1 Algorithms are important for interview

2 SQL

2.1 Innter join, outer join

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

3 Coin Change Problem

3.1 This is fun question

Example 1. *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

3.2 Find the minimum number of coins are added up to the S

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

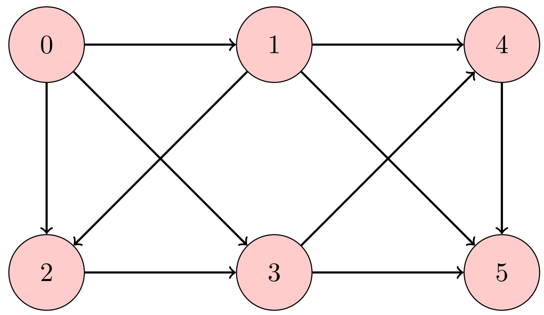
```
inorder node r
    stack
    while r not null or stack is not empty
        if r not null
            stack.push r
            r = r.left
        else
            n = stack.pop
            print n.data
            r = r.right
```

4.3 postorder

```
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
```

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

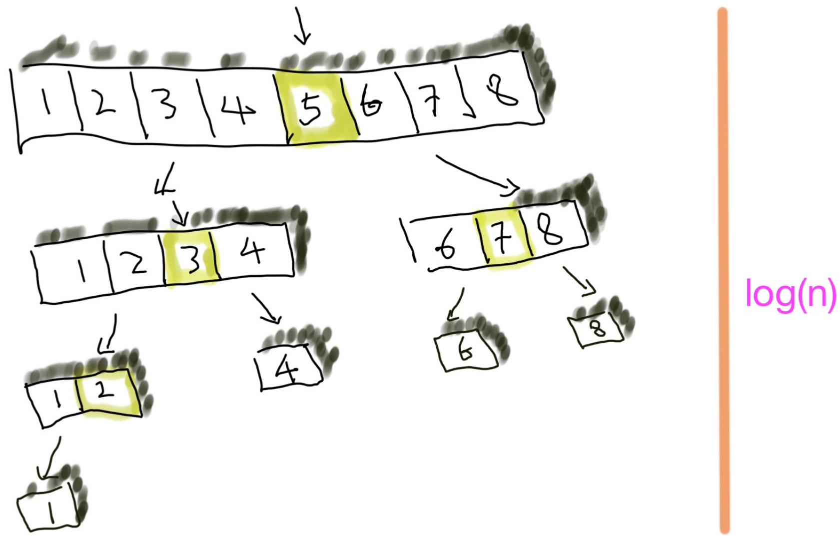
```
// connect each level
connectlevel node r
    queue q1 q2
    while r1 is not empty and r2 is not empty
        prev = null
        while r1 is not empty
            curr = q1.dequeue
            if prev is not null
                prev.next = curr
            prev = curr

            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
```


1 2 3 4 5 6 7 8
1 2 3 4
6 7 8
1 2
4
6
8
1
log(n)

