[https://w.amazon.com/skins/amazon/images/holidays/thanksgiving64w62h.png](https://w.amazon.com/bin/view/Main)

Top of Form



Bottom of Form

**Interview**

Last modified 3 months ago by [puneeb](https://w.amazon.com/bin/view/XWiki/puneeb).

|  |
| --- |
| **Contents**   * [1 Technical Interview Questions](https://w.amazon.com/bin/view/User:Arman/Interview#Technical_Interview_Questions)   + [~~1.1 Sub Trees~~](https://w.amazon.com/bin/view/User:Arman/Interview#Sub_Trees)   + [~~1.2 ASCII~~](https://w.amazon.com/bin/view/User:Arman/Interview#ASCII)   + [~~1.3 Reverse Linked List~~](https://w.amazon.com/bin/view/User:Arman/Interview#Reverse_Linked_List)   + [~~1.4 Reverse String~~](https://w.amazon.com/bin/view/User:Arman/Interview#Reverse_String)   + [~~1.5 Hamming Distance~~](https://w.amazon.com/bin/view/User:Arman/Interview#Hamming_Distance)   + [~~1.6 Add Digits~~](https://w.amazon.com/bin/view/User:Arman/Interview#Add_Digits)   + [~~1.7 Max Consecutive Ones~~](https://w.amazon.com/bin/view/User:Arman/Interview#Max_Consecutive_Ones)   + [~~1.8 Two Sum~~](https://w.amazon.com/bin/view/User:Arman/Interview#Two_Sum)   + [~~1.9 Converging Linked Lists~~](https://w.amazon.com/bin/view/User:Arman/Interview#Converging_Linked_Lists)   + [~~1.10 Longest Unique Substring~~](https://w.amazon.com/bin/view/User:Arman/Interview#Longest_Unique_Substring)   + [~~1.11 Power Function~~](https://w.amazon.com/bin/view/User:Arman/Interview#Power_Function)   + [~~1.12 Making Change~~](https://w.amazon.com/bin/view/User:Arman/Interview#Making_Change)   + [~~1.13 Container With Most Water~~](https://w.amazon.com/bin/view/User:Arman/Interview#Container_With_Most_Water)   + [1.14 Maximum Sub Array](https://w.amazon.com/bin/view/User:Arman/Interview#Maximum_Sub_Array)   + [~~1.15 Zig-Zag Tree Traversal~~](https://w.amazon.com/bin/view/User:Arman/Interview#Zig-Zag_Tree_Traversal)   + [~~1.16 Symmetric Tree~~](https://w.amazon.com/bin/view/User:Arman/Interview#Symmetric_Tree)   + [~~1.17 Stairs~~](https://w.amazon.com/bin/view/User:Arman/Interview#Stairs)   + [~~1.18 Curly braces~~](https://w.amazon.com/bin/view/User:Arman/Interview#Curly_braces)   + [~~1.19 BST to Array and back~~](https://w.amazon.com/bin/view/User:Arman/Interview#BST_to_Array_and_back)   + [~~1.20 Island Perimeter~~](https://w.amazon.com/bin/view/User:Arman/Interview#Island_Perimeter)   + [~~1.21 Candy~~](https://w.amazon.com/bin/view/User:Arman/Interview#Candy)   + [1.22 Binary Search Tree](https://w.amazon.com/bin/view/User:Arman/Interview#Binary_Search_Tree)   + [~~1.23 Hash Table~~](https://w.amazon.com/bin/view/User:Arman/Interview#Hash_Table)   + [~~1.24 Min Stack~~](https://w.amazon.com/bin/view/User:Arman/Interview#Min_Stack)   + [~~1.25 Unique Characters~~](https://w.amazon.com/bin/view/User:Arman/Interview#Unique_Characters)   + [~~1.26 Anagrams~~](https://w.amazon.com/bin/view/User:Arman/Interview#Anagrams)   + [~~1.27 Merge two sorted lists~~](https://w.amazon.com/bin/view/User:Arman/Interview#Merge_two_sorted_lists)   + [~~1.28 Subsets~~](https://w.amazon.com/bin/view/User:Arman/Interview#Subsets)   + [~~1.29 String Permutations~~](https://w.amazon.com/bin/view/User:Arman/Interview#String_Permutations)   + [~~1.30 Array Intersection~~](https://w.amazon.com/bin/view/User:Arman/Interview#Array_Intersection)   + [1.31 Sorting](https://w.amazon.com/bin/view/User:Arman/Interview#Sorting)   + [~~1.32 Missing Number~~](https://w.amazon.com/bin/view/User:Arman/Interview#Missing_Number)   + [~~1.33 Missing Spaces~~](https://w.amazon.com/bin/view/User:Arman/Interview#Missing_Spaces)   + [1.34 Nth smallest element in a binary search tree](https://w.amazon.com/bin/view/User:Arman/Interview#Nth_smallest_element_in_a_binary_search_tree)   + [~~1.35 Implement a Queue~~](https://w.amazon.com/bin/view/User:Arman/Interview#Implement_a_Queue)   + [1.36 LinkedList of Nodes in a Binary Tree](https://w.amazon.com/bin/view/User:Arman/Interview#LinkedList_of_Nodes_in_a_Binary_Tree)   + [1.37 Inorder Successor in Binary Search Tree](https://w.amazon.com/bin/view/User:Arman/Interview#Inorder_Successor_in_Binary_Search_Tree)   + [~~1.38 Find min element in rotated, sorted array~~](https://w.amazon.com/bin/view/User:Arman/Interview#Find_min_element_in_rotated.2C_sorted_array)   + [~~1.39 Palindrome~~](https://w.amazon.com/bin/view/User:Arman/Interview#Palindrome)   + [~~1.40 Heap~~](https://w.amazon.com/bin/view/User:Arman/Interview#Heap)   + [~~1.41 Median Value~~](https://w.amazon.com/bin/view/User:Arman/Interview#Median_Value)   + [~~1.42 Calculate Angle~~](https://w.amazon.com/bin/view/User:Arman/Interview#Calculate_Angle)   + [~~1.43 Implement a LRU Cache~~](https://w.amazon.com/bin/view/User:Arman/Interview#Implement_a_LRU_Cache)   + [~~1.44 Rotate Array~~](https://w.amazon.com/bin/view/User:Arman/Interview#Rotate_Array)   + [~~1.45 Rotate 2D Array~~](https://w.amazon.com/bin/view/User:Arman/Interview#Rotate_2D_Array)   + [~~1.46 Implement a Queue using two stacks~~](https://w.amazon.com/bin/view/User:Arman/Interview#Implement_a_Queue_using_two_stacks)   + [~~1.47 Balanced Binary Tree~~](https://w.amazon.com/bin/view/User:Arman/Interview#Balanced_Binary_Tree)   + [~~1.48 Lowest Common Ancestor~~](https://w.amazon.com/bin/view/User:Arman/Interview#Lowest_Common_Ancestor)   + [~~1.49 Fibonacci~~](https://w.amazon.com/bin/view/User:Arman/Interview#Fibonacci)   + [~~1.50 First Unique Character~~](https://w.amazon.com/bin/view/User:Arman/Interview#First_Unique_Character)   + [~~1.51 Largest number in BST which is less than or equal to N~~](https://w.amazon.com/bin/view/User:Arman/Interview#Largest_number_in_BST_which_is_less_than_or_equal_to_N) |

