1 Algorithms are important for interview 1

2 SQL

2.1 Inner join, outer join

- Inner join, outer join
- foreign key
- Database Sharing, indexing
- noSQL, key and value

3 Shuffling array

```
shuffleArray int[] arr
    len = arr.length
    Random ran = new Random()
    for i=0 i<len - i i++
        r = ran.nextInt(len - i)
        tmp = arr[len - 1 - i]
        arr[len - 1 - i] = arr[r]
        arr[r] = tmp
```

4 Binary Tree

4.1 preorder

4.1.1 Application

more info

```
preorder node r
    stack
    while r not null or stack is not empty
        if r not null
            print r.data
            stack.push r
            r = r.left
        else
            n = stack.pop
            r = r.right
```

4.2 Inorder

```
inorder node r
    stack
    while r not null or stack is not empty
        if r not null
            stack.push r
            r = r.left
        else
            n = stack.pop
            print n.data
            r = r.right
```

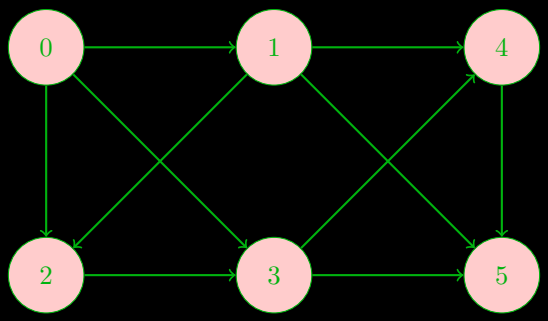
4.3 postorder, it is similar to breadth first search BFS with two queues

```
postorder node r
    stack s1 s2
        if r not null
            s1.push r
            while s1 is not empty
                n = s1.pop
                if n.left is not null
                    s1.push n.left
                if n.right is not null
                    s1.push n.right
                s2.push n

            while s2 is not empty
                print s2.pop
```

4.4 postorder 2 with one stack and isVisited flag

5 path problem

Find the all paths from two given nodes



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
// list.add n1
allpath arr2d n1 n2 list
    height = arr2d.length
    width = arr2d[0].length
    if n1 < height
        if n1 != n2
            for i=0 i<width i++
                if arr2d[n1][i] == 1
                    list.add i
                    path arr2d i n2
                    list.remove i
        else
            print list
```

Find the shortest path from two given nodes

```
shortest arr2d n1 n2 list mlist
    height = arr2d.length
    width = arr2d[0].length
    if n1 < height
        if n1 != n2
            for i=0 i<width i++
                if arr2d[n1][i] == 1
                    list.add i
                    shortest arr2d i n2 mlist
                    list.remove i
        else
            if mlist.size == 0
                mlist = list
            else
                if list.size < mlist.size
                    mlist = list
```

6 level order

print each levels from a binary tree

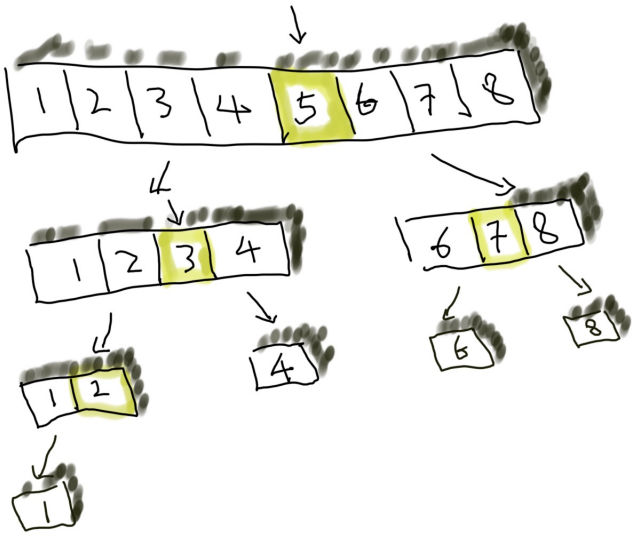
```
// connect each level
connectlevel node r
    queue q1 q2
    while r1 is not empty and r2 is not empty
        prev = null
        while r1 is not empty
            curr = q1.dequeue
            if prev is not null
                prev.next = curr
            prev = curr

            if curr.left is not null
                q2.add(curr.left
            if curr.right is not null
                q2.add(curr.right)

        prev = null
        while r2 is not empty
            curr = q2.dequeue
            if prev is not null
                prev.next = curr;
            prev = curr;
            if curr.left is not null
                q1.add(curr.left)
            if curr.right is not null
                q1.add(curr.right)

connectAllLevel node r
    queue q1
    if r is not null
        q1.add r
        prev = null
        while q1 is not empty
            curr = q1.dequeue()
            if prev is not null
                prev.next = curr
            prev = curr

            if curr.left is not null
                q1.add curr.left
            if curr.right is not null
                q1.add curr.right


// level order with one queue
levelorder node r
    queue q
    if r is not null
        q.enqueue r
        while q is not empty
            n = q.dequene
            if n.left is not null
                q.enqueue n.left
            if n.right is not null
                q.enqueue n.right


levelorder node r
queue q1 q2
if r not null
    q1.enqueue r
    while q1 is not empty or q2 is not empty
```

```
while q1 is not empty
    n = q1.dequene
    if n.left is not null
        q2.enqueue n.left
    if n.right is not null
        q2.enqueue n.right

while q2 is not empty
    n = q2.dequene
    if n.left is not null
        q2.enqueue n.left
    if n.right is not null
        q2.enqueue n.right
```

7 Binary Search

Binary Search is very simple algorithm, but it is hard to get it right in the first shot. It is easy to missing the one element case

```
// Java
bs k arr lo hi
    ret = false
    if lo <= hi
        mid = (lo + hi)/2
        if k < arr[mid]
            bs k arr lo (mid - 1)
        if k > arr[mid]
            bs k arr (mid + 1) hi
        else
            ret = true
    return ret

-- Haskell
bs::(Ord a)=>a -> [a]-> Bool
bs k [] = False
bs k cx = if l == 1 then k == head cx
                    else (if k < head re then bs k le
                          else (if k > head re then bs k (tail re) else True))
                  where
                     l = length cx
                     m = div l 2
                     le = take m cx
                     re = drop m cx
```



log(n)

8 Rotate square 2d array

Use reflection on x-Axis and $y = x$ 1. Swap elements along each column.
2. Reflect along the $y = x$

```
rotate array
    len = array.length
    for k=0 k<len/2 k++
        for i=k i<len-1-k i++
            tmp = array[k][i]
            array[k][i] = array[len-1-i][k]
            array[len-1-i][k] = array[len-1-k][len-1-i]
            array[len-1-k][len-1-i] = array[i][len-1-k]
            array[i][len-1-k] = tmp
```

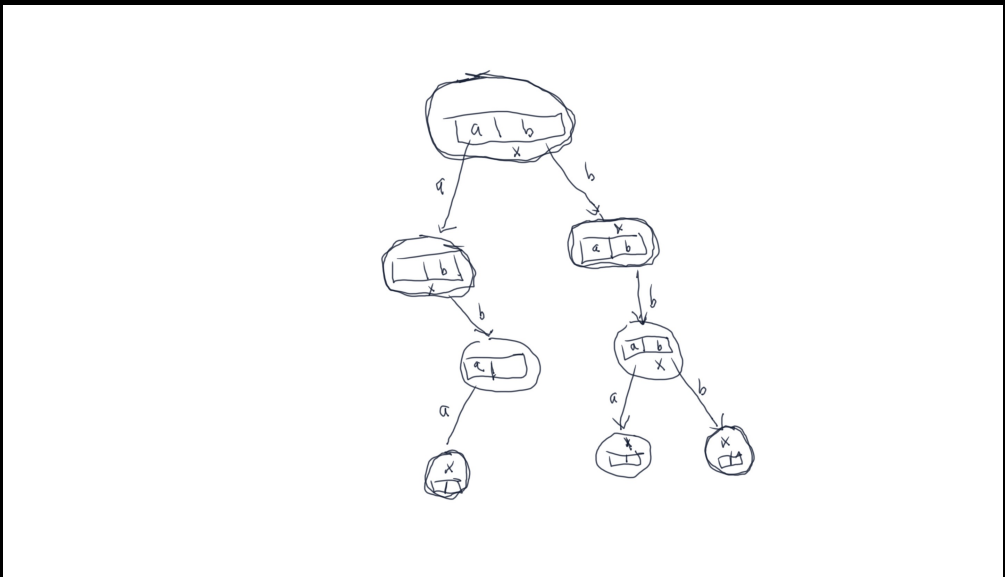
If we use the above code to print out **spiral** from a array, change to:

```
    for k=0 k<=len/2 k++
```

e.g. If len is 3, then $len/2 = 1$ then $k < len/2$ is missing the center element from the array.
There is no problem for rotation, but center element is missing if spiral is printed

```
rotate array
    len = array.length
    for k=0 k<len/2 k++
        for i=k i<len-1-k i++
            tmp = array[k][i]
            array[k][i] = array[len-1-i][k]
            array[len-1-i][k] = array[len-1-k][len-1-i]
            array[len-1-k][len-1-i] = array[i][len-1-k]
            array[i][len-1-k] = tmp
```

9 Tries insert and contains

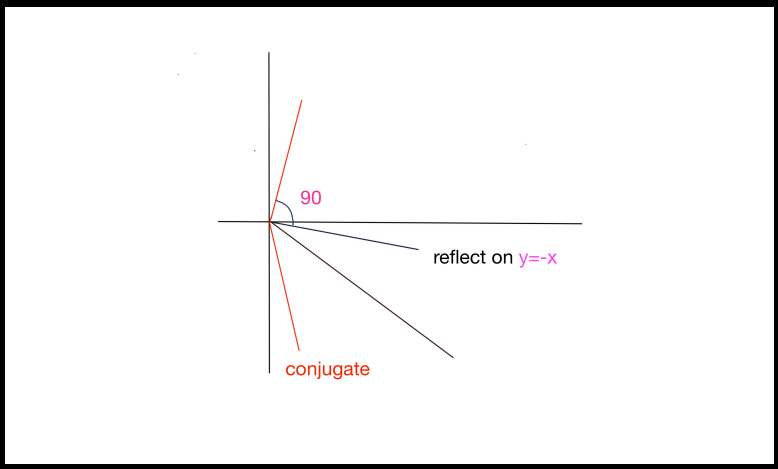



```
class TNode
        Map<Character, TNode> map = new HashMap<>()
        boolean isEnded = true
        public TNode boolean isEnded
            this.isEnded = isEnded

insert TNode curr, String s, Integer i
    if  s != null
        if i == s.length()
            curr.isEnded = true
        else
            c = new Character(s.charAt(i))
            node = curr.map.get(c)
            if node == null
               node = new TNode(false)
               curr.map.put(c, node)
            insert(node, s, i+1);


contains TNode curr, String s, int i
    if i == s.length()
        return curr.isEnded == true
    else
        if curr != null
            Character c = new Character(s.charAt(i))
            TNode node = curr.map.get(c)
             node != null
                return contains(node, s, i+1)
        return false
```

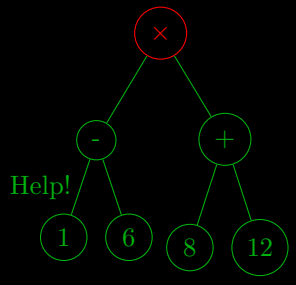
10 Rotate square 2d array 90 degrees CW with geometry technic

- Given $p(x, y)$
- conjugate of $p(x, y) \Rightarrow p(x, -y)$
- p reflects on $y = -x \Rightarrow p(y, -x)$
- dot product: $(x, y) \circ (y, -x) = 0$