8 Rotate square 2d array

```
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++
```

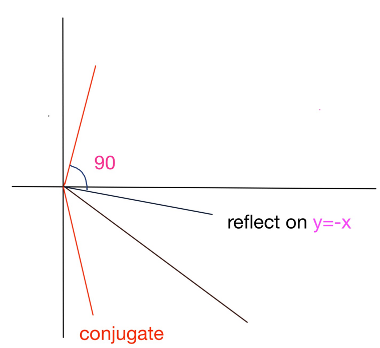
e.g. If len is 3, then $len/2 = 1$ then $k < len/2$ is missing the center element from the array. There is no problem for rotation, but center element is missing if spiral is printed

```
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 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: $(x, y) \circ (y, -x) = 0$

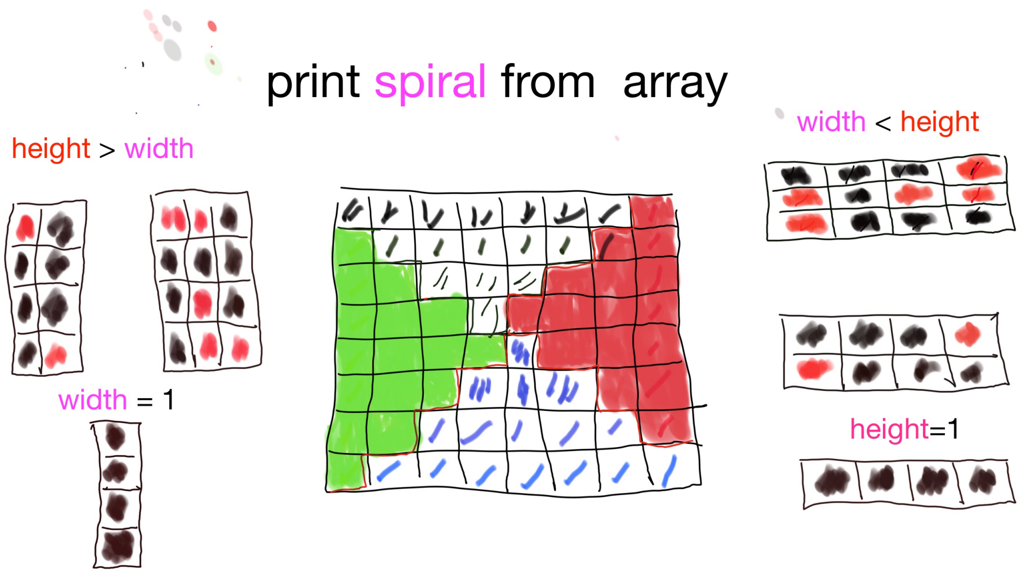
```
void rotate90degree(int[][] arr){
    int len = arr.length;
    // reflect on y = x
    for(int i=0; i<len; i++){
        for(int j=i; j<len; j++){
            int tmp = arr[i][j];
            arr[i][j] = arr[j][i];
            arr[j][i] = tmp;
        }
    }
    // reflect on x = len/2
    for(int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



10 Serialize Binary Tree

```
serializeGeneralTree(Node r)
    if r is not null
        write(r.data);
        for(Node n : r.list)
            serializeGeneralTree(n)
        write("# ");

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

    for(String s : ar)
        list.add(s.strim());
return list;

deserializeGeneralTree(List<String> list)
    Stack<String> 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();
```

11 Print spiral shape from 2d array



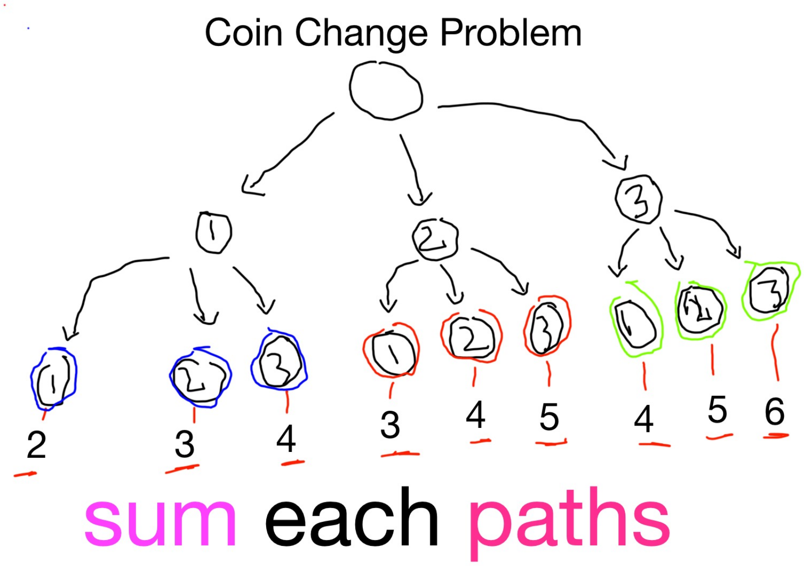
- If (width - 2*k == 1) or (height - 2*k == 1), then we can terminate the code after that.
- Otherwise, min(width, height) is even, then one row or one column DOES NOT exist.
- If the array is one row or one column then it depends on **width** < **height** or **width** > **height**
- <= is import here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

