#	Title	Acceptance	Difficulty
206	Reverse Linked List		Easy
151	Reverse Words in a String		Medium
1	Two Sum		Easy
138	Copy List with Random Pointer		Hard
8	String to Integer (atoi)	需要练习	Easy
235	Lowest Common Ancestor of a Binary Search Tree	需要练习	Easy
75	Sort Colors		Medium
2	Add Two Numbers		Medium
141	Linked List Cycle		Easy
171	Excel Sheet Column Number		Easy
88	Merge Sorted Array		Easy
98	Validate Binary Search Tree		Medium
4	Median of Two Sorted Arrays	需要练习	Hard
200	Number of Islands		Medium
21	Merge Two Sorted Lists		Easy
146	LRU Cache		Hard
54	Spiral Matrix		Medium
53	Maximum Subarray		Medium
237	Delete Node in a Linked List		Easy

273	Integer to English Words	Hard
20	Valid Parentheses	Easy
48	Rotate Image	Medium
116	Populating Next Right Pointers in Each Node	Medium
236	Lowest Common Ancestor of a Binary Tree	Medium
168	Excel Sheet Column Title	Easy
117	Populating Next Right Pointers in Each Node II	Hard
24	Swap Nodes in Pairs	Easy
26	Remove Duplicates from Sorted Array	Easy
191	Number of 1 Bits	Easy
121	Best Time to Buy and Sell Stock	Easy
15	3Sum	Medium
13	Roman to Integer	Easy
268	Missing Number	Medium
160	Intersection of Two Linked Lists	Easy
33	Search in Rotated Sorted Array	Hard
218	The Skyline Problem	Hard
5	Longest Palindromic Substring	Medium
215	Kth Largest Element in an Array	Medium
238	Product of Array Except Self	Medium

73	Set Matrix Zeroes	Medium
102	Binary Tree Level Order Traversal	Easy
204	Count Primes	Easy
189	Rotate Array	Easy
153	Find Minimum in Rotated Sorted Array	Medium
208	Implement Trie (Prefix Tree)	Medium
174	Dungeon Game	Hard
297	Serialize and Deserialize Binary Tree	Hard
103	Binary Tree Zigzag Level Order Traversal	Medium
212	Word Search II	Hard

23	Merge k Sorted Lists	Hard
79	Word Search	Medium
232	Implement Queue using Stacks	Easy
101	Symmetric Tree	Easy
173	Binary Search Tree Iterator	Medium
106	Construct Binary Tree from Inorder and Postorder Traversal	Medium
186	Reverse Words in a String II	Medium
125	Valid Palindrome	Easy
165	Compare Version Numbers	Easy

25	Reverse Nodes in k-Group	Hard
71	Simplify Path	Medium
162	Find Peak Element	Medium
258	Add Digits	Easy
94	Binary Tree Inorder Traversal	Medium
112	Path Sum	Easy
46	Permutations	Medium
91	Decode Ways	
28	Implement strStr()	Easy
124	Binary Tree Maximum Path Sum	Hard

56	Merge Intervals	Hard
348	Design Tic-Tac-Toe	Medium
47	Permutations II	Medium
285	Inorder Successor in BST	Medium
365	Water and Jug Problem	Medium
270	Closest Binary Search Tree Value	Easy
333	Largest BST Subtree	Medium
213	House Robber II	Medium
114	Flatten Binary Tree to Linked List	Medium

55	Jump Game	Medium
300	Longest Increasing Subsequence	Medium

基础数据结构	C#
	HEAP
	MAP
	ARRAYLIST
	QUEUE
	STACK and QUEUE each implement
	LinkedList
	Tree
	RedBlackTree
	AVL Tree
	Graph
	TREESET HASHSET LINKEDHASHSET

Summary
pre, swap pre and head.next;
split

BST inorder is sorted; all greater than root must be right, all less than root must be left; otherwise is root;
counting sort : two pass
one pass: swap red with current when current is 0, red move next, current move next; swap blue with current when current is 2, blue move previous, otherwise current move to next;

1. INT2STRING-> n--
2. STRING2INT-> Math.pow

1. Recursive using long.min and long.max as left and right bound
2. Stack, using pre to track last node, and condition is pre!=nll && pre.val <= curt.val
findkth: odd and even
DFS

