For the more mathematically inclined readers, you can read more about its algorithm complexity on Wikipedia. Code snippets are in Solution listing.

Listing 65: Solution 2

```
public class Solution {
    public int countPrimes(int n) {
        boolean[] isPrime = new boolean[n];
        for (int i = 2; i < n; i++) {
            isPrime[i] = true;
        }
        // Loop's ending condition is i * i < n instead of i < sqrt(n)
        // to avoid repeatedly calling an expensive function sqrt().
        for (int i = 2; i * i < n; i++) {
            if (!isPrime[i])
                continue;
            for (int j = i * i; j < n; j += i) {
                isPrime[j] = false;
            }
```

```
        }
        int count = 0;
        for (int i = 2; i < n; i++) {
            if (isPrime[i])
                count++;
        }
        return count;
    }
}
```

Problem 53

20 Valid Parentheses

```
Given a string containing just the characters '(', ')', '{', '}', '[' and ']',
 determine if the input string is valid.

The brackets must close in the correct order,
 "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

https://leetcode.com/problems/valid-parentheses/
```

Solution

A representative stack problem, use stack to hold left parentheses, pop when meet right hands, check if match.

Listing 66: Solution

```
public class Solution {
    public boolean isValid(String s) {
        HashMap<Character, Character> pairs
         = new HashMap<Character, Character>(){
            {
                put('(',')');
                put('[',']');
                put('{','}');
            }
        };
        Stack<Character> stack = new Stack<Character>();
        for (int idx = 0; idx < s.length(); idx ++){
            char ch = s.charAt(idx);
            if (pairs.containsKey(ch)){
                stack.push(ch);
            }
            else{
                if (stack.isEmpty()){
                    return false;
                }
                else{
                    char left = stack.pop();
                    char expRight = pairs.get(left);
                    if (expRight != ch){
                        return false;
                    }
                }
            }
        }
        if (stack.isEmpty()){
            return true;
        }
        else {
            return false;
        }
    }
```

Problem 53 continued on next page...

```
}
```

Problem 54

32 Longest Valid Parentheses

```
Given a string containing just the characters '(' and ')',
 find the length of the longest valid (well-formed) parentheses substring.

For "(()", the longest valid parentheses substring is "()",
 which has length = 2.

Another example is ")()())", where
 the longest valid parentheses substring is "()()", which has length = 4.

https://leetcode.com/problems/longest-valid-parentheses/
```

Solution

It is a dp problem, but we can first think about how to solve it in Brute force Method, which is of $O(n^2)runtime$ and will get Time Limit Exceed error.

The dp solution is as listing. Interestingly, the array iterates from the end of string, dp[i] indicates the length of longest valid parentheses starts from index i. Of course when s[i] is ‘)’ it cannot be the start point, so we only need to consider the ‘(’ condition. if dp[i+1] > 0, which means there are valid ()s starts from i+1, then we should check if s[i+dp[i+1]+1] is ‘)’, if is the valid ()s ends at i+dp[i+1]+1 then dp=dp[i+1]+2.

This is not over, since dp[i+dp[i+1]+2] may not be 0, which we also need to take into addition.(We don’t need to finish such process recursively since the process is done when we were calculating dp[i+dp[i+1]+2]).

Be sure to check every boundaries conditions!!!

Listing 67: Solution Brute force

```
int findlength(string s) {
    int paired = 0;
    int len = 0;
    stack<char> stk;
    for(int i=0; i< s.length(); ++i)
    {
        if(s[i]=='(')stk.push(')');
        else if(!stk.empty()){
            stk.pop();
            len+=2;
            if(stk.empty()&&paired<len)
                paired=len;
        }else{
            len = 0;
        }
    }
    if(stk.empty() && paired == 0)
        paired += len;
    return paired;
}
int longestValidParentheses(string s) {
    // Note: The Solution object is instantiated only once.
    int max = 0;
    int tmp = 0;
    for(int i=0; i< s.length();i++)
```

```
    {
        tmp = findlength(s.substr(i));
        max = max>tmp?max:tmp;
    }
    return max;
}
```

Listing 68: Solution DP

```
public class Solution {
    public int longestValidParentheses(String s) {
        int[] dp = new int[s.length()];
        int max = 0;
        for (int idx = s.length() - 2; idx >= 0 ; idx --){
            if (s.charAt(idx) == '('){
                int nextIdx = idx + dp[idx + 1] + 1;
                if (nextIdx >= s.length()){
                    continue;
                }
                if (s.charAt(nextIdx) == ')'){
                    dp[idx] = dp[idx + 1] + 2;
                    if (nextIdx + 1 <= s.length() - 1
                     && s.charAt(nextIdx + 1) == '('){
                        dp[idx] += dp[nextIdx + 1];
                    }
                    if (dp[idx] > max){
                        max = dp[idx];
                    }
                }
            }
        }
        return max;
    }
}
```

Problem 55

155 Min Stack

```
Design a stack that supports push, pop, top,
 and retrieving the minimum element in constant time.

    push(x) -- Push element x onto stack.
    pop() -- Removes the element on top of the stack.
    top() -- Get the top element.
    getMin() -- Retrieve the minimum element in the stack.

https://leetcode.com/problems/min-stack/
```

Solution

We can use a stack with the same size of original stack to maintain the minimum value, however, we don't have to need such a large stack. What we need to do is to record elements less than current minimum value, and take care to check equation when popping take place.

Listing 69: Solution

```
class MinStack {
    Stack<Integer> origin = new Stack<Integer>();
    Stack<Integer> min = new Stack<Integer>();

    public void push(int x) {
        origin.push(x);
        if (min.isEmpty() || x <= min.peek()){
            min.push(x);
        }
    }

    public void pop() {
        int x = origin.pop();
        if (x == min.peek()){
            min.pop();
        }
    }

    public int top() {
        return origin.peek();
    }

    public int getMin() {
        return min.peek();
    }
}
```

Problem 56

239 Sliding Window Maximum

```
Given an array nums, there is a sliding window of size k
 which is moving from the very left of the array to the very right.
  You can only see the k numbers in the window.
   Each time the sliding window moves right by one position.

For example,
Given nums = [1,3,-1,-3,5,3,6,7], and k = 3.

Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
Therefore, return the max sliding window as [3,3,5,5,6,7].

Note:
You may assume k is always valid,
 ie: 1 <= k <= input array's size for non-empty array.

Follow up:
Could you solve it in linear time?

Hint:

How about using a data structure such as deque (double-ended queue)?
The queue size need not be the same as the window's size.
Remove redundant elements and the queue
 should store only elements that need to be considered.

https://leetcode.com/problems/sliding-window-maximum/
```

Solution

There are many solutions to solve such problem, the simplest but slowest way is using Brute force, which is of $O(nk)runtime$.

Other ways of solving it are listed as below,

Method 1：最朴素的解法就是将窗口在数组上进行滑动，每滑动一次求一下窗口的最值。时间复杂度O(nk)。
Method 2：还有就是使用平衡二叉树。

I. 取出数组前k个元素，构建平衡二叉树。
II. for i = 0→(n-1)
    a. 取出平衡二叉树最大值，加入结果集result。
    b. find numsi，删除这个节点。
    c. 将numsi+k出入平衡二叉树。

时间复杂度为O(nlogk)。

Method 3：其实还有更高效的解法就是使用deque
使用一个deque，只存储当前窗口中会对之后的选择有影响的元素。插入时每次从队尾进行，每次滑动窗口时只判断队首元素是否超出窗口范围，超出的话则删除。还有就是比较队尾元素与当前元素大小，如果小的话，直接删除队尾元素，否则插入。
判断元素是否超出窗口的范围，需要根据元素的下标。然而根据元素的下标可以确定元素的值，这一可以直接用queue保存数组下标的值。

Let's see an example.

实际上一个滑动窗口可以看成是一个队列。当窗口滑动时，处于窗口的第一个数字被删除，同时在窗口的末尾添加一个新的数字。这符合队列的先进先出特性。如果能从队列中找出它的最大数，这个问题也就解决了。
我们实现了一个可以用O(1)时间得到最小值的栈。同样，也可以用O(1)时间得到栈的最大值。同时我们讨论了如何用两个栈实现一个队列。综合这两个问题的解决方法，我们发现如果把队列用两个栈实现，由于可以用O(1)时间得到栈中的最大值，那么也就可以用O(1)时间得到队列的最大值，因此总的时间复杂度也就降到了O(n)。我们可以用这个方法来解决问题。不过这样就相当于在一轮面试的时间内要做两个面试题，时间未必够用。再来看看有没有其它的方法。
下面换一种思路。我们并不把滑动窗口的每个数值都存入队列中，而只把有可能成为滑动窗口最大值的数值存入到一个两端开口的队列。接着以输入数字{2,3,4,2,6,2,5,1}为例一步分析。
数组的第一个数字是2，把它存入队列中。第二个数字是3.由于它比前一个数字2大，因此2不可能成为滑动窗口中的最大值。2先从队列里删除，再把3存入到队列中。此时队列中只有一个数字3.针对第三个数字4的步骤类似，最终在队列中只剩下一个数字4.此时滑动窗口中已经有3个数字，而它的最大值4位于队列的头部。
接下来处理第四个数字2。2比队列中的数字4小。当4滑出窗口之后2还是有可能成为滑动窗口的最大值，因此把2存入队列的尾部。现在队列中有两个数字4和2，其中最大值4仍然位于队列的头部。
下一个数字是6.由于它比队列中已有的数字4和2都大，因此这时4和2已经不可能成为滑动窗口中的最大值。先把4和2从队列中删除，再把数字6存入队列。这个时候最大值6仍然位于队列的头部。
第六个数字是2.由于它比队列中已有的数字6小，所以2也存入队列的尾部。此时队列中有两个数字，其中最大值6位于队列的头部。
接下来的数字是5.在队列中已有的两个数字6和2里，2小于5，因此2不可能是一个滑动窗口的最大值，可以把它从队列的尾部删除。删除数字2之后，再把数字5存入队列。此时队列里剩下两个数字6和5，其中位于队列头部的是最大值6.
数组最后一个数字是1，把1存入队列的尾部。注意到位于队列头部的数字6是数组的第5个数字，此时的滑动窗口已经不包括这个数字了，因此应该把数字6从队列删除。那么怎么知道滑动窗口是否包括一个数字？应该在队列里存入数字在数组里的下标，而不是数值。当一个数字的下标与当前处理的数字的下标之差大于或者等于滑动窗口的大小时，这个数字已经从滑动窗口中滑出，可以从队列中删除了。

LinkedList in Java has similar interface as deque in C++.

(BTW, there is another way to solve it...) Just think about "PROBLEM 155 Min Stack", We have already implemented a min stack with $O(1)$*runtime* pop(), push() and getMin(), we can accomplish max stack in a similar way, so that we can use the stack to finish this task.

Listing 70: Solution

```
public class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        if (k == 0 || nums.length == 0){
            return nums;
        }
        LinkedList<Integer> windows = new LinkedList();
        int[] result = new int[nums.length - k + 1];
        for (int idx = 0 ; idx < nums.length ; idx++){
            int cur = nums[idx];
            if (!windows.isEmpty() && windows.getFirst() <= idx - k){
                windows.removeFirst();
            }
            while (!windows.isEmpty() && cur > nums[windows.getLast()]){
                windows.removeLast();
            }
            windows.addLast(idx);
            if (idx >= k - 1){
                result[idx - k + 1] = nums[windows.getFirst()];
            }
        }
        return result;
    }
}
```

Problem 57

189 Rotate Array

```
Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3,
 the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note:
    Try to come up as many solutions as you can,
     there are at least 3 different ways to solve this problem.


Hint:
    Could you do it in-place with O(1) extra space?
    Related problem: Reverse Words in a String II

https://leetcode.com/problems/rotate-array/
```

Solution 1

Problem ask us to solve the problem in 3 different ways, I'll list my own solutions, and list others in below sections(only description, no code).

Solution 1 is simple, just use another array to simulate the process in description.

Listing 71: Solution 1

```
public class Solution {
    public void rotate(int[] nums, int k) {
        int[] result = new int[nums.length];
        k = k % nums.length;
        for (int i = nums.length - k ; i < nums.length ; i++){
            result[i - nums.length + k] = nums[i];
        }
        for (int j = 0 ; j < nums.length - k ; j ++){
            result[k + j] = nums[j];
        }
        for (int i = 0 ; i < nums.length ; i ++){
            nums[i] = result[i];
        }
    }
}
```

Solution 2

Solution is to move every bit backward for k digits, but which digit should I swap with? This solution forms a cycle, for example, if n = 9, k = 3, we shall swap a[0] to a[3], then a[3] to a[6], next one should be a[0], which returns to start point. Now we should finish this cycle and move cursor to next digit, namely a[1] to a[4]...

Listing 72: Solution 2

```
public class Solution {
    public void rotate(int[] nums, int k) {
```

```
        int count = 0;
        int start = 0;
        int last = nums[0];
        k %= nums.length;
        if (k == 0){
            return;
        }
        while (count < nums.length && start < nums.length - 1){
            int cur = start;
            last = nums[cur];
            do{
                int swap = nums[cur];
                nums[cur] = last;
                last = swap;
                cur = (cur + k) % nums.length;
                count ++;
            }while(cur != start);
            nums[start] = last;

            start ++;
        }
    }
}
```

Solution 3

An interesting algorithm, similar to algorithm followed in "Reverse Words in a String II". Reverse 1st half, reverse 2nd half, reverse the whole sequence.

Listing 73: Solution 3

```
public class Solution {
    public void rotate(int[] nums, int k) {
        k = k % nums.length;
        reverse(nums, 0, nums.length - k - 1);
        reverse(nums, nums.length - k, nums.length - 1);
        reverse(nums, 0, nums.length - 1);
    }

    private void reverse(int[] nums, int start, int end){
        int i = start , j = end;
        while (i < j){
            int tmp = nums[i];
            nums[i] = nums[j];
            nums[j] = tmp;

            i++;
            j--;
        }
    }
}
```

Problem 58

263 Ugly Number

```
Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.
 For example, 6, 8 are ugly while 14 is not ugly
  since it includes another prime factor 7.

Note that 1 is typically treated as an ugly number.

https://leetcode.com/problems/ugly-number/
```

Solution

This is quite an easy problem!

Listing 74: Solution

```
public class Solution {
    public boolean isUgly(int num) {
        if (num == 0){
            return false;
        }

        while (num % 2 == 0){
            num /= 2;
        }
        while (num % 3 == 0){
            num /= 3;
        }
        while (num % 5 == 0){
            num /= 5;
        }

        if (num == 1){
            return true;
        }

        return false;
    }
}
```

Problem 59

264 Ugly Number II

```
Write a program to find the n-th ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.
 For example, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of
  the first 10 ugly numbers.

Note that 1 is typically treated as an ugly number.

Hint:

The naive approach is to call isUgly for every number
 until you reach the nth one. Most numbers are not ugly.
  Try to focus your effort on generating only the ugly ones

An ugly number must be multiplied by either 2, 3, or 5 from a smaller ugly number.

The key is how to maintain the order of the ugly numbers.
 Try a similar approach of merging from three sorted lists: L1, L2, and L3.

Assume you have Uk, the kth ugly number. Then Uk+1 must be
 Min(L1 * 2, L2 * 3, L3 * 5).

https://leetcode.com/problems/ugly-number-ii/
```

Solution

Solution is as Hint said.

Listing 75: Solution

```
public class Solution {
    public int nthUglyNumber(int n) {
        int nums[] = new int[n];
        nums[0] = 1;
        int idx2 = 0 , idx3 = 0, idx5 = 0;
        int count = 0;
        while (count < n - 1){
            int val2 = nums[idx2];
            int val3 = nums[idx3];
            int val5 = nums[idx5];
            int min = Math.min(Math.min(val2 * 2 , val3 * 3) , val5 * 5);
            if (min == val2 * 2){
                idx2 ++;
            }
            if (min == val3 * 3){
                idx3 ++;
            }
            if (min == val5 * 5){
                idx5++;
            }
            count ++;
```

Problem 59 continued on next page...

```
                nums[count] = min;
            }
            return nums[n - 1];
        }
}
```

Problem 60

313 Super Ugly Number

```
Write a program to find the nth super ugly number.

Super ugly numbers are positive numbers whose all prime factors
 are in the given prime list primes of size k.
  For example, [1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32]
   is the sequence of the first 12 super ugly numbers
    given primes = [2, 7, 13, 19] of size 4.

Note:
(1) 1 is a super ugly number for any given primes.
(2) The given numbers in primes are in ascending order.
(3) 0 < k <= 100, 0 < n <= 106, 0 < primes[i] < 1000.

https://leetcode.com/problems/ugly-number-ii/
```

Solution

This is just a general circumstance of Problem 264 "Ugly Number II", so just utilize the same algorithm is fine.

Listing 76: Solution

```
public class Solution {
    public int nthSuperUglyNumber(int n, int[] primes) {
        int[] nums = new int[n];
        nums[0] = 1;
        int count = 0;
        int plen = primes.length;
        int[] indexes = new int[plen];
        while (count < n - 1){
            int min = Integer.MAX_VALUE;
            for (int i = 0 ; i < plen ; i ++){
                if (min > primes[i] * nums[indexes[i]]){
                    min = primes[i] * nums[indexes[i]];
                }
            }
            for (int i = 0 ; i < plen ; i ++){
                if (min == primes[i] * nums[indexes[i]]){
                    indexes[i] ++;
                }
            }
            count ++;
            nums[count] = min;
        }
        return nums[n - 1];
    }
}
```

Problem 61

228 Summary Ranges

```
Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

https://leetcode.com/problems/summary-ranges/
```

Solution

An easy problem. Simulation solution works.

Listing 77: Solution

```
public class Solution {
    public List<String> summaryRanges(int[] nums) {
        List<String> result = new ArrayList<String>();
        if (nums.length == 0)
            return result;
        int start = nums[0];
        int end = nums[0];
        int last = nums[0];
        for (int idx = 1 ; idx < nums.length ; idx ++){
            int cur = nums[idx];
            if (cur == last + 1){
                end = cur;
                last = cur;
                continue;
            }
            else{
                if (start != end)
                    result.add(String.valueOf(start) + "->"
                     + String.valueOf(end));
                else
                    result.add(String.valueOf(start));
                start = cur;
                end = cur;
            }
            last = cur;
        }
        if (start != end)
            result.add(String.valueOf(start) + "->" + String.valueOf(end));
        else
            result.add(String.valueOf(start));
        return result;
    }


}
```

Problem 62

88 Merge Sorted Array

```
Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1
 as one sorted array.

Note:
You may assume that nums1 has enough space
 (size that is greater or equal to m + n) to hold additional elements from nums2.
  The number of elements initialized in nums1 and nums2 are m and n respectively.

https://leetcode.com/problems/merge-sorted-array/
```

Solution

We can solve the problem in-place(using a new array is too simple, so that I won't put it here.), in order to resist conflict, we can start from the end of array, the worst situation is all elements in s2 is larger than s1, even so there won't be conflicts.

Listing 78: Solution

```
public class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        if (m == 0){
            for (int i = 0 ; i < nums1.length ; i ++){
                nums1[i] = nums2[i];
            }
        }
        if (n == 0){
            return;
        }
        int idx1 = m - 1;
        int idx2 = n - 1;
        int idx = m + n - 1;
        while (idx >= 0){
            int max;
            if (idx2 < 0){
                max = nums1[idx1--];
            }
            else if (idx1 < 0){
                max = nums2[idx2--];
            }
            else if (nums1[idx1] > nums2[idx2]){
                max = nums1[idx1 --];
            }
            else{
                max = nums2[idx2 --];
            }
            nums1[idx--] = max;
        }
    }
}
```

Problem 63

279 Perfect Squares

```
Given a positive integer n, find the least number of perfect square numbers
 (for example, 1, 4, 9, 16, ...) which sum to n.

For example, given n = 12, return 3 because 12 = 4 + 4 + 4; given n = 13,
 return 2 because 13 = 4 + 9.

https://leetcode.com/problems/perfect-squares/
```

Solution 1

Tags helps a lot. I found a tag named breadth-first search, then I realize that such combination problem which finds minimum count is perfect for BFS. Add to each element *poll*ed from queue from biggest available square number, if sum is larger then continue, smaller then $offer$ to queue, finally equal to get result. Next step is to try the DP tag...

Listing 79: Solution 1

```
public class Solution {
    public int numSquares(int n) {
        Queue<Integer> counts = new LinkedList<Integer>();
        Queue<Integer> values = new LinkedList<Integer>();
        ArrayList<Integer> squares = new ArrayList<Integer>();
        for (int i = 1 ; i <= n ; i ++){
            int square = i * i;
            if (square > n){
                break;
            }
            else{
                if (square == n){
                    return 1;
                }
                squares.add(square);
            }
        }
        for (int i = squares.size() - 1 ; i >= 0 ; i--){
            values.offer(squares.get(i));
            counts.offer(1);
        }
        int count = -1;
        while (!values.isEmpty()){
            int value = values.poll();
            count = counts.poll();

            for(int i = squares.size() - 1 ; i >= 0 ; i --){
                if (value < squares.get(i)){
                    continue;
                }
                if (value + squares.get(i) > n){
                    continue;
                }
```

```
                else if (value + squares.get(i) == n){
                    return count + 1;
                }
                else{
                    values.offer(value + squares.get(i));
                    counts.offer(count + 1);
                }
            }
        }
        return count;
    }
}
```

Solution 2

Train of thought is similar to 0-1 knapsack problems. $dp[i] = mindp[i - square_k] + 1$

Listing 80: Solution 2

```
public class Solution {
    public int numSquares(int n) {
        int[] dp = new int[n + 1];
        dp[0] = 1;
        dp[1] = 1;
        ArrayList<Integer> squares = new ArrayList<Integer>();
        for (int i = 1 ; i <= n ; i ++){
            if (i * i > n){
                break;
            }
            else{
                squares.add(i * i);
            }
        }
        for (int i = 2 ; i <= n ; i++){
            int min = Integer.MAX_VALUE;
            for (int j = 0 ; j < squares.size() ; j++){
                if (squares.get(j) > i){
                    break;
                }
                else if (squares.get(j) == i){
                    min = 1;
                    break;
                }
                else{
                   if (dp[i - squares.get(j)] + 1 < min){
                       min = dp[i - squares.get(j)] + 1;
                   }
                }
            }
            dp[i] = min;
        }
        return dp[n];
    }
}
```

Solution 3

Solution 3 is pretty mathematically. Lagrange's four-square theorem states that every natural number can be represented by the sum of no more than 4 square numbers.

还是利用四平方和定理，但是用到了很多技巧，这个解法是参考网上的答案：根据四平方和定理，任意一个正整数均可表示为4个整数的平方和，其实是可以表示为4个以内的平方数之和，那么就是说返回结果只有1,2,3或4其中的一个，首先我们将数字化简一下，由于一个数如果含有因子4，那么我们可以把4都去掉，并不影响结果，比如2和8,3和12等等，返回的结果都相同，读者可自行举更多的栗子。还有一个可以化简的地方就是，如果一个数除以8余7的话，那么肯定是由4个完全平方数组成，这里就不证明了，因为我也不会证明，读者可自行举例验证。那么做完两步后，一个很大的数有可能就会变得很小了，大大减少了运算时间，下面我们就来尝试的将其拆为两个平方数之和，如果拆成功了那么就会返回1或2，因为其中一个平方数可能为0. (注：由于输入的n是正整数，所以不存在两个平方数均为0的情况)。注意下面的!!a + !!b这个表达式，可能很多人不太理解这个的意思，其实很简单，感叹号!表示逻辑取反，那么一个正整数逻辑取反为0，再取反为1，所以用两个感叹号!!的作用就是看a和b是否为正整数，都为正整数的话返回2，只有一个是正整数的话返回1，
参见代码如下：

Listing 81: Solution 3

```
class Solution {
public:
    int numSquares(int n) {
        while (n % 4 == 0) n /= 4;
        if (n % 8 == 7) return 4;
        for (int a = 0; a * a <= n; ++a) {
            int b = sqrt(n - a * a);
            if (a * a + b * b == n) {
                return !!a + !!b;
            }
        }
        return 3;
    }
};
```

Problem 64

322 Coin Change

```
You are given coins of different denominations and a total amount of money amount.
 Write a function to compute the fewest number of coins that
  you need to make up that amount.
   If that amount of money cannot be made up by any combination of the coins,
    return -1.

Example 1:
coins = [1, 2, 5], amount = 11
return 3 (11 = 5 + 5 + 1)

Example 2:
coins = [2], amount = 3
return -1.

Note:
You may assume that you have an infinite number of each kind of coin.

https://leetcode.com/problems/coin-change/
```

Solution

Knapsack problem. Simple but I'm confused!

Listing 82: Solution

```
public class Solution {
    public int coinChange(int[] coins, int amount) {
        int[] dp = new int[amount + 1];
        for (int i = 0 ; i < dp.length ; i ++){
            dp[i] = Integer.MAX_VALUE;
        }
        dp[0] = 0;
        for (int coinIdx = 0 ; coinIdx < coins.length ; coinIdx ++){
            for (int money = coins[coinIdx] ; money <= amount ; money ++){
                if (dp[money - coins[coinIdx]] == Integer.MAX_VALUE){
                    continue;
                }
                dp[money] =
                    Math.min(dp[money], dp[money - coins[coinIdx]] + 1);
            }
        }

        return dp[amount] == Integer.MAX_VALUE ? -1 : dp[amount];
    }
}
```

We can also switch the outer and inner for loop, then compared it with Problem 279 "Perfect Squares". Notice something?

Listing 83: Solution1.2

Problem 64 continued on next page...

```java
public int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    dp[0] = 0;
    for(int i = 1; i <= amount; i++){
        dp[i] = Integer.MAX_VALUE;
        for(int j = 0; j < coins.length; j++){
            // remember to check i - coins[j] >= 0, or you will get array out of bounds
            if(i - coins[j] >= 0 && dp[i - coins[j]] != Integer.MAX_VALUE) {
                dp[i] = Math.min(dp[i],dp[i-coins[j]] + 1);
            }

        }
    }
    return (dp[amount] == Integer.MAX_VALUE)?-1:dp[amount];
}
```

Problem 65

111 Minimum Depth of Binary Tree

```
Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path
 from the root node down to the nearest leaf node.

https://leetcode.com/problems/minimum-depth-of-binary-tree/
```

Solution 1

An easy problem. Solution 1 uses DFS. Don't forget to check the condition when one of the current node's child is null while another isn't.

Listing 84: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int minDepth(TreeNode root) {
        if (root == null){
            return 0;
        }
        if (root.left == null && root.right != null){
            return minDepth(root.right) + 1;
        }
        if (root.left != null && root.right == null){
            return minDepth(root.left) + 1;
        }
        return Math.min(minDepth(root.left), minDepth(root.right)) + 1;
    }
}
```

Solution 2

BFS is simpler, faster and more natural. We can stop whenever we first meet a node without children. Edge case is root == null.

Listing 85: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
```

Problem 65 continued on next page...

```
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int minDepth(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        if (root == null){
            return 0;
        }
        queue.add(root);
        int minDepth = 1;
        while (!queue.isEmpty()){
            ArrayList<TreeNode> list = new ArrayList<TreeNode>();
            int count = 0;
            while (!queue.isEmpty()){
                list.add(queue.poll());
                count ++;
            }
            for (int i = 0 ; i < count ; i ++){
                TreeNode cur = list.get(i);
                if (cur == null || (cur.left == null && cur.right == null)){
                    return minDepth;
                }else if (cur.left == null){
                    queue.offer(cur.right);
                }else if (cur.right == null){
                    queue.offer(cur.left);
                }else{
                    queue.offer(cur.left);
                    queue.offer(cur.right);
                }
            }
            minDepth ++;
        }
        return minDepth;
    }
}
```

Problem 66

104 Maximum Depth of Binary Tree

```
Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path
 from the root node down to the farthest leaf node.

https://leetcode.com/problems/maximum-depth-of-binary-tree/
```

Solution 1

Similar to Problem 104 "Minimum Depth of Binary Tree". Solution 1 uses DFS. It's interesting that we don't need to check if one child is null while another not, since we are fetching the maximum value.

Listing 86: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null){
            return 0;
        }

        return Math.max(maxDepth(root.left) , maxDepth(root.right)) + 1;
    }
}
```

Solution 2

When using BFS, se stop when all levels are traversed, namely when queue is empty. Edge case is root == null. Don't forget to check if input parameter is null (when using BFS)!

Listing 87: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDepth(TreeNode root) {
```

Problem 66 continued on next page...

```
        if (root == null){
            return 0;
        }
        Queue<TreeNode> queue = new LinkedList<TreeNode>();

        int depth = 0;
        queue.offer(root);

        while (!queue.isEmpty()){
            depth ++;
            List<TreeNode> levelNodeList = new ArrayList<TreeNode>();

            while (!queue.isEmpty()){
                levelNodeList.add(queue.poll());
            }

            for (TreeNode levelNode : levelNodeList){
                if (levelNode.left != null){
                    queue.offer(levelNode.left);
                }
                if (levelNode.right != null){
                    queue.offer(levelNode.right);
                }
            }
        }

        return depth;
    }
}
```

Problem 67

215 Kth Largest Element in an Array

```
Find the kth largest element in an unsorted array.
 Note that it is the kth largest element in the sorted order,
  not the kth distinct element.

For example,
Given [3,2,1,5,6,4] and k = 2, return 5.

Note:
You may assume k is always valid, 1 $<=$ k $<=$ array's length.

https://leetcode.com/problems/minimum-depth-of-binary-tree/
```

Solution 1

This is a good problem! Solution 1 is great, it utilize half of QuickSort – the partition part. During quicksort, when we decide to put a pivot, the pivot is at exactly where it should be in a sorted array(then numbers on both sides are sorted respectively.). So we can check if the index of pivot is what we want - k, or more accurate, nums.length - k, if so, the question is answered.

Listing 88: Solution 1.1

```
public class Solution {
    private int partition(int[] nums, int low, int high){
        if (low == high){
            return low;
        }
        int i = low;
        int j = high;
        int pivot = nums[i];
        while (i < j ){
            while (i < j && nums[j] >= pivot){
                j--;
            }
            // actually, this is not needed
            if (i == j){
                break;
            }
            nums[i] = nums[j];
            while(i < j && nums[i] <= pivot){
                i ++;
            }
            // actually, this is not needed
            if (i == j){
                break;
            }
            nums[j] = nums[i];
        }
        nums[i] = pivot;
        return i;
    }
```

```
30
    public int findKthLargest(int[] nums, int k) {
        if (nums.length == 0 || nums.length < k || k == 0){
            return -1;
        }
35      k = nums.length - k;
        int low = 0 , high = nums.length - 1;
        int pivotIdx = partition(nums, low, high);
        while(low < high && pivotIdx != k){
            if (pivotIdx > k){
40              high = pivotIdx - 1;
            }
            else{
                low = pivotIdx + 1;
            }
45
            pivotIdx = partition(nums, low, high);
        }
        return nums[pivotIdx];
    }
50  }
```

I found another solution on Internet which also has a fantastic train of thought.(Divide and Conquer, I almost missed this point.)

In this problem, k means the index, namely how many numbers are there that is equal(itself) and bigger than the element. Then the author partitioned two array, storing elements larger and elements smaller,

when length of array of larger numbers is not adequate(k > bigger.length), find in larger set for the element that still have k elements larger than itself;

if k - 1 < bigger.length, then find in the smaller number set, k should be transformed into k - length - 1(no longer need k bigger numbers).

If (k - 1 == bigger.length), we get the result.

Listing 89: Solution 1.2

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        int length = nums.size();
5       if (length == 1) {
            return nums[0];
        }
        vector<int> left;
        vector<int> right;
10
        for (int index=1; index<length; index++) {
            if (nums[index] > nums[0]) {
                right.push_back(nums[index]);
            }else{
15              left.push_back(nums[index]);
            }
        }
        length = right.size();
        if (length >=k) {
```

```
            return findKthLargest(right, k);
        }else if(length == k-1){
            return nums[0];
        }else{
            return findKthLargest(left, k-length-1);
        }
    }
};
```

Solution 2

We can also use Heap sort to solve this problem. Since PriorityQueue is made of heap, we use PriorityQueue to solve the problem. An optimization can be take place to reduce $O(nlogn)runtime$ to $O(nlogk)$, to generate a heap with k elements at first, and insert remaining n - k elements into heap if the element is , finally get the element from the top.

Explain it. If we are finding the kth biggest element, we shall build a heap with small root using the first kth elements, so the root of the heap is exactly the kth biggest element(1st smallest) among all elements. For the rest of them, if the element is larger than top element, it will affect the ranking of elements, so poll the top element and add current element into heap; if element is smaller, just ignore it.

Listing 90: Solution 2

```
public class Solution {
    public int findKthLargest(int[] nums, int k) {
        Queue<Integer> heap = new PriorityQueue<Integer>();
        for (int idx = 0 ; idx < k ; idx ++){
            heap.add(nums[idx]);
        }
        for (int idx = k ; idx < nums.length ; idx ++){
            if (nums[idx] <= heap.peek()){
                continue;
            }
            else{
                heap.poll();
                heap.add(nums[idx]);
            }
        }
        return heap.peek();
    }
}
```

Problem 68

102 Binary Tree Level Order Traversal

```
Given a binary tree, return the level order traversal of its nodes' values.
 (ie, from left to right, level by level).

For example:
Given binary tree {3,9,20,#,#,15,7},
    3
   / \
  9  20
    /  \
   15   7
return its level order traversal as:
[
  [3],
  [9,20],
  [15,7]
]
confused what "{1,#,2,3}" means?


OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal,
 where '#' signifies a path terminator where no node exists below.

Here's an example:
   1
  / \
 2   3
    /
   4
    \
     5
The above binary tree is serialized as "{1,2,3,#,#,4,#,#,5}".

https://leetcode.com/problems/binary-tree-level-order-traversal/
```

Solution 1

A standard BFS Problem.

Listing 91: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
```

Problem 68 continued on next page...

```
    public List<List<Integer>> levelOrder(TreeNode root) {
        if (root == null){
            return new ArrayList<List<Integer>>();
        }
        List<List<Integer>> result = new ArrayList();
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.add(root);
        while (!queue.isEmpty()){
            List<Integer> list = new ArrayList<Integer>();
            List<TreeNode> nodeList = new ArrayList<TreeNode>();
            while (!queue.isEmpty()){
                TreeNode node = queue.poll();
                nodeList.add(node);
            }
            for (TreeNode node : nodeList){
                list.add(node.val);
                if (node.left != null){
                    queue.offer(node.left);
                }
                if (node.right != null){
                    queue.offer(node.right);
                }
            }
            result.add(list);
        }
        return result;
    }
}
```

Solution 2

Use recursion(somewhat DFS) to maintain nodes related to its level.

Listing 92: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();

    public void traverse(TreeNode root, int level){
        if (root == null){
            return;
        }
        if (level == result.size()){
            result.add(new ArrayList<Integer>());
        }
        result.get(level).add(root.val);
```

```
        traverse(root.left , level + 1);
        traverse(root.right, level + 1);
    }
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        traverse(root, 0);

        return result;
    }
}
```

Problem 69

107 Binary Tree Level Order Traversal II

```
Given a binary tree, return the level order traversal of its nodes' values.
 (ie, from left to right, level by level).

For example:
Given binary tree {3,9,20,#,#,15,7},
    3
   / \
  9  20
    /  \
   15   7
return its bottom-up level order traversal as:
[
  [15,7],
  [9,20],
  [3]
]
confused what "{1,#,2,3}" means?


OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal,
 where '#' signifies a path terminator where no node exists below.

Here's an example:
   1
  / \
 2   3
    /
   4
    \
     5
The above binary tree is serialized as "{1,2,3,#,#,4,#,#,5}".

https://leetcode.com/problems/binary-tree-level-order-traversal-ii/
```

Solution

Just add a reversion on list among Problem 102 "Binary Tree Level Order Traversal"

Listing 93: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
```

```
    List<List<Integer>> result = new ArrayList<List<Integer>>();

    public void traverse(TreeNode root, int level){
        if (root == null){
            return;
        }
        if (level == result.size()){
            result.add(new ArrayList<Integer>());
        }
        result.get(level).add(root.val);

        traverse(root.left , level + 1);
        traverse(root.right, level + 1);
    }
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        traverse(root, 0);

        reverse(result);
        return result;
    }

    public void reverse(List<List<Integer>> list){
        int i = 0 , j = list.size() - 1;
        while (i < j){
            List<Integer> obj = list.get(i);
            list.set(i, list.get(j));
            list.set(j, obj);

            i++;
            j--;
        }
    }
}
```

Problem 70

103 Binary Tree Zigzag Level Order Traversal

```
Given a binary tree, return the zigzag level order traversal
 of its nodes' values.
  (ie, from left to right, then right to left for the next level
   and alternate between).

For example:
Given binary tree {3,9,20,#,#,15,7},
    3
   / \
  9  20
    /  \
   15   7
return its zigzag level order traversal as:
[
  [3],
  [20,9],
  [15,7]
]

https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/
```

Solution

Not as complicated as what is said on Internet, similar to Problem 102 "Binary Tree Level Order Traversal", just to change the sequence of printing...

Listing 94: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        if (root == null){
            return new ArrayList<List<Integer>>();
        }
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        queue.offer(root);
        int direction = 0;
        while (!queue.isEmpty()){
            List<TreeNode> list = new ArrayList<TreeNode>();
            List<Integer> values = new ArrayList<Integer>();
            while (!queue.isEmpty()){
                list.add(queue.poll());
```

Problem 70 continued on next page...

```
            }

            if (direction == 0){
                for (TreeNode node : list){
                    values.add(node.val);
                }
            }else{
                for (int idx = list.size() - 1; idx >= 0 ; idx --){
                    TreeNode node = list.get(idx);
                    values.add(node.val);
                }
            }
            for (TreeNode node : list){
                if (node.left != null){
                    queue.offer(node.left);
                }
                if (node.right != null){
                    queue.offer(node.right);
                }
            }
            direction = 1 - direction;
            result.add(values);
        }
        return result;
    }
}
```

Problem 71

100 Same Tree

```
Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical
 and the nodes have the same value.

https://leetcode.com/problems/same-tree/
```

Solution

Similar to Problem 101 “Symmetric Tree”, except for the nodes for comparison.

Listing 95: Solution

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

Problem 72

101 Symmetric Tree

```
Given a binary tree, check whether it is a mirror of itself
 (ie, symmetric around its center).

For example, this binary tree is symmetric:

    1
   / \
  2   2
 / \ / \
3  4 4  3
But the following is not:
    1
   / \
  2   2
   \   \
   3    3
Note:
Bonus points if you could solve it both recursively and iteratively.

https://leetcode.com/problems/symmetric-tree/
```

Solution 1

Recursion solution is kind of simple(but I still can't figure it out at first sight...).

Listing 96: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean checkSymmetric(TreeNode node1 , TreeNode node2){
        if (node1 == null && node2 == null){
            return true;
        }
        else if (node1 != null && node2 != null && node1.val == node2.val){
            return checkSymmetric(node1.left , node2.right)
             && checkSymmetric(node1.right , node2.left);
        }
        else{
            return false;
        }
    }

    public boolean isSymmetric(TreeNode root) {
```

Problem 72 continued on next page...

```
        if (root == null){
            return true;
        }
        return checkSymmetric(root.left, root.right);
    }
}
```

Solution 2

BFS solution by myself, given the fact that if the tree is symmetric, the indexes of nodes in pair must sum up to $2^{level} - 1$, here level is the level in tree, root is 0, children of root is 1... index is its serial number from left to right, index of current node is 2*parent or 2*parent+1(depending on whether the node is on left side or right side.)

Listing 97: Solution 2.1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null){
            return true;
        }
        if (root.left == null && root.right == null){
            return true;
        }
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        Queue<Integer> indexes = new LinkedList<Integer>();
        queue.offer(root);
        indexes.offer(0);
        int level = 0;
        while (!queue.isEmpty()){
            ArrayList<TreeNode> list = new ArrayList<TreeNode>();
            ArrayList<Integer> indexList = new ArrayList<Integer>();

            while (!queue.isEmpty()){
                list.add(queue.poll());
                indexList.add(indexes.poll());
            }
            if (list.size() % 2 == 1 && level != 0){
                return false;
            }
            int i = 0, j = list.size() - 1;
            while (i < j){
                if (list.get(i).val != list.get(j).val
                 || indexList.get(i) + indexList.get(j) != (int)Math.pow(2, level) - 1){
                    return false;
```

```
                }

                i++;
                j--;
            }
            for (int idx = 0 ; idx < list.size() ; idx ++){
                TreeNode node = list.get(idx);
                int index = indexList.get(idx);
                if (node.left != null){
                    queue.offer(node.left);
                    indexes.offer(index * 2);
                }
                if (node.right != null){
                    queue.offer(node.right);
                    indexes.offer(index * 2 + 1);
                }
            }
            level ++;
        }
        return true;
    }
}
```

Solution on Internet, much simpler than my solution...

Listing 98: Solution 2.2

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode *root) {
        // Start typing your C/C++ solution below
        // DO NOT write int main() function
        if(root==NULL)return true;
        queue<TreeNode*> lt,rt;
        if(root->left) lt.push(root->left);
        if(root->right) rt.push(root->right);
        TreeNode* l;
        TreeNode* r;
        while(!lt.empty() && !rt.empty())
        {
            l = lt.front();lt.pop();
            r = rt.front();rt.pop();
            if(l == NULL && r == NULL) continue;
            if(l == NULL || r == NULL) return false;
            if(l->val != r->val) return false;
```

```
            lt.push(l->left);
            lt.push(l->right);
            rt.push(r->right);
            rt.push(r->left);
        }
        if(lt.empty() && rt.empty())
            return true;
        else
            return false;
    }
};
```

Problem 73

62 Unique Paths

```
A robot is located at the top-left corner of a m * n grid
 (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time.
 The robot is trying to reach the bottom-right corner of the grid
  (marked 'Finish' in the diagram below).

Figure is shown below the description.

How many possible unique paths are there?


Above is a 3 * 7 grid. How many possible unique paths are there?

Note: m and n will be at most 100.

https://leetcode.com/problems/unique-paths/
```

Solution

This is maybe the 1st dp problem I can solve by myself...

Listing 99: Solution

```
public class Solution {
    public int uniquePaths(int m, int n) {
        int dp[][] = new int[m][n];

        for (int i = 0 ; i < m ; i++){
            dp[i][0] = 1;
        }
        for (int i = 0 ; i < n ; i ++){
            dp[0][i] = 1;
        }

        for (int i = 1 ; i < m ; i ++){
```

```
            for (int j = 1 ; j < n ; j ++){
                dp[i][j]  = dp[i - 1][j] + dp[i][j - 1];
            }
        }

        return dp[m - 1][n - 1];
    }
}
```

Problem 74

63 Unique Paths II

```
Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids.
 How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example,
There is one obstacle in the middle of a 3x3 grid as illustrated below.

[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
The total number of unique paths is 2.

Note: m and n will be at most 100.

https://leetcode.com/problems/unique-paths-ii/
```

Solution

Just add checking process to see if there are obstacles, if so, dp[i][j] = 0; Mention that in initialization phase(to initialize 1st row and 1st column), when we first met obstacle, the latter grids are all impassable, then dp value should be 0 too.

Listing 100: Solution

```
public class Solution {
    public int uniquePathsWithObstacles(int[][] obstacleGrid) {
        int m = obstacleGrid.length;
        if (m == 0){
            return 0;
        }
        int n = obstacleGrid[0].length;
```

```
        if (n == 0){
            return 0;
        }
        int dp[][] = new int[m][n];
        for (int i = 0 ; i < m ; i ++){
            if (obstacleGrid[i][0] == 0){
                dp[i][0] = 1;
            }
            else{
                break;
            }
        }
        for (int i = 0 ; i < n ; i ++){
            if (obstacleGrid[0][i] == 0){
                dp[0][i] = 1;
            }
            else{
                break;
            }
        }
        for (int i = 1 ; i < m ; i ++){
            for (int j = 1 ; j < n ; j++){
                dp[i][j] = (obstacleGrid[i][j] == 1) ?
                     0 : dp[i - 1][j] + dp[i][j - 1];
            }
        }
        return dp[m - 1][n - 1];
    }
}
```

Problem 75

64 Minimum Path Sum

```
Given a m x n grid filled with non-negative numbers,
 find a path from top left to bottom right which minimizes
  the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

https://leetcode.com/problems/minimum-path-sum/
```

Solution

A bit more complicated than Problem 62 “Unique Paths”, see code in listing 101

Listing 101: Solution

```
public class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        if (m == 0){
            return 0;
        }
        int n = grid[0].length;
        int dp[][] = new int[m][n];
        dp[0][0] = grid[0][0];
        for (int i = 1 ; i < m ; i ++){
            dp[i][0] = grid[i][0] + dp[i - 1][0];
        }
        for (int i = 1 ; i < n ; i ++){
            dp[0][i] = grid[0][i] + dp[0][i - 1];
        }
        for (int i = 1 ; i < grid.length; i ++){
            for (int j = 1 ; j < grid[0].length ; j ++){
                dp[i][j] = Math.min(dp[i - 1][j] , dp[i][j - 1]) + grid[i][j];
            }
        }
        return dp[m - 1][n - 1];
    }
}
```

Problem 76

174 Dungeon Game

```
The demons had captured the princess (P) and imprisoned her
 in the bottom-right corner of a dungeon.
  The dungeon consists of M x N rooms laid out in a 2D grid.
   Our valiant knight (K) was initially positioned
    in the top-left room and must fight his way through
     the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer.
 If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health
 (negative integers) upon entering these rooms;
  other rooms are either empty (0's) or contain magic orbs
   that increase the knight's health (positive integers).

In order to reach the princess as quickly as possible,
 the knight decides to move only rightward or downward in each step.


Write a function to determine the knight's minimum initial health
 so that he is able to rescue the princess.

For example, given the dungeon below, the initial health of the knight
 must be at least 7 if he follows
  the optimal path RIGHT$->$ RIGHT $->$ DOWN $->$ DOWN.

 -----------------
|-2(K)|  -3 |  3  |
|-----------------|
|  -5 |    -10 |  1  |
|-----------------|
| 10  |    30  |-5(P)|
 -----------------

Notes:

The knight's health has no upper bound.
Any room can contain threats or power-ups,
 even the first room the knight enters
  and the bottom-right room where the princess is imprisoned.

https://leetcode.com/problems/dungeon-game/
```

Solution

Much more complicated than Problems 'Unique Path's. You'll find dynamic programming from top left corner is extremely difficult and confusing. Try start from the bottom right corner to get guaranteed health value. Another thing needs to be carefully taken into consideration is when doing DP calculation, since we are starting from bottom right corner(get guaranteed health), extra positive health are useless, since the Knight cannot respawn with extra health, we still need to guarantee his health of his previous steps, so 0 is

Problem 76 continued on next page...

enough.

Listing 102: Solution

```
public class Solution {
    public int calculateMinimumHP(int[][] dungeon) {
        int m = dungeon.length;
        if (m == 0){
            return 1;
        }
        int n = dungeon[0].length;

        int guaranteedHealth[][] = new int[m][n];
        guaranteedHealth[m - 1][n - 1] = (dungeon[m - 1][n - 1] > 0)
         ? 0 : Math.abs(dungeon[m - 1][n - 1]);

        for (int i = m - 2 ; i >= 0 ; i --){
            int health = guaranteedHealth[i + 1][n - 1] - dungeon[i][n - 1];
            guaranteedHealth[i][n - 1] = Math.max(0, health);
        }
        for (int i = n - 2 ; i >= 0 ; i --){
            int health = guaranteedHealth[m - 1][i + 1] - dungeon[m - 1][i];
            guaranteedHealth[m - 1][i] = Math.max(0, health);
        }
        for (int i = m - 2 ; i >= 0 ; i--){
            for (int j = n - 2 ; j >= 0 ; j --){
                guaranteedHealth[i][j] = Math.max(
                    Math.min(guaranteedHealth[i + 1][j]
                     , guaranteedHealth[i][j + 1]) - dungeon[i][j], 0);
            }
        }

        return guaranteedHealth[0][0] + 1;
    }
}
```

Problem 77

232 Implement Queue using Stacks

```
Implement the following operations of a queue using stacks.

push(x) -- Push element x to the back of queue.
pop() -- Removes the element from in front of queue.
peek() -- Get the front element.
empty() -- Return whether the queue is empty.

Notes:
    You must use only standard operations of a stack
     -- which means only push to top, peek/pop from top, size,
      and is empty operations are valid.

    Depending on your language, stack may not be supported natively.
     You may simulate a stack by using a list or deque (double-ended queue),
      as long as you use only standard operations of a stack.

    You may assume that all operations are valid
     (for example, no pop or peek operations will be called on an empty queue).

https://leetcode.com/problems/implement-queue-using-stacks/
```

Solution

Use 2 containers to implement another kind of container, just see listings.

Listing 103: Solution

```
class MyQueue {
    Stack<Integer> stk1 = new Stack<Integer>();
    Stack<Integer> stk2 = new Stack<Integer>();
    // Push element x to the back of queue.
    public void push(int x) {
        stk1.push(x);
    }

    // Removes the element from in front of queue.
    public void pop() {
        if (stk2.isEmpty()){
            while(!stk1.isEmpty()){
                stk2.push(stk1.pop());
            }
        }
        stk2.pop();
    }

    // Get the front element.
    public int peek() {

        if (stk2.isEmpty()){
            while(!stk1.isEmpty()){
                stk2.push(stk1.pop());
```

```
            }
        }
        return stk2.peek();
    }

    // Return whether the queue is empty.
    public boolean empty() {
        return (stk1.isEmpty() && stk2.isEmpty());
    }
}
```

Problem 78

225 Implement Stack using Queues

```
Implement the following operations of a stack using queues.

push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
empty() -- Return whether the stack is empty.

Notes:

    You must use only standard operations of a queue
     -- which means only push to back, peek/pop from front,
      size, and is empty operations are valid.

    Depending on your language, queue may not be supported natively.
     You may simulate a queue by using a list or deque (double-ended queue),
      as long as you use only standard operations of a queue.

    You may assume that all operations are valid
     (for example, no pop or top operations will be called on an empty stack).



https://leetcode.com/problems/implement-queue-using-stacks/
```

Solution

See listings.

Listing 104: Solution

```
class MyStack {
    Queue<Integer> queue1 = new LinkedList<Integer>();
    Queue<Integer> queue2 = new LinkedList<Integer>();

    // Push element x onto stack.
    public void push(int x) {
        queue2.offer(x);

        while (!queue1.isEmpty()){
            queue2.offer(queue1.poll());
        }

        while (!queue2.isEmpty()){
            queue1.offer(queue2.poll());
        }
    }

    // Removes the element on top of the stack.
    public void pop() {
        queue1.poll();
    }
```

```
    // Get the top element.
    public int top() {
        return queue1.peek();
    }

    // Return whether the stack is empty.
    public boolean empty() {
        return (queue1.isEmpty() && queue2.isEmpty());
    }
}
```

Problem 79

148 Sort List

```
Sort a linked list in O(n log n) time using constant space complexity.

https://leetcode.com/problems/sort-list/
```

Solution

First, review all kinds of sorting methods!

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	Shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	不稳定
归并排序		$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数。

We can see that Heap sort, Quick sort and Merge sort satisfy the requirements of problem. They all need extra space at first sight, which may not meet the space requirement. However, when we look at the attribute of LinkedList, we can find out that we do not need to allocate extra space to store the sorted list after merging, what we need to do is just to reconnect the nodes.

Listing 105: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null){
            return head;
        }
        ListNode slow = head;
        ListNode fast = head;
        ListNode prev = slow;
        while (fast != null && fast.next != null){
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }
```

```
        prev.next = null;
        ListNode l1 = sortList(head);
        ListNode l2 = sortList(slow);
        return mergeList(l1, l2);
    }

    public ListNode mergeList(ListNode l1, ListNode l2){
        ListNode dummy = new ListNode(-1);
        ListNode cur = dummy;
        ListNode cur1 = l1, cur2 = l2;
        while (cur1 != null && cur2 != null){
            if (cur1.val < cur2.val){
                cur.next = cur1;
                cur1 = cur1.next;
                cur = cur.next;
            }
            else{
                cur.next = cur2;
                cur2 = cur2.next;
                cur = cur.next;
            }
        }
        while (cur1 != null){
            cur.next = cur1;
            cur = cur.next;
            cur1 = cur1.next;
        }
        while (cur2 != null){
            cur.next = cur2;
            cur = cur.next;
            cur2 = cur2.next;
        }
        cur.next = null;
        return dummy.next;
    }
}
```

Problem 80

147 Insertion Sort List

```
Sort a linked list using insertion sort.

https://leetcode.com/problems/insertion-sort-list/
```

Solution

It's an easy but a complicated problem which is not easy to be bug-free.

Listing 106: Solution

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode insertionSortList(ListNode head) {
        if(head == null || head.next == null){
            return head;
        }
        ListNode dummy = new ListNode(Integer.MIN_VALUE);
        dummy.next = head;
        ListNode curNode = head.next;
        ListNode prevNode = head;

        while (curNode != null){
            // find insertion position
            boolean found = false;
            ListNode searchNode = dummy.next;
            ListNode searchPrevNode = dummy;
            while (searchNode != curNode){
                if (curNode.val < searchNode.val){
                    // reconnect curNode
                    prevNode.next = curNode.next;

                    // connect searchNode & currentNode
                    searchPrevNode.next = curNode;
                    curNode.next = searchNode;

                    curNode = prevNode.next;
                    found = true;
                    break;
                }
                searchPrevNode = searchNode;
                searchNode = searchNode.next;
            }
            if (curNode == searchNode){
                // no need to change
```

```
                prevNode = curNode;
                curNode = curNode.next;
                continue;
            }

            // insert node
            if (found){
                continue;
            }
            prevNode = curNode;
            curNode = curNode.next;
        }

        return dummy.next;
    }
}
```

Problem 81

86 Partition List

```
Given a linked list and a value x, partition it such that all nodes
 less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes
 in each of the two partitions.

For example,
Given 1->4->3->2->5->2 and x = 3,
return 1->2->2->4->3->5.

https://leetcode.com/problems/partition-list/
```

Solution

Maintain 2 List, one for elements less than x, one for elements larger or equal to x, finally reconnect them.

Listing 107: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode partition(ListNode head, int x) {
        if (head == null || head.next == null){
            return head;
        }

        ListNode dummySmall = new ListNode(-1);
        ListNode dummyLarge = new ListNode(-1);

        ListNode tailSmall = dummySmall;
        ListNode tailLarge = dummyLarge;

        ListNode curNode = head;
        while (curNode != null){
            if (curNode.val < x){
                tailSmall.next = curNode;
                tailSmall = tailSmall.next;
            }
            else if(curNode.val >= x){
                tailLarge.next = curNode;
                tailLarge = tailLarge.next;
            }

            curNode = curNode.next;
        }
```

```
            tailLarge.next = null;

            tailSmall.next = dummyLarge.next;

            return dummySmall.next;
        }
}
```

Problem 82

136 Single Number

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

https://leetcode.com/problems/single-number/
```

Solution 1

Review the whole set of "Single number(I/II/III)", an interesting set of problems.

First solution is to use a Hashset, if the element appears once, add it into Set; if it has appeared, remove it from Set. Finally the only element remains in Set is the 'Single Number'.

Listing 108: Solution 1

```
public class Solution {
    public int singleNumber(int[] nums) {
        Set<Integer> set = new HashSet<Integer>();
        for (int num : nums){
            if (set.contains(num)){
                set.remove(num);
            }
            else{
                set.add(num);
            }
        }
        Iterator<Integer> ite = set.iterator();
        if (ite.hasNext()){
            return ite.next();
        }
        else{
            return 0;
        }
    }
}
```

Solution 2

Bit manipulation is more proper to this SET of problems. Xor is the key to this problem. Apply xor to 2 same numbers will get the result 0, and since xor is commutative(a xor b = b xor a) and associative(a xor b xor c = a xor (b xor c)), xor all numbers will get the single number!

In fact, solution in next Problem "Single Number II" is more proper for this SET of problems.

Listing 109: Solution 2

```
public class Solution {
    public int singleNumber(int[] nums) {
        if (nums.length == 1){
            return nums[0];
        }
        int number = nums[0];
        for (int idx = 1; idx < nums.length ; idx++){
            number ^= nums[idx];
```

```
            }

        return number;
    }
}
```

Problem 83

137 Single Number II

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

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

https://leetcode.com/problems/single-number-ii/
```

Solution

Solution 1 of Problem 136 "Single Number" is no longer suitable for this problem. Bit manipulation still works, but this time a bit complicated.

E.g.: array consists of 1,2,2,2,3,3,3, namely 001 010 010 010 011 011 011, for units digit, there are four 1s, so the 1 is from the single number... Follow such train of thought, we can solve this problem.

Listing 110: Solution

```
public class Solution {
    public int singleNumber(int[] nums) {
        int result = 0;
        for (int i = 0 ; i < 32 ; i ++){
            int digit = 0;
            int mask = (1 << i);
            for(int idx = 0 ; idx < nums.length ; idx ++){
                if ((nums[idx] & mask) != 0){
                    digit ++;
                }
            }
            if (digit % 3 != 0){
                result |= mask;
            }
        }
        return result;
    }
}
```

Problem 84

260 Single Number III

```
Given an array of numbers nums, in which exactly two elements appear
 only once and all the other elements appear exactly twice.
  Find the two elements that appear only once.

For example:

Given nums = [1, 2, 1, 3, 2, 5], return [3, 5].

Note:
The order of the result is not important. So in the above example,
 [5, 3] is also correct.
Your algorithm should run in linear runtime complexity.
 Could you implement it using only constant space complexity?

https://leetcode.com/problems/single-number-iii/
```

Solution

The key to solve this problem if to separate the two single numbers into 2 different groups so that we can apply algorithm from 'Single Number' to each group.

How to achieve that? First we xor all numbers to get a result from which we can conclude a result that when the digit of the result is 1, it means that 2 single numbers are differ among this digit, so that in terms of all numbers, the number of 1 should be odd number(consisting 1 single number), so that we can group numbers with digit valued 1 and digit valued 0 on that digit into 2 groups. Finally apply xor algorithm to get the single number from each group.

Listing 111: Solution

```
public class Solution {
    public int[] singleNumber(int[] nums) {
        int total = 0;
        int count1 = 0, count2 = 0;
        for (int num : nums){
            total ^= num;
        }
        int mDigit = -1;
        for (int i = 0 ; i < 32 ; i++){
            int mask = (1 << i);
            if ((total & mask) != 0){
                // find the digit
                mDigit = i;
                for (int j = 0 ; j < nums.length ; j++){
                    if ((nums[j] & mask) != 0){
                        count1 ++;
                    }
                    else{
                        count2 ++;
                    }
                }
```

```
                break;
            }
        }
        int[] g1 = new int[count1];
        int[] g2 = new int[count2];
        int idx1 = 0 , idx2 = 0;
        int mask = (1 << mDigit);
        for (int i = 0 ; i < nums.length ; i ++){
            if ((nums[i] & mask) != 0){
                g1[idx1] = nums[i];
                idx1 ++;
            }else{
                g2[idx2] = nums[i];
                idx2 ++;
            }
        }
        int num1 = 0 , num2 = 0;
        for (int num : g1){
            num1 ^= num;
        }
        for (int num : g2){
            num2 ^= num;
        }
        int[] result = new int[2];
        result[0] = num1;
        result[1] = num2;
        return result;
    }
}
```

Problem 85

282 Expression Add Operators

```
Given a string that contains only digits 0-9 and a target value,
 return all possibilities to add binary operators
  (not unary) +, -, or * between the digits so they evaluate to the target value.

Examples:
"123", 6 -> ["1+2+3", "1*2*3"]
"232", 8 -> ["2*3+2", "2+3*2"]
"105", 5 -> ["1*0+5","10-5"]
"00", 0 -> ["0+0", "0-0", "0*0"]
"3456237490", 9191 -> []

https://leetcode.com/problems/expression-add-operators/
```

Solution

A good DFS problem.

The difficulties focus on those edge cases(String startsWith 0 - 01 is illegal; length of remaining string) and how to deal with multiply operation. In this solution uses a 'diff' to save previous result, for example, 1 - 2 * 3, previous result should be -1(1 - 2), but 1-2*3=-5, which means we should apply -1 - (-2) + (-2 *3), here diff = -2, namely previous number.

However, when a new problem arise, I still can't get the solution method!

Listing 112: Solution

```
public class Solution {
    List<String> result = new ArrayList<String>();

    public List<String> addOperators(String num, int target) {
        dfs(num, 0, target, 0, "");
        return result;
    }

    public void dfs(String remaining, long cur, int target, long diff,
     String expr){
        if (remaining.length() == 0 && target == cur){
            result.add(expr);

            return;
        }

        for (int i = 1; i <= remaining.length() ; i++){
            String sub = remaining.substring(0, i);
            // important
            if (sub.length() > 1 && sub.charAt(0) == '0'){
                return;
            }
            String curRem = remaining.substring(i);
            long subNum = Long.parseLong(sub);

            if (expr.length() == 0){
```

```
                dfs(curRem, subNum ,target , subNum, sub);
            }
            else{
                dfs(curRem, cur + subNum , target, subNum, expr + "+" + sub);
                dfs(curRem, cur - subNum, target, 0 - subNum, expr + "-" + sub);
                dfs(curRem, cur - diff + diff * subNum , target,
                 diff * subNum, expr + "*" + sub);
            }
        }
    }
}
```

Problem 86

200 Number of Islands

```
Given a 2d grid map of '1's (land) and '0's (water),
 count the number of islands.
  An island is surrounded by water and is formed
   by connecting adjacent lands horizontally or vertically.
    You may assume all four edges of the grid are all surrounded by water.

Example 1:

11110
11010
11000
00000
Answer: 1

Example 2:

11000
11000
00100
00011
Answer: 3

https://leetcode.com/problems/number-of-islands/
```

Solution

It's just a simple DFS problem...

Listing 113: Solution

```
public class Solution {
    public int numIslands(char[][] grid) {
        int count = 0;
        for (int i = 0 ; i < grid.length ; i ++){
            for (int j = 0 ; j < grid[0].length ; j ++){
                if (grid[i][j] == '1'){
                    wander(grid, i, j);
                    count ++;
                }
            }
        }
        return count;
    }

    public void wander(char[][] grid, int i , int j){
        if (i < 0 || j < 0 || i >= grid.length || j >= grid[0].length){
            return;
        }
        if (grid[i][j] == '1'){
            grid[i][j] = 'X';
            wander(grid, i , j + 1);
```

Problem 86 continued on next page...

```
                wander(grid, i + 1, j);
                wander(grid, i - 1, j);
                wander(grid, i, j - 1);
            }

        }
    }
```

Problem 87

287 Find the Duplicate Number

```
Given an array nums containing n + 1 integers where each integer
 is between 1 and n (inclusive),
  prove that at least one duplicate number must exist.
   Assume that there is only one duplicate number, find the duplicate one.

Note:
You must not modify the array (assume the array is read only).
You must use only constant, O(1) extra space.
Your runtime complexity should be less than O(n^2).
There is only one duplicate number in the array,
 but it could be repeated more than once.

https://leetcode.com/problems/find-the-duplicate-number/
```

Solution 1

A pretty interesting problem.

There is a good blog which shows all kinds of solutions for this problem,

see `https://segmentfault.com/a/1190000003817671`.

Binary Search is available with its $O(nlogn) runtime and O(1) space$, with the idea to binary search for the value from 1 to n to match numbers in array, instead of searching in the array itself.

Listing 114: Solution 1

```java
public class Solution {
    public int findDuplicate(int[] nums) {
        int min = 1, max = nums.length - 1;
        while (min <= max){
            int mid = (min + max) / 2;

            int count = 0;
            int countN = 0;
            for (int num : nums){
                if (num <= mid){
                    count ++;

                    if (num == mid){
                        countN ++;
                    }
                }
            }

            if (countN >= 2){
                return mid;
            }

            if (count > mid){
                max = mid;
            }else{
                min = mid + 1;
```

Problem 87 continued on next page...

```
                }
            }

            return -1;
        }
}
```

Solution 2

An $O(n)runtime, O(1)space$ solution, fantastic train of thought!

See http://keithschwarz.com/interesting/code/?dir=find-duplicate

Main idea is like "Find Linked List Cycle II", use 'slow = array[slow], fast = array[array[fast]]' mappings to check if there is a cycle and get the entry. The entry is exactly what we are looking for!(Since there is a array[i] = array[j], where i != j).

Listing 115: Solution 2

```
public class Solution {
    public int findDuplicate(int[] nums) {
        int slow = 0;
        int fast = 0;

       do{
            slow = nums[slow];
            fast = nums[nums[fast]];
        }while (slow != fast);

        fast = 0;
        int prev = 0;

        while(slow != fast){
            prev = slow;
            slow = nums[slow];
            fast = nums[fast];
        }

        return slow;
        //return nums[prev];
        // Actually, we can return slow directly.
    }
}
```

Problem 88

268 Missing Number

```
Given an array containing n distinct numbers taken from
 0, 1, 2, ..., n, find the one that is missing from the array.

For example,
Given nums = [0, 1, 3] return 2.

Note:
Your algorithm should run in linear runtime complexity.
 Could you implement it using only constant extra space complexity?

https://leetcode.com/problems/missing-number/
```

Solution - Using Sum

Since there is only one missing element, the different between sum of the array and 0 to n is exactly what is missing.

Binary search similar to Problem "Find the Duplicate Number" is also proper for this problem.(Remember to sort the array first if we don't want to calculate the count.)

Listing 116: Solution 1

```
public class Solution {
    public int missingNumber(int[] nums) {
        int sumExp = 0;
        int sumAct = 0;
        int n = nums.length;
        for (int i = 0 ; i <= n ; i++){
            sumExp += i;
        }
        for (int num : nums){
            sumAct += num;
        }
        return sumExp - sumAct;
    }
}
```

Solution - Mark Position

Solution 2 is to set up a new array, with a length more than 1 element of former array. In the new array, the elements are the same of (index + 1). Every time we find a number in old array, we set element in corresponding position in new array -1. Finally we iterate the new array, the element which is not changed to -1 is what we want.

Some similar solutions including Using Hashset or sorted array to store the count of each number to see if there is a non-existing element.

Listing 117: Solution 2

```
public class Solution {
    public int missingNumber(int[] nums) {
        int[] nums2 = new int[nums.length + 1];
        for (int i = 0; i < nums2.length; i++) {
```

Problem 88 continued on next page...