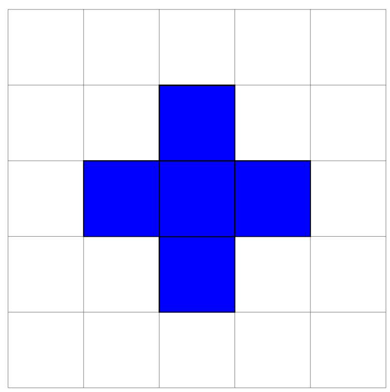
12 Coin Change problem

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



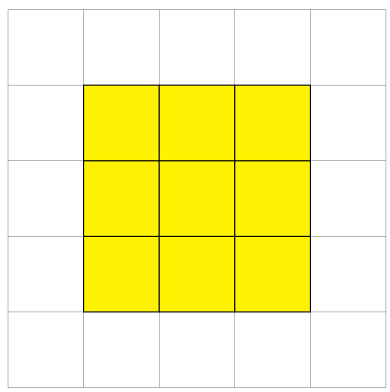
13 Connected island problem four directions

```
// four directories
count arr2d h w height width
    if arr2d[h][w] == 1
        arr2d[h][w] = 2
        int n1, n2, n3, n4
        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
```



14 Connected island problem Eight directions including diagonal

```
// 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] is 1
        arr2d[h][w] = 1
        for(int hh=0; hh<=1; hh++)
            for(int ww=0; ww<=1; ww++)
                if h + hh >= 0 && h + hh < height ||
                   w + ww >= 0 && w + ww < width
                   sum +=count8 arr2d h+hh w+ww height width
    return sum
```



15 Sudoku Solver

15.1 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
```

16 Eight Queen

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

16.1 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`

17 Permutation

1. Networking
   - Three ways hands shake
   - slide window inside the package
   - DNS, resolve domain name, IP
   - TCP, UDP, HTTPS
2. Design
   - load balance
   - DB replication
   - Sub-Pub
   - command pattern
   - visit pattern
   - Singleton,
   - Double-checked locking
   - message queue
   - consume and producer
3. 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
     - If the keys keep the same relative orders after keys are sorted then the sort algorithm is stable. Otherwise it is unstable.

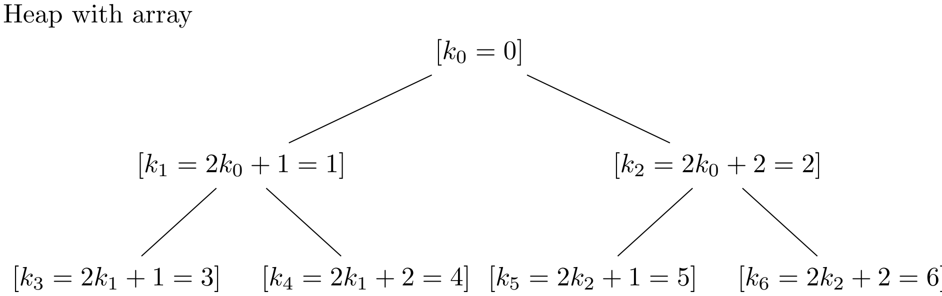
       Sort the second coordinates

       (4,1) (2,3) (4,3)

Sort the first coordinates [stable sort]

(2,3) (4,1) (4,3)

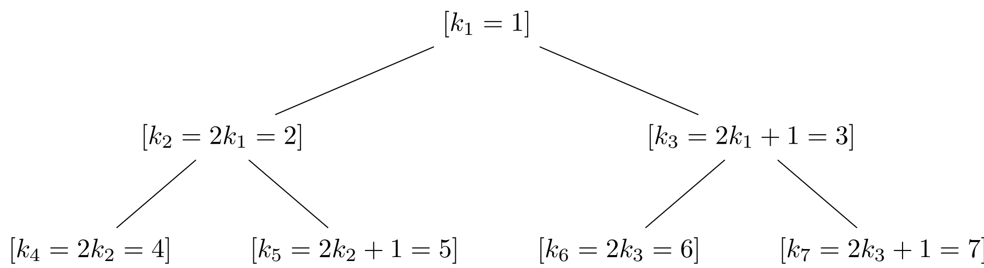
- Find the Kth smaller element in a given unsorted array in $\mathcal{O}(n)$
- Merge Sort,
  * The average and worst runtime is $\mathcal{O}(n \log n)$
  * Stable sort
- Max distance for $j - i$ given $arr[j] > arr[i]$
- Single linked list
  * reverse, iteration, recursion
  * remove,
  * insert to sorted list
  * clone list
  * check circular linkedlist
- Double linkedlist
  * remove node
  * insert node
  * append node
- Eight queen problem
- Sudoku Solver problem
- Connected island
- Implement heap with array
  * Heap with array

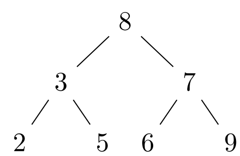


- Serialize Binary Tree
  * Use technic similar to the implementation of Heap with array level nodes

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

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