MAP; LinkedList for store last access;
remove and sethead
from left to right, from right to bottom, for right to left, from bottom to top;
iterate until result size get m*n
using top, left, right, bottom to tracking last postion which has visited
minsum: max of previous plus iTh elem or iTh elem
max: tracking maximum value

1.loop in num >0; num % 1000 get every part; num /= 1000 to computing next part
2.using string array to mapping every part
3.using "" to fill align position
dfs: left--,right add "(", left, right-- add ")"
matrix[i][j] = matrix[length - j - 1][i],clockwise to swap adjacent elem
row: [0->length /2); column [0->(length + 1) / 2)

curr for set next of two children, loop by curr = curr.next;

1. recursion: if no parent nodes
2. add path into list, compare two paths
1. INT2STRING-> n--
2. STRING2INT-> Math.pow
ADDING PRE TO TRACKING FIRST NOT OF EVERY LEVEL
Level iterate by node = dumm.next, means next level from first node of current level;
dummy

slow and fast, ++slow = fast

n&(n-1) != 0 when n!=0
Find min, return max=Pi - min

map, from right to left, compare i < i +1 for plus, i > i+1 subtract
bit: x^[0-n] then x^nums[0-n)
link 1 and 2, then find cycle in new list.
Compare with start, then compare in which area, most left and most right
original split to edge,sort, priorityqueue
DP odd or even, i. i-j was p and longest
Simple: from i as center, do i - count and i + count for find maximum count
quickselect: 1. n - k +1 2.pivot, **not nums[mid]**
from left to right, then from right to left; p=1 as initial value
Recursive: nums, forwardproduct, index, len

1. two variables mark first row and column exist 0
2. from对角线 to update first row and column
3. from first row and colum to update对角线
4. according to two variables to update first row and column

queue is not empty, iterate elem of queue to add list and offer children of per node

<=2->0; boolean array to tracking which is prime, iterate i:[0->sqrt(n-1)], if current i is prime then all of multiple set non-prime; last check boolean array to count

k %= nums.length, three steps to reverse

target = nums[end]

Trie structure: using 26 size array to store 26 alphabet. Not store char, depends specify position is null or not null

search: recursive forward index to generate next level children, if index == length then completeword

find: recursive to find which level was not null, and only last level node has completeword

from Princess position to find up(down) and left(right) which minimum HP

Compare with 1, because Knight only live when he has more than 0

1. serialize: flatten treenode to arraylist, once curt node== null then continue; otherwise add left and right; trailing null of end, finally iterate all element of list
2. deserialize: split to val array, ArrayList to tracking all of node, index mark level, isLeft mark left or right, iterate val array: non "null" then according isLeft to add left or right, if right then index++ to next parent node, swap isleft to change

Queue implement is more readable when serialize, but deserialize implemented by for loop and list is readable

1. queue, leftDirection: Collections.reverse
2. double stack, iterate current push to next, when current level stack is empty, current = next;

Trie: with trie tree study together

1. Divide Conquer : two part helper -> start, mid ; mid + 1, end; merge two lists. Also can using merge two each

2. Heap, PriorityQueue offer head of list, then put minimum head.next into pq

DP find equals startwith first char of word in board, find function is recursive,
1. EXIT: index = word.length
2. check :x,y must in board, char index of word equal board x-y char
3. using special char to fill board x-y, thus doesnot visited again.
4. recursive means :up, down, left, right four directions to find next index
5. backtracking original char of board x-y

double stack, when tail stack is empty, then move all of head stack to tail stack
helper: left and right all null - > true; either left or right null - > false; left.val != right.val - > false; check left.left with right.right and left.right with right.left;
inorder traversal: come to most left and push stack, return pop then point to right
IO&PO: root is last of postorder, recursive to build left tree (,instart, pos -1,,postart, postart + pos - 1 - instart), recursive to build right tree(, pos + 1, inend,,postart + pos - instart, poend -1);

PRE&IO: root is first of preorder, recursive to build left tree(,prestart + 1, prestart + pos -1 - instart,,instart, pos -1);recursive to build right tree (, prestart + pos + 1 - instart, preend,,pos + 1, inend);

Important is caculate left length, then caculate right lenth
reverse all string, then reverse every word
i = j = 0; j++; j == length || j = ' ' then reverse(i, j - 1; move i to new word i =j + 1
tow pointer, Character.isLetterOrDigit; Character.toLoweCase();
split("\\.")