**Technical Interview Questions**

**Making Change**

Question: For US currency, there are six coins that are in use today: 1¢, 5¢, 10¢, 25¢, 50¢, 1$. Write a function that returns the number of possible combinations of change that can be made from n ¢.

Example:

input: 7 output: 2 (7p, 1n 2p)

input: 10 output: 4 (1d, 10p, 2n, 1n 5p)

input: 15 output: 6 (1d 1n, 1d 5p, 3n, 2n 5p, 1n 10p, 15p)

**public** **static** int combinations(int n) {  
    **return** combinations(n,100);  
}  
   
**public** **static** int combinations(int n,int m) {  
    **if**(n < 0)  
        **return** 0;  
   
    **if**(n == 0)  
        **return** 1;  
   
    int combinations = 0;  
   
    **if**(m == 100)  
        combinations+= combinations(n-100,100);  
    **if**(m >= 50)  
        combinations+= combinations(n-50,50);  
    **if**(m >= 25)  
        combinations+= combinations(n-25,25);  
    **if**(m >= 10)  
        combinations+= combinations(n-10,10);  
    **if**(m >= 5)  
        combinations+= combinations(n-5,5);  
   
    combinations+= combinations(n-1,1);  
   
    **return** combinations;  
}

**Container With Most Water**

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.

public int maxArea(int[] height) { int max = 0; int i = 0; int j = height.length - 1; while (i < j) { int k = Math.min(height[i], height[j]) \* (j - i); if(k > max) { max = k; } if(height[i]>=height[j]){ j--; } else { i++; } } return max; }

**Maximum Sub Array**

Question: Given an array with at least one positive integer, find the contiguous sub-array with the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the contiguous sub-array with the largest sum is 4, −1, 2, 1, with sum 6.