```
// [0 : 8] [1 : 3] [2 : 7] [3 : 2] [4 : 5] [5 : 6] [6 : 9]
level_serial(Node root, int index, String file){
    if(root != null){
        try{
        BufferedWriter bf = new BufferedReader(new FileWriter(file));
            bf.write(index + ":" + root.data + "\n");
            level_serial(root.left, 2*index+1, file);
            level_serial(root.right, 2*index+2, file);
        }catch(IOException e){
            e.printStackTrack();
        }

    }
}

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

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

- Rotate square array
- Serialize general tree
- Tree Traveral
  - preorder
    - pretty print
    - use it to serialize Binary Tree, postorder can be deserialized BT.
  - inorder
    - Check if a Binary Tree is Binary Search Tree
  - postorder
    - deserialize
  - levelorder
    - use two queues
    - one queue for odd level
    - other queue for even level
  - print level without queues
    - 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 index){
    if(root != null){
        mpa.put(index, root.data);
        print(root.data);
        level(root.left, 2*index)
        level(root.right, 2*index + 1)
    }
}

void printLevel(Map<Integer, Integer> map){
    int count = 0;
    int size = 0;
```

```
        while(size < map.size()){
            int m = (int)Math.pow(2, k);
            for(int i=1; i<=m; i++){
                Integer n = map.get(count + i);
                if(n != null){
                    print(n.data);
                    size++;
                }
            }
            count += m;
        }
    }

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

∗ iteration preorder, inorder, postorder, levelorder

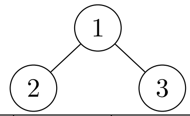
∗ iteration preorder

- · 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
  - xxx



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);
```

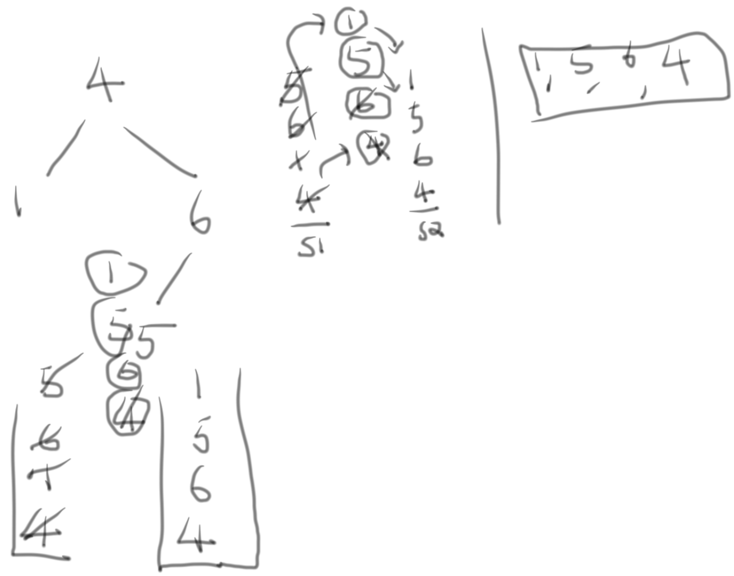


Figure 1: Iteration postorder traveral with two stacks