```
            nums2[i] = i;
        }
        for (int i = 0; i < nums.length; i++) {
            nums2[nums[i]] = -1;
        }
        for (int i = 0; i < nums2.length; i++) {
            if (nums2[i] != -1) {
                return i;
            }
        }
        return -1;
    }
}
```

Solution - Bit Manipulation

This solution is interesting. See the steps first.

1. XOR all the array elements, let the result of XOR be X1.
2. XOR all numbers from 1 to n, let XOR be X2.
3. XOR of X1 and X2 gives the missing number.

Feel familiar? The insight of these steps is exactly like "Single Number", or to be more specific, the characteristic of XOR operator.

Listing 118: Solution 3

```
#include<stdio.h>
/* getMissingNo takes array and size of array as arguments*/
int getMissingNo(int a[], int n)
{
    int i;
    int x1 = a[0]; /* For xor of all the elements in array */
    int x2 = 1; /* For xor of all the elements from 1 to n+1 */
    for (i = 1; i< n; i++)
        x1 = x1^a[i];
    for ( i = 2; i <= n+1; i++)
        x2 = x2^i;

    return (x1^x2);
}
/*program to test above function */
int main()
{
    int a[] = {1, 2, 4, 5, 6};
    int miss = getMissingNo(a, 5);
    printf("%d", miss);
    getchar();
}
```

Problem 89

41 First Missing Positive

```
Given an unsorted integer array, find the first missing positive integer.

For example,
Given [1,2,0] return 3,
and [3,4,-1,1] return 2.

Your algorithm should run in O(n) time and uses constant space.

https://leetcode.com/problems/first-missing-positive/
```

Solution

This is somewhat like the idea of radix sort, to put each element to where it should be(arr[idx] = idx), then the 1st grid which doesn't satisfy this attribute is exactly the number we want.

Listing 119: Solution 1.1

```
public class Solution {
    public int firstMissingPositive(int[] nums) {
        for (int i = 0 ; i < nums.length ; i ++){
            nums[i] --;
        }
        int idx = 0;
        while (idx < nums.length){
            if (nums[idx] >= nums.length || nums[idx] < 0){
                idx ++;
            }
            else{
                if (nums[idx] == nums[nums[idx]] || nums[idx] == idx){
                    idx ++;
                }
                else{
                    int swap = nums[nums[idx]];
                    nums[nums[idx]] = nums[idx];
                    nums[idx] = swap;
                }
            }
        }

        for (idx = 0 ; idx < nums.length ; idx ++){
            if (nums[idx] != idx){
                return idx + 1;
            }
        }
        return nums.length + 1;
    }
}
```

Below is a solution without reducing 1 to all elements.

Listing 120: Solution 1.2

```
public class Solution {
    public int firstMissingPositive(int[] nums) {
        int idx = 0;
        while (idx < nums.length){
            int value = nums[idx] - 1;
            if (value >= nums.length || value < 0){
                idx ++;
            }
            else{
                if (value + 1 == nums[value] || nums[idx] == idx + 1){
                    idx ++;
                }
                else{
                    nums[idx] = nums[value];
                    nums[value] = value + 1;
                }
            }
        }

        for (idx = 0 ; idx < nums.length ; idx ++){
            if (nums[idx] != idx + 1){
                return idx + 1;
            }
        }
        return nums.length + 1;
    }
}
```

Problem 90

98 Validate Binary Search Tree

```
Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

    The left subtree of a node contains
     only nodes with keys less than the node's key.
    The right subtree of a node contains
     only nodes with keys greater than the node's key.
    Both the left and right subtrees must
     also be binary search trees.

https://leetcode.com/problems/validate-binary-search-tree/
```

Solution 1

Plenty of methods to solve the problem. First solution is to use recursion to solve the problem. However, code on "Clean Code Handbook" is extremely clean and charming!

Listing 121: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean validBST(TreeNode root, Integer min, Integer max){

        if (root == null){
            return true;
        }
        if (min != null){
            if (root.val <= min){
                return false;
            }
        }
        if (max != null){
            if (root.val >= max){
                return false;
            }
        }

        return validBST(root.left, min, root.val)
         && validBST(root.right, root.val, max);
    }

    public boolean isValidBST(TreeNode root) {
```

Problem 90 continued on next page...

```
        return validBST(root, null, null);

    }
}
```

Solution 2

A valid binary search tree has the attribute to keep ascending order when traversing the tree in an inorder order.

Listing 122: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void inorderTraverse(TreeNode root , List<Integer> traversal){
        if (root == null){
            return;
        }

        inorderTraverse(root.left, traversal);
        traversal.add(root.val);
        inorderTraverse(root.right, traversal);
    }

    public boolean isValidBST(TreeNode root) {
       List<Integer> traversal = new ArrayList<Integer>();
       inorderTraverse(root , traversal);
       for (int idx = 1 ; idx < traversal.size() ; idx ++){
           if (traversal.get(idx) <= traversal.get(idx - 1)){
               return false;
           }
       }

       return true;
    }
}
```

Problem 91

150 Evaluate Reverse Polish Notation

```
Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /.
 Each operand may be an integer or another expression.

Some examples:
  ["2", "1", "+", "3", "*"] -$>$ ((2 + 1) * 3) $->$ 9
  ["4", "13", "5", "/", "+"] -$>$ (4 + (13 / 5)) $->$ 6

https://leetcode.com/problems/evaluate-reverse-polish-notation/
```

Solution 1

Reverse Polish expression evaluation is a standard stack problem, normally use stack to solve this.

Listing 123: Solution 1

```
public class Solution {
    public boolean isOperator(String token){
        if (token.equals("+") || token.equals("-") || token.equals("*")
         || token.equals("/")){
            return true;
        }
        else{
            return false;
        }
    }

    public int calculate(int num1, int num2, String token){
        if (token.equals("+")){
            return num1 + num2;
        }
        if (token.equals("-")){
            return num1 - num2;
        }
        if (token.equals("*")){
            return num1 * num2;
        }
        if (token.equals("/")){
            return num1 / num2;
        }

        return -1;
    }

    public int evalRPN(String[] tokens) {
        Stack<String> stack = new Stack<String>();
        for (String token : tokens){
            if (isOperator(token)){
                int num1 = Integer.parseInt(stack.pop());
                int num2 = Integer.parseInt(stack.pop());
```

Problem 91 continued on next page...

```
                int result = calculate(num2, num1, token);

                stack.push(String.valueOf(result));
            }
            else{
                stack.push(token);
            }
        }

        return Integer.parseInt(stack.pop());

    }
}
```

Solution 2

However, when we want to add a new operator, the structure now is not so extensive since we should modify two parts of program. Use a Map to extend the program(It's a solution followed in "Clean Code Handbook"):

Listing 124: Solution 2

```
interface Operator{
    public int eval(int a , int b);
}

public class Solution {
    private static final HashMap<String, Operator> cal
     = new HashMap<String, Operator>(){
        {
            put("+" , new Operator(){
                public int eval(int a , int b){
                    return a + b;
                }
            });
            put("-" , new Operator(){
                public int eval(int a , int b){
                    return a - b;
                }
            });
            put("*" , new Operator(){
                public int eval(int a , int b){
                    return a * b;
                }
            });
            put("/" , new Operator(){
                public int eval(int a , int b){
                    return a / b;
                }
            });
        }
    };
```

```
    public int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack<Integer>();
        for (String token : tokens){
            if (cal.containsKey(token)){
                int num2 = stack.pop();
                int num1 = stack.pop();
                int result = cal.get(token).eval(num1, num2);

                stack.push(result);
            }
            else{
                stack.push(Integer.parseInt(token));
            }
        }

        return stack.pop();

    }
}
```

Problem 92

130 Surrounded Regions

```
Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:

X X X X
X X X X
X X X X
X O X X

https://leetcode.com/problems/surrounded-regions/
```

Solution

DFS will get StackOverflowException, so use BFS to solve the problem.
(Honestly, I don't know why my DFS solution StackOverflow?!)

Listing 125: Solution

```
class Pair{
    public Pair(int i , int j){
        this.i = i;
        this.j = j;
    }

    public int i, j;
}
public class Solution {
    public void escape(char[][] board, int i , int j){

        int m = board.length;
        int n = board[0].length;

        if (board[i][j] == 'O'){
            Queue<Pair> queue = new LinkedList<Pair>();

            queue.offer(new Pair(i, j));
            while(!queue.isEmpty()){
                Pair pair = queue.poll();
                if (board[pair.i][pair.j] != 'O'){
                    continue;
                }
                else{
                    board[pair.i][pair.j] = 'E';
                }
```

Problem 92 continued on next page...

```
                if (pair.i < m - 1 && board[pair.i + 1][pair.j] == 'O'){
                    queue.offer(new Pair(pair.i + 1 , pair.j));
                }if (pair.j < n - 1 && board[pair.i][pair.j + 1] == 'O'){
                    queue.offer(new Pair(pair.i , pair.j + 1));
                }if (pair.i > 0 && board[pair.i - 1][pair.j] == 'O'){
                    queue.offer(new Pair(pair.i - 1 , pair.j));
                }if (pair.j > 0 && board[pair.i][pair.j - 1] == 'O'){
                    queue.offer(new Pair(pair.i , pair.j - 1));
                }

            }
        }
    }

    public void solve(char[][] board) {
        int m = board.length;
        if (m == 0){
            return;
        }
        int n = board[0].length;
        for (int i = 0  ; i < m ; i ++){
            if (board[i][0] == 'O'){
                escape(board, i , 0);
            }
            if (board[i][n - 1] == 'O'){
                escape(board, i, n - 1);
            }
        }

        for (int i = 0 ; i < n ; i++){
            if (board[0][i] == 'O'){
                escape(board, 0 , i);
            }
            if (board[m - 1][i] == 'O'){
                escape(board, m - 1 , i);
            }
        }

        for (int i = 0 ; i < m ; i++){
            for (int j = 0 ; j < n ; j ++){
                if (board[i][j] == 'O'){
                    board[i][j] = 'X';
                }
                else if (board[i][j] == 'E'){
                    board[i][j] = 'O';
                }
            }
        }
    }
}
```

Problem 93

179 Largest Number

```
Given a list of non negative integers, arrange them such that
 they form the largest number.

For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330.

Note: The result may be very large, so you need to return a string
 instead of an integer.

https://leetcode.com/problems/largest-number/
```

Solution

Re-design the compare to judge two strings(A tactic is when we want to get bigger one of s1 and s2, check relation between s1s2 and s2s1, e.g. '93' > '39' so '9' > '3')

Don't forget the condition when result consists several 0s!

Listing 126: Solution

```java
public class Solution {
    public void quickSort(int[] nums , int low, int high){
        if (low < 0 || high > nums.length - 1){
            return ;
        }
        if (low > high){
            return;
        }
        int pivot = nums[low];
        int i = low;
        int j = high;

        while (i < j){
            while (i < j && noLessThan(nums[j], pivot)){
                j --;
            }
            nums[i] = nums[j];
            while (i < j && noLessThan(pivot , nums[i])){
                i++;
            }
            nums[j] = nums[i];
        }

        nums[i] = pivot;
        quickSort(nums, low, i - 1);
        quickSort(nums, i + 1, high);
    }

    public String largestNumber(int[] nums) {
        quickSort(nums, 0 , nums.length - 1);

        StringBuilder sb = new StringBuilder();
```

```
        int idx = nums.length - 1;
        while (idx >= 0 && nums[idx] == 0){
            idx--;
        }

        while(idx >= 0){
            sb.append(String.valueOf(nums[idx]));
            idx --;
        }


        return sb.toString().length() > 0 ? sb.toString() : "0";

    }

    public boolean noLessThan(int num1 , int num2){
        String com1 = String.valueOf(num1) + String.valueOf(num2);
        String com2 = String.valueOf(num2) + String.valueOf(num1);
        if (Long.parseLong(com1) >= Long.parseLong(com2)){
            return true;
        }

        return false;
    }

}
```

Problem 94

224 Basic Calculator

```
Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ),
 the plus + or minus sign -, non-negative integers and empty spaces .

You may assume that the given expression is always valid.

Some examples:
"1 + 1" = 2
" 2-1 + 2 " = 3
"(1+(4+5+2)-3)+(6+8)" = 23

https://leetcode.com/problems/basic-calculator/
```

Solution

A complicated problem.... String evaluation...

Reference: how to get Reverse Polish Expression :
http://blog.csdn.net/signsmile/article/details/2877729

Listing 127: Solution

```
public class Solution {
    Stack<String> operators = new Stack<String>();
    Stack<Integer> operands = new Stack<Integer>();
    public boolean isOperator(String op){
        return (op.equals("+") || op.equals("-")
         || op.equals("(") || op.equals(")"));
    }
    public int basicCal(String operator){
        int num2 = operands.pop();
        int num1 = operands.pop();
        if (operator.equals("+")){
            return num1 + num2;
        }
        if (operator.equals("-")){
            return num1 - num2;
        }
        return -1;
    }
    public int calculate(String s) {
        int idx = 0 ;
        String entity = "";
        while (idx < s.length()){
            char ch = s.charAt(idx);
            if (ch == ' '){
                idx ++;
                continue;
            }
            String numStr = "";
```

```
            while(ch != '+' && ch != '-' && ch != '(' && ch != ')' && ch != ' '
             && idx <= s.length() - 1){
                numStr += ch;
                idx ++;
                if (idx <= s.length() - 1){
                    ch = s.charAt(idx);
                }
            }
            if (numStr.length() > 0){
                operands.push(Integer.parseInt(numStr));
            }
            if (isOperator(String.valueOf(ch))){
                if (ch == '('){
                    operators.push(String.valueOf(ch));
                }else if (ch  == ')'){
                    while (!operators.isEmpty()
                     && !operators.peek().equals("(")){
                        String operator = operators.pop();
                        int result = basicCal(operator);
                        operands.push(result);
                    }
                    operators.pop();
                }else{
                    // + or -
                    while (!operators.isEmpty()
                     && (operators.peek().equals("+")
                     || operators.peek().equals("-"))){
                        String operator = operators.pop();
                        int result = basicCal(operator);
                        operands.push(result);
                    }
                    operators.push(String.valueOf(ch));
                }

            }

            idx ++;
        }
        while (!operators.isEmpty()){
            String operator = operators.pop();
            int result = basicCal(operator);
            operands.push(result);
        }
        return operands.pop();
    }
}
```

Problem 95

227 Basic Calculator II

```
Implement a basic calculator to evaluate a simple expression string.

The expression string contains only non-negative integers,
 +, -, *, / operators and empty spaces .
  The integer division should truncate toward zero.

You may assume that the given expression is always valid.

Some examples:
"3+2*2" = 7
" 3/2 " = 1
" 3+5 / 2 " = 5

Note: Do not use the eval built-in library function.

https://leetcode.com/problems/climbing-stairs/
```

Solution

A more complicated "Basic Calculator", calculate numbers in stack first whose priority is not larger than current operator to push.(e.g.: when current operator to push is *, calculate until the operator on peek of stack is +/-; if the current operator is +/-, calculate all elements in stack.

Listing 128: Solution

```
public class Solution {
    public int getResult(int num1 , int num2 , char op){
        if (op == '+'){
            return num1 + num2;
        } else if (op == '-'){
            return num1 - num2;
        } else if (op == '*'){
            return num1 * num2;
        } else if (op == '/'){
            return num1 / num2;
        }
        return -1;
    }
    public int calculate(String s) {
        Stack<Integer> operands = new Stack<Integer>();
        Stack<Character> operators = new Stack<Character>();
        int idx = 0;
        while (idx < s.length()){
            char curChar = s.charAt(idx);
            while (curChar == ' ' && ++idx < s.length()){
                curChar = s.charAt(idx);
            }
            String numberStr = "";
            while (idx < s.length() && curChar >= '0' && curChar <= '9'){
                numberStr  += curChar;
                idx ++;
```

```
                if (idx >= s.length()){
                    break;
                }
                curChar = s.charAt(idx);
            }
            if (numberStr.length() > 0){
                int number = Integer.parseInt(numberStr);
                operands.push(number);
            }
            if (curChar == ' ')
                continue;
            if (curChar == '+' || curChar == '-'
                 || curChar == '*' || curChar == '/'){
                if (operators.isEmpty()){
                    operators.push(curChar);
                } else {
                    // char recentOp = operators.peek();
                    if (curChar == '+' || curChar == '-'){
                        while(!operators.isEmpty()){
                             // calculate numbers in stack first
                            int num2 = operands.pop();
                            int num1 = operands.pop();
                            char op = operators.pop();

                            int result = getResult(num1, num2, op);
                            operands.push(result);
                        }
                        operators.push(curChar);
                    }
                    else if ((curChar == '*' || curChar == '/')){
                        while (!operators.isEmpty()
                         && operators.peek() != '+'
                          && operators.peek() != '-'){
                             // calculate numbers in stack first
                            int num2 = operands.pop();
                            int num1 = operands.pop();
                            char op = operators.pop();
                            int result = getResult(num1, num2, op);
                            operands.push(result);
                        }
                        operators.push(curChar);
                    }
                }
                idx ++;
            }
        }
        while (!operators.isEmpty()){
            int num2 = operands.pop();
            int num1 = operands.pop();
            char op = operators.pop();
            int result = getResult(num1, num2, op);
            operands.push(result);
        }
```

Problem 95 continued on next page...

```
80              return operands.pop();
        }
}
```

Problem 96

25 Reverse Nodes in k-Group

```
Given a linked list, reverse the nodes of a linked list k
 at a time and return its modified list.

If the number of nodes is not a multiple of k then left-out nodes
 in the end should remain as it is.

You may not alter the values in the nodes, only nodes
 itself may be changed.

Only constant memory is allowed.

For example,
Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

https://leetcode.com/problems/reverse-nodes-in-k-group/
```

Solution

A bad news is that I debugged it in my Eclipse IDE.....

Listing 129: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverse(ListNode start, ListNode end){
        if (start == null){
            return null;
        }
        // tail = start;
        ListNode cur = start;
        ListNode next = cur.next;
        cur.next = null;
        while (next != null && next != end){
            ListNode backup = next.next;
            next.next = cur;
            cur = next;
            next = backup;
        }
        return cur;
    }
```

Problem 96 continued on next page...

```
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode cur = head;
        ListNode last = cur;
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode tail = dummy;
        while (cur != null){
            int count = k;
            last = cur;
            while (count-- > 0){
                cur = cur.next;
                if (cur == null){
                    break;
                }
            }
            if (count > 0 && cur == null){
                // end up ahead of time
                tail.next = last;
                return dummy.next;
            }
            ListNode newPart = reverse(last, cur);
            tail.next = newPart;
            tail = last;
        }

        return dummy.next;
    }
}
```

Problem 97

70 Climbing Stairs

```
You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps.
 In how many distinct ways can you climb to the top?

https://leetcode.com/problems/climbing-stairs/
```

Solution

A simple dp problem, so similar to Fibonacci problem.

Listing 130: Solution

```
public class Solution {
    public int climbStairs(int n) {
        if (n == 0 || n == 1){
            return 1;
        }
        int cs[] = new int[n + 1];
        cs[0] = 1;
        cs[1] = 1;
        for (int idx = 2; idx <= n ; idx ++){
            cs[idx] = cs[idx - 1] + cs[idx - 2];
        }

        return cs[n];
    }
}
```

Problem 98

3 Longest Substring Without Repeating Characters

```
Given a string, find the length of the longest substring
 without repeating characters.
  For example, the longest substring without repeating letters
   for "abcabcbb" is "abc", which the length is 3.
    For "bbbbb" the longest substring is "b",
     with the length of 1.

https://leetcode.com/problems/longest-substring-without-repeating-characters/
```

Solution

Brute force will have $O(n^2)runtime$ complexity, which will get TLE.

So using a clever way of thought. Maintain 2 pointers, one point to the start of sequence while another point to end of sequence. A array of boolean is used to check if current character has exists in previous sequence. If current character doesn't exists, well good news, we can continue iterating and check if we've got the longest subsequence; if it exists, iterate start pivot to recover the states of those characters, make them as they have not been visited, until meet the character same as current character, rearrange the position of pointer, continue iterating.

Don't forget to check if the sequence at end of whole String is the longest.

Listing 131: Solution

```
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s.length() == 0 || s.length() == 1){
            return s.length();
        }
        int start = 0 , end = 0;
        int maxLength = 0;
        boolean exists[] = new boolean[256];
        Arrays.fill(exists, false);

        while (end < s.length()){
            // iterate to find duplicated element
            maxLength = Math.max(maxLength, end - start);
            if (start != end){
                if (!exists[s.charAt(end)]){
                    exists[s.charAt(end)] = true;
                    end ++;
                }
                else{
                    while (s.charAt(start) != s.charAt(end)){
                        exists[s.charAt(start)] = false;
                        start ++;
                    }
                    start ++;
                    end ++;
                }
            }else{
                exists[s.charAt(start)] = true;
```

```
                    end ++;
                }
            }

            maxLength = Math.max(maxLength, s.length() - start);

            return maxLength;
        }
    }
```

Problem 99

96 Unique Binary Search Trees

```
Given n, how many structurally unique BST's (binary search trees)
 that store values 1...n?

For example,
Given n = 3, there are a total of 5 unique BST's.

   1          3     3      2      1
    \        /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3

https://leetcode.com/problems/unique-binary-search-trees/
```

Solution

A picture will be good for our understanding of this problem.

思路

这道题目我想了两天，一直试图找出递推的规律，昨晚才想明白思路有问题，这应该是一道典型的动态规划题目，因为有重叠的子问题，当前决策依赖于子问题的解

我设dp[i]表示共有i个节点时，能产生的BST树的个数

n == 0 时，空树的个数必然为1，因此dp[0] = 1

n == 1 时，只有1这个根节点，数量也为1，因此dp[1] = 1

n == 2时，有两种构造方法，如下图所示：

暂无图片

因此，dp[2] = dp[0] * dp[1] + dp[1] * dp[0]

n == 3时，构造方法如题目给的示例所示，dp[3] = dp[0] * dp[2] + dp[1] * dp[1] + dp[2] * dp[0]

同时，当根节点元素为 1, 2, 3, 4, 5, ..., i, ..., n时，基于以下原则的BST树具有唯一性：

以i为根节点时，其左子树构成为[0,...,i-1],其右子树构成为[i+1,...,n]构成

因此，dp[i] = sigma(dp[0...k] * dp[k+1...i]) 0 <= k < i - 1

Listing 132: Solution

```java
public class Solution {
    public int numTrees(int n) {
        if(n == 0 || n == 1){
            return 1;
        }
        int possibilities[] = new int[n + 1];
        possibilities[0] = 1;
        possibilities[1] = 1;

        for (int rootVal = 2 ; rootVal <= n ; rootVal ++){
```

```
            int sum = 0;
            for (int left = 0 ; left <= rootVal - 1 ; left++){
                int right = rootVal - 1 - left;
                sum += possibilities[left] * possibilities[right];
            }
            possibilities[rootVal] = sum;
        }

        return possibilities[n];
    }
}
```

Problem 100

95 Unique Binary Search Trees II

```
Given n, generate all structurally unique BST's (binary search trees)
 that store values 1...n.

For example,
Given n = 3, your program should return all 5 unique BST's shown below.

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3

https://leetcode.com/problems/unique-binary-search-trees-ii/
```

Solution

This is an interesting recursion problem since we can return a set of solutions to iteratively add them to left side and right side onto root node.

There are many other problems which set left and right children recursively.

Listing 133: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<TreeNode> generateTree(int start, int end){
        List<TreeNode> uniqueTrees = new ArrayList<TreeNode>();
        if (start > end){
            uniqueTrees.add(null);
        }
        for (int rootVal = start ; rootVal <= end ; rootVal ++){
            List<TreeNode> leftTrees = generateTree(start, rootVal - 1);
            List<TreeNode> rightTrees = generateTree(rootVal + 1 , end);
            for (TreeNode leftNode : leftTrees){
                for (TreeNode rightNode : rightTrees){
                    TreeNode rootNode = new TreeNode(rootVal);
                    rootNode.left = leftNode;
                    rootNode.right = rightNode;

                    uniqueTrees.add(rootNode);
                }
            }
        }
```

Problem 100 continued on next page...

```
        return uniqueTrees;
    }
    public List<TreeNode> generateTrees(int n) {

        if (n == 0){
            return new ArrayList<TreeNode>();
        }
        return generateTree(1 , n);
    }
}
```

Problem 101

241 Different Ways to Add Parentheses

```
Given a string of numbers and operators, return all possible results
 from computing all the different possible ways to
  group numbers and operators.
   The valid operators are +, - and *.


Example 1
Input: "2-1-1".

((2-1)-1) = 0
(2-(1-1)) = 2
Output: [0, 2]


Example 2
Input: "2*3-4*5"

(2*(3-(4*5))) = -34
((2*3)-(4*5)) = -14
((2*(3-4))*5) = -10
(2*((3-4)*5)) = -10
(((2*3)-4)*5) = 10
Output: [-34, -14, -10, -10, 10]

https://leetcode.com/problems/different-ways-to-add-parentheses/
```

Solution

The train of thought is similar to Problem "Unique Binary Search Tree II", get all possibilities, and combine them all.

Listing 134: Solution

```
public class Solution {
    public List<Integer> diffWaysToCompute(String input) {
        List<Integer> result = new ArrayList<Integer>();
        for (int idx = 0 ; idx < input.length() ; idx ++){
            if (input.charAt(idx) == '+' || input.charAt(idx) == '-'
             || input.charAt(idx) == '*'){
                List<Integer> leftResults =
                    diffWaysToCompute(input.substring(0 , idx));
                List<Integer> rightResults =
                    diffWaysToCompute(input.substring(idx + 1 , input.length()));

                for (int leftResult : leftResults){
                    for (int rightResult : rightResults){
                        switch(input.charAt(idx)){
                            case '+':
                                result.add(leftResult + rightResult);
                                break;
                            case '-':
```

```
                                result.add(leftResult - rightResult);
                                break;
                            default:
                                result.add(leftResult * rightResult);
                        }
                    }
                }
            }
        }
        if (result.size() == 0){
            result.add(Integer.parseInt(input));
        }
        return result;
    }
}
```

Problem 102

129 Sum Root to Leaf Numbers

```
Given a binary tree containing digits from 0-9 only,
 each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

For example,

    1
   / \
  2   3
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.

Return the sum = 12 + 13 = 25.

https://leetcode.com/problems/sum-root-to-leaf-numbers/
```

Solution

A DFS question, mention the edge cases.

Listing 135: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    int sum = 0;
    public void traverseSum(TreeNode curNode , int curSum){
        if (curNode == null){
            return;
        }
        curSum  = curSum * 10 + curNode.val;
        if (curNode.left == null && curNode.right == null){
            sum += curSum;

            return;
        }

        traverseSum(curNode.left , curSum);
        traverseSum(curNode.right  , curSum);
    }
```

```
    public int sumNumbers(TreeNode root) {
        traverseSum(root, 0);
        return sum;
    }
}
```

Problem 103

173 Binary Search Tree Iterator

```
Implement an iterator over a binary search tree (BST).
 Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

Note: next() and hasNext() should run in average
 O(1) time and uses O(h) memory, where h is the height of the tree.

https://leetcode.com/problems/binary-search-tree-iterator/
```

Solution

BST is really a tactical data structure. Iterator asks us to return the next smallest element, namely return a ascending list if we call the next() function several times. Recall something? The inorder traversal of BST! Then what we only need to do is to repeat what we have done in process of inorder traversal.

Listing 136: Solution

```
/**
 * Definition for binary tree
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class BSTIterator {
    Stack<TreeNode> stack = new Stack<TreeNode>();

    public BSTIterator(TreeNode root) {
        if (root != null){
            stack.push(root);

            while (root != null && root.left != null){
                stack.push(root.left);
                root = root.left;
            }
        }
    }
    /** @return whether we have a next smallest number */
    public boolean hasNext() {
        return (!stack.isEmpty());
    }
    /** @return the next smallest number */
    public int next() {
        TreeNode nextNode = stack.pop();

        TreeNode rightChild = nextNode.right;
        while (rightChild != null){
            stack.push(rightChild);
```

Problem 103 continued on next page...

```
            rightChild = rightChild.left;
        }
        return nextNode.val;
    }
}
/**
 * Your BSTIterator will be called like this:
 * BSTIterator i = new BSTIterator(root);
 * while (i.hasNext()) v[f()] = i.next();
 */
```

Problem 104

297 Serialize and Deserialize Binary Tree

```
Serialization is the process of converting a data structure or object into
 a sequence of bits so that it can be stored in a file or memory buffer,
  or transmitted across a network connection link to be reconstructed later
   in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree.
 There is no restriction on how your serialization/deserialization algorithm
  should work. You just need to ensure that a binary tree can be serialized to
   a string and this string can be deserialized to the original tree structure.

For example, you may serialize the following tree

    1
   / \
  2   3
     / \
    4   5
as "[1,2,3,null,null,4,5]", just the same as how LeetCode OJ
 serializes a binary tree. You do not necessarily need to follow this format,
  so please be creative and come up with different approaches yourself.
Note: Do not use class member/global/static variables to store states.
 Your serialize and deserialize algorithms should be stateless.

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/
```

Solution

Honestly, my solution is not perfect at all(or to say not good at all) since I serialize all elements including null, which is different from LeetCode OJ.

Listing 137: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Codec {

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        if (root == null){
            return "[]";
        }
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(root);
        ArrayList<Integer> valueList = new ArrayList<Integer>();
```

```
20
        while (!queue.isEmpty()){
            TreeNode curNode = queue.poll();
            if (curNode != null){
                valueList.add(curNode.val);
25          } else{
                valueList.add(null);
                continue;
            }

30          // if (queue.isEmpty()){
            //     // push non-null node
            //     if (curNode.left != null && curNode.right != null){
            //         queue.offer(curNode.left);
            //         queue.offer(curNode.right);
35          //     }else if (curNode.left != null){
            //         queue.offer(curNode.left);
            //     }else if (curNode.right != null){
            //         queue.offer(curNode.right);
            //     }
40          // }else{
                // push all children, including null node
                queue.offer(curNode.left);
                queue.offer(curNode.right);
            // }
45      }

        StringBuilder resultBuilder = new StringBuilder();
        resultBuilder.append("[");
        for (int valueIdx = 0 ; valueIdx < valueList.size() - 1 ; valueIdx ++){
50          if (valueList.get(valueIdx) == null){
                resultBuilder.append("#");
            } else{
                resultBuilder.append(String.valueOf(valueList.get(valueIdx)));
            }
55          resultBuilder.append(",");
        }

        if (valueList.get(valueList.size() - 1) == null){
            resultBuilder.append("#");
60      }else{
            resultBuilder.append(
                String.valueOf(valueList.get(valueList.size() - 1)));
        }
        resultBuilder.append("]");
65
        return resultBuilder.toString();
    }

    // Decodes your encoded data to tree.
70  public TreeNode deserialize(String data) {
        if (data.equals("[]")){
            return null;
```

```
        }

        data = data.substring(1, data.length() - 1);

        Queue<TreeNode> resolvingQueue = new LinkedList<TreeNode>();
        Queue<TreeNode> waitingQueue = new LinkedList<TreeNode>();

        String[] values = data.split(",");
        for (int idxValue = 0 ; idxValue < values.length ; idxValue ++){
            if (values[idxValue].equals("#")){
                waitingQueue.offer(null);
            }else{
                waitingQueue.offer(
                    new TreeNode(Integer.parseInt(values[idxValue])));
            }
        }
        TreeNode rootNode = waitingQueue.poll();
        resolvingQueue.offer(rootNode);
        while (!resolvingQueue.isEmpty() && !waitingQueue.isEmpty()){
            TreeNode resolvingNode = resolvingQueue.poll();
            if (resolvingNode != null) {
                TreeNode leftChild = waitingQueue.poll();
                TreeNode rightChild = waitingQueue.poll();


                resolvingNode.left = leftChild;
                resolvingNode.right = rightChild;

                resolvingQueue.offer(leftChild);
                resolvingQueue.offer(rightChild);
            }
        }

        return rootNode;
    }
}

// Your Codec object will be instantiated and called as such:
// Codec codec = new Codec();
// codec.deserialize(codec.serialize(root));
```

Problem 105

284 Peeking Iterator

```
Given an Iterator class interface with methods: next() and hasNext(),
 design and implement a PeekingIterator that support the peek() operation
  -- it essentially peek() at the element
   that will be returned by the next call to next().

Here is an example. Assume that the iterator is initialized
 to the beginning of the list: [1, 2, 3].

Call next() gets you 1, the first element in the list.

Now you call peek() and it returns 2, the next element.
 Calling next() after that still return 2.

You call next() the final time and it returns 3, the last element.
 Calling hasNext() after that should return false.

Hint:

Think of "looking ahead". You want to cache the next element.
Is one variable sufficient? Why or why not?
Test your design with call order of peek() before next() vs next() before peek().
For a clean implementation, check out Google's guava library source code.
Follow up: How would you extend your design to be generic and work with all types,
 not just integer?

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/
```

Solution

My solution is copied from guava source code in some extent, so I will list the standard solution here in Solution 1.2.

Listing 138: Solution 1.1

```
// Java Iterator interface reference:
// https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> iterator;
    private boolean hasPeeked;
    private Integer peekedElement;

    public PeekingIterator(Iterator<Integer> iterator) {
        // initialize any member here.
        this.iterator = iterator;
    }

    // Returns the next element in the iteration without advancing the iterator.
    public Integer peek() {
        if (!hasPeeked){
            if (iterator.hasNext()){
                hasPeeked = true;
```

```
                peekedElement = iterator.next();
            }
            else{
                // throw new Exception("Nothing to peek");
                peekedElement = null;
            }
        }

        return peekedElement;
    }

    // hasNext() and next() should behave the same as in the Iterator interface.
    // Override them if needed.
    @Override
    public Integer next() {
        if (hasPeeked){
            hasPeeked = false;
            return peekedElement;
        }
        else{
            return iterator.next();
        }
    }

    @Override
    public boolean hasNext() {
        return hasPeeked || iterator.hasNext();
    }
}
```

Listing 139: Solution 1.2

```
/**
   * Implementation of PeekingIterator that avoids peeking unless necessary.
   */
  private static class PeekingImpl<E> implements PeekingIterator<E> {

    private final Iterator<? extends E> iterator;
    private boolean hasPeeked;
    private E peekedElement;

    public PeekingImpl(Iterator<? extends E> iterator) {
      this.iterator = checkNotNull(iterator);
    }

    @Override
    public boolean hasNext() {
      return hasPeeked || iterator.hasNext();
    }

    @Override
    public E next() {
      if (!hasPeeked) {
        return iterator.next();
```

```
      }
      E result = peekedElement;
      hasPeeked = false;
      peekedElement = null;
      return result;
    }

    @Override
    public void remove() {
      checkState(!hasPeeked, "Can't remove after you've peeked at next");
      iterator.remove();
    }

    @Override
    public E peek() {
      if (!hasPeeked) {
        peekedElement = iterator.next();
        hasPeeked = true;
      }
      return peekedElement;
    }
  }
```

Problem 106

226 Invert Binary Tree

```
Invert a binary tree.

     4
   /   \
  2     7
 / \   / \
1   3 6   9
to
     4
   /   \
  7     2
 / \   / \
9   6 3   1
Trivia:
This problem was inspired by this original tweet by Max Howell:
Google: 90\% of our engineers use the software you wrote (Homebrew),
 but you can't invert a binary tree on a whiteboard so fuck off.

https://leetcode.com/problems/invert-binary-tree/
```

Solution 1

A simple problem. Solution 1 is recursive solution.

Listing 140: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode invertTree(TreeNode root) {
        if (root == null){
            return null;
        }
        TreeNode swap = root.left;
        root.left = root.right;
        root.right = swap;

        invertTree(root.left);
        invertTree(root.right);
        return root;
    }
}
```

Problem 106 continued on next page...

Solution 2

Solution 2 is an iterative solution.

Listing 141: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode invertTree(TreeNode root) {
        if (root == null){
            return null;
        }
        Stack<TreeNode> stack = new Stack<TreeNode>();

        stack.push(root);

        while (!stack.isEmpty()){
            TreeNode curNode = stack.pop();
            TreeNode swap = curNode.left;
            curNode.left = curNode.right;
            curNode.right = swap;

            if (curNode.left != null){
                stack.push(curNode.left);
            }
            if (curNode.right != null){
                stack.push(curNode.right);
            }
        }

        return root;
    }
}
```

Problem 107

257 Binary Tree Paths

```
Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:

   1
 /   \
2     3
 \
  5
All root-to-leaf paths are:

["1->2->5", "1->3"]

https://leetcode.com/problems/binary-tree-paths/
```

Solution

A recursive problem.

Listing 142: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<String> paths = new ArrayList<String>();
    public void traversePath(TreeNode curNode, String prefix){
        if (curNode == null)
            return;
        if (curNode.left == null && curNode.right == null)
            paths.add(prefix + curNode.val);
        prefix += curNode.val + "->";
        if (curNode.left != null)
            traversePath(curNode.left , prefix);
        if (curNode.right != null)
            traversePath(curNode.right, prefix);
    }
    public List<String> binaryTreePaths(TreeNode root) {
        traversePath(root, "");
        return paths;
    }
}
```

Problem 108

144 Binary Tree Preorder Traversal

```
Given a binary tree, return the preorder traversal of its nodes' values.

For example:
Given binary tree {1,#,2,3},
   1
    \
     2
    /
   3
return [1,2,3].

Note: Recursive solution is trivial, could you do it iteratively?

https://leetcode.com/problems/binary-tree-preorder-traversal/
```

Solution 1

First come up with the recursive solution.

Listing 143: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void preorder(TreeNode curNode, List<Integer> result){
        if (curNode != null){
            result.add(curNode.val);
            preorder(curNode.left, result);
            preorder(curNode.right , result);
        }
    }

    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        preorder(root, result);

        return result;
    }
}
```

Solution 2

In description, we are asked to finish this task in an iterative way, so we should consider it carefully. Preorder traversal is the easiest traversal in terms of coding. Using a stack to store nodes visited, but remember to push right child first so that in stack they will be popped after left child.

Listing 144: Solution 2

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        if (root == null){
            return new ArrayList<Integer>();
        }
        List<Integer> result = new ArrayList<Integer>();
        Stack<TreeNode> traversalStack = new Stack<TreeNode>();
        traversalStack.push(root);
        while (!traversalStack.isEmpty()){
            TreeNode curNode = traversalStack.pop();

            result.add(curNode.val);

            if (curNode.right != null){
                traversalStack.push(curNode.right);
            }
            if (curNode.left != null){
                traversalStack.push(curNode.left);
            }
        }
        return result;
    }
}
```

Problem 109

94 Binary Tree Inorder Traversal

```
Given a binary tree, return the inorder traversal of its nodes' values.

For example:
Given binary tree {1,#,2,3},
   1
    \
     2
    /
   3
return [1,3,2].

Note: Recursive solution is trivial, could you do it iteratively?

https://leetcode.com/problems/binary-tree-inorder-traversal/
```

Solution 1

First come up with the recursive solution.

Listing 145: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void inorder(TreeNode curNode, List<Integer> result){
        if(curNode != null){
            inorder(curNode.left, result);
            result.add(curNode.val);
            inorder(curNode.right, result);
        }
    }

    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        inorder(root, result);

        return result;
    }
}
```

Solution 2

For iterative solution, we use a stack to maintain the order of nodes. This time a bit different- we first push all left nodes of root node iteratively, and when we pop out a TreeNode, after adding the value into result list, we should push right child of current node and all its left children into stack.

Listing 146: Solution 2

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        if (root == null){
            return new ArrayList<Integer>();
        }
        List<Integer> result = new ArrayList<Integer>();
        Stack<TreeNode> traversalStack = new Stack<TreeNode>();

        TreeNode curNode = root;
        while (curNode != null){
            traversalStack.push(curNode);
            curNode = curNode.left;
        }
        while (!traversalStack.isEmpty()){
            curNode = traversalStack.pop();
            result.add(curNode.val);

            TreeNode addNode = curNode.right;

            while (addNode != null){
                traversalStack.push(addNode);
                addNode = addNode.left;
            }
        }

        return result;
    }
}
```

Problem 110

145 Binary Tree Postorder Traversal

```
Given a binary tree, return the postorder traversal of its nodes' values.

For example:
Given binary tree {1,#,2,3},
   1
    \
     2
    /
   3
return [3,2,1].

Note: Recursive solution is trivial, could you do it iteratively?

https://leetcode.com/problems/binary-tree-postorder-traversal/
```

Solution 1

First come up with the recursive solution.

Listing 147: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void postorderRecursion(TreeNode curNode, List<Integer> result){
        if (curNode != null){
            postorderRecursion(curNode.left, result);
            postorderRecursion(curNode.right, result);
            result.add(curNode.val);
        }
    }
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();

        postorderRecursion(root, result);

        return result;
    }
}
```

Solution 2

Postorder traversal is more complicated than preorder and inorder. Its complexity lays in the fact that when we first pop the node, we should not print it immediately but should wait for all its children to be

resolved, so we shall set a tag to check if the node is visited, if not, push its children into stack and mark the parent node as visited. So when the next time we meet, we can directly put it into our result list.

Listing 148: Solution 2.1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class TagTreeNode{
    TreeNode node;
    boolean visited;
    public TagTreeNode(TreeNode node, boolean visited){
        this.node = node;
        this.visited = visited;
    }
}
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        if (root == null){
            return new ArrayList<Integer>();
        }
        List<Integer> result = new ArrayList<Integer>();
        Stack<TagTreeNode> traversalStack = new Stack<TagTreeNode>();
        traversalStack.push(new TagTreeNode(root, false));
        while (!traversalStack.isEmpty()){
            TagTreeNode curNode = traversalStack.pop();
            if(curNode.visited){
                result.add(curNode.node.val);
            }
            else{
                curNode.visited = true;
                traversalStack.push(curNode);
                if (curNode.node.right != null){
                    traversalStack.push(
                        new TagTreeNode(curNode.node.right,false));
                }
                if (curNode.node.left != null){
                    traversalStack.push(
                        new TagTreeNode(curNode.node.left,false));
                }
            }
        }
        return result;
    }
}
```

Another interesting solution is to reverse the result since postorder leads to (left, right, root), when we reverse it, it appears to be (root, right, left), which is so similar to preorder traversal.

Listing 149: Solution 2.2

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        if (root == null){
            return new ArrayList<Integer>();
        }
        List<Integer> result = new ArrayList<Integer>();
        Stack<TreeNode> traversalStack = new Stack<TreeNode>();
        traversalStack.push(root);
        while (!traversalStack.isEmpty()){
            TreeNode curNode = traversalStack.pop();
            result.add(curNode.val);
            if (curNode.left != null){
                traversalStack.push(curNode.left);
            }
            if (curNode.right != null){
                traversalStack.push(curNode.right);
            }
        }
        return reverseList(result);
    }
    private List<Integer> reverseList(List<Integer> list){
        int idxStart = 0 , idxEnd = list.size() - 1;
        while (idxStart < idxEnd){
            Integer swapInt = list.get(idxStart);
            list.set(idxStart, list.get(idxEnd));
            list.set(idxEnd, swapInt);
            idxStart ++;
            idxEnd --;
        }
        return list;
    }
}
```

Problem 111

199 Binary Tree Right Side View

```
Given a binary tree, imagine yourself standing on the right side of it,
 return the values of the nodes you can see ordered from top to bottom.

For example:
Given the following binary tree,
   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
You should return [1, 3, 4].

https://leetcode.com/problems/binary-tree-postorder-traversal/
```

Solution

BFS problem, add value of rightmost node for each level to get the right side view.

Listing 150: Solution

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        if (root == null){
            return new ArrayList<Integer>();
        }
        Queue<TreeNode> resolvingNodes = new LinkedList<TreeNode>();
        resolvingNodes.offer(root);
        List<Integer> rightMostNodes = new ArrayList<Integer>();
        while(!resolvingNodes.isEmpty()){
            List<TreeNode> nodeListInCurLevel = new ArrayList<TreeNode>();
            while(!resolvingNodes.isEmpty()){
                nodeListInCurLevel.add(resolvingNodes.poll());
            }
            rightMostNodes.add(
                nodeListInCurLevel.get(nodeListInCurLevel.size() - 1).val);
            for (TreeNode curNode : nodeListInCurLevel){
                if (curNode.left != null){
                    resolvingNodes.offer(curNode.left);
                }
                if (curNode.right != null){
                    resolvingNodes.offer(curNode.right);
                }
```

```
                }
            }
            return rightMostNodes;
        }
}
```

Problem 112

116 Populating Next Right Pointers in Each Node

```
Given a binary tree

struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
Populate each next pointer to point to its next right node.
 If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

Note:

You may only use constant extra space.
You may assume that it is a perfect binary tree
 (ie, all leaves are at the same level, and every parent has two children).
For example,
Given the following perfect binary tree,
         1
       /  \
      2    3
     / \  / \
    4  5  6  7
After calling your function, the tree should look like:
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL

https://leetcode.com/problems/populating-next-right-pointers-in-each-node/
```

Solution

Similar to Problem "Binary Tree Right Side View", use BFS.

Listing 151: Solution

```
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void connect(TreeLinkNode root) {
        if (root == null){
            return;
```

```
        }
        Queue<TreeLinkNode> resolvingNodes = new LinkedList<TreeLinkNode>();
        resolvingNodes.offer(root);
        while (!resolvingNodes.isEmpty()){
            List<TreeLinkNode> nodesCurLevel = new ArrayList<TreeLinkNode>();
            while (!resolvingNodes.isEmpty()){
                nodesCurLevel.add(resolvingNodes.poll());
            }
            for (int nodeIdx = 0 ; nodeIdx < nodesCurLevel.size() ; nodeIdx ++){
                if (nodeIdx != nodesCurLevel.size() - 1){
                    nodesCurLevel.get(nodeIdx).next =
                     nodesCurLevel.get(nodeIdx + 1);
                }
                else{
                    nodesCurLevel.get(nodeIdx).next = null;
                }
                if (nodesCurLevel.get(nodeIdx).left != null){
                    resolvingNodes.offer(nodesCurLevel.get(nodeIdx).left);
                }
                if (nodesCurLevel.get(nodeIdx).right != null){
                    resolvingNodes.offer(nodesCurLevel.get(nodeIdx).right);
                }
            }
        }
    }
}
```

Problem 113

117 Populating Next Right Pointers in Each Node II

```
Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree?
 Would your previous solution still work?

Note:

You may only use constant extra space.
For example,
Given the following binary tree,
         1
       /  \
      2    3
     / \    \
    4   5    7
After calling your function, the tree should look like:
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \    \
    4-> 5 -> 7 -> NULL

https://leetcode.com/problems/populating-next-right-pointers-in-each-node-ii/
```

Solution

$O(1)$*space* is a big challenge. If the tree is a perfect binary tree, an interesting way is to connect left child and right child of each node, and when current node has a next object not pointing to null, connect right child of current node to left child of next node of current node.

However, it is said that the tree is just a common binary tree, things seems a bit more complicated, we shall consider all kinds of circumstances and edge cases, following former train of thought.

Listing 152: Solution

```
/**
 * Definition for binary tree with next pointer.
 * public class TreeLinkNode {
 *     int val;
 *     TreeLinkNode left, right, next;
 *     TreeLinkNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void connect(TreeLinkNode root) {
        TreeLinkNode currentLevelHead = root, nextLevelHead = null;

        while (currentLevelHead != null){
            TreeLinkNode currentNode = currentLevelHead;
            TreeLinkNode prevTail = null;
```

Problem 113 continued on next page...

```
            while (currentNode != null){
                if (currentNode.left != null && currentNode.right != null){
                    currentNode.left.next = currentNode.right;
                }
                if (currentNode.left == null && currentNode.right == null){
                    currentNode = currentNode.next;
                    continue;
                }
                else{
                    TreeLinkNode firstNodeInRow = (currentNode.left == null)
                     ? currentNode.right : currentNode.left;
                    if (nextLevelHead == null){
                        nextLevelHead = firstNodeInRow;
                        prevTail = firstNodeInRow;
                    }
                    else{
                        prevTail.next = firstNodeInRow;
                    }

                    while (prevTail.next != null){
                        prevTail = prevTail.next;
                    }
                }

                currentNode = currentNode.next;
            }
            currentLevelHead = nextLevelHead;
            nextLevelHead = null;
        }
    }
}
```

Problem 114

105 Construct Binary Tree from Preorder and Inorder Traversal

```
Given preorder and inorder traversal of a tree, construct the binary tree.

https://leetcode.com/problems/-
        construct-binary-tree-from-preorder-and-inorder-traversal/
```

Solution

How to split the two arrays is a important stuff to consider, recursion is a natural way to achieve this.

Listing 153: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode buildTree(int[] preorder, int startPre, int endPre,
     int[] inorder, int startIn, int endIn){
        if (startPre > endPre || startIn > endIn){
            return null;
        }
        int split = preorder[startPre];

        TreeNode rootNode = new TreeNode(split);
        if (startPre == endPre){
            return rootNode;
        }
        int idx = startIn;
        for (idx = startIn ; idx <= endIn ; idx ++){
            if (inorder[idx] == split){
                break;
            }
        }
        rootNode.left = buildTree(preorder, startPre + 1,
         startPre + idx - startIn , inorder, startIn , idx - 1);
        rootNode.right = buildTree(preorder ,
         startPre + idx - startIn + 1 ,endPre, inorder, idx + 1, endIn);

        return rootNode;
    }

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        return buildTree(preorder, 0, preorder.length - 1,
         inorder , 0 , inorder.length - 1);
    }
}
```

Problem 114 continued on next page...

Problem 115

106 Construct Binary Tree from Inorder and Postorder Traversal

```
Given inorder and postorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

https://leetcode.com/problems/-
        construct-binary-tree-from-inorder-and-postorder-traversal/
```

Solution

Similar to Problem 105 "Construct Binary Tree from Preorder and Inorder Traversal".

Listing 154: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode buildTree(int[] inorder, int startInorder, int endInorder,
     int[] postorder, int startPost, int endPost){
        if (startInorder > endInorder){
            return null;
        }

        int split = postorder[endPost];

        TreeNode rootNode = new TreeNode(split);
        if (startInorder == endInorder){
            return rootNode;
        }

        int splitIdx = -1;
        for (int idx = startInorder; idx <= endInorder; idx ++){
            if (inorder[idx] == split){
                splitIdx = idx;
                break;
            }
        }

        rootNode.left = buildTree(inorder , startInorder, splitIdx - 1 ,
         postorder , startPost , startPost + splitIdx - 1 - startInorder);
        rootNode.right = buildTree(inorder, splitIdx + 1 , endInorder ,
         postorder , startPost + splitIdx - startInorder , endPost - 1);

        return rootNode;
```

Problem 115 continued on next page...

```
    }

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        return buildTree(inorder, 0, inorder.length - 1,
         postorder , 0 , postorder.length - 1);
    }
}
```

Problem 116

242 Valid Anagram

```
Given two strings s and t, write a function to determine
 if t is an anagram of s.

For example,
s = "anagram", t = "nagaram", return true.
s = "rat", t = "car", return false.

Note:
You may assume the string contains only lowercase alphabets.

Follow up:
What if the inputs contain unicode characters?
 How would you adapt your solution to such case?

https://leetcode.com/problems/valid-anagram/
```

Solution

For ascii characters, maintain 2 arrays containing appearing count for each letter. As for the follow up, can extending the range of array to 65535 work?

Well, there is another solution which uses sorting method. s.toCharArray() then Arrays.sort to see if sorted s and sorted t are equal.....

Listing 155: Solution

```
public class Solution {
    public boolean isAnagram(String s, String t) {
        int[] countS = new int[256];
        int[] countT = new int[256];
        Arrays.fill(countS, 0);
        Arrays.fill(countT, 0);
        if (s.length() != t.length()){
            return false;
        }
        for (int idxS = 0 ; idxS < s.length() ; idxS ++){
            char chS = s.charAt(idxS);
            countS[chS] ++;
        }
        for (int idxT = 0 ; idxT < t.length() ; idxT ++){
            char chT = t.charAt(idxT);
            countT[chT] ++;
        }
        for (int idx = 0 ; idx < 256 ; idx ++){
            if (countS[idx] != countT[idx]){
                return false;
            }
        }
        return true;
    }
}
```

Problem 117

49 Group Anagrams

```
Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],
Return:

[
  ["ate", "eat","tea"],
  ["nat","tan"],
  ["bat"]
]
Note:
For the return value, each inner list's elements
 must follow the lexicographic order.
All inputs will be in lower-case.

https://leetcode.com/problems/anagrams/
```

Solution

I think the solution is somewhat opportunistic... Use Arrays.sort to serve as the fundamental part of the whole program...

However, when I was looking for a more proper solution in the Discuss section, I found that most solutions are of the same trains of thought. In fact, the key insight of solution is not the sorting of charArray but finding a proper hashcode for the Hashmap which stores each group. We may use expressions like "a2b3" to represent each word(don't forget to arrange the order of each letter).

There is another optimal point to check length of each word, words with same length will never be anagrams.

Listing 156: Solution

```
public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> result = new ArrayList<List<String>>();
        HashMap<String, List<String>> anagramGroups
         = new HashMap<String, List<String>>();
        Arrays.sort(strs);
        for (String str : strs){
            char[] charArr = str.toCharArray();
            Arrays.sort(charArr);
            String sortedString = new String(charArr);
            List<String> anaList = new ArrayList<String>();
            if (anagramGroups.containsKey(sortedString)){
                anaList = anagramGroups.get(sortedString);
            }
            anaList.add(str);
            anagramGroups.put(sortedString, anaList);
        }
        Set<String> anagramKeySet = anagramGroups.keySet();
        for (String anagramKey : anagramKeySet){
            result.add(anagramGroups.get(anagramKey));
```

Problem 117 continued on next page...

```
            }
            return result;
        }
}
```

Problem 118

112 Path Sum

```
Given a binary tree and a sum, determine if the tree has a
 root-to-leaf path such that adding up all the values along
  the path equals the given sum.

For example:
Given the below binary tree and sum = 22,
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \      \
        7    2      1
return true, as there exist a root-to-leaf path 5->4->11->2
 which sum is 22.

https://leetcode.com/problems/path-sum/
```

Solution

DFS, bring remaining sum into parameter.

Listing 157: Solution

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public boolean findPath(TreeNode curNode, int remainingSum){
        if (curNode == null)
            return false;
        remainingSum -= curNode.val;
        if (curNode.left == null && curNode.right == null){
            if (remainingSum == 0)
                return true;
            else
                return false;
        }
        return findPath(curNode.left, remainingSum)
         || findPath(curNode.right, remainingSum);
    }
    public boolean hasPathSum(TreeNode root, int sum) {
        return findPath(root, sum);
    }
}
```

Problem 119

113 Path Sum II

```
Given a binary tree and a sum, find all root-to-leaf paths
 where each path's sum equals the given sum.

For example:
Given the below binary tree and sum = 22,
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
return
[
   [5,4,11,2],
   [5,8,4,5]
]

https://leetcode.com/problems/path-sum-ii/
```

Solution

I'm using a new ArrayList to do the dfs since List may be changed during recursion. I don't think this is necessary.

Listing 158: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();

    public void findPath(TreeNode curNode, int remainingSum,
                         List<Integer> curPath){
        if (curNode == null){
            return;
        }
        remainingSum -= curNode.val;
        curPath.add(curNode.val);

        if (curNode.left == null && curNode.right == null){
            if (remainingSum == 0){
                result.add(new ArrayList(curPath));
                return;
```

```
            }
            else{
                return;
            }
        }

        findPath(curNode.left, remainingSum, new ArrayList(curPath));
        findPath(curNode.right, remainingSum, new ArrayList(curPath));
    }

    public List<List<Integer>> pathSum(TreeNode root, int sum) {
        List<Integer> curPath = new ArrayList<Integer>();
        findPath(root, sum, curPath);

        return result;
    }
}
```

Problem 120

110 Balanced Binary Tree

```
Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree
 is defined as a binary tree in which the depth of
  the two subtrees of every node never differ by more than 1.

https://leetcode.com/problems/balanced-binary-tree/
```

Solution 1

Use maxDepth to see, use a top down recursion, though with $O(n^2)runtime$ complexity.
Be aware of the importance of short-circuit logic

Listing 159: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDepth(TreeNode curNode){
        if (curNode == null){
            return 0;
        }

        return Math.max(maxDepth(curNode.left), maxDepth(curNode.right)) + 1;
    }

    public boolean isBalanced(TreeNode root) {
        if (root == null){
            return true;
        }

        return  (Math.abs(maxDepth(root.left) - maxDepth(root.right)) <= 1)
         && isBalanced(root.left) && isBalanced(root.right);
    }
}
```

Solution 2

Another solution is from "Clean Code Handbook". A bottom-up recursion, use -1 to indicate that the subtree is already proved non-balanced.

Listing 160: Solution 2

```
/**
```

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int maxDepth(TreeNode curNode){
        if (curNode == null){
            return 0;
        }

        int lMax = maxDepth(curNode.left);
        if (lMax == -1){
            return -1;
        }
        int rMax = maxDepth(curNode.right);
        if (rMax == -1){
            return -1;
        }

        return (Math.abs(lMax - rMax) > 1)  ? -1 : Math.max(lMax, rMax) + 1;
    }

    public boolean isBalanced(TreeNode root) {
        if (root == null){
            return true;
        }

        return  maxDepth(root) != -1;
    }
}
```

Problem 121

230 Kth Smallest Element in a BST

```
Given a binary search tree, write a function kthSmallest
 to find the kth smallest element in it.

Note:
You may assume k is always valid, 1 <= k <= BST's total elements.

Follow up:
What if the BST is modified (insert/delete operations)
 often and you need to find the kth smallest frequently?
  How would you optimize the kthSmallest routine?

Hint:

Try to utilize the property of a BST.
What if you could modify the BST node's structure?
The optimal runtime complexity is O(height of BST).

https://leetcode.com/problems/kth-smallest-element-in-a-bst/
```

Solution 1

My first solution is to use inorder traversal since the result of such traversal is a list in order, I can easily pick out the kth one, which is what I want. However, it cannot solve the follow up problem(or adding values in the sorted list), neither can it be called a 'clean' code and has the optimized complexity of $O(heightofBST)time$.

Listing 161: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<Integer> result = new ArrayList<Integer>();
    public void findKth(TreeNode curNode){
        if (curNode == null){
            return;
        }
        else{
            findKth(curNode.left);
            result.add(curNode.val);
            findKth(curNode.right);
        }
    }

    public int kthSmallest(TreeNode root, int k) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
```

```
25
            findKth(root);

            return result.get(k - 1);
        }
30  }
```

Solution 2

However, the problem is definitely a standard binary-search like problem. By judging node number of left subtree and right subtree, we can determine which subtree can be used to continue searching.

More information about the train of thought, see

Add property `size` to BST node. Looks like:

```
class TreeNode(object):
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
        self.size = 1  # the size of subtree whose root is current node
```

When `insert/delete`, update `size` too. Then we can get kth smallest element according to `size`.

Visit from `root` of the `BST`:

- if `root.size == k`, return `root.val`.
- if `root.size > k`, search the `k` smallest element of `root.left`
- if `root.size < k`, search the `k - root.size` smallest element of `root.right`

At most we search `height of BST` times, so it's `O(height of BST)`.

Listing 162: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
 class TagTreeNode{
     TreeNode node;
     int size;
     TagTreeNode left;
     TagTreeNode right;
     public TagTreeNode(TreeNode node, int size){
         this.node = node;
         this.size = size;
     }
 }
public class Solution {
    public TagTreeNode buildNode(TreeNode root){
        if (root == null){
            return null;
```

```
        }
        TagTreeNode leftTagNode = buildNode(root.left);
        TagTreeNode rightTagNode = buildNode(root.right);
        int lSize = getSize(leftTagNode);
        int rSize = getSize(rightTagNode);
        int newSize = 1 + lSize + rSize;

        TagTreeNode tagNode = new TagTreeNode(root, newSize);
        tagNode.left = leftTagNode;
        tagNode.right = rightTagNode;
        return tagNode;
    }
    public int getSize(TagTreeNode curNode){
        if (curNode == null){
            return 0;
        }
        return curNode.size;
    }
    public int kthSmallest(TreeNode root, int k) {
        TagTreeNode tagRoot = buildNode(root);
        int result = binarySearch(tagRoot, k);
        return result;
    }
    public int binarySearch(TagTreeNode tagNode, int k){
        int size = getSize(tagNode.left);
        if (size + 1 == k){
            return tagNode.node.val;
        }
        else if (size + 1 > k){
            return binarySearch(tagNode.left, k);
        }else{
            return binarySearch(tagNode.right, k - size - 1);
        }
    }
}
```

Problem 122

278 First Bad Version

```
You are a product manager and currently leading a team to develop a new product.
 Unfortunately, the latest version of your product fails the quality check.
  Since each version is developed based on the previous version,
   all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to
 find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version)
 which will return whether version is bad.
  Implement a function to find the first bad version.
   You should minimize the number of calls to the API.

https://leetcode.com/problems/first-bad-version/
```

Solution

A very simple binary search problem, BUT! use "mid = low + (high - low) / 2;" instead of "mid = (low + high) / 2;" since it may overflow to be negative so that the loop will never end.

Listing 163: Solution

```
/* The isBadVersion API is defined in the parent class VersionControl.
      boolean isBadVersion(int version); */

public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        if (n <= 1){
            return n;
        }
        int low = 1, high = n;
        while (low < high){
            int mid = low + (high - low) / 2;
            if (isBadVersion(mid)){
                high = mid;
            }
            else{
                low = mid + 1;
            }
        }
        return low;
    }
}
```

Problem 123

74 Search a 2D Matrix

```
Write an efficient algorithm that searches for a value in an m x n matrix.
 This matrix has the following properties:

Integers in each row are sorted from left to right.
The first integer of each row is greater than the last integer of the previous row.
For example,

Consider the following matrix:

[
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
Given target = 3, return true.

https://leetcode.com/problems/search-a-2d-matrix/
```

Solution

Starting from [0,0] is very difficult to solve the problem since when we've got a larger element, we can go both downward and to the right. Starting from the top right corner. If current element is less than target, go to the left side; if larger, go downward.

I've missed the condition that 'The first integer of each row is greater than the last integer of the previous row.' So I'm using the solution which also goes for the next Problem "Search a 2D Matrix II".

Listing 164: Solution

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int startRow = 0;
        if (matrix.length == 0)
            return false;
        int startCol = matrix[0].length - 1;
        while (startCol >= 0 && startRow < matrix.length){
            int compare = matrix[startRow][startCol];
            if (compare == target)
                return true;
            else{
                if (target < compare){
                    startCol --;
                }
                else{
                    startRow ++;
                }
            }
        }
        return false;
    }
}
```

Problem 124

240 Search a 2D Matrix II

```
Write an efficient algorithm that searches for a value in an m x n matrix.
 This matrix has the following properties:

Integers in each row are sorted in ascending from left to right.
Integers in each column are sorted in ascending from top to bottom.
For example,

Consider the following matrix:

[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
Given target = 5, return true.

Given target = 20, return false.

https://leetcode.com/problems/search-a-2d-matrix-ii/
```

Solution

Since I've used the method to solve the previous problem, I won't spend too many words here explaining it.

Listing 165: Solution

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int curRow = 0;
        if (matrix.length == 0){
            return false;
        }
        int curCol = matrix[0].length - 1;
        while (curRow < matrix.length && curCol >= 0){
            if (target == matrix[curRow][curCol]){
                return true;
            }
            else if (target < matrix[curRow][curCol]){
                curCol --;
            }
            else{
                curRow ++;
            }
        }
        return false;
    }
}
```

Problem 125

99 Recover Binary Search Tree

```
Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note:
A solution using O(n) space is pretty straight forward.
 Could you devise a constant space solution?

https://leetcode.com/problems/recover-binary-search-tree/
```

Solution 1

Regardless of the problem requirements of $O(1)space$, the problem can be solved using inorder traversal since for BST, the inorder traversal list should be ascending, so if there is one or two mistaken parts, we should swap the values of nodes.