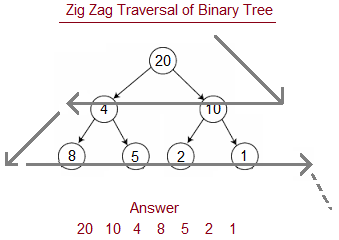
**public** **static** int [] maxSubArray (int [] array) {  
    int starting\_index = 0;  
    int length = 0;  
    int sum = 0;  
   
    int t\_starting\_index = 0;  
    int t\_length = 0;  
    int t\_sum = 0;  
   
    **for** (int i = 0; i < array.length; i++) {  
        **if**(t\_sum + array[i] > 0) {  
            t\_sum += array[i];  
            t\_length++;  
   
            **if**(t\_sum > sum) {  
                sum = t\_sum;  
                length = t\_length;  
                starting\_index = t\_starting\_index;  
            }  
        } **else** {  
            t\_sum = 0;  
            t\_length = 0;  
            t\_starting\_index = i + 1;  
        }  
    }  
   
    int [] sub = **new** int[length];  
   
    System.arraycopy(array, starting\_index, sub, 0, length);  
   
    **return** sub;  
}

Solution to find the max sum:

**public** **static** int maxSubArray(int [] array) {  
    int max\_ending\_here = 0;  
    int max\_so\_far = 0;  
   
    **for**(int i: array) {  
        max\_ending\_here = Math.max(0, max\_ending\_here + i);  
        max\_so\_far = Math.max(max\_so\_far, max\_ending\_here);  
    }  
   
    **return** max\_so\_far;  
}

**Zig-Zag Tree Traversal**

Question: Print out items of a Binary Tree in a zig-zag fashion.



**public** **static** void zigZag (Node root) {  
    Stack<Node> s1 = **new** Stack<Node>();  
    Stack<Node> s2 = **new** Stack<Node>();  
   
    s1.push(root);  
   
    **while**(!s1.isEmpty() || !s2.isEmpty()) {  
        **while**(!s1.isEmpty()) {  
            Node n = s1.pop();  
   
            System.out.println(n.data);  
   
            **if**(n.right != **null**)  
                s2.push(n.right);  
   
            **if**(n.left != **null**)  
                s2.push(n.left);  
        }  
   
        **while**(!s2.isEmpty()) {  
            Node n = s2.pop();  
   
            System.out.println(n.data);  
   
            **if**(n.left != **null**)  
                s1.push(n.left);  
   
            **if**(n.right != **null**)  
                s1.push(n.right);  
        }  
    }  
}

**Symmetric Tree**

Write a program to check if the given binary tree is symmetric tree or not. A symmetric tree is defined as a tree which is mirror image of itself about the root node. For example, following tree is a symmetric tree.

**public** **static** boolean isMirror(Node n1, Node n2) {  
        **if**(n1 == **null** && n2 == **null**)  
            **return** **true**;  
   
        **if**(n1 == **null** || n2 == **null**)  
            **return** **false**;  
   
        **if**(n1.data != n2.data)  
            **return** **false**;  
   
        **return** isMirror(n1.left,n2.right) && isMirror(n1.right, n2.left);  
    }

**Stairs**

Question: A man is walking up a set of stairs. He can either take 1 or 2 steps at a time. Given n number of stairs, find out how many combinations of steps he can take to reach the top of the stairs.

**public** **static** int combinations(int stairs) {  
    **if**(stairs == 0)  
        **return** 0;  
   
    int i = 0;  
    int j = 1;  
   
    **for**(int k = 0; k <= stairs; k++) {  
        int temp = j;  
        j+= i;  
        i = temp;  
    }  
    **return** i;  
}

**Curly braces**

Question: Curly braces can be used in programming to provide scope-limit. Write a function to print all valid( properly opened and closed) combinations of n-pairs of curly braces.

Example:

input: 1 output: {}

input: 2 output: {}{}, Legacy Wiki template overrides

input: 3 output: {}{}{}, {}Legacy Wiki template overrides, Legacy Wiki template overrides{}, [[:Template:}{]], {{{}}}

input: 4 output: {}{}{}{}, {}{}Legacy Wiki template overrides, {}Legacy Wiki template overrides{}, {}[[:Template:}{]], {}{{{}}}, Legacy Wiki template overrides{}{}, Legacy Wiki template overridesLegacy Wiki template overrides, [[:Template:}{]]{}, [[:Template:}{}]], [[:Template:}Legacy Wiki template overrides]] {{{}}}{}, {Legacy Wiki template overrides{}}, {[[:Template:}{}]], [Template:Legacy Wiki template overrides](https://w.amazon.com/bin/view/Template:Legacy_Wiki_template_overrides)

**public** **static** void printBraces(int n) {  
        printBraces(**new** String(), n, n);  
    }  
   
    **public** **static** void printBraces(String str, int start, int end) {  
        **if** (start == 0 && end == 0) {  
            System.out.println(str);  
            **return**;  
        }  
   
        **if** (start > 0) {  
            printBraces(str+"{", start-1, end);  
        }  
   
        **if** (start < end) {  
            printBraces(str+"}", start, end-1);  
        }  
    }

**BST to Array and back**

Question: Write a function to convert a sorted array to a BST of minimum height. Write a function to convert a BST to a sorted array.