```
                }

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

level order sequence with two stacks

- Use two stacks → print Binary Tree in sequency order

```
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)
            }
        }
    }
}
```

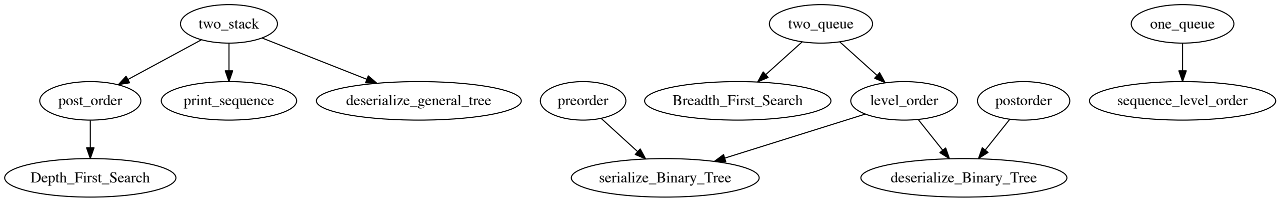


Figure 2: two_stack.gv

- 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 $\mathcal{O}(n)$
- Graphic Problem
  - Graph
  - find a path from two nodes (Use Breadth First Search)
    (a) Check if the n1 and n2 are equal
    (b) if n1 and n2 are equal. we are done!
    (c) if n1 and n2 are not equal.
    (d) add n1 to a list
    (e) 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
    * move to second position and test $[a - z]$ recursively
    * 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

4. Lease Recent Used[LRU]

5. Implements HashMap

- Context Switch invoke switching reg, stack pointer, program counter

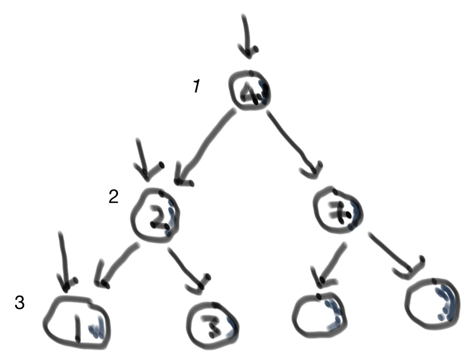
- Synchronize LinkedList
  - *add*() and *delete*()
  - Use two locks
    * if the node is a head, lock it and delete it, easy!
    * if the node is not head, then lock the previous and current nodes
    * if a node is found, delete it
  - Why it works?
    * If the current node is locked, you can delete 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

18 Binary Tree Problems

18.1 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

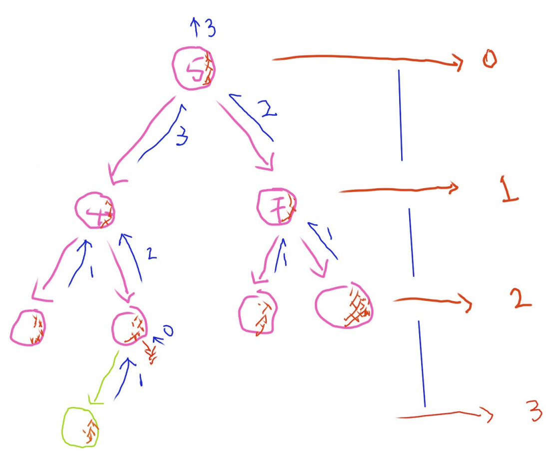



```
class Node
    Int data
    Node left
    Node right
    Node(Int data)
        this.data = data

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)
```

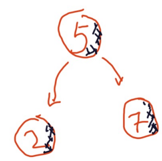
18.2 Maximum path in a Binary Tree

```
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
```