1. reverse: reverse interval of two node(pre, next) - > exclusive; last and curr, loop in curr != next, swap curr and last(last.next=curr.next; curr.next = pre.next; pre.next = curr; curr = last.next;)
2. dummy, pre = dummy; i % k == 0, pre = reverse(pre, head.next)
split("/+") for split paths by arbitrary count /
store per path into arraylist
1. ".." remove last element of arraylist, means up to parent directory
2. "" or "." ignore it

merge arraylist by "/" for formulate new paths
determine increase area (mid ^ mid + 1) or decrease area(mid ^ mid - 1)^2
1. loop for sum
2. (num - 1) % 9 + 1
NonRecursion: curt down to last left, then pop, add, but curt = curt.right
recursive: root==null->false, root.left==null and root.right==null
detemine root.val==sum; call recursive left or right child when sum = sum - root.val
dfs, pos=[0-n), ! list.contains(nums[i]), backtracking last element
DP, i means sum of first i-1 and i-2(if i-2 i-1 (two digit) between 10-26)
corner case: '0' start
1. haystack loop [0->hlen - nlen + 1), needle loop [0, nlen)
2. if any of haystack[i+j] != needle[j] break needle loop
3. if j == needle, means all of char in needle has all found, return haystack loop index i
1. using array to pass an object value fro recursive
2. caculate left and right subtree
3. find max from root.val , root.val + left and root.val + right, means if left or ritgh was negative, can discard it; Then put maximum value into current which is current recursion level
4. find maximum from current, left+ root.val + right and maxarray, put to maxarray.

1. SelfComparator implements Comparator<T> {@Override compare}
2. sort Intervals
3. current or last interval comare with next interval. A: next.start<= current.end then update current.end(maximum of curr.end and next.end) ; B next.start > current.end then add result and current = next;
4. last, don't forget add last (current) to result

Subscribe?

visited array
continue: has visited or (current == previous and previous has not visited)
backtracking: remove last, and restore visited to unvisit

1. BST feature: if p.val < root.val then search left (root = root.left) otherwise sear right(root = root.right)
2. Recursion: exit: root == null return; if root.val <=p.val then recusive right, (otherwise left = recusive left, if left != null, return left, otherwise return root)

return x + y ==z || (x + y > z) && (z % gcd(x, y) == 0)
formula: gcd(int x, int y) {return y == 0 ? x : gcd(y, x % y);}

BST feature, closest to track minimum difference of per node with target; using target to detemine which subtree shoule to go. Left < root < right

NonRecursion1: preorder to list, set every node.left == null, node.right = list.get(i), node = node.right to level by level to construct
NonRecursion2: stack, curr=root,loop curr not null or stack not empty: if right not null push right; if left not null then curr.right = curr.left, then curr.left = null, otherwise if stack not empty, curr.right = stack.pop(); curr = curr.right to continue
Recursion: using global node to track last node, if lastnode not null, set lastnode.left =null, lastnode.right = root; set lastnode = root, keep root.right into temp node, then flatten root.left, then flatten tempnode(previous root.right)

DP can[i] means j=0->i) exists any j can jump to i, can[j] && j + nums[j] >= i,but DP = > TLE;

Greedy: farest to track greatest i + A[i] when i < farest; detemine farest >= nums.length -1; means can jump to last element

DP : max[i] represent the length of the longest increasing subsequence so far. If any element before i is smaller than nums[i], then max[i] = max(max[i], max[j]+1).

