1. Search a 2D Matrix

```
public boolean searchMatrix(int[][] matrix, int target) {
    if (matrix==null || matrix.length==0 || matrix[0].length==0) return false;
    int m = matrix.length;
    int n = matrix[0].length;
    int row = m-1;
    int col = 0;
    while (row>=0 && row<m && col>=0 && col<n) {
        int cur = matrix[row][col];
        if (cur == target) {
            return true;
        }
        else if (cur > target) {
            row--;
        }
        else col++;
    }
    return false;
}
```

```
    if(matrix==null || matrix.length==0 || matrix[0].length==0)
        return false;

    int m = matrix.length;
    int n = matrix[0].length;

    int start = 0;
    int end = m*n-1;

    while(start<=end){
        int mid=(start+end)/2;
        int midX=mid/n;
        int midY=mid%n;

        if(matrix[midX][midY]==target)
            return true;

        if(matrix[midX][midY]<target){
            start=mid+1;
        }else{
            end=mid-1;
        }
    }

    return false;
}
```

2. Count Primes

```
public int countPrimes(int n) {
    boolean notPrime[] = new boolean[n+2];
    notPrime[0] = notPrime[1] = true;
    for(int i=2; i*i<n ; i++) {
        if(!notPrime[i]) {
            int c = i * i;
            while(c < n) {
                notPrime[c] = true;
                c += i;
            }
        }

    int ans = 0;
    for(int i = 0; i < n; i ++) {
        if(!notPrime[i]) ans ++;
    }
    return ans;
}
```

http://www.cnblogs.com/grubbyskyer/p/3852421.html

基本思想：素数的倍数一定不是素数

3. Summary Ranges

```
public List<String> summaryRanges(int[] nums) {
    List<String> res = new ArrayList<String>();
    if(nums == null || nums.length < 1)
        return res;
    int s = 0, e = 0;
    while(e < nums.length) {
        if(e + 1 < nums.length && nums[e + 1] == nums[e] + 1)
            e++;
        else {
            if(s == e) {
                res.add(Integer.toString(nums[s]));
            } else {
                String str = nums[s] + "->" + nums[e];
```

```
                res.add(str);
            }
            e++;
            s = e;
        }
    }
    return res;

}
```

4. First Missing Positive

```
public int firstMissingPositive(int[] A) {
    int n = A.length;
    //排序
        for (int i = 0; i < n; i++) {
                while (A[i] != i + 1) {
                        if (A[i] <= 0 || A[i] >= n)
                                break;

                if(A[i]==A[A[i]-1])
                        break;

                        int temp = A[i];
                        A[i] = A[temp - 1];
                        A[temp - 1] = temp;
                }
        }

    //检验
        for (int i = 0; i < n; i++){
                if (A[i] != i + 1){
                        return i + 1;
                }
        }

        return n + 1;
}
```

http://www.programcreek.com/2014/05/leetcode-first-missing-positive-java/

5. Group Anagrams

```
public List<String> anagrams(String[] strs) {
      Map<String, ArrayList<String>> hash=new
HashMap<String, ArrayList<String>>();
       List<String> ret=new ArrayList<String>();
       if(strs.length<2)
       {
          return ret;
       }
       for(int i=0;i<strs.length;i++)
       {

          if(hash.containsKey(cur))
          {
             hash.get(cur).add(strs[i]);
          }else {
             ArrayList<String> al=new ArrayList<String>();
             al.add(strs[i]);
             hash.put(cur, al);
          }
       }

        for (ArrayList<String> value : hash.values()) {
          if (value.size() > 1) {
             ret.addAll(value);
          }
        }
       return ret;
```

```
    }
```

6. Product of Array Except Self

```
public int[] productExceptSelf(int[] nums) {
if(nums == null)
    return null;
int[] res = new int[nums.length];
for(int i = 0; i < nums.length; i++){
    if(i == 0)
        res[i] = 1;
    else
        res[i] = res[i - 1] * nums[i - 1];
}
int prod = 1;
for(int i = nums.length - 1; i >= 0; i--){
    res[i] = res[i] * prod;
    prod *= nums[i];
}
return res;
}
```