```
void rotate90degree(int[][] arr){
    int len = arr.length;
    // reflect on y = x
    for(int i=0; i<len; i++){
        for(int j=i; j<len; j++){
            int tmp = arr[i][j];
            arr[i][j] = arr[j][i];
            arr[j][i] = tmp;
        }
    }
    // reflect on x = len/2
    for(int i=0; i<len; i++){
        for(int j=0; j<len/2; j++){
            int tmp = arr[i][j];
            arr[i][j] = arr[i][len-1-j];
            arr[i][len-1-j] = tmp;
        }
    }
}
```



11 Build Binary Tree from preorder



(× (- 1 6) (+ 8 12))

file:///Users/cat/myfile/bitbucket/java/EvaluateTree.java

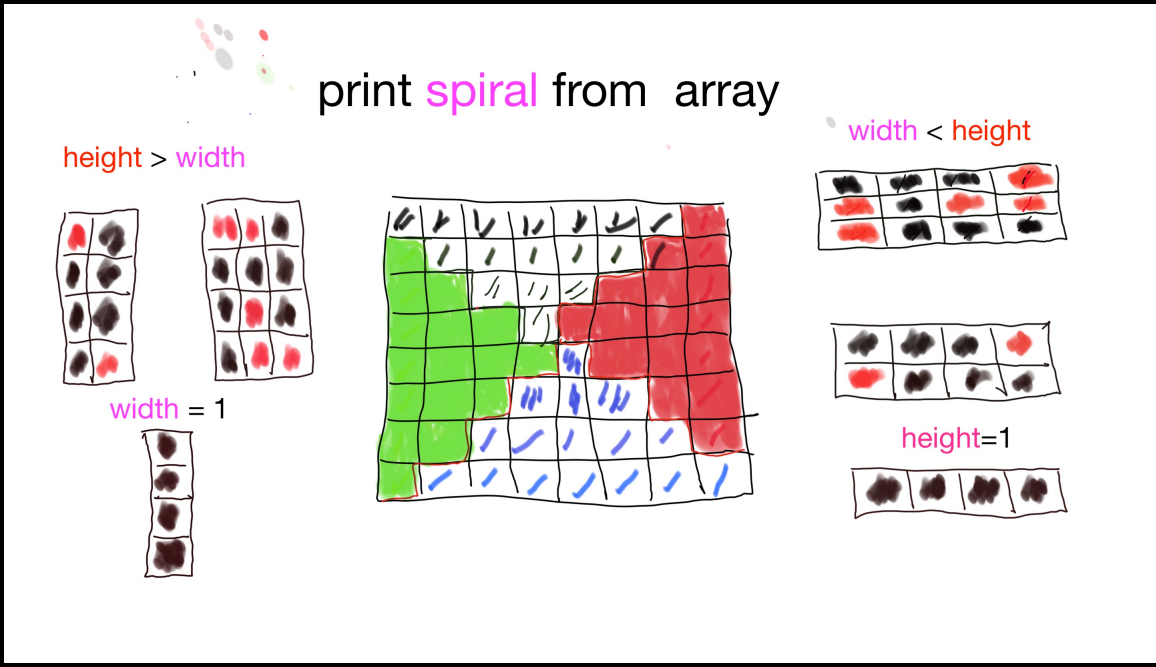
12 Serialize Binary Tree

```
try{

    BufferedWriter bw = new BufferedWriter(new FileWriter(fname))
    serializeGeneralTree(Node r, BufferedWriter bw)
        if r is not null
            bw.write(r.data)
            for(Node n : r.list)
                serializeGeneralTree(n, bw)
        bw.write("# ")
    bw.close()
}
catch(IOException io){
}

readFile(String file)
    list = new ArrayList<String>()
    try{
        BufferedReader bf = new BufferedReader(new FileReader(file));
        String line;
        while (line = bf.readLine()) != null
            String[] arr = line.split("\\s+")
            break

        for String s : ar
            list.add(s.strim())
    }catch(IOException io){
    }
return list

deserializeGeneralTree(List<String> list)
    stack = new Stack<String>()
    for String s : list
        if !s.equals("#")
            stack.push(new Node(s))
        else
            if stack.size() > 1
                Node pop = stack.pop()
                stack.peek().list.add(peek)
return stack.peek()
```

12.1 Serialize Binary Tree With Binary Heap

1. Use Binary Heap to index each node in the Binary Tree
2. Write the index ⇒ node to a file
3. Restrieve the index ⇒ node and store it in Map
4. Build the tree with postorder traveral

13 Print spiral shape from 2d array



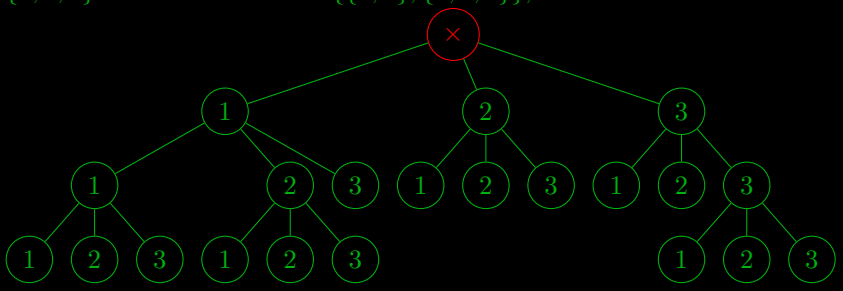
- If (width - 2*k == 1) or (height - 2*k == 1), then we can terminate the code after that.
- Otherwise, min(width, height) is even, then one row or one column DOES NOT exist.
- If the array is one row or one column then it depends on **width < height** or **width > height**
- <= is import here because if the width or height is odd, the center element is missing if < is used

```
spiral arr
    height = arr.length
    width  = arr[0].length
    k = 0
    while k <= Math.min(height, width)/2
        if height - 2*k == 1
            for i=k i<width-k i++
                print arr[k][i] // horizontal
            break
        else if width - 2*k == 1
            for(int i=k; i<height-k; i++)
                print arr[i][k] // vertical
            break
        else
            for i=k i<width-1-k i++
                print arr[k][i]
            for int i=k i<height-1-k i++
                print arr[i][width-1-k]
            for i=k i<width-1-k i++
                print arr[height-1-k][width-1-i]
            for i=k i<height-1-k i++
                print arr[height-1-i][k]
        k++
```

14 Coin Change Problem

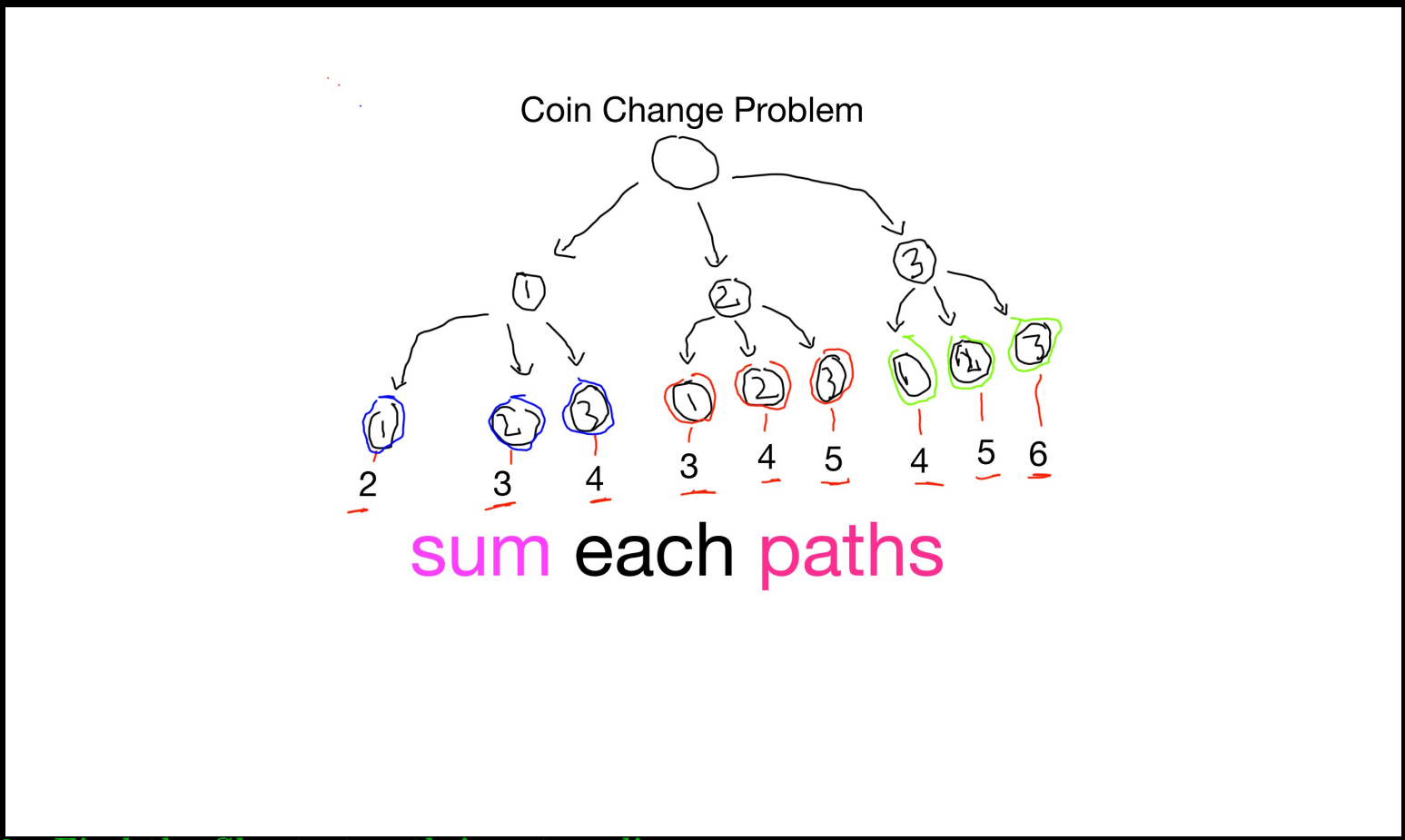
14.1 This is fun question

Given a set of coins and an integer S, find all the combination of coins are added up to the integer S. $\{2, 4, 3\}$ and $S = 7$ we have $\{\{4, 3\}, \{2, 2, 4\}\}$, the minimum number of coin is $\{\{4, 3\}\}$



15 Coin Change problem