BS: using ArrayList to track increasing subsequence, when comes nums[i] > last elem of list, directly put into list, otherwise binary search where is insert

http://www.programcreek.com/2013/03/hashset-vs-treeset-vs-linkedhashset/

link

NO	#	Title	Difficulty
1	338	Counting Bits	Medium
2	366	Find Leaves of Binary Tree	Medium
3	280	Wiggle Sort	Medium
4	136	Single Number	Medium
44	137	Single Number II	Medium
12	260	Single Number III	Medium
5	370	Range Addition	Medium
6	369	Plus One Linked List	Medium
7	362	Design Hit Counter	Medium
8	167	Two Sum II - Input array is sorted	Medium
9	311	Sparse Matrix Multiplication	Medium
10	364	Nested List Weight Sum II	Medium
11	245	Shortest Word Distance III	Medium
13	382	Linked List Random Node	Medium
14	281	Zigzag Iterator	Medium
15	238	Product of Array Except Self	Medium

16	384	Shuffle an Array	Medium
17	256	Paint House	Medium
18	323	Number of Connected Components in an Undirected Graph	Medium
19	348	Design Tic-Tac-Toe	Medium
20	347	Top K Frequent Elements	Medium
21	122	Best Time to Buy and Sell Stock II	Medium
22	357	Count Numbers with Unique Digits	Medium
23	343	Integer Break	Medium
24	294	Flip Game II	Medium
25	268	Missing Number	Medium
26	320	Generalized Abbreviation	Medium
27	94	Binary Tree Inorder Traversal	Medium
28	144	Binary Tree Preorder Traversal	Medium
29	319	Bulb Switcher	Medium
30	360	Sort Transformed Array	Medium
31	378	Kth Smallest Element in a Sorted Matrix	Medium
32	318	Maximum Product of Word Lengths	Medium
33	325	Maximum Size Subarray Sum Equals k	Medium
34	12	Integer to Roman	Medium
35	328	Odd Even Linked List	Medium
36	156	Binary Tree Upside Down	Medium

37	230	Kth Smallest Element in a BST	Medium
38	259	3Sum Smaller	Medium
96	39	Combination Sum	Medium
128	40	Combination Sum II	Medium
43	216	Combination Sum III	Medium
39	377	Combination Sum IV	Medium
100	213	House Robber II	Medium
40	337	House Robber III	Medium
41	286	Walls and Gates	Medium
42	22	Generate Parentheses	Medium
45	108	Convert Sorted Array to Binary Search Tree	Medium

46	96	Unique Binary Search Trees	Medium
121	95	Unique Binary Search Trees II	Medium
50	241	Different Ways to Add Parentheses	Medium
47	351	Android Unlock Patterns	Medium

48	309	Best Time to Buy and Sell Stock with Cooldown	Medium
49	250	Count Univalue Subtrees	Medium
51	35	Search Insert Position	Medium
52	298	Binary Tree Longest Consecutive Sequence	Medium
53	89	Gray Code	Medium
54	46	Permutations	Medium
143	31	Next Permutation	Medium
127	47	Permutations II	Medium
156	60	Permutation Sequence	Medium
68	77	Combinations	Medium
55	255	Verify Preorder Sequence in Binary Search Tree	Medium
56	62	Unique Paths	Medium
57	53	Maximum Subarray	Medium
58	153	Find Minimum in Rotated Sorted Array	Medium
59	254	Factor Combinations	Medium
60	116	Populating Next Right Pointers in Each Node	Medium
61	199	Binary Tree Right Side View	Medium
62	367	Valid Perfect Square	Medium
63	173	Binary Search Tree Iterator	Medium

