```java
package problems;

public class Adding2NumbersList
{
        static int carry = 0;

        private static void padZeros(MyLinkedList<Integer> l, int len)
        {
                for (int i = 0; i < len; i++)
                        l.insertAtStart(0);
        }

        private static MyLinkedList<Integer> addLists(MyLinkedList<Integer> l1, MyLinkedList<Integer> l2)
        {
                if (l1 == null && l2 == null)
                        return null;

                if (l1 == null || l1.getHead() == null)
                        return l2;

                if (l2 == null || l2.getHead() == null)
                        return l1;

                int len1 = l1.length();
                int len2 = l2.length();

                if (len1 > len2)
                        padZeros(l2, len1 - len2);
                else
                        padZeros(l1, len2 - len1);

                l1.printList();
                l2.printList();
                MyLinkedList<Integer> l3 = new MyLinkedList<Integer>();
                l3.setHead(addLists(l1.getHead(), l2.getHead()));

                if (carry > 0)
                {
                        l3.insertAtStart(carry);
                        carry = 0;
                }

                return l3;
        }

        private static ListNode<Integer> addLists(ListNode<Integer> t1, ListNode<Integer> t2)
        {
                if (t1 == null && t2 == null)
```

```java
                        return null;

                ListNode<Integer> t = addLists(t1.next, t2.next);
                int sum = t1.data + t2.data + carry;
                ListNode<Integer> t3 = new ListNode<Integer>(sum % 10);
                carry = sum / 10;
                t3.next = t;

                return t3;
        }

        public static void main(String[] args)
        {
                MyLinkedList<Integer> l1 = new MyLinkedList<Integer>();
                l1.insertAtEnd(7);
                l1.insertAtEnd(1);
                l1.insertAtEnd(6);
                l1.insertAtEnd(6);

                // l1.printList();
                // l1.setHead(l1.reverseList(l1.getHead()));
                // l1.printList();

                MyLinkedList<Integer> l2 = new MyLinkedList<Integer>();
                l2.insertAtEnd(5);
                l2.insertAtEnd(9);
                l2.insertAtEnd(5);

                MyLinkedList<Integer> l3 = addLists(l1, l2);
                if (l3 != null)
                        l3.printList();
        }

}
```

```java
package problems;

public class Adding2NumbersListRev
{

        public static MyLinkedList<Integer> addLists(MyLinkedList<Integer> l1, MyLinkedList<Integer> l2)
        {
                if (l1 == null && l2 == null)
                        return null;

                ListNode<Integer> t1 = (l1 != null) ? l1.getHead() : null;
                ListNode<Integer> t2 = (l2 != null) ? l2.getHead() : null;

                MyLinkedList<Integer> l3 = new MyLinkedList<Integer>();

                int sum = 0, carry = 0;

                while (t1 != null && t2 != null)
                {
                        sum = (t1.data + t2.data + carry);
                        carry = sum / 10;
                        l3.insertAtEnd(sum % 10);
                        t1 = t1.next;
                        t2 = t2.next;
                }

                while (t1 != null)
                {
                        sum = t1.data + carry;
                        l3.insertAtEnd(sum % 10);
                        carry = sum / 10;
                        t1 = t1.next;
                }

                while (t2 != null)
                {
                        sum = t2.data + carry;
                        l3.insertAtEnd(sum % 10);
                        carry = sum / 10;
                        t2 = t2.next;
                }

                if (carry > 0)
                        l3.insertAtEnd(carry);

                return l3;

        }

        public static void main(String[] args)
        {
                 MyLinkedList<Integer> l1 = new MyLinkedList<Integer>();
                 l1.insertAtEnd(7);
                 l1.insertAtEnd(1);
                 l1.insertAtEnd(6);
                 l1.printList();

                MyLinkedList<Integer> l2 = new MyLinkedList<Integer>();
                l2.insertAtEnd(5);
                l2.insertAtEnd(9);
                l2.insertAtEnd(5);
                l2.printList();

                MyLinkedList<Integer> l3 = addLists(l1, l2);
                if (l3 != null)
                        l3.printList();
        }
}
```

```java
package problems;

public class ArrayHopper
{
        // prints the path from first touch down till end of the canyon
        static void printPath(int[] path, int idx)
        {
                if (idx == 0)
                        return;
                printPath(path, path[idx]);
                System.out.print(path[idx] + ", ");
        }

        // finds the last touch down which can lead out of the canyon
        static int findLastIndex(int[] canyons)
        {
                int lastIndex = -1;
                for (int i = canyons.length - 1; i >= 0; i--)
                {
                        if ((canyons[i] + i) >= canyons.length)
                                lastIndex = i;
                }
                return lastIndex;
        }

        static void findPath(int[] canyons)
        {
                int size = canyons.length;
                if (size == 0 || canyons[0] == 0)
                {
                        System.out.println("failure");
                        return;
                }
                int hops[] = new int[size]; // stores the min touch downs from to reach
                                                                        // all points in the array.
                int path[] = new int[size]; // stores the previous touch down for all
                                                                        // points in the array.
                hops[0] = 0;

                for (int i = 1; i < size; i++)
                {
                        hops[i] = Integer.MAX_VALUE;
                        for (int j = 0; j < i; j++)
                        {
                                if (i <= j + canyons[j] && hops[j] != Integer.MAX_VALUE)
                                {
                                        if (hops[i] > hops[j] + 1)
                                        {
                                                path[i] = j;
                                                hops[i] = hops[j] + 1;
                                        }
                                        break;
                                }
                        }
                }
                int lastIndex = findLastIndex(canyons);
                if (hops[size - 1] == Integer.MAX_VALUE || lastIndex == -1)
                {
                        System.out.println("failure");
                        return;
                }

                printPath(path, lastIndex);
                System.out.print(lastIndex + ", out");
        }

        public static void main(String args[]) throws Exception
        {
                int[] canyons = { 1, 0, 0, 4, 0, 0, 0, 2, 0 };
                findPath(canyons);
        }
}
```

```java
package problems;

public class ArrayRearrange
{
        public static void swap(int a[], int head, int tail)
        {
                int c = a[head];
                a[head] = a[tail];
                a[tail] = c;
        }

        public static void printArray(int a[])
        {
                int length = 0;
                while (length < a.length)
                {
                        System.out.print(a[length] + " ");
                        length++;
                }
                System.out.println();
        }

        public static void main(String[] args)
        {
                int a[] = { -1, 0, -2, 0, 6, -5, 0, -3, -4 };

                int head = 0, pass = 0;
                while (pass < 2)
                {
                        int tail = a.length - 1;
                        while (head <= tail)
                        {
                                if (pass > 0 ? (a[head] == 0) : (a[head] < 0))
                                {
                                        head++;
                                        continue;
                                }
                                else
                                {
                                        if (pass > 0 ? (a[tail] == 0) : (a[tail] < 0))
                                        {
                                                swap(a, head, tail);
                                                head++;
                                        }
                                        tail--;
                                }
                        }
                        pass++;
                        System.out.println(head + " " + tail);
                        printArray(a);
                }
        }
}
```

```java
package problems;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

class A
{

}

public class B extends A
{
        public void getA()
        {
                System.out.println("A");
        }

        public static void main(String[] args) throws IOException
        {
                A a = new A();
                A b = new B();
                ((B) a).getA(); // throws a Runtime exception - ClassCastException
                ((B) b).getA();

                File file = new File("/Users/Anand/Documents/home_value.txt");

                FileWriter writer = new FileWriter(file);

                writer.write("50000 26000\n");
                for (int i = 0; i < 50000; i++)
                        writer.write(i + " ");
                writer.flush();
                writer.close();
        }

}
```

```java
package problems;

import java.util.LinkedList;
import java.util.Queue;

class TreeNode
{

        public int data;
        public TreeNode left = null;
        public TreeNode right = null;

        public TreeNode(int d)
        {
                data = d;
        }
}

public class BinaryTree
{
        TreeNode root;

        public void deleteTree()
        {
                root = null;
        }

        public int height(TreeNode root)
        {
                if (root == null)
                        return -1;

                return Math.max(height(root.left), height(root.right)) + 1;
        }

        public static void printInOrder(TreeNode root)
        {
                if (root == null)
                        return;
                printInOrder(root.left);
                System.out.print(root.data + " ");
                printInOrder(root.right);
        }

        static TreeNode last_node = null;

        public TreeNode inOrderSuccessor(TreeNode root, TreeNode node)
        {
                if (root == null || node == null)
                        return null;
                TreeNode succ = null;

                succ = inOrderSuccessor(root.left, node);
                if (last_node != null && last_node == node)
                        return root;
                last_node = root;
                succ = inOrderSuccessor(root.right, node);

                return succ;
        }

        public TreeNode find(TreeNode root, int d)
        {
                if (root == null)
                        return null;

                if (root.data == d)
                        return root;

                TreeNode temp;

                temp = find(root.left, d);
                if (temp != null)
                        return temp;
                temp = find(root.right, d);
                return temp;

        }

        public TreeNode insert(TreeNode root, int d)
        {
                return null;
        }

        public void printPaths()
        {
                int[] path = new int[height(root)];
                printPaths(root, path, 0);
        }

        private void printPaths(TreeNode node, int[] path, int level)
        {
                if (node == null)
                        return;

                path[level] = node.data;
                // System.out.println("level: " + level);

                if (node.left == null && node.right == null)
                {
                        printArray(path, level);
                }
                else
                {
                        printPaths(node.left, path, level + 1);
                        printPaths(node.right, path, level + 1);
                }
        }

        private void printArray(int[] ints, int level)
        {
                for (int i = 0; i <= level; i++)
                        System.out.print(ints[i] + " ");
                System.out.println();
        }

        public boolean BFS(TreeNode root, int d)
        {
                if (root == null)
                        return false;

                if (root.data == d)
                        return true;

                Queue<TreeNode> q = new LinkedList<TreeNode>();
                TreeNode temp;
                q.offer(root);

                while (!q.isEmpty())
                {
                        temp = q.poll();
                        if (temp != null)
                        {
                                if (temp.data == d)
                                        return true;
                                if (temp.left != null)
                                        q.offer(temp.left);
                                if (temp.right != null)
                                        q.offer(temp.right);
                        }
                }
                return false;
        }

```

```java
        public boolean DFS(TreeNode root, int d)
        {
                if (root == null)
                        return false;

                if (root.data == d)
                        return true;

                return (DFS(root.left, d) || DFS(root.right, d));
        }

        public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
        {
                if (p == null || q == null)
                        return null;

                if (root == null)
                        return null;

                if (root == p || root == q)
                        return root;

                TreeNode leftLCA = lowestCommonAncestor(root.left, p, q);
                TreeNode rightLCA = lowestCommonAncestor(root.right, p, q);

                if (leftLCA != null && rightLCA != null)
                        return root;

                return (leftLCA == null) ? rightLCA : leftLCA;
        }

        private boolean isBST(TreeNode root, int min, int max)
        {
                if (root == null)
                        return true;

                if (root.data <= min || root.data > max)
                        return false;

                return isBST(root.left, min, root.data) && isBST(root.right, root.data, max);
        }

        public boolean isBST(TreeNode root)
        {
                return isBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
        }
}
```

```java
package problems;

import java.util.Scanner;

public class BitFlip
{
        static int findWindow(int[] bin)
        {
                int start = 0, end = 0;
                int len = bin.length, max = 0, numZeros = 0;
                int globalstart = 0;
                for (int i = 0; i < len; i++)
                {
                        if (numZeros == 0 && bin[i] == 1)
                        {
                                start = i + 1;
                                continue;
                        }
                        numZeros = bin[i] == 0 ? numZeros + 1 : numZeros - 1;
                        if (max < numZeros)
                        {
                                max = numZeros;
                                end = i;
                                globalstart = start;
                        }
                }
                if (globalstart == len)
                        return len;

                int i = 0, res = 0;
                while (i < globalstart)
                {
                        if (bin[i++] == 1)
                                res++;
                }

                while (i <= end)
                {
                        if (bin[i++] == 0)
                                res++;
                }

                while (i < len)
                {
                        if (bin[i++] == 1)
                                res++;
                }

                return res;
        }

        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                int n = sc.nextInt();
                int bin[] = new int[n];
                for (int i = 0; i < n; i++)
                        bin[i] = sc.nextInt();
                System.out.println(findWindow(bin));
                sc.close();
        }

}
```

```java
package problems;

import java.util.*;

public class BiValuedSlice
{
        public static int solution(int[] A)
        {
                int idx = 0, numCount = 0;
                Set<Integer> visited = new HashSet<Integer>();

                while (idx < A.length)
                {
                        int tmp = 0;
                        for (int i = idx; i < A.length; i++)
                        {
                                visited.add(A[i]);
                                if (visited.size() == 3)
                                {