```
coinChange coins list, s
    if s == 0
        print list
    if s > 0
        for n : coins
        coinChange coins list s − n
```



16 Find the Shortest path in a two dim array

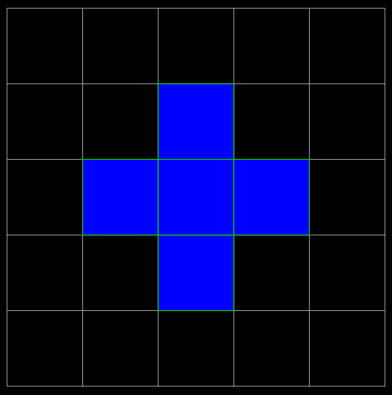
0	1	2	3	4	5	6	7
r=root	$r \to l$	$r \to l \to l$	null	$2 \to l$=null	$1 \to r$	$3 \to l$=null	$3 \to r$=null
stack	push 1	push 2	pop 2	pop 1	push 3	pop 3	x
print	x	x	2	1	x	3	x

- Given the goal is 2 in the two dim array.
- Find the shortest path from the top left corner of two dim array.
-

- Use Depath First Search
- If we hit the target `arr[h][w] == 2`, return `1`
- If the current position is valid, recurve to all the directions. e.g `h + 1, h - 1, w + 1, w - 1`
- If the current position is NOT valid, return a infinite number.
- Find the minimum of each branch
- Reset the current position `arr[h][w] = 1`

17 Connected island problem four directions

```
// four directories
int height = arr.length
int width = arr[0].length

count arr2d h w height width
    if arr2d[h][w] == 1
        arr2d[h][w] = 0
        int n1 = 0, n2 = 0, n3 = 0, n4 = 0
        if h + 1 < height
            n1 = count(arr2d, h+1, w, height, width)
        if h - 1 >= 0
            n2 = count(arr2d, h-1, w, height, width)
        if w + 1 < width
            n3 = count(arr2d, h, w+1, height, width)
        if w - 1 >= 0
            n4 = count(arr2d, h, w-1, height, width)
        return n1 + n2 + n3 + n4 + 1;
return 0
```



18 Connected isLand and longer path in Binary Tree

The two questions are similar
1. Go down to the bottom of a Tree
2. If we hit the bottom, then return 0 or 1
3. In the current node, return all the children node value + the current node which is 1

```
longerPath Node r
    if r != null
        le = longerPath r.left
        ri = longerPath r.right
        return le + ri + 1
    return 0
```

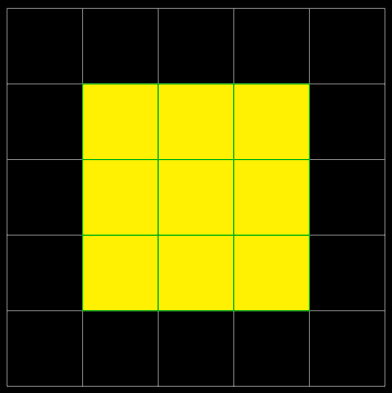
18.1 Connected isLand ⇒ Coin Change problem

The two problem is essentially the same.
1. Connected isLand is finding the longer path from a given node
   11. The terminated condition is when the block is NOT 1 or the index is out of bound
2. Coin Change is findind the shortest path from a given node
   22. The terminated condition is when the sum is equal the a number

19 Connected island problem Eight directions including diagonal

Iterate nine directions
The problem is similar to → find all words in a two dime array.

```
// eight or nice directions, h = 0 or w = 0 can be ignored in the loop
count8 arr2d h w height width
    sum = 0
    if arr2d[h][w] == 1
        arr2d[h][w] = 0
        for hx= −1 hx <= 1 hx++
            for wx = −1 wx <=1 wx++
              if 0 <= h + hx && h + hx < height && 0 <= w + wx && w + wx < width
                    sum += count8 arr2d h + hx w + wx height width

        // current position is valid movement
        sum += 1
    return sum
```



20 Sudoku Solver

20.1 BackTracking

BackTracking is general algorithm for finding all/some solutions to contraint satisfaction problems, that incrementally builds candidates to the solution, and abandons each partial c[backtracks] as soon as it determines that c cannot be possibly completed to a valid solution.

For Sudoku problem, the constraint/restriction is row/column and 3×3 square have no duplicated number from 1 to 9. If the number is not valid candidates, reset the cell to previous value and return back to parent. Check Row and Column have no duplicated number Check each 3×3 square has no duplicated number Try an empty cell with 1 to 9 If the number is valid in the cell, then recur to next empty cell Otherwise, set the cell to original value and return back to parent

```
solver arr index
    c = index / 9, r = index % 9
    if index == 9*9
        print arr
    else
        for i 1 to 9
            if arr[c][r] == 0
                if checkRowCol arr c r i && checkSquare arr c r i
                    arr[c][r] = i
                    solver arr index + 1
                    arr[c][r] = 0
            else
                solver arr index + 1

checkRowCol arr int c int r int n
    for i 0 to 9
        if arr[c][i] == n || arr[i][r] == n
            return false
    return true
```

```
checkSquare arr c r n
    ic = c/3 ir = r/3
    for c 0 to 3
        for r to 3
            arr[c + 3*ic][r + 3*ir] == n
                return false
    return true
```

21 Eight Queen

Eight Queen problem is similar Sudoku problem, and can be solved with Backtracking algorithm The constraint satisfaction is simpler than Sudoku.

21.1 Runtime is $\mathcal{O}(8^n)$

Check whether a queen is in the same column as all other queens Check whether a queen is in the same main diagonal or minor diagonal with all other queens
No two queens are on the same row or column Recur down each row If a cell is valid, then go to next row Otherwise, reset the cell and return back to parent/previous call

```
eightQueen
```

22 Permutation

Networking
Three ways hands shake
slide window inside the package
DNS, resolve domain name, IP
TCP, UDP, HTTPS
Design
load balance
B replication
Sub-Pub
command pattern
visit pattern
Singleton,
Double-checked locking
message queue
consume and producer

1. Quick Sort
    - The average runtime is $\mathcal{O}(n \log n)$
    - The worst case is $\mathcal{O}(n^2)$
        - Given an array $[1, 2, 3, 4]$, choose the right most element as pivot which is [4] => [1, 2, 3][4] => [1, 2][3] => [1][2]
        - How to choose the pivot is critical.
    - Memory space is $\mathcal{O}(1)$
    - Untable sort and Stable sort

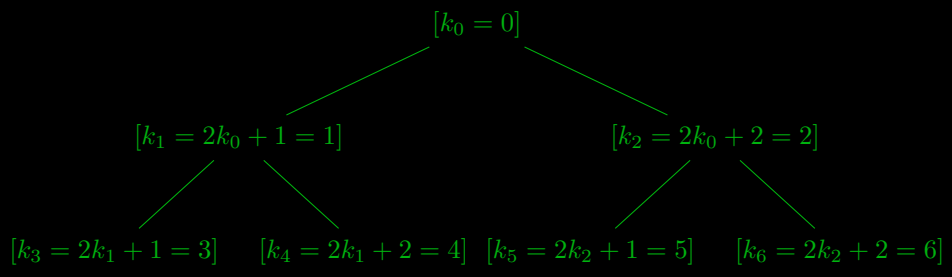
(4,1) (2,3) (4,3)
Sort the first coordinates [stable sort]
(2,3) (4,1) (4,3)

1. Find the Kth smaller element in a given unsorted array in $\mathcal{O}(n)$
2. Merge Sort,
3. The average and worst runtime is $\mathcal{O}(n \log n)$
4. Stable sort

1. Max distance for $j - i$ given $arr[j] > arr[i]$
2. Single linked list
3. reverse, iteration, recursion
4. remove,
5. insert to sorted list
6. clone list
7. check circular linkedlist

1. Double linkedlist
2. remove node
3. insert node
4. append node

1. Eight queen problem
2. Sudoku Solver problem
3. Connected island
4. Implement heap with array
5. Heap with array

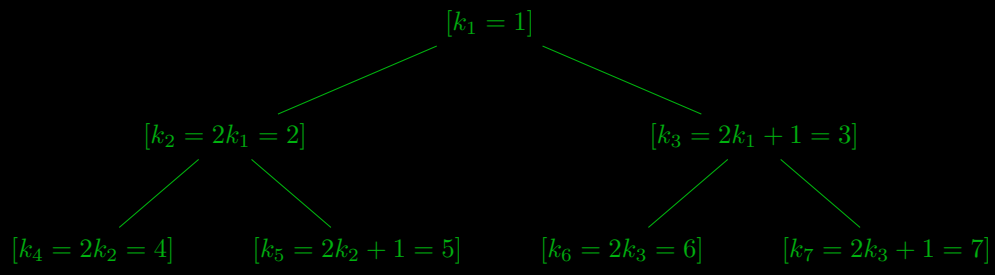


23 Serialize Binary Tree

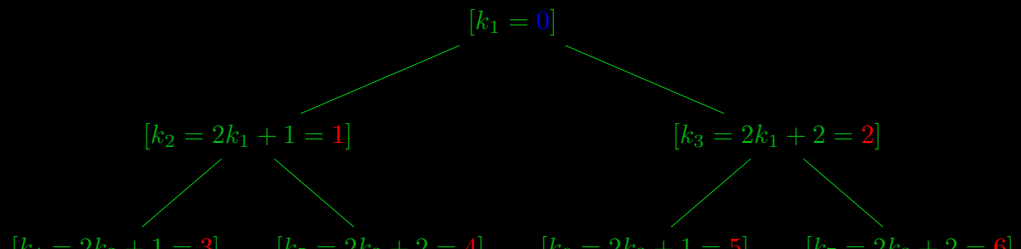
Use technic similar to the implementation of Binary Heap with array level 1

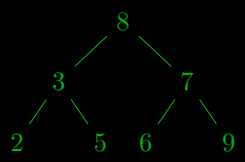
$$2^0 + 2^1 + \cdots + 2^k$$

$$2^{k-1} = 1, 2$$



Start with zero in the root node is better because it can be extended to k number of children





// [0 : 8] [1 : 3] [2 : 7] [3 : 2] [4 : 5] [5 : 6] [6 : 9]
Extend to n number of children, e.g. $n = 3$
Start from $k_0 = 0$

$$k_1 = 3k_0 + 1, 3k_0 + 2, 3k_0 + 3$$
$$k_2 = 3k_1 + 1, 3k_1 + 2, 3k_1 + 3$$
$$\vdots$$
$$k_m = 3k_{m-1} + 1, 3k_{m-1} + 2, 3k_{m-1} + 3$$

```
level_serial Node root, int k, String file
    if root != null
        try
            BufferedWriter bf = new BufferedReader(new FileWriter(file))
            bf.write(k+ ":" + root.data + "\n")
            level_serial(root.left, 2*k, file)
            level_serial(root.right, 2*k+1, file)
        catch(IOException e)
            e.printStackTrack()

createMap String file
    bf = new BufferedReader (FileReader(file))
    Map<Integer, Integer> map = new HashMap<>()
    String line
    try( (line = bf.readLine()) != null)
        String[] arr = line.split(":")
        map.put(arr[0], arr[1])
    catch(IOException e)
        e.printStackTrack()
    return map
```