7. Largest Number

```
public String largestNumber(int[] nums) {
    String[] strs = new String[nums.length];
    for(int i=0; i<nums.length; i++){
```

```
            strs[i] = String.valueOf(nums[i]);
        }

        Arrays.sort(strs, new Comparator<String>(){
            public int compare(String s1, String s2){
                String s12 = s1 + s2;
                        String s21 = s2 + s1;
                    return (int) (Long.parseLong(s21) -
Long.parseLong(s12));
            }
        });

        StringBuilder sb = new StringBuilder();
        for(String s: strs){
            sb.append(s);
        }

        while(sb.charAt(0)=='0' && sb.length()>1){
            sb.deleteCharAt(0);
        }

/*

[0,0]
"00"
*/

        return sb.toString();
```

```
}
```

http://wlcoding.blogspot.com/2015/03/largest-number.html?view=sidebar

8. Candy

```
public int candy(int[] ratings) {
    if (ratings == null || ratings.length == 0) {
        return 0;
    }

    int[] candies = new int[ratings.length];
    candies[0] = 1;

    //from let to right
    for (int i = 1; i < ratings.length; i++) {
        if (ratings[i] > ratings[i - 1]) {
            candies[i] = candies[i - 1] + 1;
        } else {
            // if not ascending, assign 1
            candies[i] = 1;
        }
    }

    int result = candies[ratings.length - 1];

    //from right to left
    for (int i = ratings.length - 2; i >= 0; i--) {
        int cur = 1;
        if (ratings[i] > ratings[i + 1]) {
            cur = candies[i + 1] + 1;
        }

        result += Math.max(cur, candies[i]);
        candies[i] = cur;
    }

    return result;
}
```

9. LRU Cache

```
private class Node {
    int key, value;
    Node prev, next;
    public Node(int key, int value) {
        this.key = key;
        this.value = value;
    }
}

private int capacity;
private Map<Integer, Node> map; // store <key, node>
private Node head, tail;

public LRUCache(int capacity) {
    this.capacity = capacity;
    map = new HashMap<Integer, Node>();
    head = null;
    tail = null;
}

public int get(int key) {
    if (!map.containsKey(key))
        return -1;
    else {
        moveToEnd(map.get(key));
        return map.get(key).value;
    }
}

public void set(int key, int value) {
    if (!map.containsKey(key)) {    // if the node doesn't exist, add a new node to end
        // remove LRU (first node) if it reaches capacity
        if (map.size() == capacity) {
            map.remove(head.key);
            remove(head);
        }
        // add a new node at the end
        Node node = new Node(key, value);
        map.put(key, node);
        addToEnd(node);
    } else {                        // if the node exists, revise it and move to end
        map.get(key).value = value;
        moveToEnd(map.get(key));
    }
```

```
    }

    private void moveToEnd(Node node) {
        if (node == tail) {
            return;
        } else {    // node is either head or middle node
            remove(node);
            addToEnd(node);
        }
    }

    private void remove(Node node) {
        if (node == head) {
            head = head.next;
            if (head != null)
                head.prev = null;   // List has only one node
        } else if (node == tail) {
            tail = tail.prev;
            tail.next = null;
        } else {
            node.prev.next = node.next;
            node.next.prev = node.prev;
        }
    }

    private void addToEnd(Node node) {
        if (head == null) { // List is empty
            head = node;
            tail = node;
        } else {
            node.prev = tail;
            tail.next = node;
            tail = node;
        }
    }
```