**public** **static** Node arrayToBST(int [] array, int start, int end) {  
    **if**(end < start)  
        **return** **null**;  
   
    int mid = (start + end)/2;  
   
    Node n = **new** Node(array[mid]);  
   
    n.left = arrayToBST(array, start, mid-1);  
    n.right= arrayToBST(array, mid+1, end);  
   
    **return** n;  
}  
   
**public** **static** Integer [] BSTtoArray(Node root) {  
    Stack<Node> stack = **new** Stack<Node>();  
    Node node = root;  
   
    ArrayList<Integer> list = **new** ArrayList<Integer>();  
   
    **while**(!stack.isEmpty() || node != **null**) {  
        **if**(node != **null**) {  
            stack.push(node);  
            node = node.left;  
        } **else** {  
            node = stack.pop();  
            list.add(node.data);  
            node = node.right;  
        }  
    }  
   
    Integer [] array = **new** Integer [list.size()];  
    list.toArray(array);  
    **return** array;  
}

**Island Perimeter**

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water. Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells). The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example

[0,1,0,0], [1,1,1,0], [0,1,0,0], [1,1,0,0](https://w.amazon.com/bin/view/0%2C1%2C0%2C0%5D%2C_%5B1%2C1%2C1%2C0%5D%2C_%5B0%2C1%2C0%2C0%5D%2C_%5B1%2C1%2C0%2C0)

Answer: 16 Explanation: The perimeter is the 16 yellow stripes in the image below:

**public** **static** int islandPerimeter(int[][] grid) {  
        int islandPerimeter = 0;  
        **for** (int i = 0; i < grid.length; i++) {  
            **for** (int j = 0; j < grid[i].length; j++) {  
                **if**(grid[i][j] == 1){  
                    **if** (i - 1 < 0 || grid[i-1][j] != 1) {  
                        islandPerimeter++;  
                    }  
                    **if** (j - 1 < 0 || grid[i][j-1] != 1) {  
                        islandPerimeter++;  
                    }  
                    **if** (j + 1 >= grid[i].length || grid[i][j+1] != 1 ){  
                        islandPerimeter++;  
                    }  
                    **if** (i + 1 >= grid.length || grid[i+1][j] != 1){  
                        islandPerimeter++;  
                    }  
                }  
            }  
        }  
        **return** islandPerimeter;  
    }

**Candy**

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

Each child must have at least one candy. Children with a higher rating get more candies than their neighbors. What is the minimum candies you must give?

Solution

**public** **static** int candy(int[] ratings) {  
    int[] left = **new** int[ratings.length];  
    int[] right = **new** int[ratings.length];  
      
    Arrays.fill(left, 1);  
    Arrays.fill(right, 1);  
      
    **for** (int i = 1; i < ratings.length; i++) {  
        **if** (ratings[i] > ratings[i-1]) {  
            left[i] = left[i-1] + 1;  
        }  
    }  
      
    **for** (int i = ratings.length -2; i >= 0; i--) {  
        **if** (ratings[i] > ratings[i+1]) {  
            right[i] = right[i+1] + 1;  
        }  
    }  
      
    int candies = 0;  
      
    **for** (int i = 0; i < ratings.length; i++) {  
        candies += Math.max(left[i], right[i]);  
    }  
    **return** candies;  
}

**Min Stack**

Question: Write an implementation of a Stack that supports push(), pop(), and min() functions in constant time.

**public** **class** **Stack** {  
    **public** **class** **Node** {  
        Integer value;  
        Node next;  
  
        **public** Node(Integer value) {  
            **this**.value = value;  
        }  
    }  
  
    **private** Node head;  
    **private** Node minHead;  
  
    **public** Stack() {  
        head = **null**;  
        minHead = **null**;  
    }  
  
    **public** void push(Integer value) {  
        Node node = **new** Node(value);  
  
        **if**(head == **null**) {  
            head = node;  
            minHead = **new** Node(value);  
        } **else** {  
            node.next = head;  
            head = node;  
  
            **if** (value <= minHead.value) {  
                Node newMin = **new** Node(value);  
                newMin.next = minHead;  
                minHead = newMin;  
            }  
        }  
    }  
  
    **public** Integer pop() {  
        **if** (head == **null**)  
            **return** **null**;  
  
        Node n = head;  
        head = head.next;  
  
        **if** (n.value == minHead.value)  
            minHead = minHead.next;  
  
        **return** n.value;  
    }  
  
    **public** Integer min() {  
        **if** (minHead == **null**)  
            **return** **null**;  
  
        **return** minHead.value;             
    }  
}

**Merge two sorted lists**

Question: Write a function, that given two sorted lists of integers as input, returns a single sorted list with items from both lists with no duplicate elements.