```
// k = 0
Node buildTree(Map<Integer, Integer> map, int k)
    Node r = null
    Integer n = map.get(k)
    if n != null
        r = new Node(n)
        r.left = buildTree(map, 2*k + 1)
        r.right = buildTree(map, 2*k + 2)
    return r;
```

24 Tree Traveral

preorder, inorder and postorder
pretty print
Use those algorithms to serialize Binary Tree, postorder can be used to deserialize BT.
Inorder can be used to check whether a binary tree is BST.

25 Level order

use two queues to print Binary tree in level order.
One queue is for odd level, other queue is for even level
Print level order without queue.

compute the height of of the Binary Tree
$Node(k) = 2^{k-1} \quad k = 1, \ldots, n$

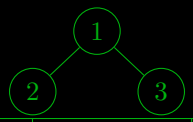
```
level Node root, Map<Integer, Integer> map, int k
    if root not null
        mpa.put(k, root.data)
        print(root.data)
        level(root.left, 2*k)
        level(root.right, 2*k+ 1)

printLevel (Map<Integer, Integer> map)
    c = 0, l = 0
    while count < map.size()
        p = (int)Math.pow(2, l)
        for int i=0 i<p i++
            key = map.get(p + i)
            if(key != null){
                System.out.print(key)
                c++
        l++

Node buildTree(Map<Integer, Integer> map, int k)
    Node r = null
    Integer n = map.get(k)
    if n != null
        r = new Node(n)
        r.left = buildTree(map, 2*k + 1)
        r.right = buildTree(map, 2*k+ 2)
    return r
```

iteration preorder, inorder, postorder, levelorder
iteration preorder

initialize $r = root$, stack
iterate left children with r and push the r to stack, at the same time printing out the data
if r is null, then pop a node from the stack and set the curr reference to the right child of the node
it will terminate if r is null and stack is empty

```
 preorder(Node r){
     if(r != null){
         r.data
         preorder(r.left)  // implicitly r = r.left
         preorder(r.right) // implicitly pop() => r = r.right
     }
 }

preorderIte(Node r) {
    Stack<Integer> stack = new Stack<>();
    while(r != null || !stack.isEmpty()) {
        if(r != null) {
            print(r.data)
            stack.push(r)
            r = r.left
        } else {
            Node n = stack.pop()
            r = n.right
        }
    }
}
```

iteration inorder



0	1	2	3	4	5	6	7
r=root	$r \to l$	$r \to l \to l$	null	$2 \to l$=null	$1 \to r$	$3 \to l$=null	$3 \to r$=null
stack	push 1	push 2	pop 2	pop 1	push 3	pop 3	x
print	x	x	2	1	x	3	x

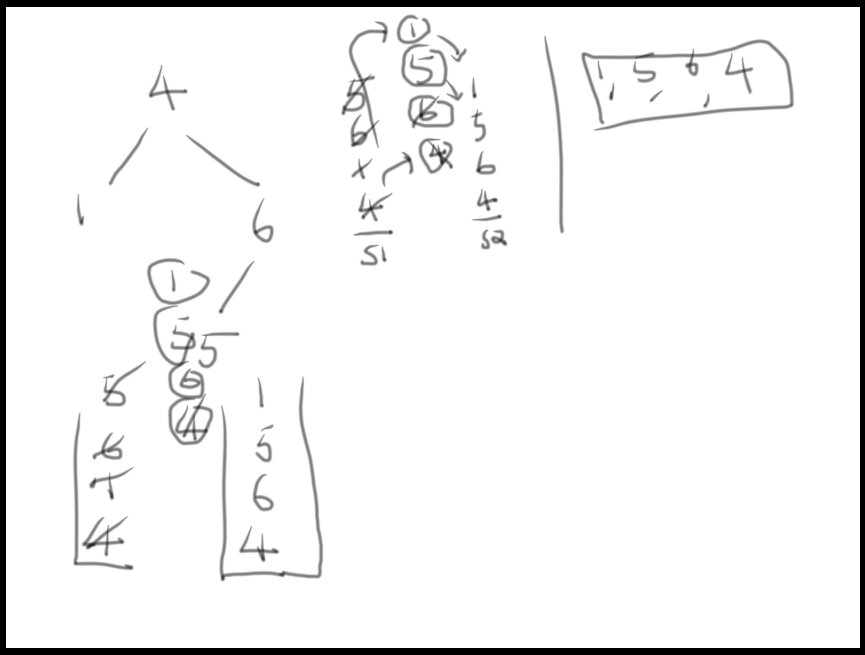
```
inorder(Node r){
    if(r != null){
        inorder(r.left)  // => r = r.left, push to stack
        print(r.data)    // => pop(), print(r.data)
        inorder(r.right) // => r = r.right
    }
}

inorderIte(Node r){
    Stack<Node> stack = new Stack<>();
    while(r != null || !stack.isEmpty()){
        if(r != null){
            stack.push(r)
            r = r.left
        }else{
            Node n = stack.pop();
            print(n.data)
            r = n.right
        }
    }
}
```

iteration postorder

```
postorderIte(Node r){
    Stack<Node> s1, s2 = new Stack<>();
    if(r != null){
        s1.push(r);
        while(!s1.empty()){
            Node top = s1.pop();
            if(top.left != null)
                s1.push(top.left);
            if(top.right != null)
                s1.push(top.right);

            s2.push(top);
        }

        while(!s2.empty())
            s2.pop()
    }
}
```

level order sequence with two stacks
Use two stacks → print Binary Tree in sequency order (inorder traveral without recursion)

```
public static void printSequence(Node r){
   Stack<Node> s1, s2  = new Stack<>()
   if(r != null){
       s1.push(r)
       while(!s1.empty() || !s2.empty()){
           while(!s1.empty()){
               Node n = s1.pop()
               Print.p(n.data)
```



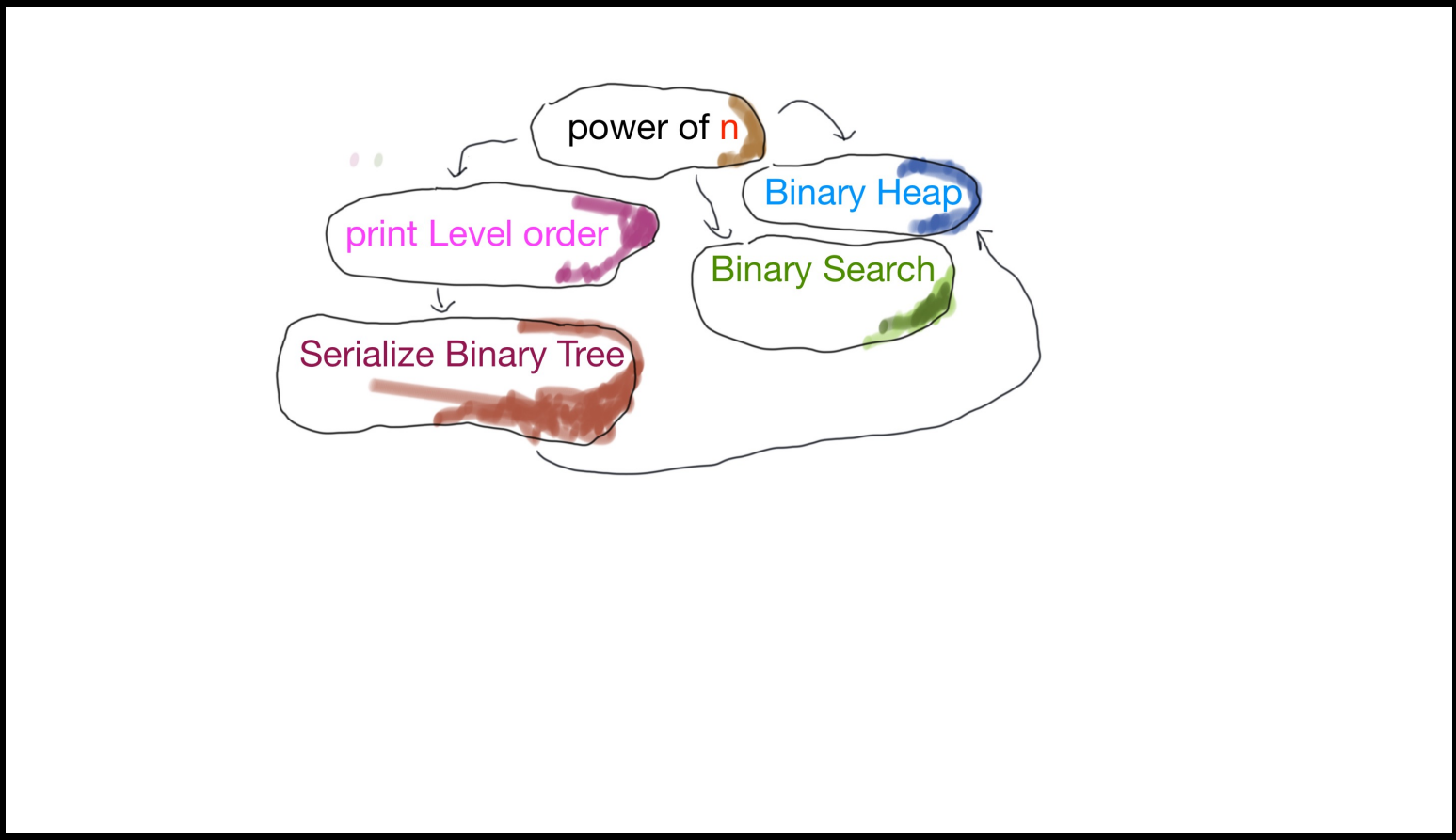

```
            if(n.left != null)
                s2.push(n.left)
            if(n.right != null)
                s2.push(n.right)
        }
        while(!s2.empty()){
            Node n = s2.pop()
            Print.p(n.data)
            if(n.right != null)
                s1.push(n.right)
            if(n.left != null)
                s1.push(n.left)
        }
    }
  }
}
```

26 Power of n

```
pow(int n, int e)
    if(e == 0)
        return 1
    else
        if e % 2 == 0
            return pow(n, e/2) * pow(n, e/2)
        elwse
            return n*pow(n, e/2) * pow(n, e/2)
```

The inverse of power of n is check whether n is the power of n

```
isPowerOfK(int n, int k)
    // zero is not the power of k
    r = 0
    c = 0
    while(n > 0)
       r = n % k
       if r == 1
          c++
       n = n / k
    return r == 1 && c == 1;
```



27 Get the maximum value from a stack