LinkedHashMap 原理

http://yikun.github.io/2015/04/02/Java-LinkedHashMap%E5%B7%A5%E4%BD%9C%E5%8E%9F%E7%90%86%E5%8F%8A%E5%AE%9E%E7%8E%B0/

http://www.cnblogs.com/-OYK/archive/2012/12/04/2801799.html

10. Set Matrix Zeroes

```
public void setZeroes(int[][] matrix) {
    boolean firstRowZero = false;
    boolean firstColumnZero = false;

    //set first row and column zero or not
    for(int i=0; i<matrix.length; i++){
        if(matrix[i][0] == 0){
            firstColumnZero = true;
            break;
        }
    }

    for(int i=0; i<matrix[0].length; i++){
        if(matrix[0][i] == 0){
            firstRowZero = true;
            break;
        }
    }



    //mark zeros on first row and column
    for(int i=1; i<matrix.length; i++){
        for(int j=1; j<matrix[0].length; j++){
            if(matrix[i][j] == 0){
              matrix[i][0] = 0;
              matrix[0][j] = 0;
            }
        }
    }

    //use mark to set elements
    for(int i=1; i<matrix.length; i++){
        for(int j=1; j<matrix[0].length; j++){
            if(matrix[i][0] == 0 || matrix[0][j] == 0){
              matrix[i][j] = 0;
            }
        }
    }
```

```
        //set first column and row
        if(firstColumnZero){
            for(int i=0; i<matrix.length; i++)
                matrix[i][0] = 0;
        }

        if(firstRowZero){
            for(int i=0; i<matrix[0].length; i++)
                matrix[0][i] = 0;
        }

    }
```

11. Best Time to Buy and Sell Stock III

```
    public int maxProfit(int[] prices) {
        if(prices.length<=1){
            return 0;
        }
        int ret=0;
        int[] ps=new int[prices.length];
        int min=prices[0];
        for(int i=1;i<prices.length;i++){
            min=Math.min(min, prices[i]);
            ps[i]=Math.max(ps[i-1], prices[i]-min);
        }
        int []pe=new int[prices.length];
        int max=prices[prices.length-1];
        for(int i=prices.length-2;i>=0;i--){
            max=Math.max(max, prices[i]);
            pe[i]=Math.max(pe[i+1], max-prices[i]);
        }
```

```
        for(int i=0;i<prices.length;i++){
            ret=Math.max(ret, pe[i]+ps[i]);
        }
        return ret;
    }
```

12. Find Peak Element

```
    public int findPeakElement(int[] num) {
    int lo = 0, hi = num.length - 1;
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (mid < num.length - 1 &&  num[mid] < num[mid + 1])
            lo = mid + 1;
        else if (mid > 0 && num[mid] < num[mid - 1])
            hi = mid;
        else return mid;

    return -1;  // peak doesn't exist
    }
```

13. Word Break

```
    public boolean wordBreak(String s, Set<String> wordDict) {
        //Define an array t[] such that t[i]==true => 0-(i-1) can be segmented using
dictionary
       //Initial state t[0] == true
        boolean[] t = new boolean[s.length()+1];
        t[0] = true; //set first to be true, why?
        //Because we need initial state

        for(int i=0; i<s.length(); i++){
            //should continue from match position
            if(!t[i])
                continue;

            for(String a: wordDict){
                int len = a.length();
                int end = i + len;
                if(end > s.length())
                    continue;

                if(t[end]) continue;

                if(s.substring(i, end).equals(a)){
                    t[end] = true;
                }
            }
        }

        return t[s.length()];
    }
```

14. String to Integer (atoi)

```
    public int myAtoi(String str) {
          if (str == null || str.length() < 1)
```

```
            return 0;

        // trim white spaces
        str = str.trim();

        char flag = '+';

        // check negative or positive
        int i = 0;
        if (str.charAt(0) == '-') {
            flag = '-';
            i++;
        } else if (str.charAt(0) == '+') {
            i++;
        }
        // use double to store result
        double result = 0;

        // calculate value
        while (str.length() > i && str.charAt(i) >= '0' && str.charAt(i) <= '9') {
            result = result * 10 + (str.charAt(i) - '0');
            i++;
        }

        if (flag == '-')
            result = -result;

        // handle max and min
        if (result > Integer.MAX_VALUE)
            return Integer.MAX_VALUE;

        if (result < Integer.MIN_VALUE)
            return Integer.MIN_VALUE;

        return (int) result;
    }
```

```
public int myAtoi(String str) {
    str = str.trim();
    if(str.length() == 0) return 0;
    int index = 0;
    int isNeg = 1;
    if(str.charAt(0) == '-'){
        index++;
        isNeg = -1;
```

```
    }
    if(str.charAt(0) == '+'){
        index++;
    }

    long result = 0;
    for(; index < str.length(); index++){
        if(str.charAt(index) >= '0' && str.charAt(index) <= '9'){
            result = result * 10 + str.charAt(index) - '0';
            if(result > Integer.MAX_VALUE) break;
        }
        else break;
    }

    if(result * isNeg > Integer.MAX_VALUE) return Integer.MAX_VALUE;
    if(result * isNeg < Integer.MIN_VALUE) return Integer.MIN_VALUE;
    return (int)result * isNeg;
}
```