18.3 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
```

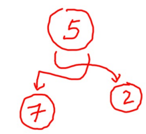


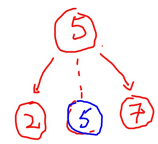
2 → 5 → 7 Inorder

5 → 2 → 7 preorder

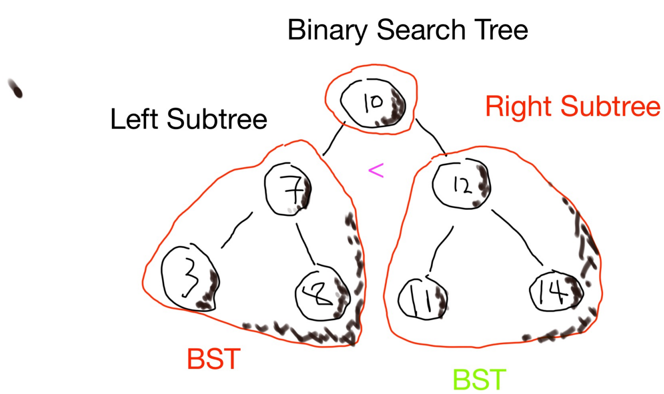
2 → 7 → 5 postorder

7 -> 2 -> 5 ?





18.4 Check whether a Binary Tree is BST



18.4.1 Binary Search Tree Definition

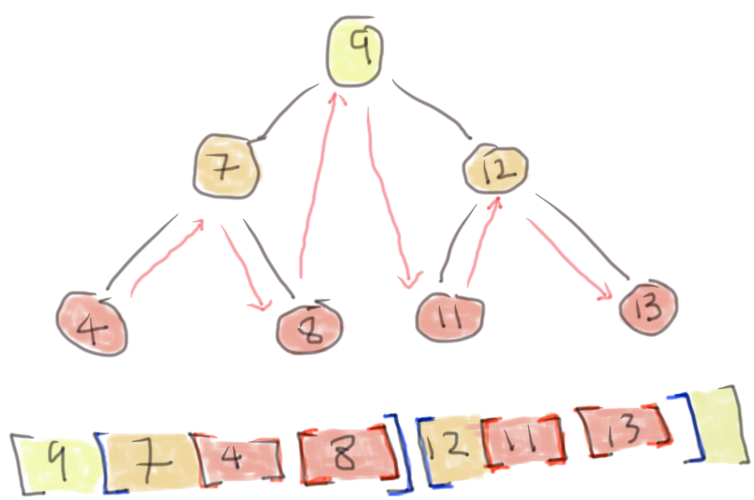
- Empty tree is BST
- Left substree is BST
- Right substree is BST
- The maximum left subtree < parent node
- The minimum right substree > parent node
- The whole Binary Tree is BST

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

```
// 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
        // the cond is NOT true, it is not a BST
        if root != null && !(leftMax root < curr.data && rightMin > curr.data)
            return false
    return true
```

18.5 Use Recursive Algorithm, following the picture



19 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.

19.1 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
```

- **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

19.2 Quick Sort Recursion

choose the pivot and partition array to two parts

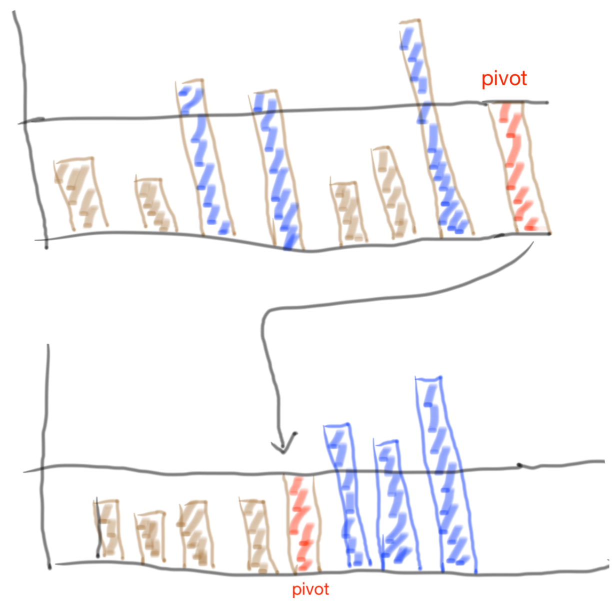
all the elem are less than pivot are on left part excluding the pivot