```
class MyStack{
    ArrayList<Integer> list;
    public MyStack(){}
    public void push(Integer n){
        list.add(n);
    }
    public Integer pop() throws Exception{
        if(list.size() > 0)
            return list.get(list.size() - 1);
        else
            throws new Exception("The stack is empty");
    }
}
```

Print rectangle array in spiral shape.
Multiply long integers using array
Find the maximum elements from sorted array are shifted
Find the minimum elements from sorted array are shifted
Merge sorted arrays
Merge k sorted arrays
Maximum and minimum heap
Print n prime and prime number
Use array to represent heap

The technic can be used to serialize and deserialize Binary Tree

heap sort
Dynamic Programming

maximum continuous sum in $\mathcal{O}(n)$

how to support negative number
print the indexes out

maximum non-continuous sum $\mathcal{O}(n)$
multiply all the elements in an array except current element $\mathcal{O}(n)$

Graph
find a path from two nodes (Use Breadth First Search)
Check if the n1 and n2 are equal
if n1 and n2 are equal. we are done!
if n1 and n2 are not equal.
add n1 to a list
get the first child of n1 and recur with the child.

find the shortest path from two nodes
find the minimum weight path from two nodes
how to find the loop in a graph find all the neighbours which are kth distance from a given node

How to represent a Graph

- adjecent matrix
- adjecent list
- What is the different between the two data structures

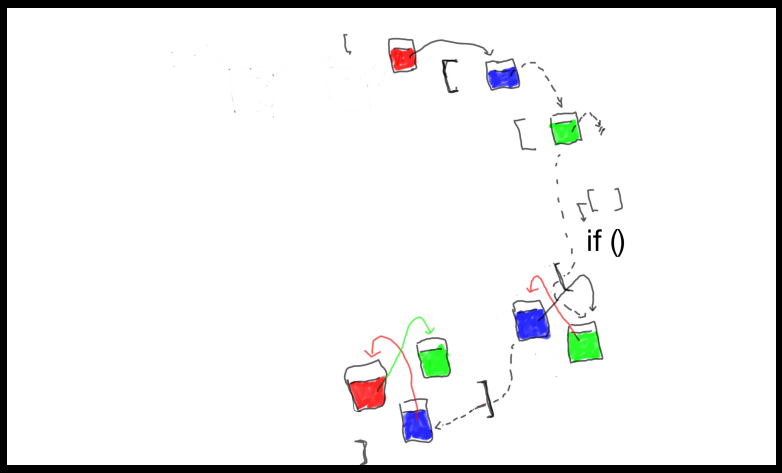
BackTracking
Coin Change problem.
Connected island Problem.
find the minimum number of coins [shortest path from the root]
find the maximum number of coins [longest path from the root]
dynamic programming with HashMap
Eight Queen Problem
Sudoku Solver
Find the maximum number of connected dots in an 2d array
Find the path from one word to other word that you can change one letter to a valid word in dictionary only once for each step. (Facebook question)
Use Depth First Search, DFS
Given two word1 and word2 and a dictinary
start from the first word
change the first position $word[0]$ from $[a-z]$
if the new word is a valid word in the dictionary
goto next position, `pos + 1` recurvely.
if try all possible words from $[a-z]$ and non of them are valid word
return back to the previous recursive call and try the next letter
until the second word2 is found.
otherwise, there is no path from word1 to word2

Binary Tree
Check a Binary Tree is Binary Search Tree
recursion technic
defintion technic
Check whether two Binary Tree are isomorphic
Find the mirror of a Binary Tree
Find the longest path in a Binary Tree
Print all the paths in a Binary Tree
Find the maximum sum of path in a Binary Tree
Invert a Binary Tree
Binary Tree to linkedlist
Binary Tree to circular double linked list [hard]
Binary Tree to single linked list with one queue
Delete whole tree
Delete whole tree
Delete whole tree
Use two queues
Post order traveral
Use memory space $\mathcal{O}(1)$
Move one branch to branch
Lease Recent Used[LRU]
Implements HashMap
Context Switch invoke switching reg, stack pointer, program counter
Synchronize LinkedList
add() and *delete*()
Use two locks
if the node is a head, lock it and delete it, easy!
if the node is not head, then lock the previous and current nodes
if a node is found, delete it
Why it works?
If the current node is locked, you can delete the next node safely
If current node is deleted, then previous node must be locked
What is deadlock
What is starvation
Mutex is same for Binary Semaphore
Semaphore is synchronization construct that can be used to provide

mutual exclusion and conditional synchronization
Context Switch
Singleton, double-checked locking
Consumer and Producer
Single LinkedList with two locks add() and delete()
Java concurrent.Atomic library, AtomicInt, AtomicRef
Compare and Set [CAS]
Thread, Process, Lock, Spinlocks, Mutex, Semaphore, Compare And Set[CAS]
Synchronize delete or add node in LinkedList. when and where to lock

28 Reverse Linkedlist with recursion

- Idea: use the previous node to carry the head node out.
- Recurve down to the last node
- If the current node is head, return it
- If current node is not a head, use previous node links the current node
- And attach the head to current next
- The caller needs to check node.next or node
- If the node next is not null, then node.next is head. Otherwise node is the head. (only one node)
-
- How to check the return node is head or not?
- Use prev == null ⇒ curr is head



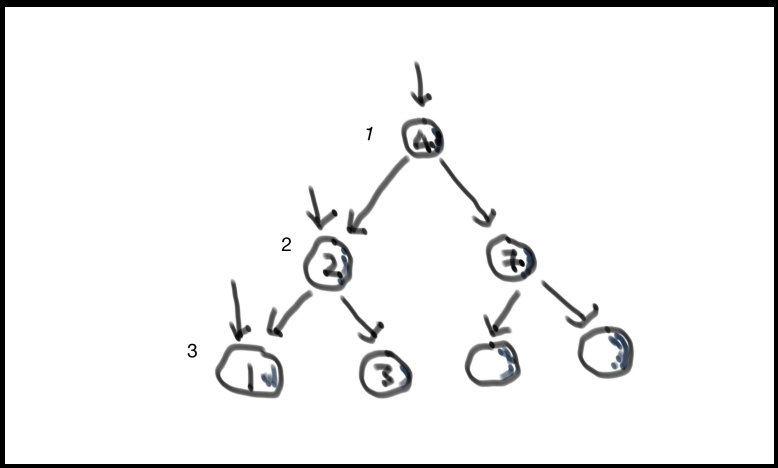

```
Node reverseSingleLinkedList(Node curr)
  if curr != Null
      Node prev = reverseSingleLinkedList(curr.next)
      if prev != null
        Node head = prev.next == null ? prev : prev.next
        prev.next = curr    // link prev and curr
        curr.next = head    // attach head to curr.next

  return curr

  // caller
  Node prev = reverseSingleLinkedList(curr)
  Node head = prev
  if prev != null
    if prev.next != null
      head = prev.next      // two or more nodes
      prev.next = null                        // remove the head from the tail

 // Put all in one function
 Node reverseLinkedListRecur Node curr
      Node prev = reverseSingleLinkedList(curr)
      Node head = prev
      if prev.next != null
         head = prev.next
         prev.next = null   // make sure to remove the head from the tail, otherwise there will be extra element

      return head
```

29 Binary Tree Problems

29.1 Insert a number into a Binary Tree

Given a Binary Tree, insert a node to the Binary Tree.

- Given a root and a node, insert the node to the tree
- If the root node is null, then new Node
- Otherwise, compare the current node with the number
- If the number less than the current node, then check if the left substree is null
- If the left substree is null, then create new Node in the left substree, break, End
- Otherwise, goto left substree
- If the right substree is null, then create new Node in the right substree, break, End
- Otherwise, goto the right substree

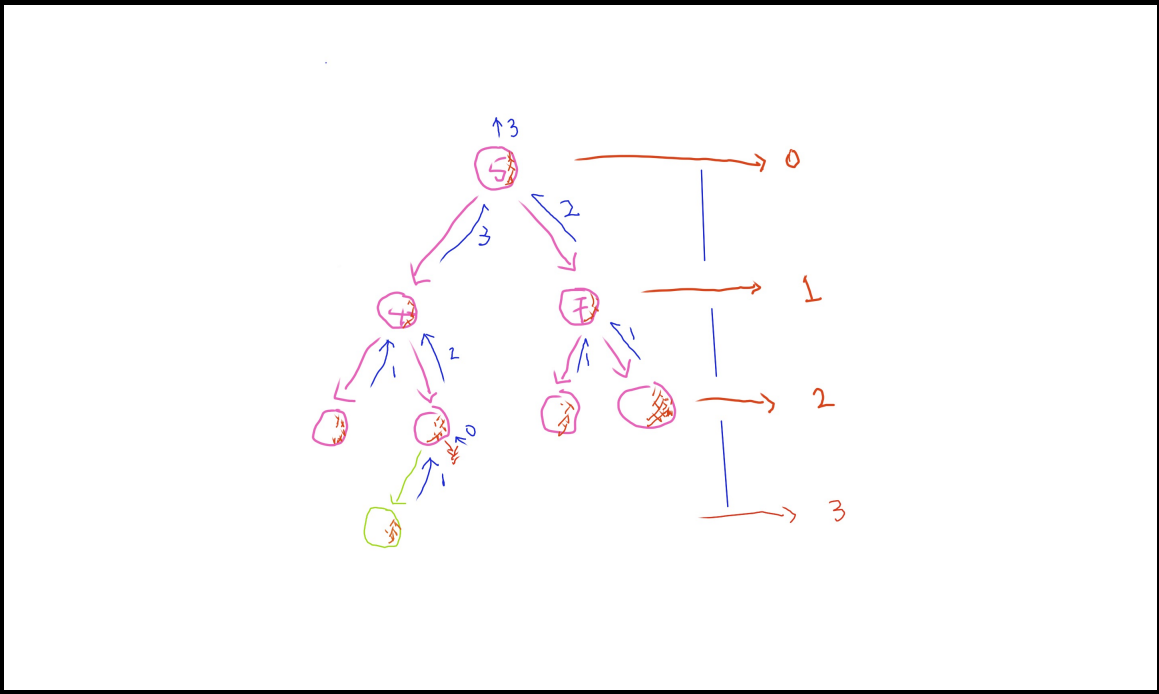



```
class Node
    Int data
    Node left
    Node right
    Node(Int data)
        this.data = data

class BinaryTree
    Node root
    public void insert(Int data)
        Node r = root
        if r == null
            r = root = new Node(data)
        else
            while r != null
                if data < r.data
                    if r.left == null
                        r.left = new Node(data)
                        break
                    else r = r.left

                else
                    if r.right == null
                        r.right = new Node(data)
                        break
                    else
                        r = r.right

// Recursive intersection
// root will be changed after the call
//
// Node myroot = root;
```