15. Search in Rotated Sorted Array

```
public int search(int[] nums, int target) {
    int left = 0;
    int right= nums.length-1;

    while(left<=right){
        int mid = left + (right-left)/2;
        if(target==nums[mid])
            return mid;

        if(nums[left]<=nums[mid]){
            if(nums[left]<=target&& target<nums[mid]){
                right=mid-1;
            }else{
                left=mid+1;
            }
        }else{
            if(nums[mid]<target&& target<=nums[right]){
                left=mid+1;
            }else{
                right=mid-1;
            }
        }
    }

    return -1;
```

```
    }
```

16. Maximum Product Subarray

```
    public int maxProduct(int[] nums) {
        if(nums==null || nums.length ==0) {
            return 0;
        }
        int maxLocal = nums[0];
        int minLocal = nums[0];
        int global = nums[0];
        for(int i=1; i<nums.length; i++){
            int temp = maxLocal;
            maxLocal = Math.max(Math.max(nums[i]*maxLocal, nums[i]),
nums[i]*minLocal);
            minLocal = Math.min(Math.min(nums[i]*temp, nums[i]), nums[i]*minLocal);
            global = Math.max(global, maxLocal);
        }
        return global;
    }
```

17. Divide Two Integers

```
    public int divide(int dividend, int divisor) {
        if(dividend == Integer.MIN_VALUE && divisor == -1) return
Integer.MAX_VALUE;
        long  a =Math.abs((long)dividend);
        long  b =Math.abs((long)divisor);

        int  ret = 0;
        while (a >= b) {
            long c = b;
            for (int i = 0; a >= c; ++i, c <<= 1) {
                a -= c;
                ret += 1 << i;
            }
        }
        if ((dividend < 0 && divisor > 0) || (dividend > 0 && divisor < 0)) {
                return -ret;
            } else{
```

```
                return ret;
            }
        }
```

18. Pow(x, n)

```
    public double myPow(double x, int n) {
        // Note: The Solution object is instantiated only once and is reused by each test
case.
        if(x == 0.0 && n == 0) {
            throw new NullPointerException();
        }
        int p=Math.abs(n);
        double ret=1;
        while(p>0)
        {
            if(p%2==1)
            {
                ret*=x;
            }
            p>>=1;
            x*=x;
        }
        if(n<0)
        {
            return 1/ret;
        }
        return ret;
    }
```

19. Happy Number

```
    public boolean isHappy(int n) {
        Set<Integer> s = new HashSet<Integer>();
        while(n != 1) {
            int sum = 0;
            while(n != 0) {
```

```
            int mod = n % 10;
            sum += mod * mod;
            n /= 10;
        }
        if(s.contains(sum))
            return false;
        n = sum;
        s.add(sum);
    }
    return true;
}
```