Example: input: a = {1,2,3}; b = {4,5,6}; output: c = {1,2,3,4,5,6}; input: a = {7,8,9}; b = {1,8,20,24}; output: c = {1,7,8,9,20,24}; input: a = {3,3,4}; b = {4}; output: c = {3,4}; input: a = {1,2,2,3,3,4,5,6,7}; b = {4,5,6,7,8,8,8}; output: c = {1,2,3,4,5,6,7,8};

**public** **static** List<Integer> merge(int [] a, int [] b) {  
        List<Integer> list = **new** ArrayList<Integer>();  
   
        int a\_index = 0;  
        int b\_index = 0;  
   
        Integer i = **null**;  
   
        **while**(a\_index < a.length && b\_index < b.length) {  
            **if**(a[a\_index] < b[b\_index]) {  
                **if**(i == **null** || i < a[a\_index]) {  
                    i = a[a\_index];  
                    list.add(i);  
                }  
   
                a\_index++;  
            } **else** {  
                **if**(i == **null** || i < b[b\_index]) {  
                    i = b[b\_index];  
                    list.add(i);  
                }  
   
                b\_index++;  
            }  
        }  
   
        **while** (a\_index < a.length) {  
            **if** (i == **null** || i < a[a\_index]) {  
                i = a[a\_index];  
                list.add(i);  
            }  
            a\_index++;  
        }  
   
        **while** (b\_index < b.length) {  
            **if** (i == **null** || i < b[b\_index]) {  
                i = b[b\_index];  
                list.add(i);  
            }  
   
            b\_index++;  
        }  
   
        **return** list;  
    }

**Sorting**

Question: Write an efficient sorting algorithm.

**public** **static** int[] quickSort(int [] array, int low, int high) {  
        **if**(array == **null**)  
            **return** **null**;  
   
        **if**(array.length==0)  
            **return** array;  
   
        int i = low, j = high;  
   
        int pivot = array[low + (high-low)/2];  
   
        **while** (i <= j) {  
            **while** (array[i] < pivot)  
                i++;  
   
            **while** (array[j] > pivot)  
                j--;   
   
            **if** (i <= j) {  
                int temp = array[i];  
                array[i] = array[j];  
                array[j] = temp;  
                i++;  
                j--;  
            }  
        }  
   
        **if** (low < j)  
            quickSort(array, low, j);  
   
        **if** (i < high)  
            quickSort(array, i, high);  
   
        **return** array;  
    }  
   
    **public** **static** int[] mergeSort(int [] array) {  
        **if**(array == **null**)  
            **return** **null**;  
   
        **if**(array.length < 2)  
            **return** array;  
   
        int [] array1 = **new** int[array.length/2];  
   
        System.arraycopy(array, 0, array1, 0, array1.length);  
   
        int [] array2;  
   
        **if**(array.length%2==0)  
            array2 = **new** int[array.length/2];  
        **else**   
            array2 = **new** int[array.length/2+1];  
   
        System.arraycopy(array, array1.length, array2, 0, array2.length);  
   
        array1 = mergeSort(array1);  
        array2 = mergeSort(array2);  
   
        int a = 0;  
        int a1 = 0;  
        int a2 = 0;  
   
        **while**(a1 < array1.length && a2 < array2.length)  
            **if**(array1[a1]<array2[a2])  
                array[a++]=array1[a1++];  
            **else**  
                array[a++]=array2[a2++];  
   
        **while**(a1 < array1.length)  
            array[a++]=array1[a1++];  
   
        **while**(a2 < array2.length)  
            array[a++]=array2[a2++];  
   
        **return** array;  
    }

**Missing Spaces**

Question: You are given a sentence with no spaces and dictionary containing thousands of words. Write an algorithm to reconstruct the sentence by inserting spaces in the appropriate positions.