```
// insert(root, 3)
insert(Node root, int n)
    if(root == null)
        root = new Node(n)
    if(n < root.data)
        if root.left == null
            root.left = new Node(n)
        else insert(root.left, n)
    else
        if root.right == null
            root.right = new Node(n)
        else insert(root.right, n)
```

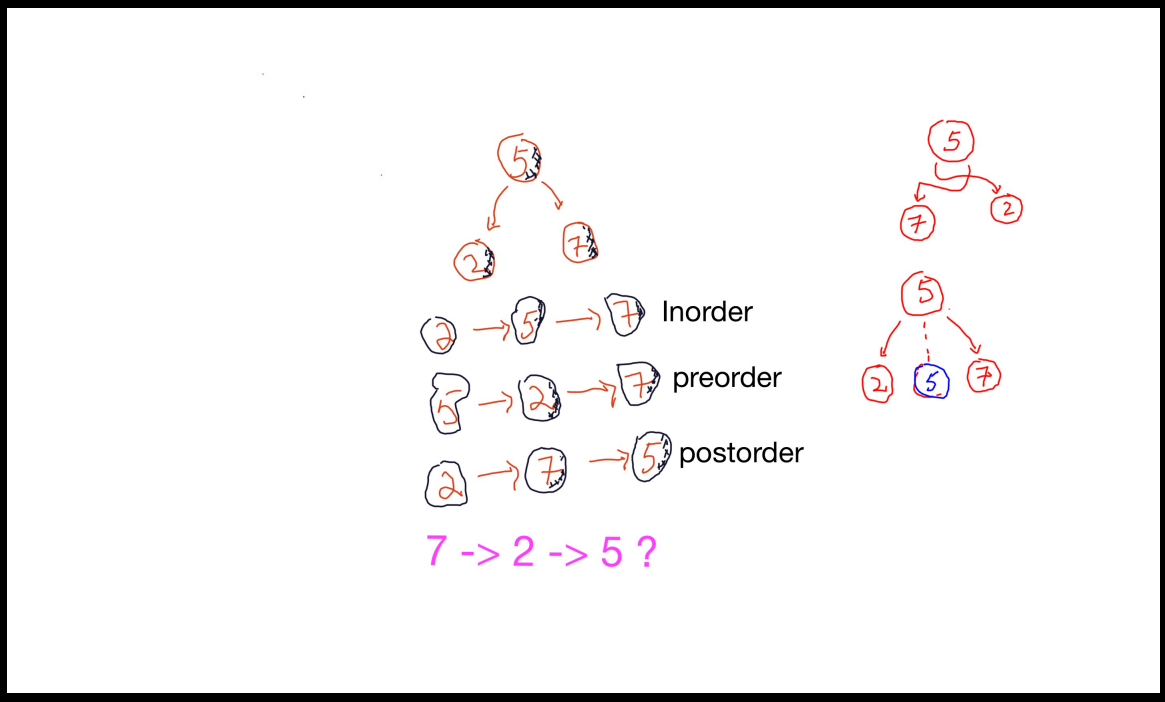
29.2 Maximum path in a Binary Tree

```
Int maxlevel(Node root)
    if root != null
        Int l = maxPath(root.left)
        Int r = maxPath(root.right)
        return max(l, r) + 1
    else
        return 0

Int maxHeight(Node root){
    return level(root) - 1
}

or
Int maxHeight(Node root)
    if root != null
        Int l = maxPath(root.left)
        Int r = maxPath(root.right)
        return max(l, r) + 1
    else
        return -1
```

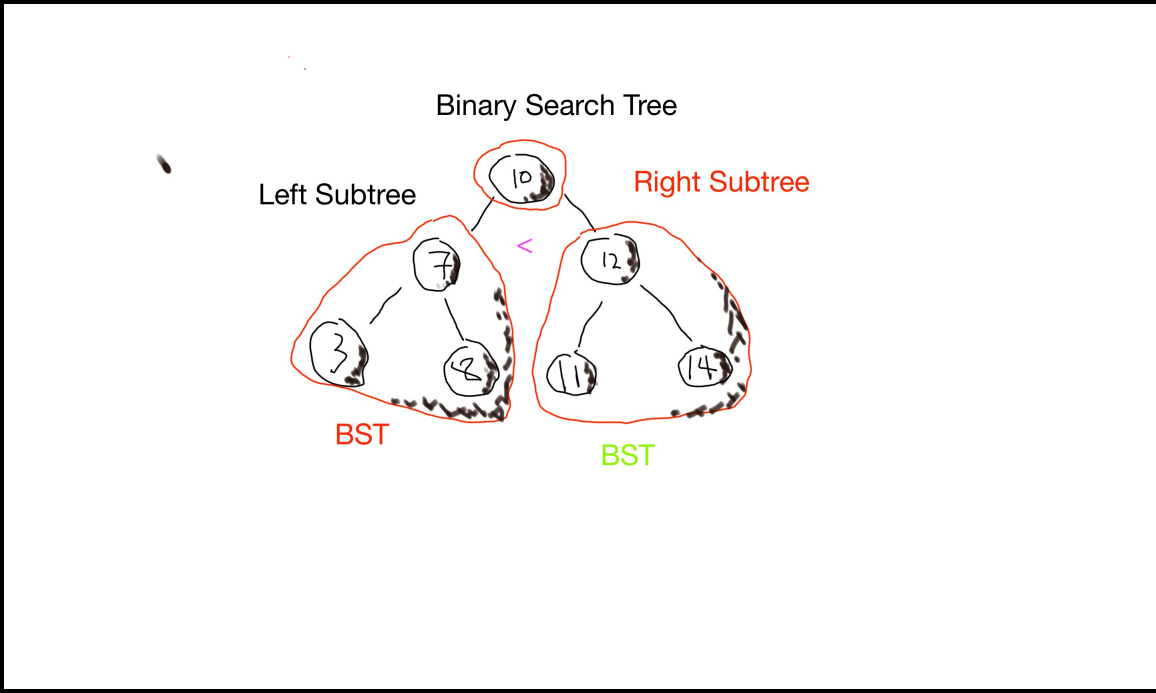


29.3 Binary Tree Preorder, Inorder and Postorder

```
inorder Node root
    if root != null
        inorder root.left
        root.data
        inorder root.right

inorderRev Node root
    if root != null
        inorderRev root.right
        root.data
        inorderRev root.left

preorder Node root
    if root != null
        root.data
        preorder root.left
        preorder root.right

postorder(Node root)
    if root != null
        postorder root.left
        postorder root.right
        root.data

right_left_top Node root
    if root != null
       right_left_top root.right
       right_left_top root.left
       root.data
```



29.4 Check whether a Binary Tree is BST



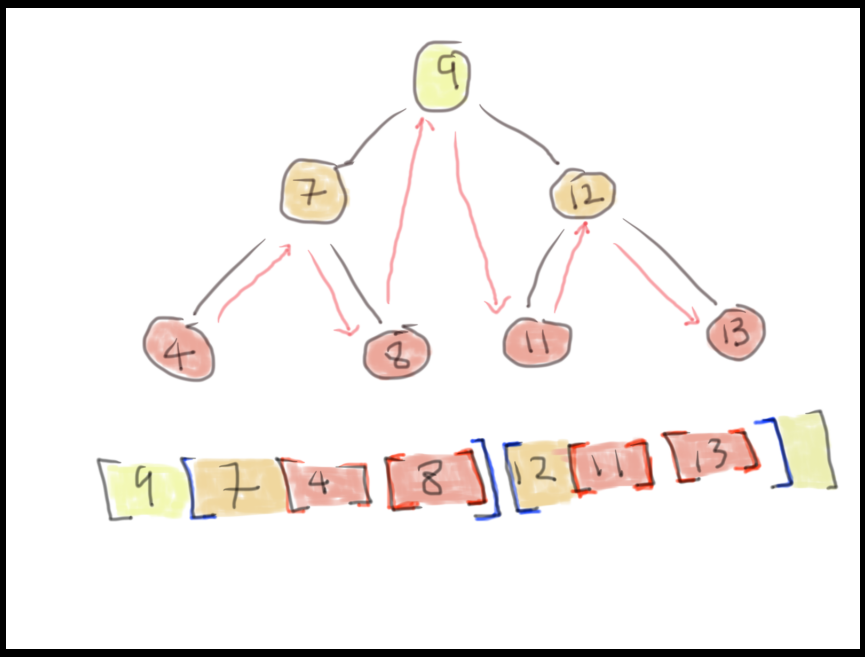
29.4.1 Binary Search Tree Definition

- Empty tree is BST
- Left substree is BST
- Right substree is BST
- The maximum left subtree < parent node
- The minimum right substree > parent node
- The whole Binary Tree is BST

$$BST = \begin{cases} \{\text{Empty tree is BST} \\ \begin{cases} \text{Left substree is BST} \\ \text{Right substree is BST} \\ \text{The maximum left subtree} < \text{parent node} \\ \text{The minimum right substree} > \text{parent node} \end{cases} \end{cases}$$

```
// root != null
int leftMax Node curr
    if curr.right != null
        leftMax curr.right
    return curr.data

// root != null
int rightMin Node curr
    if curr.left != null
        rightMax curr.left
    return curr.data

isBST Node root
    if root != null
        if !isBST root.left  // left substree is NOT a BST
            return false
        if !isBST root.right // right substree is NOT a BST
            return false
        if root.left != null && leftMax(root.left) > root.data
            return false
        if root.right != null && rightMin(root.right) < root.data
            return false
    return true
```

29.5 Use Recursive Algorithm, following the picture



30 Quick Sort

QuickSort is important in sorting algorithms family because the runtime of QuickSort is $\mathcal{O}(n \log n)$ on average case and the memory cost is $\mathcal{O}(1)$, it means you don't need a buffer to hold the elements. Although the worst runtime is $\mathcal{O}(n^2)$, it is very unlikely case in real life data.

30.1 Difference Between Quick Sort and Merge Sort

- **Quick Sort** is **unstable sort**
- **Merge Sort** is **stable sort**

Given a list as following:

$$[(1,2),(3,5),(2,4),(6,2)]$$

Quick Sort the first column

$$[(1,2),(2,4),(3,5),(6,2)]$$

Quick Sort the second column

$$[(1,2),(6,2),(2,4),(3,5)] \text{ or } [(6,2),(1,2),(2,4),(3,5)]$$

If we use Merge Sort, the last step will be only one **unique** output

$$[(1,2),(6,2),(2,4),(3,5)]$$

For Quick Sort, the output can depend on how you choose the pivot.
For Merge Sort, the output xxx The main reason for that is the **Merge Algorithm** e.g. merge two sorted lists

- **Quick Sort** is used **Preorder Traveral** - top down
  - Partition the list into left pivot right
  - Recurse the left sublist
  - Keep the pivot position (index)
  - Recurse the right sublist

```
quickSort::(Ord a)=>[a] -> [a]
quickSort (x:cx) = (quickSort [ l | l <- cx, l < x]) ++ [x] ++ (quickSort [ r | r <- cx, r >= x])
```

- **Merge Sort** is used **Postorder Traveral** - bottom up
  - Partition the list into left half and right half
  - Sort the left sublist
  - Keep the pivot position (index)
  - Sort the right sublist

30.2 Quick Sort Recursion

choose the pivot and partition array to two parts

all the elem are less than pivot are on left part excluding the pivot

all the elem are greater than pivot are on right part excluding the pivot

30.2.1 Quick Sort Partition

The tricky part of quick sort is how to partition an array to two parts. A Naive way is go through the element one by one and copy all the elements are less than the pivot to an array and do the same for the elements are greater than the pivot. The above solution works but the memory cost is $\mathcal{O}(n)$
There is in-place solution for partition.

- Let p to be the index of **array** that tracks the element is greater than the pivot.
- Let i to be the index from 0 to $len(\textbf{array}) - 1$
- p and i are both start from 0 and advance at the same time.
- If p is greater than the pivot, p stop increasing. Otherwise swap p and i

```
partition array
    p = 0
    for i=0 i<len i++
        if array[i] <= pivot
            swap p i
            if p < array.length - 1
                p++

return p
```

30.2.2 Preorder pattern in Quick Sort

The recursion part of the quick sort is using preorder traveral.