                                        visited.clear();
                                        if (tmp > numCount)
                                                numCount = tmp;
                                        break;
                                }
                                tmp++;
                        }
                        idx++;
                }
                return numCount;
        }

        public static void main(String[] args)
        {
        }

}
```

```java
package problems;

public class BST extends BinaryTree
{
        public BST()
        {
                root = null;
        }

        @Override
        public TreeNode find(TreeNode root, int d)
        {
                if (root == null)
                        return null;

                while (root != null)
                {
                        if (root.data == d)
                                return root;

                        if (d < root.data)
                                root = root.left;
                        else
                                root = root.right;
                }

                return null;
        }

        @Override
        public TreeNode insert(TreeNode root, int d)
        {
                if (root == null)
                        return new TreeNode(d);

                if (d < root.data)
                        root.left = insert(root.left, d);

                else
                        root.right = insert(root.right, d);

                return root;
        }

        private int minVal(TreeNode root)
        {
                TreeNode node = root;
                while (node.left != null)
```

```java
                        node = node.left;
                return node.data;
        }

        public TreeNode deleteNode(TreeNode root, int key)
        {
                if (root == null)
                        return null;

                if (root.data > key)
                        root.left = deleteNode(root.left, key);
                else if (root.data < key)
                        root.right = deleteNode(root.right, key);
                else
                {
                        if (root.right == null)
                                return root.left;
                        else if (root.left == null)
                                return root.right;

                        root.data = minVal(root.right);
                        root.right = deleteNode(root.right, root.data);
                }
                return root;
        }

        @Override
        public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
        {
                if (root == null || p == null || q == null)
                        return null;

                if (root.data < p.data && root.data < q.data)
                        return lowestCommonAncestor(root.right, p, q);
                if (root.data > p.data && root.data > q.data)
                        return lowestCommonAncestor(root.left, p, q);
                return root;
        }

        public static void main(String[] args)
        {
                BinaryTree tree = new BST();
                tree.root = tree.insert(tree.root, 10);
                tree.root = tree.insert(tree.root, 6);
                tree.root = tree.insert(tree.root, 8);
                tree.root = tree.insert(tree.root, 14);
                tree.root = tree.insert(tree.root, 16);
                tree.root = tree.insert(tree.root, 9);
                tree.root = tree.insert(tree.root, 5);

                TreeNode temp = tree.find(tree.root, 9);
                System.out.println("temp: " + ((temp == null) ? "null" : temp.data));

                System.out.println(tree.inOrderSuccessor(tree.root, temp).data);

                System.out.println(tree.BFS(tree.root, 7));

                System.out.println(tree.DFS(tree.root, 7));

                System.out.println(tree.isBST(tree.root));

        }
}
```

```java
package problems;

import java.util.Arrays;

public class CanPartition
{
        static boolean canPartition(int[] nums)
        {
                if (nums == null || nums.length < 2)
                        return false;

                int target = 0;
                for (int i : nums)
                        target += i;

                if ((target & 1) == 1)
                        return false;

                target /= 2;
                boolean[] part = new boolean[target + 1];
                part[0] = true;
                for (int i = 0; i < nums.length; i++)
                {
                        for (int j = target; j >= nums[i]; j--)
                                part[j] = part[j] || part[j - nums[i]];
                        System.out.println(Arrays.toString(part));
                }
                return part[target];
        }

        public static void main(String[] args)
        {
                int[] a = { 4, 6, 8 };
                System.out.println(canPartition(a));
        }

}
```

```java
package problems;

import java.util.Arrays;

public class ClimbSteps
{
        public static long countWays(int s, long arr[])
        {
                if (s < 0)
                        return 0;
                if (s == 0)
                        return 1;
                if (arr[s] >= 0)
                        return arr[s];
                // since we count no of ways there is no "1 + prev value" logic.
                // it is only for finding min no: of steps
                arr[s] = countWays(s - 1, arr) + countWays(s - 2, arr) + countWays(s - 3, arr);
                return arr[s];
        }

        public static int countWaysSlow(int s)
        {
                if (s < 0)
                        return 0;
                if (s == 0)
                        return 1;
                return countWaysSlow(s - 1) + countWaysSlow(s - 2) + countWaysSlow(s - 3);
        }

        public static void main(String[] args)
        {
                long[] arr = new long[37];
                Arrays.fill(arr, -1);
                System.out.println(countWays(4, arr));
                // System.out.println(countWaysSlow(36));
        }
}
```

```java
package problems;

import java.util.Arrays;

public class CoinChange
{
        static int countWays(int[] coins, int n)
        {
                int[] ways = new int[n + 1];
                ways[0] = 1;

                for (int i = 0; i < coins.length; i++)
                {
                        // for every new coin that you see update the array with new ways
                        // only starting from that new coin since lesser denoms can't be
                        // obtained.
                        for (int j = coins[i]; j <= n; j++)
                        {
                                ways[j] += ways[j - coins[i]];
                                System.out.println(Arrays.toString(ways));
                        }
                }

                System.out.println(Arrays.toString(ways));
                return ways[n];
        }

        public static void main(String[] args)
        {
                int[] coins = { 4, 2, 3 };
                System.out.println("No: of ways: " + countWays(coins, 7));
        }
}
```

```java
package problems;

import java.util.*;

public class CommonMin
{

        public static int solution(int[] A, int[] B)
        {
                int result = Integer.MAX_VALUE;
                Set<Integer> hashSet = new HashSet<Integer>();
                for (int num : A)
                        hashSet.add(num);

                for (int num : B)
                {
                        if (!hashSet.contains(num))
                                continue;
                        if (num < result)
                                result = num;
                }

                return (result == Integer.MAX_VALUE) ? -1 : result;
        }

        public static void main(String[] args)
        {
                int a = Integer.MAX_VALUE, b = Integer.MAX_VALUE;
                int c = Math.addExact(a, b);
                System.out.println(c);
        }

}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class ConfusingFibbonaci
{
        public static void main(String[] args)
        {
                List<Integer> l = new ArrayList<Integer>();
                l.add(0);
                l.add(1);
                int n = l.size();

                while (n < 8)
                {
                        n = l.size();
                        l.add(l.get(n - 1) + l.get(n - 2));
                        n++;
                }
                System.out.println(Arrays.toString(l.toArray()));
                System.out.println(l.get(l.size() - 1));
        }
}
```

```java
package problems;

import java.util.Arrays;

public class CountingSort
{
        static void sort(char[] a)
        {
                int n = a.length;
                char output[] = new char[n];

                int freq[] = new int[26];
                for (int i = 0; i < n; ++i)
                        freq[a[i] - ’a’]++;

                for (int i = 1; i < 26; i++)
                        freq[i] += freq[i - 1];

                for (int i = n - 1; i >= 0; --i)
                {
                        output[freq[a[i] - ’a’] - 1] = a[i];
                        freq[a[i] - ’a’]--;
                }

                for (int i = 0; i < n; ++i)
                        a[i] = output[i];
        }

        public static void main(String[] args)
        {
                char a[] = { ’a’, ’n’, ’a’, ’n’, ’d’, ’k’, ’u’, ’m’, ’a’, ’r’ };
                sort(a);
                System.out.print(Arrays.toString(a));
        }

}
```

```java
package problems;

public class CountIslands
{
        static int[] rowNum = { -1, -1, -1, 0, 0, 1, 1, 1 };
        static int[] colNum = { 0, -1, 1, -1, 1, -1, 1, 0 };

        static void merge(char[][] grid, int i, int j)
        {
                int m = grid.length, n = grid[0].length;

                if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] != ’1’)
                        return;

                grid[i][j] = ’X’;

                for (int a = 0; a < rowNum.length; a++)
                        merge(grid, i + rowNum[a], j + colNum[a]);
        }

        public static int numIslands(char[][] grid)
        {
                if (grid == null || grid.length == 0 || grid[0].length == 0)
                        return 0;

                int m = grid.length, n = grid[0].length, count = 0;
                for (int i = 0; i < m; i++)
                {
                        for (int j = 0; j < n; j++)
                        {
                                if (grid[i][j] == ’1’)
                                {
                                        count++;
                                        merge(grid, i, j);
                                }
                        }
                }
                return count;
        }

        public static void main(String[] args)
        {
                char[][] grid = { { ’1’, ’1’, ’0’, ’0’, ’0’ }, { ’0’, ’1’, ’0’, ’0’, ’1’ }, { ’1’, ’0’, ’0’, ’1’, ’1’ },
                                { ’0’, ’0’, ’0’, ’0’, ’0’ }, { ’1’, ’0’, ’1’, ’0’, ’1’ } };
                System.out.println(numIslands(grid));
        }

}
```

```java
package problems;

public class CountOccurencesSortedArray
{
        public static int countOccurences(int[] a, int x)
        {
                int i = firstOccurence(a, x);
                System.out.println("first: " + i);
                if (i == -1)
                        return 0;
                int j = lastOccurence(a, i, x);
                System.out.println("last: " + j);
                return j - i + 1;
        }

        // go left if a[mid] is equal to search element
        private static int firstOccurence(int a[], int x)
        {
                int start = 0, end = a.length - 1;
                while (start <= end)
                {
                        int mid = (start + end) / 2;
                        if ((mid == 0 || a[mid - 1] < x) && a[mid] == x)
                                return mid;
                        else if (a[mid] >= x)
                                end = mid - 1;
                        else
                                start = mid + 1;
                }
                return -1;
        }

        // go right if a[mid] is equal to search element
        private static int lastOccurence(int a[], int i, int x)
        {
                int start = i, end = a.length - 1;
                while (start <= end)
                {
                        int mid = (start + end) / 2;
                        if ((mid == a.length - 1 || a[mid + 1] > x) && a[mid] == x)
                                return mid;
                        else if (a[mid] > x)
                                end = mid - 1;
                        else
                                start = mid + 1;
                }
                return -1;
        }

        public static void main(String[] args)
        {
                int[] a = { 2, 2, 2, 2, 2, 7, 8, 9 };
                System.out.println(countOccurences(a, 2));
        }
}
```

```java
package problems;

public class DecimalValue
{
        public static void main(String[] args)
        {
                int[][] input = { { 0, 1, 0 }, { 1, 1, 0 }, { 0, 0, 1 } };
                int max = 0;
                for (int i = 0; i < input.length; i++)
                {
                        int tmp = 0;
                        for (int j = input[i].length - 1; j >= 0; j--)
                        {
                                tmp |= (input[i][j] << (input[i].length - 1 - j));
                                System.out.println(tmp);
                        }
                        if (tmp > max)
                                max = tmp;
                }
                System.out.println("Max: " + max);
        }
}
```

```java
package problems;

public class DecipherMsg
{
        public static void main(String args[])
        {
                // String cipher = "jussDs sfsfs fwfsldfms Atvt hrqgse, Cnikg";
                String cipher = "Li, ailu jw au facntll";
                String plain = decipher(cipher);
                System.out.println(plain);
        }

        private static String decipher(String encrypted_message)
        {
                StringBuilder plain = new StringBuilder("");
                String key = "8251220";
                int keycounter = 0;
                for (int i = 0; i < encrypted_message.length(); i++)
                {
                        char temp = encrypted_message.charAt(i);
                        if (keycounter == key.length())
                                keycounter = 0;
                        if (temp >= 65 && temp <= 90)
                        {
                                char plainChar = (char) (encrypted_message.charAt(i)
                                                - Character.getNumericValue(key.charAt(keycounter)));
                                if (plainChar < 65)
                                {
                                        int diff = 65 - plainChar;
                                        plainChar = (char) (91 - diff);
                                }
                                System.out.println(encrypted_message.charAt(i));
                                System.out.println(key.charAt(keycounter));
                                System.out.println(plainChar);
                                System.out.println();
                                plain.append(plainChar);
                                keycounter++;
                        }
                        else if (temp >= 97 && temp <= 122)
                        {
                                char plainChar = (char) (encrypted_message.charAt(i)
                                                - Character.getNumericValue(key.charAt(keycounter)));
                                if (plainChar < 97)
                                {
                                        int diff = 97 - plainChar;
                                        plainChar = (char) (123 - diff);
                                }
                                System.out.println(encrypted_message.charAt(i));
                                System.out.println(key.charAt(keycounter));
                                System.out.println(plainChar);
                                System.out.println();
                                plain.append(plainChar);
                                keycounter++;
                        }
                        else
                        {
                                plain.append(temp);
                        }
                }
                return plain.toString();
        }
}
```

```java
package problems;

public class EquilibriumIndex
{
        public static int solution(int[] a)
        {
                long sumL = 0, sumR = 0;
                int len = a.length;
                for (int i = 0; i < len; i++)
                        sumR += a[i];
                for (int i = 0; i < len; i++)
                {
                        sumR -= a[i];
                        if (sumL == sumR)
                                return i;
                        sumL += a[i];
                }
                return -1;
        }

        public static void main(String[] args)
        {
                int[] a = { -1, 3, -4, 5, 1, -6, 2, 1 };
                System.out.println(solution(a));

        }