Listing 166: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    List<TreeNode> inorderList = new ArrayList<TreeNode>();

    public void inorderTraverse(TreeNode curNode){
        if (curNode == null){
            return;
        }

        inorderTraverse(curNode.left);
        inorderList.add(curNode);
        inorderTraverse(curNode.right);
    }

    public void recoverTree(TreeNode root) {
        inorderTraverse(root);

        boolean firstToFind = true;
        TreeNode mistakenNode1 = null, mistakenNode2 = null;

        for (int nodeIdx = 0 ; nodeIdx < inorderList.size() - 1 ; nodeIdx++){
            if (inorderList.get(nodeIdx).val >= inorderList.get(nodeIdx + 1).val){
                if (firstToFind){
                    mistakenNode1 = inorderList.get(nodeIdx);
                    mistakenNode2 = inorderList.get(nodeIdx + 1);
```

```
                    firstToFind = false;
                }else{
                    mistakenNode2 = inorderList.get(nodeIdx + 1);
                }
            }
        }

        // swap
        int swapVal = mistakenNode1.val;
        mistakenNode1.val = mistakenNode2.val;
        mistakenNode2.val = swapVal;
    }
}
```

Solution 2

Since the requirement is $O(1)space$, we need to think something strange to us – the Morris traversal. See figures:

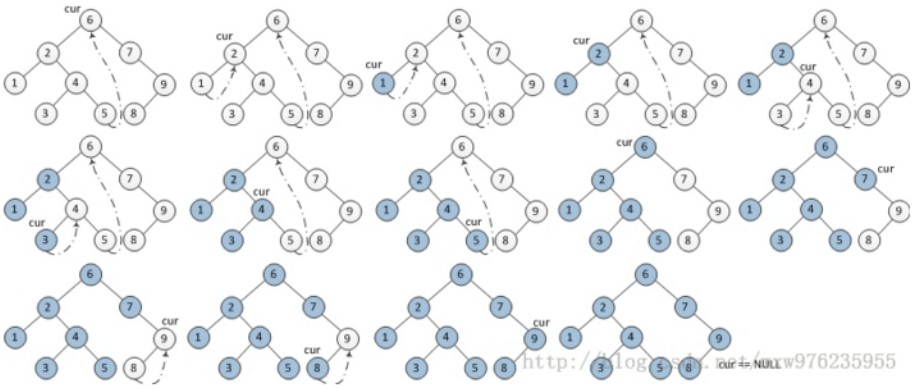
一、中序遍历

步骤：

1. 如果当前节点的左孩子为空，则输出当前节点并将其右孩子作为当前节点。
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。
   a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。当前节点更新为当前节点的左孩子。
   b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空（恢复树的形状）。输出当前节点。当前节点更新为当前节点的右孩子。
3. 重复以上1、2直到当前节点为空。

图示：

下图为每一步迭代的结果（从左至右，从上到下），cur代表当前节点，深色节点表示该节点已输出



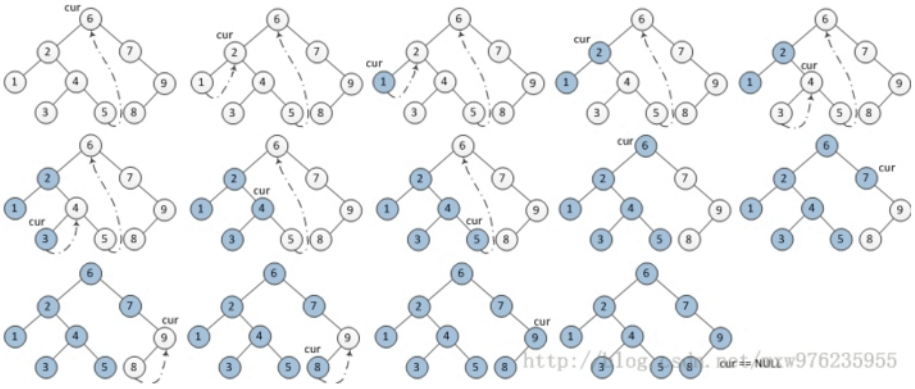
二、前序遍历

前序遍历与中序遍历相似，代码上只有一行不同，不同就在于输出的顺序。

步骤：

1. 如果当前节点的左孩子为空，则输出当前节点并将其右孩子作为当前节点。
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。
   a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。输出当前节点（**在这里输出，这是与中序遍历唯一一点不同**）。当前节点更新为当前节点的左孩子。
   b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空。当前节点更新为当前节点的右孩子。
3. 重复以上1、2直到当前节点为空。

图示：



三、后序遍历

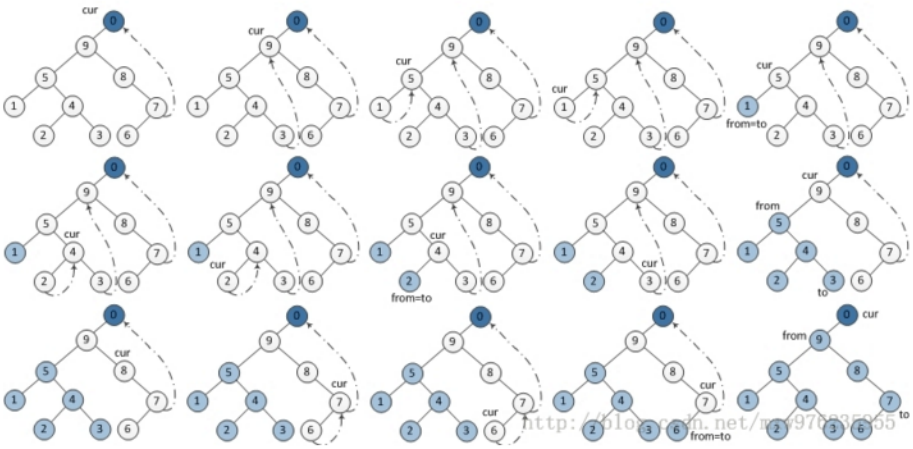
后续遍历稍显复杂，需要建立一个临时节点dump，令其左孩子是root。并且还需要一个子过程，就是倒序输出某两个节点之间路径上的各个节点。

步骤：

当前节点设置为临时节点dump。

1. 如果当前节点的左孩子为空，则将其右孩子作为当前节点。
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。
   a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。当前节点更新为当前节点的左孩子。
   b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空。倒序输出从当前节点的左孩子到该前驱节点这条路径上的所有节点。当前节点更新为当前节点的右孩子。
3. 重复以上1、2直到当前节点为空。

图示：



So simulating the process is the key to achieve this. In the algorithm, when we want to 'print', it means it's the time for us to refresh the value of previous node and make the check if the array is still sorted.

Listing 167: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    TreeNode mistakenNode1 = null, mistakenNode2 = null;
    TreeNode prev = null;
    boolean firstFound = true;
    public void recoverTree(TreeNode root) {
        TreeNode curNode = root;
        while (curNode != null){
            if (curNode.left == null){
                // print
```

```
                if (prev != null){
                    if (curNode.val < prev.val){
                        if (firstFound){
                            mistakenNode1 = prev;
                            mistakenNode2 = curNode;
                            firstFound = false;
                        }
                        else{
                            mistakenNode2 = curNode;
                        }
                    }
                }
                prev = curNode;
                curNode = curNode.right;
            }
            else{
                //curNode.left != null, so find rightmost(predecessor)
                // node in left subtree
                TreeNode movingNode = curNode.left;
                while (movingNode.right != null && movingNode.right != curNode){
                    movingNode = movingNode.right;
                }
                if (movingNode.right == null){
                    movingNode.right = curNode;
                    curNode = curNode.left;
                    // This is important, we should
                    // not add this statement since this is not the right time
                    // To be more specific, now curNode = curNode.left,
                    // so the prev should points to curNode.left.predecessor,
                    // which is exactly what prev used to be
                    // prev = movingNode;
                }
                else{
                    movingNode.right = null;
                    prev = movingNode;
                     if (prev != null){
                        if (curNode.val < prev.val){
                            if (firstFound){
                                mistakenNode1 = prev;
                                mistakenNode2 = curNode;
                                firstFound = false;
                            }
                            else{
                                mistakenNode2 = curNode;
                            }
                        }
                    }
                    prev = curNode;
                    curNode = curNode.right;
                }
            }
        }
```

```
        // swap
        int swapVal = mistakenNode1.val;
        mistakenNode1.val = mistakenNode2.val;
        mistakenNode2.val = swapVal;
    }
}
```

Solution 3

And a recursion solution ... I can hardly understand it.

Listing 168: Solution 3

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
TreeNode *p,*q;
TreeNode *prev;
    void recoverTree(TreeNode *root)
    {
        p=q=prev=NULL;
        inorder(root);
        swap(p->val,q->val);
    }
    void inorder(TreeNode *root)
    {
        if(root->left)inorder(root->left);
        if(prev!=NULL&&(prev->val>root->val))
        {
            if(p==NULL)p=prev;
            q=root;
        }
        prev=root;
        if(root->right)inorder(root->right);
    }
};
```

Problem 126

235 Lowest Common Ancestor of a Binary Search Tree

```
Given a binary search tree (BST), find the lowest common ancestor
 (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia:
  The lowest common ancestor is defined between
  two nodes v and w as the lowest node in T that has
   both v and w as descendants (where we allow a node
    to be a descendant of itself).

          _______6______
        /               \
     ___2__           ___8__
    /      \         /      \
    0      _4        7       9
          /  \
          3   5
For example, the lowest common ancestor (LCA) of
 nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2,
  since a node can be a descendant of itself according to the LCA definition.

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/
```

Solution 1

LCA is a typical kind of problem. If possible, review Tarjan & find-union.

First solution makes use of the characteristics of binary SEARCH tree, when finding the lowest ancestor, the value of the ancestor node v_an should be v_min <= v_an <= v_max.

Listing 169: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q){
        TreeNode curNode = root;
        int minVal = Math.min(p.val, q.val);
        int maxVal = Math.max(p.val, q.val);

        while (!(curNode.val >= minVal && curNode.val <= maxVal)){
            if (curNode.val > maxVal){
                curNode = curNode.left;
            }
            if (curNode.val < minVal){
```

```
                curNode = curNode.right;
            }
        }

        return curNode;
    }
}
```

Solution 2

Solution use the path to solve the problem. Still, we are using the feature of BST. In the next problem we will show a more general algorithm which can be applied to Binary tree.

Listing 170: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<TreeNode> getPath(TreeNode startNode, TreeNode target){
        TreeNode curNode = startNode;
        List<TreeNode> result = new ArrayList<TreeNode>();

        while (curNode != target){
            result.add(curNode);
            if (curNode.val > target.val){
                curNode = curNode.left;
            }else if (curNode.val < target.val){
                curNode = curNode.right;
            }else{
                break;
            }
        }
        result.add(target);

        return result;

    }

    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        List<TreeNode> pathP = new ArrayList<TreeNode>();
        List<TreeNode> pathQ = new ArrayList<TreeNode>();

        pathP = getPath(root, p);
        pathQ = getPath(root, q);

        for (int idx = 0 ; idx < Math.min(pathP.size() , pathQ.size()) ; idx ++){
            if (pathP.get(idx) != pathQ.get(idx)){
```

```
                    return pathP.get(idx - 1);
                }
            }
            return (pathP.size() > pathQ.size()) ? pathQ.get(pathQ.size() - 1)
             : pathP.get(pathP.size() - 1) ;
        }
}
```

Problem 127

236 Lowest Common Ancestor of a Binary Tree

```
Given a binary tree, find the lowest common ancestor (LCA)
 of two given nodes in the tree.

According to the definition of LCA on Wikipedia:
  The lowest common ancestor is defined between
  two nodes v and w as the lowest node in T that has
   both v and w as descendants (where we allow a node
    to be a descendant of itself).
         _______3______
        /              \
     ___5__          ___1__
    /      \        /      \
    6      _2       0       8
          /  \
          7   4
For example, the lowest common ancestor (LCA) of nodes 5 and 1 is 3.
 Another example is LCA of nodes 5 and 4 is 5,
  since a node can be a descendant of itself according to the LCA definition.

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/
```

Solution 1

Since this is just a binary tree, we can only use the characteristic that the two target nodes are respectively at the left and right subtree of their LCA. It's a bottom-up solution.

Listing 171: Solution 1

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode findChild(TreeNode curNode, TreeNode targetP, TreeNode targetQ){
        if (curNode == null){
            return null;
        }
        if (curNode == targetP || curNode == targetQ){
            return curNode;
        }
        TreeNode leftChild = findChild(curNode.left, targetP , targetQ);
        TreeNode rightChild = findChild(curNode.right, targetP, targetQ);
        if (leftChild != null && rightChild != null){
            return curNode;
        }
        return (leftChild == null) ? rightChild : leftChild;
```

```
    }
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        TreeNode lca = findChild(root, p, q);
        return lca;
    }
}
```

Solution 2

Still, use path to check LCA, but this time a little complicated.

Listing 172: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<TreeNode> findPath(TreeNode curNode, TreeNode targetNode,
     List<TreeNode> curPath){
        if (curNode == null){
            return null;
        }
        curPath.add(curNode);
        if (curNode == targetNode){
            return new ArrayList(curPath);
        }
        List<TreeNode> leftPath = findPath(curNode.left, targetNode, curPath);
        List<TreeNode> rightPath = findPath(curNode.right, targetNode, curPath);
        curPath.remove(curPath.size() - 1);
        return (leftPath == null) ? rightPath : leftPath;
    }

    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        List<TreeNode> pathP = new ArrayList<TreeNode>();
        List<TreeNode> pathQ = new ArrayList<TreeNode>();
        pathP = findPath(root, p, new ArrayList<TreeNode>());
        pathQ = findPath(root, q, new ArrayList<TreeNode>());

        for (int idx = 0 ; idx < Math.min(pathP.size() , pathQ.size()) ; idx ++){
            if (pathP.get(idx) != pathQ.get(idx)){
                return pathP.get(idx - 1);
            }
        }
        // lca is the tail of shorter path
        return (pathP.size() > pathQ.size()) ? pathQ.get(pathQ.size() - 1)
         : pathP.get(pathP.size() - 1);
    }
}
```

Problem 128

222 Count Complete Tree Nodes

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

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

https://leetcode.com/problems/count-complete-tree-nodes/
```

Solution 1

Recursive solution. Firstly, simply applying recursion to check all nodes in $O(n)time$ will get TLE, so we need optimization. Given the tree is a complete binary tree, then when left height of left subtree equals to right height of right subtree, the tree is a full binary tree, so the number of nodes can be worked out easily through $2^{height} - 1$.

Note: this is not enough. Using inefficient function like 'Math.pow' will still get TLE, use $1 << height$ instead.

Besides, there is another thing we can optimize is to save the depth of subtree calculated since when we move curNode to its left child, the height of the new left subtree is height(former) - 1, unnecessary to calculate again.

Listing 173: Solution 1

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int getLeftHeight(TreeNode curNode){
        if (curNode == null){
            return 0;
        }
        int height = 0;
        while (curNode != null){
            height ++;
            curNode = curNode.left;
        }
        return height;
    }
    public int getRightHeight(TreeNode curNode){
        if (curNode == null){
            return 0;
        }
        int height = 0;
```

```
        while (curNode != null){
            height ++;
            curNode = curNode.right;
        }
        return height;
    }
    public int countNodes(TreeNode root) {
        int count = 0;
        if (root == null){
            return 0;
        }
        if (getLeftHeight(root.left) == getRightHeight(root.right)){
            return (1 << getLeftHeight(root)) - 1;
        }
        else{
            return 1 + countNodes(root.left) + countNodes(root.right);
        }
    }
}
```

Solution 2

Iterative solution from `https://segmentfault.com/a/1190000003818177`. Look at the left maximum height of left subtree and right subtree, if left equals right, then left tree is full, the end node is at the right subtree and if left is larger, it means that the end node is in left subtree.

Listing 174: Solution 2

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int getLeftHeight(TreeNode curNode){
        if (curNode == null){
            return 0;
        }
        int height = 0;

        while (curNode != null){
            height ++;
            curNode = curNode.left;
        }

        return height;
    }

    public int countNodes(TreeNode root) {
        int count = 0;
```

```
        if (root == null){
            return 0;
        }

        TreeNode curNode = root;
        while (curNode != null){
            int lHeight = getLeftHeight(curNode.left);
            int rHeight = getLeftHeight(curNode.right);

            if (lHeight > rHeight){
                // +1 means addding the root node
                count += ((1 << rHeight) - 1) + 1;
                curNode = curNode.left;
            }
            else {
                // lHeight == rHeight
                count += ((1 << lHeight) - 1) + 1;
                curNode = curNode.right;
            }
        }

        return count;
    }
}
```

Problem 129

108 Convert Sorted Array to Binary Search Tree

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

Given an array where elements are sorted in ascending order,
 convert it to a height balanced BST.

https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/
```

Solution

A binary search problem. Root node is the middle element where left subtree and right subtree are formed from elements from both sides of middle elements.

Listing 175: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public TreeNode buildNode(int[] nums, int start, int end){
        if (start > end){
            return null;
        }
        int mid = start + (end - start) / 2;
        TreeNode rootNode = new TreeNode(nums[mid]);
        rootNode.left = buildNode(nums, start, mid - 1);
        rootNode.right = buildNode(nums, mid + 1 , end);

        return rootNode;
    }

    public TreeNode sortedArrayToBST(int[] nums) {
        return buildNode(nums, 0, nums.length - 1);
    }
}
```

Problem 130

109 Convert Sorted List to Binary Search Tree

```
Given a singly linked list where elements are sorted in ascending order,
 convert it to a height balanced BST.

https://leetcode.com/problems/convert-sorted-list-to-binary-search-tree/
```

Solution

Certainly we can use the solution in previous "Convert Sorted Array to Binary Search Tree", however, it will cost us extra runtime to get the middle element of list(in array, finding a element is of $O(1)runtime$).

So we go for another direction. We can find out that the sorted list is exactly what an inorder traversal of a tree looks like, so we transform the print(or other) operations with iterating the list and form a rootNode.

There is another problem which is related but (much?) more difficult than this one, of "Convert Binary Search Tree (BST) to Sorted Doubly-Linked List", worth pondering.

See `http://articles.leetcode.com/convert-binary-search-tree-bst-to/`

Listing 176: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    private ListNode currentListNode;
    public TreeNode buildTree(int low,  int high){
        if (low > high){
            return null;
        }
        int mid = low + (high - low) / 2;
        TreeNode leftChild = buildTree(low , mid - 1);

        TreeNode rootNode = new TreeNode(currentListNode.val);
        currentListNode = currentListNode.next;

        rootNode.left = leftChild;
        TreeNode rightChild = buildTree(mid + 1 , high);
        rootNode.right = rightChild;
        return rootNode;
```

Problem 130 continued on next page...

```
    }

    public TreeNode sortedListToBST(ListNode head) {
        if (head == null){
            return null;
        }
        currentListNode = head;
        ListNode curNode = head;
        int length = 0;
        while (curNode != null){
            length ++;
            curNode = curNode.next;
        }

        return buildTree(0 , length - 1);
    }
}
```

Problem 131

114 Flatten Binary Tree to Linked List

```
Given a binary tree, flatten it to a linked list in-place.

For example,
Given

         1
        / \
       2   5
      / \   \
     3   4   6
The flattened tree should look like:
   1
    \
     2
      \
       3
        \
         4
          \
           5
            \
             6


Hints:
If you notice carefully in the flattened tree,
 each node's right child points to the next node of a pre-order traversal.

https://leetcode.com/problems/flatten-binary-tree-to-linked-list/
```

Solution

The solution of this one is in fact direct. We can easily notice that the new tree structure follows the preorder traversal order. Then we just:

1. check if curNode.left == null, if not, recurse on its right node(since in preorder traversal, right node comes latter from root node)
2. if curNode.left != null, then left child should be the next element in preorder traversal, so root.left = null, root.right = formerLeft. BUT what about the former right node? Remember that former right node always follows after all nodes in left subtree of rootNode when doing preorder traversal, so we just append them to the right child of the rightMost node of formerLeft(namely current root.right).
3. repeat 1 & 2

Listing 177: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
```

```
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public void flatten(TreeNode root) {
        if (root == null){
            return;
        }

        if (root.left != null){
            TreeNode formerLeft = root.left;
            TreeNode formerRight = root.right;
            root.left = null;
            root.right = formerLeft;

            TreeNode rightMost = formerLeft;
            while(rightMost.right != null){
                rightMost = rightMost.right;
            }
            rightMost.right = formerRight;
        }
        flatten(root.right);
    }
}
```

Problem 132

341 Flatten Nested List Iterator

```
Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list --
 whose elements may also be integers or other lists.

Example 1:
Given the list [[1,1],2,[1,1]],

By calling next repeatedly until hasNext returns false,
the order of elements returned by next should be: [1,1,2,1,1].

Example 2:
Given the list [1,[4,[6]]],

By calling next repeatedly until hasNext returns false,
 the order of elements returned by next should be: [1,4,6].

https://leetcode.com/problems/flatten-nested-list-iterator/
```

Solution

Seemingly I list 2 solutions here, they are in nature similar, the difference just lies in when we flatten the list(to flatten the whole list or flatten it every time we pop an element out of the stack.)

Listing 178: Solution - Flatten it using Stack

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single integer, rather than a nested lis
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger holds, if it holds a single inte
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // @return the nested list that this NestedInteger holds, if it holds a nested list
 *     // Return null if this NestedInteger holds a single integer
 *     public List<NestedInteger> getList();
 * }
 */
public class NestedIterator implements Iterator<Integer> {
    private Stack<NestedInteger> stack = new Stack<NestedInteger>();
    private NestedInteger curInt;

    public NestedIterator(List<NestedInteger> nestedList) {
        push(nestedList);
    }

```

```
    @Override
    public Integer next() {
        return curInt.getInteger();
    }

    @Override
    public boolean hasNext() {
        while (!stack.isEmpty()){
            curInt = stack.pop();
            if (curInt.isInteger()){
                return true;
            }
            push(curInt.getList());
        }
        return false;
    }

    private void push(List<NestedInteger> list){
        for (int nIdx = list.size() - 1; nIdx >= 0; nIdx --){
            stack.push(list.get(nIdx));
        }
    }
}

/**
 * Your NestedIterator object will be instantiated and called as such:
 * NestedIterator i = new NestedIterator(nestedList);
 * while (i.hasNext()) v[f()] = i.next();
 */
```

Listing 179: Solution - Flatten the whole list at first sight

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * public interface NestedInteger {
 *
 *     // @return true if this NestedInteger holds a single integer, rather than a nested lis
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger holds, if it holds a single inte
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // @return the nested list that this NestedInteger holds, if it holds a nested list
 *     // Return null if this NestedInteger holds a single integer
 *     public List<NestedInteger> getList();
 * }
 */
public class NestedIterator implements Iterator<Integer> {
    List<Integer> list = new ArrayList<Integer>();
    Iterator<Integer> ite;

    public NestedIterator(List<NestedInteger> nestedList) {
```

```
        for (NestedInteger nestedInt : nestedList){
            flatten(nestedInt);
        }
        ite = list.iterator();
    }

    private void flatten(NestedInteger nestedInt){
        if (nestedInt.isInteger()){
            list.add(nestedInt.getInteger());
        }else{
            List<NestedInteger> nestedList = nestedInt.getList();
            for (NestedInteger nested : nestedList){
                flatten(nested);
            }
        }
    }

    @Override
    public Integer next() {
        return ite.next();
    }

    @Override
    public boolean hasNext() {
        return ite.hasNext();
    }

}

/**
 * Your NestedIterator object will be instantiated and called as such:
 * NestedIterator i = new NestedIterator(nestedList);
 * while (i.hasNext()) v[f()] = i.next();
 */
```

Problem 133

50 Pow(x, n)

```
Implement pow(x, n).

https://leetcode.com/problems/powx-n/
```

Solution

Implementing using for loop will get TLE(I can hardly imagine how slow loops can be). Moreover, the edge cases are pretty interesting.

So use the formula in figure:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & n\text{为偶数} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & n\text{为奇数} \end{cases}$$

Lots of things to consider!! BTW, use n >>> 1, not n >> 1 to substitute n / 2.

Listing 180: Solution

```
public class Solution {
    public double myPow(double x, int n) {
        if (x - 0.0000001 == 0){
            return 0.0;
        }
        if (n < 0){
            return (1 / pow(x, (long)(-n)));
        }
        return pow(x, (long)n);
    }
    public double pow(double x, long n){
        if (n == 0){
            return 1.0;
        }
        if (n == 1){
            return x;
        }
        double sqrtVal = pow(x, (n >>> 1));
        if ((n & 1) == 1){
            // odd
            return x * sqrtVal * sqrtVal;
        }
        else{
            return sqrtVal * sqrtVal;
        }
    }
}
```

Problem 134

69 Sqrt(x)

```
Implement int sqrt(int x).

Compute and return the square root of x.

https://leetcode.com/problems/sqrtx/
```

Solution

Brute force will definitely TLE. Use binary search to solve the problem. Be aware of the final result returned. For example, sqrt(3) = 1, but $1 * 1 = 1 < 3(3/1 > 1)$

Listing 181: Solution

```
public class Solution {
    public int mySqrt(int x) {
        if (x == 0 || x == 1){
            return x;
        }
        int start = 1;
        int end = x - 1;
        int mid = start + (end - start) / 2;
        while (start <= end){
            mid = start + (end - start) / 2;
            if (mid  == x / mid){
                return mid;
            }
            else{
                if (mid > x / mid){
                    end = mid - 1;
                }
                else{
                    start = mid + 1;
                }
            }
        }
        if (mid > x / mid){
            return mid - 1;
        }
        else{
            return mid;
        }
    }
}
```

Problem 135

65 Valid Number

```
Validate if a given string is numeric.

Some examples:
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
Note: It is intended for the problem statement to be ambiguous.
 You should gather all requirements up front before implementing one.

https://leetcode.com/problems/valid-number/
```

Solution

RIts difficulty lies in so many edge cases of the problem.

Listing 182: Solution

```
public class Solution {
    public boolean isNumber(String s) {
        int idx = 0;
        // skip spaces in the front
        while(idx < s.length() && s.charAt(idx) == ' '){
            idx ++;
        }

        if (idx < s.length() && (s.charAt(idx) == '+' || s.charAt(idx) == '-')){
            idx ++;
        }
        boolean isNumeric = false;
        while (idx < s.length() && Character.isDigit(s.charAt(idx))){
            isNumeric = true;
            idx ++;
        }

        if (idx < s.length() && s.charAt(idx) == '.'){
            idx ++;
            while (idx < s.length() && Character.isDigit(s.charAt(idx))){
                isNumeric = true;
                idx ++;
            }
        }

        if (idx < s.length() && isNumeric && (s.charAt(idx) == 'e'
         || s.charAt(idx) == 'E')){
            idx ++;
            isNumeric = false;
            if (idx < s.length() && (s.charAt(idx) == '+'
             || s.charAt(idx) == '-')){
                idx ++;
```

Problem 135 continued on next page...

```
            }

            while (idx < s.length() && Character.isDigit(s.charAt(idx))){
                isNumeric = true;
                idx ++;
            }
        }

        // skip spaces in the end
        while (idx < s.length() && s.charAt(idx) == ' '){
            idx ++;
        }

        return isNumeric && idx == s.length();
    }
}
```

Problem 136

124 Binary Tree Maximum Path Sum

```
Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some
 starting node to any node in the tree along the parent-child connections.
  The path does not need to go through the root.

For example:
Given the below binary tree,

       1
      / \
     2   3
Return 6.

https://leetcode.com/problems/binary-tree-maximum-path-sum/
```

Solution

It's hard to distinguish whether the problem is hard or not. The idea of solution is the most important.

Listing 183: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    int maxSum = Integer.MIN_VALUE;
    public int findMax(TreeNode root){
        if (root == null){
            return 0;
        }
        int leftMax = findMax(root.left);
        int rightMax = findMax(root.right);
        maxSum = Math.max(maxSum , root.val + leftMax + rightMax);
        int maxPath = Math.max(leftMax, rightMax) + root.val;

        return maxPath > 0 ? maxPath : 0;
    }
    public int maxPathSum(TreeNode root) {
        findMax(root);

        return maxSum;
    }
}
```

Problem 137

26 Remove Duplicates from Sorted Array

```
Given a sorted array, remove the duplicates in place
 such that each element appear only once and return the new length.

Do not allocate extra space for another array,
 you must do this in place with constant memory.

For example,
Given input array nums = [1,1,2],

Your function should return length = 2,
 with the first two elements of nums being 1 and 2 respectively.
  It doesn't matter what you leave beyond the new length.

https://leetcode.com/problems/remove-duplicates-from-sorted-array/
```

Solution

Keep a clean thought will help you solve the problem quickly.

Listing 184: Solution

```
public class Solution {
    public int removeDuplicates(int[] nums) {
        if (nums.length == 0){
            return 0;
        }
        int formerElement = nums[0];
        int length = 1;

        for (int idx = 0 ; idx < nums.length ; idx ++){
            if (nums[idx] != formerElement){
                nums[length ++] = nums[idx];
                formerElement = nums[idx];
            }
        }
        return length;
    }
}
```

Problem 138

80 Remove Duplicates from Sorted Array II

```
Follow up for "Remove Duplicates":
What if duplicates are allowed at most twice?

For example,
Given sorted array nums = [1,1,1,2,2,3],

Your function should return length = 5,
 with the first five elements of nums being 1, 1, 2, 2 and 3.
  It doesn't matter what you leave beyond the new length.

https://leetcode.com/problems/remove-duplicates-from-sorted-array-ii/
```

Solution

Add a flag to check if we still allow a same element exists.

Listing 185: Solution

```
public class Solution {
    public int removeDuplicates(int[] nums) {
        if (nums.length == 0){
            return nums.length;
        }
        int length = 1;
        int formerElement = nums[0];
        boolean allowAnother = true;

        for (int idx = 1 ; idx < nums.length ; idx ++){
            if (formerElement != nums[idx]){
                formerElement = nums[idx];
                nums[length ++] = nums[idx];
                allowAnother = true;
            }
            else{
                // former == current
                if (allowAnother){
                    nums[length ++] = nums[idx];
                    allowAnother = false;
                }
            }
        }

        return length;
    }
}
```

Problem 139

27 Remove Element

```
Given an array and a value, remove all instances of that value
 in place and return the new length.

The order of elements can be changed. It doesn't matter
 what you leave beyond the new length.

https://leetcode.com/problems/remove-element/
```

Solution

So stupid to get 2 WA and a RE on such problem.....

Listing 186: Solution

```
public class Solution {
    public int removeElement(int[] nums, int val) {
        int idx = 0;
        int length = nums.length;

        while (idx < length){
            if (nums[idx] != val){
                idx ++;
                continue;
            }
            while (length > 0 && idx < length && nums[idx] == val){
                int swap = nums[idx];
                nums[idx] = nums[length - 1];
                nums[length - 1] = swap;
                length --;
            }
            idx ++;
        }

        return length;
    }
}
```

Problem 140

283 Move Zeroes

```
Given an array nums, write a function to move all 0's to the end of it
 while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12],
 after calling your function, nums should be [1, 3, 12, 0, 0].

Note:
1. You must do this in-place without making a copy of the array.
2. Minimize the total number of operations.

https://leetcode.com/problems/move-zeroes/
```

Solution

An easy problem.

Listing 187: Solution

```
public class Solution {
    public void moveZeroes(int[] nums) {
        int nonZeroLength = 0;
        int idx = 0;
        while (idx < nums.length){
            if (nums[idx] != 0){
                if (idx != nonZeroLength){
                    nums[nonZeroLength] = nums[idx];
                    nums[idx] = 0;
                }
                nonZeroLength ++;
            }
            idx ++;
        }
    }
}
```

Problem 141

138 Copy List with Random Pointer

```
A linked list is given such that each node
 contains an additional random pointer
  which could point to any node in the list or null.

Return a deep copy of the list.

https://leetcode.com/problems/copy-list-with-random-pointer/
```

Solution 1

Solution 1 makes use of HashMap, mapping original node and copied node so that we can get the random node by querying from HashMap. Use a 2-pass solution to respectively deal with next pointer and random pointer.

Listing 188: Solution 1

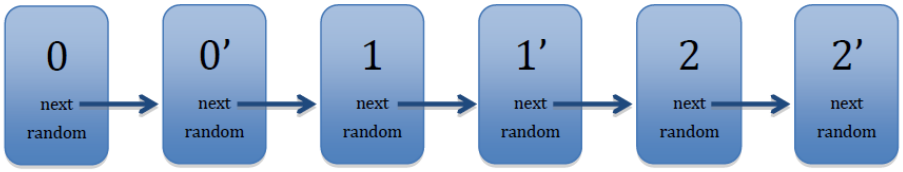
```
/**
 * Definition for singly-linked list with a random pointer.
 * class RandomListNode {
 *     int label;
 *     RandomListNode next, random;
 *     RandomListNode(int x) { this.label = x; }
 * };
 */
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head) {
        if (head == null){
            return null;
        }
        HashMap<RandomListNode,RandomListNode> copyMap
         = new HashMap<RandomListNode,RandomListNode>();

        RandomListNode curNode = head;
        RandomListNode dummy = new RandomListNode(-1);
        RandomListNode curCopyNode = dummy;

        while (curNode != null){
            RandomListNode newCopyNode = new RandomListNode(curNode.label);
            curCopyNode.next = newCopyNode;
            copyMap.put(curNode, newCopyNode);
            curNode = curNode.next;
            curCopyNode = curCopyNode.next;
        }

        curNode = head;
        curCopyNode = dummy;

        while(curNode != null){
            curCopyNode.next.random = copyMap.get(curNode.random);
            curNode = curNode.next;
            curCopyNode = curCopyNode.next;
```

Problem 141 continued on next page...

```
        }

        return dummy.next;
    }
}
```

Solution 2

Solution 2 changes the structure of the list. See figure



So that we can just use
$node.next.random = node.random.next$

Listing 189: Solution 2

```
/**
 * Definition for singly-linked list with a random pointer.
 * class RandomListNode {
 *     int label;
 *     RandomListNode next, random;
 *     RandomListNode(int x) { this.label = x; }
 * };
 */
public class Solution {
    public RandomListNode copyRandomList(RandomListNode head) {
        if (head == null){
            return null;
        }
        RandomListNode curNode = head;
        while(curNode != null){
            RandomListNode copyNode = new RandomListNode(curNode.label);
            RandomListNode nextNode = curNode.next;
            curNode.next = copyNode;
            copyNode.next = nextNode;

            curNode = curNode.next.next;
        }

        curNode = head;
        while (curNode != null){
            curNode.next.random = (curNode.random != null)
             ? curNode.random.next : null;
            curNode = curNode.next.next;
        }

        //recover the original structure, seperate the two list
        curNode = head;
        RandomListNode dummy = new RandomListNode(-1);
        RandomListNode tail = dummy;
```

```
35
        while (curNode != null){
            RandomListNode copyNode = curNode.next;
            RandomListNode nextNode = copyNode.next;
            tail.next = copyNode;
40          tail = tail.next;

            curNode.next = nextNode;
            curNode = nextNode;
        }
45
        return dummy.next;
    }
}
```

Problem 142

207 Course Schedule

```
There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0
 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs,
 is it possible for you to finish all courses?

For example:

2, [[1,0]]
There are a total of 2 courses to take.
 To take course 1 you should have finished course 0. So it is possible.

2, [[1,0],[0,1]]
There are a total of 2 courses to take.
 To take course 1 you should have finished course 0,
  and to take course 0 you should also have finished course 1.
   So it is impossible.

Note:
The input prerequisites is a graph represented by a list of edges,
 not adjacency matrices.


Hints:
1. This problem is equivalent to finding if a cycle exists in a directed graph.
 If a cycle exists, no topological ordering exists
  and therefore it will be impossible to take all courses.
2. Topological Sort via DFS - A great video tutorial (21 minutes)
 on Coursera explaining the basic concepts of Topological Sort.
3. Topological sort could also be done via BFS.

https://leetcode.com/problems/course-schedule/
```

Solution 1

A graph problem, use topological sort to solve. Equivalent to problems which identify whether there is a cycle in the graph. Use adjacency list to store edges for a easier calculation on indegrees. BFS sort.

Listing 190: Solution 1

```
public class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        if (prerequisites.length == 0){
            return true;
        }
        int[] indegree = new int[numCourses];
        List<List<Integer>> adjacency = new ArrayList<List<Integer>>(numCourses);
        for (int i = 0 ; i < numCourses ; i ++){
            adjacency.add(new ArrayList<Integer>());
```

```
        }

        for (int i = 0 ; i < prerequisites.length ; i ++){
            adjacency.get(prerequisites[i][1]).add(prerequisites[i][0]);
            indegree[prerequisites[i][0]] ++;
        }

        Queue<Integer> outQueue = new LinkedList<Integer>();

        for (int courseIdx = 0 ; courseIdx < numCourses; courseIdx++){
            if (indegree[courseIdx] == 0){
                outQueue.offer(courseIdx);
            }
        }

        int courseCount = 0;
        while (!outQueue.isEmpty()){
            int courseIdx = outQueue.poll();

            List<Integer> latterCourses = adjacency.get(courseIdx);
            for (int latterCourseIdx : latterCourses){
                indegree[latterCourseIdx] --;
                if (indegree[latterCourseIdx] == 0){
                    outQueue.offer(latterCourseIdx);
                }
            }

            courseCount ++;
        }

        return courseCount == numCourses;


    }
}
```

Solution 2

Solution 2 uses DFS. We should explain the usage of array 'visited'. For value 0, it means the node has not been visited at all; value 1 means that the adjacent nodes has already been visited; while value -1 means the node has been visited while the adjacencies are not totally visited.

Listing 191: Solution 2

```
public class Solution {

    private boolean traverse(int courseIdx,ArrayList<Integer>[] adjacency
        ,int[] visited ){
        visited[courseIdx] = -1;
        for (int adjacencyEle : adjacency[courseIdx]){
            if (visited[adjacencyEle] == -1){
                return false;
            }
            else if (visited[adjacencyEle] == 0){
```

```
                    if (!traverse(adjacencyEle, adjacency,visited)){
                        return false;
                    }
                }
            }

            visited[courseIdx] = 1;
            return true;
        }

        public boolean canFinish(int numCourses, int[][] prerequisites) {
            ArrayList<Integer>[] adjacency = new ArrayList[numCourses];
            for (int i = 0 ; i < numCourses ; i ++){
                adjacency[i] = new ArrayList<Integer>();
            }
            for (int i = 0 ; i < prerequisites.length ; i ++){
                adjacency[prerequisites[i][1]].add(prerequisites[i][0]);
            }

            int[] visited = new int[numCourses];
            for (int courseIdx = 0 ; courseIdx < numCourses ; courseIdx ++){
                if (visited[courseIdx] == 0){
                    if (!traverse(courseIdx, adjacency,visited)){
                        return false;
                    }
                }
            }


            return true;

        }
    }
```

Problem 143

210 Course Schedule II

```
There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0
 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs,
 return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them.
 If it is impossible to finish all courses, return an empty array.

For example:

    2, [[1,0]]
There are a total of 2 courses to take.
 To take course 1 you should have finished course 0.
  So the correct course order is [0,1]

    4, [[1,0],[2,0],[3,1],[3,2]]
There are a total of 4 courses to take.
 To take course 3 you should have finished both courses 1 and 2.
  Both courses 1 and 2 should be taken after you finished course 0.
   So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].

Note:
The input prerequisites is a graph represented by a list of edges,
 not adjacency matrices.


Hints:
1. This problem is equivalent to finding if a cycle exists in a directed graph.
 If a cycle exists, no topological ordering exists
  and therefore it will be impossible to take all courses.
2. Topological Sort via DFS - A great video tutorial (21 minutes)
 on Coursera explaining the basic concepts of Topological Sort.
3. Topological sort could also be done via BFS.

https://leetcode.com/problems/course-schedule-ii/
```

Solution

Similar to "Course Schedule", add a step to record each step, namely record the sequence polled from queue.

Listing 192: Solution

```
public class Solution {
    public int[] findOrder(int numCourses, int[][] prerequisites) {
        // ADD this statement will get a WA
        // if (prerequisites.length == 0){
        //     return new int[numCourses];
```

```
        // }
        int[] indegree = new int[numCourses];

        List<List<Integer>> adjacency = new ArrayList<List<Integer>>();
        for (int i = 0 ; i < numCourses ; i++){
            adjacency.add(new ArrayList<Integer>());
        }

        for (int preIdx = 0 ; preIdx < prerequisites.length ; preIdx ++){
            adjacency.get(prerequisites[preIdx][1]).
            add(prerequisites[preIdx][0]);
            indegree[prerequisites[preIdx][0]] ++;
        }

        Queue<Integer> startingCourses = new LinkedList<Integer>();
        for (int courseIdx = 0 ; courseIdx < numCourses ; courseIdx ++){
            if (indegree[courseIdx] == 0){
                startingCourses.offer(courseIdx);
            }
        }

        int[] result = new int[numCourses];
        int courseNum = 0;
        while (!startingCourses.isEmpty()){
            int courseIdx = startingCourses.poll();
            for (int latterCourseIdx : adjacency.get(courseIdx)){
                if (--indegree[latterCourseIdx] == 0){
                    startingCourses.offer(latterCourseIdx);
                }
            }

            result[courseNum] = courseIdx;
            courseNum ++;
        }

        return (courseNum == numCourses) ? result : new int[0];
    }
}
```

Problem 144

1 Two Sum

```
Given an array of integers, return indices of the two numbers
 such that they add up to a specific target.

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

Example:
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].

https://leetcode.com/problems/two-sum/
```

Solution

It's a very interesting series of problems.

For all kSum conclusion , there is a good blog here

`http://www.sigmainfy.com/blog/summary-of-ksum-problems.html`

As for this problem, there are mainly two ways, except for the Brute force method. First way is to use HashMap.

Solution utilizes two pointers to get the **unique** result also works. Sorting the list first is needed as a prerequisite. However, we need to keep the original index of number array. So here we just list the HashMap solution.

Listing 193: Solution

```
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        int[] result = new int[2];
        HashMap<Integer, Integer> valueIdxMap = new HashMap<Integer, Integer>();

        for (int idx = 0 ; idx < nums.length ; idx ++){
            if (valueIdxMap.containsKey(target - nums[idx])){
                result[0] = valueIdxMap.get(target - nums[idx]);
                result[1] = idx;
                break;
            }
            valueIdxMap.put(nums[idx], idx);
        }

        return result;

    }
}
```

Problem 145

167 Two Sum II - Input array is sorted

```
Given an array of integers that is already sorted in ascending order,
 find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers
 such that they add up to the target, where index1 must be less than index2.
  Please note that your returned answers (both index1 and index2) are not 0-based.

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

Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2

https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/
```

Solution - Two Pointers

Continue with solution discussion in preceding problem. Another way of solving the problem is to use 2-pointers. Note that the array should be sorted, which may lead to larger complexity among unsorted array. 2 pointers starts respectively from the beginning and ending of array, and when current sum is larger than target, move ending pointer forward, while on the contrast moving beginning pointer backward until they meet.

Moreover, a trick for not getting TLE is to skip duplicate elements. Imagine an sample input of [0,0,...,0,4,9,...,9] with target=4. For pointers, we can feel free to use while loop to skip duplicates to prevent from costing too much time. If the target is 0, then it will be obtained by both idx1 and idx2, not influenced by a skipping process of a single idx pointer.

Listing 194: Solution

```
public class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int[] result = new int[2];
        int idx1 = 0, idx2 = numbers.length - 1;
        while (numbers[idx1] + numbers[idx2] != target){
            if (numbers[idx1] + numbers[idx2] < target){
                idx1 ++;
                while (idx1 < idx2 && numbers[idx1] == numbers[idx1 - 1]){
                    idx1 ++;
                }
            }
            else{
                idx2 --;
                while (idx1 < idx2 && numbers[idx2] == numbers[idx2 + 1]){
                    idx2 --;
                }
            }
        }
        result[0] = idx1 + 1;
        result[1] = idx2 + 1;
        return result;
    }
```

Problem 145 continued on next page...

```
}
```

Solution - Binary Search

Another way is to make use of binary search. Iterate elements in array(until the element is larger or equals to target / 2, think of it), and use binary search to find (target - cur) in array backwards. However the complexity may be a bit larger($O(nlogn)$) than 2-pointers method.

Listing 195: Solution

```
public class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int[] result = new int[2];
        int idx1 = 0;
        while (numbers[idx1] <= target / 2){
            int rem = target - numbers[idx1];
            int idx2 = binarySearch(numbers, rem, idx1 + 1);
            if (idx2 != -1){
                result[0] = idx1 + 1;
                result[1] = idx2 + 1;
                break;
            }
            idx1 ++;

            while (numbers[idx1] <= target / 2
           && numbers[idx1] == numbers[idx1 - 1]){
                idx1 ++;
            }
        }
        return result;
    }

    private int binarySearch(int[] numbers, int rem, int startIdx){
        int low = startIdx, high = numbers.length - 1;
        while (low < high){
            int mid = low + (high - low) / 2;
            if (numbers[mid] == rem){
                return mid;
            }
            else if (numbers[mid] > rem){
                high = mid - 1;
            }
            else{
                low = mid + 1;
            }
        }
        if (numbers[low] != rem){
            return -1;
        }
        else{
            return low;
        }
    }
}
```

Problem 146

15 3Sum

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

Note:
Elements in a triplet (a,b,c) must be in non-descending order. (ie, a <= b <= c)
The solution set must not contain duplicate triplets.
    For example, given array S = {-1 0 1 2 -1 -4},

    A solution set is:
    (-1, 0, 1)
    (-1, -1, 2)

https://leetcode.com/problems/3sum/
```

Solution

Finally we can use the 2 pointers solution.

Note that in the problem, the description request us to prevent from producing duplicate triples, so we need a Set or just skip those duplicate elements to solve this problem.

Listing 196: Solution

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

        for (int idx1 = 0 ; idx1 < nums.length - 2 ; idx1 ++){
            while (idx1 > 0 && idx1 < nums.length - 2
             && nums[idx1] == nums[idx1 - 1]){
                idx1 ++;
            }
            int val1 , val2 , val3;
            val1 = nums[idx1];

            int idx2 = idx1 + 1, idx3 = nums.length - 1;
            while (idx2 < idx3){
                val2 = nums[idx2];
                val3 = nums[idx3];

                int sum = val1 + val2 + val3;

                if (sum == 0){
                    List<Integer> triple = new ArrayList<Integer>();
                    triple.add(val1);
                    triple.add(val2);
                    triple.add(val3);

                    results.add(triple);
```

```
                    while(idx2 < idx3 && nums[idx3] == nums[idx3 - 1]){
                        idx3 --;
                    }
                    idx3 --;
                    idx2++;
                    while(idx2 < idx3 && nums[idx2 - 1] == nums[idx2]){

                        idx2 ++;
                    }
                }else if (sum > 0){

                    while(idx2 < idx3 && nums[idx3] == nums[idx3 - 1]){
                        idx3 --;
                    }
                    idx3 --;
                }else{
                    idx2 ++;
                    while(idx2 < idx3 && nums[idx2 - 1] == nums[idx2]){

                        idx2 ++;
                    }
                }
            }

        }

        return results;
    }
}
```

Problem 147

16 3Sum Closest

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

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

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

https://leetcode.com/problems/3sum-closest/
```

Solution

Personally, I think this is a bit easier than 3sum since I don't need to care about the duplicates.

Listing 197: Solution

```java
public class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);

        int diff = Integer.MAX_VALUE;
        int closest = -1;

        for (int idx1 = 0 ; idx1 < nums.length - 2 ; idx1 ++){
            int idx2 = idx1 + 1;
            int idx3 = nums.length - 1;
            while (idx2 < idx3){
                int sum = nums[idx1] + nums[idx2] + nums[idx3];
                int curDiff = Math.abs(sum - target);
                if (curDiff == 0){
                    return target;
                }
                if (curDiff < diff){
                    diff = curDiff;
                    closest = sum;
                }
                if (sum < target){
                    idx2 ++;
                }
                if (sum > target){
                    idx3 --;
                }

            }
        }

        return closest;
    }
}
```

Problem 148

18 4Sum

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

Note:
Elements in a quadruplet (a,b,c,d) must be in non-descending order.
 (ie, a    b    c    d)
The solution set must not contain duplicate quadruplets.
    For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

    A solution set is:
    (-1,  0, 0, 1)
    (-2, -1, 1, 2)
    (-2,  0, 0, 2)

https://leetcode.com/problems/4sum/
```

Solution

Solving the kSum problem! Just one try!!!(Though not bug free...)

Listing 198: Solution

```
public class Solution {
    public List<List<Integer>> kSum(int k , int[] nums, int target,
     int start, int end){
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        if (k == 2){
            return twoSum(nums, target, start, end);
        }
        else{
            for (int idx = start; idx <= end; idx ++){
                while (idx > start && idx <= end && nums[idx] == nums[idx - 1]){
                    idx ++;
                }
                if (idx > end){
                    break;
                }
                int curEle = nums[idx];
                List<List<Integer>> oldList = kSum(k - 1 , nums,
                 target - curEle, idx + 1, end);
                if (oldList != null){
                    result.addAll(combine(curEle, oldList));
                }
            }
        }
        return (result.size() != 0) ? result : null;
    }
    public List<List<Integer>> twoSum(int[] nums, int target,
     int start, int end){
        List<List<Integer>> result = new ArrayList<List<Integer>>();
```

Problem 148 continued on next page...

```
        int idx1 = start;
        int idx2 = end;
        while (idx1 < idx2){
            int val1 = nums[idx1];
            int val2 = nums[idx2];
            int sum = val1 + val2;
            if (sum == target){
                List<Integer> newPair = new ArrayList<Integer>();
                newPair.add(val1);
                newPair.add(val2);
                result.add(newPair);
                idx2 --;
                while(idx2 >= start && idx2 < end
                 && nums[idx2] == nums[idx2 + 1]){
                    idx2 --;
                }
                idx1 ++;
                while (idx1 > start && idx1 <= end
                 && nums[idx1] == nums[idx1 - 1]){
                    idx1 ++;
                }
            }
            else if (sum > target){
                idx2 --;
            }
            else{
                idx1 ++;
            }
        }
        return (result.size() == 0) ? null : result;
    }
    public List<List<Integer>> combine(int ele, List<List<Integer>> oldList){
        List<List<Integer>> newList = new ArrayList<List<Integer>>();
        for (List<Integer> oldItem : oldList){
            List<Integer> newItem = new ArrayList<Integer>();
            newItem.add(ele);
            newItem.addAll(oldItem);
            newList.add(newItem);
        }
        return newList;
    }
    public List<List<Integer>> fourSum(int[] nums, int target) {
        Arrays.sort(nums);
        List<List<Integer>> result = kSum(4, nums, target, 0, nums.length - 1);
        return result == null ? new ArrayList<List<Integer>>() : result ;
    }
}
```

Problem 149

55 Jump Game

```
Given an array of non-negative integers, you are initially
 positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:
A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.

https://leetcode.com/problems/jump-game/
```

Solution

A greedy or DP problem. The maximum steps we can go on is Math.max(curMax, nums[idx]), don't forget to reduce curMax by 1 simulating every step we proceed.

Listing 199: Solution

```
public class Solution {
    public boolean canJump(int[] nums) {
        if (nums.length == 0 || nums.length == 1){
            return true;
        }
        int[] dp = new int[nums.length];
        int curMax = nums[0] + 1;

        for (int idx = 0 ; idx < nums.length ; idx ++){
            curMax --;
            curMax = Math.max(nums[idx] , curMax);

            if (curMax == 0){
                return false;
            }
            if (curMax + idx >= nums.length - 1){
                return true;
            }

        }

        return true;
    }
}
```

A more concise version. Do not need to deduct 1, only need to check if we can reach the first elements finally.

Problem 149 continued on next page...

Listing 200: Solution1.1

```
public class Solution {
    public boolean canJump(int[] nums) {
        int des = nums.length - 1;
        for(int i = des - 1; i >= 0; i--){
            if(nums[i] >= des - i){
                des = i;
            }
        }
        return des <= 0;
    }
}
```

Problem 150

45 Jump Game II

```
Given an array of non-negative integers,
 you are initially positioned at the first index of the array.

Each element in the array represents
 your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

For example:
Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2.
 (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

Note:
You can assume that you can always reach the last index.

https://leetcode.com/problems/jump-game-ii/
```

Solution

Still greedy, but needs some time to figure out the correctness....(Honestly, I don't think this is a complete greedy problem, since the solution is not that greedy. It's more like a DP problem.)

Let's say the range of the current jump is [curBegin, lastMax], curMax is the farthest point that all points in [curBegin, lastMax] can reach. Once the current point reaches lastMax, then trigger another jump, and set the new lastMax with curMax.

Listing 201: Solution

```
public class Solution {
    public int jump(int[] nums) {
        int curMax = 0;
        int lastMax = 0;
        int stepCnt = 0;

        for (int currentIdx = 0 ; currentIdx < nums.length ; currentIdx ++){
            // Of no use since you can always reach the last index
            // if (i < curMax){
            //     return -1;
            // }

            if (currentIdx > lastMax){
                lastMax = curMax;
                stepCnt ++;
            }

            curMax = Math.max(curMax, currentIdx + nums[currentIdx]);
        }

        return stepCnt;
```

```
        }
    }
```

Problem 151

120 Triangle

```
Given a triangle, find the minimum path sum from top to bottom.
 Each step you may move to adjacent numbers on the row below.

For example, given the following triangle
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

Note:
Bonus point if you are able to do this using only O(n) extra space,
 where n is the total number of rows in the triangle.

https://leetcode.com/problems/triangle/
```

Solution

A dp problem, I think it is maybe in an easier level of dp problems.

As for the bonus, manipulating on triangle variable itself is a fine idea. Just remember to mention the boundaries.

Listing 202: Solution

```java
public class Solution {
    public int jump(int[] nums) {
        int curMax = 0;
        int lastMax = 0;
        int stepCnt = 0;

        for (int currentIdx = 0 ; currentIdx < nums.length ; currentIdx ++){
            // Of no use since you can always reach the last index
            // if (i < curMax){
            //     return -1;
            // }

            if (currentIdx > lastMax){
                lastMax = curMax;
                stepCnt ++;
            }

            curMax = Math.max(curMax, currentIdx + nums[currentIdx]);
        }

        return stepCnt;
    }
}
```

Problem 152

187 Repeated DNA Sequences

```
All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T,
 for example: "ACGAATTCCG". When studying DNA,
  it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings)
 that occur more than once in a DNA molecule.

For example,

Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",

Return:
["AAAAACCCCC", "CCCCCAAAAA"].

https://leetcode.com/problems/repeated-dna-sequences/
```

Solution

An interesting problem but not that easy. Using String Hashmap will get MLE. So we need to reduce the size of our data. That's when "Bit Manipulation" comes into sight. Since there are only 4 kinds of chars, we can mark them as '00','01','10','11', so that we can use a 20-bit integer as the hash code of each 10-digit string, as it won't cost as much memory as String. See code in listing.

Listing 203: Solution

```
public class Solution {
    public List<String> findRepeatedDnaSequences(String s) {
        List<String> result = new ArrayList<String>();
        if (s.length() <= 9){
            return new ArrayList<String>();
        }

        HashMap<Character, Integer> digitMapping
         = new HashMap<Character, Integer>();
        digitMapping.put('A', 0);
        digitMapping.put('C', 1);
        digitMapping.put('G', 2);
        digitMapping.put('T', 3);

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

        int hash = 0;
        int curLen = 0;
        for (int strIdx = 0 ; strIdx < s.length() ; strIdx ++){
            curLen ++;

            hash = (hash << 2) + digitMapping.get(s.charAt(strIdx));
            if(curLen > 9) {
                hash = (hash & ((1 << 20) - 1));
```

Problem 152 continued on next page...

```
                if (countMapping.containsKey(hash)
                 && countMapping.get(hash) == 1){
                    countMapping.put(hash, countMapping.get(hash) + 1);
                    String repeatedStr = s.substring(strIdx - 9 , strIdx + 1);
                    result.add(repeatedStr);
                }
                else if (!countMapping.containsKey(hash)){
                    countMapping.put(hash, 1);
                }
            }
        }

        return result;
    }
}
```

Problem 153

331 Verify Preorder Serialization of a Binary Tree

```
One way to serialize a binary tree is to use pre-order traversal.
 When we encounter a non-null node, we record the node's value.
  If it is a null node, we record using a sentinel value such as #.

      _9_
     /   \
    3     2
   / \   / \
  4   1  #  6
 / \ / \   / \
 # # # #   # #
For example, the above binary tree can be serialized
 to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values,
 verify whether it is
  a correct preorder traversal serialization of a binary tree.
  Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be
 either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid,
 for example it could never contain two consecutive commas such as "1,,3".

Example 1:
    "9,3,4,#,#,1,#,#,2,#,6,#,#"
    Return true

Example 2:
    "1,#"
    Return false

Example 3:
    "9,#,#,1"
    Return false

https://leetcode.com/problems/verify-preorder-serialization-of-a-binary-tree/
```

Solution

A stack problem may be more suitable for the orientation of this problem. The train of thought is somewhat like the Evaluation of Reverse Polish Expression.

When scrape nodes from a complete subtree together, we may combine them into a '#' to simplify the tree structure, finally we should only get a single '#' in the stack.

To be more specific, we first push every element we get into stack; if depth of current stack is larger or equal to 3 and the top 2 elements are both '#' when the 3rd element is not(which will never happen, even if the original string contains 3 successive '#', the first 2 will be combined into one single '#'). We pop the 2 '#'s out along with the 3rd non-'#' element and push an extra '#' instead.

Problem 153 continued on next page...

Repeat the whole process.
Finally check if the remaining element in Stack is '#'.

Listing 204: Solution

```
public class Solution {
    public boolean isValidSerialization(String preorder) {
        Stack<String> serialization = new Stack<String>();
        String[] parts = preorder.split(",");
        for (String part : parts){
            if (!part.equals("#")){
                serialization.push(part);
            }
            else{
                if (serialization.isEmpty()
                 || !(serialization.peek().equals("#"))){
                    serialization.push(part);
                }
                else{
                    while (!serialization.isEmpty()
                     && serialization.peek().equals("#")){
                        String sharp = serialization.pop();

                        if (serialization.isEmpty()){
                            return false;
                        }
                        String formerElement = serialization.pop();
                        if (formerElement.equals("#")){
                            return false;
                        }
                    }
                    serialization.push("#");
                }
            }
        }

        if (serialization.isEmpty()){
            return false;
        }
        if (!serialization.pop().equals("#")
         || !serialization.isEmpty()){
            return false;
        }
        return true;
    }
}
```

Problem 154

72 Edit Distance

```
Given two words word1 and word2,
 find the minimum number of steps required to convert word1 to word2.
  (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

a) Insert a character
b) Delete a character
c) Replace a character

https://leetcode.com/problems/edit-distance/
```

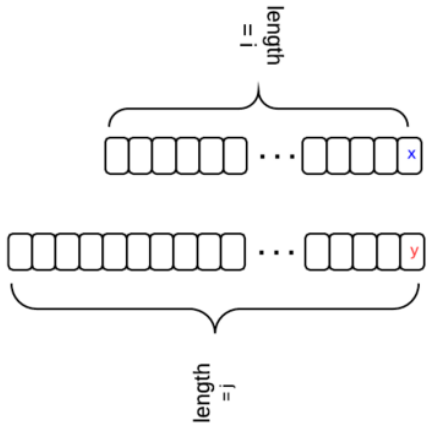
Solution

A representative DP problem.

The train of thought is listed as below:

Let dp[i][j] stands for the edit distance between two strings with length *i* and *j*, i.e., word1[0,...,i-1] and word2[0,...,j-1].

There is a relation between dp[i][j] and dp[i-1][j-1]. Let's say we transform from one string to another. The first string has length *i* and it's last character is "x"; the second string has length *j* and its last character is "y". The following diagram shows the relation.



1. if x == y, then dp[i][j] == dp[i-1][j-1]
2. if x != y, and we insert y for word1, then dp[i][j] = dp[i][j-1] + 1
3. if x != y, and we delete x for word1, then dp[i][j] = dp[i-1][j] + 1
4. if x != y, and we replace x with y for word1, then dp[i][j] = dp[i-1][j-1] + 1
5. When x!=y, dp[i][j] is the min of the three situations.

Initial condition:

dp[i][0] = i, dp[0][j] = j

Listing 205: Solution

```java
public class Solution {
    public int minDistance(String word1, String word2) {
```

```
        int[][] minDis = new int[word1.length() + 1][word2.length() + 1];

        for (int idx1 = 0 ; idx1 <= word1.length() ; idx1 ++){
            minDis[idx1][0] = idx1;
        }
        for (int idx2 = 0 ; idx2 <= word2.length() ; idx2 ++){
            minDis[0][idx2] = idx2;
        }
        for (int idx1 = 1 ; idx1 <= word1.length() ; idx1 ++){
            for (int idx2 = 1 ; idx2 <= word2.length() ; idx2 ++){
                int diff = 0;
                if (word1.charAt(idx1 - 1) != word2.charAt(idx2 - 1)){
                    diff = 1;
                }
                minDis[idx1][idx2] = Math.min(minDis[idx1 - 1][idx2 - 1]
                 + diff, Math.min(minDis[idx1 - 1][idx2] + 1
                  , minDis[idx1][idx2 - 1] + 1));
            }
        }

        return minDis[word1.length()][word2.length()];
    }
}
```

Problem 155

319 Bulb Switcher

```
There are n bulbs that are initially off. You first turn on all the bulbs.
 Then, you turn off every second bulb.
  On the third round, you toggle every third bulb
   (turning on if it's off or turning off if it's on).
    For the ith round, you toggle every i bulb. For the nth round,
     you only toggle the last bulb. Find how many bulbs are on after n rounds.

Example:

Given n = 3.

At first, the three bulbs are [off, off, off].
After first round, the three bulbs are [on, on, on].
After second round, the three bulbs are [on, off, on].
After third round, the three bulbs are [on, off, off].

So you should return 1, because there is only one bulb is on.

https://leetcode.com/problems/bulb-switcher/
```

Solution

This is a very very very very interesting problem! Simulating the process is too time-consuming, so we need to find another way.

It can be easily discovered that when we switch the light for odd times, the state of light will change, while switching for even times won't. Now consider all numbers, for example, 12 = 1*12; 2*6; 3*4, will be switched 6 times, its state won't change; however, as for 25= 1*25; 5*5, it will change its state!!

So the problem is exactly asking us to find how many square numbers are in the range of 1 n.

Listing 206: Solution

```java
public class Solution {
    public int bulbSwitch(int n) {
        return new Double(Math.sqrt(n)).intValue();
    }
}
```

Problem 156

42 Trapping Rain Water

```
Given n non-negative integers representing an elevation map
 where the width of each bar is 1,
  compute how much water it is able to trap after raining.

For example,
Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.


The above elevation map is represented by array
 [0,1,0,2,1,0,1,3,2,1,2,1]. In this case,
  6 units of rain water (blue section) are being trapped.

https://leetcode.com/problems/trapping-rain-water/
```

Solution

Use 2 pointers is the key to this problem. One from left, one from right, find the lower value for the candidate and find 'traps' whose depth is lower than current minimum(between left and right) and add the depth to amount of total water, until finding a new peak.

Listing 207: Solution

```
public class Solution {
    public int trap(int[] height) {
        int startIdx = 0 , endIdx = height.length -1;

        if (height.length == 0){
            return 0;
        }
        int lHeight = height[startIdx];
        int rHeight = height[endIdx];
        int lowerHeight = 0;
        int totalWater = 0;

        while(startIdx < endIdx){
            lowerHeight = Math.min(lHeight , rHeight);
            if (lHeight == lowerHeight){
                lHeight = height[++startIdx];
```

```
                while (startIdx < endIdx && lowerHeight >= lHeight){
                    totalWater += lowerHeight - lHeight;
                    startIdx ++;
                    lHeight = height[startIdx];
                }
            }
            else if (rHeight == lowerHeight){
                rHeight = height[--endIdx];
                while (startIdx < endIdx && lowerHeight >= rHeight){
                    totalWater += lowerHeight - rHeight;
                    endIdx --;
                    rHeight = height[endIdx];
                }
            }
        }

        return totalWater;
    }
}
```

Problem 157

11 Container With Most Water

```
Given n non-negative integers a1, a2, ..., an,
 where each represents a point at coordinate (i, ai).
  n vertical lines are drawn such that the two endpoints
    of line i is at (i, ai) and (i, 0). Find two lines,
     which together with x-axis forms a container,
      such that the container contains the most water.

https://leetcode.com/problems/container-with-most-water/
```

Solution

Somewhat like 2Sum, use 2 pointers to achieve this. The greedy algorithm needs proving.

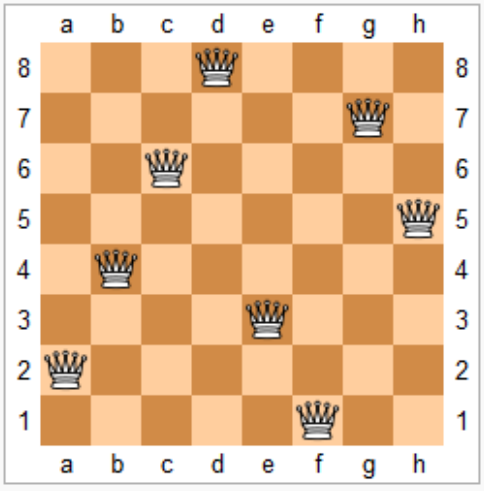
Listing 208: Solution

```
public class Solution {
    public int maxArea(int[] height) {
        int max = -1;
        int i = 0 , j = height.length - 1;
        while (i < j){
            max = Math.max(max, (j - i) * Math.min(height[i] , height[j]));
            if (height[i] < height[j])
                i ++;
            else
                j --;
        }
        return max;
    }
}
```

Problem 158

51 N-Queens

```
The n-queens puzzle is the problem of placing n queens
 on an n*n chessboard such that no two queens attack each other.



Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of
 the n-queens' placement,
  where 'Q' and '.' both indicate a queen and an empty space respectively.

For example,
There exist two distinct solutions to the 4-queens puzzle:

[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]

https://leetcode.com/problems/n-queens/
```



One solution to the eight queens puzzle

Solution

"N-queens" is a representative series of problem in backtracking problems.
However the train of thought is not that difficult, instead it's somewhat straightforward.
In the next problem 'N-queen-ii', another solution with $O(n)space$ will be displayed.

Listing 209: Solution

```
public class Solution {
        List<List<String>> puzzles = new ArrayList<List<String>>();

        public void solveNQueens(int n, int curRow, int[][] puzzle){
            if (curRow == n){
                puzzles.add(generatePuzzle(puzzle));
                return ;
            }
            for (int col = 0 ; col < n ; col ++){
                if (isValid(puzzle, curRow, col)){
                    puzzle[curRow][col] = 1;
                    solveNQueens(n, curRow + 1, puzzle);
                    puzzle[curRow][col] = 0;
                }
            }
        }

        public List<List<String>> solveNQueens(int n) {
            int[][] puzzle = new int[n][n];

            solveNQueens(n, 0, puzzle);

            return puzzles;
        }

        public boolean isValid(int[][] puzzle, int row, int col){
            // check row
            for (int i = 0 ; i < col ; i++){
                if (puzzle[row][i] == 1){
                    return false;
                }
            }

            // check column
            for (int i = 0 ; i < row ; i++){
                if (puzzle[i][col] == 1){
                    return false;
                }
            }
            // check top left
            for (int i = 1 ; i <= Math.min(row, col) ; i++){
                if (puzzle[row - i][col - i] == 1){
                    return false;
                }
            }
            // check top right
            for (int i = 1 ; i <= Math.min(row, puzzle.length - col - 1) ; i ++){
                if (puzzle[row - i][col + i] == 1){
                    return false;
                }
            }
```

```
            return true;
        }

        public List<String> generatePuzzle(int[][] puzzle){
            List<String> result = new ArrayList<String>();
            for (int i = 0 ; i < puzzle.length ; i ++){
                String newLine = "";
                for (int j = 0 ; j < puzzle[0].length ; j++){
                    if (puzzle[i][j] == 0){
                        newLine += ".";
                    }
                    else{
                        newLine += "Q";
                    }
                }
                result.add(newLine);
            }

            return result;
        }
}
```

Problem 159

52 N-Queens II

```
Follow up for N-Queens problem.