```
preorder node r
    if r is not null
        r.data
        preorder r.left
        preorder r.right

quicksort array lo hi
    if lo < hi
        p = partition array lo hi
        quicksort array lo p-1
        quicksort array P+1 lo
```

30.3 When is the runtime $\mathcal{O}(n^2)$

Example 1. *Given a sorted list* $[1, 2, \dots n]$ *and the right most element are chosen as pivot then* $[1, 2, \dots n-1][p = n]$ *is generated in the first partition. So we have following*
$[1, 2, \dots n-1][p = n]$
$[1, 2, \dots n-2][p = n-1][p = n]$
$[1, 2, \dots n-3][p = n-2][p = n-1][p = n]$

Every step, only the left part of the array which is the full array excluding the pivot are shuffled. Thereforce, the runtime is $\mathcal{O}(n^2)$

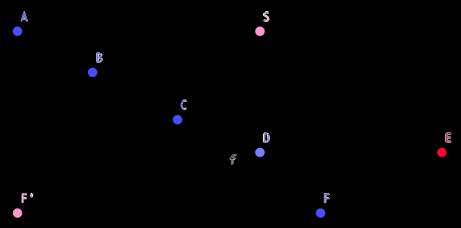
30.4 Quick Sort Iteration

Use preorder iteration algorithm $\rightarrow$ one stack

```
preorderIteration Node r
    while(r is not null or stack is not empty)
        if r is not null
            print r.data
            stack.push r     // f(r ← r.left)
            r = r.left
        else
            n = stack.pop()
            r = n.right      // f(r ← r.right)
```

31 Quick Sort Partition Algorithm

- Choose pivot - last element in the array
- Two pointer i, j points to the beginning of the array
- If arr[j] is greater than the pivot stop increasing j
- Else increasing j by 1
- If arr[i] $<$ pivot then swap arr[i], arr[j]
- Increase j by 1 if i is NOT in the last index

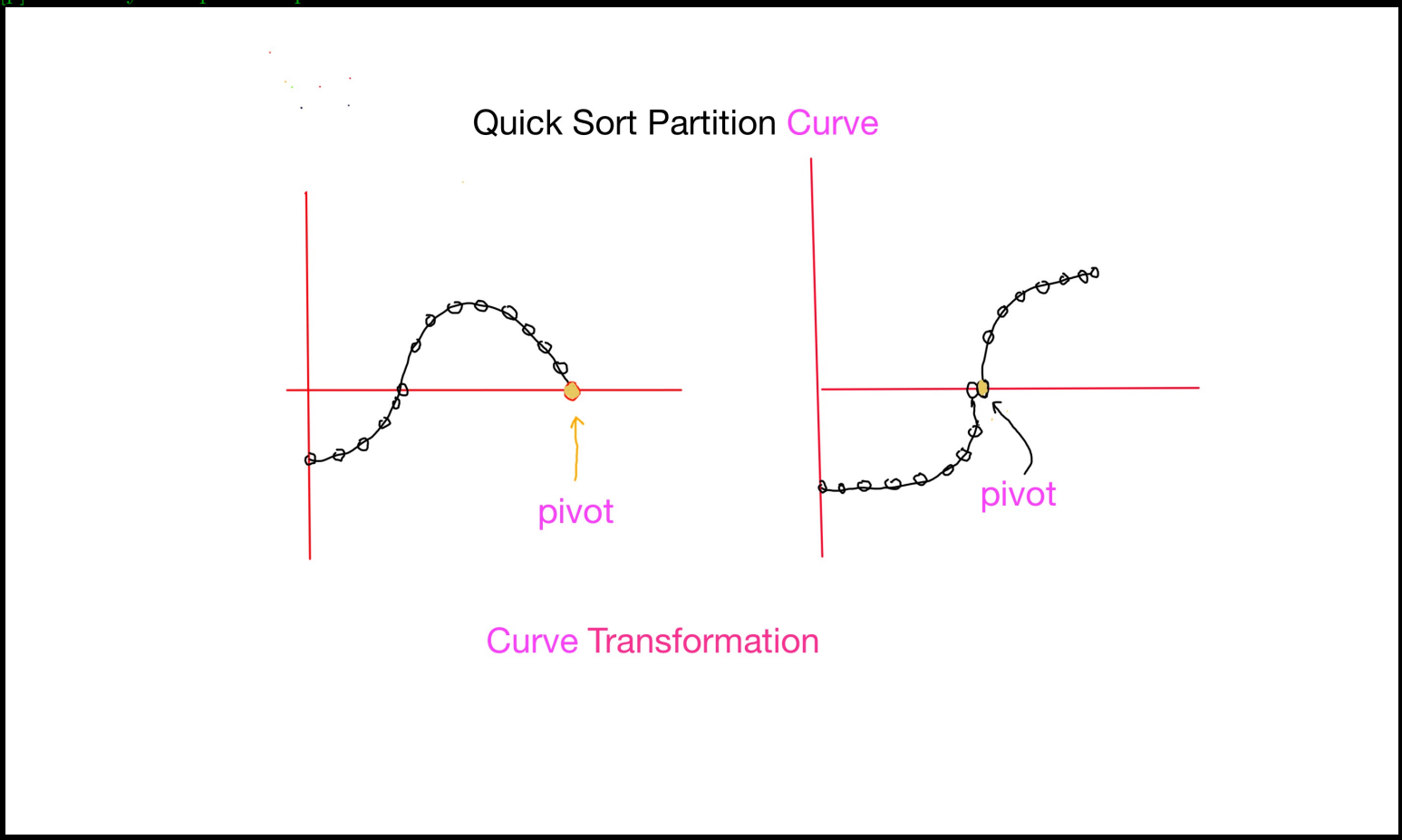


index p depends on $arr[i]$ <= pivot only

```
partition arr
    pivot = arr[len-1]
    p = 0
    for i=0 i<len i++
        if arr[i] <= pivot
            swap arr[i] arr[j]
            if p < len - 1
                p++
return p
```

31.1 Important test cases

$[1, 5, 3, 2, 2] \Rightarrow [1, 2, 2, 5, 3]$ pivot index $= 2$, it is NOT 1
arr[p] will always compare the pivot at the end



32 Print All Prime from 2 to N

```
prime n // print all prime from 2 to n
    list.add 2
    for k=3 to n
        isPrime = true
        for(n : list)
            if k mod n == 0
                isPrime = false
                break
        if isPrime
            list.add k
allprime n // print n prime
    if n >= 1
        list.add 2, k = 1, i = 3
        while k < n
            isPrime = true
            for n : list
                if i mod n == 0
                    isPrime = false;
                    break;
            if(isPrime)
                list.add i
                k++
```

33 Check Prime Number

```
isPrime n
    if n == 2
        return true
    else
        for d=2 d*d <= n d++
            if n % d == 0
                return false
    return true
```

Why $d * d < n$

Proof. The main idea is we try to find the factor of n from 2 and up let $\alpha = \{d_1, d_2, \dots d_{k-1}\}$ If α has no factor of n, d_k might be the smallest factor of n. If d_k is the smallest factor of n then d_{k+1} must be equal or greater than d_k, imply

$$d_k d_k <= n$$

□

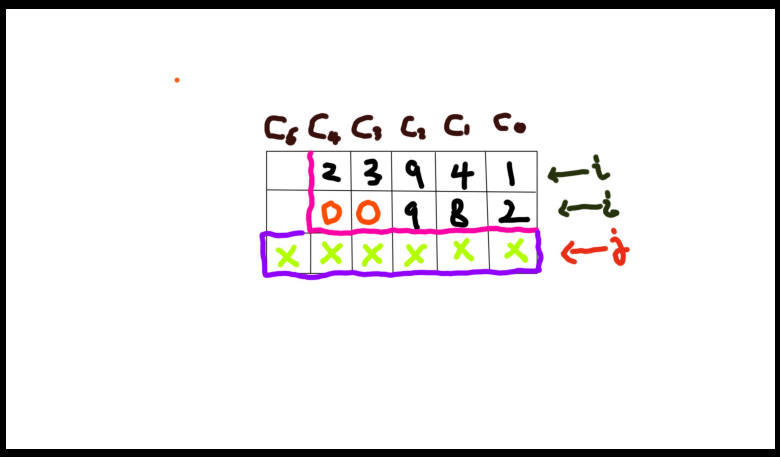
Example 2. *Let* $n = 101$ *and* $\{2, 3, \cdots, 9\}$ *are not the factor of* n *If the next integer* $d = 10$ *is the factor of* n *then we have following*

$$\begin{aligned} & \frac{n}{d} = m \\ \Rightarrow \quad & m \leq d \\ \Rightarrow \quad & m \leq 10 \end{aligned}$$

34 Add two long Integer

Given Two array which represent two integers int[] a1 = {1, 2, 3}, int[] a2 = {2, 3} add them up and return int[] a3

- make the two arrays in the same length
  a2 = {0, 2, 3} ⇒ a2 = {0, 2, 3}
- create new array with (length + 1)
- add up each element from Right to Left
- make sure the array[0] = carries
- NOTE the index of result array and index two arrays are different, i, j




```
int[] padArr(int len, int[] arr)
    // x x x
    // _ y y
    la = arr.length // la = 2
    int[] a = new int[len] // len = 3
    diff = len - la
    for int i=0 i<len i++
        a[i] = i < diff ? 0 : arr[i - diff]; // i=0, i=1=> 0, i=2 => 1
return a

int[] addLongInt int[] arr1, int[] arr2
    if arr1 != null && arr2 != null
        len1 = arr1.length
        len2 = arr2.length
        len = Math.max(len1, len2)
        int[] re = new int[len + 1]
        if len1 < len2
            int[] a1 = padArr(len2, arr1)
            c = 0
            for int i=0 i<len2 i++
                r = len2 - 1 - i
                s = c + (a1[r] + arr2[r]);
                re[len + 1 - 1 - i] = s % 10
                c = s / 10
            re[0] = c
        else
            int[] a2 = padArr(len1, arr2)
            c = 0
            for int i=0 i<len1 i++
                r = len1 - 1 - i
                s = c + (arr1[r] + a2[r])
                re[len + 1 - 1 - i] = s % 10
                c = s / 10
            re[0] = c
        return re
return null
```

35 Find all words from two dimensional grids filled with letters from a to z

1. count all the prefix words, e.g. dogs → dog, dogs