}
```

```java
package problems;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;
import java.util.Set;

class Edge
{
        int dest;
        int weight;

        public Edge(int d, int w)
        {
                dest = d;
                weight = w;
        }
}

class DG
{
        int v;
        List<Edge>[] adj;
        static int freq[];
        static boolean vis[];

        @SuppressWarnings("unchecked")
        public DG(int v)
        {
                this.v = v;
                adj = new ArrayList[v + 1];
                for (int i = 1; i <= v; i++)
                        adj[i] = new ArrayList<Edge>();
                freq = new int[10];
                vis = new boolean[v + 1];
        }

        public void addEdge(int a, int b, int w)
        {
                Edge e = new Edge(b, w);
                adj[a].add(e);
                e = new Edge(a, 1000 - w);
                adj[b].add(e);
        }

        public void findAllPaths()
        {
                for (int i = 1; i <= v; i++)
                {
                        vis = new boolean[v + 1];
                        Set<Integer> proc = new HashSet<Integer>();
                        findAllPaths(i, i, 0, proc);
                }
                for (int i = 0; i < 10; i++)
                        System.out.println(freq[i]);
        }

        public void findAllPaths(int src, int curr, int w, Set<Integer> proc)
        {
                vis[curr] = true;
                for (Edge e : adj[curr])
                {
                        if (e.dest == src || e.dest == curr || proc.contains(e.dest))
                                continue;
                        // System.out.println(w + e.weight);
                        freq[(w + e.weight) % 10]++;
                        proc.add(e.dest);
                        if (!vis[e.dest])
                                findAllPaths(src, e.dest, w + e.weight, proc);
                        if (curr == src)
                        {
                                proc.clear();
                                vis = new boolean[v + 1];
                        }
                }
        }
}

public class FindAllPathsDG
{
        public static void main(String[] args)
        {
                Scanner in = new Scanner(System.in);
                int n = in.nextInt();
                DG g = new DG(n);
                int e = in.nextInt();
                for (int a0 = 0; a0 < e; a0++)
                {
                        int x = in.nextInt();
                        int y = in.nextInt();
                        int r = in.nextInt();
                        g.addEdge(x, y, r);
                }
                g.findAllPaths();
                in.close();
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.List;

public class FindAnagrams
{
        static List<Integer> findAnagrams(String s, String p)
        {
                List<Integer> list = new ArrayList<>();
                if (s == null || s.length() == 0 || p == null || p.length() == 0)
                        return list;

                int[] freq = new int[256];

                for (char c : p.toCharArray())
                        freq[c]++;

                int left = 0, right = 0, count = p.length();
                while (right < s.length())
                {
                        if (freq[s.charAt(right)] >= 1)
                                count--;
                        freq[s.charAt(right)]--;
                        right++;

                        if (count == 0)
                                list.add(left);

                        if (right - left == p.length())
                        {
                                // if we are throwing away a valid char, increase the count
                                // a char is valid if it occurs in p
                                if (freq[s.charAt(left)] >= 0)
                                        count++;
                                // restore the count of left most char before throwing it away
                                freq[s.charAt(left)]++;
                                left++;
                        }
                }
                return list;
        }

        public static void main(String[] args)
        {
                // System.out.println(findAnagrams("abab", "ab"));
                System.out.println(findAnagrams("caababaaaa", "aaba"));
                // System.out.println(findAnagrams("bbbbcacaa", "aaba"));
        }

}
```

```java
package problems;

public class FindDuplicate
{
        static int findDuplicate(int[] nums)
        {
                int l = 1, r = nums.length - 1;

                while (l < r)
                {
                        int m = l + (r - l) / 2;
                        int count = 0;

                        for (int a : nums)
                        {
                                if (a <= m)
                                        count++;
                        }

                        // System.out.println("mid: " + m + " count: " + count);
                        if (count > m)
                                r = m;
                        else
                                l = m + 1;
                }
                return l;
        }

        static int findDuplicateFast(int[] nums)
        {
                /*
                 * since the array contains numbers only between [1..N], the array will
                 * have atleast one cycle. starting at 0 is the key. since the array has
                 * no 0, the cycle we encounter starting at 0 will not be based on the
                 * number at 0th index, i.e. it will definitely contain the duplicate
                 * element, which will be the starting node of cycle.
                 */
                int slow = nums[0], fast = nums[slow];

                while (slow != fast)
                {
                        slow = nums[slow];
                        fast = nums[nums[fast]];
                }

                slow = 0;
                while (fast != slow)
                {
                        fast = nums[fast];
                        slow = nums[slow];
                }
                return slow;
        }

        public static void main(String args[])
        {
                int a[] = { 4, 1, 3, 2, 6, 7, 5, 1 };
                System.out.println(findDuplicateFast(a));
        }

}
```

```java
package problems;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class FindMutation
{
        public static void main(String args[])
        {
                String bank[] = { "AAAAAAAA", "AAAAAAAT", "AAAAAATT", "AAAAATTT" };
                String start = "AAAAAAAA";
                String end = "AAAAATTT";
                int dist = minMutation(start, end, bank);
                System.out.println(dist);
        }

        public static int minMutation(String start, String end, String[] bank)
        {
                if (bank == null || start == null || start.length() == 0 || end == null || end.length() == 0)
                        return 0;

                Set<String> bankStrings = new HashSet<String>();
                for (String s : bank)
                        bankStrings.add(s);

                if (!bankStrings.contains(end))
                        return -1;

                int count = minMutation(start, end, bankStrings);
                return count == 0 ? -1 : count;
        }

        public static int minMutation(String start, String end, Set<String> bankStrings)
        {
                List<Integer> indices = new ArrayList<>();
                int length = start.length();
                for (int i = 0; i < length; ++i)
                {
                        if (start.charAt(i) != end.charAt(i))
                                indices.add(i);
                }
                int curCount = 0;
                String temp = "";
                for (int i : indices)
                {
                        temp = start.substring(0, i) + "" + end.charAt(i) + "" + start.substring(i + 1, length);
                        if (bankStrings.contains(temp))
                                curCount = 1 + minMutation(temp, end, bankStrings);
                }
                return curCount;
        }
}
```

```
package problems;

import java.util.Arrays;

public class Frog
{
        public static int minSecondsToCross(int A[], int X, int D)
        {
                if (D >= X)
                        return 0;

                int finalMin = -1, localMin = A.length, localStart = 0, finalPos = 0, loopIdx = 0;
                int[] times = new int[X];
                Arrays.fill(times, -1);

                // Storing the seconds in an array indexed by position of the leaf
                for (; loopIdx < A.length; loopIdx++)
                {
                        // if multiples leaves fall in same position, consider only the leaf
                        // that fell first
                        if (times[A[loopIdx]] >= 0)
                                continue;

                        times[A[loopIdx]] = loopIdx;
                }

                /*
                 * Idea: 1. Split the pond into equal gaps of width ’D’ The frog can
                 * jump only if there is at least one leaf in all these gaps. 2. Find
                 * minimum times at which a leaf fall in each of these gaps 3. Maximum
                 * of the set of minimum times found in step 2 would be the required
                 * answer.
                 */
                for (loopIdx = 0; loopIdx < times.length; loopIdx++)
                {
                        if ((localStart + D + 1) == loopIdx)
                        {
                                localStart = loopIdx - 1; // mark the start of each gap

                                if (finalMin < localMin)
                                        finalMin = localMin;

                                localMin = A.length; // reset localMin at the start of each gap
                        }

                        if (times[loopIdx] < 0)
                                continue; // continue if there is no leaf at this position

```

```java
                        if (finalPos == 0 && loopIdx > D) // no leaf to start-off, so return
                                                          // -1
                                return -1;

                        if (localMin > times[loopIdx])
                                localMin = times[loopIdx];

                        if (loopIdx > finalPos && loopIdx - finalPos <= D)
                        {
                                finalPos = loopIdx;
                                if (finalPos + D >= X)
                                        break;
                        }

                }

                if (localMin < A.length && finalMin < localMin)
                        finalMin = localMin;

                if (finalPos + D >= X)
                        return finalMin;

                return -1;
        }

        public static void main(String[] args)
        {
                int X = 4, D = 2; // X is diameter of the pond. D is the max distance
                                  // the frog can caver in a single jump
                int A[] = { 2, 2, 2, 2, 2, 2 }; // A[k] denotes the position at which a
                                                // leaf fall in kth second

                // Given the above parameters, find the minimum seconds in which the
                // frog can jump across the pond
                int ans = minSecondsToCross(A, X, D);
                if (ans < 0)
                {
                        System.out.println("Frog can’t cross the pond");
                }
                else
                {
                        System.out.println("Frog can cross the pond in " + ans + " seconds");
                }
        }
}
```

```java
package problems;

import java.util.LinkedList;
import java.util.List;

@SuppressWarnings("unchecked")
public class Graph
{
        int vCount;
        List<Integer>[] adj;

        public Graph()
        {}

        public Graph(int v)
        {
                vCount = v;
                adj = new LinkedList[v];
                for (int i = 0; i < v; i++)
                        adj[i] = new LinkedList<Integer>();
        }

        public void addEdge(int a, int b)
        {
                adj[a].add(b);
                adj[b].add(a);
        }
}

class DirectedGraph extends Graph
{

        public DirectedGraph(int v)
        {
                super(v);
        }

        @Override
        public void addEdge(int a, int b)
        {
                adj[a].add(b);
        }

}
```

```java
package problems;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.Scanner;

public class GraphBFS
{
        public static void printReach(Graph g, int v, int st)
        {
                int reach[] = new int[v + 1];
                boolean visited[] = new boolean[v + 1];
                LinkedList<Integer> q = new LinkedList<Integer>();
                q.offer(st);
                visited[st] = true;
                int level = 1, currNodeCount = 1, newNodeCount = 0;
                while (!q.isEmpty())
                {
                        int s = q.poll();
                        currNodeCount--;
                        Iterator<Integer> it = g.adj[s].iterator();
                        while (it.hasNext())
                        {
                                int n = it.next();
                                if (!visited[n])
                                {
                                        reach[n] += (6 * level);
                                        visited[n] = true;
                                        q.offer(n);
                                        newNodeCount++;
                                }
                        }
                        if (currNodeCount == 0)
                        {
                                level++;
                                currNodeCount = newNodeCount;
                                newNodeCount = 0;
                        }
                }
                for (int i = 1; i <= v; i++)
                {
                        if ((i != st))
                        {
                                if (reach[i] != 0)
                                        System.out.print(reach[i] + " ");
                                else if (reach[i] == 0)
                                        System.out.print("-1 ");
                        }
                }
                System.out.println();
        }

        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                int q = sc.nextInt();
                for (int i = 0; i < q; i++)
                {
                        int v = sc.nextInt();
                        Graph g = new Graph(v);
                        int e = sc.nextInt();
                        for (int j = 0; j < e; j++)
                                g.addEdge(sc.nextInt(), sc.nextInt());
                        int st = sc.nextInt();
                        printReach(g, v, st);
                }
                sc.close();
        }
}
```

```java
package problems;

public class GraphDFS
{
        static void DFSUtil(DirectedGraph g, int v, boolean[] visited)
        {
                visited[v] = true;
                System.out.print(v + " ");
                for (int neighbor : g.adj[v])
                {
                        if (!visited[neighbor])
                                DFSUtil(g, neighbor, visited);
                }
        }

        static void DFS(DirectedGraph g, int v)
        {
                boolean visited[] = new boolean[v];
                for (int i = 0; i < v; i++)
                {
                        if (!visited[i])
                                DFSUtil(g, i, visited);
                }
        }

        static void DFS(DirectedGraph g, int v, int n)
        {
                boolean visited[] = new boolean[v];
                if (!visited[n])
                        DFSUtil(g, n, visited);
        }

        public static void main(String[] args)
        {
                DirectedGraph g = new DirectedGraph(6);
                g.addEdge(5, 2);
                g.addEdge(5, 0);
                g.addEdge(4, 0);
                g.addEdge(4, 1);
                g.addEdge(2, 3);
                g.addEdge(3, 1);

                System.out.println("Depth First Traversal");
                DFS(g, 6);
                System.out.println();
                System.out.println("Depth First Traversal from 5: ");
                DFS(g, 6, 5);
        }

}
```

```java
package problems;

import java.util.Scanner;

public class HomeValue
{
        /*
         * diff denotes # of increasing subranges - # of decreasing subranges in a
         * window of given size
         */
        static long diff = 0;

        /*
         * enum Direction is to denote if the subrange is increasing or decreasing
         * or none (i.e. when all elements of subarray are equal)
         */
        enum Direction
        {
                NONE, DECR, INCR;
        }

        static int updateDiff(int start, int end, Direction dir)
        {
                /*
                 * Variable subArrays will hold # of inc/dec subranges between start and
                 * end index. It’s calculated based on the formula that a set of size N
                 * has N(N+1)/2 subarrays. But here we don’t need subsets of size 1.