176	54	Spiral Matrix	Medium
64	59	Spiral Matrix II	Medium
65	253	Meeting Rooms II	Medium
66	285	Inorder Successor in BST	Medium
83	74	Search a 2D Matrix	Medium
67	240	Search a 2D Matrix II	Medium
69	64	Minimum Path Sum	Medium
70	334	Increasing Triplet Subsequence	Medium
71	313	Super Ugly Number	Medium
116	264	Ugly Number II	Medium
72	247	Strobogrammatic Number II	Medium
73	48	Rotate Image	Medium
74	300	Longest Increasing Subsequence	Medium
75	361	Bomb Enemy	Medium
76	251	Flatten 2D Vector	Medium
77	75	Sort Colors	Medium
78	289	Game of Life	Medium
79	11	Container With Most Water	Medium
80	277	Find the Celebrity	Medium
81	244	Shortest Word Distance II	Medium
82	215	Kth Largest Element in an Array	Medium
84	284	Peeking Iterator	Medium
85	73	Set Matrix Zeroes	Medium
86	376	Wiggle Subsequence	Medium
87	261	Graph Valid Tree	Medium
88	162	Find Peak Element	Medium
89	279	Perfect Squares	Medium
90	341	Flatten Nested List Iterator	Medium
91	129	Sum Root to Leaf Numbers	Medium
92	80	Remove Duplicates from Sorted Array II	Medium
93	78	Subsets	Medium
94	331	Verify Preorder Serialization of a Binary Tree	Medium
95	275	H-Index II	Medium
97	314	Binary Tree Vertical Order Traversal	Medium
98	81	Search in Rotated Sorted Array II	Medium
99	114	Flatten Binary Tree to Linked List	Medium
101	201	Bitwise AND of Numbers Range	Medium
102	90	Subsets II	Medium
103	109	Convert Sorted List to Binary Search Tree	Medium
104	368	Largest Divisible Subset	Medium

105	142	Linked List Cycle II	Medium
106	120	Triangle	Medium
107	372	Super Pow	Medium
108	375	Guess Number Higher or Lower II	Medium
109	274	H-Index	Medium
110	147	Insertion Sort List	Medium
111	86	Partition List	Medium
112	163	Missing Ranges	Medium
113	17	Letter Combinations of a Phone Number	Medium
114	103	Binary Tree Zigzag Level Order Traversal	Medium
115	63	Unique Paths II	Medium
117	34	Search for a Range	Medium
118	106	Construct Binary Tree from Inorder and Postorder Traversal	Medium
119	16	3Sum Closest	Medium
120	200	Number of Islands	Medium
122	105	Construct Binary Tree from Preorder and Inorder Traversal	Medium
123	267	Palindrome Permutation II	Medium
124	113	Path Sum II	Medium
125	161	One Edit Distance	Medium
126	49	Group Anagrams	Medium
129	236	Lowest Common Ancestor of a Binary Tree	Medium
130	186	Reverse Words in a String II	Medium
131	55	Jump Game	Medium
132	92	Reverse Linked List II	Medium
133	131	Palindrome Partitioning	Medium
134	207	Course Schedule	Medium
135	356	Line Reflection	Medium
136	333	Largest BST Subtree	Medium
137	134	Gas Station	Medium
138	82	Remove Duplicates from Sorted List II	Medium
139	209	Minimum Size Subarray Sum	Medium
140	310	Minimum Height Trees	Medium
141	50	Pow(x, n)	Medium
142	187	Repeated DNA Sequences	Medium
144	271	Encode and Decode Strings	Medium
145	373	Find K Pairs with Smallest Sums	Medium
146	229	Majority Element II	Medium
147	306	Additive Number	Medium
148	385	Mini Parser	Medium
149	227	Basic Calculator II	Medium
150	139	Word Break	Medium
151	386	Lexicographical Numbers	Medium
152	222	Count Complete Tree Nodes	Medium
153	332	Reconstruct Itinerary	Medium
154	69	Sqrt(x)	Medium
155	148	Sort List	Medium
157	228	Summary Ranges	Medium

158	379	Design Phone Directory	Medium
159	322	Coin Change	Medium
160	208	Implement Trie (Prefix Tree)	Medium
161	221	Maximal Square	Medium
162	133	Clone Graph	Medium
163	388	Longest Absolute File Path	Medium
164	150	Evaluate Reverse Polish Notation	Medium
165	18	4Sum	Medium
166	93	Restore IP Addresses	Medium
167	43	Multiply Strings	Medium
168	2	Add Two Numbers	Medium
169	79	Word Search	Medium
170	353	Design Snake Game	Medium
171	324	Wiggle Sort II	Medium
172	5	Longest Palindromic Substring	Medium
173	143	Reorder List	Medium
174	61	Rotate List	Medium
175	365	Water and Jug Problem	Medium
177	152	Maximum Product Subarray	Medium
178	355	Design Twitter	Medium
179	3	Longest Substring Without Repeating Characters	Medium
180	71	Simplify Path	Medium
181	210	Course Schedule II	Medium
182	304	Range Sum Query 2D - Immutable	Medium
183	98	Validate Binary Search Tree	Medium
184	179	Largest Number	Medium
185	211	Add and Search Word - Data structure design	Medium
186	15	3Sum	Medium
187	127	Word Ladder	Medium
188	220	Contains Duplicate III	Medium
189	307	Range Sum Query - Mutable	Medium
190	91	Decode Ways	Medium
191	130	Surrounded Regions	Medium
192	166	Fraction to Recurring Decimal	Medium
193	29	Divide Two Integers	Medium
194	151	Reverse Words in a String	Medium