Now, instead outputting board configurations,
 return the total number of distinct solutions.

https://leetcode.com/problems/n-queens-ii/
```

Solution

This is not much harder than N-Queen, the ideas are the same. But this time a $O(n)space$ solution will be displayed. It doesn't use a two-dimensional array to keep the condition of the puzzle, instead use an a[i] array where a[i] representing the index of i^{th} row.

So how to judge whether the position is available? The array is of 1-dimension, so there won't be conflict in rows, checking whether columns conflict are also not difficult, what about diagonalization? Observation gives us the answer that when two point fit the attribute that x1 + y1 = x2 + y2 OR x1 - y1 = x2 - y2, eg (1,2) & (2,1) OR (1,2) & (2,3), they are on the same diagonal line.

Listing 210: Solution

```java
public class Solution {
    int count = 0;

    public void solveNQueens(int total, int row, int[] cols){
        if (total == row){
            count++;
            return;
        }
        for (int col = 0 ; col < total; col ++){
            if (isValid(cols, row, col)){
                cols[row] = col;
                solveNQueens(total, row + 1 , cols);
                cols[row] = -1;
            }
        }
    }

    public boolean isValid(int[] cols, int row, int col){

        for (int i = 0 ; i < row ; i++){
            // check same col
            if (cols[i] == col){
                return false;
            }
            // check top left & top right
            if (row - col == i - cols[i] || row + col == i + cols[i]){
                return false;
            }
        }

        return true;
    }
```

```
    public int totalNQueens(int n) {
        int[] cols = new int[n];
        for (int i = 0 ; i < n ; i++){
            cols[i] = -1;
        }
        solveNQueens(n, 0, cols);
        return count;
    }
}
```

Problem 160

217 Contains Duplicate

```
Given an array of integers, find if the array contains any duplicates.
 Your function should return true if any value appears
  at least twice in the array,
   and it should return false if every element is distinct.

https://leetcode.com/problems/contains-duplicate/
```

Solution

A HashSet is enough for the problem series I and II.

Listing 211: Solution

```
public class Solution {
    public boolean containsDuplicate(int[] nums) {
        Set<Integer> uniqueElements = new HashSet<Integer>();
        for (int num : nums){
            if (uniqueElements.contains(num)){
                return true;
            }
            uniqueElements.add(num);
        }

        return false;
    }
}
```

Problem 161

219 Contains Duplicate II

```
Given an array of integers and an integer k,
 find out whether there are two distinct indices i and j
  in the array such that nums[i] = nums[j]
   and the difference between i and j is at most k.

https://leetcode.com/problems/contains-duplicate-ii/
```

Solution

Either use a HashMap to record the latest index of element or use a k-length slide window is fine. Here I'm using a k-length slide window(made of HashSet) to solve the problem.

Listing 212: Solution

```
public class Solution {
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        Set<Integer> kSlidingWindow = new HashSet<Integer>();
        for (int idx = 0 ; idx < nums.length ; idx ++){
            if (kSlidingWindow.contains(nums[idx])){
                return true;
            }
            kSlidingWindow.add(nums[idx]);
            if (kSlidingWindow.size() > k){
                kSlidingWindow.remove(nums[idx - k]);
            }
        }

        return false;
    }
}
```

Problem 162

220 Contains Duplicate III

```
Given an array of integers,
 find out whether there are two distinct indices
  i and j in the array such that the difference
   between nums[i] and nums[j] is at most t and
    the difference between i and j is at most k.

https://leetcode.com/problems/contains-duplicate-iii/
```

Solution

Use a existence data structure TreeSet, which has both the attribute of Set and Binary Search Tree, a perfect match of this problem.

Listing 213: Solution

```
public class Solution {
    public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {
        TreeSet<Integer> almostUniqueElements = new TreeSet<Integer>();

        for (int idx = 0 ; idx < nums.length ; idx ++){
            if ((almostUniqueElements.floor(nums[idx]) != null &&
            (nums[idx] <= t + almostUniqueElements.floor(nums[idx]))) ||
            (almostUniqueElements.ceiling(nums[idx]) != null &&
            (almostUniqueElements.ceiling(nums[idx]) <= t + nums[idx] ))
            ){
                return true;
            }

            almostUniqueElements.add(nums[idx]);
            if (almostUniqueElements.size() > k){
                almostUniqueElements.remove(nums[idx - k]);
            }
        }

        return false;
    }
}
```

Problem 163

75 Sort Colors

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

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

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


Follow up:
A rather straight forward solution is a two-pass algorithm
 using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's,
 then overwrite array with total number of 0's, then 1's and followed by 2's.

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

https://leetcode.com/problems/sort-colors/
```

Solution

Use two pointers to mark where the index of 0 and 2 are(In fact, there are 3 pointers, the main idx pointer marks where 1s are.)

In LintCode, there is a Sort Colors II, for k colors. The key insight is like bucket sorting.

Listing 214: Solution

```java
public class Solution {
    public void sortColors(int[] nums) {
        int idxZero = 0 , idxTwo = nums.length - 1;
        int idx = 0;
        while (idx <= idxTwo){
            if (nums[idx] == 0){
                swap(nums, idx, idxZero);
                idx ++;
                idxZero ++;
            }
            else if (nums[idx] == 2){
                swap(nums, idx, idxTwo);
                // idx ++;
                idxTwo --;
            }
            else{
                idx ++;
            }
        }
    }
```

```
    public void swap(int[] nums, int idx1, int idx2){
        int swap = nums[idx1];
        nums[idx1] = nums[idx2];
        nums[idx2] = swap;
    }
}
```

Problem 164

39 Combination Sum

```
Given a set of candidate numbers (C) and a target number (T),
 find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:
All numbers (including target) will be positive integers.
Elements in a combination (a1, a2, ... , ak) must be in non-descending order.
 (ie, a1 <= a2 <= ... <= ak).
The solution set must not contain duplicate combinations.
For example, given candidate set 2,3,6,7 and target 7,
A solution set is:
[7]
[2, 2, 3]

https://leetcode.com/problems/combination-sum/
```

Solution

A dfs problem. Remember to sort the array first since *Elements in a combination* $(a1, a2, ..., ak)$ *must be in non−descending order.*

Listing 215: Solution

```
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        Arrays.sort(candidates);
        List<Integer> combination = new ArrayList<Integer>();
        findSum(candidates, target, 0, combination);
        return result;
    }
    public void findSum(int[] candidates, int target, int curIdx,
     List<Integer> curCombination){
        int curSum = 0;
        for (int com : curCombination)
            curSum += com;
        if (curSum > target)
            return;
        if (curSum == target){
            result.add(new ArrayList(curCombination));
            return;
        }
        for (int idx = curIdx ; idx < candidates.length; idx ++){
            curCombination.add(candidates[idx]);
            findSum(candidates, target, idx, curCombination);
            curCombination.remove(curCombination.size() - 1);
        }
    }
}
```

Problem 165

40 Combination Sum II

```
Given a collection of candidate numbers (C) and a target number (T),
 find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:
All numbers (including target) will be positive integers.
Elements in a combination (a1, a2, ... , ak)
 must be in non-descending order. (ie, a1 <= a2 <= ... <= ak).
The solution set must not contain duplicate combinations.
For example, given candidate set 10,1,2,7,6,1,5 and target 8,
A solution set is:
[1, 7]
[1, 2, 5]
[2, 6]
[1, 1, 6]

https://leetcode.com/problems/combination-sum-ii/
```

Solution

More difficult than "Combination Sum", mainly on how to deal with duplicates. Except for set the startIdx to lastIdx + 1 to avoid using the same elements, we should also use a while loop to avoid using elements with the same value after we have already combined them.

Listing 216: Solution

```
public class Solution {
    List<List<Integer>> combinations = new ArrayList<List<Integer>>();

    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        Arrays.sort(candidates);
        List<Integer> curCombination = new ArrayList<Integer>();
        // curIdx is initially -1 since the array should start iterating from 0
        // and -1 + 1 = 0
        findSum(candidates, target, -1, curCombination);
        return combinations;
    }

    private void findSum(int[] candidates, int target, int curIdx,
    List<Integer> curCombination){
        int curSum = 0;
        for (int element : curCombination){
            curSum += element;
        }
        if (target < curSum){
            return;
        }
        if (target == curSum){
            combinations.add(new ArrayList(curCombination));
            return;
```

```
        }

        for (int idx = curIdx + 1 ; idx < candidates.length ; idx ++){
            curCombination.add(candidates[idx]);
            findSum(candidates, target, idx, curCombination);
            curCombination.remove(curCombination.size() - 1);
            // To avoid duplicates, think of it.
            // For example, 11167,
            // after 1->11->111->1116->11167->1117-> next what?
            // (116..., if we don't add such while loop,
            //           it will continue to add 111...)
            while (idx < candidates.length - 1
                 && candidates[idx] == candidates[idx + 1]){
                idx ++;
            }
        }
    }
}
```

Problem 166

216 Combination Sum III

```
Find all possible combinations of k numbers that add up to a number n,
 given that only numbers from 1 to 9 can be used and
  each combination should be a unique set of numbers.

Ensure that numbers within the set are sorted in ascending order.


Example 1:

Input: k = 3, n = 7

Output:

[[1,2,4]]

Example 2:

Input: k = 3, n = 9

Output:

[[1,2,6], [1,3,5], [2,3,4]]

https://leetcode.com/problems/combination-sum-iii/
```

Solution

Don't know if it is my illusion, I think this Problem is a bit easier than "Combination Sum II" since it doesn't need to be processed to avoid duplicates.

Listing 217: Solution

```
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    public List<List<Integer>> combinationSum3(int k, int n) {
        List<Integer> curCombination = new ArrayList<Integer>(k);
        for (int i = 0 ; i < k ; i++){
            curCombination.add(0);
        }
        findSum(k , 0 , 0 , n, curCombination);

        return result;
    }
    private void findSum(int maxDepth, int curDepth,
     int curNum,  int target, List<Integer> curCombination){
        int curSum = 0;
        for (int element : curCombination){
            curSum += element;
        }

        if (curSum > target){
```

```
            return;
        }
        if (curDepth == maxDepth){
            if (curSum == target){
                result.add(new ArrayList(curCombination));
            }
            return;
        }

        for (int num = curNum + 1 ; num <= 9 ; num ++){
            curCombination.set(curDepth, num);
            findSum(maxDepth, curDepth + 1, num, target, curCombination);
            curCombination.set(curDepth, 0);
        }

    }
}
```

Problem 167

377 Combination Sum IV

```
Given an integer array with all positive numbers and no duplicates,
 find the number of possible combinations that add up to a positive integer target.
Example:
nums = [1, 2, 3]
target = 4

The possible combination ways are:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)

Note that different sequences are counted as different combinations.

Therefore the output is 7.

Follow up:
What if negative numbers are allowed in the given array?
How does it change the problem?
What limitation we need to add to the question to allow negative numbers?

https://leetcode.com/problems/combination-sum-iv/
```

Solution - TLE

Raw backtracking will get TLE, like this.

Listing 218: Solution 1.1 TLE backtracking

```
public class Solution {
    int count = 0;
    public int combinationSum4(int[] nums, int target) {
        combination4(nums, target, 0);
        return count;
    }

    private void combination4(int[] nums, int target, int sum){
        if (sum == target){
            count ++;
            return;
        }
        else if (sum > target){
            return;
        }
        else{
            for (int i = 0 ; i < nums.length; i ++){
                combination4(nums, target, sum + nums[i]);
            }
```

Problem 167 continued on next page...

```
        }
    }
}
```

Solution - Pruning with memoization

We may use a Hashmap to store results for pruning.

Listing 219: Solution 1.2 Memoization backtracking

```
public class Solution {
    Map<Integer, Integer> sums = new HashMap<Integer, Integer>();
    public int combinationSum4(int[] nums, int target) {
        int count = 0;
        if (target < 0){
            return 0;
        }
        if (target == 0){
            return 1;
        }
        for (int num : nums){
            if (sums.containsKey(target - num)){
                count += sums.get(target - num);
                continue;
            }
            count += combinationSum4(nums, target - num);
        }

        sums.put(target, count);
        return count;
    }

}
```

Solution - DP

The thought is alike memoization, use dp[i] += dp[i - num]

Listing 220: Solution 2 DP

```
public class Solution {
    int count = 0;
    public int combinationSum4(int[] nums, int target) {
        int[] res = new int[target + 1];
        for (int sum = 1 ; sum <= target ; sum ++){
            for (int num : nums){
                if (num > sum){
                    continue;
                }
                else if (num == sum){
                    res[sum] ++;
                }
                else{
                    res[sum] += res[sum - num];
                }
```

```
                }
            }
            return res[target];
        }

}
```

Problem 168

48 Rotate Image

```
You are given an n * n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up:
Could you do this in-place?

https://leetcode.com/problems/rotate-image/
```

Solution

Finding the rule of swapping is the key to solving the problem. (Honestly, my solution is not exactly in-place since I used a boolean array to check if the number in grid has been changed. I'll list the proper code downward my solution.

However, the world has taught us not to under-estimate every problem, there is also another interesting way of solving the problem: when we regard the matrix as a paper, we fold the paper up and down, then fold it along the diagonal, we will realize this is exactly the result we want!

Listing 221: Solution

```
public class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        if (n == 0){
            return;
        }
        boolean[][] swapped = new boolean[n][n];
        for (int i = 0 ; i < n ; i ++){
            for (int j = 0 ; j < n ; j ++){
                if (swapped[i][j]){
                    continue;
                }
                int startX = i , startY = j;
                int swapX = startY, swapY = n - 1 - startX;
                swapped[i][j] = true;
                int swap = matrix[startX][startY];

                while (startX != swapX || startY != swapY){
                    int tmp = matrix[swapX][swapY];
                    matrix[swapX][swapY] = swap;
                    swap = tmp;
                    swapped[swapX][swapY] = true;
                    // swap swapX & swapY
                    swapX = swapX + swapY;
                    swapY = swapX - swapY;
                    swapX = swapX - swapY;
                    swapY = n - 1 - swapY;
                }
                matrix[startX][startY] = swap;
            }
```

```
        }
    }
}


// Solution WITHOUT boolean array
public class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        int limit = (n-1)/2;
        for(int i=0;i<= limit; i++){
            for(int j=i;j<n-1-i;j++){
                int temp = matrix[i][j];
                matrix[i][j] = matrix[n-1-j][i];
                matrix[n-1-j][i] = matrix[n-1-i][n-1-j];
                matrix[n-1-i][n-1-j] = matrix[j][n-1-i];
                matrix[j][n-1-i] = temp;
            }
        }
    }
}
```

Problem 169

53 Maximum Subarray

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

For example, given the array [-2,1,-3,4,-1,2,1,-5,4],
the contiguous subarray [4,-1,2,1] has the largest sum = 6.


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

https://leetcode.com/problems/maximum-subarray/
```

Solution

This is a conventional DP problem, it's worth us carefully thinking about it.

What's the first solution comes into mind at first sight? Enumeration, of course, the Brute force method. However, when we talk about its complexity, we may need $O(n)$ to iterate both i and j, along with an extra $O(n)$ effort to sum up a[i...j], $O(N^3)$ in total.

Listing 222: Guide 1

```
for(int i = 1; i <= n; i++){
    for(int j = i; j <= n; j++){
        int sum = 0;
        for(int k = i; k <= j; k++)
            sum += a[k];
        max = Max(max, sum);
    }
}
```

How to optimize? Can we calculate the sum during the process of enumeration? The sum is actually formerSum + a[idx-1].

Listing 223: Guide 2

```
for(int i = 1; i <= n; i++){
    int sum = 0;
    for(int j = i; j <= n; j++){
        sum += a[j];
        max = Max(max, sum);
    }
}
```

Divide and Conquer can even promote our solution further more. We may find a middle point, and the maximum subarray sum lies either on left, right, or go through the middle element(since we are finding consecutive sequence). Then the steps of algorithm is to separate subarray according to a middle element and calculate optimal sum sum1 and sum2 start from middle element(With the time complexity $2*T(n/2)+O(n)=O(nlogn)$);

Problem 169 continued on next page...

Finally we can use DP to solve it. Let dp[i] be the maximum we can get until idx i, so dp[i] = maxdp[i - 1] + nums[i], nums[i].

Listing 224: Solution

```
public class Solution {
    public int maxSubArray(int[] nums) {
        int[] maxDp = new int[nums.length];
        if (nums.length == 0){
            return 0;
        }
        int totalMax = nums[0];
        maxDp[0] = nums[0];

        for (int idx = 1 ; idx < nums.length ; idx ++){
            maxDp[idx] = Math.max(maxDp[idx - 1] + nums[idx], nums[idx]);
            totalMax = Math.max(maxDp[idx], totalMax);
        }

        return totalMax;
    }
}
```

Solution Follow-up

The follow up need us to do it in a divide and conquer way which is mentioned above, as listing, the code is copied from Internet, respectively calculating sums [startIdx, currentIdx], [currentIdx + 1, endIdx] and [startIdx, endIdx], use a binary search method to decide the value of currentIdx(so the runtime complexity is $O(nlogn)$).

Listing 225: Solution Follow-up

```
public class Solution {
    public int maxSum(int[] A, int left, int right )
    {
        if( left == right ){
            return A[left];
        }
        int center = (left + right) / 2;
        int maxLeftSum  = maxSum( A, left, center);
        int maxRightSum = maxSum( A, center+1, right);


        int maxLeft = Integer.MIN_VALUE, tempLeft = 0;
        int maxRight = Integer.MIN_VALUE, tempRight = 0;
        for (int i=center; i>=left; --i){
            tempLeft += A[i];
            if (tempLeft > maxLeft){
                maxLeft = tempLeft;
            }
        }
        for (int i=center+1; i<=right; ++i){
            tempRight += A[i];
            if (tempRight > maxRight){
                maxRight = tempRight;
```

```
            }
        }

        int maxCenterSum = maxLeft + maxRight;

        return maxCenterSum > maxLeftSum ?
            (maxCenterSum > maxRightSum ?
                maxCenterSum : maxRightSum) : maxLeftSum > maxRightSum ?
                    maxLeftSum : maxRightSum;
    }

    public int maxSubArray(int[] A){
        int len = A.length;
        return maxSum(A,0,len-1);
    }
}
```

Problem 170

152 Maximum Product Subarray

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

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

https://leetcode.com/problems/maximum-product-subarray/
```

Solution

The solution of this problem is a bit like "Maximum Subarray", except for the fact that we use an extra array to store the minimum product until idx since minimum value may also produce maximum value since negatives multiply negatives equal positives.

Listing 226: Solution

```
public class Solution {
    public int maxProduct(int[] nums) {
        int[] curMin = new int[nums.length];
        int[] curMax = new int[nums.length];
        if (nums.length == 0){
            return 0;
        }
        int totalMax = nums[0];
        curMax[0] = curMin[0] = nums[0];

        for (int idx = 1 ; idx < nums.length ; idx ++){
            curMax[idx] = Math.max(
                Math.max(curMax[idx - 1] * nums[idx],
                    curMin[idx - 1] * nums[idx]),
                nums[idx]);
            curMin[idx] = Math.min(
                Math.min(curMax[idx - 1] * nums[idx],
                    curMin[idx - 1] * nums[idx]),
                nums[idx]);
            totalMax = Math.max(curMax[idx], totalMax);
        }

        return (int)totalMax;
    }
}
```

Problem 171

238 Product of Array Except Self

```
Given an array of n integers where n > 1, nums,
 return an array output such that output[i]
  is equal to the product of all the elements of nums except nums[i].

Solve it without division and in O(n).

For example, given [1,2,3,4], return [24,12,8,6].

Follow up:
Could you solve it with constant space complexity?
 (Note: The output array does not count as extra space
  for the purpose of space complexity analysis.)

https://leetcode.com/problems/product-of-array-except-self/
```

Solution

Solution using $O(n)space$ is trivial, so I just put the $O(1)space$(except for the result array) solution here. It's like multiplying a matrix, from start of array to the end, result[i] = result[i - 1] * a[i - 1](avoiding multiplying a[i]), then backward, result[i] = result[i] * a[i + 1].

Listing 227: Solution

```
public class Solution {
    public int[] productExceptSelf(int[] nums) {
        if (nums.length == 0){
            return new int[0];
        }
        int[] products = new int[nums.length];
        products[0] = 1;
        for (int idx = 1 ; idx < nums.length ; idx++){
            products[idx] = products[idx - 1] * nums[idx - 1];
        }

        int product = 1;
        for (int idx = nums.length - 2 ;  idx >= 0 ; idx --){
            product *= nums[idx + 1];
            products[idx] *= product;
        }

        return products;
    }
}
```

Problem 172

133 Clone Graph

```
Clone an undirected graph.
 Each node in the graph contains a label and a list of its neighbors.


OJ's undirected graph serialization:
Nodes are labeled uniquely.

We use # as a separator for each node, and ,
 as a separator for node label and each neighbor of the node.
As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes,
 and therefore contains three parts as separated by #.

First node is labeled as 0. Connect node 0 to both nodes 1 and 2.
Second node is labeled as 1. Connect node 1 to node 2.
Third node is labeled as 2. Connect node 2 to node 2 (itself),
 thus forming a self-cycle.
Visually, the graph looks like the following:

       1
      / \
     /   \
    0 --- 2
         / \
         \_/

https://leetcode.com/problems/clone-graph/
```

Solution

Using a HashMap to map original node and cloned node(just like what we did in 'Copy List With Random Pointer'), then use BFS or DFS to clone the whole graph. For practise, I list DFS solution here.

Listing 228: Solution

```
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     List<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors
 *       = new ArrayList<UndirectedGraphNode>(); }
 * };
 */
public class Solution {
    HashMap<UndirectedGraphNode, UndirectedGraphNode> map
     = new HashMap<UndirectedGraphNode, UndirectedGraphNode>();
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null){
            return null;
```

```
        }

        UndirectedGraphNode newNode = new UndirectedGraphNode(node.label);
        if (!map.containsKey(node)){
            map.put(node, newNode);
        }
        else{
            return map.get(node);
        }
        for (UndirectedGraphNode neighborNode : node.neighbors){
            if (map.containsKey(neighborNode)){
                map.get(node).neighbors.add(map.get(neighborNode));
            }
            else{
                map.get(node).neighbors.add(cloneGraph(neighborNode));
            }
        }

        return newNode;
    }
}
```

Problem 173

153 Find Minimum in Rotated Sorted Array

```
Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

You may assume no duplicate exists in the array.

https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/
```

Solution

A binary search problem. Of course we can do it in $O(n)runtime$ by iterating the list linearly, but binary search is of course better for its $O(logn)runtime$.

Listing 229: Solution

```
public class Solution {
    public int findMin(int[] nums) {
        if (nums.length == 0){
            return -1;
        }
        int start = 0, end = nums.length - 1;

        while (start < end){
            int mid = start + (end - start) / 2;
            if (nums[start] == nums[mid] && nums[start] > nums[start + 1]){
                return nums[start + 1];
            }
            else{
                if (nums[start] < nums[mid]){
                    start = mid;
                }
                else{
                    end = mid;
                }
            }
        }

        return nums[0];
    }
}
```

Problem 174

154 Find Minimum in Rotated Sorted Array II

```
Follow up for "Find Minimum in Rotated Sorted Array":
What if duplicates are allowed?

Would this affect the run-time complexity? How and why?
Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

The array may contain duplicates.

https://leetcode.com/problems/find-minimum-in-rotated-sorted-array-ii/
```

Solution

This one is following previous problem "Find Minimum in Rotated Sorted Array", and though there are only several lines of code that are different, the problem is much harder than previous one. Take care of the condition when there are plenty of duplicates, for example, when the list is 10,1,10,10,10, this is when a[low] < a[high], which is not we want when using binary search(Think of why?).

Listing 230: Solution

```
public class Solution {
    public int findMin(int[] nums) {
        if (nums.length == 0)
            return 0;
        int start = 0 , end = nums.length - 1;
        while (start < end){
            // this is weird since we should always consider the condition
            // that a[l] > a[r]
            if (nums[start] < nums[end])
                return nums[start];
            int mid = start + (end - start) / 2;
            if (mid == start){
                if (nums[start] > nums[start + 1])
                    return nums[start + 1];
            }
            if (nums[start] > nums[mid])
                end = mid;
            else if (nums[end] < nums[mid])
                start = mid;
            else
                start ++;
        }
        return nums[0];
    }
}
```

Problem 175

33 Search in Rotated Sorted Array

```
Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search.
 If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

https://leetcode.com/problems/search-in-rotated-sorted-array/
```

Solution

A complicated binary search problem, need to check if the target is in monotone interval(always increasing) or on the other side. Don't forget to check if the element matches start or end pointer of list.

Listing 231: Solution

```
public class Solution {
    public int search(int[] nums, int target) {
        int start = 0 , end = nums.length - 1;
        while (start <= end){
            int mid = start + (end - start) / 2;
            if (nums[mid] == target){
                return mid;
            }
            if (nums[start] < nums[end]){
                if (target > nums[mid]){
                    start = mid + 1;
                }
                else{
                    end = mid - 1;
                }
            }
            else{
                if (nums[start] < nums[mid]){
                    if (target == nums[start]){
                        return start;
                    }
                    if (target > nums[start] && target < nums[mid]){
                        end = mid - 1;
                    }
                    else{
                        start = mid;
                    }
                }
                else{
                    if (target == nums[end]){
                        return end;
                    }
                    if (target > nums[mid] && target < nums[end]){
```

Problem 175 continued on next page...

```
                        start = mid + 1;
                    }
                    else{
                        end = mid - 1;
                    }
                }
            }
        }

        return -1;
    }
}
```

Problem 176

81 Search in Rotated Sorted Array II

```
Follow up for "Search in Rotated Sorted Array":
What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

https://leetcode.com/problems/search-in-rotated-sorted-array-ii/
```

Solution

With the experience of last problem "Search in Rotated Sorted Array", solving this problem is not that difficult. Still, when we find nums[start] == nums[mid] == nums[end], we just increase start by 1 since we are not sure where our target is([10,1,10,10,10] & [10,10,10,1,10]).

What's more, in this solution I slightly simplify my code in previous problem.

Listing 232: Solution

```
public class Solution {
    public boolean search(int[] nums, int target) {
        int start = 0 , end = nums.length - 1;
        while (start <= end){
            int mid = start + (end - start) / 2;
            if (target == nums[mid]){
                return true;
            }
            if (nums[start] < nums[end]){
                if (target > nums[mid]){
                    start = mid + 1;
                }
                else{
                    end = mid - 1;
                }
            }
            else{
                if (nums[start] > nums[mid]){
                    if (target > nums[mid] && target <= nums[end]){
                        start = mid + 1;
                    }
                    else{
                        end = mid - 1;
                    }
                }
                else if (nums[start] < nums[mid]){
                    if (target >= nums[start] && target < nums[mid]){
                        end = mid - 1;
                    }
                    else{
                        start = mid + 1;
                    }
```

Problem 176 continued on next page...

```
                    }
                    else{
                        start ++;
                    }
                }
            }

            return false;
        }
}
```

Problem 177

76 Minimum Window Substring

```
Given a string S and a string T,
 find the minimum window in S which
  will contain all the characters in T in complexity O(n).

For example,
S = "ADOBECODEBANC"
T = "ABC"
Minimum window is "BANC".

Note:
If there is no such window in S
 that covers all characters in T, return the empty string "".

If there are multiple such windows,
 you are guaranteed that there will always be
  only one unique minimum window in S.

https://leetcode.com/problems/minimum-window-substring/
```

Solution

A troublesome problem. Use two pointers to maintain current window, containing all characters we need in t string. And under the guaranteeing circumstances that our window contains all digits we want, we may shrink the window by moving the left pointer to right; when the window is no longer adequate, we move the right pointer forward to meet the requirement.

Listing 233: Solution

```
public class Solution {
    public String minWindow(String s, String t) {
        if (s.length() == 0 || t.length() == 0){
            return "";
        }
        int[] require = new int[256];
        boolean[] needLetter = new boolean[256];

        for (int i = 0 ; i < t.length() ; i ++){
            needLetter[t.charAt(i)] = true;
            require[t.charAt(i)] ++;
        }
        int count = t.length();
        int minLength = Integer.MAX_VALUE;
        String minStr = "";

        int i = 0;
        int j = -1;

        while (i < s.length() && j < s.length()){
            if (count > 0){
                j ++;
```

```
                if (j >= s.length()){
                    break;
                }
                require[s.charAt(j)] --;
                if (needLetter[s.charAt(j)] && require[s.charAt(j)] >= 0){
                    count --;
                }
            }else{
                int curLength = j - i + 1;
                if (curLength < minLength){
                    minLength = curLength;
                    minStr = s.substring(i, j + 1);
                }

                require[s.charAt(i)] ++;
                if (needLetter[s.charAt(i)] && require[s.charAt(i)] > 0){
                    count ++;
                }
                i ++;

            }
        }

        return minStr;
    }
}
```

Problem 178

78 Subsets

```
Given a set of distinct integers, nums,
 return all possible subsets.

Note:
Elements in a subset must be in non-descending order.
The solution set must not contain duplicate subsets.
For example,
If nums = [1,2,3], a solution is:

[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]

https://leetcode.com/problems/subsets/
```

Solution 1

Take [1,2,3] for example.

Initially, the result is empty, namely [[]].

Then, we add 1 to result, combine the result with former ones, that is [[],**[1]**]

Next, we add 2 to every list in result, also combine the new and old list together, [[],[1],**[2],[1,2]**]

Finally, it's 3's turn, we get [[],[1],[2],[1,2],**[3],[1,3],[2,3],[1,2,3]**] as our final result.

Listing 234: Solution 1

```
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        result.add(new ArrayList<Integer>());

        for (int num : nums){
            int resultLength = result.size();
            for (int idx = 0 ; idx < resultLength ; idx ++){
                List<Integer> newList = new ArrayList(result.get(idx));
                newList.add(num);
                // Collections.sort(newList);
                result.add(newList);
            }
        }

        return result;
    }
```

```
}
```

Solution 2

The second solution is a DFS method. Interesting since it utilized 2 same function call to simulate picking or not picking element at curIdx.

Listing 235: Solution 2

```
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();

    public void findSubsets(List<Integer> curList, int[] nums, int curIdx){
        if (curIdx == nums.length){
            result.add(new ArrayList(curList));
            return;
        }
        curList.add(nums[curIdx]);
        findSubsets(curList, nums, curIdx + 1);
        curList.remove(curList.size() - 1);
        findSubsets(curList, nums, curIdx + 1);
    }
    public List<List<Integer>> subsets(int[] nums) {
        Arrays.sort(nums);
        findSubsets(new ArrayList<Integer>(), nums, 0);

        return result;
    }
}
```

Problem 179

90 Subsets II

```
Given a collection of integers that might contain duplicates,
 nums, return all possible subsets.

Note:
Elements in a subset must be in non-descending order.
The solution set must not contain duplicate subsets.
For example,
If nums = [1,2,2], a solution is:

[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]

https://leetcode.com/problems/subsets-ii/
```

Solution 1

The train of thought is similar to Solution 1 in Problem "Subsets I"(Subsets II can also be solved using Solution 2). The key is on how to deal with the duplicates. Mention that when current element equals to last element, eg, 2 & 2 in [1,2,2], the new list generated are only from what has just been generated in last round.

Listing 236: Solution 1

```
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        if (nums.length == 0){
            return result;
        }
        Arrays.sort(nums);
        List<List<Integer>> generatedList = new ArrayList<List<Integer>>();
        result.add(new ArrayList<Integer>());
        for (int eleIdx = 0 ; eleIdx < nums.length ; eleIdx ++){
            int num = nums[eleIdx];
            if (eleIdx > 0 && nums[eleIdx] == nums[eleIdx - 1]){
                List<List<Integer>> newGeneratedList = new ArrayList<List<Integer>>();
                for (int formerIdx = 0 ; formerIdx < generatedList.size() ; formerIdx ++){
                    List<Integer> formerElement = new ArrayList(generatedList.get(formerIdx));
                    formerElement.add(num);
                    newGeneratedList.add(formerElement);
                }
                result.addAll(newGeneratedList);
                generatedList = newGeneratedList;
            }
```

Problem 179 continued on next page...

```
            else{
                generatedList.clear();
                for (int formerIdx = 0 ; formerIdx < result.size() ; formerIdx ++){
                    List<Integer> formerElement = new ArrayList(result.get(formerIdx));
                    formerElement.add(num);
                    generatedList.add(formerElement);
                }
                result.addAll(generatedList);
            }
        }

        return result;
    }
}
```

Solution 2

The backtracking method. Dealing with duplicates are also the key insight. The train of thought is somewhat like solution in "Combination Sum II".

Listing 237: Solution 2

```
public class Solution {
    List<List<Integer>> result = new ArrayList<List<Integer>>();
    public void findSubsets(List<Integer> curList, int[] nums, int curIdx){
        if (curIdx == nums.length){
            result.add(new ArrayList(curList));
            return;
        }
        curList.add(nums[curIdx]);
        findSubsets(curList, nums, curIdx + 1);
        curList.remove(curList.size() - 1);
        int diff = 1;
        while (curIdx + diff < nums.length
         && nums[curIdx + diff - 1] == nums[curIdx + diff]){
            diff ++;
        }
        findSubsets(curList, nums, curIdx + diff);
    }
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        findSubsets(new ArrayList<Integer>(), nums, 0);
        return result;
    }
}
```

Problem 180

303 Range Sum Query - Immutable

```
Given an integer array nums,
 find the sum of the elements between indices i and j (i <= j), inclusive.

Example:
Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
Note:
You may assume that the array does not change.
There are many calls to sumRange function.

https://leetcode.com/problems/range-sum-query-immutable/
```

Solution

Not a bad problem. Since the sumRange function may be called often, adding them every time we call the function sounds a bad idea. Use a sum array to store the sums for convenience. Mention the boundaries.

Listing 238: Solution

```java
public class NumArray {
    private int[] sums;
    public NumArray(int[] nums) {
        sums = new int[nums.length];
        int sum = 0;
        for (int idx = 0 ; idx < nums.length ; idx ++){
            sum += nums[idx];
            sums[idx] = sum;
        }
    }

    public int sumRange(int i, int j) {
        if (i == 0){
            return sums[j];
        }
        else{
            return sums[j] - sums[i - 1];
        }

    }
}


// Your NumArray object will be instantiated and called as such:
// NumArray numArray = new NumArray(nums);
// numArray.sumRange(0, 1);
// numArray.sumRange(1, 2);
```

Problem 181

307 Range Sum Query - Mutable

```
Given an integer array nums, find the sum
 of the elements between indices i and j (i <= j), inclusive.

The update(i, val) function modifies nums by
 updating the element at index i to val.
Example:
Given nums = [1, 3, 5]

sumRange(0, 2) -> 9
update(1, 2)
sumRange(0, 2) -> 8
Note:
The array is only modifiable by the update function.
You may assume the number of calls to update and
 sumRange function is distributed evenly.

https://leetcode.com/problems/range-sum-query-mutable/
```

Solution

Using solution in former problem "Range Sum Query - Immutable" will get TLE, so try some other solutions.

Make use of a freaking awesome data structure of niubility! Binary Index Tree(or Fenwick Tree)!

For more reference on index trees, see

http://www.doc88.com/p-9475113829694.html

http://blog.csdn.net/int64ago/article/details/7429868

https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-tre

Something need to be considered or mistakes I have made:

nums[0] is not in BIT since lowbit(0) == 0 will cause a infinite loop, so nums[0] is resolved seperately.

I forgot to update nums array but just update BIT, which brings me confusion on what is wrong with my solution.

Be aware of the boundaries.

Listing 239: Solution

```
public class NumArray {
    private int[] biTree;
    private int[] nums;

    private int lowbit(int x){
        return x & (-x);
    }
    public NumArray(int[] nums) {
        if (nums.length == 0){
            return;
        }
        this.nums = nums;
        biTree = new int[nums.length];
        biTree[0] = nums[0];
        for (int idx = 1 ; idx < nums.length ; idx ++){
```

Problem 181 continued on next page...

```
            for (int j = idx - lowbit(idx) + 1; j <= idx ; j ++){
                biTree[idx] += nums[j];
            }
        }
    }
    void update(int i, int val) {
        int diff = val - nums[i];
        nums[i] = val;
        while (i < nums.length){
            biTree[i] += diff;
            i += lowbit(i);
            if (i == 0){
                break;
            }
        }
    }

    private int getSum(int idx){
        int sum = 0;
        while (idx > 0){
            sum += biTree[idx];
            idx -= lowbit(idx);
        }
        sum += biTree[0];

        return sum;
    }
    public int sumRange(int i, int j) {
        if (i == 0){
            return getSum(j);
        }
        else{
            return getSum(j) - getSum(i - 1);
        }
    }
}


// Your NumArray object will be instantiated and called as such:
// NumArray numArray = new NumArray(nums);
// numArray.sumRange(0, 1);
// numArray.update(1, 10);
// numArray.sumRange(1, 2);
```

Problem 182

304 Range Sum Query 2D - Immutable

```
Given a 2D matrix 'matrix', find the sum of the elements
 inside the rectangle defined by its upper left corner
  (row1, col1) and lower right corner (row2, col2).

Range Sum Query 2D
The above rectangle (with the red border) is defined by
 (row1, col1) = (2, 1) and (row2, col2) = (4, 3), which contains sum = 8.

Example:
Given matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]
]

sumRegion(2, 1, 4, 3) -> 8
sumRegion(1, 1, 2, 2) -> 11
sumRegion(1, 2, 2, 4) -> 12
Note:
You may assume that the matrix does not change.
There are many calls to sumRegion function.
You may assume that row1    row2 and col1    col2.

https://leetcode.com/problems/range-sum-query-2d-immutable/
```

Solution

An easy dp problem but costs me half an hour and 10 WA or RE since I made a mistake on dealing with the array index. Reserve an buffer is a good choice.

Listing 240: Solution

```
public class NumMatrix {
    int[][] sums;

    public NumMatrix(int[][] matrix) {
        if (matrix == null || matrix.length == 0) {
            return;
        }
        sums = new int[matrix.length + 1][matrix[0].length + 1];
        sums[1][1] = matrix[0][0];
        for (int row = 1; row < matrix.length; row++) {
            sums[row + 1][1] += sums[row][1] + matrix[row][0];
        }
        for (int col = 1; col < matrix[0].length; col++) {
            sums[1][col + 1] += sums[1][col] + matrix[0][col];
        }

        for (int row = 1; row < matrix.length; row++) {
```

```
            for (int col = 1; col < matrix[0].length; col++) {
                sums[row + 1][col + 1] = sums[row][col + 1] + sums[row + 1][col]
                        - sums[row][col] + matrix[row][col];
            }
        }
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        return sums[row2 + 1][col2 + 1] - sums[row2 + 1][col1]
             - sums[row1][col2 + 1] + sums[row1][col1];
    }
}


// Your NumMatrix object will be instantiated and called as such:
// NumMatrix numMatrix = new NumMatrix(matrix);
// numMatrix.sumRegion(0, 1, 2, 3);
// numMatrix.sumRegion(1, 2, 3, 4);
```

Problem 183

198 House Robber

```
You are a professional robber planning to rob houses along a street.
 Each house has a certain amount of money stashed,
  the only constraint stopping you from robbing each of them
   is that adjacent houses have security system connected and
    it will automatically contact the police if two adjacent houses
     were broken into on the same night.

Given a list of non-negative integers representing
 the amount of money of each house,
  determine the maximum amount of money you can rob tonight
   without alerting the police.

https://leetcode.com/problems/house-robber/
```

Solution

A bit like finding the maximum increasing sequence, the dp function is exactly maxa[i-1], a[i-2] + nums[i], but take care of the initialization part(I made a mistake here), a[0] = nums[0], a[1] = Math.max(nums[0],nums[1]);

Listing 241: Solution

```
public class Solution {
    public int rob(int[] nums) {
        if (nums.length == 0){
            return 0;
        }
        if (nums.length == 1){
            return nums[0];
        }
        if (nums.length == 2){
            return Math.max(nums[0], nums[1]);
        }
        int[] sums = new int[nums.length];

        sums[0] = nums[0];
        sums[1] = Math.max(nums[0],nums[1]);
        for (int idx = 2; idx < nums.length ; idx ++){
            sums[idx] = Math.max(sums[idx - 2] + nums[idx], sums[idx - 1]);
        }
        return sums[nums.length - 1];
    }
}
```

Problem 184

213 House Robber II

```
Note: This is an extension of House Robber.

After robbing those houses on that street,
 the thief has found himself a new place for his thievery
  so that he will not get too much attention.
   This time, all houses at this place are arranged in a circle.
    That means the first house is the neighbor of the last one.
     Meanwhile, the security system for these houses remain
      the same as for those in the previous street.

Given a list of non-negative integers representing
 the amount of money of each house,
  determine the maximum amount of money you can rob tonight
   without alerting the police.

https://leetcode.com/problems/house-robber-ii/
```

Solution

A cyclic DP problem, we can separate the problem into 2 linear DP problems by assuming whether rob the first house(so that the last can't be robbed) or don't rob it.

Listing 242: Solution

```java
public class Solution {
    public int rob(int[] nums) {
        if (nums.length == 0)
            return 0;
        if (nums.length == 1)
            return nums[0];
        int[] sums = new int[nums.length];
        // not robbing the 1st house
        sums[0] = 0;
        sums[1] = nums[1];
        for (int idx = 2 ; idx < nums.length ; idx ++){
            sums[idx] = Math.max(sums[idx - 1], sums[idx - 2] + nums[idx]);
        }
        int max = sums[nums.length - 1];
        // robbing the 1st house
        sums[0] = nums[0];
        sums[1] = Math.max(nums[0],nums[1]);
        for (int idx = 2; idx < nums.length - 1 ; idx ++){
            sums[idx] = Math.max(sums[idx - 1], sums[idx - 2] + nums[idx]);
        }
        return Math.max(max, sums[nums.length - 2]);
    }
}
```

Problem 185

337 House Robber III

```
The thief has found himself a new place for his thievery again.
 There is only one entrance to this area, called the "root."
  Besides the root, each house has one and only one parent house.
   After a tour, the smart thief realized that
    "all houses in this place forms a binary tree".
     It will automatically contact the police if
      two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can
 rob tonight without alerting the police.

Example 1:
     3
    / \
   2   3
    \   \
     3   1
Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.
Example 2:
     3
    / \
   4   5
  / \   \
 1   3   1
Maximum amount of money the thief can rob = 4 + 5 = 9.

https://leetcode.com/problems/house-robber-iii/
```

Solution

When the thief see a house, he has 2 choices, rob it or don't rob it. When he decides to rob it, things are simple, he cannot rob the 2 children houses; but not robbing is a bit complicated, he need to re-decide when he meets the 2 children houses whether he should rob it.

Listing 243: Solution

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public int rob(TreeNode root) {
        if (root == null){
            return 0;
        }
        int exc = robExcludeRoot(root);
```

```
        int inc = robIncludeRoot(root);
        return Math.max(exc, inc);
    }

    private int robExcludeRoot(TreeNode root) {
        if (root == null){
            return 0;
        }
        return rob(root.left) + rob(root.right);
    }

    private int robIncludeRoot(TreeNode root){
        if (root == null){
            return 0;
        }
        return root.val + robExcludeRoot(root.left) + robExcludeRoot(root.right);
    }
}
```

Problem 186

10 Regular Expression Matching

```
Implement regular expression matching with support for '.' and '*'.

'.' Matches any single character.
'*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") -> false
isMatch("aa","aa") -> true
isMatch("aaa","aa") -> false
isMatch("aa", "a*") -> true
isMatch("aa", ".*") -> true
isMatch("ab", ".*") -> true
isMatch("aab", "c*a*b") -> true

https://leetcode.com/problems/regular-expression-matching/
```

Solution

A very complicated backtracking problem, takes me 1 and a half hour to finish it(and even in an IDE!!!)..... Both this problem and next problem needs to be supplemented with missing DP or recursion solutions. See `http://www.xuebuyuan.com/1936978.html`, good illustration.

Listing 244: Solution 1.1

```
public class Solution {
    private boolean isMatch(String s, int idxS, String p, int idxP) {
        if (idxS < s.length() && idxP >= p.length()) {
            return false;
        }
        if (idxP == p.length() && idxS == s.length()) {
            return true;
        }

        if ((idxP < p.length() - 1) && p.charAt(idxP + 1) == '*') {
            if (idxS < s.length()
                    && (p.charAt(idxP) == s.charAt(idxS) || (p.charAt(idxP) == '.'))) {
                return isMatch(s, idxS + 1, p, idxP + 2)
                        || isMatch(s, idxS + 1, p, idxP)
                        || isMatch(s, idxS, p, idxP + 2);
            } else {
                return isMatch(s, idxS, p, idxP + 2);
            }
        }
        if (idxS < s.length()
                && (p.charAt(idxP) == '.' || s.charAt(idxS) == p.charAt(idxP))) {
```

```
            return isMatch(s, idxS + 1, p, idxP + 1);
        }
        return false;
    }

    public boolean isMatch(String s, String p) {
        // if (s.length() == 0)
        return isMatch(s, 0, p, 0);
    }
}
```

Append a rather simple code...

Listing 245: Solution 1.2

```
public class Solution {
    public boolean isMatch(String s, String p) {
        for(int i = 0; i < p.length(); s = s.substring(1)) {
            char c = p.charAt(i);
            if(i + 1 >= p.length() || p.charAt(i + 1) != '*')
                i++;
            else if(isMatch(s, p.substring(i + 2)))
                return true;

            if(s.isEmpty() || (c != '.' && c != s.charAt(0)))
                return false;
        }

        return s.isEmpty();
    }
}
```

Problem 187

44 Wildcard Matching

```
Implement wildcard pattern matching with support for '?' and '*'.

'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).

The function prototype should be:
bool isMatch(const char *s, const char *p)

Some examples:
isMatch("aa","a") -> false
isMatch("aa","aa") -> true
isMatch("aaa","aa") -> false
isMatch("aa", "*") -> true
isMatch("aa", "a*") -> true
isMatch("ab", "?*") -> true
isMatch("aab", "c*a*b")    false

https://leetcode.com/problems/wildcard-matching/
```

Solution

Displaying the DP solution.

Listing 246: Solution

```
public class Solution {
    public boolean isMatch(String s, String p) {
        if (s.length() == 0){
            int idx = 0;
            while (idx < p.length() && p.charAt(idx) == '*'){
                idx ++;
            }

            if (idx == p.length()){
                return true;
            }

            return false;
        }

        boolean dp[][] = new boolean[s.length() + 1][p.length() + 1];

        dp[0][0] = true;
        for (int j = 1 ; j <= p.length(); j ++ ){
            for (int i = 1 ; i <= s.length() ; i++){
                if (dp[i - 1][j - 1] && (s.charAt(i - 1) == p.charAt(j - 1)
                 || p.charAt(j - 1) == '?')){
                    dp[i][j] = true;
                }
```

```
25
                if (p.charAt(j - 1) == '*'){
                    if (dp[i - 1][j - 1]){
                        for (int k = i - 1 ; k <= s.length() ; k ++){
                            dp[k][j] = true;
30                      }
                        break;
                    }
                    else if (dp[i][j - 1]){
                        for (int k = i; k <= s.length(); k ++){
35                          dp[k][j] = true;
                        }
                        break;
                    }
                }
40          }
        }

        return dp[s.length()][p.length()];
    }
45  }
```

Problem 188

73 Set Matrix Zeroes

```
Given a m x n matrix, if an element is 0,
 set its entire row and column to 0. Do it in place.

Follow up:
Did you use extra space?
A straight forward solution using O(mn) space is probably a bad idea.
A simple improvement uses O(m + n) space, but still not the best solution.
Could you devise a constant space solution?

https://leetcode.com/problems/set-matrix-zeroes/
```

Solution

Solution using $O(mn)space$ is too trivial. First show the result using $O(mn)space$, then the $O(1)space$ solution.

Listing 247: Solution 1.1

```
public class Solution {
    public void setZeroes(int[][] matrix) {
        int rowLength = matrix.length;
        if (rowLength == 0){
            return ;
        }
        int colLength = matrix[0].length;
        boolean rowZero[] = new boolean[rowLength];
        boolean colZero[] = new boolean[colLength];

        for (int rowIdx = 0 ; rowIdx < rowLength ; rowIdx ++){
            for (int colIdx = 0 ; colIdx < colLength; colIdx ++){
                if(matrix[rowIdx][colIdx] == 0){
                    rowZero[rowIdx] = true;
                    colZero[colIdx] = true;
                }
            }
        }

        for (int rowIdx = 0 ; rowIdx < rowLength ; rowIdx ++){
            if (rowZero[rowIdx]){
                for (int colIdx = 0 ; colIdx < colLength ; colIdx ++){
                    matrix[rowIdx][colIdx] = 0;
                }
            }
        }

        for (int colIdx = 0 ; colIdx < colLength ; colIdx ++){
            if (colZero[colIdx]){
                for (int rowIdx = 0 ; rowIdx < rowLength ; rowIdx ++){
                    matrix[rowIdx][colIdx] = 0;
                }
            }
```

Problem 188 continued on next page...

```
        }
    }
}
```

The $O(1)$*space* solution. Use the 1st(index 0) row and 1st column to mark 0 when there is a grid valued 0 with the same rowIdx or colIdx. Attention! We should deal with the 1st row and column separately, or it will be influenced and make all grids 0!

Listing 248: Solution 1.2

```
public class Solution {
    public void setZeroes(int[][] matrix) {
        int m = matrix.length;
        if (m == 0){
            return;
        }
        int n = matrix[0].length;
        boolean zeroInFirstRow = false;
        boolean zeroInFirstCol = false;
        for (int rowIdx = 0 ; rowIdx < m ; rowIdx ++){
            if (matrix[rowIdx][0] == 0){
                zeroInFirstCol = true;
                break;
            }
        }
        for (int colIdx = 0 ; colIdx < n ; colIdx ++){
            if (matrix[0][colIdx] == 0){
                zeroInFirstRow = true;
                break;
            }
        }

        for (int rowIdx = 0 ; rowIdx < m ; rowIdx ++){
            for (int colIdx = 0 ; colIdx < n ; colIdx ++){
                if (matrix[rowIdx][colIdx] == 0){
                    matrix[rowIdx][0] = 0;
                    matrix[0][colIdx] = 0;
                }
            }
        }

        for (int rowIdx = 1 ; rowIdx < m ; rowIdx ++){
            for (int colIdx = 1 ; colIdx < n ; colIdx ++){
                if (matrix[rowIdx][0] == 0 || matrix[0][colIdx] == 0){
                    matrix[rowIdx][colIdx] = 0;
                }
            }
        }

        if (zeroInFirstRow){
            for (int colIdx = 0 ; colIdx < n ; colIdx ++){
                matrix[0][colIdx] = 0;
            }
```

```
            }
            if (zeroInFirstCol){
                for (int rowIdx = 0 ; rowIdx < m ; rowIdx ++){
                    matrix[rowIdx][0] = 0;
                }
            }

        }
}
```

Problem 189

329 Longest Increasing Path in a Matrix

```
Given an integer matrix,
 find the length of the longest increasing path.

From each cell, you can either move to four directions:
 left, right, up or down.
  You may NOT move diagonally or move outside of the boundary
   (i.e. wrap-around is not allowed).

Example 1:

nums = [
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
Return 4
The longest increasing path is [1, 2, 6, 9].

Example 2:

nums = [
  [3,4,5],
  [3,2,6],
  [2,2,1]
]
Return 4
The longest increasing path is [3, 4, 5, 6].
 Moving diagonally is not allowed.

https://leetcode.com/problems/longest-increasing-path-in-a-matrix/
```

Solution

A DP(?), DFS and memoization problem. Not that difficult.

Listing 249: Solution

```
public class Solution {
    int[][] directions = {{0,1},{0,-1},{1,0},{-1,0}};

    public int findLongestPath(int[][] matrix, int[][] pathLen ,  int x , int y){
        if (pathLen[x][y] != 0){
            return pathLen[x][y];
        }
        pathLen[x][y] = 1;
        for (int dirIdx = 0 ; dirIdx < directions.length ; dirIdx ++){
            int i = x + directions[dirIdx][0];
            int j = y + directions[dirIdx][1];

            if (i < 0 || j < 0 || i >= matrix.length
                || j >= matrix[0].length || matrix[i][j] <= matrix[x][y]){
```

```
                continue;
            }
            else{
                pathLen[x][y] = Math.max(pathLen[x][y]
                    , findLongestPath(matrix, pathLen, i, j) + 1);
            }
        }

        return pathLen[x][y];
    }

    public int longestIncreasingPath(int[][] matrix) {
        int m = matrix.length;
        if (m == 0){
            return 0;
        }
        int n = matrix[0].length;
        int[][] pathLen = new int[m][n];

        int max = Integer.MIN_VALUE;

        for (int i = 0 ; i < m ; i ++){
            for (int j = 0 ; j < n ; j ++){
                max = Math.max(max, findLongestPath(matrix, pathLen, i, j));
            }
        }

        return max;
    }
}
```

Problem 190

93 Restore IP Addresses

```
Given a string containing only digits,
 restore it by returning all possible valid IP address combinations.

For example:
Given "25525511135",

return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)

https://leetcode.com/problems/restore-ip-addresses/
```

Solution

Honestly, this is not such a difficult question, but there are so many edge cases in this problem, which make us busy debugging when we meet a new edge case.

Take care of those:

(1) numbers starts with '0' except for 0 itself;
(2) length out of bounds
(3) more than 4 numbers generated
(4) number more than 255
(5) number with the length of more than 3

Listing 250: Solution

```
public class Solution {
    List<String> result = new ArrayList<String>();

    private void findIpAddresses(String s, int startIdx, List<Integer> curList){
        if (startIdx > s.length()){
            return;
        }
        if (curList.size() >= 4 && startIdx < s.length()){
            return;
        }
        if (curList.size() == 4 && startIdx == s.length()){
            result.add(formIpString(curList));
            return;
        }

        for (int endIdx = startIdx + 1 ;
                 endIdx <= Math.min(s.length(), startIdx + 3) ; endIdx ++){
            String newNum = s.substring(startIdx, endIdx);
            if (endIdx > startIdx + 1 && s.charAt(startIdx) == '0'){
                return;
            }
            if (Integer.parseInt(newNum) > 255){
                return;
            }
            curList.add(Integer.parseInt(newNum));
            findIpAddresses(s, endIdx, curList);
            curList.remove(curList.size() - 1);
```

Problem 190 continued on next page...

```
            }
        }

    private String formIpString(List<Integer> list){
        return list.get(0) + "." + list.get(1) + "." +
            list.get(2) + "." + list.get(3);
    }

    public List<String> restoreIpAddresses(String s) {
        findIpAddresses(s, 0, new ArrayList<Integer>());
        return result;
    }
}
```

Problem 191

131 Palindrome Partitioning

```
Given a string s, partition s such that
 every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given s = "aab",
Return

  [
    ["aa","b"],
    ["a","a","b"]
  ]

https://leetcode.com/problems/palindrome-partitioning/
```

Solution

A not so difficult backtracking problem, add string to list if the string is palindrome.

Listing 251: Solution

```
public class Solution {
    List<List<String>> result = new ArrayList<List<String>>();

    private boolean isPalindrome(String str){
        int start = 0 , end = str.length() - 1;
        while (start < end){
            if (str.charAt(start) != str.charAt(end)){
                return false;
            }
            start ++;
            end --;
        }
        return true;
    }

    private void findPalindrome(String str, int startIdx, List<String> curList){
        if (startIdx == str.length()){
            result.add(new ArrayList(curList));
            return;
        }
        for (int endIdx = startIdx + 1 ; endIdx <= str.length() ; endIdx ++){
            String newStr = str.substring(startIdx, endIdx);
            if (isPalindrome(newStr)){
                curList.add(newStr);
                findPalindrome(str, endIdx, curList);
                curList.remove(curList.size() - 1);
            }
        }
    }

```

```
    public List<List<String>> partition(String s) {
        findPalindrome(s, 0, new ArrayList<String>());
        return result;
    }
}
```

Problem 192

132 Palindrome Partitioning II

```
Given a string s, partition s such that every substring
 of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example, given s = "aab",
Return 1 since the palindrome partitioning ["aa","b"]
 could be produced using 1 cut.

https://leetcode.com/problems/palindrome-partitioning-ii/
```

Solution

Solution from previous Problem "Palindrome Partitioning" will get TLE with long sequence with so many palindromes within it.(Even when I changed the train of thought to find the longest palindrome first and do the pruning when result list size is larger than min)

So DP will be the alternative.

Listing 252: Solution

```
public class Solution {
    public int minCut(String s) {
        boolean[][] isPal = new boolean[s.length()][s.length()];
        int cut[] = new int[s.length()];
        isPal[0][0] = true;
        cut[0] = 0;
        for (int i = 0 ; i < s.length() ; i++){
            cut[i] = i;
            for (int j = 0 ; j <= i ; j ++){
                if (s.charAt(i) == s.charAt(j)
                 && (i - j <= 1 || isPal[j + 1][i - 1])){
                    isPal[j][i] = true;
                    if (j == 0)
                        // no need to cut
                        cut[i] = 0;
                    else
                        cut[i] = Math.min(cut[i] , cut[j - 1] + 1);
                }

            }
        }

        return cut[s.length() - 1];
    }
}
```

Problem 193

38 Count and Say

```
The count-and-say sequence is the sequence of integers
 beginning as follows:
1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.
Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string.

https://leetcode.com/problems/count-and-say/
```

Solution

Attention for the easy problems, not as easy as I thought.

Listing 253: Solution

```
public class Solution {
    private String generateNumber(String formerStr){
        String newStr = "";
        char compare = formerStr.charAt(0);
        int dupCount = 1;
        for (int idx = 1 ; idx < formerStr.length() ; idx ++){
            while (idx < formerStr.length() && formerStr.charAt(idx) == compare){
                dupCount ++;
                idx ++;
            }
            newStr = newStr + dupCount + compare;
            if (idx < formerStr.length()){
                compare = formerStr.charAt(idx);
                dupCount = 1;
            }
        }
        if(dupCount == 1)
            newStr += "1" + compare;
        return newStr;
    }
    public String countAndSay(int n) {
        String initialStr = "1";
        while (n > 1){
            initialStr = generateNumber(initialStr);
            n --;
        }
        return initialStr;
    }
}
```

Problem 194

324 Wiggle Sort II

```
Given an unsorted array nums,
 reorder it such that nums[0] < nums[1] > nums[2] < nums[3]....

Example:
(1) Given nums = [1, 5, 1, 1, 6, 4],
     one possible answer is [1, 4, 1, 5, 1, 6].
(2) Given nums = [1, 3, 2, 2, 3, 1],
     one possible answer is [2, 3, 1, 3, 1, 2].

Note:
You may assume all input has valid answer.

Follow Up:
Can you do it in O(n) time and/or in-place with O(1) extra space?

https://leetcode.com/problems/wiggle-sort-ii/
```

Solution

Using wiggleIndex to re-index the array is the key to solving this. the formula is $(2 * i + 1)\%(n|1)$. A[0,1,2,3,4,4] will be mapped to a[1,3,5,0,2,4], respectively.

Listing 254: Solution

```
public class Solution {
    private int wiggleIndex(int idx , int length){
        return (2 * idx + 1) % (length | 1);
    }

    private int partitionMedian(int[] nums){
        int low = 0 , high = nums.length - 1, targetIdx = nums.length / 2;

        while(true){
            int pivotIdx = -1;
            int pivot = nums[low];
            int start = low, end = high;

            while (low < high){
                while (low < high && nums[high] <= pivot){
                    high --;
                }
                nums[low] = nums[high];
                while (low < high && nums[low] >= pivot){
                    low ++;
                }
                nums[high] = nums[low];
            }
            if (low == high){
                pivotIdx = low;
            }
            nums[pivotIdx] = pivot;
```

```
            if (pivotIdx == targetIdx){
                return pivot;
            }

            if (pivotIdx > targetIdx){
                high = pivotIdx - 1;
                low = start;
            }
            else{
                low = pivotIdx + 1;
                high = end;
            }
        }
    }

    public void wiggleSort(int[] nums) {
        int median = partitionMedian(nums);
        int current = 0, low = nums.length - 1, high = 0;
        while (current <= low){
            if (nums[wiggleIndex(current, nums.length)] == median){
                current ++;
            }
            else if (nums[wiggleIndex(current, nums.length)] > median){
                int swap = nums[wiggleIndex(current, nums.length)];
                nums[wiggleIndex(current, nums.length)]
                     = nums[wiggleIndex(high, nums.length)];
                nums[wiggleIndex(high, nums.length)] = swap;
                high ++;
                current ++;
            }
            else{
                int swap = nums[wiggleIndex(current, nums.length)];
                nums[wiggleIndex(current, nums.length)]
                     = nums[wiggleIndex(low, nums.length)];
                nums[wiggleIndex(low, nums.length)] = swap;
                low--;
            }
        }
    }
}
```

Problem 195

289 Game of Life

```
According to the Wikipedia's article:
 "The Game of Life, also known simply as Life,
  is a cellular automaton devised by the British mathematician
   John Horton Conway in 1970."

Given a board with m by n cells,
 each cell has an initial state live (1) or dead (0).
  Each cell interacts with its eight neighbors
   (horizontal, vertical, diagonal) using the following four rules
    (taken from the above Wikipedia article):

Any live cell with fewer than two live neighbors dies,
 as if caused by under-population.
Any live cell with two or three live neighbors lives on to the next generation.
Any live cell with more than three live neighbors dies,
 as if by over-population..
Any dead cell with exactly three live neighbors becomes a live cell,
 as if by reproduction.
Write a function to compute the next state
 (after one update) of the board given its current state.

Follow up:
Could you solve it in-place?
 Remember that the board needs to be updated at the same time:
  You cannot update some cells first and then use
   their updated values to update other cells.
In this question, we represent the board using a 2D array.
 In principle, the board is infinite,
  which would cause problems when the active area
   encroaches the border of the array.
    How would you address these problems?

https://leetcode.com/problems/game-of-life/
```

Solution

Bit manipulation is the first thought which comes into my mind, use 1st digit as next state and 2nd digit for current state.

Listing 255: Solution

```
public class Solution {
    int[][] directions = {{-1,0},{0,-1},{-1,-1},{1,0}
                         ,{0,1},{1,1},{-1,1},{1,-1}};
    public void gameOfLife(int[][] board) {
        if (board.length == 0){
            return;
        }
        for(int i = 0 ; i < board.length ; i++){
            for (int j = 0 ; j < board[0].length ; j++){
                int liveNeighbors = 0;
```

```
                for (int dirIdx = 0 ; dirIdx < 8 ; dirIdx ++){
                    int x = i + directions[dirIdx][0];
                    int y = j + directions[dirIdx][1];
                    if (x < 0 || y < 0 || x >= board.length
                     || y >= board[0].length){
                        continue;
                    }
                    liveNeighbors += board[x][y] & 1;
                }

                if (liveNeighbors < 2 || liveNeighbors > 3){
                    board[i][j] = (0 << 1) + board[i][j];
                } else if (liveNeighbors == 2){
                    board[i][j] = (board[i][j] << 1) + board[i][j];
                } else if (liveNeighbors == 3){
                    board[i][j] = (1 << 1) + board[i][j];
                }
            }
        }

        for (int i = 0 ; i < board.length ; i ++){
            for (int j = 0 ; j < board[0].length ; j++){
                board[i][j] = (board[i][j] >>> 1);
            }
        }
    }
}
```

Problem 196

165 Compare Version Numbers

```
Compare two version numbers version1 and version2.
If version1 > version2 return 1, if version1 < version2 return -1,
 otherwise return 0.

You may assume that the version strings are
 non-empty and contain only digits and the . character.
The . character does not represent
 a decimal point and is used to separate number sequences.
For instance, 2.5 is not "two and a half" or
 "half way to version three",
  it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

0.1 < 1.1 < 1.2 < 13.37

https://leetcode.com/problems/compare-version-numbers/
```

Solution

Though simple, do not miss the edge cases when '1.0.0' equals '1'

Listing 256: Solution

```java
public class Solution {
    public int compareVersion(String version1, String version2) {
        int idx1 = 0;
        int idx2 = 0;

        while (idx1 < version1.length() && idx2 < version2.length()){
            int num1 = 0, num2 = 0;
            while (idx1 < version1.length() && version1.charAt(idx1) != '.'){
                num1 = num1 * 10 + (version1.charAt(idx1) - '0');
                idx1 ++;
            }
            while (idx2 < version2.length() && version2.charAt(idx2) != '.'){
                num2 = num2 * 10 + (version2.charAt(idx2) - '0');
                idx2 ++;
            }

            if (num1 > num2){
                return 1;
            }
            else if (num2 > num1){
                return -1;
            }

            idx1++;
            idx2++;
        }
        if (idx1 < version1.length()){
```

```
            while (idx1 < version1.length()
                && (version1.charAt(idx1) == '.'
                  || version1.charAt(idx1) == '0')){
                idx1 ++;
            }
            if (idx1 < version1.length()){
                return 1;
            }
        }
        else if (idx2 < version2.length()){
            while (idx2 < version2.length()
             && (version2.charAt(idx2) == '.'
               || version2.charAt(idx2) == '0')){
                idx2 ++;
            }
            if (idx2 < version2.length()){
                return -1;
            }
        }

        return 0;
    }
}
```

Problem 197

58 Length of Last Word

```
Given a string s consists of upper/lower-case alphabets
 and empty space characters ' ',
  return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence
 consists of non-space characters only.

For example,
Given s = "Hello World",
return 5.

https://leetcode.com/problems/length-of-last-word/
```

Solution

Counting from backward will benefit a lot.

Listing 257: Solution

```
public class Solution {
    public int lengthOfLastWord(String s) {
        int len = s.length();
        int idx = len - 1;
        int count = 0;
        while (idx >= 0){
            // skip spaces in the end
            while (idx >= 0 && s.charAt(idx) == ' '){
                idx --;
            }
            if (idx < 0){
                return 0;
            }
            while (idx >= 0 && s.charAt(idx) != ' '){
                count ++;
                idx --;
            }
            break;
        }
        return count;
    }
}
```

Problem 198

118 Pascal's Triangle

```
Given numRows, generate the first numRows of Pascal's triangle.

For example, given numRows = 5,
Return

[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]

https://leetcode.com/problems/pascals-triangle/
```

Solution

Such an easy problem.

Listing 258: Solution

```
public class Solution {
    public List<List<Integer>> generate(int numRows) {
        if (numRows == 0){
            return new ArrayList<List<Integer>>();
        }
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        List<Integer> oldList = new ArrayList<Integer>();
        if (numRows != 0){
            oldList.add(1);
        }
        result.add(new ArrayList<Integer>(oldList));
        for (int i = 0; i < numRows - 1 ; i ++){
            List<Integer> newList = new ArrayList<Integer>();
            for (int newIdx = 0 ; newIdx < oldList.size() + 1 ; newIdx ++){
                if (newIdx == 0 || newIdx == oldList.size()){
                    newList.add(1);
                }
                else{
                    newList.add(oldList.get(newIdx - 1) + oldList.get(newIdx));
                }
            }

            result.add(new ArrayList<Integer>(newList));
            oldList = newList;
        }

        return result;
    }
}
```

Problem 199

119 Pascal's Triangle II

```
Given an index k, return the kth row of the Pascal's triangle.

For example, given k = 3,
Return [1,3,3,1].