                 * Thus N(N-1)/2 subsets.
                 */
                long size = end - start + 1;
                long subArrays = (size * (size - 1)) / 2;

                /* if the prev subrange is positive then add subArrays to diff */
                if (dir == Direction.INCR)
                        diff += subArrays;
                /* if the prev subrange is negative then subtract subArrays from diff */
                else if (dir == Direction.DECR)
                        diff -= subArrays;

                /* return the next start position */
                return end;
        }

        public static void calcWindowDiff(int[] a, int k)
        {
                if (a == null || a.length == 0 || k <= 0 || k > a.length)
                {
                        System.out.println("Invalid Input!");
                        return;
                }

                int left = 0, right = k - 1, n = a.length;

                /* start denotes start of an inc/dec subrange */
                int start = 0;

                Direction dir = Direction.NONE;

                for (; right < n; right++, left++)
                {
                        start = left;
                        diff = 0;
                        for (int i = left; i < right; i++)
                        {
                                if (a[i] < a[i + 1])
                                {
                                        /*
                                         * if subrange starts increasing then update diff with # of
                                         * decreasing subranges seen so far
                                         */
                                        if (dir == Direction.DECR)
                                                start = updateDiff(start, i, dir);
                                        dir = Direction.INCR;
                                }
                                else if (a[i] > a[i + 1])
                                {
                                        /*
                                         * if subrange starts decreasing then update diff with # of
                                         * increasing subranges seen so far
                                         */
                                        if (dir == Direction.INCR)
                                                start = updateDiff(start, i, dir);
                                        dir = Direction.DECR;
                                }
                                else
                                {
                                        /*
                                         * if numbers in subrange are equal then update diff with #
                                         * of decreasing/increasing subranges seen so far
                                         */
                                        start = updateDiff(start, i, dir) + 1;
                                        dir = Direction.NONE;
                                }

                                /*
                                 * if we reach the end of a window then update diff with # of
                                 * decreasing/increasing subranges seen so far
                                 */
                                if (i == right - 1)
                                {
                                        updateDiff(start, right, dir);
                                        dir = Direction.NONE;
                                }
                        }
                        System.out.println(diff);
                }
        }

        public static void main(String[] args)
        {
                Scanner input = new Scanner(System.in);
                int n = input.nextInt();
                int k = input.nextInt();
                int a[] = new int[n];
                for (int i = 0; i < n; i++)
                        a[i] = input.nextInt();
                calcWindowDiff(a, k);
                input.close();
        }

}
```

```java
package problems;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class JavaStream
{
        public static void main(String args[])
        {
                List<Integer> myList = Arrays.asList(1, 2, 3, 4, 5, 6);
                List<Integer> list = myList.parallelStream().map(a -> a + 1).collect(Collectors.toList());
                System.out.println(Arrays.toString(list.toArray()));
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;

/* class Point implements Comparable<Point>
{
        double x, y;

        public Point()
        {
                x = 0;
                y = 0;
        }

        public Point(double a, double b)
        {
                x = a;
                y = b;
        }

        public double distFromOrigin()
        {
                return Math.hypot(x, y);
        }

        @Override
        public int compareTo(Point o)
        {
                if (this.distFromOrigin() < o.distFromOrigin())
                        return 1;
                else if (this.distFromOrigin() > o.distFromOrigin())
                        return -1;
                return 0;
        }
} */

class Point
{
        double x, y;

        public Point()
        {
                x = 0;
                y = 0;
        }

        public Point(double a, double b)
        {
                x = a;
                y = b;
        }

        public double distFromOrigin()
        {
                return Math.hypot(x, y);
        }
}

public class KNearestPoints
{
        static void findKNearestPoints(List<Point> points, int k)
        {
                PriorityQueue<Point> maxHeap = new PriorityQueue<Point>(k, new Comparator<Point>()
                {
                        @Override
                        public int compare(Point o1, Point o2)
                        {
                                if (o1.distFromOrigin() > o2.distFromOrigin())
                                        return -1;
                                if (o1.distFromOrigin() < o2.distFromOrigin())
                                        return 1;
                                return 0;
                        }
                });
                for (Point p : points)
                {
                        if (maxHeap.size() < k)
                        {
                                maxHeap.offer(p);
                        }
                        else
                        {
                                if (maxHeap.peek().distFromOrigin() > p.distFromOrigin())
                                {
                                        maxHeap.poll();
                                        maxHeap.offer(p);
                                }
                        }
                }
                while (maxHeap.size() > 0)
                {
                        Point p = maxHeap.poll();
                        System.out.println(p.x + "," + p.y);
                }
        }

        public static void main(String[] args)
        {
                List<Point> points = new ArrayList<Point>();
                points.add(new Point(1, 1));
                points.add(new Point(1, 2.5));
                points.add(new Point(-1, 1.4));
                points.add(new Point(-1, 2));
                points.add(new Point(1.5, -1.5));
                points.add(new Point(1.6, -1));
                points.add(new Point(-1, -1.5));
                points.add(new Point(-1, 3));
                points.add(new Point(2, 2));
                findKNearestPoints(points, 5);
        }
}
```

```java
package problems;

import java.util.Arrays;

class KGraph
{
        class KEdge implements Comparable<KEdge>
        {
                int src, dst, weight;

                @Override
                public int compareTo(KEdge that)
                {
                        return this.weight - that.weight;
                }
        }

        class KSubset
        {
                int parent, rank;
        }

        int V, E;
        KEdge[] edge;

        public KGraph(int v, int e)
        {
                V = v;
                E = e;
                edge = new KEdge[E];
                for (int i = 0; i < E; i++)
                        edge[i] = new KEdge();
        }

        public int find(KSubset[] subset, int i)
        {
                if (subset[i].parent != i)
                        subset[i].parent = find(subset, subset[i].parent);
                return subset[i].parent;
        }

        public void union(KSubset[] subset, int x, int y)
        {
                int xParent = find(subset, x);
                int yParent = find(subset, y);

                if (subset[xParent].rank < subset[yParent].rank)
                {
                        subset[xParent].parent = yParent;
                }
                else if (subset[yParent].rank < subset[xParent].rank)
                {
                        subset[yParent].parent = xParent;
                }
                else
                {
                        subset[xParent].parent = yParent;
                        subset[yParent].rank++;
                }
        }

        public void kruskalMST()
        {
                int i = 0;
                int e = 0;
                KEdge mst[] = new KEdge[E];
                for (i = 0; i < E; i++)
                        mst[i] = new KEdge();

                Arrays.sort(edge);

                KSubset subset[] = new KSubset[V];
                for (i = 0; i < V; i++)
                {
                        subset[i] = new KSubset();
                        subset[i].parent = i;
                        subset[i].rank = 0;
                }
                i = 0;
                while (e < V - 1)
                {
                        KEdge curr_edge = edge[i++];

                        int x = find(subset, curr_edge.src);
                        int y = find(subset, curr_edge.dst);
                        if (x != y)
                        {
                                mst[e++] = curr_edge;
                                union(subset, x, y);
                        }
                }

                for (i = 0; i < E; i++)
                {
                        System.out.println("Edge " + mst[i].src + "-->" + mst[i].dst + " (" + mst[i].weight + ")");
                }
        }
}

public class KruskalMST
{
        public static void main(String[] args)
        {
                int V = 4; // Number of vertices in graph
                int E = 5; // Number of edges in graph
                KGraph graph = new KGraph(V, E);

                // add edge 0-1
                graph.edge[0].src = 0;
                graph.edge[0].dst = 1;
                graph.edge[0].weight = 10;

                // add edge 0-2
                graph.edge[1].src = 0;
                graph.edge[1].dst = 2;
                graph.edge[1].weight = 6;

                // add edge 0-3
                graph.edge[2].src = 0;
                graph.edge[2].dst = 3;
                graph.edge[2].weight = 5;

                // add edge 1-3
                graph.edge[3].src = 1;
                graph.edge[3].dst = 3;
                graph.edge[3].weight = 15;

                // add edge 2-3
                graph.edge[4].src = 2;
                graph.edge[4].dst = 3;
                graph.edge[4].weight = 4;

                graph.kruskalMST();
        }
}
```

```java
package problems;

import java.util.LinkedList;
import java.util.Queue;

public class LargestSubTree
{
        static int getLeftMostChild(TreeNode root)
        {
                TreeNode temp = root;
                while (temp.left != null)
                        temp = temp.left;
                return temp.data;
        }

        static int getRightMostChild(TreeNode root)
        {
                TreeNode temp = root;
                while (temp.right != null)
                        temp = temp.right;
                return temp.data;
        }

        static TreeNode getLargeRoot(TreeNode root, int i, int j)
        {
                if (root == null)
                        return null;
                Queue<TreeNode> q = new LinkedList<TreeNode>();
                q.offer(root);
                while (!q.isEmpty())
                {
                        TreeNode curr = q.poll();
                        if (curr.data >= i && curr.data <= j)
                        {
                                int lMost = getLeftMostChild(curr);
                                int rMost = getRightMostChild(curr);
                                if (lMost >= i && rMost <= j)
                                        return curr;
                        }
                        if (curr.left != null)
                                q.offer(curr.left);
                        if (curr.right != null)
                                q.offer(curr.right);
                }
                return null;
        }

        public static void main(String[] args)
        {
                BinaryTree tree = new BST();
                tree.root = tree.insert(tree.root, 10);
                tree.root = tree.insert(tree.root, 6);
                tree.root = tree.insert(tree.root, 8);
                tree.root = tree.insert(tree.root, 14);
                tree.root = tree.insert(tree.root, 16);
                tree.root = tree.insert(tree.root, 9);
                tree.root = tree.insert(tree.root, 5);
                tree.root = tree.insert(tree.root, 12);
                tree.root = tree.insert(tree.root, 7);

                TreeNode largeRoot = getLargeRoot(tree.root, 5, 6);
                if (largeRoot != null)
                        System.out.println(largeRoot.data);
                else
                        System.out.println("No such tree is possible...");
        }
}
```

```java
package problems;

import java.util.HashMap;
import java.util.Map;

public class LinkedListRandPtr
{
        public RandomListNode copyRandomList(RandomListNode head)
        {
                if (head == null)
                        return null;

                Map<RandomListNode, RandomListNode> map = new HashMap<RandomListNode, RandomListNode>();
                RandomListNode temp = head;

                while (temp != null)
                {
                        RandomListNode node = new RandomListNode(temp.label);
                        map.put(temp, node);
                        temp = temp.next;
                }

                RandomListNode cpy_temp = null;
                temp = head;
                while (temp != null)
                {
                        cpy_temp = map.get(temp);
                        cpy_temp.next = map.get(temp.next);
                        cpy_temp.random = map.get(temp.random);
                        temp = temp.next;
                }

                return map.get(head);
        }

        public static void main(String[] args)
        {

        }

}
```

```java
package problems;

public class LinkedListToBST
{
        static TreeNode convertSLLtoBST(MyLinkedList<Integer> list)
        {
                if (list.head == null)
                        return null;

                return recurSLLtoBST(list, 0, list.length() - 1);
                // return recurSLLtoBST(list, list.length());

        }

        static TreeNode recurSLLtoBST(MyLinkedList<Integer> list, int low, int high)
        {
                if (low > high)
                        return null;

                int mid = (low + high) / 2;
                TreeNode left = recurSLLtoBST(list, low, mid - 1);
                TreeNode root = new TreeNode(list.head.data);
                root.left = left;
                list.head = list.head.next;
                root.right = recurSLLtoBST(list, mid + 1, high);
                return root;
        }

        public static void main(String[] args)
        {
                MyLinkedList<Integer> list = new MyLinkedList<Integer>();
                list.insertAtEnd(1);
                list.insertAtEnd(2);
                list.insertAtEnd(3);
                list.insertAtEnd(4);
                list.insertAtEnd(5);
                list.insertAtEnd(6);
                list.insertAtEnd(7);
                list.insertAtEnd(8);
                list.printList();

                TreeNode root = convertSLLtoBST(list);
                BinaryTree.printInOrder(root);
        }

}
```

```java
package problems;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class LongestDirectoryPath
{
        public static int longestDirPath(String s)
        {
                Map<Integer, Integer> dirMap = new HashMap<Integer, Integer>();
                int maxLen = 0;
                for (String entry : s.split("\n"))
                {
                        if (entry.isEmpty())
                                continue;

                        int currLevel = 0, currLen = 0;
                        String dirEntry = entry.replaceAll("^\\s+", "");
                        int len = dirEntry.length();
                        currLevel = entry.length() - len;
                        if (dirEntry.indexOf('.') == -1)
                        {
                                currLen = ((currLevel == 0) ? 0 : (dirMap.get(currLevel - 1) + 1)) + len;
                                dirMap.put(currLevel, currLen);
                        }
                        else
                        {
                                maxLen = Math.max(maxLen,
                                                (dirMap.containsKey(currLevel - 1) ? (dirMap.get(currLevel - 1) + 1) + len : len));