1. 1-2-4-8-16-32 : count =1
2. 3= 2+1 means 1(2) + 1(1) =2; 7 = 4 + 3 means 1(4) + 2(3) = 3
3. loop i :[1~num] if i == pow then result[i] = 1; pow <<=1; p=1 means step into next pow; otherwise result[i] = result[p] + 1; p++;

x ^= nums[i]; return x

如果我们把 第 ith 个位置上所有数字的和对3取余, 因此取余的结果就是那个 “Single Number”.
对每一位的和做%3运算，来消去所有重复3次的数

```
if (((nums[j] >> i) & 1) == 1) {count[i]++;}
result |= ((count[i] % 3) << i);
```

```
easy readable
for (int i = 31; i >= 0; i--) {
        int sum = 0;
        int mask = 1 << i;
        for (int j = 0; j < nums.length; j++) {
          if ((nums[j] & mask) != 0) {
            sum++;
          }
        }
        result = (result << 1) + (sum % 3);
      }
```

Throught x^nums[i] to get x1Xorx2, using last1bit to split original array
1. int last1bits = x - (x & (x - 1));
2. if ((last1bits & n) == 0)

two pointer : meet type: left-> <-right

random.nextInt(cnt) == 0 ????
Get a random length of list to get value?

N non-repeat random number

```
int num = random.Next(0, end + 1);
output[i] = sequence[num];
sequence[num] = sequence[end];
end--;
```

1. two pass, p=1 as initial value: first from left to right (1,a1,a1a2,a1a2a3),second from right to left (a2a3a4, a3a4,a4,1);
2. Recursive: (nums, forwardproduct, index, len) BUT can stackoverflow

Reservoir sampling 1. Random, keep original array, declare output array, Arrays.copyOf(nums, nums.length) 2. loop i in output:[0~n)int tempPos = random.nextInt(i + 1); swap i and tempPos value
Map get num with count, PriorityQueue to get top K 1. Pair class{num, count} 2. PriorityQueue<>(new Comparator<Pair>(){…}); 3. Map.Entry<Integert, Integer> entry : map.entrySet() ; queue.offer(new Pair(entry.getKey(), entry.getValue())); if queue.size>k then queue.poll() 4. Add all pair.num into list 5. reverse list
if i+1 > i then profit +=prices[i+1]- prices[i]
Sigma f(k) = 9 * 9 * 8 * ... (9 - k + 2); 0 also count in
from 7-10 to find regular rule, hence res = 1, while (n > 4) { res *= 3; n -= 3; } return res
Exclusive or: x=0, x^=i (i:[0~len]), then x^=nums[i] (i:[0~len-1], finanlly x is missing num
stack and curt, find leftest node and push node into stack in per iterate (curt = curt.left),then curt = pop(), list.add(curt.val), lastly curt = curt.right;
stack.push(root), while(!stack.isEmpty()) node = pop(), list.add(node.val), push right first, then push left when right or left is not null
对于第n个灯泡，只有当次数是n的因子的之后，才能改变灯泡的状态，即n能被当前次数整除，比如当n为36时，它的因数有(1,36), (2,18), (3,12), (4,9), (6,6), 可以看到前四个括号里成对出现的因数各不相同，括号中前面的数改变了灯泡状态，后面的数又变回去了，等于锁的状态没有发生变化，只有最后那个(6,6)，在次数6的时候改变了一次状态，没有对应其它的状态能将其变回去了，所以锁就一直是打开状态的。所以所有平方数都有这么一个相等的因数对，即所有平方数的灯泡都将会是打开的状态。
Heap and visited array 1. class number : x, y, val; define comparator 2. offer matrix[0][0] into heap, visited[0][0]=true; 3. loop i [0~k-1), because in loop will offer a new Number, thus the kTh number will be in the peek of heap 4. Down and right direction to find next element can be put in heap;dx= {0, 1) dy={1,0}
BITMASK for store char of word appeared, Integer has 32(4*8) bit, enough for store 26 letters. 1. preprocess: mask[n], mask[i] |= 1 << (words[i].charAt(j) - 'a'); to mark per word has which char 2. two loop: i:[0~n) j:[i+1~n) if ((mask[i] & mask[j]) == 0) means no same char in two words 3. max record maximum words[i].length() * words[j].length()
int array store all integer of roman mapping; string array store all string of roman mapping 1. x=num/integer[i]; while(x>0) {result+=roman[i]; x--} num=num%integer[i]
Two head to connect odd and even, then connect odd and even