```
String append(String s, Char c)
    return s = s + (c + "")

boolean isWord(String s, Set<String> dict)
    return dict.conatins(s)

void find(Char[][] arr, int h, int w, String s, Set<String> dict)
    char c = arr[h][w]
    s = append(s, c)
    arr[h][w] = '0'
    if isWord dict s
        print s

for i in range(-1, 1)          // i = -1, 0, 1
    for j in range(-1, 1)        // j = -1, 0, 1
        if 0 <= h + i and h + i < arr.length and 0 <= w + j and w + j < arr[0].length
            if arr[h+i][w+j] != '0'                    // not visited yet
            find(arr, h + i, w + j, s, dict)

arr[h][w] = c


// print all words
for h in range(arr.length)
for w in range(arr[0].length)
    String s = ""
    find(arr, i, h, w, s, dict)
```

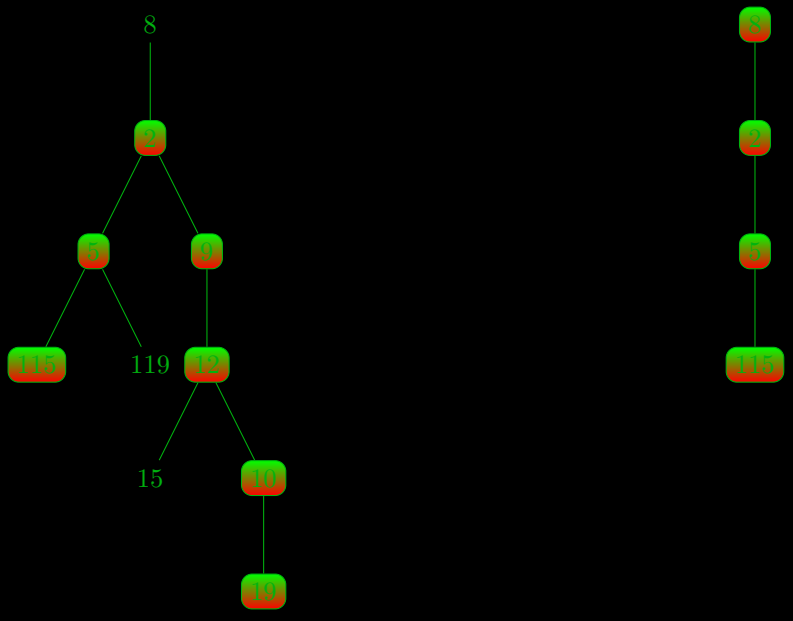
The Complexity would be $\mathcal{O}(\log mn^2)$ The Complexity of find function is $\mathcal{O}(\log mn)$

36 Find a diameter of a binary tree

A diameter of binary tree is defined as the longest path between two leaf nodes in a binary tree. The longtest path of the following binary tree are { 115, 5, 2, 9, 12, 10, 19 } and { 8, 2, 5, 115 }

36.1 Binary Tree Defintion

Find the maximum height of left subtree
Find the maximum height of right subtree



```
int max(Node root)
    if root != null
        l = max(root.left)
        r = max(root.right)
        return Math.max(l, r) + 1
    else
        return 0

int diameter(Node root)
    if root != null
        l = max(root.left)     // max height of left subtree
        r = max(root.right)    // max height of right subtree
        dl = diameter(root.left)  // left subtree
        dr = diameter(root.right) // right subtree
        return Math.max(l + r + 1, Math.max(dl, dr))
    else
        return 0
```

37 Maximum Continuous Sum

Not all elements are negative
Given a list of not all negative integers, find the maximum continuous sum.
{ 2 }, { -2, 3 }, {-1, 3, -4 }, {-1, 2, 3, -6, 9 }

```
Integer maxContinuousSum(Integer[] arr)
  Integer m1 = Integer.MIN_VALUE
  Integer curr = m1
  if arr != null
  for(Integer n : arr)
    if curr < 0
     curr = 0
    curr += n

    if(curr > m1)
      m1 = curr

  return m1
```

If all elements are negative, then we can use simple algorithm to find the maximum

```
Integer maxSum(Integer[] arr)
  Integer m2 = Integer.MIN_VALUE
  Integer curr = m2
  for Integer n : arr
    if(n > m2)
      m2 = n
  return m2
```

We can combine two algorithms to ONE algorithms to find the continuous maximum sum of all elements

38 Find all multiplication of all elements excluding the current element.

$\{2, 3\} \rightarrow \{1 \cdot 3, 1 \cdot 2\}$
$\{2, 3, 4\} \rightarrow \{3 \cdot 4, 2 \cdot 4, 2 \cdot 3\}$
We need two accumulators: `acc1[], acc2[]` to accumulate products.

$$\begin{vmatrix} 4 & 4 & 4 \\ 3 & 3 & 3 \\ 2 & 2 & 2 \end{vmatrix}$$

```
Integer[] allMultiplication(Integer[] arr)
  Integer[] arr1 = null
  Integer[] arr2 = null

  // at least two elements
  if arr != null && arr.length > 1
      int len = arr.length
      arr1 = new Integer[len]
      arr2 = new Integer[len]
      arr1[0] = 1
      arr2[len-1] = 1

      for i=0; i+1 <len; i++
          arr1[i+1] = arr1[i] * arr[i]
          // inverse the index
          arr2[len - 1 - (i+1)] = arr2[len - 1 - i] * arr[len - 1 - i]

      for i=0; i<len; i++
          arr1[i] = arr1[i] * arr2[i];

      return arr1
```

39 Merge two sorted single linked lists

- The problem is similar to merge two sorted arrays
  - But it is trick to add node when the curr is null or NOT null
- When the current node is null and curr node is NOT null
- If the current node is null, create a new node and assign to curr
- If the current node is NOT null, assign new node to the current node and move current to next

```
LNode mergeSortedLinkedList SLL s1, SLL s2
LNode curr = head = null
LNode c1 = s1.head
LNode c2 = s2.head
while c1 != null || c2 != null
  // at the end of s1
  if c1 == null
    if curr == null
      curr = head = new LNode(c2.data)
    else
      curr.next = new LNode(c2.data)
      curr = curr.next
    c2 = c2.next
  // at the end of c1
  else if c2 == null
    if curr == null
      curr = head = new LNode(c1.data)
    else
      curr.next = new LNode(c1.data)
      curr = curr.next
    c1 = c1.next
  else
    // c1 and c2 both
    if c1.data < c2.data
      if curr == null
        curr = head = new LNode(c1.data)
      else
        curr.next = new LNode(c1.data)
        curr = curr.next
      c1 = c1.next
    else
      if curr == null
        curr = head = new LNode(c2.data)
      else
        curr.next = new LNode(c2.data)
        curr = curr.next
      c2 = c2.next
 return head
```

Better trick to merge two sorted lists
Add a dummy node to the curr list. Check null is unnecessary when a new node is added.

```
  public static Node mergeSortedLinkedLists Node h1, Node h2
      Node c1 = h1
      Node c2 = h2
      Node curr = null
      Node head = null
      // add a dummy node
      curr = head = new Node(0)
      while c1 != null || c2 != null
          if c1 == null
              while c2 != null
                 curr.next = new Node(c2.data)
                 curr = curr.next
                 c2 = c2.next
```

```
    else if c2 == null
        while c1 != null
           curr.next = new Node(c1.data)
           curr = curr.next
           c1 = c1.next

    else
        if c1.data < c2.data
            curr.next = new Node(c1.data)
            c1 = c1.next
        else
            curr.next = new Node(c2.data)
            c2 = c2.next

        curr = curr.next


head = head.next
return head
```

40 Merge Two Sorted lists

Given two sorted lists, merge two sorted list

```
    mergeTwoSortedLists s1, s2
        i = 0 j = 0
        len1 = s1.size()
        len2 = s2.size()
        ls = new ArrayList<>()
        while i < len1 || j < len2
            if i >= len1
                    ls.add(j)
                    j++
                else if j >= len2
                    ls.add(i)
                    i++
                else
                    if s1.get(i) < s2.get(j)
                        ls.add(s1.get(i))
                        i++
                    else
                        ls.add(s2.get(j))
                        j++
        return ls
```

41 Longest Increasing Subsequence

Given a list $\{1, 2, 9, 0, 1, 10, 2, 4\}$
The longest $\Rightarrow \{0, 1, 2, 9, 10\}$

41.1 Dynamic Programming $\mathcal{O}(n)$

41.2 Recursion Version, Slow $\mathcal{O}(n)$

41.3 Brute Force $\mathcal{O}(n^2)$

1. Go through each suffix list

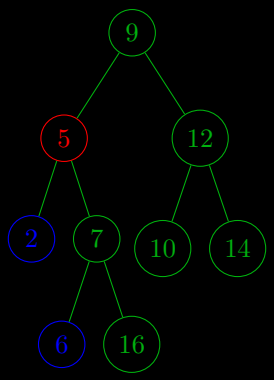
```
/**
 * Use brute force to find the longest increasing subsequence.
 *
 */
int longestIncreasingSubsequence2 Integer[] arr
   len = arr.length
   max = 0
   listMax = new Integer[len]

   for i=0; i<len; i++
       initOne(listMax)
       for j=i j+1 <len j++
       if arr[j] < arr[j+1]
           listMax[j + 1] = listMax[j] + 1

       for k=0 k<len k++
           if listMax[k] > max
               max = listMax[k]

   return max
```

				$\rightarrow i$				
1	2	9	0	1	10	2	4	
	2	9	0	1	10	2	4	
		9	0	1	10	2	4	
			0	1	10	2	4	$\downarrow$
				1	10	2	4	j
					10	2	4	
						2	4	
							4	

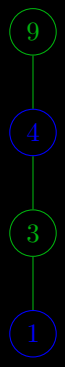
42 Lease common ancestor

Given a Binary Tree and two nodes, find the lease common ancestor node.

Example 3. *Given node 2 and node 6, the lease common ancestor is node 5*



Example 4. *Given node 1 and node 4, the lease common ancestor is node 4*



43 Number of Different Integers in a String

You are given a string word that consists of digits and lowcase English letters. You will replace every non-digit character with a space. For example, "a123bc34d8ef34" will become "123 34 8 34". Notice that you are left with some integers that are separated by at lease one space: "123", "34", "8", "34". Return the number of *different* integers after performing the replacement operations on **word** Two integers are different if their decimal representations without any leading zeros are different.

Example 5. *Input: word = "12ab33a033"*
Outpt: 3
Explanation: *After all non-digits are replaced with space, word becomes "12 33 033". "33" and "033" are same number so the total different numbers are 2.*

43.1 Java code

```
String s = "a12bc012e33k";
String[] arr = s.split("[a-z]+");
Set<Integer> set = new HashSet<Integer>();

for(String e :  arr) {
  // It might contain empty string
  if(len(e.trim()) > 0){
    set.add(Integer.parseInt(e));
  }
}

return set.size();
```

44 Geometry Area Problem

https://open.kattis.com
Given an input file contain an Integer n in the first line, and the following lines contain vertices/vertexes form a *simple polygon*.
Each line contain only one vertex (x, y) where $x, y \geq 0 \wedge x, y \in \mathbb{N}$. The input is terminated by a case beginning with 0.

- Try to use Convex Hull /technic/technics/technique, it seems it does not work.
- Find all the valid segments, use vertex and segments to find all the valid triangles.. how?