                        }
                }
                return maxLen;
        }

        public static void main(String[] args)
        {
                Scanner sc = null;
                try
                {
                        sc = new Scanner(new File("/Users/Anand/Documents/input.txt"));
                        StringBuilder sb = new StringBuilder();
                        while (sc.hasNext())
                        {
                                sb.append(sc.nextLine());
                                sb.append(System.lineSeparator());
                        }
                        System.out.println(sb.toString());
                        System.out.println(longestDirPath(sb.toString()));
                }
                catch (FileNotFoundException e)
                {
                        System.out.println("Error reading input: " + e.getMessage());
                }
                finally
                {
                        sc.close();
                }
        }
}
```

```java
package problems;

public class LongestPalindromicChunks
{
        static int longestPalindrome(String s)
        {
                if (s.length() == 0)
                        return 0;

                int inpLen = s.length();
                int start = 0, end = inpLen, chunkCount = 0, matchedLen = 0;

                for (int i = 1; i <= inpLen / 2; i++)
                {
                        if (s.substring(start, i).equals(s.substring(inpLen - i, end)))
                        {
                                chunkCount += 2;
                                int len = s.substring(start, i).length();
                                matchedLen += (2 * len);
                                start += len;
                                end -= len;
                        }
                }
                if (matchedLen < inpLen)
                        chunkCount++;
                return chunkCount;
        }

        public static void main(String args[])
        {
                System.out.println(longestPalindrome("antaprezatepzapreanta"));
                System.out.println(longestPalindrome("merchant"));
                System.out.println(longestPalindrome("volvo"));
                System.out.println(longestPalindrome("ghiabcdefhelloadamhelloabcdefghi"));

        }
}
```

```java
package problems;

public class LongestPalindromicSubstring
{
        public static String longestPalindrome(String s)
        {
                int len = s.length();
                if (len == 1)
                        return s;
                String longest = "";
                for (int i = 0; i < len; i++)
                {
                        String tmp = getPalindrome(s, i, i);
                        if (tmp.length() > longest.length())
                                longest = tmp;

                        tmp = getPalindrome(s, i, i + 1);
                        if (tmp.length() > longest.length())
                                longest = tmp;
                }
                return longest;
        }

        private static String getPalindrome(String s, int start, int end)
        {
                while (start >= 0 && end < s.length() && (s.charAt(start) == s.charAt(end)))
                {
                        start--;
                        end++;
                }
                return s.substring(start + 1, end);
        }

        public static void main(String[] args)
        {
                String s = "456789zazasxabcdeedcba123";
                System.out.println(longestPalindrome(s));
        }
}
```

```java
package problems;

import java.util.HashSet;
import java.util.Set;

public class LongestSubstring2Unique
{

        static String findLongestSubstring(String input)
        {
                if (input == null || input.length() == 0)
                        return input;
                int len = input.length();
                int globalStart = 0, localStart = 0;
                int maxLen = Integer.MIN_VALUE, currLen = 0;
                char lastChar = 0, lastCharCount = 0;
                /*
                 * HashSet stores 2 unique characters seen so far
                 */
                Set<Character> uniqueChars = new HashSet<Character>();
                for (int i = 0; i < len; i++)
                {
                        char c = input.charAt(i);
                        /*
                         * if the HashSet contains the current character, then increase the
                         * running length.
                         */
                        if (uniqueChars.contains(c))
                        {
                                currLen++;
                        }
                        else
                        {
                                /*
                                 * if the current character is a new character, then add it to
                                 * HashSet.
                                 */
                                if (uniqueChars.size() < 2)
                                {
                                        uniqueChars.add(c);
                                        currLen++;
                                }
                                /*
                                 * If the new character is 3rd unique character, then update the
                                 * HashSet, globalStart and maxLen accordingly.
                                 */
                                else
                                {
                                        uniqueChars.clear();
                                        uniqueChars.add(lastChar);
                                        uniqueChars.add(c);
                                        if (maxLen < currLen)
                                        {
                                                maxLen = currLen;
                                                globalStart = localStart;
                                                currLen = lastCharCount + 1;
                                                localStart = i - lastCharCount;
                                        }
                                }
                        }
                        if (c != lastChar)
                        {
                                lastChar = c;
                                lastCharCount = 0;
                        }
                        lastCharCount++;
                }
                /*
                 * include the character at end of the string to our running length and
                 * global length if needed.
                 */
                if (maxLen < currLen)
                {
                        maxLen = currLen;
                        globalStart = localStart;
                }
                return input.substring(globalStart, globalStart + maxLen);
        }

        public static void main(String[] args)
        {
                System.out.println(findLongestSubstring("aabaaaccacccadef"));
        }

}
```

```java
package problems;

import java.util.HashMap;
import java.util.Map;

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;
        Map<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);
```

```java
                setHead(created);

            }
            else
            {
                setHead(created);
            }

            map.put(key, created);
        }
    }
}
```

```java
package problems;

import java.util.Arrays;

public class MagicIndex
{
        /*
         * provide different solution for Dup and NoDup variations. NoDup solution
         * is very fast O(logn). Dup solution will work fast only for arrays with
         * duplicates. For non-dup arrays, it is equivalent to linear search.
         */
        // static int i = 1;

        static int magicIndex(int[] a, int low, int high)
        {
                if (low > high)
                        return -1;

                // System.out.println(i++ + ": " + low + " " + high);

                int mid = low + (high - low) / 2;
                if (a[mid] == mid)
                        return mid;

                int lIndex = magicIndex(a, low, Math.min(a[mid], mid - 1));
                if (lIndex >= 0)
                        return lIndex;

                int rIndex = magicIndex(a, Math.max(a[mid], mid + 1), high);
                return rIndex;
        }

        static int magicIndexNoDup(int[] a, int low, int high)
        {
                while (low <= high)
                {
                        // System.out.println(i++ + ": " + low + " " + high);

                        int mid = low + (high - low) / 2;
                        if (a[mid] == mid)
                                return mid;
                        else if (a[mid] < mid)
                                low = mid + 1;
                        else
                                high = mid - 1;
                }
                return -1;
        }

        public static void main(String[] args)
        {
                int a[] = { -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13 };
                System.out.println(Arrays.toString(a));
                System.out.println(magicIndex(a, 0, a.length - 1));
                // i = 0;
                System.out.println(magicIndexNoDup(a, 0, a.length - 1));
        }
}
```

```java
package problems;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

public class MagicTime
{
        public static boolean IsMagicNumber(long num)
        {
                boolean[] isPresent = new boolean[10];
                int count = 0;
                while (num > 0)
                {
                        int digit = (int) (num % 10);
                        if (!isPresent[digit])
                        {
                                isPresent[digit] = true;
                                count++;
                        }
                        num /= 10;
                }
                return count == 2 ? true : false;
        }

        public static void printMagicTime(String a, String b) throws ParseException
        {
                DateFormat df = new SimpleDateFormat("yyyy/MM/dd HH:mm");
                Date d1 = df.parse(a);
                Date d2 = df.parse(b);
                df = new SimpleDateFormat("yyyyMMddHHmm");
                Calendar cal = new GregorianCalendar();
                cal.setTime(d1);
                while (cal.getTime().before(d2))
                {
                        long tempDate = Long.parseLong(df.format(cal.getTime()));
                        if (IsMagicNumber(tempDate))
                                System.out.println(cal.getTime());
                        cal.add(Calendar.MINUTE, 1);
                }
        }

        public static void main(String[] args)
        {
                try
                {
                        printMagicTime("1100/10/05 12:07", "1110/01/20 12:09");
                }
                catch (ParseException e)
                {
                        e.printStackTrace();
                }
        }

}
```

```java
package problems;

import java.util.*;
import java.util.LinkedList;;

class FNode
{

        public String data;
        public FNode left = null;
        public FNode right = null;

        public FNode(String d)
        {
                data = d;
        }
}

public class Main
{
        FNode root;

        public Main()
        {
                root = null;
        }

        public FNode find(FNode root, String d)
        {
                if (root == null)
                        return null;

                if (root.data.equals(d))
                        return root;

                FNode temp;

                temp = find(root.left,d);
                if(temp != null)
                        return temp;
                temp = find(root.right,d);
                return temp;
        }

        public FNode insert(FNode root, String parent, String d)
        {
                FNode newNode = new FNode(d);
                if(root == null)
                {
                        FNode newParent = new FNode(parent);
                        root = newParent;
                        root.left = newNode;
                        return root;
                }

                FNode temp = find(root, parent);

                if(temp.left == null)
                        temp.left = newNode;
                else
                        temp.right = newNode;

                return root;
        }

        public void BFS(FNode root, String d)
        {
                if(root == null) return;

                Queue<FNode> q = new LinkedList<FNode>();
                Set<String> level = new TreeSet<String>();
                int currLevel = 0, nextLevel = 0;
                FNode temp = root;

                q.offer(temp);
                currLevel++;
                level.add(temp.data);

                if(level.contains(d))
                {
                        printSiblings(level);
                        return;
                }

                level.clear();
                while(!q.isEmpty())
                {
                        temp = q.poll();
                        currLevel--;
                        if(temp.left != null)
                        {
                                q.offer(temp.left);
                                nextLevel++;
                                level.add(temp.left.data);
                        }
                        if(temp.right != null)
                        {
                                q.offer(temp.right);
                                nextLevel++;
                                level.add(temp.right.data);
                        }
                        if(currLevel == 0)
                        {
                                currLevel = nextLevel;
                                nextLevel = 0;
                                if(level.contains(d))
                                {
                                        printSiblings(level);
                                        return;
                                }
                                level.clear();
                        }
                }
        }

        public void printSiblings(Set<String> set)
        {
                StringBuilder sb = new StringBuilder();
                for(String a : set)
                        sb.append(a + ",");
                sb.setLength(sb.length()-1);
                System.out.println(sb.toString());
        }

        public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                Main tree = new Main();
                String s = "";
                while(sc.hasNext())
                {
                        s = sc.nextLine();
                        System.out.println(s);
                        String[] s1 = s.split(",");
                        String person = s1[s1.length-1];
                        for(int i=0;i<s1.length-1;i++)
                        {
                                String parent = s1[i].split("->")[0];
                                String child = s1[i].split("->")[1];
                                tree.root = tree.insert(tree.root, parent, child);
                        }
                        tree.BFS(tree.root,person);
                }
                sc.close();
        }
}
```

```
package problems;

public class MaxHeap
{
        private int[] Heap;
        private int size;
        private int maxsize;

        private static final int FRONT = 1;

        public MaxHeap(int maxsize)
        {
                this.maxsize = maxsize;
                this.size = 0;
                Heap = new int[this.maxsize + 1];
                Heap[0] = Integer.MAX_VALUE;
        }

        private int parent(int pos)
        {
                return pos / 2;
        }

        private int leftChild(int pos)
        {
                return (2 * pos);
        }

        private boolean isLeaf(int pos)
        {
                if (pos > (size / 2) && pos <= size)
                {
                        return true;
                }
                return false;
        }

        private void swap(int fpos, int spos)
        {
                int tmp;
                tmp = Heap[fpos];
                Heap[fpos] = Heap[spos];
                Heap[spos] = tmp;
        }

        private void maxHeapify(int pos)
        {
                while (!isLeaf(pos))
```

```java
                {
                        int newPos = leftChild(pos);
                        if ((newPos < size) && Heap[newPos + 1] > Heap[newPos])
                                newPos++; // move to right child
                        if (Heap[newPos] < Heap[pos])
                                break;
                        swap(pos, newPos);
                        pos = newPos;
                }
        }

        public void insert(int element)
        {
                Heap[++size] = element;
                int current = size;
                while (Heap[current] > Heap[parent(current)])
                {
                        swap(current, parent(current));
                        current = parent(current);
                }
        }

        public void maxHeap() // this function is just to form a maxheap from
                                                        // existing array
        {
                for (int pos = (size / 2); pos >= 1; pos--)
                        maxHeapify(pos);
        }

        public int remove()