Example:

input: "theskyisblue" output: "the sky is blue"

input: "thegrassisgreen" output: "the grass is green"

**public** **static** String makeSentence(String str, Set<String> dictionary) {  
        char [] array = str.toCharArray();  
   
        StringBuilder prefix = **new** StringBuilder();  
   
        **for**(int i = 0; i < array.length; i++) {  
            prefix.append(array[i]);  
   
            **if**(dictionary.contains(prefix.toString())) {  
                **if** (prefix.length() == array.length)  
                    **return** prefix.toString();  
   
                String suffix = makeSentence(**new** String(array,prefix.length(),array.length-prefix.length()), dictionary);  
   
                **if** (suffix != **null**) {  
                    prefix.append(" ");  
                    prefix.append(suffix);  
                    **return** prefix.toString();  
                }  
            }  
        }  
   
        **return** **null**;  
    }

**Nth smallest element in a binary search tree**

Question: Write a function to find the nth smallest element in a binary search tree

**class** **Node** {  
    int i;  
    Node left;  
    Node right;  
}  
  
**public** **static** Node nthSmallestElement(Node root, int n) {  
    Stack<Node> stack = **new** Stack<Node>();  
    Node node = root;  
    int i = 0;  
   
    **while** (!stack.isEmpty() || node != **null**) {  
        **if** (node != **null**) {  
            stack.push(node);  
            node = node.left;  
        } **else** {  
            node = stack.pop();  
            **if**(++i == n) {  
                **return** node;  
            }  
            node = node.right;  
        }  
    }  
   
    **return** **null**;  
}

**LinkedList of Nodes in a Binary Tree**

Given a binary tree, design an algorithm which creates a linked list of all the nodes at each depth.

**public** **static** ArrayList<LinkedList<Node>> makeLinkedLists(Node root) {  
    LinkedList<Node> current = **new** LinkedList<Node>();  
    LinkedList<Node> nextLevel = **new** LinkedList<Node>();  
    ArrayList<LinkedList<Node>> lists = **new** ArrayList<LinkedList<Node>>();  
  
    current.add(root);  
  
    **while** (!current.isEmpty()) {  
        lists.add(current);  
          
        **for** (Node n : current)) {  
            **if** (n.left != **null**) nextLevel.add(n.left);  
            **if** (n.right != **null**) nextLevel.add(n.right);  
        }  
  
        current = nextLevel;  
        nextLevel = **new** LinkedList<Node>();  
    }  
    **return** lists;  
}

**Inorder Successor in Binary Search Tree**

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

**public** **static** Node inorderSuccessor(Node root, Node n) {  
    **if**(n.right != **null**){  
        n = n.right;  
        **while**(n.left !=**null**){  
            n = n.left;  
        }  
        **return** n;  
    }  
    Node successor = **null**;  
    **while**(root != **null**) {  
        **if**(n.data < root.data){  
            successor = root;  
            root = root.left;  
        } **else** **if**(n.data > root.data){  
            root = root.right;  
        } **else** {  
            **break**;  
        }  
    }  
    **return** successor;  
}

**Find min element in rotated, sorted array**

Find the minimum element in a rotated, sorted array.

Example: 4,5,6,7,8,2,3 Answer: 2

**public** **static** int (int[] array) {  
    int start = 0;  
    int end = array.length - 1;  
  
    **while**(start < end) {  
        **if** (array[start] < array[end]) {  
            **return** array[start];  
        }  
          
        int mid = array[(start + end)/2];  
        **if** (array[mid] >= array[start]) {  
            start = mid+1;  
       } **else** {  
           end = mid;  
       }  
    }  
    **return** num[start];  
}

**Palindrome**

Question: Write a function to find out whether a string is a palindrome.

**public** **static** boolean isPalindrome(String input) {  
        int i = 0;  
        int j = input.length() - 1;  
   
        **while** (i < j) {  
            **if** (input.charAt(i) != input.charAt(j)) {  
                **return** **false**;  
            }  
            i++;  
            j--;  
        }  
   
        **return** **true**;  
    }

**Heap**

Implement a min heap

**public** **class** **BinaryHeap**<T **extends** Comparable<T>>{  
    **private** **static** **final** int DEFAULT\_CAPACITY = 10;  
    **private** int size;  
    **private** T[] heap;  
  
    **public** BinaryHeap() {  
        size = 0;  
        heap = (T[]) **new** Comparable[DEFAULT\_CAPACITY];  
    }  
  
    **public** T peek() {  
        **if** (**this**.isEmpty()) {  
            **return** **null**;  
        }  
        **return** heap[0];  
    }  
  
    **public** void add(T value) {  
        **if** (size >= heap.length) {  
            heap = **this**.resize();  
        }  
  
        heap[size++] = value;  
        bubbleUp();  
    }  
  
    **public** T remove() {  
        T result = peek();  
  
        heap[0] = heap[size - 1];  
        heap[size - 1] = **null**;  
        size--;  
  
        bubbleDown();  
  