Note:
Could you optimize your algorithm to use only O(k) extra space?

https://leetcode.com/problems/pascals-triangle-ii/
```

Solution

Such an easy problem, too.

Listing 259: Solution

```
public class Solution {
    public List<Integer> getRow(int rowIndex) {
        List<Integer> list = new ArrayList<Integer>(rowIndex + 1);
        for (int i = 0 ; i <= rowIndex ; i++){
            list.add(0);
        }
        for (int i = 0 ; i <= rowIndex ; i ++){
            list.set(i, 1);
            for (int j = i - 1 ; j > 0 ; j --){
                list.set(j, list.get(j) + list.get(j - 1));
            }
        }
        return list;
    }
}
```

Problem 200

6 ZigZag Conversion

```
The string "PAYPALISHIRING" is written in a zigzag pattern
 on a given number of rows like this:
  (you may want to display this pattern
   in a fixed font for better legibility)

P   A   H   N
A P L S I I G
Y   I   R
And then read line by line: "PAHNAPLSIIGYIR"
Write the code that will take a string and
 make this conversion given a number of rows:

string convert(string text, int nRows);
convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

https://leetcode.com/problems/zigzag-conversion/
```

Solution

Such an easy problem, three, find the rule and the problem can be solved easily.

Listing 260: Solution

```java
public class Solution {
    public String convert(String s, int numRows) {
        String result = "";
        if (s.length() == 0 || numRows <= 1){
            return s;
        }
        for (int row = 0 ; row < numRows ; row ++){
            for (int col = row ; col < s.length() ; col += (numRows - 1) * 2){
                result += s.charAt(col);

                if (row != 0 && row != numRows - 1){
                    // except for 1st and last row
                    int right = 2 * row;
                    int left = 2 * (numRows - 1) - right;
                    if (col + left < s.length()){
                        result += s.charAt(col + left);
                    }
                }
            }
        }

        return result;
    }
}
```

Problem 201

273 Integer to English Words

```
Convert a non-negative integer to its english words representation.
 Given input is guaranteed to be less than 231 - 1.

For example,
123 -> "One Hundred Twenty Three"
12345 -> "Twelve Thousand Three Hundred Forty Five"
1234567 ->
 "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
Hint:

Did you see a pattern in dividing the number into chunk of words? For example,
 123 and 123000.
Group the number by thousands (3 digits).
 You can write a helper function that takes a number
  less than 1000 and convert just that chunk to words.
There are many edge cases. What are some good test cases?
 Does your code work with input such as 0? Or 1000010?
  (middle chunk is zero and should not be printed out)

https://leetcode.com/problems/integer-to-english-words/
```

Solution

Hint is enough for solving the problem, honestly. But, never enough for all the edge cases(especially when involved with spaces).

Listing 261: Solution

```
public class Solution {
    String[] underTwenty = { "", "One", "Two", "Three", "Four", "Five", "Six",
            "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen",
            "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen",
            "Nineteen" };
    String[] productTen = { "","","Twenty", "Thirty", "Forty", "Fifty", "Sixty",
            "Seventy", "Eighty", "Ninety" };
    String[] thousands = {"",  "Thousand", "Million", "Billion" };

    public String numberToWords(int num) {
        String result = "";
        int cur = num;
        int thousandCnt = 0;
        if (num == 0){
            return "Zero";
        }
        List<String> numbers = new ArrayList<String>();
        while (cur > 0){
            int qou = cur / 1000;
            int rem = cur % 1000;

            numbers.add(transform(rem));
            if (qou != 0){
```

```
                thousandCnt ++;
            }
            cur = qou;
        }
        for (int i = thousandCnt ; i >= 0 ; i --){
            String part = numbers.get(i);
            if (part.trim().length() != 0){
                result = result + part + thousands[i] + " ";
            }
        }

        int endIdx = result.length() - 1;
        while (result.charAt(endIdx) == ' '){
            endIdx --;
        }
        result = result.substring(0, endIdx + 1);
        return result;
    }
    private String transform(int num) {
        String result = "";
        int hun = num / 100;
        int tens = num % 100;
        int ten = tens / 10;
        int one = num % 10;
        if (hun > 0) {
            result += underTwenty[hun] + " Hundred ";
        }
        if (tens < 20 && tens != 0) {
            result += underTwenty[tens] + " ";
        } else if (tens != 0){
            result += productTen[ten] + " " +
                 ((one > 0) ? underTwenty[one] + " " : "");
        }

        return result;
    }
}
```

Problem 202

310 Minimum Height Trees

```
For a undirected graph with tree characteristics,
 we can choose any node as the root.
  The result graph is then a rooted tree.
   Among all possible rooted trees,
   those with minimum height are called minimum height trees (MHTs).
    Given such a graph,
     write a function to find all the MHTs and return a list of their root labels.

Format
The graph contains n nodes which are labeled from 0 to n - 1.
You will be given the number n and a list of undirected edges
 (each edge is a pair of labels).

You can assume that no duplicate edges will appear in edges.
 Since all edges are undirected, [0, 1] is the same as [1, 0]
  and thus will not appear together in edges.

Example 1:

Given n = 4, edges = [[1, 0], [1, 2], [1, 3]]

        0
        |
        1
       / \
      2   3
return [1]

Example 2:

Given n = 6, edges = [[0, 3], [1, 3], [2, 3], [4, 3], [5, 4]]

     0  1  2
      \ | /
        3
        |
        4
        |
        5
return [3, 4]

Hint:

How many MHTs can a graph have at most?
Note:

(1) According to the definition of tree on Wikipedia:
 ``a tree is an undirected graph in which any two vertices
  are connected by exactly one path. In other words,
```

```
    any connected graph without simple cycles is a tree.''

(2) The height of a rooted tree is the number of edges
 on the longest downward path between the root and a leaf.

https://leetcode.com/problems/minimum-height-trees/
```

Solution

The idea is like topological sort, since this is an undirected graph, find the node with degree of 1, remove them from the graph, until 1 or 2 nodes left.

Back news is: I used 'while (remaining> 2)' to check if I get the result, which means I'm not totally understanding it.

Take care when n == 1.

Listing 262: Solution

```
public class Solution {
    public List<Integer> findMinHeightTrees(int n, int[][] edges) {
        List<Integer> result = new ArrayList<Integer>();
        if (n == 1){
            result.add(0);
            return result;
        }

        int degree[] = new int[n];
        List[] adjacency = new ArrayList[n];
        for (int i = 0 ; i < n ; i ++){
            adjacency[i] = new ArrayList<Integer>();
        }

        for (int[] edge : edges){
            int node1 = edge[0];
            int node2 = edge[1];
            degree[node1] ++;
            degree[node2] ++;
            adjacency[node1].add(node2);
            adjacency[node2].add(node1);
        }

        Queue<Integer> leaves = new LinkedList<Integer>();
        for (int i = 0 ; i < n ; i ++){
            if (degree[i] == 1){
                leaves.offer(i);
            }
        }

        int remaining = n;
        while (remaining > 2){
            remaining -= leaves.size();
            List<Integer> leavesList = new ArrayList<Integer>();
            while (!leaves.isEmpty()){
                leavesList.add(leaves.poll());
            }
            for (int leafIdx : leavesList){
```

```
                List<Integer> neighbors = adjacency[leafIdx];
                for (int neighborIdx : neighbors){
                    degree[neighborIdx] --;
                    if (degree[neighborIdx] == 1){
                        leaves.offer(neighborIdx);
                    }
                }
            }

        }
        while (!leaves.isEmpty()){
            result.add(leaves.poll());
        }
        return result;
    }
}
```

Problem 203

300 Longest Increasing Subsequence

```
Given an unsorted array of integers,
 find the length of longest increasing subsequence.

For example,
Given [10, 9, 2, 5, 3, 7, 101, 18],
The longest increasing subsequence is [2, 3, 7, 101],
 therefore the length is 4.
  Note that there may be more than one LIS combination,
   it is only necessary for you to return the length.

Your algorithm should run in O(n2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

https://leetcode.com/problems/longest-increasing-subsequence/
```

Solution

As for the $O(n^2)$ solution, it is an easy DP problem, start of LCS(Longest Common Substring/Subsequence).

However, optimizing it to $O(nlogn)$ is tricky.

First, the $O(n^2)$ solution.

Listing 263: Solution 1.1

```
public class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length == 0 || nums.length == 1){
            return nums.length;
        }
        int dp[] = new int[nums.length];
        int max = 0;
        for (int idx = 0 ; idx < nums.length ; idx++){
            dp[idx] = 1;
            for (int formerIdx = 0 ; formerIdx < idx ; formerIdx ++){
                if (nums[idx] > nums[formerIdx]){
                    dp[idx] = Math.max(dp[idx], dp[formerIdx] + 1);
                }
            }
            max = Math.max(max, dp[idx]);
        }

        return max;
    }
}
```

Then the $O(nlogn)$ solution. The key insight is to find the $O(logn)$ factor. Below are train of thought from Discuss module:(or refer to this blog:)

https://segmentfault.com/a/1190000003819886

Runtime: To get an O(n log n) runtime, I'm going to create a second list S. (Stick with me for now – I'll get rid of it in a minute to get O(1) space.) I'll do a single pass through nums, and as I look at each element:

The length of S will be equal to the length of the longer subsequence I've found to that point.
The last element of S will be the last element of that subsequence. (However, the earlier elements may no longer be part of that sequence – S is not actually the subsequence itself.)
At the end, the length of S will be our solution.

S will be sorted at all times. Each new element is inserted into S, replacing the smallest element in S that is not smaller than it (which we can find with a binary search). If that element is larger than the last element of S, then we extend S by one – maintaining both properties.

For example, if
nums = [5,6,7,1,2,8,3,4,0,5,9]
then after we prcoess the 7:
S = [5,6,7]
after we process the 2:
S = [1,2,7]
after we process the 8:
S = [1,2,7,8]
Then we process the 3:
S = [1,2,3,8]
We process the 4:
S = [1,2,3,4]
and now the next three elements:
S = [0,2,3,4,5,9]
S is not the actual subsequence, but it is the right length (end ends in the right number).

We are making n passes, and doing a binary search each time. So $O(nlogn)$ time.

Space: Assuming we are allowed to destroy the list, we don't need S. Since S will never be larger then the number of elements we have looked at, and we only need to look at each element once, we can just use the beginning of nums for S (keeping track of the size of "S" in a separate variable.

Listing 264: Solution 1.2

```
public class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length == 0 || nums.length == 1){
            return nums.length;
        }
        int[] tails = new int[nums.length];
        int len = 0;
        tails[len ++] = nums[0];

        for (int idx = 1 ; idx < nums.length ; idx ++){
            if (nums[idx] < tails[0]){
                tails[0] = nums[idx];
            } else if (nums[idx] > tails[len - 1]){
                tails[len ++] = nums[idx];
            } else{
                int fitPos = binarySearch(tails, nums[idx], len);
                tails[fitPos] = nums[idx];
            }
        }

        return len;
```

```
    }

    private int binarySearch(int[] tails, int target, int curLen){
        int start = 0 , end  = curLen - 1;
        while (start < end){
            int mid = (end - start) / 2 + start;
            if (tails[mid] == target){
                return mid;
            }
            else if (tails[mid] > target){
                end = mid;
            }
            else{
                start = mid + 1;
            }
        }
        if (start == end){
            return start;
        }
        else{
            return (tails[start] > target) ? start : end;
        }
    }
}
```

Moreover, if we want to recover the actual subsequence, we may need to do some extra work. My solution is to maintain a LinkedList for each position, when there is a element bigger than former ones coming, add -1 to its head, the number of -1 depends on the maximum depth of former linked list. See code in listing 265

Listing 265: Solution 1.2 - Recover actual subsequence

```
import java.io.*;
import java.util.*;
public class LIS {
    public static void main(String[] args){
        try {
            BufferedReader br = new BufferedReader(
                    new InputStreamReader(System.in));
            int count = Integer.parseInt(br.readLine());
            int nums[] = new int[count];
            for (int i = 0 ; i < count ; i ++){
                nums[i] = Integer.parseInt(br.readLine());
            }

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

            List<Integer> initialList = new ArrayList<Integer>();
            initialList.add(nums[0]);
            seq.add(initialList);

            for (int i = 1; i < count ; i ++){
                List<Integer> lastEle = seq.get(seq.size() - 1);
                if (nums[i] > lastEle.get(lastEle.size() - 1)){
```

```
                    List<Integer> endList = new ArrayList<Integer>();
                    int maxLevel = getMaxLevel(seq, seq.size());
                    for (int idx = 0 ; idx < maxLevel ; idx ++){
                        endList.add(-1);
                    }
                    endList.add(nums[i]);
                    seq.add(endList);
                }
                else if (nums[i] == lastEle.get(lastEle.size() - 1)){
                    continue;
                }
                else{
                    insertIntoSeq(seq, nums[i]);
                }
            }

            int maxLevel = getMaxLevel(seq, seq.size());
            System.out.println(seq.size());
            for (int seqIdx = seq.size() - 1 ; seqIdx >= 0 ; seqIdx --){
                List<Integer> curList = seq.get(seqIdx);
                if (curList.size() - 1 <= maxLevel){
                    maxLevel = curList.size() - 1;
                }
                System.out.print(curList.get(maxLevel) + "  ");
            }
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static int getMaxLevel(List<List<Integer>> seq, int idx) {
        int maxLevel = -1;
        for (int seqIdx = 0 ; seqIdx < idx ; seqIdx ++){
            maxLevel = Math.max(maxLevel, seq.get(seqIdx).size() - 1);
        }
        return maxLevel;
    }

    private static void insertIntoSeq(List<List<Integer>> seq, int num) {
        int start = 0 , end = seq.size() - 1;
        while (start < end){
            int mid = start + (end - start) / 2;
            int cur = seq.get(mid).get(seq.get(mid).size() - 1);
            if (num < cur){
                end = mid;
            }
            else{
                start = mid + 1;
            }
        }
        int maxLevel = getMaxLevel(seq, start);
        int sub = maxLevel - seq.get(start).size();
```

```
            List<Integer> newList = new ArrayList<Integer>();
            for (int i = 0 ; i < sub ; i ++){
                newList.add(-1);
            }
            for (int ele : seq.get(start)){
                newList.add(ele);
            }
            newList.add(num);
            seq.set(start, newList);
        }
    }
```

Problem 204

334 Increasing Triplet Subsequence

```
Given an unsorted array return whether an increasing subsequence
 of length 3 exists or not in the array.

Formally the function should:
Return true if there exists i, j, k
such that arr[i] < arr[j] < arr[k] given 0 <= i < j < k <= n-1 else return false.
Your algorithm should run in O(n) time complexity and O(1) space complexity.

Examples:
Given [1, 2, 3, 4, 5],
return true.

Given [5, 4, 3, 2, 1],
return false.

https://leetcode.com/problems/increasing-triplet-subsequence/
```

Solution

Use idea of Solution 1.2 of Problem "Longest Increasing Subsequence", by having smaller, larger integers to store current minimum element and second smallest element after the minimum element, if there is another element larger than larger(2nd smallest after smallest/ or maybe before smallest, but has another smaller element ahead), there will be an increasing sequence with the length of 3.

Listing 266: Solution

```
public class Solution {
    public boolean increasingTriplet(int[] nums) {
        if (nums.length < 3){
            return false;
        }
        int len = 1;
        int smaller = nums[0];
        int larger = -1;

        for (int idx = 1 ; idx < nums.length ; idx ++){
            if (nums[idx] < smaller){
                smaller = nums[idx];
            }
            else if (nums[idx] > smaller){
                if (len == 1){
                    larger = nums[idx];
                    len++;
                }
                if (len == 2){
                    if (nums[idx] > larger){
                        return true;
                    }
                    else{
                        larger = nums[idx];
```

```
                        }
                    }
                }
            }
            return false;
        }
}
```

Problem 205

368 Largest Divisible Subset

```
Given a set of distinct positive integers,
 find the largest subset such that every pair (Si, Sj) of elements
  in this subset satisfies: Si % Sj = 0 or Sj % Si = 0.
If there are multiple solutions, return any subset is fine.
Example 1:
nums: [1,2,3]

Result: [1,2] (of course, [1,3] will also be ok)

Example 2:
nums: [1,2,4,8]

Result: [1,2,4,8]

https://leetcode.com/problems/largest-divisible-subset/
```

Solution

Follow the similar solution for 'Longest Increasing Subsequence'. $O(n^2)$ complexity. Can probably be optimized a bit to track both previous element and max length of path using array instead of creating arrays of Set/List.

Listing 267: Solution

```
public class Solution {
    public List<Integer> largestDivisibleSubset(int[] nums) {
        List<Integer> maxList = new ArrayList<Integer>();
        Arrays.sort(nums);

        List[] lists = new ArrayList[nums.length];
        for (int idxNum = 0 ; idxNum < nums.length ; idxNum ++){
            lists[idxNum] = new ArrayList<Integer>();
            lists[idxNum].add(nums[idxNum]);
            for (int idxFormer = 0 ; idxFormer < idxNum ; idxFormer ++){
                if (nums[idxNum] % nums[idxFormer] == 0){
                    if (lists[idxFormer].size() >= lists[idxNum].size()){
                        lists[idxNum] = new ArrayList<Integer>(lists[idxFormer]);
                        lists[idxNum].add(nums[idxNum]);
                    }
                }
            }
            if (lists[idxNum].size() > maxList.size()){
                maxList = new ArrayList<Integer>(lists[idxNum]);
            }
        }

        return maxList;
    }
}
```

Problem 206

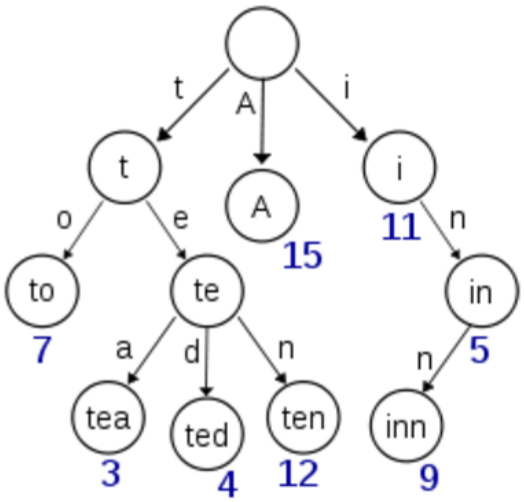
208 Implement Trie (Prefix Tree)

```
Implement a trie with insert, search, and startsWith methods.

https://leetcode.com/problems/implement-trie-prefix-tree/
```

Solution

The problem itself is not difficult at all, but Trie(Prefix Tree) is what we need to know.

Trie树又被称为字典树、前缀树，是一种用于快速检索的多叉树。Tried树可以利用字符串的公共前缀来节省存储空间。
但如果系统存在大量没有公共前缀的字符串，相应的Trie树将非常消耗内存。（下图为Wiki上的Trie树示意图， https://en.wikipedia.org/wiki/Trie）



Trie树的基本性质如下：

- 根节点不包括字符，除根节点外每个节点包括一个支付。
- 从根节点到某一节点，路径上经过的字符连接起来，即为对应的字符串。
- 每个节点的所有子节点包含的字符串各不相同。

本题中的Trie树可以简单实现如下：
Trie树节点数据结构定义如下：

1. val表示该节点对应的字符
2. 子节点简单用一个数组表示，这样实现比较简单，但比较耗费内存
3. isWord标记从根节点到该节点是否表示一个单词，还是另一单词的前缀。

Listing 268: Solution

```
class TrieNode {
    // Initialize your data structure here.
    TrieNode[] children = new TrieNode[26];
    boolean isWord;

    public TrieNode() {
        this.isWord = false;
        for (int i = 0 ; i < 26 ; i ++){
            children[i] = null;
        }
```

```
    }
}

public class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    // Inserts a word into the trie.
    public void insert(String word) {
        if (word.length() == 0){
            return;
        }
        TrieNode cur = this.root;

        for (int wordIdx = 0 ; wordIdx < word.length() ; wordIdx ++){
            int childrenIdx = word.charAt(wordIdx) - 'a';
            if (cur.children[childrenIdx] == null){
                cur.children[childrenIdx] = new TrieNode();
            }

            cur = cur.children[childrenIdx];
        }
        cur.isWord = true;
    }

    // Returns if the word is in the trie.
    public boolean search(String word) {
        if (word.length() == 0){
            return false;
        }
        TrieNode cur = this.root;

        for (int wordIdx = 0 ; wordIdx < word.length() ; wordIdx ++){
            int childrenIdx = word.charAt(wordIdx) - 'a';
            if (cur.children[childrenIdx] == null){
                return false;
            }
            cur = cur.children[childrenIdx];
        }
        return cur.isWord;
    }

    // Returns if there is any word in the trie
    // that starts with the given prefix.
    public boolean startsWith(String prefix) {
        if (prefix.length() == 0){
            return false;
        }
        TrieNode cur = this.root;
        for (int prefixIdx = 0 ; prefixIdx < prefix.length() ; prefixIdx ++){
```

```
            int childrenIdx = prefix.charAt(prefixIdx) - 'a';
            if (cur.children[childrenIdx] == null){
                return false;
            }
            cur = cur.children[childrenIdx];
        }

        return true;
    }
}

// Your Trie object will be instantiated and called as such:
// Trie trie = new Trie();
// trie.insert("somestring");
// trie.search("key");
```

Problem 207

211 Add and Search Word - Data structure design

```
Design a data structure that supports the following two operations:

void addWord(word)
bool search(word)
search(word) can search a literal word or a regular expression string
 containing only letters a-z or .
 A . means it can represent any one letter.

For example:

addWord("bad")
addWord("dad")
addWord("mad")
search("pad") -> false
search("bad") -> true
search(".ad") -> true
search("b..") -> true
Note:
You may assume that all words are consist of lowercase letters a-z.


You should be familiar with how a Trie works.
 If not, please work on this problem: Implement Trie (Prefix Tree) first.

https://leetcode.com/problems/add-and-search-word-data-structure-design/
```

Solution

Not the same as Problem "Implement Trie", a bit more difficult, use a recursive solution to solve the problem.

Listing 269: Solution

```
class TrieNode{
    TrieNode[] children = new TrieNode[26];
    boolean isWord;
    char val;

    public TrieNode(){
        isWord = false;
        for (int childrenIdx = 0 ;
             childrenIdx < children.length ; childrenIdx++){
            children[childrenIdx] = null;
        }
    }

    public TrieNode(char val){
        this();
        this.val = val;
    }
}
```

Problem 207 continued on next page...

```
public class WordDictionary {
    TrieNode root = new TrieNode();
    // Adds a word into the data structure.
    public void addWord(String word) {
        TrieNode cur = root;
        for (int idx = 0 ; idx < word.length() ; idx ++){
            int cIdx = word.charAt(idx) - 'a';
            if (cur.children[cIdx] == null){
                cur.children[cIdx] = new TrieNode(word.charAt(idx));
            }
            cur = cur.children[cIdx];
        }
        cur.isWord = true;
    }

    // Returns if the word is in the data structure. A word could
    // contain the dot character '.' to represent any one letter.
    public boolean search(String word) {
        TrieNode cur = root;
        return match(cur, word, 0);
    }

    private boolean match(TrieNode cur , String word, int startIdx){
        boolean result = false;

        if (cur == null){
            return false;
        }

        if (startIdx >= word.length()){
            return cur.isWord;
        }
        char curCh = word.charAt(startIdx);
         if (curCh == '.'){
            for (TrieNode select : cur.children){
                if (select != null){
                    result = result | match(select, word, startIdx + 1);
                    if (result){
                        return true;
                    }
                }
            }
        }
        else{
            cur = cur.children[curCh - 'a'];
            if (cur == null){
                return false;
            }
            else{
                char compareCh = cur.val;
                // Is this condition check necessary?
                if (curCh == compareCh){
                    return match(cur, word, startIdx + 1);
```

```
                }
                else{
                    return false;
                }
            }
        }

        return result;
    }
}

// Your WordDictionary object will be instantiated and called as such:
// WordDictionary wordDictionary = new WordDictionary();
// wordDictionary.addWord("word");
// wordDictionary.search("pattern");
```

Problem 208

79 Word Search

```
Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell,
 where "adjacent" cells are those horizontally or vertically neighboring.
  The same letter cell may not be used more than once.

For example,
Given board =

[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]
word = "ABCCED", -> returns true,
word = "SEE", -> returns true,
word = "ABCB", -> returns false.

https://leetcode.com/problems/word-search/
```

Solution

Problem 79.

Solution is using straightforward backtracking method, be sure to return as soon as we get result, otherwise will get a TLE error.(I stuck in misusing || as |).

Listing 270: Solution

```
public class Solution {
    // int[][] directions = {{1,0},{-1,0},{0,1},{0,-1}};
    private boolean wordSearch(char[][] board, boolean[][] visited,
     int curX, int curY, String word, int wordIdx){
        int m = board.length;
        int n = board[0].length;

        if (wordIdx >= word.length()){
            return true;
        }
        if (curX < 0|| curY < 0|| curX >= board.length|| curY >= board[0].length
            || word.charAt(wordIdx) != board[curX][curY]
            || visited[curX][curY] == true){
            return false;
        }

        visited[curX][curY] = true;

        boolean result=wordSearch(board,visited, curX-1, curY, word, wordIdx+1)
                    || wordSearch(board,visited, curX+1, curY, word, wordIdx+1)
                    || wordSearch(board,visited, curX, curY-1, word, wordIdx+1)
                    || wordSearch(board,visited, curX, curY+1, word, wordIdx+1);
```

Problem 208 continued on next page...

```
        visited[curX][curY] = false;

        return result;
    }

    public boolean exist(char[][] board, String word) {
        if (board.length == 0 || board[0].length == 0){
            return false;
        }
        boolean[][] visited = new boolean[board.length][board[0].length];

        for (int i = 0 ; i < board.length ; i ++){
            for (int j = 0 ; j < board[0].length ; j ++){
                if (wordSearch(board, visited, i, j, word, 0)){
                    return true;
                }
            }
        }

        return false;
    }
}
```

Problem 209

212 Word Search II

```
Given a 2D board and a list of words from the dictionary,
 find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell,
 where "adjacent" cells are those horizontally or vertically neighboring.
  The same letter cell may not be used more than once in a word.

For example,
Given words = ["oath","pea","eat","rain"] and board =

[
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
Return ["eat","oath"].
Note:
You may assume that all inputs are consist of lowercase letters a-z.

click to show hint.

You would need to optimize your backtracking to pass the larger test.
 Could you stop backtracking earlier?

If the current candidate does not exist in all words' prefix,
 you could stop backtracking immediately.
  What kind of data structure could answer such query efficiently?
   Does a hash table work? Why or why not? How about a Trie?
    If you would like to learn how to implement a basic trie,
     please work on this problem: Implement Trie (Prefix Tree) first.

https://leetcode.com/problems/word-search-ii/
```

Solution

This is the first problem I cannot finish for the first time!! Because I even cannot finish the testing of big data in Eclipse! More than 65536 bytes! And it stuck my computer!

For optimizations, see
https://leetcode.com/discuss/77851/java-15ms-easiest-solution-100-00%25

Some notes:
Combing them, Trie is the natural choice. Notice that:
TrieNode is all we need. search and startsWith are useless.
No need to store character at TrieNode. c.next[i] != null is enough.
Never use c1 + c2 + c3. Use StringBuilder.
No need to use $O(n^2)$ extra space visited[m][n].
No need to use StringBuilder. Storing word itself at leaf node is enough.
No need to use HashSet to de-duplicate. Use "one time search" trie.

Some optimizations:

1. 59ms: Use search and startsWith in Trie class like this popular solution.
2. 33ms: Remove Trie class which unnecessarily starts from root in every dfs call.
3. 30ms: Use w.toCharArray() instead of w.charAt(i).
4. 22ms: Use StringBuilder instead of c1 + c2 + c3.
5. 20ms: Remove StringBuilder completely by storing word instead of boolean in TrieNode.
6. 20ms: Remove visited[m][n] completely by modifying board[i][j] = '#' directly.
7. 18ms: check validity, e.g., if(i > 0) dfs(...), before going to the next dfs.
8. 17ms: De-duplicate c - a with one variable i.
9. 15ms: Remove HashSet completely. dietpepsi's idea is awesome.

I'll list my solution and optimized solution on my note.

Listing 271: My Solution

```
class TrieNode {
    TrieNode[] next = new TrieNode[26];
    String word;
}

public class Solution {

    public List<String> findWords(char[][] board, String[] words) {
        if (board.length == 0 || board[0].length == 0){
            return new ArrayList<String>();
        }

        List<String> res = new ArrayList<>();
        TrieNode root = buildTrie(words);
        for(int i = 0; i < board.length; i++) {
            for(int j = 0; j < board[0].length; j++) {
                wordSearch(board, i, j, root, res);
            }
        }
        return res;
    }

    private void wordSearch(char[][] board, int curX, int curY,
     TrieNode curTrie, List<String> res){
        if (curX < 0 || curY < 0
            || curX >= board.length || curY >= board[0].length
            || board[curX][curY] == '#'){
            return;
        }

        char curC = board[curX][curY];
```

```
            curTrie = curTrie.next[curC - 'a'];
            if (curTrie == null){
                return;
            }
            if (curTrie.word != null){
                res.add(curTrie.word);
                curTrie.word = null;
                // cannot return here! may have a word whose prefix is also a word
                // return;
            }
            board[curX][curY] = '#';
            wordSearch(board, curX + 1, curY, curTrie, res);
            wordSearch(board, curX - 1, curY, curTrie, res);
            wordSearch(board, curX, curY + 1, curTrie, res);
            wordSearch(board, curX, curY - 1, curTrie, res);

            board[curX][curY] = curC;
        }

        public TrieNode buildTrie(String[] words) {
        TrieNode root = new TrieNode();
        for(String w : words) {
            TrieNode p = root;
            for(char c : w.toCharArray()) {
                int i = c - 'a';
                if(p.next[i] == null) p.next[i] = new TrieNode();
                p = p.next[i];
            }
           p.word = w;
        }
        return root;
    }
}
```

And a much better solution.

Listing 272: Better Solution

```
public List<String> findWords(char[][] board, String[] words) {
    List<String> res = new ArrayList<>();
    TrieNode root = buildTrie(words);
    for(int i = 0; i < board.length; i++) {
        for(int j = 0; j < board[0].length; j++) {
            dfs(board, i, j, root, res);
        }
    }
    return res;
}

public void dfs(char[][] board, int i, int j, TrieNode p, List<String> res) {
    char c = board[i][j];
    if(c == '#' || p.next[c - 'a'] == null) return;
    p = p.next[c - 'a'];
    if(p.word != null) {   // found one
```

```
            res.add(p.word);
            p.word = null;      // de-duplicate
        }

        board[i][j] = '#';
        if(i > 0) dfs(board, i - 1, j ,p, res);
        if(j > 0) dfs(board, i, j - 1, p, res);
        if(i < board.length - 1) dfs(board, i + 1, j, p, res);
        if(j < board[0].length - 1) dfs(board, i, j + 1, p, res);
        board[i][j] = c;
    }

    public TrieNode buildTrie(String[] words) {
        TrieNode root = new TrieNode();
        for(String w : words) {
            TrieNode p = root;
            for(char c : w.toCharArray()) {
                int i = c - 'a';
                if(p.next[i] == null) p.next[i] = new TrieNode();
                p = p.next[i];
            }
            p.word = w;
        }
        return root;
    }

    class TrieNode {
        TrieNode[] next = new TrieNode[26];
        String word;
    }
```

(BTW. I solved this problem using his TrieNode...)

Problem 210

139 Word Break

```
Given a string s and a dictionary of words dict,
 determine if s can be segmented into a space-separated sequence
  of one or more dictionary words.

For example, given
s = "leetcode",
dict = ["leet", "code"].

Return true because "leetcode" can be segmented as "leet code".

https://leetcode.com/problems/word-break/
```

Solution

A DP problem, not so difficult if you know the key idea of solution.

The state transition equation is like dp[i+word.length()] = dp[i], where s.substring(i, i+word.length()).equals(word).

Listing 273: Solution

```
public class Solution {
    public boolean wordBreak(String s, Set<String> wordDict) {
        boolean dp[] = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 0 ; i < s.length() ; i ++){
            if (dp[i]){
                for (String word : wordDict){
                    if (i + word.length() <= s.length()){
                        String compare = s.substring(i, i + word.length());
                        if (compare.equals(word)){
                            dp[i + word.length()] = true;
                        }
                    }
                }
            }
        }

        return dp[s.length()];
    }
}
```

Problem 211

140 Word Break II

```
Given a string s and a dictionary of words dict,
 add spaces in s to construct a sentence where
  each word is a valid dictionary word.

Return all such possible sentences.

For example, given
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].

https://leetcode.com/problems/word-break-ii/
```

Solution

First have a glance at Problem "Work Break", on how to finish the split process.

For this problem, we should also maintain the dp array, but this time not boolean, but String, a list of String to be more specific. dp[i] is the list of words in dictionary ends at index i with similar train of thought as "Word Break";

Finally, use dfs to combine all possibilities.

Listing 274: Solution

```
public class Solution {
    List<String> result = new ArrayList<String>();
    private void findWordCombinations(List<String> curResult, int endIdx,
     String s, List<String>[] wordSnippets){
        if (endIdx > s.length()){
            return;
        }
        if (endIdx == 0){
            String combine = "";
            for (int wordIdx = curResult.size() - 1 ;
                     wordIdx >= 0 ; wordIdx --){
                combine += curResult.get(wordIdx);
                if (wordIdx != 0){
                    combine += " ";
                }
            }
            result.add(combine);
            return;
        }
        if (wordSnippets[endIdx] == null){
            return;
        }
        for (String word : wordSnippets[endIdx]){
            curResult.add(word);
            findWordCombinations(curResult, endIdx - word.length()
             , s, wordSnippets);
            curResult.remove(curResult.size() - 1);
```

```
        }
    }
    public List<String> wordBreak(String s, Set<String> wordDict) {
        List<String>[] wordSnippets = new ArrayList[s.length() + 1];
        wordSnippets[0] = new ArrayList<String>();
        for (int i = 0 ; i < s.length() ; i ++){
            if (wordSnippets[i] != null){
                // Should have a precessor
                for (String word : wordDict){
                    int wordLen = word.length();
                    if (wordLen + i <= s.length()){
                        String part = s.substring(i, i + wordLen);
                        if (part.equals(word)){
                            if (i + wordLen > s.length()){
                                continue;
                            }
                            if (wordSnippets[i + wordLen] == null){
                                wordSnippets[i + wordLen]
                                = new ArrayList<String>();
                            }
                            wordSnippets[i + wordLen].add(part);
                        }
                    }
                }
            }
        }
        findWordCombinations(new ArrayList<String>(),
             s.length() , s, wordSnippets);
        return result;
    }
}
```

Problem 212

127 Word Ladder

```
Given two words (beginWord and endWord),
 and a dictionary's word list,
  find the length of shortest transformation sequence
   from beginWord to endWord, such that:

Only one letter can be changed at a time
Each intermediate word must exist in the word list
For example,

Given:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
As one shortest transformation is
     "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.

Note:
    Return 0 if there is no such transformation sequence.
    All words have the same length.
    All words contain only lowercase alphabetic characters.

https://leetcode.com/problems/word-ladder/
```

Solution

First we may realize that this is a graph(undirected) problem using BFS. Two nodes are neighbors when they have an distance of 1(Only one letter differs). So we may write code as below:

Listing 275: TLE Solution

```
public class Solution {
public int ladderLength(String start, String end, Set<String> dict) {

        //create graph
5       HashMap<String, ArrayList<String>> graph =
            new HashMap<String, ArrayList<String>>();

        graph.put(start, new ArrayList<String>());
        graph.put(end, new ArrayList<String>());
10      for(String d : dict) {
            graph.put(d, new ArrayList<String>());
        }
        for(String s : graph.keySet()) {
            ArrayList<String> list = graph.get(s);
15          for(String t : graph.keySet()) {
                if(getDiff(s,t) == 1) {
                    list.add(t);
                }
            }
20      }
        // use BFS to traverse the node in the graph, we begin with "start"
```

```
        int step = 0;
        HashSet<String> visited = new HashSet<String>();
        ArrayList<String> firstLevel = new ArrayList<String>(graph.get(start));

        while (firstLevel.size() != 0) {
            step++;
            ArrayList<String> nextLevel = new ArrayList<String>();
            for (String s : firstLevel) {
                if (s.equals(end)) return step + 1;
                visited.add(s);
                nextLevel.addAll(graph.get(s));
            }
            firstLevel.clear();
            for (String t : nextLevel) {
                if (!visited.contains(t)) {
                    firstLevel.add(t);
                }
            }
            nextLevel.clear();
        }
        return 0;
    }

    public int getDiff(String w1, String w2) {
        int count = 0;
        for(int i = 0; i < w1.length(); i++) {
            if(w1.charAt(i) != w2.charAt(i)) {
                count++;
            }
        }
        return count;
    }
}
```

A simplified method is to avoid process that generates a graph. Still a BFS framework, but this time a different method. Substitute each letter of a word from 'a' to 'z' to see if generated word exists in dictionary, which is in nature same as former solution, but has a better performance when the count of word is large(since few of them are of 1 distance, we waste our time comparing them) and word is short.

However, a one-pass solution will also get TLE error. So we need to optimize it further. Do it with 2 pass. Go from both sourceSet and targetSet, set the shorter one to 'sourceSet', guaranteeing that we are always searching from a shorter set, which will accelerate our execution.

Listing 276: Solution

```
public class Solution {
    public int ladderLength(String start, String end, Set<String> dict) {
        Set<String> sourceSet = new HashSet<String>();
        Set<String> targetSet = new HashSet<String>();
        sourceSet.add(start);
        targetSet.add(end);
        return findLadder(sourceSet, targetSet, dict, 1);
    }
```

```
    private int findLadder(Set<String> sourceSet,
            Set<String> targetSet, Set<String> dict, int depth){
        if (sourceSet.size() == 0){
            return 0;
        }
        if (targetSet.size() < sourceSet.size()){
            return findLadder(targetSet, sourceSet, dict, depth);
        }
        for (String s : sourceSet){
            dict.remove(s);
        }
        for (String s : targetSet){
            dict.remove(s);
        }
        Set<String> newSet = new HashSet<String>();
        for (String source : sourceSet){
            for (int strIdx = 0 ; strIdx < source.length() ; strIdx ++){
                // If I put
                //  char[] sourceCharArr = source.toCharArray();
                // here, I will get TLE?!
                for (char subChar = 'a' ; subChar <= 'z' ; subChar ++){
                    if (source.charAt(strIdx) == subChar){
                        continue;
                    }
                    // This statement...
                    char[] sourceCharArr = source.toCharArray();
                    sourceCharArr[strIdx] = subChar;
                    String newSource = new String(sourceCharArr);
                    for (String target : targetSet){
                        if (target.equals(newSource)){
                            return depth + 1;
                        }
                    }
                    if (dict.contains(newSource)){
                        newSet.add(newSource);
                    }
                }
            }
        }
        return findLadder(newSet, targetSet, dict, depth + 1);
    }
}
```

Problem 213

126 Word Ladder II

```
Given two words (beginWord and endWord),
 and a dictionary's word list,
  find all shortest transformation sequence(s)
   from beginWord to endWord, such that:

Only one letter can be changed at a time
Each intermediate word must exist in the word list
For example,

    Given:
    beginWord = "hit"
    endWord = "cog"
    wordList = ["hot","dot","dog","lot","log"]
    Return
      [
        ["hit","hot","dot","dog","cog"],
        ["hit","hot","lot","log","cog"]
      ]
Note:
    All words have the same length.
    All words contain only lowercase alphabetic characters.

https://leetcode.com/problems/word-ladder-ii/
```

Solution

Similar to Problem "Word Ladder", still a two-pass method, set beginWords the shorter set which will reduce the time burden.

Building the result path is another tricky part(which is maybe more tricky). Generating a HashMap with key of source String and value of strings it can generated by just changing one letter of the word is able to solve the problem. Just DFS the Map, we will get the final result.

Listing 277: Solution

```
public class Solution {
    List<List<String>> result = new ArrayList<List<String>>();
    public List<List<String>> findLadders(String beginWord, String endWord,
                                          Set<String> wordList) {
        Map<String, List<String>> succs = new HashMap<String, List<String>>();
        Set<String> beginWords = new HashSet<String>();
        Set<String> endWords = new HashSet<String>();
        beginWords.add(beginWord);
        endWords.add(endWord);
        bfsLadders(beginWords, endWords, wordList, succs, false);

        List<String> pathStarts = new ArrayList<String>();
        pathStarts.add(beginWord);
        buildPath(beginWord, endWord, succs, new ArrayList(pathStarts));
        return result;
    }
    private void bfsLadders(Set<String> beginWords, Set<String> endWords,
```

```
        Set<String> dict, Map<String, List<String>> succs, boolean reversed){

    if (beginWords.isEmpty()){
      return;
    }
    if (beginWords.size() > endWords.size()){
        bfsLadders(endWords, beginWords, dict, succs, !reversed);
        return;
    }
    for (String beginWord : beginWords){
        dict.remove(beginWord);
    }
    for (String endWord : endWords){
        dict.remove(endWord);
    }
    Set<String> generated = new HashSet<String>();
    boolean found = false;
    for (String beginWord : beginWords){
        for (int strIdx = 0 ; strIdx < beginWord.length() ; strIdx ++){
            char[] wordCharArr = beginWord.toCharArray();
            for (char subChar = 'a' ; subChar <= 'z' ; subChar ++){
                // Don't add this!
                // if (wordCharArr[strIdx] == subChar){
                //     continue;
                // }
                wordCharArr[strIdx] = subChar;
                String newStr = new String(wordCharArr);
                String key = reversed ? newStr : beginWord;
                String value = reversed ? beginWord : newStr;
                List<String> list = succs.containsKey(key) ?
                         succs.get(key) : new ArrayList<String>();

                if (endWords.contains(newStr)){
                        found = true;

                        list.add(value);
                        succs.put(key, list);
                }
                else{
                    if (!dict.contains(newStr)){
                        continue;
                    }
                    else{
                        if (!found){
                            generated.add(newStr);

                            list.add(value);
                            succs.put(key, list);
                        }
                    }
                }
            }
        }
```

```
            }

            if (found){
                return;
            }
            bfsLadders(generated, endWords, dict, succs, reversed);
        }

        private void buildPath(String beginWord, String endWord,
                 Map<String, List<String>> succs, List<String> curList){
            if (beginWord.equals(endWord)){
                result.add(new ArrayList(curList));
                return;
            }

            if (!succs.containsKey(beginWord)){
                return;
            }

            for (String next : succs.get(beginWord)){
                curList.add(next);
                buildPath(next, endWord, succs, curList);
                curList.remove(curList.size() - 1);
            }
        }
    }
```

Problem 214

5 Longest Palindromic Substring

```
Given a string S,
 find the longest palindromic substring in S.
  You may assume that the maximum length of S is 1000,
   and there exists one unique longest palindromic substring.

https://leetcode.com/problems/longest-palindromic-substring/
```

Solution - DP

First solution is DP solution with $O(n^2)$, it can output a result through 'Run'(within an acceptable time limit, I guess), but will get TLE when submit.

Mention the variables in loop - len and i rather than i and j since we should get dp value of shorter string first to get longer ones.

Listing 278: Solution 1

```
public class Solution {
    public String longestPalindrome(String s) {
        int len = s.length();
        if (len == 0 || len == 1){
            return s;
        }
        int[][] dp = new int[len][len];
        int max = 1;
        int startIdx = -1;
        for (int i = 0 ; i < len ; i ++){
            dp[i][i] = 1;
            if (max == 1){
                startIdx = i;
            }
            if (i > 0 && s.charAt(i) == s.charAt(i - 1)){
                dp[i - 1][i] = 2;
                max = 2;
                startIdx = i - 1;
            }
        }
        for (int subLen = 3 ; subLen <= len  ; subLen ++){
            for (int i = 0 ; i < len - subLen + 1 ; i ++){
                int j = i + subLen - 1;
                if (s.charAt(i) == s.charAt(j) && dp[i + 1][j - 1] != 0){
                    dp[i][j] = Math.max(dp[i][j] , dp[i + 1][j - 1] + 2);
                    if (max < dp[i][j]){
                        max = dp[i][j];
                        startIdx = i;
                    }
                }
            }
        }
        return s.substring(startIdx, startIdx + max);
    }
}
```

Problem 214 continued on next page...

Solution - Expanding

Solution is an $O(n^2)$*runtime* solution. The train of thought is interesting and reversing. Instead of finding the longest, the solution start from the middle of palindrome and form a palindrome from the middle. Note that there are two ways of expanding the middle element(s), one from single element, while another from two adjacent elements.

Listing 279: Solution 2

```
public class Solution {
    public String longestPalindrome(String s) {
        if (s.length() == 0 || s.length() == 1){
            return s;
        }
        int max = 0;
        int startIdx = -1;
        for (int i = 0 ; i < s.length() ; i ++){
            int singleMax = expandToSides(s, i, i);
            if (max < singleMax){
                max = singleMax;
                startIdx = i - max / 2;
            }
            if (i > 0){
                int doubleMax = expandToSides(s, i - 1, i);
                if (max < doubleMax){
                    max = doubleMax;
                    startIdx = i - max / 2;
                }
            }
        }
        return s.substring(startIdx, startIdx + max);
    }
    public int expandToSides(String s, int left, int right){
        int start = left, end = right;
        int max = 0;
        while (start >= 0 && end < s.length()){
            if (s.charAt(start) == s.charAt(end)){
                max = Math.max(end - start + 1, max);
            }
            else{
                break;
            }
            start --;
            end ++;
        }
        return max;
    }
}
```

Solution - Manacher

Such a neat, elegant, excellent algorithm that I still cannot totally understand it...

Though there is a blog for Manacher in Chinese, I think the English version in Leetcode articles is much better.(Maybe with the example?) I'll post the screenshots and code here for future use.

An O(N) Solution (Manacher's Algorithm):

First, we transform the input string, S, to another string T by inserting a special character '#' in between letters. The reason for doing so will be immediately clear to you soon.

For example: S = "abaaba", T = "#a#b#a#a#b#a#".

To find the longest palindromic substring, we need to expand around each T_i such that $T_{i-d} \ldots T_{i+d}$ forms a palindrome. You should immediately see that *d* is the length of the palindrome itself centered at T_i.

We store intermediate result in an array P, where P[i] equals to the length of the palindrome centers at T_i. The longest palindromic substring would then be the maximum element in P.

Using the above example, we populate P as below (from left to right):

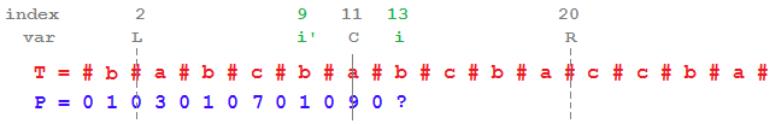
```
T = # a # b # a # a # b # a #
P = 0 1 0 3 0 1 6 1 0 3 0 1 0
```

Looking at P, we immediately see that the longest palindrome is "abaaba", as indicated by $P_6 = 6$.

Did you notice by inserting special characters (#) in between letters, both palindromes of odd and even lengths are handled graciously? (Please note: This is to demonstrate the idea more easily and is not necessarily needed to code the algorithm.)

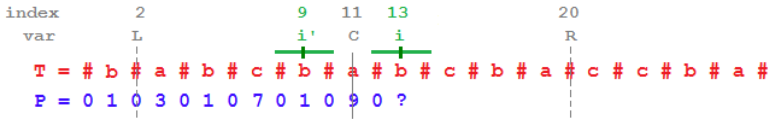
Now, imagine that you draw an imaginary vertical line at the center of the palindrome "abaaba". Did you notice the numbers in P are symmetric around this center? That's not only it, try another palindrome "aba", the numbers also reflect similar symmetric property. Is this a coincidence? The answer is yes and no. This is only true subjected to a condition, but anyway, we have great progress, since we can eliminate recomputing part of P[i]'s.

Let us move on to a slightly more sophisticated example with more some overlapping palindromes, where S = "babcbabcbaccba".



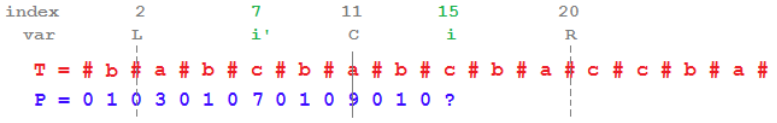
Above image shows T transformed from S = "babcbabcbaccba". Assumed that you reached a state where table P is partially completed. The solid vertical line indicates the center (C) of the palindrome "abcbabcba". The two dotted vertical line indicate its left (L) and right (R) edges respectively. You are at index i and its mirrored index around C is i'. How would you calculate P[i] efficiently?

Assume that we have arrived at index i = 13, and we need to calculate P[13] (indicated by the question mark ?). We first look at its mirrored index i' around the palindrome's center C, which is index i' = 9.



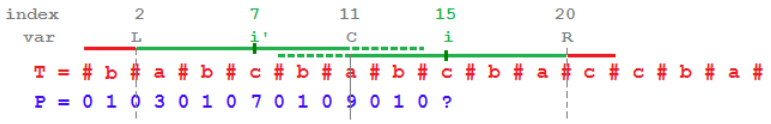
The two green solid lines above indicate the covered region by the two palindromes centered at i and i'. We look at the mirrored index of i around C, which is index i'. P[i'] = P[9] = 1. It is clear that P[i] must also be 1, due to the symmetric property of a palindrome around its center.

As you can see above, it is very obvious that P[i] = P[i'] = 1, which must be true due to the symmetric property around a palindrome's center. In fact, all three elements after C follow the symmetric property (that is, P[12] = P[10] = 0, P[13] = P[9] = 1, P[14] = P[8] = 0).



Now we are at index i = 15, and its mirrored index around C is i' = 7. Is P[15] = P[7] = 7?

Now we are at index i = 15. What's the value of P[i]? If we follow the symmetric property, the value of P[i] should be the same as P[i'] = 7. But this is wrong. If we expand around the center at T_{15}, it forms the palindrome "a#b#c#b#a", which is actually shorter than what is indicated by its symmetric counterpart. Why?



Colored lines are overlaid around the center at index i and i'. Solid green lines show the region that must match for both sides due to symmetric property around C. Solid red lines show the region that might not match for both sides. Dotted green lines show the region that crosses over the center.

It is clear that the two substrings in the region indicated by the two solid green lines must match exactly. Areas across the center (indicated by dotted green lines) must also be symmetric. Notice carefully that P[i '] is 7 and it expands all the way across the left edge (L) of the palindrome (indicated by the solid red lines), which does not fall under the symmetric property of the palindrome anymore. All we know is P[i] ≥ 5, and to find the real value of P[i] we have to do character matching by expanding past the right edge (R). In this case, since P[21] ≠ P[1], we conclude that P[i] = 5.

Let's summarize the key part of this algorithm as below:

if P[i'] ≤ R – i,
then P[i] ← P[i']
else P[i] ≥ P[i']. (Which we have to expand past the right edge (R) to find P[i].

See how elegant it is? If you are able to grasp the above summary fully, you already obtained the essence of this algorithm, which is also the hardest part.

The final part is to determine when should we move the position of C together with R to the right, which is easy:

If the palindrome centered at i does expand past R, we update C to i, (the center of this new palindrome), and extend R to the new palindrome's right edge.

In each step, there are two possibilities. If P[i] ≤ R – i, we set P[i] to P[i'] which takes exactly one step. Otherwise we attempt to change the palindrome's center to i by expanding it starting at the right edge, R. Extending R (the inner while loop) takes at most a total of N steps, and positioning and testing each centers take a total of N steps too. Therefore, this algorithm guarantees to finish in at most 2*N steps, giving a linear time solution.

Listing 280: Solution 3

```
public class Solution {
    public String longestPalindrome(String s) {
        if (s.length() <= 2){
            if (s.length() != 2 || s.charAt(1) == s.charAt(0)){
                return s;
            }
            else{
                return s.substring(0, 1);
            }
        }
        int maxLen = 1;
        String curStr = preprocess(s);
        int[] lens = new int[2 * curStr.length() + 1];
        int rightBound = 0;
        int mid = 1;
        for (int idx = 1 ; idx < lens.length ; idx ++){
            lens[idx] = (rightBound > idx)
             ? Math.min(lens[2 * mid - idx], rightBound - idx): 0;
```

```
            while (idx + lens[idx] < curStr.length() && idx - lens[idx] >= 0
                    && curStr.charAt(idx + lens[idx])
                     == curStr.charAt(idx - lens[idx])){
                lens[idx] ++;
            }
            if (idx + lens[idx] > rightBound){
                rightBound = idx + lens[idx];
                mid = idx;
            }
        }
        int maxPos = 0;
        for (int lenIdx = 0 ; lenIdx < lens.length ; lenIdx ++){
            int len = lens[lenIdx];
            if (maxLen < len - 1){
                maxLen = len - 1;
                maxPos = lenIdx;
            }
        }
        String result = s.substring((maxPos - maxLen)/2,
            maxLen + (maxPos - maxLen) / 2);
        return result;
    }

    private String preprocess(String curStr){
        StringBuilder sb = new StringBuilder();
        sb.append("#");
        for (int idxCh = 0 ; idxCh < curStr.length() ; idxCh ++){
            sb.append(curStr.charAt(idxCh));
            sb.append("#");
        }
        return sb.toString();
    }
}
```

Problem 215

221 Maximal Square

```
Given a 2D binary matrix filled with 0's and 1's,
 find the largest square containing all 1's and return its area.

For example, given the following matrix:

1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
Return 4.

https://leetcode.com/problems/maximal-square/
```

Solution

A simple DP problem. Take care of the input: not int[][] matrix, but char[][] matrix.

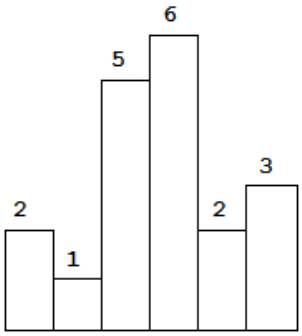
Listing 281: Solution

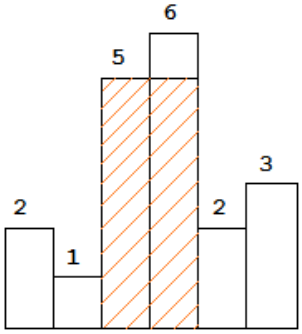
```
public class Solution {
    public int maximalSquare(char[][] matrix) {
        int m = matrix.length;
        if (m == 0){
            return 0;
        }
        int n = matrix[0].length;
        int dp[][] = new int[m][n];
        int maxSide = 0;
        for (int i = 0 ; i < m ; i++){
            dp[i][0] = matrix[i][0] - '0';
        }
        for (int i = 0 ; i < n ; i++){
            dp[0][i] = matrix[0][i] - '0';
        }
        for (int i = 1; i < m ; i++){
            for (int j = 1 ; j < n ; j ++){
                if (matrix[i][j] == '1'){
                    dp[i][j] = Math.min(dp[i - 1][j - 1],
                     Math.min(dp[i - 1][j], dp[i][j - 1])) + 1;
                }
            }
        }
        for (int i = 0 ; i < m ; i ++){
            for (int j = 0 ; j < n ; j ++){
                maxSide = Math.max(maxSide, dp[i][j]);
            }
        }
        return maxSide * maxSide;
    }
}
```

Problem 216

84 Largest Rectangle in Histogram

```
Given n non-negative integers representing the histogram's bar
 height where the width of each bar is 1,
  find the area of largest rectangle in the histogram.


Above is a histogram where width of each bar is 1,
 given height = [2,1,5,6,2,3].


The largest rectangle is shown in the shaded area,
 which has area = 10 unit.

For example,
Given heights = [2,1,5,6,2,3],
return 10.

https://leetcode.com/problems/largest-rectangle-in-histogram/
```





Solution

A simple train of thought is like Problem "Longest Palindromic Substring", by starting from middle and expand to both sides. However, when there is a set of big data with similar values, we have to compute nearly all values for n^2, which will of course lead to TLE.

Listing 282: TLE solution

```
public class Solution {
    public int largestRectangleArea(int[] heights) {
        int max = 0;
        for (int i = 0 ; i < heights.length ; i ++){
```

```
            int height = heights[i];
            int left = i , right = i;
            while (left > 0 && heights[left - 1] >= height){
                left --;
            }
            while (right < heights.length - 1
             && heights[right + 1] >= height){
                right ++;
            }
            int width = right - left + 1;

            max = Math.max(width * height, max);
        }

        return max;
    }
}
```

There is another brilliant train of thought of dealing with this problem.

Use a 'ascending stack' to solve the problem. Iterate along array, when heights[idx] is larger than top element of stack, push it; otherwise pop all elements larger than current element, and since popped elements are in descending order, we can just use (height of popped element * width++) to get a potential maximum rectangle. The width of rectangle is another difficulty, see figure.

这道题brute force的方法很直接，就是对于每一个窗口，找到其中最低的高度，然后求面积，去其中最大的矩形面积。总共有n^2个窗口，找最低高度是O(n)的操作，所以复杂度是O(n^3)。

接下来我们讨论一种比较容易理解的思路，就是从每一个bar往两边走，以自己的高度为标准，直到两边低于自己的高度为止，然后用自己的高度乘以两边走的宽度得到矩阵面积。因为我们对于任意一个bar都计算了以自己为目标高度的最大矩阵，所以最好的结果一定会被取到。每次往两边走的复杂度是O(n)，总共有n个bar，所以时间复杂度是O(n^2)。因为代码比较简单，这里就不列出来了哈。

最后我们谈谈最优的解法，思路跟Longest Valid Parentheses比较类似，我们要维护一个栈，这个栈从低向上的高度是依次递增的，如果遇到当前bar高度比栈顶元素低，那么就出栈直到满足条件，过程中检测前面满足条件的矩阵。关键问题就在于在出栈时如何确定当前出栈元素的所对应的高度的最大范围是多少。答案跟Longest Valid Parentheses中括号的范围相似，就是如果栈已经为空，说明到目前为止所有元素（当前下标元素除外）都比出栈元素高度要大（否则栈中肯定还有元素），所以矩阵面积就是高度乘以当前下标i。如果栈不为空，那么就是从当前栈顶元素的下一个到当前下标的元素之前都比出栈元素高度大（因为栈顶元素第一个比当前出栈元素小的）。具体的实现代码如下：

Listing 283: Solution

```
public class Solution {
    public int largestRectangleArea(int[] heights) {
        Stack<Integer> ascendingStack = new Stack<Integer>();

        if (heights.length == 0){
            return 0;
        }
        if (heights.length == 1){
            return heights[0];
        }

        int max = 0;
        for (int i = 0 ; i < heights.length ; i ++){
            if (ascendingStack.isEmpty()
             || heights[i] > heights[ascendingStack.peek()]){
                ascendingStack.push(i);
                continue;
```

```
            }
            else{
                while (!ascendingStack.isEmpty()
                 && heights[ascendingStack.peek()] >= heights[i]){
                    int heightIdx = ascendingStack.pop();
                    int height = heights[heightIdx];
                    int width = (ascendingStack.isEmpty())
                     ? i : i - ascendingStack.peek() - 1;
                    max = Math.max(width * height , max);
                }
                ascendingStack.push(i);
            }
        }

        while (!ascendingStack.isEmpty()){
            int heightIdx = ascendingStack.pop();
            int height = heights[heightIdx];
            int width = (ascendingStack.isEmpty())
             ? heights.length: heights.length - ascendingStack.peek() - 1;
            max = Math.max(width * height , max);
        }

        return max;
    }
}
```

Problem 217

85 Maximal Rectangle

```
Given a 2D binary matrix filled with 0's and 1's,
 find the largest rectangle containing all ones and return its area.

https://leetcode.com/problems/maximal-rectangle/
```

Solution

Inspired by Problem "Largest Rectangle in Histogram", the problem can be transformed to a problem doing m(m is the total number of rows) times with algorithm in previous problem.(Think of why)

Listing 284: Solution

```
public class Solution {
    public int maximalRectangle(char[][] matrix) {
        Stack<Integer> stack = new Stack<Integer>();
        int m = matrix.length;
        if (m == 0){
            return 0;
        }
        int n = matrix[0].length;

        int[][] nums = new int[m][n];

        for (int i = m - 1 ; i >= 0 ; i --){
            for (int j = 0 ; j < n ; j ++){
                if (i == m - 1){
                    nums[i][j] = matrix[i][j] - '0';
                }
                else{
                    if (matrix[i][j] == '1'){
                        nums[i][j] = nums[i + 1][j] + (matrix[i][j] - '0');
                    }
                    else{
                        nums[i][j] = 0;
                    }
                }
            }
        }

        int max = 0;

        for (int i = m - 1 ; i >= 0 ; i --){
            stack = new Stack<Integer>();
            for (int j = 0 ; j <= n ; j ++){
                if (j != n && (stack.isEmpty()
                     || nums[i][stack.peek()] < nums[i][j])){
                    stack.push(j);
                    continue;
                }
                else{
                    while (!stack.isEmpty()
                     && (j == n || nums[i][stack.peek()] >= nums[i][j])){
```

```
                        int idx = stack.pop();
                        int height = nums[i][idx];
                        int width = (stack.isEmpty())? j: j - stack.peek() - 1;
                        max = Math.max(width * height, max);
                    }

                    if (j < n){
                        stack.push(j);
                    }
                }
            }
        }
        return max;
    }
}
```

Problem 218

43 Multiply Strings

```
Given two numbers represented as strings,
 return multiplication of the numbers as a string.

Note: The numbers can be arbitrarily large and are non-negative.

https://leetcode.com/problems/multiply-strings/
```

Solution

I unexpectedly solve the problem in just one attempt!

However, I missed the given condition that numbers are non-negative, which means I forgot to consider the negative numbers at first sight(but I do consider the 0 result).

Listing 285: Solution

```java
public class Solution {
    public String multiply(String num1, String num2) {
        if (num1.length() < num2.length()){
            return multiply(num2, num1);
        }

        if (num1.equals("0") || num2.equals("0")){
            return "0";
        }

        int[] numArr1 = new int[num1.length()];
        int[] numArr2 = new int[num2.length()];
        for (int idx = 0 ; idx < num1.length() ;idx ++){
            numArr1[idx] = num1.charAt(idx) - '0';
        }
        for (int idx = 0 ; idx < num2.length() ;idx ++){
            numArr2[idx] = num2.charAt(idx) - '0';
        }

        int[] result = new int[num1.length() + num2.length()];

        for (int num2Idx = num2.length() - 1 ; num2Idx >= 0 ; num2Idx --){
            int carry = 0;
            int resultCarry = 0;
            int endIdx = num2.length() - 1 - num2Idx;
            for (int num1Idx = num1.length() - 1 ; num1Idx >= 0 ; num1Idx --){
                int multiplyResult = numArr1[num1Idx] * numArr2[num2Idx];
                int digit = (multiplyResult + carry) % 10;
                carry = (multiplyResult + carry) / 10;
                int addResult = digit + result[endIdx];
                result[endIdx] = (addResult + resultCarry) % 10;
                resultCarry = (addResult + resultCarry) / 10;
                endIdx ++;
            }
            if (carry != 0 || resultCarry != 0){
                result[endIdx] += resultCarry + carry;
```

```
                if (result[endIdx] >= 10){
                    result[endIdx + 1] = result[endIdx];
                    result[endIdx] %= 10;
                }
            }
        }
        String finalResult = "";
        boolean firstDigit = false;
        for (int idx = result.length - 1; idx >= 0 ; idx --){
            if (!firstDigit && result[idx] != 0){
                firstDigit = true;
            }
            if (!firstDigit && result[idx] == 0){
                continue;
            }
            finalResult += String.valueOf(result[idx]);
        }

        return finalResult;
    }
}
```

Problem 219

29 Divide Two Integers

```
Divide two integers without using multiplication,
 division and mod operator.

If it is overflow, return MAX_INT.

https://leetcode.com/problems/divide-two-integers/
```

Solution

At first sight I thought this is a problem similar to "Multiply Strings", however, this is indeed a Math and Binary Search problem.

Without division and multiplication, remaining operators are +, - , and bit operators. The most straightforward way is to reduce divisor from dividend until remaining number is less than divisor, which will get TLE.

Have you noticed that, << 1 is the same as * 2? That's the key insight to this problem. See code in listing.(There are still many edge cases!)

Something to take care. See in the listing, use 'divisor <= (*dividend* >> 1)' instead of 'dividend < divisor', the latter will take 903ms to calculate Integer.MAX_VALUE / 2 while the former takes 0ms...

Listing 286: Solution

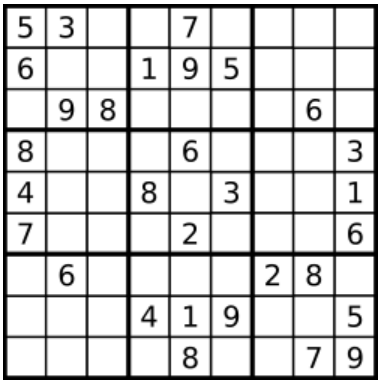
```
public class Solution {
    public int divide(int dividend, int divisor) {
        if (divisor == 0){
            return Integer.MAX_VALUE;
        }

        int result = 0;

        if(dividend==Integer.MIN_VALUE){
            result = 1;
            dividend += Math.abs(divisor);
        }
        if(divisor==Integer.MIN_VALUE)
            return result;

        // boolean negative = ((dividend^divisor)>>>31==1)?true:false;
        boolean negative = (dividend > 0 && divisor < 0||
                            dividend < 0 && divisor > 0);
        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);

        if (dividend == 0 || dividend < divisor){
            return negative ? -result : result;
        }


        int count = 0;
        // while (dividend < divisor){
        while (divisor <= (dividend >> 1)){
```

```
            divisor <<= 1;
            count ++;
        }
        while (count >= 0){
            while (dividend >= divisor){
                dividend -= divisor;
                result += (1 << count);
            }
            divisor >>= 1;
            count --;
        }
        if (result == Integer.MIN_VALUE && !negative){
            return Integer.MAX_VALUE;
        }

        return negative ? -result : result;
    }
}
```

Problem 220

36 Valid Sudoku

```
Determine if a Sudoku is valid, according to:
 Sudoku Puzzles - The Rules.

The Sudoku board could be partially filled,
 where empty cells are filled with the character '.'.


A partially filled sudoku which is valid.