        {
                int popped = Heap[FRONT];
                if (size == FRONT)
                {
                        size--;
                }
                else
                {
                        Heap[FRONT] = Heap[size--];
                        maxHeapify(FRONT);
                }
                return popped;
        }

        public void removeAll()
        {
                while (size > 0)
                        System.out.print(remove() + " ");
                System.out.println();
        }

        public static void main(String... arg)
        {
                System.out.println("The Max Heap is ");
                MaxHeap maxHeap = new MaxHeap(10);
                maxHeap.insert(5);
                maxHeap.insert(3);
                maxHeap.insert(17);
                maxHeap.insert(10);
                maxHeap.insert(84);
                maxHeap.insert(19);
                maxHeap.insert(6);
                maxHeap.insert(22);
                maxHeap.insert(9);
                maxHeap.insert(100);

                maxHeap.removeAll();
        }
}
```

```java
package problems;

import java.util.ArrayDeque;
import java.util.Deque;

public class MaxInSlidingWindow
{
        public static void printMaxInSlidingWindow(int[] a, int n, int k)
        {
                if (a == null || a.length == 0)
                        return;

                Deque<Integer> deq = new ArrayDeque<Integer>();
                int idx = 0;
                for (; idx < k; idx++)
                {
                        while (!deq.isEmpty() && a[idx] >= a[deq.peekLast()])
                                deq.pollLast();
                        deq.offer(idx);
                }

                for (; idx < n; idx++)
                {
                        System.out.print(a[deq.peek()] + " ");

                        // Reason for storing indices in deque:
                        // since index is used here to delete old elements from window.
                        if (!deq.isEmpty() && deq.peek() <= (idx - k))
                                deq.poll();

                        while (!deq.isEmpty() && a[idx] >= a[deq.peekLast()])
                                deq.pollLast();
                        deq.offer(idx);
                }
                if (!deq.isEmpty())
                        System.out.print(a[deq.peek()]);
        }

        public static void understandDeque(int[] a, int n)
        {
                Deque<Integer> deq = new ArrayDeque<Integer>();
                for (int i = 0; i < n; i++)
                {
                        deq.offer(a[i]);
                        System.out.println(deq);
                }
                System.out.println("peekFirst: " + deq.peekFirst());
                System.out.println("peek: " + deq.peek());
                System.out.println("peekLast: " + deq.peekLast());
                System.out.println("poll: " + deq.poll());
                System.out.println("pollLast: " + deq.pollLast());
                System.out.println(deq);
                deq.offer(100);
                System.out.println(deq);
        }

        public static void main(String[] args)
        {
                // Scanner sc = new Scanner(System.in);
                // int n = sc.nextInt();
                // int k = sc.nextInt();
                // int[] a = new int[n];
                int[] a = { 8, 5, 10, 7, 9, 4, 15, 12, 90, 13 };
                // for (int i = 0; i < n; i++)
                // a[i] = sc.nextInt();
                printMaxInSlidingWindow(a, 10, 3);
                // understandDeque(a, 10);
                // sc.close();
        }

}
```

```
package problems;

import java.util.TreeSet;

public class MaxModSumSubarray
{
        static void maxModSumSubarray(long[] a, int m)
        {
                TreeSet<Long> s = new TreeSet<Long>();
                long sum = 0, ans = -1, n = a.length;
                for (int i = 0; i < n; i++)
                {
                        if (i == 0)
                        {
                                sum = a[i] % m;
                                ans = Math.max(ans, sum);
                                s.add(sum);
                        }
                        else
                        {
                                sum = (sum + (a[i] % m)) % m;
                                Long temp = s.higher(sum);
                                ans = Math.max(ans, (sum - ((temp == null) ? 0 : temp) + m) % m);
                                s.add(sum);
                        }
                }
                System.out.println(ans);
        }

        public static void main(String[] args)
        {
                long a[] = { 3, 3, 9, 9, 5 };
                maxModSumSubarray(a, 7);
        }
}
```

```java
package problems;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

public class MaxSet
{
        public static boolean AddToSet(int[] A, int[] B, int i, int temp)
        {
                if (B[i] > 0)
                        return false;
                B[i] = temp++;
                boolean ret = AddToSet(A, B, A[i], temp);
                return ret;
        }

        public static int solution(int A[], int N)
        {
                if (A.length == 0)
                        return 0;
                int[] B = new int[N];
                Set<Integer> set = new HashSet<Integer>();
                int totalSize = 0;
                for (int i = 0; i < N; i++)
                {
                        if (B[i] > 0)
                                continue;

                        while (AddToSet(A, B, i, 1));
                        System.out.println(set);
                        System.out.println(Arrays.toString(B));

                        totalSize += set.size();
                        System.out.println(totalSize);
                        if (totalSize >= N)
                                break;
                        set.clear();
                }

                int max = -1;
                for (int i = 0; i < B.length; i++)
                {
                        if (B[i] > max)
                                max = B[i];
                }
                return max;
        }

        public static void main(String[] args)
        {
                int A[] = { 5, 4, 1, 0, 3, 6, 2 };
                // int A[] = {5,4,0,3,1,6,2};
                // int A[] = {0,1,2,3,4,5,6};
                System.out.println(solution(A, A.length));
        }
}
```

```java
package problems;

import java.util.Arrays;

public class MaxSumSubarray
{
        static int maxSumSubSeq(int[] nums)
        {
                int globalMax = nums[0];
                int len = nums.length;
                for (int i = 1; i < len; i++)
                        globalMax = Math.max(nums[i], Math.max(globalMax, globalMax + nums[i]));
                return globalMax;
        }

        static int maxSumSubArray(int[] nums)
        {
                int localMax = nums[0];
                int globalMax = nums[0];
                int len = nums.length;
                for (int i = 1; i < len; i++)
                {
                        localMax = Math.max(nums[i], localMax + nums[i]);
                        globalMax = Math.max(globalMax, localMax);
                }
                return globalMax;
        }

        static void maxSubArray(int[] nums)
        {
                int localMax = nums[0];
                int globalMax = nums[0];
                int localStart = 0, globalStart = 0, globalEnd = 0;
                int len = nums.length;
                for (int i = 1; i < len; i++)
                {
                        if (localMax + nums[i] > nums[i])
                        {
                                localMax += nums[i];
                        }
                        else
                        {
                                localMax = nums[i];
                                localStart = i;
                        }

                        if (globalMax < localMax)
                        {
                                globalMax = localMax;
                                globalStart = localStart;
                                globalEnd = i;
                        }
                }
                System.out.println("start: " + globalStart + " end: " + globalEnd);
        }

        public static void main(String[] args)
        {
                int[] a = { -2, 1, -3, 4, -1, 2, 1, -9, 4 };
                System.out.println(Arrays.toString(a));
                System.out.println("Max sum of subarray: " + maxSumSubArray(a));
                maxSubArray(a);
                System.out.println("Max sum of subsequence: " + maxSumSubSeq(a));
        }
}
```

```java
package problems;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class MaxUniqueNumsSlidingWindow
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                Deque<Integer> deque = new ArrayDeque<Integer>();
                Map<Integer, Integer> map = new HashMap<Integer, Integer>();
                int n = sc.nextInt();
                int m = sc.nextInt();
                int max = 0;

                for (int i = 0; i < n; i++)
                {
                        int num = sc.nextInt();
                        deque.offer(num);
                        if (map.containsKey(num))
                                map.put(num, map.get(num) + 1);
                        else
                                map.put(num, 1);

                        if (deque.size() > m)
                        {
                                int head = deque.poll();
                                if (map.get(head) > 1)
                                        map.put(head, map.get(num) - 1);
                                else
                                        map.remove(head);
                        }
                        max = Math.max(max, map.size());
                }
                System.out.println(max);
                sc.close();
        }
}
```

```java
package problems;

public class MaxValueSubTree
{
        static void decorateTree(TreeNode root)
        {
                if (root == null)
                        return;

                decorateTree(root.left);
                decorateTree(root.right);

                if (root.left != null)
                        root.data = Math.max(root.data, root.left.data);
                if (root.right != null)
                        root.data = Math.max(root.data, root.right.data);
        }

        public static void main(String[] args)
        {
                BinaryTree tree = new BST();
                tree.root = tree.insert(tree.root, 10);
                tree.root = tree.insert(tree.root, 6);
                tree.root = tree.insert(tree.root, 8);
                tree.root = tree.insert(tree.root, 14);
                tree.root = tree.insert(tree.root, 12);
                tree.root = tree.insert(tree.root, 16);
                tree.root = tree.insert(tree.root, 9);
                tree.root = tree.insert(tree.root, 5);
                tree.printPaths();
                System.out.println();
                decorateTree(tree.root);
                tree.printPaths();
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

class Interval
{
        int start;
        int end;

        Interval()
        {
                start = 0;
                end = 0;
        }

        Interval(int s, int e)
        {
                start = s;
                end = e;
        }
}

public class MergeIntervals
{
        public List<Interval> merge(List<Interval> intervals)
        {
                List<Interval> ans = new ArrayList<Interval>();

                if (intervals == null || intervals.size() == 0)
                        return ans;

                Collections.sort(intervals, new Comparator<Interval>()
                {
                        @Override
                        public int compare(Interval i1, Interval i2)
                        {
                                if (i1.start != i2.start)
                                        return i1.start - i2.start;
                                else
                                        return i1.end - i2.end;
                        }
                });

                Interval prev = intervals.get(0);
                for (int i = 1; i < intervals.size(); i++)
                {
                        Interval curr = intervals.get(i);
                        if (curr.start > prev.end)
                        {
                                ans.add(prev);
                                prev = curr;
                        }
                        else
                        {
                                Interval merged = new Interval(prev.start, Math.max(prev.end, curr.end));
                                prev = merged;
                        }
                }
                ans.add(prev);
                return ans;
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.PriorityQueue;

public class MergeKSortedLists
{
        public static ListNode<Integer> Merge(ArrayList<ListNode<Integer>> lists)
        {
                if (lists == null || lists.isEmpty())
                        return null;

                PriorityQueue<ListNode<Integer>> minheap = new PriorityQueue<ListNode<Integer>>(lists.size(),
                                new Comparator<ListNode<Integer>>()
                                {
                                        @Override
                                        public int compare(ListNode<Integer> o1, ListNode<Integer> o2)
                                        {
                                                return o1.data - o2.data;
                                        }
                                });

                for (ListNode<Integer> list : lists)
                        minheap.add(list);

                ListNode<Integer> head = minheap.poll();
                ListNode<Integer> end = head;

                while (!minheap.isEmpty())
                {
                        if (end.next != null)
                                minheap.add(end.next);

                        end.next = minheap.poll();
                        end = end.next;
                }

                return head;
        }

        public static void main(String args[])
        {
                ArrayList<ListNode<Integer>> lists = new ArrayList<ListNode<Integer>>();
                MyLinkedList<Integer> l1 = new MyLinkedList<Integer>();
                l1.insertAtEnd(10);
                l1.insertAtEnd(12);
                l1.insertAtEnd(13);
                l1.insertAtEnd(15);
                lists.add(l1.head);
                l1.printList();

                MyLinkedList<Integer> l2 = new MyLinkedList<Integer>();
                l2.insertAtEnd(1);
                l2.insertAtEnd(5);
                l2.insertAtEnd(6);
                l2.insertAtEnd(8);
                lists.add(l2.head);
                l2.printList();

                MyLinkedList<Integer> l3 = new MyLinkedList<Integer>();
                l3.insertAtEnd(2);
                l3.insertAtEnd(3);
                l3.insertAtEnd(4);
                l3.insertAtEnd(9);
                lists.add(l3.head);
                l3.printList();

                MyLinkedList<Integer> l4 = new MyLinkedList<Integer>();
                l4.insertAtEnd(7);
                l4.insertAtEnd(11);
                l4.insertAtEnd(14);
                l4.insertAtEnd(16);
                lists.add(l4.head);
                l4.printList();

                MyLinkedList<Integer> l5 = new MyLinkedList<Integer>();
                l5.head = Merge(lists);
                l5.printList();
        }
}
```

```java
package problems;

import java.util.Arrays;

public class MergeSort
{
        static void mergeSort(int[] a)
        {
                mergeSort(a, 0, a.length - 1);
        }

        static void mergeSort(int[] a, int start, int end)
        {
                if (start < end)
                {
                        int mid = start + (end - start) / 2;
                        mergeSort(a, start, mid);
                        mergeSort(a, mid + 1, end);

                        merge(a, start, mid, end);
                }
        }

        static void merge(int[] a, int start, int mid, int end)
        {
                int n1 = mid - start + 1;
                int n2 = end - mid;
                int left[] = new int[n1];
                int right[] = new int[n2];
                for (int i = 0; i < n1; i++)
                        left[i] = a[start + i];
                for (int i = 0; i < n2; i++)
                        right[i] = a[mid + i + 1];

                int i = 0, j = 0, k = start;
                while (i < left.length && j < right.length)
                {
                        if (left[i] <= right[j])
                                a[k++] = left[i++];
                        else
                                a[k++] = right[j++];
                }

                while (i < left.length)
                        a[k++] = left[i++];

                while (j < right.length)
                        a[k++] = right[j++];
        }

        public static void main(String[] args)
        {
                int[] a = { 32, 4, 15, 66, 7, 2, 88, 45, 3, 9, 4, 23 };
                mergeSort(a);
                System.out.println(Arrays.toString(a));
        }
}
```

```java
package problems;

import java.util.Date;

/*
   If V == 0, then 0 coins required.