1. Naïve : Inorder BST
2. if can modified tree datastructure
if k == node.leftNum + 1, return node
if k > node.leftNum + 1, make k -= node.leftNum + 1, and then node = node.right
otherwise, node = node.left

SORT first
DFS: dfs(List<List<Integer>> result, List<Integer> list, int index, int target, int[] candidates) {
1. target ==0 result.add(list) -> if list not exist in result at first
2. target < candidates[i] return
3. backtracking

SORT first
DFS: dfs(List<List<Integer>> result, List<Integer> list, int index, int target, int[] candidates) {
1. target ==0 result.add(list) -> if list not exist in result at first
2. target < candidates[i] return
2.1 add prev to tracking last access element, if (cadidates[i]!=prev) then dfs next level, meantime prev=candidates[i]
3. backtracking

SORT first
DFS: dfs(List<List<Integer>> result, List<Integer> list, int start, int k, int sum) {
0. sum <0 return;
1. target ==0&&list.size()==k result.add(list) -> if list not exist in result at first
2.loop i [start~9] dfs(result, list, i + 1, k, sum - i);
3. backtracking

DP : Coin change
dp = new int[target+1]
if (i - nums[j] > 0) {dp[i] += dp[i - nums[j]];} else if (i - nums[j] == 0) {dp[i] += 1;}
dp[i]表示组成和为i的方法由多少种，那么dp[i] = sum(dp[i-nums[j]])，其中j为遍历nums的下标索引。
比如nums=[1,2,3], target=4。那么dp[4] = dp[4-1] + dp[4-2] + dp[4-3]。
在初始化时，如果i为nums中的元素，那么dp[i]为1。

Don't count last to find maximum a, then don't count first to find maximum b, finally maximum of a and b
helper to computing two times
need initial dp[start] and dp[start + 1]
dp[start] = nums[start];dp[start + 1] = Math.max(dp[start], nums[start + 1]);
loop i [start+2~end];dp[i] = Math.max(dp[i - 1], dp[i-2] + nums[i]);

D&C recursive
1. define int[2]: 0-Inclusive root, 1-exclusive root
2. result[0]=root.val + left[1]+right[1]; result[1] = Max(left[0],left[1])+Max(right[0],right[1])

DFS
1. left==0&&right==0 then result.add(s); return;
2. dfs(result, s + "(", left - 1, right); dfs(result, s + ")", left, right - 1); when left and right >0 respectively

DFS recursive: mid is root, start~mid -1 is left subtree; mid +1~end is right subtree
exit: if (start > end) return;

DP
count[i] += count[j] * count[i - j - 1];
以i为根节点的树，其左子树由[0, i-1]构成， 其右子树由[i+1, n]构成。

Let count[i] be the number of unique binary search trees for i. The number of trees are determined by the number of subtrees which have different root node.