Note:
A valid Sudoku board (partially filled)
 is not necessarily solvable.
  Only the filled cells need to be validated.

https://leetcode.com/problems/valid-sudoku/
```



Solution

It's an easy problem, use 3 arrays to validate value on rows, columns and grids(with 3 * 3 size) are used. Description in Note makes the problem even easier.

Listing 287: Solution

```
public class Solution {
    int max_value = 9;
    public boolean isValidSudoku(char[][] board) {
        boolean[][] rowFilled = new boolean[board.length][max_value];
        boolean[][] colFilled = new boolean[board[0].length][max_value];
        boolean[][] gridFilled = new boolean[9][max_value];

        for (int i = 0; i < board.length; i ++){
            for (int j = 0 ; j < board[0].length ; j ++){
                if (board[i][j] == '.'){
                    continue;
                }
                int curVal = board[i][j] - '0' - 1;
                if (rowFilled[i][curVal] || colFilled[j][curVal]
                    || gridFilled[i / 3 * 3 + j / 3][curVal]){
                    return false;
```

```
                }
                rowFilled[i][curVal] = true;
                colFilled[j][curVal] = true;
                gridFilled[i / 3 * 3 + j / 3][curVal] = true;
            }
        }

        return true;
    }
}
```

Problem 221

37 Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

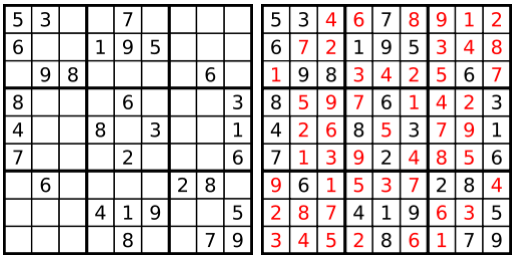
Empty cells are indicated by the character '.'.

You may assume that there will be only one unique solution.

A sudoku puzzle...

...and its solution numbers marked in red.

https://leetcode.com/problems/sudoku-solver/



Solution

Just like "N-Queens", pre-processing will much accelerate the speed.

Listing 288: Solution

```
public class Solution {
    boolean[][] rowFilled = new boolean[9][9];
    boolean[][] colFilled = new boolean[9][9];
    boolean[][] gridFilled = new boolean[9][9];

    private boolean fill(char[][] board, int pos){
        int curX = pos / 9;
        int curY = pos % 9;
        if (board[curX][curY] == '.'){
            for (int ava = 0 ; ava < 9 ; ava ++){
                if (rowFilled[curX][ava] || colFilled[curY][ava]
                    || gridFilled[curX / 3 * 3 + curY / 3][ava]){
                    continue;
                }
                else{
                    // have a try
                    rowFilled[curX][ava] = true;
                    colFilled[curY][ava] = true;
                    gridFilled[curX / 3 * 3 + curY / 3][ava] = true;
                    board[curX][curY] = (char)('0' + 1 + ava);
                    if (pos == 80){
                        return true;
                    }
```

```
                    if (!fill(board, pos + 1)){
                        //recover
                        board[curX][curY] = '.';
                        rowFilled[curX][ava] = false;
                        colFilled[curY][ava] = false;
                        gridFilled[curX / 3 * 3 + curY / 3][ava] = false;
                    }
                    else{
                        return true;
                    }
                }
            }
            return false;
        }
        else{
            if (pos == 80){
                return true;
            }

            return fill(board, pos + 1);
        }
    }

    public void solveSudoku(char[][] board) {
        for (int i = 0 ; i < 9 ; i ++){
            for (int j = 0 ; j < 9 ; j++){
                if (board[i][j] == '.'){
                    continue;
                }
                int curVal = board[i][j] - '0' - 1;
                rowFilled[i][curVal] = true;
                colFilled[j][curVal] = true;
                gridFilled[i / 3 * 3 + j / 3][curVal] = true;
            }
        }

        fill(board, 0);
    }
}
```

Problem 222

34 Search for a Range

```
Given a sorted array of integers,
 find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example,
Given [5, 7, 7, 8, 8, 10] and target value 8,
return [3, 4].

https://leetcode.com/problems/search-for-a-range/
```

Solution

Binary search problem. The key to problem is to do binary search twice.

Listing 289: Solution

```
public class Solution {
    public int[] searchRange(int[] nums, int target) {
        int[] result = new int[2];
        int left = find(nums, target, false);
        int right = find(nums, target, true);
        result[0] = left;
        result[1] = right;
        return result;
    }

    private int find(int[] nums, int target, boolean forward){
        int start = 0 , end = nums.length - 1;

        while (start <= end){
            int mid = start + (end - start) / 2;
            if (nums[mid] == target){
                if (forward){
                    if (mid == nums.length - 1 || nums[mid] < nums[mid + 1]){
                        return mid;
                    }
                    else{
                        start = mid + 1;
                    }
                }else{
                    if (mid == 0 || nums[mid - 1] < nums[mid]){
                        return mid;
                    }
                    else{
                        end = mid - 1;
                    }
                }
            }
            else if(nums[mid] < target){
```

```
                start = mid + 1;
            }
            else {
                end = mid - 1;
            }
        }

        return -1;
    }
}
```

Problem 223

209 Minimum Size Subarray Sum

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

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

click to show more practice.

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

https://leetcode.com/problems/search-for-a-range/
```

Solution 1

Use 2 pointers to maintain the sum and check if less than current minimum condition.
$O(n)$ solution.

Listing 290: Solution 1

```
public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        if (nums.length == 0){
            return 0;
        }
        int low = 0 , high = 0;
        int sum = 0;
        int minLen = Integer.MAX_VALUE;
        while (high < nums.length){
            if (sum >= s){
                minLen = Math.min(high - low , minLen);
                sum -= nums[low];
                low ++;
            }
            else{
                sum += nums[high];
                high ++;
            }
        }
        boolean changed = false;
        while (low < nums.length && sum >= s){
            changed = true;
            sum -= nums[low];
            low ++;
            if (sum < s){
                break;
            }
        }
        if (changed){
            minLen = Math.min(minLen, high - low + 1);
```

Problem 223 continued on next page...

```
        }
        return minLen == Integer.MAX_VALUE ? 0 : minLen;
    }
}
```

Solution 2

Second solution is to use binary search to find proper length to calculate the sum. See listing.

The $O(nlogn)$ solution.

Listing 291: Solution 2

```
public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        int min = Integer.MAX_VALUE;
        int i = 1 , j = nums.length;
        while (i <= j){
            int mid = i + (j - i) / 2;
            if (hasWindow(nums, mid,s)){
                min = Math.min(min, mid);
                j = mid - 1;
            }
            else{
                i = mid + 1;
            }
        }

        return min == Integer.MAX_VALUE ? 0 : min;
    }
    public boolean hasWindow(int[] nums, int curLen, int s){
        int sum = 0;
        for (int i = 0 ;i < nums.length ; i++){
            if (i >= curLen){
                sum -= nums[i - curLen];
            }
            sum += nums[i];
            if (sum >= s){
                return true;
            }
        }
        return false;
    }

}
```

Problem 224

292 Nim Game

```
You are playing the following Nim Game with your friend:
 There is a heap of stones on the table,
  each time one of you take turns to remove 1 to 3 stones.
   The one who removes the last stone will be the winner.
    You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game.
 Write a function to determine whether
  you can win the game given the number of stones in the heap.

For example, if there are 4 stones in the heap,
 then you will never win the game:
  no matter 1, 2, or 3 stones you remove,
   the last stone will always be removed by your friend.

Hint:

If there are 5 stones in the heap, could you figure out
 a way to remove the stones such that you will always be the winner?

https://leetcode.com/problems/nim-game/
```

Solution

An interesting problem with a solution with only one line!! Solved by myself!

Listing 292: Solution

```
public class Solution {
    public boolean canWinNim(int n) {
        return (n % 4 != 0);
    }
}
```

Problem 225

290 Word Pattern

```
Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection
 between a letter in pattern and a non-empty word in str.

Examples:
pattern = "abba", str = "dog cat cat dog" should return true.
pattern = "abba", str = "dog cat cat fish" should return false.
pattern = "aaaa", str = "dog cat cat dog" should return false.
pattern = "abba", str = "dog dog dog dog" should return false.
Notes:
You may assume pattern contains only lowercase letters,
 and str contains lowercase letters separated by a single space.

https://leetcode.com/problems/word-pattern/
```

Solution

Not difficult, solved by Set and a array(similar to a Hashmap)
Someone solved it by a Set and 2 Hashmaps...

Listing 293: Solution

```
public class Solution {
    public boolean wordPattern(String pattern, String str) {
        String[] strs = new String[26];
        String[] parts = str.split(" ");
        if (parts.length != pattern.length()){
            return false;
        }
        Set<String> strOcc = new HashSet<String>();
        for (int i = 0 ; i < 26 ; i++){
            strs[i] = "";
        }
        for (int idx = 0 ; idx < pattern.length() ; idx ++){
            int curOffset = pattern.charAt(idx) - 'a';
            if (strs[curOffset].length() == 0){
                if (strOcc.contains(parts[idx]))
                    return false;
                else
                    strOcc.add(parts[idx]);
                strs[curOffset] = parts[idx];
                continue;
            }
            else{
                if (!strs[curOffset].equals(parts[idx]))
                    return false;
            }
        }
        return true;
    }
}
```

Problem 226

316 Remove Duplicate Letters

```
Given a string which contains only lowercase letters,
 remove duplicate letters so that every letter appear once and only once.
  You must make sure your result is the smallest
   in lexicographical order among all possible results.

Example:
Given "bcabc"
Return "abc"

Given "cbacdcbc"
Return "acdb"

https://leetcode.com/problems/remove-duplicate-letters/
```

Solution‘

List the train of thought as below:

1. Calculate a array of count where that how many times each letter appears.
2. Find a non-repeating sequence, break the iterating when any character has no successors using the count array maintained.
3. Get the smallest letter in sequence, remove letters in front of it and all same letters behind the element.
4. Iterate to get next element.

Solution description in Chinese can be seen
https://segmentfault.com/a/1190000004188227.

Listing 294: Solution

```
public class Solution {
    public String removeDuplicateLetters(String s) {
        if (s.length() == 0)
            return "";
        int[] count = new int[26];
        for (int i = 0 ; i < s.length() ; i ++)
            count[s.charAt(i) - 'a'] ++;
        char minChar = 'z';
        int minPos = 0;
        for (int i = 0 ; i < s.length() ; i++){
            if (minChar > s.charAt(i)){
                minPos = i;
                minChar = s.charAt(i);
            }
            count[s.charAt(i) - 'a'] --;
            if (count[s.charAt(i) - 'a'] == 0)
                break;
        }
        return minChar + removeDuplicateLetters(
            s.substring(minPos + 1).replaceAll("" + minChar, ""));
```

```
        }
    }
```

Problem 227

89 Gray Code

```
The gray code is a binary numeral system where
 two successive values differ in only one bit.

Given a non-negative integer n representing
 the total number of bits in the code,
  print the sequence of gray code.
   A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2].
 Its gray code sequence is:

00 - 0
01 - 1
11 - 3
10 - 2
Note:
For a given n, a gray code sequence is not uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence
 according to the above definition.

For now, the judge is able to judge based on one instance of
 gray code sequence. Sorry about that.

https://leetcode.com/problems/gray-code/
```

Solution

Consider when n=2 and n=3

When n=2, the sequence is 0,1,3,2, namely 00,01,11,10

When n=3, the sequence is 0**00**,0**01**,0**11**,0**10**,1**10**,1**11**,1**01**,1**00**, just the sequence from 2 and add 1 to the front reversely.

Listing 295: Solution 1.1

```
public class Solution {
    public List<Integer> grayCode(int n) {
        List<Integer> result = new ArrayList<Integer>();
        result.add(0);
        for (int i = 0 ; i < n ; i ++){
            int high = 1 << i;
            for (int j = result.size() - 1; j >= 0; j --){
                result.add(result.get(j) + high);
            }
        }

        return result;
    }
}
```

There is even a solution looks like this.

Listing 296: Solution 1.2

```
public class Solution {
    public List<Integer> grayCode(int n) {
        List<Integer> gray = new ArrayList<Integer>();
        for(int i = 0 ; i < 1<<n ; i++) {
            int temp = i>>1;
            gray.add(i^temp);
        }
        return gray;
    }
}
```

Another alternative solution.

Gray Code， 每次看每次都不记得。写下来让自己好温习。

Gray Code 0 = 0, 下一项是toggle最右边的bit(LSB), 再下一项是toggle最右边值为 "1" bit的左边一个bit。然后重复

如: 3bit

Gray Code: 000, 001, 011, 010, 110, 111, 101, 100, 最右边值为 "1" 的bit在最左边了，结束。

Binary : 000, 001, 010, 011, 100, 101, 110, 111

再者就是Binary Code 转换为Gray Code了。

如:

Binary Code : 1011 要转换成Gray Code

1011 = 1（照写第一位）, 1(第一位与第二位异或 1^0 = 1), 1(第二位异或第三位， 0^1=1), 0 (1^1 =0) = 1110

其实就等于 (1011 >> 1) ^ 1011 = 1110

有了上面的等式写code就简单了

Problem 228

134 Gas Station

```
There are N gas stations along a circular route,
 where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and
 it costs cost[i] of gas to travel from station i
  to its next station (i+1).
   You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index
 if you can travel around the circuit once, otherwise return -1.

Note:
The solution is guaranteed to be unique.

https://leetcode.com/problems/gas-station/
```

Solution 1

First, get all diffs by gas[i] & cost[i], which tells us how much oil will be left after one station.

Then, do you realize that the problem has been transformed into a problem finding the sub-array with maximum sum, but in a **circular** array.

(BTW, how to deal with subarray problems, refer to
http://blog.csdn.net/hcbbt/article/details/10454947)

Back to the problem. We have known how to solve sub-array problems, how about circular array?!

Don't be afraid(since I'm stuck for such a long while). The difficulty lays on how to find max sum of subarray starts from latter half of array and ends at former half of it. How about get the minimum of current array? The total sum is given, minimum sum means maximum on the other side. What we need to do differently from sequential problems is to check which one is larger, max or totalSum-min.

Listing 297: Solution 1

```
public class Solution {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int[] diff = new int[gas.length];
        int[] sum = new int[gas.length];
        int totalGas = 0;
        // Of course I can combine the calculation of totalSum
        // with calculating max and min
        for (int i = 0 ; i < gas.length ; i++){
            diff[i] = gas[i] - cost[i];
            totalGas += diff[i];
        }
        if (totalGas < 0){
            return -1;
        }
        int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
        int maxIdx = 0, minIdx = 0;
        int curMax = 0 , curMin = 0;
        for (int i = 0 ; i < gas.length ; i++){
            if (curMax < 0){
                curMax = diff[i];
```

```
                maxIdx = i;
            }
            else{
                curMax += diff[i];
            }
            if (curMin > 0){
                curMin = diff[i];
            }
            else{
                curMin += diff[i];
            }
            if (curMin < min){
                min = curMin;
                minIdx = (i + 1) % gas.length;
            }
            if (curMax > max){
                max = curMax;
            }
        }
        return (max > totalGas - min) ? maxIdx : minIdx;
    }
}
```

Solution 2

Another solution is the jump solution.

Listing 298: Solution 2

```
public class Solution {
    public int canCompleteCircuit(int[] gas, int[] cost) {
        int[] diff = new int[gas.length];
        int totalGas = 0;
        for (int i = 0 ; i < gas.length ; i++){
            diff[i] = gas[i] - cost[i];
            totalGas += diff[i];
        }
        if (totalGas < 0){
            return -1;
        }
        int startIdx = 0;
        int sum = 0;
        for (int i = 0 ; i < gas.length ; i++){
            sum += diff[i];
            if (sum < 0){
                startIdx = (i + 1) % gas.length;
                sum = 0;
            }
        }
        return startIdx;
    }
}
```

Problem 229

135 Candy

```
There are N children standing in a line.
Each child is assigned a rating value.

You are giving candies to these children
 subjected to the following requirements:

Each child must have at least one candy.
Children with a higher rating get more candies than their neighbors.
What is the minimum candies you must give?

https://leetcode.com/problems/candy/
```

Solution 1

Solution 1 is greedy. Iterating from front to end, when there is a pair which ratings[i] < ratings[i + 1], give candies[i] = 1 OR stay unchanged; give candies[i + 1] = candies[i] + 1; Then iterate backward, doing the similar operation, except for the condition when the i^{th} people has already been given the candies, that's when we give the man maxcandies[i] , candies[i + 1] + 1.

Finally, don't forget to give all children who aren't given any candies 1 candy for comfort.(Consider the input of [2,2,1])

Listing 299: Solution 1

```
public class Solution {
    public int candy(int[] ratings) {
        if (ratings.length == 0){
            return 0;
        }
        if (ratings.length == 1){
            return 1;
        }
        int candies[] = new int[ratings.length];
        Arrays.fill(candies, 1);
        for (int i = 0 ; i < ratings.length - 1 ; i++){
            if (ratings[i] < ratings[i + 1]){
                candies[i + 1] = candies[i] + 1;
            }
        }
        for (int i = ratings.length - 1 ; i > 0 ; i--){
            if (ratings[i] < ratings[i - 1]){
                candies[i - 1] = Math.max(candies[i] + 1, candies[i - 1]);
            }
        }

        int sum = 0;
        for (int i = 0 ; i < candies.length ; i ++){
            sum += candies[i];
        }

        return sum;
    }
```

```
}
```

Solution 2

Solution 2 is tactical and can avoid using extra $O(n)space$. Let's consider the non-descending condition - [1,2,3,4,5] OR [1,2,3,3,4], when the sequence is ascending, we don't need to worry about the candy-giving thing, since the result is obvious after one pass - [1,2,3,4,5] & [1,2,3,1,2]. Now consider something more complicated - [1,5,4,3,2], still we give kid_0(with rating of 1) 1 candy, kid_1 2 candies temporarily, then we meet kid_2 with 4 ratings, give him 1(we'll know why afterwards), and now we are in a descending sequence with length of 2 by now.

Then the important part. We meet kid_3 with 3 ratings, which is still in the descending sequence, and that's when we know we have underestimated kid_1, so we give him an extra candy, namely total++; don't forget kid_3, we give him 2 candies(here we are balancing kid_2 and kid_3, in fact kid_2 is the one who should get 2 candies by now, not kid_3, since we are calculating the total amount of candies, so it doesn't matter who get the candy.) Similarly, when kid_4 arrives, we give him(or kid_2, to be more precious) 3 candies and compensate kid_1 again.

Listing 300: Solution 2

```
public class Solution {
    public int candy(int[] ratings) {
        if (ratings.length == 0){
            return 0;
        }
        if (ratings.length == 1){
            return 1;
        }
        int sum = 1;
        int descLen = 0;
        int ascCandyCnt = 1;
        int descCandyCnt = 0;
        for (int i = 1 ; i < ratings.length ; i++){
            if (ratings[i] > ratings[i - 1]){
                if (descLen != 0){
                    ascCandyCnt = 1;
                }
                ascCandyCnt ++;
                sum += ascCandyCnt;
                descLen = 0;
                descCandyCnt = 0;
            }
            else if (ratings[i] == ratings[i - 1]){
                sum ++;
                ascCandyCnt = 1;
                descLen = 0;
                descCandyCnt = 0;
            } else{
                // ratings[i] < ratings[i - 1]
                descLen ++;
                if (ascCandyCnt <= descLen){
                    sum ++;
                }
                descCandyCnt ++;
                sum += descCandyCnt;
```

```
                }
            }

            return sum;
        }
}
```

Problem 230

205 Isomorphic Strings

```
Given two strings s and t,
 determine if they are isomorphic.

Two strings are isomorphic if
 the characters in s can be replaced to get t.

All occurrences of a character must be replaced with
 another character while preserving the order of characters.
  No two characters may map to the same character but
   a character may map to itself.

For example,
Given "egg", "add", return true.

Given "foo", "bar", return false.

Given "paper", "title", return true.

Note:
You may assume both s and t have the same length.

https://leetcode.com/problems/isomorphic-strings/
```

Solution

An easy problem using HashMap..... Wait a minute! HashMaps(or a map with a set)! Consider when the input is “ab” & “aa”.

Listing 301: Solution

```
public class Solution {
    public boolean isIsomorphic(String s, String t) {
        if (s.length() == 0){
            return true;
        }
        Map<Character, Character> map = new HashMap<Character, Character>();
        Map<Character, Character> reverseMap
         = new HashMap<Character, Character>();

        for (int idx = 0 ; idx < s.length() ; idx++){
            char charS = s.charAt(idx);
            char charT = t.charAt(idx);

            if (map.containsKey(charS)){
                char compare = map.get(charS);
                if (compare != charT){
                    return false;
                }
            }
            else{
                if (reverseMap.containsKey(charT)){
                    return false;
```

```
                    }
                    map.put(charS, charT);
                    reverseMap.put(charT, charS);
                }
            }

            return true;
        }
}
```

Problem 231

4 Median of Two Sorted Arrays

```
There are two sorted arrays nums1 and nums2 of
 size m and n respectively.
  Find the median of the two sorted arrays.
   The overall run time complexity should be O(log (m+n)).

https://leetcode.com/problems/median-of-two-sorted-arrays/
```

Solution 1

Honestly, this is not a correct solution though it got AC on OJ. The complexity is $O(m+n)$ rather than $O(log(m+n))$.

Listing 302: Solution 1

```
public class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int len1 = nums1.length;
        int len2 = nums2.length;
        int mid = (len1 + len2) / 2, midVal = 0;
        int mid2 = mid + 1, mid2Val = 0;
        if (mid == 0){
            return (nums1.length == 0) ? nums2[0] : nums1[0];
        }
        int idx1 = 0 , idx2 = 0;
        int idx = 0;
        while (idx1 < nums1.length && idx2 < nums2.length){
            int compare1 = nums1[idx1], compare2 = nums2[idx2];
            idx ++;
            int compare = 0;
            if (compare1 <= compare2){
                compare = compare1;
                idx1 ++;
            }
            else{
                compare = compare2;
                idx2 ++;
            }
            if (idx == mid){
                midVal = compare;
            }
            if (idx == mid2){
                mid2Val = compare;
                break;
            }
        }
        if (idx < mid2){
            while (idx1 < nums1.length){
                idx++;
                if (idx == mid){
                    midVal = nums1[idx1];
                }
                if (idx == mid2){
```

```
                    mid2Val = nums1[idx1];
                }
                idx1++;
            }
            while (idx2 < nums2.length){
                idx++;
                if (idx == mid){
                    midVal = nums2[idx2];
                }
                if (idx == mid2){
                    mid2Val = nums2[idx2];
                }
                idx2++;
            }
        }


        if ((len1 + len2) % 2 == 0){
            return (midVal + mid2Val) * 1.0 / 2;
        }
        else{
            return mid2Val;
        }
    }
}
```

Solution 2

Solution 2 makes use of Binary Search, with a pretty complicated process on calculating the related index. The train of thought is quite interesting.

For detailed description on solution , visit `http://blog.csdn.net/zxzxy1988/article/details/8587244`

Listing 303: Solution 2

```
public class Solution {
    private double findMedian(int[] nums1, int[] nums2,
                              int startIdx1, int startIdx2 , int remaining){
        if (nums1.length - startIdx1 < nums2.length - startIdx2){
            return findMedian(nums2, nums1, startIdx2, startIdx1, remaining);
        }
        // Since length1 > length2, no need to add this snippet
        // if (startIdx1 == nums1.length){
        //     return nums2[startIdx2 + remaining - 1];
        // }
        if (startIdx2 == nums2.length){
            return nums1[startIdx1 + remaining - 1];
        }
        if (remaining == 1){
            return Math.min(nums1[startIdx1], nums2[startIdx2]);
        }
        int offset2 = Math.min(nums2.length - startIdx2, remaining / 2);
        int offset1 = remaining - offset2;
        if (nums1[startIdx1 + offset1 - 1] == nums2[startIdx2 + offset2 - 1]){
```

```
            return nums1[startIdx1 + offset1 - 1];
        }
        else if (nums1[startIdx1 + offset1 - 1] > nums2[startIdx2 +  offset2 - 1]){
            return findMedian(nums1, nums2,
             startIdx1, startIdx2 + offset2, remaining - offset2);
        }
        else{
            return findMedian(nums1, nums2,
             startIdx1 + offset1, startIdx2, remaining - offset1);
        }
    }

    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int m = nums1.length, n = nums2.length;
        if ((m + n) % 2 != 0){
            return findMedian(nums1, nums2, 0, 0, (m + n) / 2 + 1);
        }
        else{
            return (findMedian(nums1, nums2, 0, 0, (m + n) / 2)
                  + findMedian(nums1, nums2, 0, 0, (m + n) / 2 + 1)) / 2;
        }
    }
}
```

Problem 232

295 Find Median from Data Stream

```
Median is the middle value in an ordered integer list.
 If the size of the list is even, there is no middle value.
  So the median is the mean of the two middle value.

Examples:
[2,3,4] , the median is 3

[2,3], the median is (2 + 3) / 2 = 2.5

Design a data structure that supports the following two operations:

void addNum(int num) - Add a integer number from the data stream
                       to the data structure.
double findMedian() - Return the median of all elements so far.
For example:

add(1)
add(2)
findMedian() -> 1.5
add(3)
findMedian() -> 2

https://leetcode.com/problems/find-median-from-data-stream/
```

Though almost all solutions on Internet like CSDN has only one way to solve this - using two heaps, one max heap and one min heap, this is still a problem worth 2+ hours to think carefully about it, to be honest.

Solution 1 - Array & List

Unsorted array is enough for the problem. Partition is what we usually do in QuickSort, so as to finish the task of finding the median. Partition is of $O(n)$ *runtime*, where inserting to unsorted array only costs $O(1)$. This method will get TLE in OJ.

Listing 304: Solution 1.1 - Unsorted Array

```
public class MedianFinder {
    static final int MAX_LEN = 100000;
    int[] nums = new int[MAX_LEN];
    int length = 0;
    // Adds a number into the data structure.
    public void addNum(int num) {
        nums[length ++] = num;
    }

    // Returns the median of current data stream
    public double findMedian() {
        if (length % 2 == 0){
            return (partition(nums, 0, length - 1, length / 2 - 1)
             + partition(nums, 0, length - 1, length / 2)) / 2;
        }
        else{
            return partition(nums, 0, length - 1, length / 2);
```

```
        }

    }

    private double partition(int[] nums, int start, int end, int target){
//      if (start > end){
//          return -1;
//      }
        int pivot = nums[start];
        int low = start, high = end;
        while (start < end){
            while (start < end && nums[end] >= pivot){
                end --;
            }
            nums[start] = nums[end];

            while (start < end && nums[start] <= pivot){
                start ++;
            }
            nums[end] = nums[start];
        }
        nums[start] = pivot;
        if (start == target){
            return pivot;
        }
        else if (start > target){
            return partition(nums, low, start - 1, target);
        }
        else{
            return partition(nums, start + 1, high, target);
        }

    }
};

// Your MedianFinder object will be instantiated and called as such:
// MedianFinder mf = new MedianFinder();
// mf.addNum(1);
// mf.findMedian();
```

Sorted array may make getting median faster(but inserting elements to container slower). Inserting an element to the array and keep it sorted needs $O(1)runtime$, but getting median only need $O(1)runtime$. This solution gets AC in OJ.

Listing 305: Solution 1.2 - Sorted Array

```
public class MedianFinder {
    static final int MAX_LEN = 100000;
    int[] nums = new int[MAX_LEN];
    int length = 0;
    // Adds a number into the data structure.
    public void addNum(int num) {
        int pos = 0;
```

```
        for (int idx = length - 1 ; idx >= 0 ; idx --){
            if (nums[idx] > num){
                nums[idx + 1] = nums[idx];
            }
            else{
                pos = idx + 1;
                break;
            }
        }
        nums[pos] = num;
        length ++;
    }

    // Returns the median of current data stream
    public double findMedian() {
        if (length % 2 == 1){
            return nums[length / 2] * 1.0;
        }
        else{
            return (nums[length / 2 - 1] + nums[length / 2]) * 1.0 / 2;
        }
    }
};

// Your MedianFinder object will be instantiated and called as such:
// MedianFinder mf = new MedianFinder();
// mf.addNum(1);
// mf.findMedian();
```

What's more, we can use Sorted List to solve the problem, adding 2 pointers pointing at 2 medians can reduce the complexity of getting median to $O(1)$(when the length is odd, the pointers point at the same element.)

Solution 2 - Heaps(Max & Min) in form of PriorityQueue

Solution is more proper to solve the problem.

Maintain 2 heaps, one max and another min. Keep them have the same size or max heap is 1 element larger than min heap in size. We store the biggest k elements in min heap while the others(k OR k + 1) in max heap. Then either the median is root of max heap OR average of roots from both trees.

How to maintain their sizes? We always push eleme

Listing 306: Solution 2 - Heaps

```
public class MedianFinder {
    PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>();
    PriorityQueue<Integer> maxHeap =
     new PriorityQueue<Integer>(10000, new Comparator<Integer>(){
        @Override
        public int compare(Integer x, Integer y){
            return y - x;
        }
    });
    // Adds a number into the data structure.
    public void addNum(int num) {
        if (minHeap.size() == maxHeap.size()){
```

```
            maxHeap.offer(num);
        }
        else{
            minHeap.offer(num);
        }
        if (!minHeap.isEmpty() && (maxHeap.peek() > minHeap.peek())){
            minHeap.offer(maxHeap.poll());
            maxHeap.offer(minHeap.poll());
        }
    }

    // Returns the median of current data stream
    public double findMedian() {
        if ((minHeap.size() + maxHeap.size()) % 2 == 0){
            return (maxHeap.peek() + minHeap.peek()) * 1.0 / 2;
        }
        else{
            return maxHeap.peek();
        }
    }
};

// Your MedianFinder object will be instantiated and called as such:
// MedianFinder mf = new MedianFinder();
// mf.addNum(1);
// mf.findMedian();
```

Solution3 - Trees(BST, Binary Search Tree OR AVL Tree)

Since in Java there are no built-in structures similar to BST or AVL, so I won't list the code here, just think of how to finish out task.

We may add a field in TreeNode to make searching for median faster(to $O(logn)$).

To avoid the worst condition of BST, we may use AVL- the highly balanced tree structure to finish what BST need to do.

In summary, all sorts of methods can be concluded as figure below,

Type of Data Container	Time to Insert	Time to Get Median
Unsorted Array	O(1)	O(n)
Sorted Array	O(n)	O(1)
Sorted List	O(n)	O(1)
Binary Search Tree	O(log n) on average, O(n) for the worst case	O(log n) on average, O(n) for the worst case
AVL	O(log$n$)	O(1)
Max and Min Heap	O(log$n$)	O(1)

Problem 233

17 Letter Combinations of a Phone Number

```
Given a digit string, return all possible letter combinations
 that the number could represent.

A mapping of digit to letters (just like on the telephone buttons)
 is given below.



Input:Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
Note:
Although the above answer is in lexicographical order,
 your answer could be in any order you want.

https://leetcode.com/problems/letter-combinations-of-a-phone-number/
```

Solution

A backtracking problem, similar to Problem "Combinations".

Listing 307: Solution

```
public class Solution {
    String[] mapping = {"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
    List<String> result = new ArrayList<String>();

    public List<String> letterCombinations(String digits) {
        traversePhone(digits, 0, new ArrayList<Character>());
        return result;
    }

    private void traversePhone(String digits, int depth, List<Character> list){
        if (digits.length() == 0){
            return;
        }
        if (depth == digits.length()){
            String newStr = "";
            for (char ele : list){
                newStr += ele;
            }
            result.add(newStr);
            return;
```

```
        }
        String possibilities = mapping[digits.charAt(depth) - '0'];
        if (possibilities.length() == 0){
            return;
        }
        else{
            for (int idx = 0 ; idx < possibilities.length() ; idx ++){
                list.add(possibilities.charAt(idx));
                traversePhone(digits, depth + 1, list);
                list.remove(list.size() - 1);
            }
        }

    }
}
```

Problem 234

71 Simplify Path

```
Given an absolute path for a file (Unix-style), simplify it.

For example,
path = "/home/", => "/home"
path = "/a/./b/../../c/", => "/c"
click to show corner cases.

Corner Cases:
    Did you consider the case where path = "/../"?
    In this case, you should return "/".
    Another corner case is the path might
     contain multiple slashes '/' together, such as "/home//foo/".
    In this case, you should ignore redundant slashes
     and return "/home/foo".

https://leetcode.com/problems/simplify-path/
```

Solution

It's obvious that the problem is proper to be solved using Stack, there are 2 things to be taken into consideration:

1. Mind those corner cases.(See detailed cases in problem description)

2. Don't forget to reverse those remaining paths in Stack, /home/test will be in the order of "test $->$ home" in Stack.

Listing 308: Solution

```
public class Solution {
    public String simplifyPath(String path) {
        if (path.length() == 0){
            return "/";
        }
        String result = "";
        Stack<String> stack = new Stack<String>();
        int startIdx = 0;

        while (startIdx < path.length() && path.charAt(startIdx) == '/'){
            startIdx ++;
        }

        while (startIdx < path.length()){
            int curIdx = startIdx;
            while (curIdx < path.length() && path.charAt(curIdx) != '/'){
                curIdx ++;
            }
            int endIdx = curIdx;
            String curNode = path.substring(startIdx, endIdx);
            if (!curNode.equals(".")){
                if (curNode.equals("..")){
                    if (!stack.isEmpty()){
```

```
                        stack.pop();
                    }
                }
                else{
                    stack.push(curNode);
                }
            }
            while (curIdx < path.length() && path.charAt(curIdx) == '/'){
                curIdx ++;
            }
            startIdx = curIdx;
        }
        Stack<String> resultStack = new Stack<String>();
        while (!stack.isEmpty()){
            resultStack.push(stack.pop());
        }
        while (!resultStack.isEmpty()){
            result += "/";
            result += resultStack.pop();
        }
        if (result.length() == 0){
            result += "/";
        }

        return result;
    }
}
```

Problem 235

22 Generate Parentheses

```
Given n pairs of parentheses,
 write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

https://leetcode.com/problems/generate-parentheses/
```

Solution

An easy backtracking problem, when remaining left parentheses number equals right number, we must pick left one so that there won't be illegal pairs; when left less than right, we can pick either left or right parentheses.

Listing 309: Solution

```
public class Solution {
    List<String> result = new ArrayList<String>();
    private void dfs(String curStr, int leftRem, int rightRem){
        if (leftRem == 0 && rightRem == 0){
            result.add(curStr);
            return;
        }
        if (leftRem > rightRem){
            return;
        }
        else if (leftRem == rightRem){
            dfs(curStr + "(" , leftRem - 1, rightRem);
        }
        else{
            // leftRem < rightRem
            if (leftRem > 0){
                dfs(curStr + "(", leftRem - 1, rightRem);
            }
            dfs(curStr + ")" , leftRem, rightRem - 1);
        }
    }
    public List<String> generateParenthesis(int n) {
        dfs("", n, n);
        return result;
    }
}
```

Problem 236

338 Counting Bits

```
Given a non negative integer number num.
 For every numbers i in the range 0 <= i <= num
  calculate the number of 1's in their binary representation and
   return them as an array.

Example:
For num = 5 you should return [0,1,1,2,1,2].

Follow up:

It is very easy to come up with a solution
 with run time O(n*sizeof(integer)).
  But can you do it in linear time O(n) /possibly in a single pass?
Space complexity should be O(n).
Can you do it like a boss? Do it without using
 any builtin function like __builtin_popcount in c++ or in any other language.
Hint:

You should make use of what you have produced already.
Divide the numbers in ranges like [2-3], [4-7], [8-15] and so on.
 And try to generate new range from previous.
Or does the odd/even status of the number
 help you in calculating the number of 1s?

https://leetcode.com/problems/counting-bits/
```

Solution

There is a cycle within all numbers, [[[[0],1],10,11],100,101,110,111], eg, for 0,1,10,11, the 100,101,110,111 is just adding 1 in front of them.

Listing 310: Solution 1.1

```
public class Solution {
    public int[] countBits(int num) {
        int[] result = new int[num + 1];
        if (num == 0){
            return result;
        }
        int tail = 1;
        result[0] = 0;
        int count = 0;
        while (tail <= num){
            int prevTail = tail;
            for (int i = 0 ; i < prevTail ; i ++){
                result[tail ++] = result[i] + 1;
                if (tail > num){
                    return result;
                }
            }
        }
```

Problem 236 continued on next page...

```
        return result;
    }
}
```

Another solution, the train of thought is similar in nature with mine, simpler.

Listing 311: Solution 1.2

```
public int[] countBits(int num) {
    int[] result = new int[num + 1];
    result[0] = 0;
    if(num == 0){ return result;}
    int leftMostOne = 1; // will increase as 2, 4, 8, 16, 32 ....
    for(int i = 1; i <= num; i++){
        if(i == 2 * leftMostOne){ leftMostOne = i;}
        result[i] = 1 + result[i - leftMostOne];
    }
    return result;
}
```

Problem 237

299 Bulls and Cows

```
You are playing the following Bulls and Cows game with your friend:
 You write down a number and ask your friend to guess what the number is.
  Each time your friend makes a guess, you provide a hint that indicates
   how many digits in said guess match your secret number exactly
    in both digit and position (called "bulls") and
     how many digits match the secret number but
      locate in the wrong position (called "cows").
       Your friend will use successive guesses and hints to eventually
        derive the secret number.

For example:

    Secret number:  "1807"
    Friend's guess: "7810"
Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.)
Write a function to return a hint according to the secret number
 and friend's guess, use A to indicate the bulls and B to indicate the cows.
  In the above example, your function should return "1A3B".

Please note that both secret number and friend's guess may
 contain duplicate digits, for example:

    Secret number:  "1123"
    Friend's guess: "0111"
In this case, the 1st 1 in friend's guess is a bull,
 the 2nd or 3rd 1 is a cow, and your function should return "1A1B".
You may assume that the secret number and your friend's guess only
 contain digits, and their lengths are always equal.

https://leetcode.com/problems/bulls-and-cows/
```

Solution

I'm following an easy simulation, deal with 'A' & 'B' separately, but I think there is space for optimization. Last FREE Easy Problem solved till 3rd, Apr., 2016.

Listing 312: Solution

```
public class Solution {
    public String getHint(String secret, String guess) {
        int[] count = new int[10];
        boolean[] visited = new boolean[secret.length()];
        int cntA = 0, cntB = 0;
        for (int i = 0 ; i < secret.length() ; i ++){
            if (secret.charAt(i) == guess.charAt(i)){
                cntA ++;
                visited[i] = true;
            }
            else{
                count[secret.charAt(i) - '0'] ++;
            }
```

```
        }
        for (int i = 0 ; i < secret.length() ; i ++){
            if (!visited[i]){
                char guessChar = guess.charAt(i);
                if (count[guessChar - '0'] > 0){
                    count[guessChar - '0']--;
                    cntB ++;
                }
            }
        }

        return cntA + "A" + cntB + "B";
    }
}
```

Problem 238

162 Find Peak Element

```
A peak element is an element that is greater than its neighbors.

Given an input array where num[i] != num[i+1],
 find a peak element and return its index.

The array may contain multiple peaks,
 in that case return the index to any one of the peaks is fine.

You may imagine that num[-1] = num[n] = - infinity(since I cannot type the symbol here)

For example, in array [1, 2, 3, 1], 3 is a peak element
 and your function should return the index number 2.

Note:
Your solution should be in logarithmic complexity.

https://leetcode.com/problems/find-peak-element/
```

Solution 1

$O(n)$ method, simple simulation, sequential method.

Listing 313: Solution 1

```
public class Solution {
    public int findPeakElement(int[] nums) {
        if (nums.length == 0){
            return -1;
        }
        for (int i = 0 ; i < nums.length ; i++){
            if (i == 0 && i == nums.length - 1){
                return 0;
            }
            else if (i == 0){
                if (nums[0] > nums[1]){
                    return 0;
                }
            }
            else if (i == nums.length - 1){
                if (nums[i] > nums[i - 1]){
                    return i;
                }
            }
            else{
                if (nums[i] > nums[i - 1] && nums[i] > nums[i + 1]){
                    return i;
                }
            }
        }
        return -1;
    }
```

```
}
```

Solution 2

Binary search will reduce the complexity to $O(logn)$.

Listing 314: Solution 2

```
public class Solution {
    public int findPeakElement(int[] nums) {
        int low = 0;
        int high = nums.length - 1;
        while (low < high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] > nums[mid + 1])
                high = mid;
            else
                low = mid + 1;
        }
        return low;
    }
}
```

Problem 239

318 Maximum Product of Word Lengths

```
Given a string array words, find the maximum value of length
(word[i]) * length(word[j]) where the two words do not share common letters.
 You may assume that each word will contain only lower case letters.
  If no such two words exist, return 0.

Example 1:
Given ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]
Return 16
The two words can be "abcw", "xtfn".

Example 2:
Given ["a", "ab", "abc", "d", "cd", "bcd", "abcd"]
Return 4
The two words can be "ab", "cd".

Example 3:
Given ["a", "aa", "aaa", "aaaa"]
Return 0
No such pair of words.

https://leetcode.com/problems/maximum-product-of-word-lengths/
```

Solution

Use 26-bit integers to represent each word(since a & aa means the same to us in this problem). For example, 'abd' is 0000..001011, then we can know if two integers carry out bit and & calculation and get a non-zero result, the two have some letters in common and vice versa.

Listing 315: Solution

```
public class Solution {
    public int maxProduct(String[] words) {
        int[] bitMap = new int[words.length];
        for (int idx = 0 ; idx < words.length; idx ++){
            String word = words[idx];
            for(int strIdx = 0 ; strIdx < word.length() ; strIdx ++){
                char curChar = word.charAt(strIdx);
                bitMap[idx] |= 1 << curChar - 'a';
            }
        }
        int max = 0;
        for (int i = 0 ; i < words.length ; i++){
            for (int j = i + 1 ; j < words.length ; j ++){
                if ((bitMap[i] & bitMap[j]) == 0){
                    max = Math.max(words[i].length() * words[j].length(), max);
                }
            }
        }
        return max;
    }
}
```

Problem 239 continued on next page...

Problem 240

164 Maximum Gap

```
Given an unsorted array,
 find the maximum difference between the successive elements
  in its sorted form.

Try to solve it in linear time/space.

Return 0 if the array contains less than 2 elements.

You may assume all elements in the array are
 non-negative integers and fit in the 32-bit signed integer range.

https://leetcode.com/problems/maximum-gap/
```

Solution

Hard problem is worthy of its name... Though this problem is not so difficult to use Bucket Sort, there are so many edge cases and annoying index calculation.

Back to the problem, the maximum gap can only appear between maximum and minimum value of 2 non-empty adjacent buckets, we can use this idea to solve the problem.

Listing 316: Solution

```
class Bucket{
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    int size = 0;
    List<Integer> list = new ArrayList<Integer>();
}
public class Solution {
    public int maximumGap(int[] nums) {
        if (nums.length < 2){
            return 0;
        }
        if (nums.length == 2){
          return Math.abs(nums[0] - nums[1]);
        }
        int totalMax = Integer.MIN_VALUE;
        int totalMin = Integer.MAX_VALUE;
        for (int i = 0 ; i < nums.length ; i ++){
            totalMax = Math.max(totalMax, nums[i]);
            totalMin = Math.min(totalMin, nums[i]);
        }

        int interval = (int) Math.ceil((double)(totalMax + 1 - totalMin)
             / (nums.length - 1));
        int bucketCnt = (int) Math.ceil((double)
                (totalMax + 1 - totalMin) / interval);
        Bucket[] buckets = new Bucket[bucketCnt];
        for (int i = 0 ; i < buckets.length ; i ++){
            buckets[i] = new Bucket();
```

```
        }

        if (interval == 0){
          return 0;
        }
        for (int i = 0 ; i < nums.length ; i ++){
            int idx = (nums[i] - totalMin) / interval;
            buckets[idx].size ++;
            buckets[idx].list.add(nums[i]);
            buckets[idx].max = Math.max(buckets[idx].max, nums[i]);
            buckets[idx].min = Math.min(buckets[idx].min, nums[i]);
        }

        int maxGap = 0;
        int lastMax = Integer.MIN_VALUE;
        for (int i = 0 ; i < buckets.length ; i ++){
            if (buckets[i].size == 0){
                continue;
            } else{
                if (lastMax == Integer.MIN_VALUE){
                    lastMax = buckets[i].max;
                }else{
                    maxGap = Math.max(maxGap, buckets[i].min - lastMax);
                    lastMax = buckets[i].max;
                }
            }
        }

        return maxGap;
    }
}
```

Problem 241

332 Reconstruct Itinerary

```
Given a list of airline tickets represented by
 pairs of departure and arrival airports [from, to],
  reconstruct the itinerary in order.
   All of the tickets belong to a man who departs from JFK.
    Thus, the itinerary must begin with JFK.

Note:
    If there are multiple valid itineraries,
     you should return the itinerary that has the smallest lexical order
      when read as a single string.
       For example,
        the itinerary ["JFK", "LGA"] has a smaller lexical
         order than ["JFK", "LGB"].
    All airports are represented by three capital letters (IATA code).
    You may assume all tickets form at least one valid itinerary.

Example 1:
    tickets
         = [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]
    Return ["JFK", "MUC", "LHR", "SFO", "SJC"].
Example 2:
    tickets
         = [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]
    Return ["JFK","ATL","JFK","SFO","ATL","SFO"].
    Another possible reconstruction is ["JFK","SFO","ATL","JFK","ATL","SFO"].
     But it is larger in lexical order.

https://leetcode.com/problems/reconstruct-itinerary/
```

Solution

A clear DFS problem in adjacency graph. However, I truly struggled for a long while on the 'smallest lexical order'. So sort them!

Listing 317: Solution

```
public class Solution {
    public List<String> findItinerary(String[][] tickets) {
        if (tickets.length == 0) {
            return new ArrayList<String>();
        }
        Map<String, List<String>> map = new HashMap<String, List<String>>();
        for (String[] ticket : tickets) {
            String from = ticket[0], to = ticket[1];
            List<String> oldList = map.containsKey(from) ? map.get(from)
                    : new ArrayList<String>();
            oldList.add(to);
            map.put(from, oldList);
        }
        for (String key : map.keySet()) {
            Collections.sort(map.get(key));
```

```
        }
        List<String> result = new ArrayList<String>();
        exploreItinerary(map, "JFK", result, tickets.length + 1);
        return result;
    }

    private boolean exploreItinerary(Map<String, List<String>> map,
            String curPort, List<String> result, int targetSize) {
        result.add(curPort);
        if (result.size() == targetSize) {
            return true;
        }
        if (map.get(curPort) != null){
            // Note: you can remove elements when iterating using index
            for (int idx = 0 ; idx < map.get(curPort).size() ; idx ++) {
                String to = map.get(curPort).get(idx);
                map.get(curPort).remove(idx);
                if (exploreItinerary(map, to, result, targetSize)) {
                    return true;
                }
                map.get(curPort).add(to);
                Collections.sort(map.get(curPort));
            }
        }
        result.remove(result.size() - 1);
        return false;
    }
}
```

Problem 242

91 Decode Ways

```
A message containing letters from A-Z is
 being encoded to numbers using the following mapping:

'A' -> 1
'B' -> 2
...
'Z' -> 26
Given an encoded message containing digits,
 determine the total number of ways to decode it.

For example,
Given encoded message "12", it could be decoded as
 "AB" (1 2) or "L" (12).

The number of ways decoding "12" is 2.

https://leetcode.com/problems/decode-ways/
```

Solution

dp[i] = ((last_digit==0) ? dp[i - 1] : 0) + ((10<=last_2_digits<=26) ? dp[i - 2] : 0);

Listing 318: Solution

```
public class Solution {
    public int numDecodings(String s) {
        if (s.length() == 0 || s.charAt(0) == '0'){
            return 0;
        }
        if(s.length() == 1){
            return 1;
        }
        int[] dp = new int[s.length()];
        dp[0] = 1;
        if (Integer.parseInt(s.substring(0, 2)) <= 26
             && Integer.parseInt(s.substring(0, 2)) >= 10){
            if (Integer.parseInt(s.substring(0, 2)) % 10 == 0)
                dp[1] = 1;
            else
                dp[1] = 2;
        }
        else{
            if (Integer.parseInt(s.substring(0, 2)) % 10 == 0)
                dp[1] = 0;
            else
                dp[1] = 1;
        }

        for (int idx = 2; idx < s.length() ; idx ++){
            int singleDigit = Integer.parseInt(s.substring(idx , idx + 1));
            int doubleDigits = Integer.parseInt(s.substring(idx - 1 , idx + 1));
```

Problem 242 continued on next page...

```
            if (singleDigit != 0){
                dp[idx] += dp[idx - 1];
            }
            if ((doubleDigits >= 10) && (doubleDigits <= 26)){
                dp[idx] += dp[idx - 2];
            }
        }

        return dp[s.length() - 1];
    }
}
```

Problem 243

166 Fraction to Recurring Decimal

```
Given two integers representing the numerator and
 denominator of a fraction, return the fraction in string format.

If the fractional part is repeating,
 enclose the repeating part in parentheses.

For example,

    Given numerator = 1, denominator = 2, return "0.5".
    Given numerator = 2, denominator = 1, return "2".
    Given numerator = 2, denominator = 3, return "0.(6)".

Hint:

No scary math, just apply elementary math knowledge.
 Still remember how to perform a long division?
Try a long division on 4/9, the repeating part is obvious.
 Now try 4/333. Do you see a pattern?
Be wary of edge cases! List out as many test cases
 as you can think of and test your code thoroughly.

https://leetcode.com/problems/fraction-to-recurring-decimal/
```

Solution

Use a HashMap to store existing states. Not every digit but every remainders, for example, $1/7 -> (1 . 3(10\%7) . 2(30\%7) . 6(20\%7) . 4(60\%7) . 5(40\%7)) . (1(50\%7) ....$

Listing 319: Solution

```
public class Solution {
    public String fractionToDecimal(int numerator, int denominator) {
        if (numerator == 0){
            return "0";
        }
        if (denominator == 0){
            return "";
        }
        String result = "";
        if ((numerator < 0 && denominator > 0)
             || (numerator > 0 && denominator < 0)){
            result += "-";
        }
        long numL = Math.abs(numerator);
        if (numerator == Integer.MIN_VALUE){
            numL = (long)(Integer.MAX_VALUE) + 1;
        }
        long denL = Math.abs(denominator);
        if (denominator == Integer.MIN_VALUE){
            denL = (long)(Integer.MAX_VALUE) + 1;
        }
```

```
        long intL = numL / denL;
        long remL = numL % denL;

        result += intL;
        if (remL == 0){
            return result;
        }
        else{
            result += ".";
        }

        Map<Long, Integer> remPos = new HashMap<Long, Integer>();
        // remPos.put(remL, 0);
        String decimal = "";
        int count = 0;
        while (remL != 0){
            if (remPos.containsKey(remL)){
                int latterIdx = remPos.get(remL);
                String part1 = decimal.substring(0, latterIdx);
                String part2 = decimal.substring(latterIdx);
                result += part1 + "(" + part2 + ")";
                return result;
            }
            remPos.put(remL, count ++);
            remL *= 10;
            intL = remL / denL;
            decimal += intL;
            remL = remL % denL;
        }
        result += decimal;
        return result;
    }
}
```

Problem 244

301 Remove Invalid Parentheses

```
Remove the minimum number of invalid parentheses in order to
 make the input string valid. Return all possible results.

Note: The input string may contain letters other than
 the parentheses ( and ).

Examples:
    "()())()" -> ["()()()", "(())()"]
    "(a)())()" -> ["(a)()()", "(a())()"]
    ")(" -> [""]

https://leetcode.com/problems/remove-invalid-parentheses/
```

Solution - BFS

The problem can be solved in both DFS & BFS way.
Basic train of thought is to remove every character in string iteratively to check if generated string is valid.
For BFS, it is a natural idea when we want to find the minimum of invalid parentheses.

Listing 320: Solution 1

```
public class Solution {
    public List<String> removeInvalidParentheses(String s) {
        Queue<String> queue = new LinkedList<String>();
        queue.offer(s);
        Set<String> visited = new HashSet<String>();
        Set<String> queueVisited = new HashSet<String>();
        List<String> result = new ArrayList<String>();
        if (isValid(s)){
            result.add(s);
            return result;
        }
        boolean found = false;
        while (!found && !queue.isEmpty()){
            List<String> curLevel = new ArrayList<String>();
            while (!found && !queue.isEmpty()){
                curLevel.add(queue.poll());
            }

            for (String cur : curLevel){
                for (int splitIdx = 0 ; splitIdx < cur.length() ; splitIdx ++){
                    if (cur.charAt(splitIdx) != '('
                     && cur.charAt(splitIdx) != ')'){
                        continue;
                    }
                    String newStr = cur.substring(0, splitIdx)
                                + cur.substring(splitIdx + 1);
                    if (queueVisited.contains(newStr)){
                        continue;
                    }
```

```
                    if (isValid(newStr)){
                        if (!visited.contains(newStr)){
                            visited.add(newStr);
                            result.add(newStr);
                            found = true;
                        }
                    }
                    queueVisited.add(newStr);
                    queue.offer(newStr);
                }
            }
        }

        if (result.size() == 0){
            result.add("");
        }
        return result;
    }
    private boolean isValid(String str){
        int count = 0;
        for (int idx = 0 ; idx < str.length() ; idx ++){
            if (str.charAt(idx) == '('){
                count ++;
            }
            if (str.charAt(idx) == ')'){
                if (count == 0){
                    return false;
                }
                count --;
            }
        }

        return count == 0;
    }
}
```

Solution - DFS

The idea is searched from Internet, copied here,

Key Points:

Generate unique answer once and only once, do not rely on Set.

Do not need preprocess.

Runtime 3 ms.

Explanation: We all know how to check a string of parentheses is valid using a stack. Or even simpler use a counter. The counter will increase when it is '(' and decrease when it is ')'. Whenever the counter is negative, we have more ')' than '(' in the prefix.

To make the prefix valid, we need to remove a ')'. The problem is: which one? The answer is any one in the prefix. However, if we remove any one, we will generate duplicate results, for example: s = ()), we can remove s[1] or s[2] but the result is the same (). Thus, we restrict ourself to remove the first) in a series of concecutive)s.

After the removal, the prefix is then valid. We then call the function recursively to solve the rest of the string. However, we need to keep another information: the last removal position. If we do not have this

position, we will generate duplicate by removing two ')' in two steps only with a different order. For this, we keep tracking the last removal position and only remove ')' after that.

Now one may ask. What about '('? What if s = '(()(()' in which we need remove '('? The answer is: do the same from right to left. However a cleverer idea is: reverse the string and reuse the code! Here is the final implement in Java.

https://leetcode.com/discuss/81478/easy-short-concise-and-fast-java-dfs-3-ms-solution

Listing 321: Solution 2

```java
public class Solution {
    public List<String> removeInvalidParentheses(String s) {
        List<String> result = new ArrayList<String>();
        dfs(s, 0, 0, new char[] {'(',')'}, result);
        return result;
    }

    private void dfs(String s, int prevI, int prevJ, char[] flag,
                 List<String> result){
        int halfCnt = 0;
        for (int i = prevI ; i < s.length() ; i ++){
            if (s.charAt(i) == flag[0]){
                halfCnt ++;
            }
            if (s.charAt(i) == flag[1]){
                halfCnt --;
            }
            if (halfCnt >= 0){
                continue;
            }
            for (int j = prevJ ; j <= i ; j ++){
                if (s.charAt(j) == flag[1]
                     && (j == prevJ || flag[1] != s.charAt(j - 1))){
                    dfs(s.substring(0, j) + s.substring(j + 1, s.length()),
                         i, j, flag, result);
                }
            }
            // I don't quite understand why here is a 'return' statement
            return;
        }

        String reversed = new StringBuilder(s).reverse().toString();
        if (flag[0] == '('){
            dfs(reversed, 0, 0, new char[] {')','('}, result);
        } else{
            result.add(reversed);
        }
    }
}
```

Problem 245

201 Bitwise AND of Numbers Range

```
Given a range [m, n] where 0 <= m <= n <= 2147483647,
 return the bitwise AND of all numbers in this range, inclusive.

For example, given the range [5, 7], you should return 4.

https://leetcode.com/problems/bitwise-and-of-numbers-range/
```

Solution

The problem is really interesting, it can finally be transformed to a problem to find common prefix of m and n in binary representation and fill 0 to the end to cover the position.

Listing 322: Solution 1.1

```
public class Solution {
    public int rangeBitwiseAnd(int m, int n) {
        int bit = 0;
        while(m != n) {
            m >>= 1;
            n >>= 1;
            bit ++;
        }
        return m << bit;
    }
}

// OR


public class Solution {
    public int rangeBitwiseAnd(int m, int n) {
        int mask = Integer.MAX_VALUE;
        while((mask&m)!=(mask&n)){
            mask=mask<<1;
        }
        return mask&m;
    }
}
```

Another solution is as below, which is in nature the same as previous method.

Listing 323: Solution 1.2

```
public class Solution {
    public int rangeBitwiseAnd(int m, int n) {
        while(n > m){
            n = n & (n - 1);
        }
        return n;
    }
}
```

Problem 246

151 Reverse Words in a String

```
Given an input string, reverse the string word by word.

For example,
Given s = "the sky is blue",
return "blue is sky the".

Update (2015-02-12):
For C programmers: Try to solve it in-place in O(1) space.


Clarification:
   Q: What constitutes a word?
   A: A sequence of non-space characters constitutes a word.
   Q: Could the input string contain leading or trailing spaces?
   A: Yes. However, your reversed string should not contain
         leading or trailing spaces.
   Q: How about multiple spaces between two words?
   A: Reduce them to a single space in the reversed string.

https://leetcode.com/problems/reverse-words-in-a-string/
```

Solution

The $O(n)$ solution(can only in C since C regards string as char array). Reverse each word, and then reverse the whole sentence.

Listing 324: Solution 1.1

```
public class Solution {
    public String reverseWords(String s) {
        s = reverse(s);
        int curIdx = 0;
        String revStr = "";
        while (curIdx < s.length()) {
            while (curIdx < s.length() && s.charAt(curIdx) == ' ') {
                curIdx++;
            }
            if (curIdx == s.length()) {
                break;
            }
            int startIdx = curIdx;
            while (curIdx < s.length() && s.charAt(curIdx) != ' ') {
                curIdx++;
            }
            int endIdx = curIdx;
            String newPart = reverse(s.substring(startIdx, endIdx));
            revStr += newPart;

            while (curIdx < s.length() && s.charAt(curIdx) == ' ') {
                curIdx++;
            }
```

```
            if (curIdx < s.length()) {
                revStr += " ";
            }
        }
        return revStr;
    }

    public String reverse(String s) {
        char[] array = s.toCharArray();
        int start = 0, end = s.length() - 1;
        while (start < end) {
            char tmp = array[start];
            array[start] = array[end];
            array[end] = tmp;
            start++;
            end--;
        }
        return new String(array);
    }
}
```

Well, in 'Discuss' section, solutions make use of built-in functions, which I think is not that proper....(But really concise and short).

Listing 325: Solution 1.2

```
public String reverseWords(String s) {
    String[] temp = s.split(" +");
    StringBuilder sb = new StringBuilder();
    for(int i = temp.length - 1; i >= 0;i --){
        sb.append(temp[i].trim()+" ");
    }
    return sb.toString().trim();
}
```

Problem 247

128 Longest Consecutive Sequence

```
Given an unsorted array of integers,
 find the length of the longest consecutive elements sequence.

For example,
Given [100, 4, 200, 1, 3, 2],
The longest consecutive elements sequence is [1, 2, 3, 4].
 Return its length: 4.

Your algorithm should run in O(n) complexity.

https://leetcode.com/problems/longest-consecutive-sequence/
```

Solution

Brute force method is to use Sorting methods to sort the array first, but with at least $O(nlogn)$,

For $O(n)$ solution, not so difficult, just use a Set to store all elements in the array, iterate each of them and find if there are ascending and descending elements to update the maximum consecutive length, and remove them from Set to avoid duplicate calculation.

Listing 326: Solution

```
public class Solution {
    public int longestConsecutive(int[] nums) {
        if (nums.length == 0)
            return 1;
        Set<Integer> set = new HashSet<Integer>();
        for (int num : nums)
            set.add(num);
        int maxLen = 0;
        for (int num : nums){
            if (!set.contains(num))
                continue;
            set.remove(num);
            int len = 1;
            int cur = num;
            while (set.contains(--cur)){
                len ++;
                set.remove(cur);
            }
            cur = num;
            while (set.contains(++cur)){
                len ++;
                set.remove(cur);
            }
            maxLen = Math.max(maxLen , len);
        }
        return maxLen;
    }
}
```

Problem 248

312 Burst Balloons

```
Given n balloons, indexed from 0 to n-1.
 Each balloon is painted with a number on it represented by array nums.
  You are asked to burst all the balloons.
   If the you burst balloon i you will get
    nums[left] * nums[i] * nums[right] coins.
     Here left and right are adjacent indices of i.
      After the burst, the left and right then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Note:
(1) You may imagine nums[-1] = nums[n] = 1.
 They are not real therefore you can not burst them.
(2) 0 <= n <= 500, 0 <= nums[i] <= 100

Example:

Given [3, 1, 5, 8]

Return 167

    nums = [3,1,5,8] --> [3,5,8] -->   [3,8]   -->  [8]  --> []
   coins =  3*1*5      +  3*5*8    +  1*3*8      + 1*8*1   = 167

https://leetcode.com/problems/burst-balloons/
```

Solution

A excellent DP problem.

The state transition equation is dp[l][r]=nums[l]*nums[m]*nums[r] + dp[l][m] + dp[m][r], l<m<r;

For detailed solution description, see
http://bookshadow.com/weblog/2015/11/30/leetcode-burst-balloons/

Listing 327: Solution

```
public class Solution {
    public int maxCoins(int[] nums) {
        int[] totalNums = new int[nums.length + 2];
        totalNums[0] = totalNums[totalNums.length - 1] = 1;
        for (int i = 0 ; i < nums.length ; i ++){
            totalNums[i + 1] = nums[i];
        }
        if (nums.length == 0){
            return 0;
        }
        int dp[][] = new int[totalNums.length][totalNums.length];
        for (int len = 2 ; len < totalNums.length ; len ++){
            for (int start = 0 ; start < totalNums.length - len ; start ++){
                int end = start + len;
                for (int mid = start + 1 ; mid < end ; mid ++){
```

```
                    dp[start][end] = Math.max(dp[start][end],
                        totalNums[start] * totalNums[mid] * totalNums[end]
                         + dp[start][mid] + dp[mid][end]);
                }
            }
        }

        return dp[0][totalNums.length - 1];
    }
}
```

Problem 249

330 Patching Array

```
Given a sorted positive integer array nums and an integer n,
 add/patch elements to the array such that any number
  in range [1, n] inclusive can be formed by the sum of
   some elements in the array.
    Return the minimum number of patches required.

Example 1:
    nums = [1, 3], n = 6
    Return 1.

    Combinations of nums are [1], [3], [1,3],
     which form possible sums of: 1, 3, 4.
    Now if we add/patch 2 to nums, the combinations are:
     [1], [2], [3], [1,3], [2,3], [1,2,3].
    Possible sums are 1, 2, 3, 4, 5, 6,
     which now covers the range [1, 6].
    So we only need 1 patch.

Example 2:
    nums = [1, 5, 10], n = 20
    Return 2.
    The two patches can be [2, 4].

Example 3:
    nums = [1, 2, 2], n = 5
    Return 0.

https://leetcode.com/problems/patching-array/
```

Solution

The idea is not so easy to figure out.

First, we sort the list from smallest to largest.

Then iterate the list(or iterate through 1 to required max-sum).

We maintain a maximum sum we can reach currently.

When we meet nums[i] <= sum, eg: [1,2,10], curSum = 0 at first. 1 <= 0+1, then curSum += 1; then 2 <= 1+1, curSum += 2;

When nums[i] > sum, eg, curSum=4 & nums[i]=10 in previous example, now we cannot reach sum more than curSum, we then add curSum+1 to out list to get more conditions for sum. What is the maximum sum we can reach now? The answer is curSum*2+1(2*(curSum+1) - 1).

Note that a common pitfall is that sum may exceed max int, so we use long instead of int. (Consider the test case [1,2,31,33], 2147483647)

Listing 328: Solution

```
public class Solution {
    public int minPatches(int[] nums, int n) {
        int count = 0;
        // Arrays.sort(nums);
        long maxSum = 0;
```

```
        for (int i = 0 ; i < nums.length ; i ++){
            if (nums[i] <= maxSum + 1){
                maxSum += nums[i];
            }
            else{
                while (nums[i] > maxSum + 1){
                    count ++;
                    maxSum = 2 * maxSum + 1;

                    if (maxSum >= n){
                        return count;
                    }
                        if (nums[i] <= maxSum + 1){
                            maxSum += nums[i];
                        }
                    }
                }
                if (maxSum >= n){
                    return count;
                }
            }
            if (maxSum == 0){
                maxSum = 1;
                count ++;
            }
            while (maxSum < n){
                maxSum = 2 * maxSum + 1;
                count++;
            }
            return count;
        }
    }

    // A more concise version
    class Solution {
    public:
        int minPatches(vector<int>& nums, int n) {
            int cnt = 0, i = 0;
            for (long known_sum = 1; known_sum <= n;) {
                if (i < nums.size() && nums[i] <= known_sum) {
                    known_sum += nums[i++];
                }
                else {
                    known_sum <<= 1;
                    cnt++;
                }
            }
            return cnt;
        }
    };
```

Problem 250

306 Additive Number

```
Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain at least three numbers.
 Except for the first two numbers,
  each subsequent number in the sequence must be the sum of the preceding two.

For example:
"112358" is an additive number because the digits can
 form an additive sequence: 1, 1, 2, 3, 5, 8.
    1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

"199100199" is also an additive number, the additive sequence is:
     1, 99, 100, 199.
    1 + 99 = 100, 99 + 100 = 199

Note: Numbers in the additive sequence cannot have leading zeros,
 so sequence 1, 2, 03 or 1, 02, 3 is invalid.

Given a string containing only digits '0'-'9',
 write a function to determine if it's an additive number.

Follow up:
How would you handle overflow for very large input integers?

https://leetcode.com/problems/additive-number/
```

Solution

Brute force, iterate to get 2 starting adjacent strings, add them until we find(or can't find) correct answer.

But, except for using 'long' type, this solution is not able to handle too large integer.(Or maybe BigInteger or applying addition on long number sequence?)

Listing 329: Solution

```
public class Solution {
    public boolean isAdditiveNumber(String num) {
        for(int i = 1 ; i < num.length() ; i ++){
            for (int j = i + 1 ; j <= num.length() ; j ++){
                String str1 = num.substring(0, i);
                String str2 = num.substring(i, j);
                if (str1.length() > 1 && str1.charAt(0) == '0'){
                    continue;
                }
                if (str2.length() > 1 && str2.charAt(0) == '0'){
                    continue;
                }
                if (str2.equals("")){
                    continue;
                }
                long num1 = Long.parseLong(str1);
```

Problem 250 continued on next page...

```
                long num2 = Long.parseLong(str2);
                long sum = num1 + num2;
                String prefix = str1 + str2;
                if (prefix.equals(num)){
                    continue;
                }

                while (prefix.length() < num.length()){
                    prefix += sum;
                    if (!num.startsWith(prefix)
                            || prefix.length() > num.length()){
                        break;
                    }
                    num1 = num2;
                    num2 = sum;
                    sum = num1 + num2;
                }
                if (num.equals(prefix)){
                    return true;
                }
            }
        }
        return false;
    }
}
```

Problem 251

56 Merge Intervals

```
Given a collection of intervals,
 merge all overlapping intervals.

For example,
    Given [1,3],[2,6],[8,10],[15,18],
    return [1,6],[8,10],[15,18].

https://leetcode.com/problems/merge-intervals/
```

Solution

Sort all intervals according to their start number(write a new Comparator). Then there are 3(or 2, to be more specific) conditions, eg, [1,100] VS [90,101] or [90,99] with [1,10] VS [90,100].