2^31-1 = 2147483647-> 810

20. Jump Game

```
public boolean canJump(int[] A) {
if(A.length <= 1)
    return true;

int max = A[0]; //max stands for the largest index that can be reached.

for(int i=0; i<A.length; i++){
    //if not enough to go to next
    if(max <= i && A[i] == 0)
        return false;

    //update max
    if(i + A[i] > max){
        max = i + A[i];
    }

    //max is enough to reach the end
    if(max >= A.length-1)
        return true;
}

return false;
}
```

http://www.programcreek.com/2014/03/leetcode-jump-game-java/

21. Lowest Common Ancestor of a Binary Tree

```
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode
q) {
        if(root==null) return null;
        if(root== p||root==q)
              return root;
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);
        //p and q are on the different branches, return root
        if(left!=null&&right!=null)
              return root;
        //p and q are on the same branch
        if(left!=null)
            return left;
        else
            return right;
    }
```

22. Lowest Common Ancestor of a Binary Search Tree

```
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode
q) {
                if (root == null || p == null || q == null)
                        return null;
                if (Math.max(p.val, q.val) < root.val)
                        return lowestCommonAncestor(root.left, p, q);
                if (Math.min(p.val, q.val) > root.val)
                        return lowestCommonAncestor(root.right, p, q);
                return root;
    }
```

23. Binary Tree Level Order Traversal

```
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> ret=new ArrayList<List<Integer>>();
        if(root == null) {
            return ret;
        }
        LinkedList<TreeNode> list = new LinkedList<TreeNode>();
        list.addLast(root);
        int num = list.size();
        while(num > 0) {
            int p = 0;
            List<Integer> curr = new ArrayList<Integer>();
            while(p < num) {
                TreeNode n = list.pollFirst();
                if(n.left != null) {
                    list.addLast(n.left);
                }
                if(n.right != null) {
                    list.addLast(n.right);
                }
                curr.add(n.val);
                p ++;
            }
            ret.add(curr);
            num = list.size();
        }
        return ret;
    }
```

24. Merge Intervals

```
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 == null || intervals.size() <= 1)
                        return intervals;

                // sort intervals by using self-defined Comparator
                Collections.sort(intervals, new IntervalComparator());

            List<Interval> result = new ArrayList<Interval>();

                Interval prev = intervals.get(0);
                for (int i = 1; i < intervals.size(); i++) {
                        Interval curr = intervals.get(i);

                        if (prev.end >= curr.start) {
                                // merged case
                                Interval merged = new Interval(prev.start,
Math.max(prev.end, curr.end));
                                prev = merged;
                        } else {
                                result.add(prev);
                                prev = curr;
                        }
                }

                result.add(prev);

                return result;
        }
}

    class IntervalComparator implements Comparator<Interval> {
        public int compare(Interval i1, Interval i2) {
                if (i1.start == i2.start)
                        return 0;
```

```
                return i1.start < i2.start ? -1 : 1;
        }
}
```

25. Merge k Sorted Lists

```
    public ListNode mergeKLists(ListNode[] lists) {
                if (lists.length == 0)
                        return null;

                //PriorityQueue is a sorted queue
                PriorityQueue<ListNode> q = new
PriorityQueue<ListNode>(lists.length,
                                new Comparator<ListNode>() {
                                        public int compare(ListNode a, ListNode b) {
                                                if (a.val > b.val)
                                                        return 1;
                                                else if(a.val == b.val)
                                                        return 0;
                                                else
                                                        return -1;
                                        }
                                });

                //add first node of each list to the queue
                for (ListNode list : lists) {
                        if (list != null)
                                q.add(list);
                }

                ListNode head = new ListNode(0);
                ListNode p = head; // serve as a pointer/cursor

                while (q.size() > 0) {
                        ListNode temp = q.poll();
                        //poll() retrieves and removes the head of the queue - q.
                        p.next = temp;

                        //keep adding next element of each list
```

```
                if (temp.next != null)
                        q.add(temp.next);

                p = p.next;
            }

            return head.next;
    }
```

26. sort colors

```
    public void sortColors(int[] A) {
        int []count = new int[3];
        for(int i = 0; i < A.length; i++) {
            count[A[i]] ++;
        }
        int p = 0;
        for(int i = 0; i < 3; i++) {
            for(int j = 0; j < count[i]; j++) {
                A[p++] = i;
            }
        }
    }
```