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

19.2.1 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
```

19.2.2 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
```

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

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

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

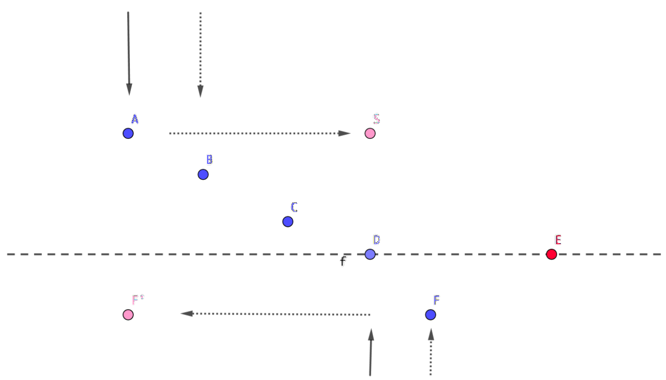
19.4 Quick Sort Iteration

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)
```

20 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

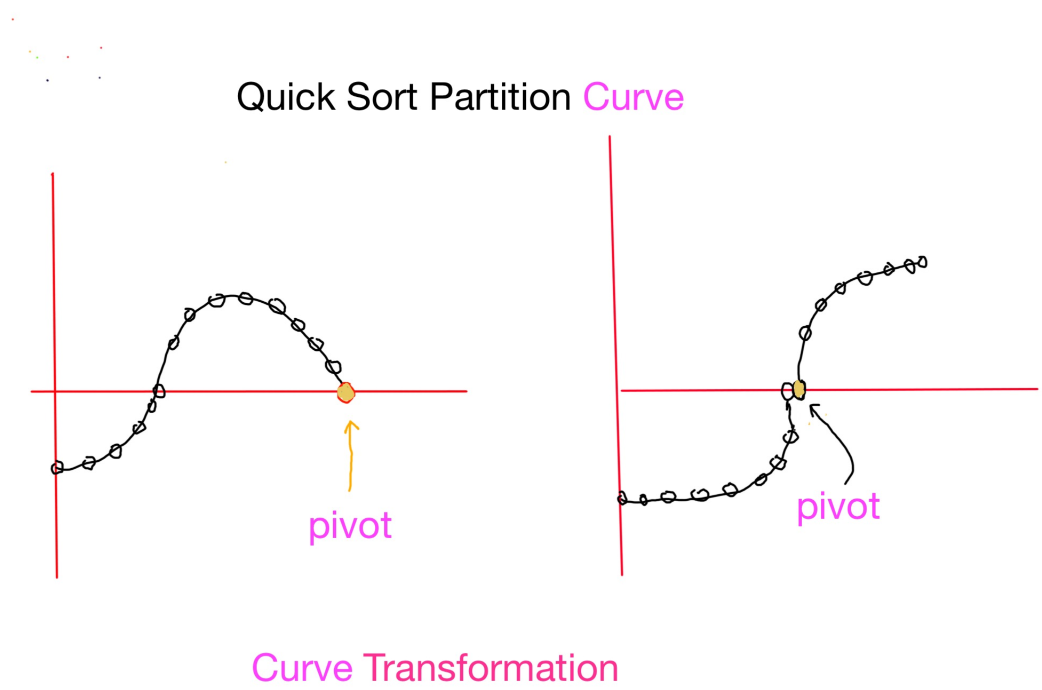


index p depends on $arr[i]$ $<=$ pivot only

```
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
```

20.1 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



21 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++
```

22 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 $\alpha = \{d_1, d_2, \ldots d_{k-1}\}$ 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

$$d_k d_k <= n$$

$\square$

Example 3. *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*

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