For those cur.end >= next.start, combine the 2 intervals, set cur.end = Math.max(next.end, cur.end); for opposite conditions, set cur = new Node(next.start, next.end);

Listing 330: Solution

```
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public List<Interval> merge(List<Interval> intervals) {
        if (intervals.size() <= 1){
            return intervals;
        }
        List<Interval> result = new ArrayList<Interval>();
        Collections.sort(intervals, new Comparator<Interval>(){
            @Override
            public int compare(Interval i1, Interval i2){
                return i1.start - i2.start;
            }
        });
        Interval cur = new Interval(intervals.get(0).start, intervals.get(0).end);
        for (int idx = 1 ; idx < intervals.size() ; idx ++){
            Interval interval = intervals.get(idx);
            if (cur.end >= interval.start){
                cur.end = Math.max(cur.end, interval.end);
            }
            else{
                result.add(cur);
                cur = new Interval(interval.start, interval.end);
            }
        }
```

```
            result.add(cur);
            return result;
        }
}
```

Problem 252

57 Insert Interval

```
Given a set of non-overlapping intervals,
 insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted
 according to their start times.

Example 1:
Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2:
Given [1,2],[3,5],[6,7],[8,10],[12,16],
 insert and merge [4,9] in as [1,2],[3,10],[12,16].

This is because the new interval [4,9]
 overlaps with [3,5],[6,7],[8,10].

https://leetcode.com/problems/insert-interval/
```

Solution

Iterate every element in former List. Three conditions are waiting:

1. cur.end < new.start, add to list, continue
2. cur.start > new.end, break, this is where we should insert our new Interval(maybe not the initial one)
3. cur interleaves with new, set new.start = mincur.start, new.start, new.end = maxcur.end, new.end, continue;

Finally don't forget to add **new** and those Intervals after **new**(cur.start > new.end) to end of result list.

(BTW, there is another solution, first use binary search to decide where to insert the new Interval, then use the same idea in Problem "Merge Intervals", to re-merge the list.)

Listing 331: Solution

```java
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public List<Interval> insert(List<Interval> intervals,Interval newInterval){
        List<Interval> result = new ArrayList<Interval>();
        if (intervals.size() == 0){
            result.add(newInterval);
            return result;
```

```
        }
        Interval compare = new Interval(newInterval.start, newInterval.end);
        int endIdx = -1;
        for (int i = 0 ; i < intervals.size() ; i ++){
            Interval cur = intervals.get(i);
            if (cur.end < compare.start){
                result.add(cur);
            }
            else if (cur.start > compare.end){
                endIdx = i;
                break;
            }
            else{
                compare.start = Math.min(compare.start, cur.start);
                compare.end = Math.max(compare.end, cur.end);
            }
        }
        result.add(compare);
        if (endIdx != -1){
            for (int i = endIdx ; i < intervals.size() ; i ++){
                result.add(intervals.get(i));
            }
        }
        return result;
    }
}
```

Problem 253

87 Scramble String

```
Given a string s1, we may represent it as
 a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of s1 = "great":

    great
   /    \
  gr    eat
 / \    /  \
g   r  e   at
           / \
          a   t
To scramble the string, we may choose any non-leaf node and
 swap its two children.

For example, if we choose the node "gr" and swap its two children,
 it produces a scrambled string "rgeat".

    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t
We say that "rgeat" is a scrambled string of "great".

Similarly, if we continue to swap the children of nodes
 "eat" and "at", it produces a scrambled string "rgtae".

    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta  e
       / \
      t   a
We say that "rgtae" is a scrambled string of "great".

Given two strings s1 and s2 of the same length,
 determine if s2 is a scrambled string of s1.

https://leetcode.com/problems/scramble-string/
```

Solution - Recursion

First the recursive solution. Still use an example to help understanding.

For s1 = 'great' and s2 = 'rgtae'. We split each string to 2 parts, the split point can be every letter in word. When we split s1 to 'gr' and 'eat', while s2 to 'rg' and 'tae'(along with 'ae' and 'rgt'), the scramble can only be possible to take place between 'gr' and 'rg'(or maybe 'ae', recurse along this idea until the 2

words equal ,length not equals, or letters they contain are not the same(aab and bab/c, etc). If we don't use the last condition to prune, we will get TLE(using bit operation neither, see comments in code).

Listing 332: Solution 1

```
public class Solution {
    public boolean isScramble(String s1, String s2) {
        if (s1.length() != s2.length()){
            return false;
        }
        if (s1.equals(s2)){
            return true;
        }
        // former solution, much more slower than saved array
        // int num1 = 0, num2 = 0;
        // for (int i = 0 ; i < s1.length() ; i++){
        //     num1 |= (1 << (s1.charAt(i) - 'a'));
        //     num2 |= (1 << (s2.charAt(i) - 'a'));
        // }
        // if (num1 != num2){
        //     return false;
        // }
        int[] saved = new int[26];
        for(int idx = 0 ; idx < s1.length() ; idx++){
            saved[s1.charAt(idx)-'a'] += 1;
            saved[s2.charAt(idx)-'a'] -= 1;
        }
        for(int i = 0 ; i < 26 ; i++){
            if(saved[i] != 0) return false;
        }

        boolean result = false;
        for (int i = 1 ; i < s1.length() ; i ++){
            String left1 = s1.substring(0, i), right1 = s1.substring(i);
            String left2 = s2.substring(0, i), right2 = s2.substring(i);
            String left2Rev = s2.substring(0, s1.length() - i),
                   right2Rev = s2.substring(s1.length() - i);
            result |= (isScramble(left1, left2) && isScramble(right1, right2))
             || (isScramble(left1, right2Rev) && isScramble(right1, left2Rev));
            if (result == true){
                return result;
            }
        }
        return result;
    }
}
```

Solution - DP

Second solution makes use of Dynamic Programming, with the state transition equation:

The idea is search from the Internet. For dp[k][i][j], it means whether s1[i to i+k-1] equals to s2[j to j+k-1]

See `http://www.acmerblog.com/leetcode-solution-scramble-string-6224.html`

Listing 333: Solution 2

```
public class Solution {
    public boolean isScramble(String s1, String s2) {
        boolean dp[][][] = new boolean[s1.length() + 1][s1.length()][s1.length()];
        for (int i = 0 ; i < s1.length() ; i ++){
            for (int j = 0 ; j < s1.length() ; j ++){
                dp[1][i][j] = (s1.charAt(i) == s2.charAt(j));
            }
        }

        for (int k = 1 ; k <= s1.length() ; k ++){
            for (int i = 0 ; i <= s1.length() - k ; i ++){
                for (int j = 0 ; j <= s1.length() - k ; j++){
                    for (int inner = 1 ; inner < k ; inner ++){
                        dp[k][i][j] |= dp[inner][i][j]
                            && dp[k - inner][i + inner][j + inner];
                        dp[k][i][j] |= dp[inner][i][j + k - inner]
                            && dp[k - inner][i + inner][j];
                        if (dp[k][i][j]){
                            break;
                        }
                    }
                }
            }
        }
        return dp[s1.length()][0][0];
    }
}
```

Problem 254

315 Count of Smaller Numbers After Self

```
You are given an integer array nums and you have to return
 a new counts array.
  The counts array has the property where counts[i]
   is the number of smaller elements to the right of nums[i].

Example:

    Given nums = [5, 2, 6, 1]

    To the right of 5 there are 2 smaller elements (2 and 1).
    To the right of 2 there is only 1 smaller element (1).
    To the right of 6 there is 1 smaller element (1).
    To the right of 1 there is 0 smaller element.
    Return the array [2, 1, 1, 0].

https://leetcode.com/problems/count-of-smaller-numbers-after-self/
```

Solution - Merge Sort

Merge sort is a typical solution on finding inversions. On Book "Offer", there is a similar problem of finding all inversions in an array which uses merge sort too.

An Example, For [2,4],[1,3], we first compare 4 & 3, when 4 is bigger, smaller count of 4 is 2(1 and 3, index(3) - index(1) + 1), similarly, smaller count of 2 is 1...

Listing 334: Solution 1

```
class Number{
    int num;
    int index;

    public Number(int num, int index){
        this.num = num;
        this.index = index;
    }
}
public class Solution {
    public List<Integer> countSmaller(int[] nums) {
        int[] count = new int[nums.length];
        Number[] numbers= new Number[nums.length];
        for (int i = 0 ; i < nums.length ; i++){
            Number number = new Number(nums[i] , i);
            numbers[i] = number;
        }
        mergeSort(numbers, 0, numbers.length - 1, count);

        List<Integer> result = new ArrayList<Integer>();
        for (int c : count){
            result.add(c);
        }
```

```
        return result;
    }

    private void mergeSort(Number[] numbers, int start, int end , int[] count){
        if (start >= end){
            return;
        }
        int mid = start + (end - start) / 2;
        mergeSort(numbers, start, mid, count);
        mergeSort(numbers, mid + 1 , end, count);
        mergeCombine(numbers, start, mid, mid + 1, end, count);
    }
    private void mergeCombine(Number[] numbers, int start1,
     int end1, int start2, int end2, int[] count){
        Number[] result = new Number[end2 - start1 + 1];
        int idx = result.length - 1;
        int idxLeft = end1, idxRight = end2;
        while(idxLeft >= start1 && idxRight >= start2){
            if (numbers[idxRight].num >= numbers[idxLeft].num){
                result[idx --] = numbers[idxRight --];
            }
            else{
                count[numbers[idxLeft].index] += idxRight - start2 + 1;
                result[idx --] = numbers[idxLeft --];
            }
        }
        while (idxLeft >= start1){
            count[numbers[idxLeft].index] += idxRight - start2 + 1;
            result[idx --] = numbers[idxLeft --];
        }
        while (idxRight >= start2){
            result[idx --] = numbers[idxRight --];
        }

        for (int i = 0 ; i < result.length ; i ++){
            numbers[start1 + i] = result[i];
        }
    }

}
```

Solution - BIT

For BIT, we firstly sort and discrete the original array to make array in BIT much shorter. Then we assign each element a index according to its position after sorted, eg: for [5,2,6,1], there will be $1 -> 1, 2 -> 2, 3 -> 5, 4 -> 6$.

Then it's time for BIT to take effect. We iterate from tail of original array, now sum of discrete[nums[i]] is count of its smaller elements, and we may then update the BIT since we have a new element counting from right.

Since we are counting 'smaller' rather than 'not bigger', we have to minus 1 to discrete[num[i]] when doing the sum operation, which is a bit forgettable.

Listing 335: Solution 2

```
class BITree{
    int capacity;
    int[] elements;

    private int lowbit(int num){
        return num & (-num);
    }

    public BITree(int capacity){
        this.capacity = capacity;
        elements = new int[capacity + 1];
    }

    public void add(int index, int value){
        while (index <= capacity){
            elements[index] += value;
            index += lowbit(index);
            if (index == 0){
                break;
            }
        }
    }
    public int sum(int index){
        int sum = 0;
        while (index > 0){
            sum += elements[index];
            index -= lowbit(index);
        }
        return sum;
    }
}
public class Solution {
    public List<Integer> countSmaller(int[] nums){
        // Sort and Discretize
        int[] clone = new int[nums.length];
        for (int numIdx = 0 ; numIdx < nums.length ; numIdx ++){
            clone[numIdx] = nums[numIdx];
        }
        // Sorting
        Arrays.sort(clone);
        // Discretization
        Map<Integer, Integer> discreteMap = new HashMap<Integer, Integer>();
        int count = 0;
        for (int dIdx = 0 ; dIdx < clone.length ; dIdx++){
            if (dIdx > 0 && clone[dIdx] == clone[dIdx - 1]){
                continue;
            }
            // discrete index is from 1~n not 0~n
            // since tree.elements[0] is
            //  to store the count less than minimum value in array
            discreteMap.put(clone[dIdx] , ++ count );
        }
        Integer[] result = new Integer[nums.length];
```

```
        BITree tree = new BITree(discreteMap.keySet().size());
        for (int numIdx = nums.length - 1 ; numIdx >= 0 ; numIdx --){
          // here -1 means finding smaller elements
            result[numIdx] = tree.sum(discreteMap.get(nums[numIdx]) - 1);
            tree.add(discreteMap.get(nums[numIdx]) , 1);
        }
        return new ArrayList<Integer>(Arrays.asList(result));
    }
}
```

There is also a BST solution in
http://whatsme.net/2015/12/16/Count-of-Smaller-Numbers-After-Self/
Segment tree:
https://leetcode.com/discuss/73917/accepted-c-solution-using-segment-tree
Divide and Conquer:
https://leetcode.com/discuss/73326/divide-and-conquer-like-merge-sort-order-statistics-tre

Problem 255

327 Count of Range Sum

```
Given an integer array nums,
 return the number of range sums that lie in [lower, upper] inclusive.
 Range sum S(i, j) is defined as
  the sum of the elements in nums between indices i and j (i <= j), inclusive.

 Note:
 A naive algorithm of O(n2) is trivial.
  You MUST do better than that.

 Example:
     Given nums = [-2, 5, -1], lower = -2, upper = 2,
     Return 3.
     The three ranges are : [0, 0], [2, 2], [0, 2]
      and their respective sums are: -2, -1, 2.

 https://leetcode.com/problems/count-of-range-sum/
```

Solution - Merge Sort

The Merge sort train of thought is a bit like Problem "Count of Smaller Numbers After Self". We first sum up [0,x] elements to get a sums array with the length of n(which means, to be more specific, sums[0] = nums[0], sums[1] = nums[0] + nums[1], ..., sums[n] = nums[0] + nums[1] + nums[2] + ...).

Then apply Merge Sort among sums array. During merging, count the sums satisfy the [lower, upper] constraint. How?

Let's see what we are looking for. What is the value of sum from index i to j? sums[i to j] = sums[j] - sums[i - 1]. So what we do is to check every possible index pair (i,j) to check if sums[i to j] satisfy the constraint.

So why Merge Sort?(Honestly, because of its $O(logn)runtime$ complexity.) And how? Let example talk.

Suppose we are doing merge sort at a circumstance where the sums array looks like [(2,5,9),(6,8,11)] with upper=7 and lower=5, then we may combine the array to [2,5,6,8,9,11]... Wait! We have something else to do. When checking 2, we should also check 6,8 and maybe 11. See 6-2=4 < 5, continue when 8-2=6 > 5, stop and mark lowerIdx=idx(8), then 11-2 > 7, mark upperIdx=idx(11), then we add count 1 to totalCnt.

Namely, the merge sort based solution counts the answer while doing the merge. During the merge stage, we have already sorted the left half [start, mid) and right half [mid, end). We then iterate through the left half with index i. For each i, we need to find two indices k and j in the right half where

j is the first index satisfy sums[j] - sums[i] > upper and
k is the first index satisfy sums[k] - sums[i] >= lower.

Then the number of sums in [lower, upper] is j-k. We also use another index t to copy the elements satisfy sums[t] < sums[i] to a cache in order to complete the merge sort.

Finally, don't forget to use long type since the last test data will be out of bound.

Listing 336: Solution 1

```
public class Solution {
    public int countRangeSum(int[] nums, int lower, int upper) {
        if (nums.length == 0){
            return 0;
        }
        long[] sums = new long[nums.length];
```

Problem 255 continued on next page...

```
        sums[0] = nums[0];
        int count = 0;
        for (int i = 1 ; i < nums.length ; i ++){
            sums[i] = sums[i - 1] + nums[i];

        }
//        System.out.println("Initial state");
        for (int i = 0 ; i < nums.length ; i ++){
//            System.out.print(nums[i] + " ");
            if (sums[i] >= lower && sums[i] <= upper){
                count ++;
            }
        }
//        System.out.println();
//        System.out.println("After Transformation.");
        return count + mergeSort(sums, 0, sums.length - 1, lower, upper);
    }

    private int mergeSort(long[] sums, int startIdx,
     int endIdx, long lower, long upper){
        if (startIdx >= endIdx){
            return 0;
        }
        int midIdx = startIdx + (endIdx - startIdx) / 2;

        int count = mergeSort(sums, startIdx, midIdx, lower, upper)
                  + mergeSort(sums, midIdx + 1 , endIdx, lower, upper);
        int lowerIdx = midIdx + 1, upperIdx = midIdx + 1;
        int mergeIdx = midIdx + 1;
        int cacheIdx = 0;
        long cache[] = new long[endIdx - startIdx + 1];
        for (int leftIdx = startIdx ; leftIdx <= midIdx ; leftIdx ++){
            while (lowerIdx <= endIdx
             && sums[lowerIdx] - sums[leftIdx] < lower){
                lowerIdx ++;
            }
            upperIdx = lowerIdx;
            while (upperIdx <= endIdx
             && sums[upperIdx] - sums[leftIdx] <= upper){
                upperIdx ++;
            }
            while (mergeIdx <= endIdx
             && sums[mergeIdx] < sums[leftIdx]){
                cache[cacheIdx ++] = sums[mergeIdx ++];
            }
            count += upperIdx - lowerIdx;
            cache[cacheIdx ++] = sums[leftIdx];
        }
        while (mergeIdx <= endIdx){
            cache[cacheIdx ++] = sums[mergeIdx ++];
        }
        for (int recoverIdx = 0 ;
         recoverIdx < endIdx - startIdx + 1 ; recoverIdx ++){
```

```
            sums[recoverIdx + startIdx] = cache[recoverIdx];
        }
//        for (int sum : sums){
//            System.out.print(sum + " ");
//        }
//        System.out.println();
        return count;
    }
}
```

Solution - BST

I didn't understand the solution at first sight, but honestly this is more intuitive. We are looking for a range sum from 'lower' to 'upper', so when we consider the 'sums' array in previous solution, we are looking for a $lower <= sums[j] - sums[i-1] <= upper$. So we build a BST to store each sums, before inserting a new sums[i], we check in the tree that is there any previous sums X which satisfy $X >= sums[j] - upper$ and $X <= sums[j] - lower$. We used an exclusive method to get final result.

Listing 337: Solution 2

```
class BSTNode{
    BSTNode left;
    BSTNode right;
    int count;
    int leftCnt;
    int rightCnt;
    long val;
    public BSTNode(long val){
        this.count = 1;
        this.val = val;
    }
}
public class Solution {
    private int countLower(BSTNode node, long val){
        if (node == null){
            return 0;
        }
        if (node.val == val){
            return node.leftCnt;
        }
        else if (node.val > val){
            return countLower(node.left, val);
        }
        else{
            return node.leftCnt + node.count + countLower(node.right, val);
        }
    }

    private int countUpper(BSTNode node, long val){
        if (node == null){
            return 0;
        }
        if (node.val == val){
            return node.rightCnt;
```

```
        }
        else if (node.val < val){
            return countUpper(node.right, val);
        }
        else {
            return node.rightCnt + node.count + countUpper(node.left, val);
        }
    }
    private BSTNode insert(BSTNode node, long val){
        if (node == null){
            return new BSTNode(val);
        }
        if (node.val == val){
            node.count ++;
            return node;
        }
        else if (node.val > val) {
            node.leftCnt ++;
            node.left = insert(node.left, val);
        }
        else{
            node.rightCnt ++;
            node.right = insert(node.right, val);
        }
        return node;
    }

    private int countInRange(BSTNode root, long lower, long upper){
        int total = root.count + root.leftCnt + root.rightCnt;
        int lowerCnt = countLower(root, lower);
        int upperCnt = countUpper(root, upper);
        return total - lowerCnt - upperCnt;
    }

    public int countRangeSum(int[] nums, int lower, int upper) {
        long sums[] = new long[nums.length + 1];
        for (int i = 0 ; i < nums.length ; i ++){
            sums[i + 1] = sums[i] + nums[i];
        }

        BSTNode root = new BSTNode(sums[0]);
        int total = 0;
        for (int i = 1 ; i < sums.length ; i++){
            total += countInRange(root, sums[i] - upper, sums[i] - lower);
            insert(root, sums[i]);
        }

        return total > 0 ? total : 0;
    }
}
```

Solution - Others(BIT)

Honestly, I don't quite understand the insight of this solution, just paste here the code searched from

Discuss part for further usage.

Listing 338: Solution 3

```
public class Solution {
    public int countRangeSum(int[] nums, int lower, int upper) {
        List<Long> cand = new ArrayList<>();
        cand.add(Long.MIN_VALUE); // make sure no number gets a 0-index.
        cand.add(0L);
        long[] sum = new long[nums.length + 1];
        for (int i = 1; i < sum.length; i++) {
            sum[i] = sum[i - 1] + nums[i - 1];
            cand.add(sum[i]);
            cand.add(lower + sum[i - 1] - 1);
            cand.add(upper + sum[i - 1]);
        }
        Collections.sort(cand); // finish discretization

        int[] bit = new int[cand.size()];
        for (int i = 0; i < sum.length; i++) plus(bit,
         Collections.binarySearch(cand, sum[i]), 1);
        int ans = 0;
        for (int i = 1; i < sum.length; i++) {
            plus(bit, Collections.binarySearch(cand, sum[i - 1]), -1);
            ans += query(bit, Collections.binarySearch(cand,
                                                upper + sum[i - 1]));
            ans -= query(bit, Collections.binarySearch(cand,
                                                lower + sum[i - 1] - 1));
        }
        return ans;
    }

    private void plus(int[] bit, int i, int delta) {
        for (; i < bit.length; i += i & -i) bit[i] += delta;
    }

    private int query(int[] bit, int i) {
        int sum = 0;
        for (; i > 0; i -= i & -i) sum += bit[i];
        return sum;
    }
}
```

Problem 256

30 Substring with Concatenation of All Words

```
You are given a string, 's', and a list of words, 'words',
 that are all of the same length.
  Find all starting indices of substring(s) in s
   that is a concatenation of each word in words exactly once
    and without any intervening characters.

 For example, given:
     s: "barfoothefoobarman"
     words: ["foo", "bar"]

 You should return the indices: [0,9].
 (order does not matter).

 https://leetcode.com/problems/substring-with-concatenation-of-all-words/
```

Solution

Just a Brute force method, the condition that all words are of the same length makes the problem much easier. Keep a given length of sequence and check every word in it if they match our given **words** array.

Listing 339: Solution

```
public class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
        List<Integer> result = new ArrayList<Integer>();
        if (words.length == 0){
            return new ArrayList<Integer>();
        }
        Map<String, Integer> mapping = new HashMap<String, Integer>();
        Map<String, Integer> current = new HashMap<String, Integer>();
        // words in 'words' array may be duplicate
        for (String word : words){
            if (mapping.containsKey(word)){
                mapping.put(word, mapping.get(word) + 1);
            }
            else{
                mapping.put(word, 1);
            }
        }
        int wordCnt = words.length;
        int wordLen = words[0].length();
        // Since words are of the same length, we split each substrings.
        for (int i = 0 ; i <= s.length() - wordCnt * wordLen; i ++){
            current.clear();
            int j = 0;
            for (j = 0 ; j < wordCnt ; j++){
                String substring = s.substring(i + j * wordLen,
                                               i + (j + 1) * wordLen);
                if (!mapping.containsKey(substring)){
                    break;
                }
```

```
                if (current.containsKey(substring)){
                    current.put(substring, current.get(substring) + 1);
                }
                else{
                    current.put(substring, 1);
                }
                if (current.get(substring) > mapping.get(substring)){
                    break;
                }
            }
            if (j == wordCnt){
                result.add(i);
            }
        }

        return result;
    }
}
```

Problem 257

149 Max Points on a Line

```
Given n points on a 2D plane,
 find the maximum number of points that lie on the same straight line.

 https://leetcode.com/problems/max-points-on-a-line/
```

Solution

Main idea is to use the slope to form a map to check the number of points on same line. Iterate both i and j from 0 to n(which I think has lots of duplicate calculation).

Other difficulties are caused by corner cases: points with same x; same points; when all points are in the same position...

Listing 340: Solution

```
/**
 * Definition for a point.
 * class Point {
 *     int x;
 *     int y;
 *     Point() { x = 0; y = 0; }
 *     Point(int a, int b) { x = a; y = b; }
 * }
 */
public class Solution {
    public int maxPoints(Point[] points) {
        if (points.length <= 1){
            return points.length;
        }
        Map<Double, Integer> slopeCnt = new HashMap<Double, Integer>();
        int max = 0;
        for (int i = 0 ; i < points.length ; i ++){
            slopeCnt.clear();
            Point pointX = points[i];
            int duplicate = 0;
            for (int j = 0 ; j < points.length ; j++){
                if (i == j){
                    continue;
                }
                Point pointY = points[j];
                if (pointX.x == pointY.x && pointX.y == pointY.y){
                    duplicate ++;
                    continue;
                }
                else if (pointX.x == pointY.x){
                    slopeCnt.put(Double.MAX_VALUE,
                     slopeCnt.containsKey(Double.MAX_VALUE)
                      ? slopeCnt.get(Double.MAX_VALUE) + 1 : 1);
                }
                else{
                    double slope =
```

```
                        (pointY.y - pointX.y) * 1.0 / (pointY.x - pointX.x);
                    slopeCnt.put(slope, slopeCnt.containsKey(slope)
                        ? slopeCnt.get(slope) + 1 : 1);
                }
            }
            // System.out.println("Current pos: " + pointX.x + " , " + pointX.y);
            for (Double key : slopeCnt.keySet()){
                // System.out.println("\t" + key + " : " + slopeCnt.get(key));
                max = Math.max(max, slopeCnt.get(key) + duplicate + 1);
            }
            if (slopeCnt.keySet().size() == 0){
                max = Math.max(max, duplicate + 1);
            }
        }

        return max;
    }
}
```

Problem 258

115 Distinct Subsequences

```
Given a string S and a string T,
 count the number of distinct subsequences of T in S.

 A subsequence of a string is a new string
  which is formed from the original string by deleting some
   (can be none) of the characters without disturbing
    the relative positions of the remaining characters.
     (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

 Here is an example:
     S = "rabbbit", T = "rabbit"

     Return 3.

 https://leetcode.com/problems/distinct-subsequences/
```

Solution

Honestly, this is not a difficult DP problem. The state transition equation is simple:

Let dp[i][j] represents match count of first ith letters in S with first jth letters in T.

When s.charAt(i-1) == t.charAt(j-1), then dp[i][j] = dp[i-1][j-1] + dp[i-1][j]; Eg, for 2nd a in rrraa(t is ra), dp equals 3 + 3, where first 3 represents how many 'ra' the first 'rrra' can produce; where last 3 represents how many can 'rrrXa(current a)' get.

While when s.charAt(i-1) != t.charAt(j-1), dp[i][j] = dp[i - 1][j];

Listing 341: Solution

```
public class Solution {
    public int numDistinct(String s, String t) {
        int lenS = s.length();
        int lenT = t.length();
        int[][] dp = new int[s.length() + 1][t.length() + 1];
        for (int i = 0 ; i <= lenS ; i++){
            dp[i][0] = 1;
        }
        for (int j = 1 ; j <= lenT ; j++){
            for (int i = 1 ; i <= lenS ; i++){
                if (s.charAt(i - 1) == t.charAt(j - 1)){
                    dp[i][j] = dp[i - 1][j] + dp[i - 1][j - 1];
                }
                else{
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }
        return dp[lenS][lenT];
    }
}
```

Problem 259

97 Interleaving String

```
Given s1, s2, s3, find whether s3 is formed
 by the interleaving of s1 and s2.

 For example,
 Given:
 s1 = "aabcc",
 s2 = "dbbca",

 When s3 = "aadbbcbcac", return true.
 When s3 = "aadbbbaccc", return false.

 https://leetcode.com/problems/interleaving-string/
```

Solution

Another DP problem involving String issues.

How to find the state transition equation is a bit difficult but the equation is not difficult to understand. Though there are 3 strings, s1, s2 and s3, len(s1) + len(s2) = len(s3)(otherwise the function must return false), consequently there are only 2 independent variables, so we let boolean dp[i][j] represents if first i digits of s1 interleaved with first j digits of s2 comes up with first i+j digits of s3.

Thus, dp[i][j] = (s1.charAt(i-1) == s3.charAt(i+j-1) && dp[i-1][j]) || (s2.charAt(j-1) == s3.charAt(i+j-1) && dp[i][j-1]);

Listing 342: Solution

```
public class Solution {
    public boolean isInterleave(String s1, String s2, String s3) {
        boolean[][] dp = new boolean[s1.length() + 1][s2.length() + 1];
        if (s1.length() + s2.length() != s3.length()){
            return false;
        }
        dp[0][0] = true;
        for (int i = 0 ; i <= s1.length(); i ++){
            for (int j = 0; j <= s2.length() ; j ++){
                if (i > 0 && s1.charAt(i - 1)
                    == s3.charAt(i + j - 1) && dp[i - 1][j]){
                    dp[i][j] = true;
                }
                if (j > 0 && s2.charAt(j - 1)
                    == s3.charAt(i + j - 1) && dp[i][j - 1]){
                    dp[i][j] = true;
                }
            }
        }

        return dp[s1.length()][s2.length()];
    }
}
```

Problem 260

335 Self Crossing

```
You are given an array x of n positive numbers.
 You start at point (0,0) and moves x[0] metres to the north,
  then x[1] metres to the west, x[2] metres to the south,
   x[3] metres to the east and so on.
    In other words,
     after each move your direction changes counter-clockwise.

 Write a one-pass algorithm with O(1) extra space to determine,
  if your path crosses itself, or not.

 Example 1:
     Given x = [2, 1, 1, 2],
      ---
     |   |
      ---+-->
         |

     Return true (self crossing)

 Example 2:
     Given x = [1, 2, 3, 4],
      ------
     |      |
     |
     |
      ----------->$

     Return false (not self crossing)

 Example 3:
     Given x = [1, 1, 1, 1],
      ---
     |   |
      ---+->
         |

     Return true (self crossing)

 https://leetcode.com/problems/self-crossing/
```

Solution

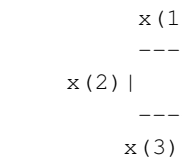
Well, it is more like a math problem or pattern-finding problem.

The key is to find out what are the serial numbers of lines that can form a cross with the 1st line(1st line is just a representation of all lines since the process of validation can be cyclic, which means we can regard 2nd line as the **'1st'** line).

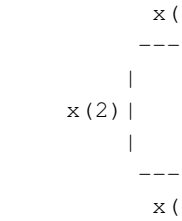
Just list the conclusion,

there are 3 situation where 2 lines form a cross with each other,

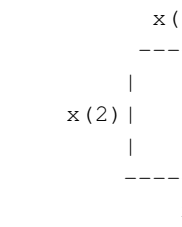
1.



2.



3.



Where each conditions can be derived by given circumstances. See code in listing.

Listing 343: Solution

```
class Move{
    int fromX, toX;
    int fromY, toY;
    public Move(int fromX, int fromY, int toX, int toY){
        this.fromX = fromX;
        this.fromY = fromY;
        this.toX = toX;
        this.toY = toY;
    }
}
public class Solution {
    public boolean isSelfCrossing(int[] x) {
        if (x.length <= 3){
            return false;
        }
        for (int moveIdx = 0 ; moveIdx < x.length ; moveIdx ++){
            // 1st line cross 4th line
            if (moveIdx >= 3){
                if (x[moveIdx - 3] >= x[moveIdx - 1]
                    && x[moveIdx] >= x[moveIdx - 2]){
                    return true;
                }
            }
            // 1st cross 5th
            if (moveIdx >= 4){
                if (x[moveIdx - 3] == x[moveIdx - 1]
```

```
                    && x[moveIdx - 2] <= x[moveIdx - 4] + x[moveIdx]){
                    return true;
                }
            }
            // 1st cross 6th
            if (moveIdx >= 5){
                if (x[moveIdx - 4] <= x[moveIdx - 2]
                    && x[moveIdx - 1] <= x[moveIdx - 3]
                    // No need to check
                    // &&x[moveIdx - 5] <= x[moveIdx - 3]
                    // && x[moveIdx] <= x[moveIdx - 2]
                    && x[moveIdx - 2] <= x[moveIdx - 4] + x[moveIdx]
                    && x[moveIdx - 3] <= x[moveIdx - 5] + x[moveIdx - 1]){
                        return true;
                    }
            }
        }
        return false;
    }
}
```

Problem 261

68 Text Justification

```
Given an array of words and a length L,
 format the text such that each line has
  exactly L characters and is fully (left and right) justified.

 You should pack your words in a greedy approach;
  that is, pack as many words as you can in each line.
   Pad extra spaces ' ' when necessary so that each line
    has exactly L characters.

 Extra spaces between words should be distributed
  as evenly as possible.
   If the number of spaces on a line do not divide evenly between words,
    the empty slots on the left will be assigned more spaces
     than the slots on the right.

 For the last line of text, it should be left justified and
  no extra space is inserted between words.

 For example,
     words: ["This", "is", "an", "example", "of", "text", "justification."]
     L: 16.

     Return the formatted lines as:
     [
        "This    is    an",
        "example  of text",
        "justification.  "
     ]
 Note: Each word is guaranteed not to exceed L in length.


 Corner Cases:
 A line other than the last line might contain only one word.
  What should you do in this case?
 In this case, that line should be left-justified.

 https://leetcode.com/problems/text-justification/
```

Solution

Just a simulation problem, but a bit complicated. Be aware of the last line and number of spaces in each line(I used Math.ceil to get spaceCnt each time, then remove spaceCnt from total space count).

Listing 344: Solution

```
public class Solution {
    public List<String> fullJustify(String[] words, int maxWidth) {
        if (words.length == 0){
            return new ArrayList<String>();
        }
        List<String> result = new ArrayList<String>();
```

Problem 261 continued on next page...

```
        boolean lastLine = false;
        int curIdx = 0;
        while (!lastLine && curIdx < words.length){
            int curWidth = 0;
            int cntInLine = 0;
            int startIdx = curIdx;
            String newLine = "";
            while (curIdx < words.length &&
                curWidth + words[curIdx].length() + curIdx - startIdx
                     <= maxWidth){
                curWidth += words[curIdx ++].length();
            }
            System.out.println(curIdx);
            if (curIdx == words.length){
                lastLine = true;
                for (int i = startIdx ; i < curIdx ; i++){
                    newLine += words[i];
                    if (i != curIdx - 1){
                        newLine += " ";
                    }
                }

                int spaceCnt = maxWidth - newLine.length();
                for (int spaceIdx = 0 ; spaceIdx < spaceCnt ; spaceIdx ++){
                    newLine += " ";
                }
            }
            else{
                int spaceCnt = maxWidth - curWidth;
                for (int i = startIdx; i < curIdx ; i ++){
                    newLine += words[i];
                    if (i != curIdx - 1){
                        int curSpaceCnt =
                         (int) Math.ceil(spaceCnt * 1.0 / (curIdx - i - 1));
                        for (int spaceIdx = 0 ;
                         spaceIdx < curSpaceCnt ; spaceIdx ++){
                            newLine += " ";
                        }
                        spaceCnt -= curSpaceCnt;
                    }
                }
                for (int spaceIdx = 0 ; spaceIdx < spaceCnt; spaceIdx ++){
                    newLine += " ";
                }
            }
            result.add(newLine);
        }

        return result;
    }
}
```

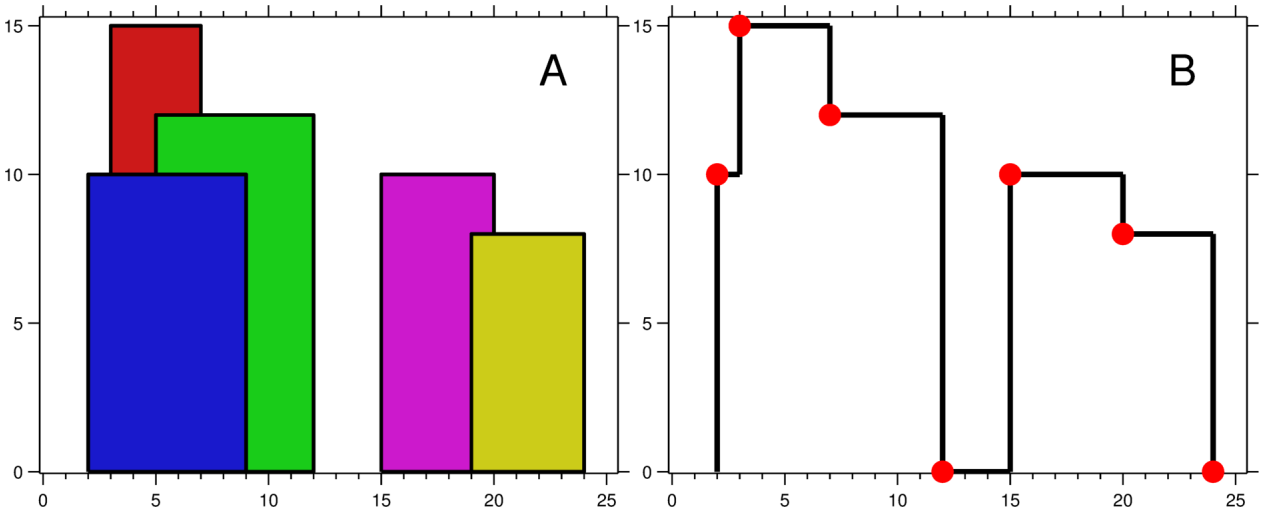
Problem 262

218 The Skyline Problem

```
A city's skyline is the outer contour of the silhouette
 formed by all the buildings in that city when viewed from a distance.
  Now suppose you are given the locations and height of
   all the buildings as shown on a cityscape photo (Figure A),
    write a program to output the skyline
     formed by these buildings collectively (Figure B).

  Buildings  Skyline Contour
 The geometric information of each building
  is represented by a triplet of integers [Li, Ri, Hi],
   where Li and Ri are the x coordinates of
    the left and right edge of the ith building, respectively,
     and Hi is its height.
      It is guaranteed that 0 <= Li, Ri <= INT_MAX,
       0 < Hi <= INT_MAX, and Ri - Li > 0.
        You may assume all buildings are perfect rectangles
         grounded on an absolutely flat surface at height 0.

 For instance, the dimensions of all buildings in Figure A
  are recorded as: [ [2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8] ] .

 The output is a list of "key points" (red dots in Figure B)
  in the format of [ [x1,y1], [x2, y2], [x3, y3], ... ]
   that uniquely defines a skyline.
    A key point is the left endpoint of a horizontal line segment.
     Note that the last key point, where the rightmost building ends,
      is merely used to mark the termination of the skyline,
       and always has zero height.
        Also, the ground in between any two adjacent buildings
         should be considered part of the skyline contour.

 For instance, the skyline in Figure B should be
  represented as:[ [2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0] ].

 Notes:

     The number of buildings in any input list is guaranteed
          to be in the range [0, 10000].
     The input list is already sorted in
          ascending order by the left x position Li.
     The output list must be sorted by the x position.
     There must be no consecutive horizontal lines of
         equal height in the output skyline.
          For instance, [...[2 3], [4 5], [7 5], [11 5], [12 7]...]
           is not acceptable;
            the three lines of height 5 should be
             merged into one in the final output as such:
              [...[2 3], [4 5], [12 7], ...]
```

Problem 262 continued on next page...

https://leetcode.com/problems/the-skyline-problem/



Solution

Not so difficult if we know the routine to solve this problem.

1. We use left top and right top corner of rectangle to represent itself, for simplicity.

2. Sort the Points(corners) according to x-axis value, from small to large ones, identifying the left corner and right corner.

3. Iterate each Point, when we meet a left corner, add it into a max heap by its y-axis value. When the value we add is larger than former peek from heap, we may find a new 'shadow', so we add the (x,y) pair into result; when there is a right corner, pop it from the heap. When new peek is smaller than the element we pop, this is a new identifying point, add it into result.

4. Don't forget to deal with some corner cases, imagine when input is [[1,2,1],[2,3,1]] OR [[1,2,1],[1,2,2],[1,2,3]] OR [[1,2,2],[1,2,3],[1,2,1]], what will happen?

Listing 345: Solution

```
class Point{
    int x, y;
    boolean left;
    int index;
    public Point(int x, int y , boolean left, int index){
        this.x = x;
        this.y = y;
        this.left = left;
        this.index = index;
    }
}
public class Solution {
    public List<int[]> getSkyline(int[][] buildings) {
        if (buildings.length == 0){
            return new ArrayList<int[]>();
        }
        List<int[]> result = new ArrayList<int[]>();
        List<Point> points = new ArrayList<Point>();
        for (int i = 0 ; i < buildings.length ; i++){
            int[] building = buildings[i];
            Point left = new Point(building[0], building[2], true, i);
```

```
            Point right = new Point(building[1], building[2], false, i);
            points.add(left);
            points.add(right);
        }
        Queue<Integer> heap = new PriorityQueue<Integer>(
            buildings.length , new Comparator<Integer>(){
            public int compare(Integer i1 , Integer i2){
                return i2 - i1;
            }
        });

        heap.offer(0);
        Collections.sort(points, new Comparator<Point>(){
            public int compare(Point p1 , Point p2){
                if (p1.x == p2.x){
                    if (p1.left && !p2.left){
                        return -1;
                    }
                    else if (!p1.left && p2.left){
                        return 1;
                    }
                    else{
                        return 0;
                    }
                }
                return p1.x - p2.x;
            }
        });
        for (Point point : points){
            if (point.left){
                if (point.y > heap.peek()){
                    if (result.size() > 0
                     && result.get(result.size() - 1)[0] == point.x){
                        result.remove(result.size() - 1);
                    }
                    result.add(new int[] {point.x, point.y});
                }
                heap.offer(point.y);
            }else{
                heap.remove(point.y);
                if (point.y > heap.peek()){
                    if (result.size() > 0
                     && result.get(result.size() - 1)[0] == point.x){
                        result.remove(result.size() - 1);
                    }
                    result.add(new int[] {point.x, heap.peek()});
                }
            }
        }
        return result;
    }
}
```

Problem 263

336 Palindrome Pairs

```
Given a list of unique words.
 Find all pairs of distinct indices (i, j) in the given list,
  so that the concatenation of the two words,
   i.e. words[i] + words[j] is a palindrome.

 Example 1:
     Given words = ["bat", "tab", "cat"]
     Return [[0, 1], [1, 0]]
     The palindromes are ["battab", "tabbat"]
 Example 2:
     Given words = ["abcd", "dcba", "lls", "s", "sssll"]
     Return [[0, 1], [1, 0], [3, 2], [2, 4]]
     The palindromes are ["dcbaabcd", "abcddcba", "slls", "llssssll"]

 https://leetcode.com/problems/palindrome-pairs/
```

Solution

First of all I must state that I don't think the problem is correct. When my input is ["aaa", "aaa", "aaa","aa","aa"] I should have gotten all combinations as my answer, but the expected answer is not what I want.

In order to pass the judgement, I have to do it in the way OJ advises.

When si+sj is palindrome, for example "as" and "llsa", there must be 'll' be a palindrome and 'as' be reversed by 'sa'. That's our key insight of solution. Iterate every word, split it into 2 parts, left and right, check if left OR right is palindrome and if it does, reverse the remaining part to see if it matches any other strings in original list.

Listing 346: Solution

```
class Solution{
    public List<List<Integer>> palindromePairs(String[] words) {
        List<List<Integer>> ret = new ArrayList<>();
        Map<String, Integer> wordIdxs = new HashMap<String, Integer>();
        for (int wordIdx = 0 ; wordIdx < words.length ; wordIdx ++){
            wordIdxs.put(words[wordIdx], wordIdx);
        }
        for (int curIdx = 0 ; curIdx < words.length ; curIdx ++){
            String curStr = words[curIdx];
            // String reverse = new StringBuilder(curStr).reverse().toString();
            for (int splitIdx = 0 ; splitIdx <= curStr.length() ; splitIdx ++){
                String left = curStr.substring(0, splitIdx);
                String right = curStr.substring(splitIdx);
                if (isPalindrome(left)){
                    String rightRev =
                         new StringBuilder(right).reverse().toString();
                    if (wordIdxs.containsKey(rightRev)
                     && wordIdxs.get(rightRev) != curIdx){
                        List<Integer> pair = new ArrayList<Integer>();
                        pair.add(wordIdxs.get(rightRev));
                        pair.add(curIdx);
```

```
                        ret.add(pair);
                    }
                }
                if (isPalindrome(right)){
                    String leftRev
                         = new StringBuilder(left).reverse().toString();
                    if (wordIdxs.containsKey(leftRev) && wordIdxs.get(leftRev)
                     != curIdx && right.length() != 0){
                        List<Integer> pair = new ArrayList<Integer>();
                        pair.add(curIdx);
                        pair.add(wordIdxs.get(leftRev));
                        ret.add(pair);
                    }
                }
            }
        }

        return ret;
    }

    private boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;
        while (left <= right) {
            if (str.charAt(left++) !=  str.charAt(right--)) return false;
        }
        return true;
    }
}
```

Problem 264

146 LRU Cache

```
Design and implement a data structure for Least Recently Used (LRU) cache.
 It should support the following operations: get and set.

 get(key) - Get the value (will always be positive)
      of the key if the key exists in the cache, otherwise return -1.
 set(key, value) - Set or insert the value
     if the key is not already present.
      When the cache reached its capacity,
       it should invalidate the least recently used item
        before inserting a new item.

 https://leetcode.com/problems/lru-cache/
```

Solution

A interesting and not so difficult problem. With plenty kinds of solutions. The most convenient method I've ever seen is to extend LinkedHashMap directly and overwrite some functions, which I think is a bit tricky.

Use a HashMap to trace each element in cache and out of cache. For each CacheEntry, assign its previous entry and next entry. A trick is to assign initially a dummy head and dummy tail node to quickly acquire the least recently used entry(by tail.prev) and most recently used(by head.next).

Listing 347: Solution

```
class CacheEntry{
    int value, key;
    CacheEntry pre, next;
    public CacheEntry(int key, int value){
        this.key = key;
        this.value = value;
    }
}
public class LRUCache {
    CacheEntry dummyHead;
    CacheEntry dummyTail;
    int capacity;
    int curSize;
    HashMap<Integer, CacheEntry> map = new HashMap<Integer, CacheEntry>();

    public LRUCache(int capacity) {
        this.capacity = capacity;
        curSize = 0;
        dummyHead = new CacheEntry(-1, -1);
        dummyTail = new CacheEntry(-1, -1);
        dummyHead.next = dummyTail;
        dummyTail.pre = dummyHead;
    }

    public int get(int key) {
        if (map.containsKey(key)){
```

```
            CacheEntry entry = map.get(key);
            moveToHead(entry);

            return entry.value;
        }
        return -1;
    }

    public void set(int key, int value) {
        if (map.containsKey(key)){
            CacheEntry entry = map.get(key);
            entry.value = value;
            moveToHead(entry);
            map.put(key, entry);
        }else{
            CacheEntry entry = new CacheEntry(key, value);
            if (curSize < capacity){
                curSize ++;
                addToHead(entry);
            }
            else{
                map.remove(dummyTail.pre.key);
                removeEntry(dummyTail.pre);
                addToHead(entry);
            }
            map.put(key, entry);
        }
    }

    private void moveToHead(CacheEntry node){
        removeEntry(node);
        addToHead(node);
    }
    private void removeEntry(CacheEntry node){
        // remove node
        node.next.pre = node.pre;
        node.pre.next = node.next;
    }
    private void addToHead(CacheEntry node){
        node.next = dummyHead.next;
        dummyHead.next.pre = node;
        node.pre = dummyHead;
        dummyHead.next = node;
    }
}
```

Problem 265

214 Shortest Palindrome

```
Given a string S, you are allowed to convert it to a palindrome
 by adding characters in front of it.
  Find and return the shortest palindrome you can find by
   performing this transformation.

 For example:

 Given "aacecaaa", return "aaacecaaa".

 Given "abcd", return "dcbabcd".

 https://leetcode.com/problems/shortest-palindrome/
```

Solution

A Brute force method is to substring from (0, s.length()) to (0,1), if the substring result is palindrome, then stop checking immediately(to get the shortest result) and reverse the remaining string when palindrome is cut from original string. Then attach the reversed string to the head of original string, which is exactly our result. Worst complexity $O(n^2)$, which will get TLE.

Another train of thought is to use the KMP algorithm.

For a quicker review of KMP, visit `http://www.cnblogs.com/c-cloud/p/3224788.html`.

For Chinese description of this problem, visit `https://segmentfault.com/a/1190000003059361`

Actually, the next[] array is enough for this problem. First we append reversed source string to tail of original string to get quite a long string for a result. Apply KMP on new string, get the largest element in next array, which is exactly the reversed part of original string.

BUT! We are extending the next array to be twice length as what former string has, which may lead to a error state that our longest prefix & suffix go through both reversed string and original string, when we only want to find longest prefix/suffix in original string. So we add a special character between s and reverse_s(here we use '#').

In conclusion, our solution can be divided into 3 steps.

1. Reverse string s as rev_s
2. Append rev_s to tail of s, namely form s#rev_s
3. Apply KMP to new string, get the maximum common length of prefix and suffix, such segment of string is actually palindrome(think of why.)
4. Split rev_s to get s.length() - common_length substring, attach it to head of string s

Listing 348: Solution

```
public class Solution {
    public String shortestPalindrome(String s) {
        if (s.length() <= 1){
            return s;
        }
        String revS = new StringBuilder(s).reverse().toString();
```

Problem 265 continued on next page...

```
        String longStr = s + "#" + revS;
        int patternIdx = 0;
        int[] next = new int[longStr.length()];
        next[0] = 0;
        for (int strIdx = 1 ; strIdx < longStr.length() ; strIdx ++){
            while (patternIdx > 0 && longStr.charAt(strIdx)
                 != longStr.charAt(patternIdx)){
                patternIdx = next[patternIdx - 1];
            }
            if (longStr.charAt(strIdx) == longStr.charAt(patternIdx)){
                patternIdx ++;
            }
            next[strIdx] = patternIdx;
            // System.out.println(strIdx + " : " + patternIdx);
        }
        return revS.substring(0, revS.length() - next[longStr.length() - 1]) + s;
    }
}
```

Problem 266

321 Create Maximum Number

```
Given two arrays of length m and n with digits 0-9 representing two numbers.
 Create the maximum number of length k $<=$ m + n
  from digits of the two. The relative order of the digits
   from the same array must be preserved.
    Return an array of the k digits.
     You should try to optimize your time and space complexity.

 Example 1:
     nums1 = [3, 4, 6, 5]
     nums2 = [9, 1, 2, 5, 8, 3]
     k = 5
     return [9, 8, 6, 5, 3]

 Example 2:
     nums1 = [6, 7]
     nums2 = [6, 0, 4]
     k = 5
     return [6, 7, 6, 0, 4]

 Example 3:
     nums1 = [3, 9]
     nums2 = [8, 9]
     k = 3
     return [9, 8, 9]

 https://leetcode.com/problems/create-maximum-number/
```

Solution

A Greedy problem, paste the solution in Chinese down here since I may not explain it correctly and accurately.

Problem 266 continued on next page...

贪心，开一个栈，每次都尽量往里塞最大的数，最后这个栈就是结果。

假设需要取n位数字，每一轮入栈前要先维护栈的状态，如果同时满足以下条件就把栈顶的元素弹出：

1. 栈不为空；

2. 栈顶元素小于当前的元素；

3. 栈顶弹出后，后面剩下没处理的子串加上栈的length要够n位，

看这个例子[3,2,1,4,5]，n = 3。

前三轮没有疑问，栈里放入3, 2, 1，stack = [3, 2, 1]。

第四轮循环的时候4比3, 2, 1都大，但不能把它们都弹出，因为不够3位了，只弹出2个，stack = [3, 4]。

最后5放入栈，stack = [3, 4, 5]。

第三点对应代码中 "stack.length + len - i > n" 这个条件。

分别取得了两个最大的子数，合并的操作就类似于merge sort。

双指针，一开始都指向0，放入一个大的，指针++，最后把剩下的都放入结果。

要注意**可能有相同的数字**，此时要看后面的数来决定哪个比较大，举例来说，[6, 1] 和 [6, 7]应该拿[6, 7]中的6。

Listing 349: Solution

```
public class Solution {
    public int[] maxNumber(int[] nums1, int[] nums2, int k) {
        int[] result = new int[k];
        if (nums1.length + nums2.length < k){
            return result;
        }
        for (int length1 = 0 ;
                length1 <= Math.min(nums1.length, k) ; length1++){
            int length2 = k - length1;
            if (length2 > nums2.length){
                continue;
            }

            int[] list1 = getKLenStack(nums1, length1);
            int[] list2 = getKLenStack(nums2, length2);
            int[] newResult = new int[k];
            mergeStacks(list1, list2, newResult);

            for (int resIdx = 0 ; resIdx < newResult.length ; resIdx ++){
                if (newResult[resIdx] == result[resIdx]){
                    continue;
                }
                else if (newResult[resIdx] > result[resIdx]){
                    result = newResult;
                    break;
                }
                else{
                    break;
                }
            }
```

```
        }
        return result;
    }

    private void mergeStacks(int[] list1, int[] list2, int[] result){

        int idx = 0;
        int idx1 = 0, idx2 = 0;
        while (idx1 < list1.length && idx2 < list2.length){
            if (list1[idx1] > list2[idx2]){
                result[idx ++] = list1[idx1 ++];
            }
            else if (list1[idx1] < list2[idx2]){
                result[idx ++] = list2[idx2 ++];
            }
            else{
                int next1 = list1[idx1];
                int next2 = list2[idx2];
                int count = 1;
                while (next1 == next2){
                    if (idx1 + count < list1.length){
                        next1 = list1[idx1 + count];
                    }
                    if (idx2 + count < list2.length){
                        next2 = list2[idx2 + count];
                    }
                    count ++;
                    if (idx1 + count > list1.length
                            && idx2 + count > list2.length){
                        break;
                    }
                }
                if (next1 >= next2){
                    result[idx ++] = list1[idx1 ++];
                }
                else{
                    result[idx ++] = list2[idx2 ++];
                }
            }
        }
        while (idx1 < list1.length){
            result[idx ++] = list1[idx1 ++];
        }
        while (idx2 < list2.length){
            result[idx ++] = list2[idx2 ++];
        }
    }

    private int[] getKLenStack(int[] nums, int length){
        int[] result = new int[length];
        Stack<Integer> stack = new Stack<Integer>();
        int curLen = 0;
        for (int numIdx = 0 ; numIdx < nums.length ; numIdx ++){
```

```
                int num = nums[numIdx];
85              if (nums.length + stack.size() - numIdx <= length){
                    stack.push(num);
                    result[curLen] = num;
                    curLen ++;
                    continue;
90              }
                while (!stack.isEmpty()
                        && num > stack.peek()
                        && stack.size() + nums.length - numIdx > length){
                    stack.pop();
95                  curLen --;
                }
                if (curLen < length){
                    stack.push(num);
                    result[curLen] = num;
100                 curLen ++;
                }
            }
            return result;
        }
105 }
```

Problem 267

343 Integer Break

```
Given a positive integer n,
 break it into the sum of at least two positive integers
  and maximize the product of those integers.
   Return the maximum product you can get.

 For example, given n = 2,
  return 1 (2 = 1 + 1); given n = 10, return 36 (10 = 3 + 3 + 4).

 Note: you may assume that n is not less than 2.

 Hint:

 There is a simple O(n) solution to this problem.
 You may check the breaking results of n ranging from
  7 to 10 to discover the regularities.

 https://leetcode.com/problems/integer-break/
```

Solution

Take as many 3 as we can, the proof process can be found below.

However, I'm not using such train of thought, but iterate each possible count of sum to get maximum product.

这道题的难点其实在于证明为什么拆出足够多的 3 就能使得乘积最大。下面我就试着证明一下。

首先证明拆出的因子大于 4 是不行的。设 x 是一个因子，$x > 4$，那么可以将这个因子再拆成两个因子 $x-2$ 和 2，易证 $(x-2) \times 2 > x$。所以不能有大于 4 的因子。

4 这个因子也是可有可无的，$4 = 2+2$，$4 = 2 \times 2$。因此 4 这个因子可以用两个 2 代替。

除非没有别的因子可用，1 也不能选作因子。一个数 x，当它大于 3 时，有 $(x-2) \times 2 > (x-1) \times 1$。

这样呢，就只剩下 2 和 3 这两个因子可以选了。下面再证明 3 比 2 好。

一个数 $x = 3m + 2n$，那么 $f = 3^m \times 2^n = 3^m \times 2^{\frac{x-3m}{2}}$ 可以对它取个对数。

$$\begin{aligned} lnf &= m\ln 3 + n\ln 2 \\ &= m\ln 3 + \frac{x-3m}{2}\ln 2 \\ &= \frac{x}{2}\ln 2 + (\ln 3 - \frac{3}{2}\ln 2)m \end{aligned}$$

其中 $\ln 3 - \frac{3}{2}\ln 2 > 0$ 所以 f 是 m 的增函数，也就是说 m 越大越好。所以 3 越多越好。

再多说一句，如果拆出的因子不限于整数的话，可以证明 $e = 2.718\ldots$ 是最佳的选择。感兴趣的可以试着证明一下。

Listing 350: Solution

```java
public class Solution {
    public int integerBreak(int n) {
        int max = -1;
        for (int digit = 2 ; digit <= n ; digit++){
            int product = 1;
```

```
                int base = n / digit;
                int rem = n % digit;
                for (int prodIdx = 0 ; prodIdx < digit ; prodIdx ++){
                    product *= (rem <= 0) ? base : (base + 1);
                    rem --;
                }
                max = Math.max(product, max);
            }
            return max;
        }
}
```

Problem 268

344 Reverse String

```
Write a function that takes a string as input and returns the string reversed.

Example:
Given s = "hello", return "olleh".

https://leetcode.com/problems/reverse-string/
```

Solution

An easy problem with plenty of ways to solve it.

Listing 351: Solution

```
public class Solution {
    public String reverseString(String s) {
        char[] charArr = s.toCharArray();
        int i = 0 , j = s.length() - 1;
        while(i < j){
            char tmp = charArr[i];
            charArr[i] = charArr[j];
            charArr[j] = tmp;
            i++;
            j--;
        }

        return new String(charArr);
    }
}
```

Problem 269

345 Reverse Vowels of a String

```
Write a function that takes a string as input and
 reverse only the vowels of a string.

 Example 1:
 Given s = "hello", return "holle".

 Example 2:
 Given s = "leetcode", return "leotcede".

 https://leetcode.com/problems/reverse-vowels-of-a-string/
```

Solution

An easy problem either, a bit bit more difficult than previous problem 'Reverse String'.

Listing 352: Solution

```
public class Solution {
    public String reverseVowels(String s) {
        Set<Character> set = new HashSet<Character>();
        set.add('A');set.add('a');
        set.add('E');set.add('e');
        set.add('I');set.add('i');
        set.add('O');set.add('o');
        set.add('U');set.add('u');
        char[] charArr = s.toCharArray();
        int i = 0 , j = s.length() - 1;
        while(i < j){
            while (i < j && !(set.contains(s.charAt(i)))){
                i++;
            }
            while (i < j && !(set.contains(s.charAt(j)))){
                j--;
            }
            if (i >= j){
                break;
            }
            char tmp = charArr[i];
            charArr[i] = charArr[j];
            charArr[j] = tmp;
            i++;
            j--;
        }

        return new String(charArr);
    }
}
```

Problem 270

349 Intersection of Two Arrays

```
Given two arrays, write a function to compute their intersection.

Example:
Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].

Note:
Each element in the result must be unique.
The result can be in any order.

https://leetcode.com/problems/intersection-of-two-arrays/
```

Solution

Using HashSet to find intersection and avoid duplicate.

Listing 353: Solution

```
public class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        Set<Integer> set1 = new HashSet<Integer>();
        Set<Integer> result = new HashSet<Integer>();
        for (int num1 : nums1){
            set1.add(num1);
        }
        for (int num2 : nums2){
            if (set1.contains(num2)){
                result.add(num2);
            }
        }
        int[] resultArr = new int[result.size()];
        int idx = 0;
        for (int res : result){
            resultArr[idx] = res;
            idx ++;
        }
        return resultArr;

    }
}
```

Problem 271

350 Intersection of Two Arrays II

```
Given two arrays, write a function to compute their intersection.

 Example:
 Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].

 Note:
 Each element in the result should appear as many times as it shows in both arrays.
 The result can be in any order.
 Follow up:
 What if the given array is already sorted? How would you optimize your algorithm?
 What if nums1's size is small compared to nums2's size? Which algorithm is better?
 What if elements of nums2 are stored on disk, and the memory is limited
  such that you cannot load all elements into the memory at once?

 https://leetcode.com/problems/intersection-of-two-arrays-ii/
```

Solution 1

The most direct solution is to use a HashMap for 1 array recording each number for occurrence and match another, maintaining the count of number in 2nd array.

Listing 354: Solution 1

```
public class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        if (nums1.length == 0 || nums2.length == 0){
            return new int[0];
        }
        ArrayList<Integer> result = new ArrayList<Integer>();
        HashMap<Integer, Integer> map1 = new HashMap<Integer, Integer>();
        for (int num1 : nums1){
            if (map1.containsKey(num1)){
                map1.put(num1, map1.get(num1) + 1);
            }
            else{
                map1.put(num1, 1);
            }
        }
        for (int num2 : nums2){
            if (map1.containsKey(num2) && map1.get(num2) > 0){
                map1.put(num2, map1.get(num2) - 1);
                result.add(num2);
            }
        }
        int[] res = new int[result.size()];
        for (int idx = 0 ; idx < result.size() ; idx ++){
            res[idx] = result.get(idx);
        }
        return res;
    }
}
```

Problem 271 continued on next page...

Solution 2

Another train of thought is by sorting and merging, can be achieved using either Arrays.sort or manually quickSort.

Listing 355: Solution 2

```
public class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        if (nums1.length == 0 || nums2.length == 0){
            return new int[0];
        }
        Arrays.sort(nums1);
        Arrays.sort(nums2);
        List<Integer> result = new ArrayList<Integer>();
        int idx1 = 0; int idx2 = 0;
        while (idx1 < nums1.length && idx2 < nums2.length){
            if (nums1[idx1] == nums2[idx2]){
                result.add(nums1[idx1]);
                idx1 ++;
                idx2 ++;
            }
            else if (nums1[idx1] > nums2[idx2]){
                idx2 ++;
            }
            else{
                idx1 ++;
            }
        }
        int[] res = new int[result.size()];
        for (int idx = 0 ; idx < result.size(); idx ++){
            res[idx] = result.get(idx);
        }
        return res;
    }
}
```

However, I still feel that we need further investigation on that, on one hand we are not quite sure about the questions in Follow up, while on the other there is no difference between I and II on algorithm, which may indicates that there must be somewhere we can optimize.

Problem 272

371 Sum of Two Integers

```
Calculate the sum of two integers a and b, but you are
 not allowed to use the operator + and -.

 Example:
 Given a = 1 and b = 2, return 3.

 https://leetcode.com/problems/sum-of-two-integers/
```

Solution

It's more a mathematical(bit manipulation) problem. For a + b, if there is no carry, the result is exactly a xor b by bit; similarly, if no carry, the new carry is (a & b) << 1, adding them still get sum, which begins a new iteration. Loop until new carry == 0, where a xor b gets final sum.

Listing 356: Solution

```java
public class Solution {
    public int getSum(int a, int b) {
        int sum = 0, carry = -1;
        while (carry != 0){
            sum = (a ^ b);
            carry = (a & b) << 1;
            a = sum;
            b = carry;
        }
        return sum;
    }
}
```

Problem 273

383 Ransom Note

```
Given an arbitrary ransom note string and another string containing letters
 from all the magazines, write a function that will return true if the ransom note
  can be constructed from the magazines ; otherwise, it will return false.

 Each letter in the magazine string can only be used once in your ransom note.

 Note:
 You may assume that both strings contain only lowercase letters.

 canConstruct("a", "b") - false
 canConstruct("aa", "ab") - false
 canConstruct("aa", "aab") - true

 https://leetcode.com/problems/ransom-note/
```

Solution

Array is enough.

Listing 357: Solution

```
public class Solution {
    public boolean canConstruct(String ransomNote, String magazine) {
        int[] letters = new int[257];
        char[] magChars = magazine.toCharArray();
        for (char magCh : magChars){
            letters[magCh] ++;
        }
        char[] ransomChars = ransomNote.toCharArray();
        for (char ransomCh : ransomChars){
            if (letters[ransomCh] == 0){
                return false;
            }
            else{
                letters[ransomCh] --;
            }
        }

        return true;
    }
}
```

Problem 274

373 Find K Pairs with Smallest Sums

```
You are given two integer arrays nums1 and nums2 sorted in ascending order
 and an integer k.

Define a pair (u,v) which consists of one element from the first array
 and one element from the second array.

Find the k pairs (u1,v1),(u2,v2) ...(uk,vk) with the smallest sums.

Example 1:
Given nums1 = [1,7,11], nums2 = [2,4,6],  k = 3

Return: [1,2],[1,4],[1,6]

The first 3 pairs are returned from the sequence:
[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
Example 2:
Given nums1 = [1,1,2], nums2 = [1,2,3],  k = 2

Return: [1,1],[1,1]

The first 2 pairs are returned from the sequence:
[1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
Example 3:
Given nums1 = [1,2], nums2 = [3],  k = 3

Return: [1,3],[2,3]

All possible pairs are returned from the sequence:
[1,3],[2,3]

https://leetcode.com/problems/find-k-pairs-with-smallest-sums/
```

Solution

Heap is the key to solution. However, this is not adequate, the train of thought is still a bit complicated. See explanation and figure from Discussion section.

Basic idea: Use min_heap to keep track on next minimum pair sum, and we only need to maintain K possible candidates in the data structure.

Some observations: For every numbers in nums1, its best partner(yields min sum) always strats from nums2[0] since arrays are all sorted; And for a specific number in nums1, its next candidate sould be [this specific number] + nums2[current_associated_index + 1], unless out of boundary;)

Here is a simple example demonstrate how this algorithm works.

Problem 274 continued on next page...