   If V > 0
   minCoin(coins[0..m-1], V) = min {1 + minCoins(V-coin[i])}
                               where i varies from 0 to m-1
                               and coin[i] < V
 */

public class MinCoins
{
        public static int minCoinsFast(int coins[], int m, int v)
        {
                // table[i] will be storing the minimum number of coins
                // required for i value. So table[v] will have result
                int[] table = new int[v + 1];

                // Base case (If given value v is 0)
                table[0] = 0;

                // Initialize all table values as Infinite
                for (int i = 1; i <= v; i++)
                        table[i] = Integer.MAX_VALUE;

                // Compute minimum coins required for all
                // values from 1 to v
                for (int i = 1; i <= v; i++)
                {
                        // Go through all coins smaller than i
                        for (int j = 0; j < m; j++)
                        {
                                if (coins[j] < i)
                                {
                                        int sub_res = table[i - coins[j]];
                                        table[i] = Math.min(sub_res + 1, table[i]);
                                }
                                // comment this else only if exact amount is required
                                // and add check for Integer.MAX_VALUE above before finding min
                                else
                                {
                                        table[i] = 1;
                                }
                                // System.out.println(Arrays.toString(table));
                        }
                }

                return table[v];
        }

        public static int minCoinsSlow(int coins[], int m, int v)
        {
                if (v <= 0)
                        return 0;

                int res = Integer.MAX_VALUE;

                for (int j = 0; j < m; j++)
                {
                        if (coins[j] <= v)
                        {
                                int sub_res = minCoinsSlow(coins, m, v - coins[j]);
                                if (sub_res + 1 < res)
                                        res = sub_res + 1;
                        }
                        else
                        {
                                return 1;
                        }
                }

                return res;
        }

        public static void main(String[] args)
        {
                int coins[] = { 3, 5, 7 };
                System.out.println(new Date());
                System.out.println("Minimum coins required is " + minCoinsFast(coins, coins.length, 100));
                System.out.println(new Date());
                System.out.println("Minimum coins required is " + minCoinsSlow(coins, coins.length, 100));
                System.out.println(new Date());
        }
}
```

```java
package problems;

import java.util.Arrays;

public class MinGates
{
        static int findMinGates(int[] arrivals, int[] departures, int flights)
        {
                if (flights < 1)
                        return 0;
                Arrays.sort(arrivals);
                Arrays.sort(departures);

                int gates = 1, minGates = 1, i = 1, j = 0;
                while (i < flights && j < flights)
                {
                        if (arrivals[i] <= departures[j])
                        {
                                gates++;
                                i++;
                                if (gates > minGates)
                                        minGates = gates;
                        }
                        else
                        {
                                gates--;
                                j++;
                        }
                }
                return minGates;
        }

        public static void main(String[] args)
        {

        }

}
```

```java
package problems;

public class MinHeap
{
        private int[] Heap;
        private int size;
        private int maxsize;

        private static final int FRONT = 1;

        public MinHeap(int maxsize)
        {
                this.maxsize = maxsize;
                this.size = 0;
                Heap = new int[this.maxsize + 1];
                Heap[0] = Integer.MIN_VALUE;
        }

        private int parent(int pos)
        {
                return pos / 2;
        }

        private int leftChild(int pos)
        {
                return (2 * pos);
        }

        private boolean isLeaf(int pos)
        {
                if (pos > (size / 2) && pos <= size) { return true; }
                return false;
        }

        private void swap(int fpos, int spos)
        {
                int tmp;
                tmp = Heap[fpos];
                Heap[fpos] = Heap[spos];
                Heap[spos] = tmp;
        }

        private void minHeapify(int pos)
        {
                while(!isLeaf(pos))
                {
                        int newPos = leftChild(pos);
                        if((newPos < size) && Heap[newPos+1] < Heap[newPos])
```

```java
                                newPos++; // move to right child
                        if(Heap[newPos] > Heap[pos])
                                break;
                        swap(pos, newPos);
                        pos = newPos;
                }
        }

        public void insert(int element)
        {
                Heap[++size] = element;
                int current = size;
                while (Heap[current] < Heap[parent(current)])
                {
                        swap(current, parent(current));
                        current = parent(current);
                }
        }

        public void minHeap() // this function is just to form a minheap from existing array
        {
                for (int pos = (size / 2); pos >= 1; pos--)
                {
                        minHeapify(pos);
                }
        }

        public int remove()
        {
                int popped = Heap[FRONT];
                if(size == FRONT)
                {
                        Heap[size--] = Integer.MAX_VALUE;
                }
                else
                {
                        Heap[FRONT] = Heap[size];
                        Heap[size--] = Integer.MAX_VALUE;
                        minHeapify(FRONT);
                }

                return popped;
        }

        public void removeAll()
        {
                while(size>0)
                        System.out.print(remove()+" ");
                System.out.println();
        }

        public static void main(String... arg)
        {
                System.out.println("The Min Heap is ");
                MinHeap minHeap = new MinHeap(10);
                minHeap.insert(5);
                minHeap.insert(3);
                minHeap.insert(17);
                minHeap.insert(10);
                minHeap.insert(84);
                minHeap.insert(19);
                minHeap.insert(6);
                minHeap.insert(22);
                minHeap.insert(9);
                minHeap.insert(100);

                minHeap.removeAll();

        }
}
```

```
package problems;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Random;

class Order
{
        String productId;
        double cost;
        int quantity;

        public Order(String id, double cost, int q)
        {
                this.productId = id;
                this.cost = cost;
                this.quantity = q;
        }

        @Override
        public String toString()
        {
                return "Q: " + this.quantity + " C: " + this.cost;
        }
}

public class MyComparator
{
        public static void main(String[] args)
        {
                List<Order> list = new ArrayList<Order>();
                Random r = new Random();
                for (int i = 0; i < 100; i++)
                {
                        Order o = new Order("Product" + i, r.nextDouble() * 100, r.nextInt(100));
                        list.add(o);
                }

                for (int i = 0; i < 100; i++)
                        System.out.print(list.get(i).toString() + " ");
                System.out.println();

                Collections.sort(list, new Comparator<Order>()
                {
                        @Override
                        public int compare(Order o1, Order o2)
                        {
                                if (o1.quantity != o2.quantity)
                                        return o1.quantity - o2.quantity;
                                else
                                        return (int) (o1.cost - o2.cost);
                        }
                });

                for (int i = 0; i < 100; i++)
                        System.out.print(list.get(i).toString() + " ");
        }

}
```

```java
package problems;

class ListNode<T>
{
        public T data;
        public ListNode<T> next;

        public ListNode()
        {
                next = null;
        }

        public ListNode(T d)
        {
                data = d;
                next = null;
        }
}

class RandomListNode
{
        int label;
        RandomListNode next, random;

        RandomListNode(int x)
        {
                this.label = x;
        }
};

public class MyLinkedList<T>
{
        ListNode<T> head;
        ListNode<T> tail;

        public MyLinkedList()
        {
                head = null;
                tail = null;
        }

        public ListNode<T> getHead()
        {
                return head;
        }

        public void setHead(ListNode<T> h)
        {
                head = h;
        }

        public ListNode<T> getTail()
        {
                return tail;
        }

        public int length()
        {
                ListNode<T> t = head;
                int len = 0;
                while (t != null)
                {
                        len++;
                        t = t.next;
                }

                return len;
        }

        void insertAtStart(T d)
        {
                ListNode<T> newNode = new ListNode<T>(d);

                if (head != null)
                        newNode.next = head;
                else
                        tail = newNode;
                head = newNode;
        }

        void insertAtEnd(T d)
        {
                ListNode<T> newNode = new ListNode<T>(d);

                if (head == null)
                        head = newNode;
                else
                        tail.next = newNode;
                tail = newNode;
        }

        void printList()
        {
                if (head == null)
                        return;

                ListNode<T> node = head;
                while (node != null)
                {
                        System.out.print(node.data);
                        if (node.next != null)
                                System.out.print("->");
                        node = node.next;
                }
                System.out.println();
        }

        ListNode<T> reverseList(ListNode<T> head)
        {
                if (head == null || head.next == null)
                        return head;

                ListNode<T> node = head;
                ListNode<T> prev = null;

                while (node != null)
                {
                        ListNode<T> next = node.next;
                        node.next = prev;
                        prev = node;
                        node = next;
                }

                return prev;
        }

        void reverseList()
        {
                ListNode<T> curr = head;
                ListNode<T> prev = null;
                ListNode<T> next = null;

                while (curr != null)
                {
                        next = curr.next;
                        curr.next = prev;
                        prev = curr;
                        curr = next;
                }

                head = prev;
        }

}
```

```java
package problems;

import java.util.Comparator;
import java.util.PriorityQueue;

public class MyPriorityQueue
{
        public static void main(String[] args)
        {
                Comparator<Integer> comp = new Comparator<Integer>()
                {
                        @Override
                        public int compare(Integer o1, Integer o2)
                        {
                                return o2 - o1;
                        }
                };
                PriorityQueue<Integer> pq = new PriorityQueue<Integer>(comp);
                pq.add(7);
                pq.add(8);
                pq.add(5);
                pq.add(9);
                while (pq.size() > 0)
                        System.out.println(pq.poll());
        }
}
```

```java
package problems;

import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Set;

class NaryTreeNode<T>
{
        T data;
        NaryTreeNode<T> firstChild;
        NaryTreeNode<T> nextSibling;

        public NaryTreeNode(T d)
        {
                data = d;
                firstChild = null;
                nextSibling = null;
        }
}

public class NaryTree<T>
{
        NaryTreeNode<T> root;

        public NaryTree()
        {
                root = null;
        }

        public NaryTreeNode<T> insert(NaryTreeNode<T> root, T parent, T d)
        {
                NaryTreeNode<T> newNode = new NaryTreeNode<T>(d);
                if (root == null)
                {
                        root = newNode;
                        return root;
                }

                NaryTreeNode<T> temp = findNode(root, parent);

                if (temp == null)
                {
                        newNode.nextSibling = root.nextSibling;
                        root.nextSibling = newNode;
                }
                else if (temp.firstChild != null)
                {
                        newNode.nextSibling = temp.firstChild.nextSibling;
                        temp.firstChild.nextSibling = newNode;
                }
                else
                {
                        temp.firstChild = newNode;
                }

                return root;
        }

        public NaryTreeNode<T> findNode(NaryTreeNode<T> root, T d)
        {
                if (root == null)
                        return null;

                if (root.data == d)
                        return root;

                NaryTreeNode<T> temp = findNode(root.firstChild, d);

                if (temp == null)
                {
                        temp = root.nextSibling;
                        while (temp != null)
                        {
                                if (temp.data == d)
                                        break;

                                NaryTreeNode<T> t = findNode(temp.firstChild, d);
                                if (t != null)
                                {
                                        temp = t;
