1.

Decode Ways:

Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).

DFS: O(2n)

DP: O(n)

DP考虑步骤：1. 判断该位为’0’， 则res[i] = 0，否则res[i] = res[i+1]

2. 如果该位不是最后一位 且满足 两种decode条件（包括下一位为0）， 则 res[i] += res[i+2]

Restore IP:

DFS: O(fact(n))

Notice: 对于i从1到3的判断条件不能缺少len + 3 <= s.length()，否则会越界

2.

Longest Valid Parentheses:

1. 建立一个目录stack，只有当遍历到’(‘时才push
2. 当遍历到’)’，如果stack为空则index = i（index初值为-1），否则pop完比较max

Largest Rectangle in Histogram:

1. 建立一个目录stack，当stack为空或者height[stack.peek()]小于height[i]（最后一位小于0）则push
2. 否则pop并比较max，其中max = Math.max(max, height[t] \* (stack.isEmpty() ? I : I - stack.peek() - 1)

Maximal Rectangle:

1. 外层循环对行数进行遍历
2. 对于matrix中每行的元素若为’1’，则h[i]++, 否则h[i] = 0
3. 对遍历完的h[]进行如Largest Rectangle in Histogram的操作，即建立stack求解

3.

Unique Binary Search Trees:

1. Catalan Number : h(n) = 2\*(2n-1)\*h(n-1)/(n+1), h(0)=1, h(1)=1
2. Catalan Number: 因为 **h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)h(0) (n>=2)**

建立数组DP：for(int i=2; i<n+1; i++)

for(int j=0; j<i; j++)

n[i] += n[j] \* n[i-j-1];

Unique Binary Search Trees 2:

1. If(start>end) res.add(null); return res;
2. for(int i=start; i<end; i++)

for(TreeNode left : generateTrees(start, i-1))

for(TreeNode right : generateTrees(i+1, end))

4.

Populating Next Right Pointers in Each Node:

1. while(root!=null && root.left!=null && root.right!=null) {

next = root.left;

for(;root!=null;root=root.next){…}

root = next;

}

Populating Next Right Pointers in Each Node 2:

1. 需要多加一个pre指针记录前驱
2. while(root!=null) {

pre = null, next = null;

for(;root!=null;root = root.next) {

if next ==null

next = root.left !=null ? left : right;

if root.left != null

if pre!=null

pre.next = left;

pre = left;

if root.right != null

if pre!=null

pre.next = right;

right = pre;

}

root = next;

}

5.

Pascal’s Triangle:

for(int i=1; i<numRows; i++) {

add(1);

for(int k=1;k<result.get(i-1).size();k++)

a[i][k] = a[i-1][k-1] +a[i-1][k];

add(1);

}

Pascal’s Triangle 2:

for(int i=2;i<=rowIndex;i++)

result.add(1,i);

for(int j=2;j<result.size()-1;j++)

result.set(j, result.get(j)+result.get(j+1));

Triangle:

Given a triangle, find the minimum path sum from top to bottom.

1. 从i-1行找起，n[i-1][j] = n[i-1][j] + Math.min(n[i][j], n[i][j+1])

6.

Best Time to Buy and Sell Stock:

One transaction:

Pro = Math.max(pro, price[i]-min);

min = Math.min(min, price[i]);

Best Time to Buy and Sell Stock 2:

Multiple transaction:

min = 最低点; （先小于length-1 并 小于后面点，再index==0 或 小于等于前面点）

max = 最高点; （先大于0 并 大于前面点，再index==length-1 或 大于等于后面点）

when min != -1 && max != -1

pro += max – min;

Best Time to Buy and Sell Stock 3:

At most two transaction:

1. 建立一个his[]存放历史最高收益: his[i] = Math.max(prices[i]-min, his[i-1])
2. 建立一个fur[]存放未来最高收益: fur[j] = Math.max(max-prices[j], fur[j+1])

Pro = Math.max(his[j]+fur[j],pro)

7.

Construct Binary Tree from Preorder and Inorder Traversal:

1. 根据preorder[0]找到inorder[]中对应的i
2. if(i>=0)

root.left = build(preorder[1,i+1], inorder[0,i]);

if(i<length)

root.right = build(preorder[i+1,length], inorder[i+1,length]);

注意边界条件不要越界

Construct Binary Tree from Preorder and Inorder Traversal:

1. 同上

8.

Interleaving String:

1. DP方法，建立boolean check[s1.length+1][s2.length+1]
2. 对check[0][i], check[i][0]赋值true，若不等则break
3. i,j两个循环对check[i][j]进行赋值，返回check[s1.length][s2.length]

9.

Palindrome Partitioning:

1. DFS
2. if(s.length==0) result.add() return result;

for(int i=0;i<=s.length)

if(isPalindrome(s.substring(0,0))

for(ArrayList<String> t : DFS(s.substring(i)))

Palindrome Partitioning 2:

1. for(int i=0;i<=len;i++)

min[i] = len – i – 1;

for(int i=len-1; i>=0; i--)

for(int j=i; j<len; j++)

if(s[i]==s[j] &&( j<=i+2 || check[i+1][j-1]))

check[i][j] = true;

min[i] = min(min[i], min[j+1]+1);

Longest Palindrome Substring:

1. if(s[i]==s[j] &&( j<=i+2 || check[i+1][j-1]))

max = j-i+1 > max.length() ? s.substring(I,j+1) : max;

min[i] = min(min[i], min[j+1]+1);

10.