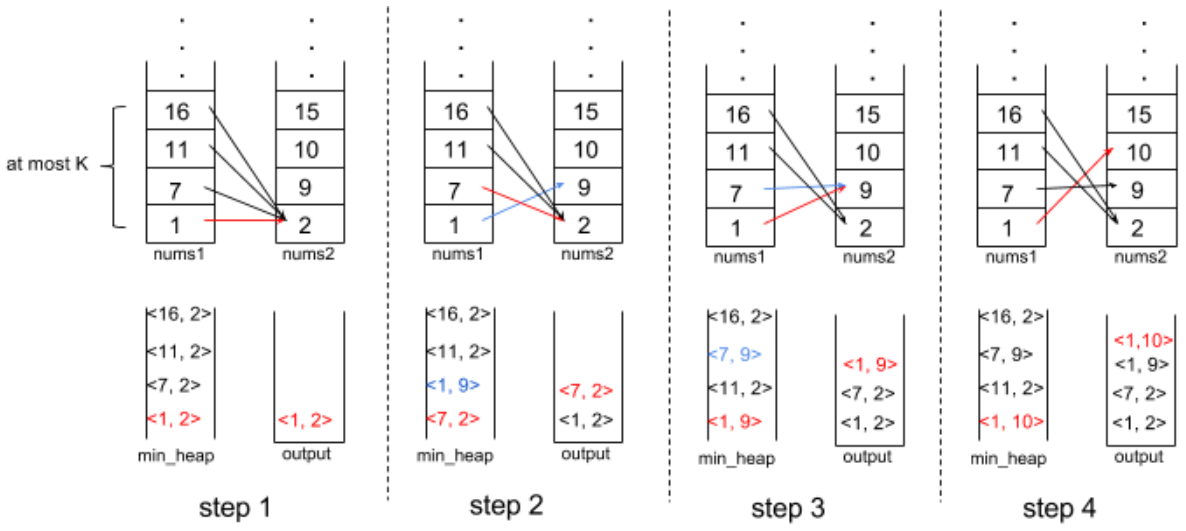


Figure_1. A simple example when K = 4 and nums1={1,7,11,16,...}, nums2={2,9,10,15,...}

The run time complexity is O(kLogk) since que.size <= k and we do at most k loop.

Listing 358: Solution

```
public class Solution {
    public List<int[]> kSmallestPairs(int[] nums1, int[] nums2, int k) {
        List<int[]> result  = new ArrayList<int[]>();
        PriorityQueue<int[]> queue = new PriorityQueue<>(
            (int[] a, int[] b) -> (a[0] + a[1] - b[0] - b[1]));
        if (nums1.length == 0 || nums2.length == 0 || k == 0){
            return result;
        }
        for (int idx = 0 ; idx < nums1.length ; idx ++){
            queue.offer(new int[]{nums1[idx], nums2[0], 0});
        }
        while (result.size() < k && !queue.isEmpty()){
            int[] cur = queue.poll();
            result.add(new int[]{cur[0], cur[1]});
            if (cur[2] == nums2.length - 1){
                continue;
            }
            queue.offer(new int[]{cur[0], nums2[cur[2] + 1], cur[2] + 1});
        }
        return result;
    }
}
```

Problem 275

378 Kth Smallest Element in a Sorted Matrix

```
Given a n * n matrix where each of the rows and columns
 are sorted in ascending order,
 find the kth smallest element in the matrix.
Note that it is the kth smallest element in the sorted order,
 not the kth distinct element.
Example:
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,

return 13.

Note:
You may assume k is always valid, 1 <= k <= n2.

https://leetcode.com/problems/find-k-pairs-with-smallest-sums/
```

Solution

It's quite similar to Problem 373 'Find K Pairs with Smallest Sums', even easier(How I wish I can add a 'wiggle' emoji here).

Put elements from 1st row(column), along with their value, rowIdx and colIdx. Poll elements from PriorityQueue which is definitely the smallest by now, then add in-bound elements from the next column(row) into PriorityQueue.

Listing 359: Solution

```
class Element{
    public int value;
    public int row;
    public int column;
    public Element(int value, int row, int column){
        this.value = value;
        this.row = row;
        this.column = column;
    }
}
public class Solution {
    public int kthSmallest(int[][] matrix, int k) {
        if (matrix.length == 0 || matrix[0].length == 0){
            return -1;
        }
        PriorityQueue<Element> elements = new PriorityQueue<Element>(k + 1,
         new Comparator<Element>(){
            public int compare(Element e1, Element e2){
                return e1.value - e2.value;
            }
```

```
        });
        for (int idxRow = 0 ; idxRow < matrix.length ; idxRow ++){
            elements.offer(new Element(matrix[idxRow][0], idxRow, 0));
        }
        Element little = null;
        int count = 0;
        while (count < k){
            little = elements.poll();
            count++;
            if (little.column < matrix[0].length - 1){
                elements.offer(
                    new Element(
                        matrix[little.row][little.column + 1],
                        little.row,
                        little.column + 1));
            }
        }
        return little.value;

    }
}
```

Problem 276

374 Guess Number Higher or Lower

```
We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API guess(int num) which returns 3 possible results
 (-1, 1, or 0):

-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
Example:
n = 10, I pick 6.

Return 6.

https://leetcode.com/problems/guess-number-higher-or-lower/
```

Solution

A standard binary search problem. Easy to implement and solved.

Listing 360: Solution

```
/* The guess API is defined in the parent class GuessGame.
   @param num, your guess
   @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
      int guess(int num); */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        int low = 1, high = n;
        int mid = low + (high - low) / 2;
        int flag = guess(mid);
        while (flag != 0 && low < high){
            if (flag == 1){
                low = mid + 1;
            }
            else if (flag == -1){
                high = mid - 1;
            }
            mid = low + (high - low) / 2;

            flag = guess(mid);
        }
        return mid;
    }
}
```

Problem 277

357 Count Numbers with Unique Digits

```
Given a non-negative integer n, count all numbers with unique digits, x,
 where 0 $<=$ x < 10n.

 Example:
 Given n = 2, return 91. (The answer should be the total numbers
  in the range of 0 $<=$ x < 100,
  excluding [11,22,33,44,55,66,77,88,99])

 Hint:

 A direct way is to use the backtracking approach.
 Backtracking should contains three states which are
  (the current number, number of steps to get that number and a bitmask
   which represent which number is marked as visited so far in the current number).
    Start with state (0,0,0) and count all valid number
     till we reach number of steps equals to 10n.
 This problem can also be solved using a dynamic programming approach and
  some knowledge of combinatorics.
 Let f(k) = count of numbers with unique digits with length equals k.
 f(1) = 10, ..., f(k) = 9 * 9 * 8 * ... (9 - k + 2)
  [The first factor is 9 because a number cannot start with 0].

 https://leetcode.com/problems/count-numbers-with-unique-digits/
```

Solution 1

Backtracking solution, recurse on each digit from 0 to 9, use a int act as bitmap to see which number has been used. Be careful when checking if 0 used. I'm using the method that when used == 0, other numbers have not been used, which also means that 0 chosen is the 1st digit, only in this case shouldn't we mark number 0 as used.

Listing 361: Solution 1 - Backtracking

```
public class Solution {
    int count = 0;
    public int countNumbersWithUniqueDigits(int n) {
        int used = 0;
        if (n == 0){
            return 1;
        }
        if (n > 10){
            return countNumbersWithUniqueDigits(10);
        }

        countNumber(1, n, used);
        return count;
    }
    private void countNumber(int digitIdx, int n, int used){
        if (digitIdx > n){
            count ++;
```

```
                return;
            }
            for (int digit = 0 ; digit <= 9 ; digit ++){
                int mask = (1 << digit);
                if ((mask & used) != 0){
                    continue;
                }
                if (digit != 0 || used != 0){
                    used |= mask;
                }

                countNumber(digitIdx + 1, n, used);
                used &= (~mask);
            }
        }
    }
```

Solution 2

Use math method as hint says. For n = 1, f(1) = 10 [0,1,2,...,9] For n = 2, f(2) = 9 * 9 [1st-digit: 1-9 & 2nd-digit: 0-9 except for 1st-digit] ... Finally, sum them up.

Listing 362: Solution 2 - Math

```
public class Solution {
    public int countNumbersWithUniqueDigits(int n) {
        if (n == 0){
            return 1;
        }
        int sum = 10, product = 9;
        for (int digit = 2 ; digit <= Math.min(n, 10) ; digit ++){
            product *= (11 - digit);
            sum += product;
        }

        return sum;
    }
}
```

Problem 278

367 Valid Perfect Square

```
Given a positive integer num, write a function which returns True
 if num is a perfect square else False.

 Note: Do not use any built-in library function such as sqrt.

 Example 1:

 Input: 16
 Returns: True
 Example 2:

 Input: 14
 Returns: False

 https://leetcode.com/problems/valid-perfect-square/
```

Solution 1

As for a binary search problem, I can't tell this should be classified as Medium difficulty. Using division instead of multiplication is a trick to prevent overflow.

Listing 363: Solution 1 - Binary Search

```
public class Solution {
    public boolean isPerfectSquare(int num) {
        if (num == 1){
            return true;
        }
        int low = 1, high = num;
        int mid = low + (high - low) / 2;
        while (low <= high){
            int div = num / mid;
            if (mid == div){
                if (num % div == 0){
                    return true;
                }
                else{
                    return false;
                }
            }
            else if(mid < div){
                low = mid + 1;
            }
            else{
                high = mid - 1;
            }
            mid = low + (high - low) / 2;
        }
        return false;
    }
}
```

Problem 278 continued on next page...

Solution 2

A interesting thing is that:

$1 = 1 \quad 4 = 1 + 3 \quad 9 = 1 + 3 + 5 \quad ....$

Listing 364: Solution 2 - Math.sum

```
public class Solution {
    public boolean isPerfectSquare(int num) {
        int i = 1;
        while (num > 0) {
            num -= i;
            i += 2;
        }
        return num == 0;
    }
}
```

Solution 3

The 3rd solution is a math method. I don't quite get the point so just leave the solution here.

Listing 365: Solution 3 - Math.Newton

```
class Solution {
public:
    bool isPerfectSquare(int n) {
        float diff;
        float x, x_prev = n;
        do{
            x = x_prev - (x_prev*x_prev - n)/(2*x_prev);
            diff = x_prev - x;
            x_prev = x;
        }
        while(diff>=1);
        // if n is a perfect square, its square root must be the floor of x
        x = (int) x;
        if (x*x == n)
            return true;
        else
            return false;

    }
};
```

Problem 279

376 Wiggle Subsequence

```
A sequence of numbers is called a wiggle sequence if the differences between
 successive numbers strictly alternate between positive and negative.
  The first difference (if one exists) may be either positive or negative.
   A sequence with fewer than two elements is trivially a wiggle sequence.

For example, [1,7,4,9,2,5] is a wiggle sequence because
 the differences (6,-3,5,-7,3) are alternately positive and negative.
  In contrast, [1,4,7,2,5] and [1,7,4,5,5] are not wiggle sequences,
   the first because its first two differences are positive and
    the second because its last difference is zero.

Given a sequence of integers, return the length of the longest subsequence that is
 a wiggle sequence. A subsequence is obtained by deleting some number of elements
  (eventually, also zero) from the original sequence,
   leaving the remaining elements in their original order.

Examples:
Input: [1,7,4,9,2,5]
Output: 6
The entire sequence is a wiggle sequence.

Input: [1,17,5,10,13,15,10,5,16,8]
Output: 7
There are several subsequences that achieve this length. One is [1,17,10,13,10,16,8].

Input: [1,2,3,4,5,6,7,8,9]
Output: 2
Follow up:
Can you do it in O(n) time?

https://leetcode.com/problems/valid-perfect-square/
```

Solution 1

A dp solution with $O(n^2)$, unfortunately this is the worst solution in time complexity.

Listing 366: Solution 1 - DP

```
public class Solution {
    public int wiggleMaxLength(int[] nums) {
        if (nums.length == 0){
            return 0;
        }
        int maxLength = -1;
        // last diff, 0 for positive, 1 for negative
        int[][] dp = new int[nums.length][2];

        for (int idx = 0; idx < nums.length ; idx ++){
            dp[idx][0] = 1;
            dp[idx][1] = 1;
            for (int formerIdx = idx - 1 ; formerIdx >= 0 ; formerIdx --){
```

```
                int maxPositive = -1, maxNegative = -1;
                int diff = nums[idx] - nums[formerIdx];
                if (diff > 0){
                    maxPositive = Math.max(dp[formerIdx][1], maxPositive);
                }
                else if (diff < 0){
                    maxNegative = Math.max(dp[formerIdx][0], maxPositive);
                }
                else{
                    continue;
                }
                dp[idx][0] = Math.max(dp[idx][0], maxPositive + 1);
                dp[idx][1] = Math.max(dp[idx][1], maxNegative + 1);
                maxLength = Math.max(maxLength , Math.max(dp[idx][0], dp[idx][1]));
            }
        }

        return Math.max(maxLength, 1);
    }
}
```

Solution 2

A $O(n)$ solution can be found in Greedy or Math. As for greedy, just find ; while as for math, just find the switching points, that is exactly the number of maxLength.

Listing 367: Solution 2 - Greedy

```
// not yet completed
```

Listing 368: Solution 2 - Math

```
// not yet completed
```

Problem 280

387 First Unique Character in a String

```
Given a string, find the first non-repeating character in it and return it's index.
 If it doesn't exist, return -1.

Examples:

s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
Note: You may assume the string contain only lowercase letters.

https://leetcode.com/problems/first-unique-character-in-a-string/
```

Solution

A simple problem, array counting its count and index is enough.

Listing 369: Solution

```
public class Solution {
    public int firstUniqChar(String s) {
        int[] appearances = new int[26];
        for (int aIdx = 0 ; aIdx < 26; aIdx ++){
            appearances[aIdx] = -1;
        }
        for (int idx = 0 ; idx < s.length() ; idx ++){
            int letterIdx = s.charAt(idx) - 'a';
            if (appearances[letterIdx] == -1){
                appearances[letterIdx] = idx;
            }
            else if (appearances[letterIdx] >= 0){
                appearances[letterIdx] = -2;
            }
        }
        int minIdx = Integer.MAX_VALUE;
        for (int aIdx = 0 ; aIdx < 26 ; aIdx ++){
            if (appearances[aIdx] >= 0){
                minIdx = Math.min(appearances[aIdx], minIdx);
            }
        }
        return (minIdx == Integer.MAX_VALUE) ? -1 : minIdx;
    }
}
```

Problem 281

365 Water and Jug Problem

```
You are given two jugs with capacities x and y litres.
 There is an infinite amount of water supply available.
  You need to determine whether it is possible to measure
   exactly z litres using these two jugs.

If z liters of water is measurable,
 you must have z liters of water contained within one or both buckets
  by the end.

Operations allowed:

Fill any of the jugs completely with water.
Empty any of the jugs.
Pour water from one jug into another till the other jug is
 completely full or the first jug itself is empty.
Example 1: (From the famous "Die Hard" example)

Input: x = 3, y = 5, z = 4
Output: True
Example 2:

Input: x = 2, y = 6, z = 5
Output: False

https://leetcode.com/problems/water-and-jug-problem/
```

Solution

If you know the solution, the problem is not so difficult. Just check if z is divisible by gcd(x,y).

BUT! Mention so may edge cases, like x == 0, y == 0, z == 0, and x + y < z(For example, for 1,2,5, though 1+2+2=5, it cannot be achieved)

Listing 370: Solution

```
public class Solution {
    public boolean canMeasureWater(int x, int y, int z) {
        if (z == 0){
            return true;
        }
        if (x + y < z){
            return false;
        }
        if (x == 0 && y == 0){
            return false;
        }
        int cd = gcd(x, y);
        return (z % cd == 0);
    }

    private int gcd(int x, int y){
```

```
        if (x < y){
            return gcd(y, x);
        }
        if (y == 0){
            return x;
        }
        int quo = 0, rem = 0;
        do{
            rem = x % y;
            x = y;
            y = rem;
        }
        while (rem != 0);
        return x;
    }
}
```

Problem 282

372 Super Pow

```
Your task is to calculate ab mod 1337 where a is a positive integer
 and b is an extremely large positive integer given in the form of an array.

Example1:

a = 2
b = [3]

Result: 8
Example2:

a = 2
b = [1,0]

Result: 1024

https://leetcode.com/problems/super-pow/
```

Solution 1

Fast Exponentiation is one solution, just aware of using high-precision solution to deal with the div 2 calculation. And, remember to mod 1337 everywhere.

Listing 371: Solution 1 - Fast Exponentiation

```
public class Solution {
    public int superPow(int a, int[] b) {
        if (isZero(b)){
            return 1;
        }
        boolean isEven = (b[b.length - 1] % 2 == 0);
        int product = superPow(a, div2(b)) % 1337;
        if (isEven){
            return product * product % 1337;
        }
        else{
            return (product * product % 1337 * (a % 1337)) % 1337;
        }
    }
    private boolean isZero(int[] b){
        for (int digit : b){
            if (digit != 0){
                return false;
            }
        }
        return true;
    }
    private int[] div2(int[] b){
        int carry = 0;
        for (int idxB = 0 ; idxB < b.length ; idxB ++){
            b[idxB] = carry * 10 + b[idxB];
```

```
            carry = b[idxB] % 2;
            b[idxB] /= 2;
        }
        return b;
    }
}
```

Solution 2

I didn't get the solution... Copy the solution here, for future inspiration.

The main idea is cashed on the repeated pattern of the remainder of a^b.

As long as we know the length of the pattern m, we just have to find an index point of this pattern based on b mod m.

In addition, if a > 1337, we can let a = a mod 1337.

Because if we let a = (1337x + c) where c = a mod 1337,

(1337x + c)(1337x + c)(1337x + c)...(1337x + c) mod 1337 == ccc...c mod 1337.

It iterates at most 1337 times to find the circle. And find the remainder to the loop size. And then look up in the history to find the final answer.

Listing 372: Solution 2 - the Remainder Repeat Pattern

```
public class Solution {
    int DIV = 1337;
    List<Integer> findLoop(int a){
        List<Integer> index = new ArrayList<>();
        boolean[] set = new boolean[DIV];
        int rem = a % DIV;
        while ( ! set[rem] ) {
            set[rem]=true;
            index.add(rem);
            rem = (rem*a) % DIV;
        }
        return index;
    }
    int modBy(int[] b, int m){
        int rem = 0;
        for (int i=0; i < b.length; i++) {
            rem = (rem*10+b[i]) % m;
        }
        return rem;
    }
    public int superPow(int a, int[] b) {
        if (a==0 || a==DIV || b==null || b.length == 0) return 0;
        if (a==1) return 1;
        if (a > DIV) return superPow( a % DIV, b);
        List<Integer> index = findLoop(a);
        int loopsize = index.size();
        int rem = modBy(b, loopsize);
        rem = rem==0? loopsize: rem;
        return index.get(rem-1);
    }
}
```

Problem 283

386 Lexicographical Numbers

```
Given an integer n, return 1 - n in lexicographical order.

For example, given 13, return: [1,10,11,12,13,2,3,4,5,6,7,8,9].

Please optimize your algorithm to use less time and space.
 The input size may be as large as 5,000,000.

https://leetcode.com/problems/lexicographical-numbers/
```

Solution 1

Just a dfs problem, not difficult at all. However, optimization on the way.

Listing 373: Solution 1 - DFS

```
public class Solution {
    List<Integer> result = new ArrayList<Integer>();
    public List<Integer> lexicalOrder(int n) {
        traverse(n, 0);
        return result;
    }
    private boolean traverse(int n, int curNum){
        if (curNum > n){
            return false;
        }
        if (curNum != 0){
            result.add(curNum);
        }
        for (int i = 0 ; i <= 9 ; i++){
            if (curNum == 0 && i == 0){
                continue;
            }
            if (!traverse(n, curNum * 10 + i)){
                break;
            }
        }
        return true;
    }
}
```

Solution 2

To be continued... Needs optimization.

Problem 284

347 Top K Frequent Elements

```
Given a non-empty array of integers, return the k most frequent elements.

For example,
Given [1,1,1,2,2,3] and k = 2, return [1,2].

Note:
You may assume k is always valid, 1 <= k <= number of unique elements.
Your algorithm's time complexity must be better than O(n log n),
 where n is the array's size.

https://leetcode.com/problems/top-k-frequent-elements/
```

Solution

For $O(nlogn)$, heap sort, merge sort and quick sort come into mind. When PriorityQueue provide us with built-in heap sort, we use it to solve the problem.

Listing 374: Solution

```
public class Solution {
    public List<Integer> topKFrequent(int[] nums, int k) {
        List<Integer> result = new ArrayList<Integer>();
        PriorityQueue<Entry> queue = new PriorityQueue<Entry>(
            (a,b) -> b.count - a.count);
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int num : nums){
            if (map.containsKey(num)){
                map.put(num, map.get(num) + 1);
            }
            else{
                map.put(num, 1);
            }
        }
        for (int key : map.keySet()){
            Entry entry = new Entry();
            entry.value = key;
            entry.count = map.get(key);
            queue.offer(entry);
        }

        for (int idx = 0 ; idx < k ; idx ++){
            Entry entry = queue.poll();
            result.add(entry.value);
        }
        return result;
    }
}
class Entry{
    public int value;
    public int count;
}
```

Problem 284 continued on next page...

Problem 285

388 Longest Absolute File Path

```
Suppose we abstract our file system by a string in the following manner:

The string "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext" represents:

dir
    subdir1
    subdir2
        file.ext
The directory dir contains an empty sub-directory subdir1
 and a sub-directory subdir2 containing a file file.ext.

The string
 "dir\n\tsubdir1\n\t\tfile1.ext\n\t
 \tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"
  represents:

dir
    subdir1
        file1.ext
        subsubdir1
    subdir2
        subsubdir2
            file2.ext
The directory dir contains two sub-directories subdir1 and subdir2.
 subdir1 contains a file file1.ext and
  an empty second-level sub-directory subsubdir1.
   subdir2 contains a second-level sub-directory subsubdir2 containing
    a file file2.ext.

We are interested in finding the longest (number of characters) absolute path
 to a file within our file system.
  For example, in the second example above,
   the longest absolute path is "dir/subdir2/subsubdir2/file2.ext",
    and its length is 32 (not including the double quotes).

Given a string representing the file system in the above format,
 return the length of the longest absolute path to file
  in the abstracted file system. If there is no file in the system, return 0.

Note:
The name of a file contains at least a . and an extension.
The name of a directory or sub-directory will not contain a ..
Time complexity required: O(n) where n is the size of the input string.

Notice that a/aa/aaa/file1.txt is not the longest file path,
 if there is another path aaaaaaaaaaaaaaaaaaaaa/sth.png.

https://leetcode.com/problems/longest-absolute-file-path/
```

Problem 285 continued on next page...

Solution

Used number of '\t' as array index, a 1D array is enough to store the value since the structure is like a tree.

Listing 375: Solution

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverse(ListNode start, ListNode end){
        if (start == null)
            return null;
        ListNode cur = start, next = cur.next;
        cur.next = null;
        while (next != null && next != end){
            ListNode backup = next.next;
            next.next = cur;
            cur = next;
            next = backup;
        }
        return cur;
    }
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode cur = head;
        ListNode last = cur;
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode tail = dummy;
        while (cur != null){
            int count = k;
            last = cur;
            while (count-- > 0){
                cur = cur.next;
                if (cur == null)
                    break;
            }
            if (count > 0 && cur == null){
                tail.next = last;
                return dummy.next;
            }
            ListNode newPart = reverse(last, cur);
            tail.next = newPart;
            tail = last;
        }
        return dummy.next;
    }
}
```

Problem 286

354 Russian Doll Envelopes

```
You have a number of envelopes with widths and heights given as
 a pair of integers (w, h).
  One envelope can fit into another if and only if
   both the width and height of one envelope
    is greater than the width and height of the other envelope.

What is the maximum number of envelopes can you Russian doll?
 (put one inside other)

Example:
Given envelopes = [[5,4],[6,4],[6,7],[2,3]],
 the maximum number of envelopes you can Russian doll is
  3 ([2,3] => [5,4] => [6,7]).

https://leetcode.com/problems/russian-doll-envelopes/
```

Solution

A very interesting problem. First, sort the envelopes according to no matter width or height, let's say width, when 2 envelopes are of the same width, put the one with larger height in the front. Then the problem is converted to 'Longest Increasing Subsequence'(and now we know why we put larger ones in front, since envelopes with same width cannot enclose each other, we use the sequence of larger $->$ smaller to avoid enclosing envelopes with same width). For 'Longest Increasing Subsequence', there is a $O(n^2)$ DP solution and $O(nlogn)$ Binary Search solution.

Listing 376: Solution

```java
public class Solution {
    public int maxEnvelopes(int[][] envelopes) {
        quickSort(envelopes, 0, envelopes.length - 1);
        List<Integer> increasing = new ArrayList<Integer>();
        for (int idx = 0 ; idx < envelopes.length ; idx ++){
            if (increasing.size() == 0
                || envelopes[idx][1] > increasing.get(increasing.size() - 1)){
                increasing.add(envelopes[idx][1]);
            }
            else{
                // binary search
                int properIdx = binarySearch(increasing, 0,
                 increasing.size() - 1, envelopes[idx][1]);
                if (properIdx < increasing.size()){
                    increasing.set(properIdx, envelopes[idx][1]);
                }
            }
        }
        return increasing.size();
    }

    private void quickSort(int[][] envelopes, int start, int end){
        if (start >= end){
```

```
            return;
        }
        int pivot = envelopes[start][0];
        int pivot2 = envelopes[start][1];
        int low = start, high = end;
        while (low < high){
            while (low < high && (envelopes[high][0] > pivot
             || (envelopes[high][0] == pivot && envelopes[high][1] <= pivot2))){
                high --;
            }
            envelopes[low][0] = envelopes[high][0];
            envelopes[low][1] = envelopes[high][1];

            while (low < high && (envelopes[low][0] < pivot
             || (envelopes[low][0] == pivot && envelopes[low][1] >= pivot2))){
                low ++;
            }
            envelopes[high][0] = envelopes[low][0];
            envelopes[high][1] = envelopes[low][1];
        }
        envelopes[low][0] = pivot;
        envelopes[low][1] = pivot2;

        quickSort(envelopes, start, low - 1);
        quickSort(envelopes, low + 1, end);
    }

    private int binarySearch(List<Integer> list, int start, int end, int target){
        int low = start, high = end;
        int mid = start + (end - start) / 2;
        while (low < high){
            mid = low + (high - low) / 2;
            if (list.get(mid) < target){
                low = mid + 1;
            }
            else if (list.get(mid) > target){
                high = mid - 1;
            }
            else{
                return mid;
            }
        }
        if (list.get(low) < target){
            return low + 1;
        }
        else{
            return low;
        }
    }
}
```

Problem 287

389 Find the Difference

```
Given two strings s and t which consist of only lowercase letters.
String t is generated by random shuffling string s and
 then add one more letter at a random position.
Find the letter that was added in t.
Example:
Input:
s = "abcd"
t = "abcde"

Output:
e

Explanation:
'e' is the letter that was added.

https://leetcode.com/problems/find-the-difference/
```

Solution - Array

Quite an easy problem. However, there are still some interesting things in it. First is a plain solution with array.

Listing 377: Solution 1

```
public class Solution {
    public char findTheDifference(String s, String t) {
        int count[] = new int[26];
        char[] arr = s.toCharArray();
        for (char ch : arr){
            count[ch - 'a']++;
        }
        arr = t.toCharArray();
        for (char ch : arr){
            count[ch - 'a']--;
            if (count[ch - 'a'] < 0){
                return ch;
            }
        }
        return 'a';
    }
}
```

Solution - Bit manipulation

Can you still remember 'Single Number' series? Can you find it similar between that series and this one?

Listing 378: Solution 2

```
public class Solution {
    public char findTheDifference(String s, String t) {
        char c = 0;
```

Problem 287 continued on next page...

```
        for (char ch : s.toCharArray()){
            c ^= ch;
        }
        for (char ch : t.toCharArray()){
            c ^= ch;
        }
        return c;
    }
}
```

Problem 288

382 Linked List Random Node

```
Given a singly linked list, return a random node's value from the linked list.
 Each node must have the same probability of being chosen.

Follow up:
What if the linked list is extremely large and its length is unknown to you?
 Could you solve this efficiently without using extra space?

Example:
    // Init a singly linked list [1,2,3].
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    Solution solution = new Solution(head);

    // getRandom() should return either 1, 2, or 3 randomly.
    Each element should have equal probability of returning.
    solution.getRandom();

https://leetcode.com/problems/linked-list-random-node/
```

Solution - Regular and Follow-up

For this problem, the solution at first sight is to calculate the length of data stream(number of elements inside stream) and get random number based on that. So the code seems like this:

Listing 379: Solution 1

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    int length = 1;
    ListNode head;
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null,
        so it contains at least one node. */
    public Solution(ListNode head) {
        ListNode cur = head;
        while (cur.next != null){
            cur = cur.next;
            length ++;
        }
        this.head = head;
    }

    /** Returns a random node's value. */
```

```
    public int getRandom() {
        int rand = (int) Math.ceil(Math.random() * length) - 1;
        // System.out.println(rand + " in " + length);
        ListNode cur = head;
        while (rand > 0){
            cur = cur.next;
            rand --;
        }
        return cur.val;
    }
}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(head);
 * int param_1 = obj.getRandom();
 */
```

HOWEVER! When the stream is extremely large so that we are not able to put it into memory, we need to refer to some other solutions. Here is the option, the Reservoir Sampling.

Here comes the references, for Chinese version, visit
http://blog.jobbole.com/42550/
And for English version, visit
http://blog.cloudera.com/blog/2013/04/hadoop-stratified-randosampling-algorithm/

The main thought is to preserve only one number, e.g.: when 1,2,3 comes from stream:
When only 1 comes, it is definitely 100% picked;
1, 2 come, both have 1/2=50% probability to be picked out, in other words, the number has 1/2 probability to be changed from 1 to 2;
Next the 3. We give it 1/3 probability to be chosen to substitute whatever we have stored, 1 or 2. Then how about the chance for 1 and 2 to be replaced? 2/3(not substituted by 3)*1/2(chosen at previous step)
....
For controlling the chance to 1/n, we use $n == newRandom().nextInt(n+1)$

Listing 380: Solution 2

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    ListNode head;
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null,
        so it contains at least one node. */
    public Solution(ListNode head) {
        this.head = head;
```

```
    }

    /** Returns a random node's value. */
    public int getRandom() {
        Random rand = new Random();
        int idx = 0;
        int result = 0;
        ListNode cur = head;
        while (cur != null){
            if (rand.nextInt(idx + 1) == idx){
                result = cur.val;
            }
            idx ++;
            cur = cur.next;
        }
        return result;
    }
}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(head);
 * int param_1 = obj.getRandom();
 */
```

Problem 289

352 Data Stream as Disjoint Intervals

```
Given a data stream input of non-negative integers a1, a2, ..., an, ...,
 summarize the numbers seen so far as a list of disjoint intervals.

For example, suppose the integers from the data stream are 1, 3, 7, 2, 6, ...,
 then the summary will be:
[1, 1]
[1, 1], [3, 3]
[1, 1], [3, 3], [7, 7]
[1, 3], [7, 7]
[1, 3], [6, 7]

Follow up:
What if there are lots of merges and the number of disjoint intervals
 are small compared to the data stream's size?

https://leetcode.com/problems/data-stream-as-disjoint-intervals/
```

Solution - At first sight

Use binary search to find the correct insertion position, then do the insertion or merge(there are conditions when number already exists, can be merged into two Intervals, can be merged into one side or create a new Interval(maybe at the very beginning, in the middle or in the end), which forms the difficulty of the problem).

Listing 381: Solution 1

```java
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class SummaryRanges {
    List<Interval> intervals = new ArrayList<Interval>();
    /** Initialize your data structure here. */
    public SummaryRanges() {

    }

    public void addNum(int val) {

        if (intervals.size() == 0){
            intervals.add(new Interval(val, val));
            return;
        }
        if (val > intervals.get(intervals.size() - 1).end){
            if (val != intervals.get(intervals.size() - 1).end + 1){
```

```
                intervals.add(new Interval(val, val));
            }
            else{
                intervals.get(intervals.size() - 1).end = val;
            }
            return;
        }
        if (val < intervals.get(0).start){
            if (val != intervals.get(0).start - 1){
                intervals.add(0, new Interval(val, val));
            }
            else{
                intervals.get(0).start = val;
            }
            return;
        }

        int idx = binarySearch(intervals, val, 0, intervals.size() - 1);

        System.out.println("Adding " + val);
        if (val <= intervals.get(idx).end && val >= intervals.get(idx).start){
            return;
        }

        if (intervals.get(idx).end < val){
            idx ++;
        }
        if (intervals.get(idx - 1).end + 1 == val
             && intervals.get(idx).start - 1 == val){
            Interval newInterval = new Interval(intervals.get(idx - 1).start,
                                                intervals.get(idx).end);
            intervals.remove(idx);
            intervals.set(idx - 1, newInterval);
        }
        else if (intervals.get(idx - 1).end + 1 == val){
            intervals.get(idx - 1).end  = val;
        }
        else if (intervals.get(idx).start - 1 == val){
            intervals.get(idx).start = val;
        }
        else{
            Interval newInterval = new Interval(val, val);
            intervals.add(idx, newInterval);
        }
    }

    public List<Interval> getIntervals() {
        return intervals;
    }

    private int binarySearch(List<Interval> intervals, int val, int start, int end){
        while (start < end){
            int mid = start + (end - start) / 2;
```

```
            if (val < intervals.get(mid).start){
                end = mid - 1;
                continue;
            }
            else if (val <= intervals.get(mid).end){
                return mid;
            }
            if (val > intervals.get(mid).end){
                start = mid + 1;
                continue;
            }
            else if (val >= intervals.get(mid).start){
                return mid;
            }
        }

        return start;
    }
}

/**
 * Your SummaryRanges object will be instantiated and called as such:
 * SummaryRanges obj = new SummaryRanges();
 * obj.addNum(val);
 * List<Interval> param_2 = obj.getIntervals();
 */
```

Solution - Follow-up

To be continued...

Listing 382: Solution 2

```
// To be continued...
```

Problem 290

363 Max Sum of Rectangle No Larger Than K

```
Given a non-empty 2D matrix matrix and an integer k,
 find the max sum of a rectangle in the matrix
  such that its sum is no larger than k.

Example:
Given matrix = [
  [1,  0, 1],
  [0, -2, 3]
]
k = 2

The answer is 2.
 Because the sum of rectangle [[0, 1], [-2, 3]] is 2
  and 2 is the max number no larger than k (k = 2).

Note:
The rectangle inside the matrix must have an area > 0.
What if the number of rows is much larger than the number of columns?

https://leetcode.com/problems/max-sum-of-sub-matrix-no-larger-than-k/
```

Solution

Though it's a bit difficult, I think it is interesting and worth thinking. The solution makes use of TreeSet to find sum in range.

First for simplification, we think of the 1D solution. It's not that difficult. DP tells us that when referring to sum[i to j], what we only need is sum[j] - sum[i - 1](remember to check boundary). Then, we need TreeSet(BST, to be more specific) to store each sum, and when a new sum comes, compare it with older sums to satisfy sum[curIdx] - sum[formerIdx] <= k, namely sum[curIdx] = treeSet.floor(sum[formerIdx] + k).

As for the 2D part, it's still in DP vision. Remember what we have done in Problem 85 'Maximal Rectangle'? We just use similar thoughts with that problem and what we used in this problem to get sums for each column based on their row index. Then, apply the 1D solution on each row.

A thing is that the test cases are special, they match the 2nd Note in description that the number of rows is much larger than the number of columns? That's why you can find in the solution below that the calculation process seems a bit weird while the code commented seems more normal.(The 'normal' code costs 800ms to solve a case when 'weird' one is solving the same in 100ms). Honestly, I think that's what shouldn't be happening.

Listing 383: Solution

```
public class Solution {
    public int maxSumSubmatrix(int[][] matrix, int k) {
        if (matrix.length == 0 || matrix[0].length == 0){
            return 0;
        }
        int[][] sum = new int[matrix.length][matrix[0].length];
        for (int i = 0 ; i < matrix[0].length ; i ++){
            for (int j = 0 ; j < matrix.length ; j ++){
```

```
                if (i == 0){
                    // sum[i][j] = matrix[i][j];
                    sum[j][i] = matrix[j][i];
                }
                else{
                    // sum[i][j] = sum[i - 1][j] + matrix[i][j];
                    sum[j][i] = sum[j][i-1] + matrix[j][i];
                }
            }
        }

        int max = Integer.MIN_VALUE;
        for (int i = 0 ; i < matrix[0].length ; i ++){
            for (int j = i ; j < matrix[0].length ; j ++){
                TreeSet<Integer> treeSet = new TreeSet<Integer>();
                int totalSum = 0;
                for (int l = 0 ; l < matrix.length ; l ++){

                    totalSum += sum[l][j] - ((i != 0) ? sum[l][i - 1] : 0);
                    // totalSum += sum[j][l] - ((i != 0) ? sum[i - 1][l] : 0);
                    if (totalSum <= k){
                        max = Math.max(totalSum, max);
                    }
                    Integer smaller = treeSet.ceiling(totalSum - k);
                    if (smaller != null){
                        max = Math.max(max, totalSum - smaller);
                    }
                    treeSet.add(totalSum);
                }
            }
        }

        return max;
    }
}
```

Problem 291

384 Shuffle an Array

```
Shuffle a set of numbers without duplicates.

Example:
// Init an array with set 1, 2, and 3.
int[] nums = {1,2,3};
Solution solution = new Solution(nums);

// Shuffle the array [1,2,3] and return its result.
 Any permutation of [1,2,3] must equally likely to be returned.
solution.shuffle();

// Resets the array back to its original configuration [1,2,3].
solution.reset();

// Returns the random shuffling of array [1,2,3].
solution.shuffle();

https://leetcode.com/problems/shuffle-an-array/
```

Solution

Using "Knuth Shuffle"(or to be more specific, the FisherYates shuffle since they are the real ones who invent this).

For reference: https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

The thought is simple, for number at index i, generate random integer between [0,i](using rand.nextInt()), and swap numbers at index i and random number.

Note that this kind of shuffle is not in a uniform probability distribution and making it evenly spread is not difficult, just change the [0,i] into [0,len].

Listing 384: Solution

```
public class Solution {
    int[] nums;
    public Solution(int[] nums) {
        this.nums = nums;
    }

    /** Resets the array to its original configuration and return it. */
    public int[] reset() {
        return Arrays.copyOf(nums, nums.length);
    }

    /** Returns a random shuffling of the array. */
    public int[] shuffle() {
        int[] result = Arrays.copyOf(nums, nums.length);
        for (int idx = 1; idx < result.length ; idx ++){
            int swapIdx = new Random().nextInt(idx + 1);

            int swap = result[swapIdx];
            result[swapIdx] = result[idx];
            result[idx] = swap;
```

Problem 291 continued on next page...

```
        }
        return result;
    }
}

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(nums);
 * int[] param_1 = obj.reset();
 * int[] param_2 = obj.shuffle();
 */
```

Problem 292

380 Insert Delete GetRandom O(1)

```
Design a data structure that supports all following operations
 in AVERAGE O(1) time.

    insert(val): Inserts an item val to the set if not already present.
    remove(val): Removes an item val from the set if present.
    getRandom: Returns a random element from current set of elements.
     Each element must have the same probability of being returned.

Example:
    // Init an empty set.
    RandomizedSet randomSet = new RandomizedSet();

    // Inserts 1 to the set. Returns true as 1 was inserted successfully.
    randomSet.insert(1);

    // Returns false as 2 does not exist in the set.
    randomSet.remove(2);

    // Inserts 2 to the set, returns true. Set now contains [1,2].
    randomSet.insert(2);

    // getRandom should return either 1 or 2 randomly.
    randomSet.getRandom();

    // Removes 1 from the set, returns true. Set now contains [2].
    randomSet.remove(1);

    // 2 was already in the set, so return false.
    randomSet.insert(2);

    // Since 1 is the only number in the set, getRandom always return 1.
    randomSet.getRandom();

https://leetcode.com/problems/insert-delete-getrandom-o1/
```

Solution

Actually a smart problem. For an O(1) insert and remove operation, it's quite hard to find a data single structure to satisfy both. Array is in O(1) insert but remove needs more time(since we have to put elements behind the deleted one into proper position, namely move forward for 1 index). So wait! We were using the word 'behind'! What if the element is just at the end of array? So that's the core thinking of our solution, when we want to remove an element, swap it with the last element in array(when we need to support checking existence in array, an extra HashMap is needed), remove it and maintain the actual size.

The solution is not even the most interesting part. We can see there is a word 'average' in problem description which is actually not in the 1st version of it. This is related to a discussion here.
https://discuss.leetcode.com/topic/53193/are-hash-tables-ok-here-they-re-not-really-o-1-ar

Main conflict is that actually, Hashtables(Map/Set) are not O(1) in inserting and removing, actually they are **amortized** O(1)(because of the array expansion and removal issues). Detailed discussion can be found in url above, which is quite interesting, I think.

Listing 385: Solution

```
public class RandomizedSet {
    static int MAX_LENGTH = 1000000;
    int[] elements;
    int size;
    HashMap<Integer, Integer> positions = new HashMap<Integer, Integer>();
    Random rand = new Random();;
    /** Initialize your data structure here. */
    public RandomizedSet() {
        elements = new int[MAX_LENGTH];
        size = 0;
    }

    /** Inserts a value to the set. Returns true if the set
     did not already contain the specified element. */
    public boolean insert(int val) {
        if (positions.containsKey(val)){
            return false;
        }
        positions.put(val, size);
        elements[size++] = val;

        return true;
    }

    /** Removes a value from the set. Returns true if the set
     contained the specified element. */
    public boolean remove(int val) {
        if (!positions.containsKey(val)){
            return false;
        }
        // swap
        int position = positions.get(val);
        if (position != size - 1){
            positions.put(elements[size - 1], position);
            int swap = elements[position];
            elements[position] = elements[size - 1];
            elements[size - 1] = swap;
        }
        size --;

        positions.remove(val);
        return true;
    }

    /** Get a random element from the set. */
    public int getRandom() {
        int idx = rand.nextInt(size);
        return elements[idx];
    }
}

/**
```

```
 * Your RandomizedSet object will be instantiated and called as such:
 * RandomizedSet obj = new RandomizedSet();
 * boolean param_1 = obj.insert(val);
 * boolean param_2 = obj.remove(val);
 * int param_3 = obj.getRandom();
 */
```

Problem 293

381 Insert Delete GetRandom O(1) - Duplicates allowed

```
Design a data structure that supports all following operations
 in average O(1) time.
Note: Duplicate elements are allowed.

insert(val): Inserts an item val to the collection.
remove(val): Removes an item val from the collection if present.
getRandom: Returns a random element from current collection of elements.
 The probability of each element being returned is linearly
  related to the number of same value the collection contains.

    Example:
    // Init an empty collection.
    RandomizedCollection collection = new RandomizedCollection();

    // Inserts 1 to the collection.
    // Returns true as the collection did not contain 1.
    collection.insert(1);

    // Inserts another 1 to the collection.
    // Returns false as the collection contained 1. Collection now contains [1,1].
    collection.insert(1);

    // Inserts 2 to the collection, returns true.
    // Collection now contains [1,1,2].
    collection.insert(2);

    // getRandom should return 1 with the probability 2/3,
    // and returns 2 with the probability 1/3.
    collection.getRandom();

    // Removes 1 from the collection, returns true.
    // Collection now contains [1,2].
    collection.remove(1);

    // getRandom should return 1 and 2 both equally likely.
    collection.getRandom();

https://leetcode.com/problems/insert-delete-getrandom-o1-duplicates-allowed/
```

Solution

Not so different as the previous Problem "Insert Delete GetRandom O(1)". Largest difference may lies in the fact that we now need a Set rather than just a number to preserve the positions of each elements; Pick the last occurrence of element will save a few effort; Don't forget to maintain the remaining position information as long as don't forgetting to remove when no such elements exists.

Listing 386: Solution

```
public class RandomizedCollection {
    static int MAX_LENGTH = 1000000;
```

```
    int[] elements;
    int size;
    HashMap<Integer, Set<Integer>> positions
     = new HashMap<Integer, Set<Integer>>();
    Random rand = new Random();

    /** Initialize your data structure here. */
    public RandomizedCollection() {
        elements = new int[MAX_LENGTH];
        size = 0;
    }

    /** Inserts a value to the collection.
     Returns true if the collection did not
      already contain the specified element. */
    public boolean insert(int val) {
        boolean result = true;
        if (positions.containsKey(val)){
            result = false;
        }
        Set<Integer> poss = positions.containsKey(val)
         ? positions.get(val) : new HashSet<Integer>();
        poss.add(size);
        positions.put(val, poss);
        elements[size++] = val;

        return result;
    }

    /** Removes a value from the collection.
     Returns true if the collection contained the specified element. */
    public boolean remove(int val) {
        if (!positions.containsKey(val)){
            return false;
        }
        // pick one position
        Set<Integer> poss = positions.get(val);
        if (poss.isEmpty()){
            return false;
        }
        Iterator<Integer> ite = poss.iterator();
        int position = ite.next();
        while (ite.hasNext()){
            position = ite.next();
        }

        // swap
        if (position != size - 1){
            Set<Integer> swapPoss = positions.get(elements[size - 1]);
            swapPoss.remove(size - 1);
            swapPoss.add(position);
            positions.put(elements[size - 1], swapPoss);

```

```
            int swap = elements[position];
            elements[position] = elements[size - 1];
            elements[size - 1] = swap;
        }
        size --;
        poss.remove(position);
        if (poss.size() == 0){
            positions.remove(val);
        }
        else{
            positions.put(val, poss);
        }

        return true;
    }

    /** Get a random element from the collection. */
    public int getRandom() {
        int idx = rand.nextInt(size);
        return elements[idx];
    }

}

/**
 * Your RandomizedCollection object will be instantiated and called as such:
 * RandomizedCollection obj = new RandomizedCollection();
 * boolean param_1 = obj.insert(val);
 * boolean param_2 = obj.remove(val);
 * int param_3 = obj.getRandom();
 */
```

Problem 294

385 Mini Parser

```
Given a nested list of integers represented as a string,
 implement a parser to deserialize it.
Each element is either an integer, or a list
 -- whose elements may also be integers or other lists.

Note: You may assume that the string is well-formed:
    String is non-empty.
    String does not contain white spaces.
    String contains only digits 0-9, [, - ,, ].

Example 1:
Given s = "324",

You should return a NestedInteger object which contains a single integer 324.

Example 2:
Given s = "[123,[456,[789]]]",

Return a NestedInteger object containing a nested list with 2 elements:

1. An integer containing value 123.
2. A nested list containing two elements:
    i.  An integer containing value 456.
    ii. A nested list with one element:
         a. An integer containing value 789.

https://leetcode.com/problems/mini-parser/
```

Solution

A typical recursive/stack problem, however I was struggling with it for a long time(Actually, I'm still struggling). The train of thought is not difficult, just split the string, use a counter to check whether the brackets are in pairs, if yes, then the comma ',' shall be the splitter.

Two kinds of solutions are listed as below, one recursive and one iterative using stack.

Listing 387: Solution - Recursion

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * public interface NestedInteger {
 *     // Constructor initializes an empty nested list.
 *     public NestedInteger();
 *
 *     // Constructor initializes a single integer.
 *     public NestedInteger(int value);
 *
 *     // @return true if this NestedInteger holds a single integer,
 *           rather than a nested list.
 *     public boolean isInteger();
```

```
 *
 *     // @return the single integer that this NestedInteger holds,
 *           if it holds a single integer
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // Set this NestedInteger to hold a single integer.
 *     public void setInteger(int value);
 *
 *     // Set this NestedInteger to hold a nested list
 *           and adds a nested integer to it.
 *     public void add(NestedInteger ni);
 *
 *     // @return the nested list that this NestedInteger holds,
 *           if it holds a nested list
 *     // Return null if this NestedInteger holds a single integer
 *     public List<NestedInteger> getList();
 * }
 */
public class Solution {
    public NestedInteger deserialize(String s) {
        if (s.length() == 0){
            return new NestedInteger();
        }
        if (s.charAt(0) != '['){
            return new NestedInteger(Integer.parseInt(s));
        }
        if (s.length() <= 2){
            return new NestedInteger();
        }
        NestedInteger res = new NestedInteger();
        int start = 1, cnt = 0;
        for (int i = 1; i < s.length() ; ++i) {
            if (cnt == 0 && (s.charAt(i) == ',' || i == s.length() - 1)) {
                res.add(deserialize(s.substring(start, i)));
                start = i + 1;
            } else if (s.charAt(i) == '['){
                ++cnt;
            }
            else if (s.charAt(i) == ']'){
                --cnt;
            }
        }
        return res;
    }
}
```

Listing 388: Solution - Stack

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * public interface NestedInteger {
 *     // Constructor initializes an empty nested list.
```

```
 *     public NestedInteger();
 *
 *     // Constructor initializes a single integer.
 *     public NestedInteger(int value);
 *
 *     // @return true if this NestedInteger holds a single integer,
 *               rather than a nested list.
 *     public boolean isInteger();
 *
 *     // @return the single integer that this NestedInteger holds,
 *               if it holds a single integer
 *     // Return null if this NestedInteger holds a nested list
 *     public Integer getInteger();
 *
 *     // Set this NestedInteger to hold a single integer.
 *     public void setInteger(int value);
 *
 *     // Set this NestedInteger to hold a nested list
 *               and adds a nested integer to it.
 *     public void add(NestedInteger ni);
 *
 *     // @return the nested list that this NestedInteger holds,
 *               if it holds a nested list
 *     // Return null if this NestedInteger holds a single integer
 *     public List<NestedInteger> getList();
 * }
 */
public class Solution {
    public NestedInteger deserialize(String s) {
        // not implemented yet
    }
}
```

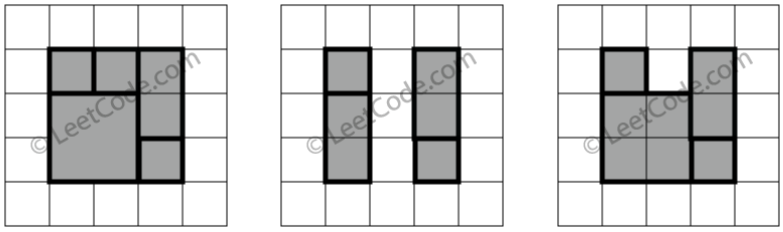
Problem 295

391 Perfect Rectangle

```
Given N axis-aligned rectangles where N > 0,
 determine if they all together form an exact cover of a rectangular region.
Each rectangle is represented as a bottom-left point and a top-right point.
 For example, a unit square is represented as [1,1,2,2].
  (coordinate of bottom-left point is (1, 1) and top-right point is (2, 2)).

Example 1:
rectangles = [
  [1,1,3,3],
  [3,1,4,2],
  [3,2,4,4],
  [1,3,2,4],
  [2,3,3,4]
]

Return true. All 5 rectangles together
 form an exact cover of a rectangular region.



Example 2:
rectangles = [
  [1,1,2,3],
  [1,3,2,4],
  [3,1,4,2],
  [3,2,4,4]
]

Return false. Because there is a gap between the two rectangular regions.



Example 3:
rectangles = [
  [1,1,3,3],
  [3,1,4,2],
  [1,3,2,4],
  [3,2,4,4]
]

Return false. Because there is a gap in the top center.



Example 4:
rectangles = [
  [1,1,3,3],
  [3,1,4,2],
  [1,3,2,4],
```

```
  [2,2,4,4]
]

Return false. Because two of the rectangles overlap with each other.

https://leetcode.com/problems/mini-parser/
```

First let me add the demonstration figures for a better visualization.



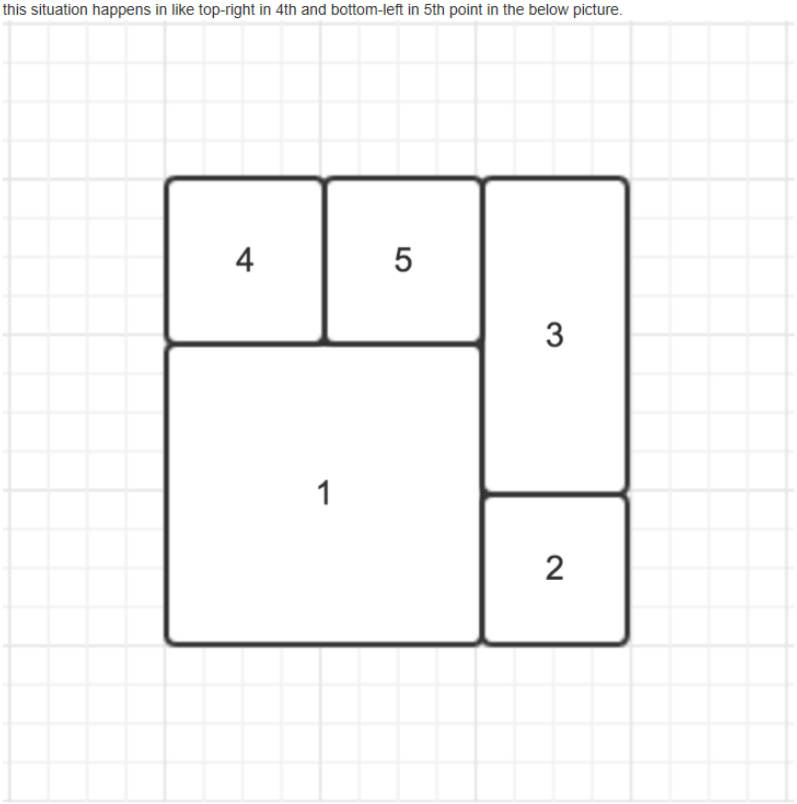
Solution - Sweep line

First solution makes use of a standard geometric search algorithm - sweep line. Basically sort blocks by x-axis $->$ put or remove y-axis values into or from TreeSet for range search $->$ find the overlaps. There are small tricks for better performance like bit manipulation. Screenshot of detailed description of algorithm is as below.

Good idea to solve this problem, I read the code very careful, so I write some to make the code easy to understand the logic.

this way first to sort x-coordinate in order to handle rectangle from left to right
when the time is same then sort by the rect[0], why do like this?

this situation happens in like top-right in 4th and bottom-left in 5th point in the below picture.



Because we want to handle rectangle from left to right and every handle process `there is only one rectangle in the vertical range`, when handle 5th rectangle 4th rect should be removed, handle 2th rect 1th rect should be removed. So we need sort by rect[0] when time is same.

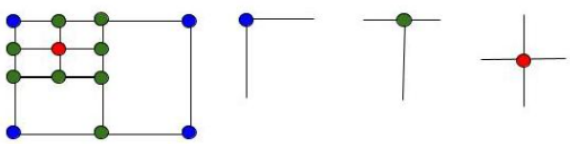
After the traversal sequence is set by `PriorityQueue`, we use the `treeSet` to judge two rect is intersections or not, and we can see every handle process, the yRange should equal the high of the largest rectangle outside if it is true.

Listing 389: Solution - Sweep line

```
public class Solution {
    public boolean isRectangleCover(int[][] rectangles) {
        // not implemented yet
    }
}
```

Solution - Tricks

Idea



Consider how the corners of all rectangles appear in the large rectangle if there's a perfect rectangular cover.
Rule 1: The local shape of the corner has to follow one of the three following patterns

- Corner of the large rectangle (blue): it occurs only once among all rectangles
- T-junctions (green): it occurs twice among all rectangles
- Cross (red): it occurs four times among all rectangles

Rule 2: A point can only be the top-left corner of at most one sub-rectangle. Similarly it can be the top-right/bottom-left/bottom-right corner of at most one sub-rectangle. Otherwise overlaps occur.

Proof of correctness

Obviously, any perfect cover satisfies the above rules. So the main question is whether there exists an input which satisfy the above rules, yet does not compose a rectangle.

First, ***any overlap is not allowed based on the above rules*** because

- aligned overlap like [[0, 0, 1, 1], [0, 0, 2, 2]] are rejected by Rule 2.
- unaligned overlap will generate a corner in the interior of another sub-rectangle, so it will be rejected by Rule 1.

Second, consider the shape of boundary for the combined shape. The cross pattern does not create boundary. The corner pattern generates a straight angle on the boundary, and the T-junction generates a straight border.
So the shape of the union of rectangles has to be rectangle(s).

Finally, if there are more than two non-overlapping rectangles, at least 8 corners will be found, and cannot be matched to the 4 bounding box corners (be reminded we have shown that there is no chance of overlapping).
So the cover has to be a single rectangle if all above rules are satisfied.

Algorithm

- **Step1:** Based on the above idea we maintain a mapping from (x, y)->mask by scanning the sub-rectangles from beginning to end.
  - (x, y) corresponds to corners of sub-rectangles
  - mask is a 4-bit binary mask. Each bit indicates whether there have been a sub-rectangle with a top-left/top-right/bottom-left/bottom-right corner at (x, y). If we see a conflict while updating mask, it suffice to return a false during the scan.
  - In the meantime we obtain the bounding box of all rectangles (which potentially be the rectangle cover) by getting the upper/lower bound of x/y values.
- **Step 2:** Once the scan is done, we can just browse through the unordered_map to check the whether ***the mask corresponds to a T junction / cross, or corner if it is indeed a bounding box corner***.
  (note: my earlier implementation uses counts of bits in mask to justify corners, and this would not work with certain cases as @StefanPochmann points out).

Complexity

The scan in step 1 is O(n) because it loop through rectangles and inside the loop it updates bounding box and unordered_map in O(1) time.

Step2 visits 1 corner at a time with O(1) computations for at most 4n corners (actually much less because either corner overlap or early stopping occurs). So it's also O(n).

This solution description is copied from the hottest topic in Discuss forum for this problem. Actually, there is an easier trick which can solve the problem. There are blue, green and red dots inside graph, an interesting fact is, if they want to form a perfect rectangle, they should follow 2 rules: 1. the total area should be exactly the same as $(max_y - min_y) * (max_x - min_x)$; 2. **There are four and only four corner points which appears once**(namely the blue points).

So we just add every corner points of each rectangle into a set, if it exists, remove it(to remove those points in pairs). Finally, we check 1. the total area; 2. whether the size points remaining is exactly four and

are the min and max points.

Listing 390: Solution - tricks

```java
public class Solution {
    public boolean isRectangleCover(int[][] rectangles) {
        int min_x = Integer.MAX_VALUE, min_y = Integer.MAX_VALUE;
        int max_x = Integer.MIN_VALUE, max_y = Integer.MIN_VALUE;
        int area = 0;
        Set<String> set = new HashSet<String>();
        for (int[] rect : rectangles){
            min_x = Math.min(rect[0], min_x);
            min_y = Math.min(rect[1], min_y);
            max_x = Math.max(rect[2], max_x);
            max_y = Math.max(rect[3], max_y);

            String p1 = rect[0] + "." + rect[1];
            String p2 = rect[0] + "." + rect[3];
            String p3 = rect[2] + "." + rect[1];
            String p4 = rect[2] + "." + rect[3];

            String[] points = {p1, p2, p3, p4};
            for (int idx = 0 ; idx < 4 ; idx ++){
                if (set.contains(points[idx])){
                    set.remove(points[idx]);
                }
                else{
                    set.add(points[idx]);
                }
            }
            area += (rect[3] - rect[1]) * (rect[2] - rect[0]);
        }
        return ((set.size() == 4
            && set.contains(min_x + "." + min_y)
            && set.contains(min_x + "." + max_y)
            && set.contains(max_x + "." + min_y)
            && set.contains(max_x + "." + max_y))
            && (area == (max_y - min_y) * (max_x - min_x)));
    }
}
```

Problem 296

392 Is Subsequence

```
Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t.
 t is potentially a very long (length ~= 500,000) string,
  and s is a short string (<=100).
A subsequence of a string is a new string which is formed from the original string
 by deleting some (can be none) of the characters
  without disturbing the relative positions of the remaining characters.
   (ie, "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:
s = "abc", t = "ahbgdc"
Return true.
Example 2:
s = "axc", t = "ahbgdc"
Return false.

https://leetcode.com/problems/is-subsequence/
```

Solution

This problem seems a cheater. First of all, "Medium" seems not the proper difficulty of this problem, just check the code to realize what I mean.

Train of thought is quite simple, just maintain 2 pointers, when same, move the forward together, otherwise only pointer in target(longer) string moves forward.

Listing 391: Solution - Two Pointers

```
public class Solution {
    public boolean isSubsequence(String s, String t) {
        if (s.length() == 0){
            return true;
        }
        int sIdx = 0 , tIdx = 0;
        for (tIdx = 0 ; tIdx < t.length() ; tIdx ++){
            if (s.charAt(sIdx) == t.charAt(tIdx)){
                sIdx ++;
            }
            if (sIdx >= s.length()){
                return true;
            }
        }
        return false;

    }
}
```

However, this solution is not good enough(though already $O(n)$ complexity). The runtime is 62ms, because of charAt!

Then the solution may look like this,

Listing 392: Solution - Faster maybe Two Pointers

```
public class Solution
{
    public boolean isSubsequence(String s, String t)
    {
        if(t.length() < s.length()) return false;
        int prev = 0;
        for(int i = 0; i < s.length();i++)
        {
            char tempChar = s.charAt(i);
            prev = t.indexOf(tempChar,prev);
            if(prev == -1) return false;
            prev++;
        }

        return true;
    }
}
```

Now the runtime is 3ms using indexOf, though this is also in $O(n)$ time complexity.

Moreover, what makes this problem more like a cheater is that the tags of it includes 'Binary Search', 'Dynamic Programming'! Do we... really need this? So I had a go,

Listing 393: Solution - Dynamic Programming

```
public class Solution {
    public boolean isSubsequence(String s, String t) {
        if (s.length() == 0){
            return true;
        }
        if (t.length() == 0){
            return false;
        }
        int[][] dp = new int[s.length() + 1][t.length() + 1];
        for (int i = 1 ; i <= s.length() ; i++){
            for (int j = 1 ; j <= t.length() ; j++){
                if (s.charAt(i - 1) == t.charAt(j - 1)){
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                }
                else{
                    dp[i][j] = Math.max(dp[i - 1][j] , dp[i][j - 1]);
                }
            }
        }

        return (dp[s.length()][t.length()] == s.length());
    }
}
```

Finally... With $O(n^2)$ complexity... Memory Limit Exceed...

Problem 297

394 Decode String

```
The encoding rule is: k[encoded_string],
 where the encoded_string inside the square brackets
  is being repeated exactly k times.
   Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid;
 No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits
 and that digits are only for those repeat numbers, k.
  For example, there won't be input like 3a or 2[4].

Examples:
    s = "3[a]2[bc]", return "aaabcbc".
    s = "3[a2[c]]", return "accaccacc".
    s = "2[abc]3[cd]ef", return "abcabccdcdcdef".

https://leetcode.com/problems/decode-string/
```

Solution

Using stacks to solve the problem. Firstly I use 2 stacks, containing the duplicate number and repeating pattern. However, when considering test case "sd2[f2[e]g]i", the solution cannot pass that. Consequently I added another stack to preserve the level of pattern.

(Maybe my solution is a bit complicated...)

Listing 394: Solution

```
public class Solution {
    public String decodeString(String s) {
        Stack<String> numbers = new Stack<String>();
        Stack<String> patterns = new Stack<String>();
        Stack<Integer> levels = new Stack<Integer>();
        int curLevel = 0;
        String curNumber = "";
        String curPattern = "";
        char[] arr = s.toCharArray();
        for (int idx = 0 ; idx < arr.length ; idx ++){
            if (arr[idx] >= '0' && arr[idx] <= '9'){
                curNumber = curNumber + arr[idx];
                if (curPattern.length() > 0){
                    patterns.push(curPattern);
                    levels.push(curLevel);
                    curPattern = "";
                }
            }
            else if ((arr[idx] >= 'a' && arr[idx] <= 'z') ||
                     (arr[idx] >= 'A' && arr[idx] <= 'Z')){
                curPattern = curPattern + arr[idx];
                if (curNumber.length() > 0){
```

```
                    numbers.push(curNumber);
                    curNumber = "";
                }

            }
            else if (arr[idx] == '['){
                if (curNumber.length() > 0){
                    numbers.push(curNumber);
                }
                curNumber = "";
                curPattern = "";
                curLevel ++;
            }
            else {
                if (curPattern.length() > 0){
                    patterns.push(curPattern);
                    levels.push(curLevel);
                }
                String topPattern = patterns.pop();
                int peekLevel = levels.pop();
                int num = Integer.parseInt(numbers.pop());
                // System.out.println(num + " * " + topPattern);
                String pattern = "";

                while (patterns.size() > 0 && levels.peek() == peekLevel){
                    topPattern = patterns.pop() + topPattern;
                    levels.pop();
                }
                for (int i = 0 ; i < num ; i ++){
                    pattern = pattern + topPattern;
                }
                curLevel --;
                patterns.push(pattern);
                levels.push(curLevel);
                curNumber = "";
                curPattern = "";
            }
        }
        if (curPattern.length() > 0){
            patterns.push(curPattern);
            levels.push(curLevel);
        }
        String result = "";
        while (patterns.size() > 0){
            result = patterns.pop() + result;
        }
        return result;
    }
}
```

Problem 298

395 Longest Substring with At Least K Repeating Characters

```
Find the length of the longest substring T of a given string
 (consists of lowercase letters only)
  such that every character in T appears no less than k times.

Example 1:

    Input:
    s = "aaabb", k = 3

    Output:
    3

    The longest substring is "aaa", as 'a' is repeated 3 times.

Example 2:
    Input:
    s = "ababbc", k = 2

    Output:
    5

    The longest substring is "ababb", as 'a' is repeated 2 times
     and 'b' is repeated 3 times.

https://leetcode.com/problems/longest-substring-with-at-least-k-repeating-characters/
```

Solution

A brilliant divide-and-conquer problem using recursion. Count the occurrence of each character, and the substring must exists at sides of the character which appears less than k. And that is exactly the core train of thought.

Listing 395: Solution

```
public class Solution {
    public int longestSubstring(String s, int k) {
        // System.out.println(s);
        char[] sArr = s.toCharArray();
        if (s.length() < k){
            return 0;
        }
        int[] count = new int[26];
        Arrays.fill(count, 0);
        for (char ch : sArr){
            count[ch - 'a'] ++;
        }
        for (int chIdx = 0 ; chIdx < sArr.length ; chIdx ++){
            if (count[sArr[chIdx] - 'a'] < k){
                return Math.max(
                        longestSubstring(s.substring(0, chIdx), k),
                        longestSubstring(s.substring(chIdx + 1,
```

```
                            sArr.length) , k)
                    );
                }
            }
            return sArr.length;
        }
    }
```