27. Unique Paths

```
    public int uniquePaths(int m, int n) {
        if(m==0 || n==0) return 0;
        if(m==1 || n==1) return 1;

        int[][] dp = new int[m][n];

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

        //top row
        for(int j=0; j<n; j++){
```

```
            dp[0][j] = 1;
        }

        //fill up the dp table
        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];
    }
```

28. Missing Number

```
    public int missingNumber(int[] A) {
        int sum = 0;
        int len = A.length;
        for(int i = 0; i < A.length; i++) {
            sum += A[i];
        }
        return len * (len + 1) / 2 - sum;
    }
```

29. Two Sum

```
    public int[] twoSum(int[] numbers, int target)
    {
        // Note: The Solution object is instantiated only once and is reused by each test
case.
    int[] result=new int[2];
    Map<Integer, Integer> map=new HashMap<Integer, Integer>();
    for(int i=0;i<numbers.length;i++)
    {
        int valNeed=target-numbers[i];
        if(map.containsKey(valNeed))
        {
```

```
            result[0]=map.get(valNeed);
            result[1]=i+1;
            break;
        }

        map.put(numbers[i],i+1);
    }

    return result;
    }
```

30. 3Sum

```
    public List<List<Integer>> threeSum(int[] num) {
    List<List<Integer>> result = new ArrayList<List<Integer>>();

        if (num.length < 3)
                return result;

        // sort array
        Arrays.sort(num);

        for (int i = 0; i < num.length - 2; i++) {
                // //avoid duplicate solutions
                if (i == 0 || num[i] > num[i - 1]) {

                        int negate = -num[i];

                        int start = i + 1;
                        int end = num.length - 1;

                        while (start < end) {
                                //case 1
                                if (num[start] + num[end] == negate) {
                                        List<Integer> temp = new ArrayList<Integer>();
                                        temp.add(num[i]);
                                        temp.add(num[start]);
                                        temp.add(num[end]);

                                        result.add(temp);
```

```
                    start++;
                    end--;
                    //avoid duplicate solutions
                    while (start < end && num[end] == num[end + 1])
                        end--;

                    while (start < end && num[start] == num[start - 1])
                        start++;
                //case 2
                } else if (num[start] + num[end] < negate) {
                    start++;
                //case 3
                } else {
                    end--;
                }
            }

        }
    }

    return result;
  }
```

31. wildcard matching

```
/*
*
*  m[i,j]=(s[i]=p[j] and m[i+1,j+1] if p[j] a normal character) m[i+1,j+1] if p[j]=="?"
   m[i,j]=m[i,j+1]        or        m[i+1,j] if p[j]="*"
*  match zero character        match 1 or multiple characters, if delete one,
there would be another recursion problem
*/
public class Solution {
    public boolean isMatch(String s, String p){
    if (s == null || p == null)  return false;

    // calculate count for non-wildcard char
    int count = 0;
    for (Character c : p.toCharArray()) {
        if (c != '*')  ++count;
```

```
}
// the count should not be larger than that of s
if (count > s.length())  return false;

    boolean [][] m=new boolean[s.length()+1][p.length()+1];

    m[s.length()][p.length()]=true;

    for(int j=p.length()-1;j>=0;j--)
    {
        if(p.charAt(j)=='*')
        {
            m[s.length()][j]=m[s.length()][j+1];
        }
        else
        {
            m[s.length()][j]=false;
        }
    }

    for(int i=s.length()-1;i>=0;i--)
    {
        m[i][p.length()]=false;
    }

    for(int i=s.length()-1;i>=0;i--)
    {
        //m[i][p.length()]=false;

        for(int j=p.length()-1;j>=0;j--)
        {
            if(p.charAt(j)=='?')
            {
                    m[i][j]=m[i+1][j+1];
            }
            else if((p.charAt(j)!='?')&&(p.charAt(j)!='*'))
            {
                if(s.charAt(i)==p.charAt(j))
                    m[i][j]=m[i+1][j+1];

            }
            else
            {
                m[i][j]=m[i][j+1]||m[i+1][j];
            }
```

```
            }
        }
        return m[0][0];
    }
}
```