Reverse Integer:

1. 取abs(x), 设res用while循环进行reverse
2. 判断res<0的溢出情况，返回相同符号的reverse integer

Palindrome Number:

1. 同上（询问负数时的返回值）

11.

Substring concatenation of All Words:

S: “barfoothefoobarman” L: [“foo”, “bar”]

1. 建立hashmap存入L中所有string
2. 对S进行便利，每次新建一个hashmap，若包含S.substring则remove，且为空时添加index；若不包含则跳出循环

12.

Wildcard Matching:

‘?’, ‘\*’

1. 先判断p.charAt(j)是否为’\*’，若是：之后一直为’\*’，return true；之后不一直为’\*’，判断isMatch(s,p,i,j)
2. 判断j>=p.length()
3. 判断当前字母相等
4. 若都不是，return false

Regular Expression Matching

‘.’, ‘\*’

1. 判断j==p.length(), 是否i==s.length()
2. 判断如果j==p.length()-1 || p.charAt(j)!=’\*’，若i==s.length(), return false, 否则匹配当前
3. （p.charAt(j+1)==’\*’）判断如果匹配，持续匹配helper(s,p,i++,j+2)
4. （p.charAt(j+1)==’\*’）若都不匹配，return helper(s,p,I,j+2)

13.

Swap Nodes inPairs:

1. Iterative Way: 利用三个指针ppre, pre, cur, ppre用来记录上回swap后的后一个node， pre记录当前第一个node，cur记录当前第二个node，循环条件是cur != null
2. Recursive Way: 新建一个second = head.next的指针，每次对当前两个指针进行swap， 每组的连接方式为head.next.next = swapPairs(head.next.next)

Reverse Nodes in k-groups:

1. 建立一个dummy结点，进入head != null的循环条件，当i % k == 0时有：

pre = reverse(pre, head.next); head = pre.next; 否则head = head.next;

1. 在reverse(pre,next)函数中，利用last = pre.next, cur = last.next, 以cur != next为条件进行reverse，返回last

14.

First Missing Positive:

1. 当swap时，注意A[i]值的改变，应为：

int temp = A[i];

A[i] = A[A[i]-1];

A[temp-1] = temp;

或者改变swap顺序

15.

Jump Game:

1. 用cur = Math.max(cur, A[i]+i)记录能够到达的最大index，若i>cur则跳不出数组

Jump Game 2:

1. 设定cur，max，step变量
2. 若i>max，则把max = cur，对当前index可以到达的最远位置更新（每次更新的依据是step++）
3. 返回step（可在循环中加入Jump Game的判断条件，否则向面试官确定是否可以跳出循环）

Maximum Subarray:

1. sum = 0, max = A[0]
2. sum = Math.max(sum+A[i], A[i])

max = Math.max(max, sum)

16.

Search in Rotated Sorted Array (no duplicate): O(logn)

1. 循环条件start<=end，对mid进行判断
2. 若A[mid] > A[start]，两种情况讨论；else，两种情况讨论

Search in Rotated Sorted Array (duplicate): O(n)

1. 同上
2. 增加判断条件else即A[mid]==A[start], 则start++

17.

Merge Intervals:

1. 对原arraylist进行sort
2. 若intervals.get(i).start <= intervals.get(i-1).end, 则

intervals.get(i-1).end = Math.max(intervals.get(i).end, intervals.get(i-1).end)且

intervals.remove(i); --i;

Insert Intervals:

1. 第一个while(i<len && newIn.start > intervals.get(i).end); (插入newIn之前interval)
2. 第二个while(i<len && newIn.end >= intervals.get(i).end); (插入newIn)
3. 第三个while(i<len); (插入newIn之后interval)

18.

Permutation Sequence:

Given n and k, return the kth permutation sequence

1. Use a ArrayList to store 1-n, and a fact[] to store factorial of 1-n, use for loop to initialize
2. String s = “”, k--;
3. for(int i=n-1;i>=0;i--) {

int r = k / fact[i];

res += store.get(r);

store.remove(r);

k %= fact[i];

}

Permutations:

Permutations2:

Combinations:

Subsets:

Subsets2:

gray code:

19.

Sqrt(int x):

1. start = 0, end = x
2. while(end – start > 1)
3. 判断条件为mid == x/mid, mid < x/mid, mid > x/mid，若用mid\*mid会溢出

Pow(double x, int n):

1. if(n==0) retun 1
2. x%2==0: temp\*temp

else x>0: temp\*temp\*x

x<0: temp\*temp\*(1/x)

divide(int dividend, int divisor)

1. long d1 = Math.abs((long)dividend), long d2 = Math.abs((long)divisor)
2. while(d1>=d2))

for(int i=0;d1>=temp;temp<<1, i++) {

d1 -= temp;

res += I;

}