        **return** result;  
    }  
  
    **public** boolean isEmpty() {  
        **return** size == 0;  
    }  
  
    **public** int size() {  
        **return** size;  
    }  
  
    **private** T[] resize() {  
        **return** java.util.Arrays.copyOf(heap, heap.length \* 2);  
    }  
  
    **private** boolean hasParent(int i) {  
        **return** i > 0;  
    }  
  
    **private** int leftIndex(int i) {  
        **return** (i \* 2) + 1;  
    }  
  
    **private** int rightIndex(int i) {  
        **return** (i \* 2) + 2;  
    }  
  
    **private** boolean hasLeftChild(int i) {  
        **return** leftIndex(i) < size;  
    }  
  
    **private** boolean hasRightChild(int i) {  
        **return** rightIndex(i) < size;  
    }  
  
    **private** T parent(int i) {  
        **return** heap[parentIndex(i)];  
    }  
  
    **private** int parentIndex(int i) {  
        **return** (i-1) / 2;  
    }  
  
    **private** void swap(int index1, int index2) {  
        T tmp = heap[index1];  
        heap[index1] = heap[index2];  
        heap[index2] = tmp;  
    }  
  
    **private** void bubbleUp() {  
        int index = **this**.size - 1;  
  
        **while** (hasParent(index)  
                && (parent(index).compareTo(heap[index]) > 0)) {  
            swap(index, parentIndex(index));  
            index = parentIndex(index);  
        }  
    }  
  
    **private** void bubbleDown() {  
        int index = 0;  
  
        **while** (hasLeftChild(index)) {  
            int smallerChild = leftIndex(index);  
  
            **if** (hasRightChild(index)  
                    && heap[leftIndex(index)].compareTo(heap[rightIndex(index)]) > 0) {  
                smallerChild = rightIndex(index);  
            }  
  
            **if** (heap[index].compareTo(heap[smallerChild]) > 0) {  
                swap(index, smallerChild);  
            } **else** {  
                **break**;  
            }  
  
            index = smallerChild;  
        }  
    }  
}

**Median Value**

Question: Write a function to find the median value from a stream of integers.

**public** **static** Double median (Iterator<Integer> stream){  
    Queue<Integer> minHeap = **new** PriorityQueue<Integer>();  
    Queue<Integer> maxHeap = **new** PriorityQueue<Integer>(20, Collections.reverseOrder());  
  
    **while**(stream.hasNext()){  
        maxHeap.add(stream.next());  
  
        **if** (!minHeap.isEmpty() && maxHeap.peek() > minHeap.peek()){  
            Integer maxHeapRoot = maxHeap.poll();  
            Integer minHeapRoot = minHeap.poll();  
            maxHeap.add(minHeapRoot);  
            minHeap.add(maxHeapRoot);  
        }  
  
        **if**(maxHeap.size() - minHeap.size() > 1) {  
            minHeap.add(maxHeap.poll());  
        }  
    }  
  
    **if** (maxHeap.size() > minHeap.size()){  
        **return** maxHeap.peek().doubleValue();  
    }  
  
    **return** (maxHeap.peek() + minHeap.peek()) / 2.0;  
}

**Calculate Angle**

Given the time, calculate the smaller angle between the hour and minute hands on an analog clock.

**public** **static** int (int hour, int min) {  
    int minuteAngle = min \* 6;

   int hourAngle = hour \* 30 + (min / 2);  
      
    int angle = Math.abs(minuteAngle - hourAngle);  
    **return** Math.min(360 -  angle, angle);  
}

**Implement a LRU Cache**

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

**class** **Node**{  
    int key;  
    int value;  
    Node pre;  
    Node next;  
   
    **public** Node(int key, int value){  
        **this**.key = key;  
        **this**.value = value;  
    }  
}  
  
**public** **class** **LRUCache** {  
    int capacity;  
    HashMap<Integer, Node> map = **new** HashMap<Integer, Node>();  
    Node head=**null**;  
    Node end=**null**;  
   
    **public** LRUCache(int capacity) {  
        **this**.capacity = capacity;  
    }  
   
    **public** int get(int key) {  
        **if**(map.containsKey(key)){  
            Node n = map.get(key);  
            remove(n);  
            setHead(n);  
            **return** n.value;  
        }  
   
        **return** -1;  
    }  
   
    **public** void remove(Node n){  
        **if**(n.pre!=**null**){  
            n.pre.next = n.next;  
        }**else**{  
            head = n.next;  
        }  
   
        **if**(n.next!=**null**){  
            n.next.pre = n.pre;  
        }**else**{  
            end = n.pre;  
        }  
   
    }  
   
    **public** void setHead(Node n){  
        n.next = head;  
        n.pre = **null**;  
   
        **if**(head!=**null**)  
            head.pre = n;  
   
        head = n;  
   
        **if**(end ==**null**)  
            end = head;  
    }  
   
    **public** void set(int key, int value) {  
        **if**(map.containsKey(key)){  
            Node old = map.get(key);  
            old.value = value;  
            remove(old);  
            setHead(old);  
        }**else**{  
            Node created = **new** Node(key, value);  
            **if**(map.size()>=capacity){  
                map.remove(end.key);  
                remove(end);  
                setHead(created);  
   
            }**else**{  
                setHead(created);  
            }      
   
            map.put(key, created);  
        }  
    }  
}

**Rotate 2D Array**

Write a function to rotate an NxN array 90 degrees.