如以1为节点，则left subtree只能有0个节点，而right subtree有2, 3两个节点。所以left/right subtree一共的combination数量为：f(0) * f(2) = 2
以2为节点，则left subtree只能为1，right subtree只能为2：f(1) * f(1) = 1
以3为节点，则left subtree有1, 2两个节点，right subtree有0个节点：f(2)*f(0) = 2

DFS recursive: iterate start~end all is root to get List<Node> collection.
exit: if (start > end) return list.add(null);
loop i:[start~end]
leftNodes = helper(start, i -1); rightNodes = helper(i+1,end)
double loop to permutation all of possiblity of i is root, per leftnode and rightnode

```
List<TreeNode> leftNodes = helper(start, i - 1);
        List<TreeNode> rightNodes = helper(i + 1, end);
        for (TreeNode lnode : leftNodes) {
           for (TreeNode rnode : rightNodes) {
              TreeNode node = new TreeNode(i);
              node.left = lnode;
              node.right = rnode;
              result.add(node);
           }
        }
```

Same with unique binary search tree

```
for (int i = 0; i < n; i++) {
        char c = s.charAt(i);
        if (c == '*' || c == '-' || c == '+') {
           List<Integer> left = diffWaysToCompute(input.substring(0,i));
           List<Integer> right = diffWaysToCompute(input.substring(i + 1));
           for (int l : left) {
              for (int r : right) {
                 if (c == '*') {
                    result.add(l * r);
                 } else if (c == '-') {
                    result.add(l - r);
                 } else if (c == '+') {
                    result.add(l + r);
```

Finally, need check result has element, if no any element means input only has digit

DP ： Two state, profit of hold stock and unhold stock
initial: hold[0] = -prices[0]; unhold[0] = 0;
loop i [1~n)
unhold[i] = Math.max(unhold[i - 1], hold[i - 1] + prices[i]);
hold[i] = Math.max(hold[i - 1], i >= 2 ? unhold[i - 2] - prices[i] : -prices[i]);

Binary Search

n=k时的Gray Code，相当于n=k-1时的Gray Code的逆序 加上 1<<k。
result = grayCode(n - 1); int numAdd = 1 << (n - 1);
for (int i = result.size() - 1; i >= 0; i--) { result.add(numAdd + result.get(i)); }

DFS recursive !list.contains(i)

reverse integer
for (int i = len - 2; i >= 0; i--) { if (num[i + 1] > num[i]) { swap ,reverse}

reverse(0, n -1);

DFS
int[] visited = new int[nums.length];
...
if (visited[i] == 1 || (i !=0 && nums[i] == nums[i - 1] && visited[i - 1] == 0)) {
  continue;
 }

有点复杂，节省时间没做，抄的九章
n个数的permutation总共有n阶乘个，基于这个性质我们可以得到某一位对应的数字是哪一个。思路是这样的，比如当前长度是n，我们知道每个相同的起始元素对应(n-1)!个permutation，也就是(n-1)!个permutation后会换一个起始元素。因此，只要当前的k进行(n-1)!取余，得到的数字就是当前剩余数组的index，如此就可以得到对应的元素。如此递推直到数组中没有元素结束。实现中我们要维护一个数组来记录当前的元素，每次得到一个元素加入结果数组，然后从剩余数组中移除，因此空间复杂度是O(n)。时间上总共需要n个回合，而每次删除元素如果是用数组需要O(n),所以总共是O(n^2)。这里如果不移除元素也需要对元素做标记，所以要判断第一个还是个线性的操作。

DFS: same with permutation, if list.size() == k then result.add(list)

DP coordinate type

DP, minSum to detect previous sum + nums[i] and nums[i] which are greater, max is last result of subarray

Biary Search target = nums[end]

ONLY using node and node.left to traversal
while (node!=null && node.left != null){curt = node;while(curt !=null) {curt.left.next = curt.right; curt.right.next = curt.next== null?null:curt.next.left;curt =curt.next;} node = node.left}

level order: queue, iterate size of queue(pre get size)
BUT add right at first

for (int i = 1; i <= num / i; i++) { if (i * i == num) {

level order

int left = 0; int right = n - 1; int top = 0; int bottom = m - 1;
while (result.size() < m * n) {
left->right, top->bottom,right->left, bottom->top

int left = 0; int right = n - 1; int top = 0; int bottom = m - 1;
while (k <= n * n) {
left->right, top->bottom,right->left, bottom->top

Binary Search

from nottom-left or top-right to find
eg: bottom-left row = m-1, col = 0;
while (row>= 0 && col < n)
 if (row,col) > target then row--; if (row,col) < target then col++

DP coordinate type