**public** **static** void rotate(int[][] matrix, int n) {  
    **for** (int layer = 0; layer < n / 2; ++layer) {  
        int first = layer;  
        int last = n - 1 - layer;  
        **for**(int i = first; i < last; ++i) {  
            int offset = i - first;  
            int top = matrix[first][i]; // save top  
  
            // left -> top  
            matrix[first][i] = matrix[last-offset][first];            
            // bottom -> left  
            matrix[last-offset][first] = matrix[last][last - offset];   
  
            // right -> bottom  
            matrix[last][last - offset] = matrix[i][last];   
  
            // top -> right  
            matrix[i][last] = top; // right <- saved top  
        }  
    }  
}

**Implement a Queue using two stacks**

**public** **class** **TwoStacks** {  
    Stack<Object> pushStack;  
    Stack<Object> popStack;  
  
    **public** TwoStacks (){  
        pushStack = **new** Stack<>();  
        popStack = **new** Stack<>();  
    }  
  
    **public** void push(Object o) {  
        pushStack.push(o);  
    }  
  
    **public** Object pop() {  
        **if** (!popStack.isEmpty()) {  
            **return** popStack.pop();  
        }  
        **while**(!pushStack.isEmpty()){  
            popStack.push(pushStack.pop());  
        }  
        **if** (!popStack.isEmpty()) {  
            **return** popStack.pop();  
        }  
        **return** **null**;  
    }  
}

**Balanced Binary Tree**

Implement an algorithm to determine is a binary tree is balanced. A balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differ by more than one.

**public** **static** boolean isBalanced(Node root){  
    **return** (getHeight(root) != -1);  
}  
**public** **static** int getHeight(Node root) {  
    **if** (root == **null**) {  
        **return** 0;  
    }  
      
    int leftHeight = getHeight(root.left);  
    **if** (leftHeight == -1) {  
        **return** -1;  
    }  
      
    int rightHeight = getHeight(root.right);  
    **if** (rightHeight == -1) {  
        **return** -1;  
    }  
      
    **if** (Math.abs(leftHeight - rightHeight) > 1) {  
        **return** -1;  
    }  
      
    **return** Math.max(leftHeight, rightHeight) + 1;  
}

**Lowest Common Ancestor**

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

**public** TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  
    **if** (root == **null** || root == p || root == q) **return** root;  
    TreeNode left = lowestCommonAncestor(root.left, p, q);  
    TreeNode right = lowestCommonAncestor(root.right, p, q);  
    **return** left == **null** ? right : right == **null** ? left : root;  
}

**Fibonacci**

Implement a function which returns the nth number in Fibonacci sequences with an input n.

**public** **static** int fibonacci(int n) {  
    **if** (n == 0 || n == 1) {  
        **return** n;  
    }  
  
    int fibMinus1 = 1;  
    int fibMinus2 = 0;  
  
    int fibN = 0;  
  
    **for** (int i = 2; i <= n; i++) {  
        fibN = fibMinus1 + fibMinus2;  
        fibMinus2 = fibMinus1;  
        fibMinus1 = fibN;  
    }  
  
    **return** fibN;  
}

**First Unique Character**

Find the first unique character in a string.

**public** **static** Character firstUniqueCharacter(char[] string) {  
    Map<Character, Integer> map = **new** HashMap<Character, Integer>();  
    **for** (char c: string) {  
        Integer i = map.get(c);  
        map.put(c, (i==**null**) ? 1: 1+i);  
    }  
    **for** (char c: string) {  
        Integer i = map.get(c);  
        **if** (i == 1) {  
            **return** c;  
        }  
    }  
    **return** **null**;  
}

**Largest number in BST which is less than or equal to N**

We have a binary search tree and a number N. Our task is to find the greatest number in the binary search tree that is less than or equal to N. Print the value of the element if it exists otherwise print -1.

**public** **static** int (Node root, int n) {  
    int val = -1;  
    **while**(root != **null**) {  
        **if**(root.data > n) {  
            root = root.left;  
        } **else** {  
            val = root.data;  
            root = root.right;  
        }  
    }  
    **return** val;  
}