                                        break;
                                }

                                temp = temp.nextSibling;
                        }
                }

                return temp;
        }

        public void printTree(NaryTreeNode<T> root)
        {
                if (root == null)
                        return;

                System.out.print(root.data + " ");

                NaryTreeNode<T> temp = root.nextSibling;
                while (temp != null)
                {
                        System.out.print(temp.data + " ");
                        printTree(temp.firstChild);
                        temp = temp.nextSibling;
                }
                System.out.println();
                printTree(root.firstChild);
        }

        public void printSiblings(T d)
        {
                if (root == null)
                        return;

                Queue<NaryTreeNode<T>> q = new LinkedList<NaryTreeNode<T>>();
                Set<T> level = new HashSet<T>();
                int currLevel = 0, nextLevel = 0;
                NaryTreeNode<T> temp = root;
                while (temp != null)
                {
                        q.offer(temp);
                        currLevel++;
                        level.add(temp.data);
                        temp = temp.nextSibling;
                }

                if (level.contains(d))
                {
                        System.out.println(level);
                        return;
                }

                level.clear();
                while (!q.isEmpty())
                {
                        temp = q.poll();
                        currLevel--;
                        if (temp.firstChild != null)
                        {
                                q.offer(temp.firstChild);
                                nextLevel++;
                                level.add(temp.firstChild.data);
                                temp = temp.firstChild;
                                while (temp.nextSibling != null)
                                {
                                        q.offer(temp.nextSibling);
                                        nextLevel++;
                                        level.add(temp.nextSibling.data);
                                        temp = temp.nextSibling;
                                }
                        }
                        if (currLevel == 0)
                        {
                                currLevel = nextLevel;
                                nextLevel = 0;
                                if (level.contains(d))
                                {
                                        System.out.println(level);
                                        return;
                                }
                                level.clear();
                                System.out.println();
                        }
                }

        }

        public static void main(String[] args)
        {
                NaryTree<String> ftree = new NaryTree<String>();

                ftree.root = ftree.insert(ftree.root, "", "adam");
                ftree.root = ftree.insert(ftree.root, "adam", "sam");
                ftree.root = ftree.insert(ftree.root, "adam", "watson");
                ftree.root = ftree.insert(ftree.root, "sam", "bob");
                ftree.root = ftree.insert(ftree.root, "sam", "jon");
                ftree.root = ftree.insert(ftree.root, "sam", "ruby");
                ftree.root = ftree.insert(ftree.root, "watson", "roger");

                ftree.printTree(ftree.root);
                ftree.printSiblings("bob");
        }
}
```

```java
package problems;

import java.util.LinkedList;
import java.util.Queue;

public class NextBiggestString
{
        public static String findNextBiggest(String in)
        {
                int len = in.length();
                int i = len - 1;
                char temp = 0;
                StringBuilder sb = new StringBuilder(in);
                StringBuilder sb1 = new StringBuilder();
                Queue<Character> queue = new LinkedList<Character>();
                for (; i > 0; i--)
                {
                        char a = in.charAt(i);
                        char b = in.charAt(i - 1);
                        if (a <= b)
                        {
                                queue.add(a);
                                continue;
                        }
                        queue.add(a);

                        System.out.println("queue " + queue.toString());

                        temp = queue.poll();
                        while (temp <= b)
                        {
                                sb1.append(temp);
                                temp = queue.poll();
                        }
                        System.out.println("sb1: " + sb1.toString());
                        sb.setCharAt(i - 1, temp);
                        sb1.append(b);

                        while (!queue.isEmpty())
                                sb1.append(queue.poll());
                        break;
                }
                System.out.println("sb1: " + sb1.toString());
                sb.setLength(i);
                return sb.append(sb1).toString();
        }

        public static void main(String args[])
        {
                String s = findNextBiggest("abcdeedcba");
                System.out.println(s);
        }
}
```

```
package problems;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

/*
    24988
        28693
        12907
        9197
        35287
 */

public class Palindrome
{

        public static boolean isPalindrome(String s)
        {
                for (int i = 0, j = s.length() - 1; i < s.length() / 2; i++, j--)
                {
                        if (s.charAt(i) != s.charAt(j))
                                return false;
                }
                return true;
        }

        public static void PalIndex(String s)
        {
                int i = 0, j = s.length() - 1;
                for (; i < s.length() / 2;)
                {
                        if (s.charAt(i) == s.charAt(j))
                        {
                                i++;
                                j--;
                                continue;
                        }
                        if (isPalindrome(s.substring(i + 1, j + 1)))
                        {
                                System.out.println(i);
                                break;
                        }
                        System.out.println(j);
                        break;
                }
        }

```

```java
        public static void main(String[] args) throws FileNotFoundException
        {
                System.out.println(new Date());
                Scanner sc = new Scanner(new File("pal1.txt"));
                int count = sc.nextInt();
                List<String> strList = new ArrayList<String>();
                for (int i = 0; i < count; i++)
                        strList.add(sc.next());
                sc.close();
                for (String s : strList)
                {
                        if (isPalindrome(s))
                        {
                                System.out.println("-1");
                                continue;
                        }
                        PalIndex(s);
                }
                System.out.println(new Date());
        }
}
```

```
package problems;

import java.util.Scanner;

public class Pangram
{
        static void isPangram(String s)
        {
                if (s == null || s.isEmpty())
                        return;

                int[] freq = new int[26];
                int len = s.length();
                for (int i = 0; i < len; i++)
                {
                        char c = s.charAt(i);
                        if (c < ’a’ || c > ’z’)
                                continue;
                        freq[c - ’a’]++;
                }

                int count = freq[0];
                for (int i = 0; i < 26; i++)
                {
                        if (count == 0)
                        {
                                System.out.println("not pangram");
                                return;
                        }
                        else if (count != freq[i])
                        {
                                System.out.println("pangram");
                                return;
                        }
                }
                System.out.println("multiple pangram " + count);
        }

        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                String s = sc.nextLine();
                isPangram(s.toLowerCase());
                sc.close();
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Date;
import java.util.HashSet;

public class Permutation
{
        public static HashSet<String> perm(String s)
        {
                if (s == null)
                        return null;
                HashSet<String> permList = new HashSet<String>();
                if (s.isEmpty())
                {
                        permList.add("");
                        return permList;
                }

                char first = s.charAt(0);
                String remaining = s.substring(1);
                HashSet<String> inter = perm(remaining);

                for (String str : inter)
                {
                        for (int i = 0; i <= str.length(); i++)
                                permList.add(insertAt(str, first, i));
                }
                return permList;
        }

        public static String insertAt(String s, char c, int i)
        {
                return (s.substring(0, i) + c + s.substring(i));
        }

        // alternative approach
        static ArrayList<String> getPerms(String s)
        {
                ArrayList<String> result = new ArrayList<>();
                getPerms("", s, result);
                return result;
        }

        static void getPerms(String prefix, String rem, ArrayList<String> result)
        {
                if (rem.length() == 0)
                        result.add(prefix);

                int len = rem.length();
                for (int i = 0; i < len; i++)
                {
                        String before = rem.substring(0, i);
                        String after = rem.substring(i + 1);
                        char c = rem.charAt(i);
                        getPerms(prefix + c, before + after, result);
                }
        }

        public static void main(String[] args)
        {
                String a = "aaaaaaaaaaaaa";
                System.out.println(new Date());
                System.out.println(perm(a).size());
                System.out.println(new Date());
                System.out.println(getPerms(a).size());
                System.out.println(new Date());
        }

}
```

```java
package problems;

public class PermutationPalindrome
{
        public static boolean isPermPalindrome(String s)
        {
                int bitVector = 0;
                for (char c : s.toCharArray())
                        bitVector = bitVector ^ (1 << (c - 'a'));
                // System.out.println(Integer.toBinaryString(bitVector) + " " +
                // Integer.toBinaryString(bitVector - 1));
                return (bitVector & (bitVector - 1)) == 0;
        }

        public static void main(String[] args)
        {
                System.out.println(isPermPalindrome("Tactooca".toLowerCase()));
        }

}
```

```java
package problems;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class PoisonousPlant
{

        public static void main(String[] args) throws FileNotFoundException
        {
                Scanner sc = new Scanner(new File("poison.txt"));
                List<Integer> nums = new ArrayList<Integer>();
                int listSize = sc.nextInt();
                for (int i = 0; i < listSize; i++)
                {
                        nums.add(sc.nextInt());
                }
                int ans = 0;
                while (true)
                {
                        int deadCnt = 0;
                        int left = -1;
                        for (int i = 1; i < nums.size();)
                        {
                                if (nums.get(i) > ((left == -1) ? nums.get(i - 1) : left))
                                {
                                        left = nums.remove(i);
                                        deadCnt++;
                                }
                                else
                                {
                                        left = -1;
                                        i++;
                                }
                        }

                        if (deadCnt == 0)
                                break;
                        ans++;
                }
                System.out.print(ans);
                sc.close();
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.List;

public class PowerSet
{
        public static void printAllSets(char[] c)
        {
                List<String> list = new ArrayList<String>();
                printSets(c, 0, list);
                System.out.println(list);
                System.out.println(list.size());
        }

        private static void printSets(char[] c, int index, List<String> list)
        {
                if (index == c.length)
                        return;
                int len = list.size();

                for (int i = 0; i < len; i++)
                        list.add(list.get(i) + c[index]);

                list.add(c[index] + "");
                index++;
                printSets(c, index, list);
        }

        public static void main(String args[])
        {
                char[] c = { ’a’, ’b’, ’c’, ’d’, ’e’ };
                printAllSets(c);
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Arrays;

public class PowerSetIterative
{
        static ArrayList<String> buildSubsequences(String s)
        {
                ArrayList<String> subsequence = new ArrayList<String>();
                int len = s.length();
                for (int i = 0; i < len; i++)
                {
                        if (subsequence.size() > 0)
                        {
                                int l = subsequence.size();
                                for (int j = 0; j < l; j++)
                                        subsequence.add(subsequence.get(j) + s.charAt(i));
                        }
                        subsequence.add(s.charAt(i) + "");
                }

                return subsequence;
        }

        public static void main(String[] args)
        {
                ArrayList<String> arr = buildSubsequences("abba");
                System.out.println(Arrays.toString(arr.toArray(new String[0])));
        }
}
```

```java
package problems;

import java.util.Arrays;
import java.util.Random;

public class QuickSort
{
        private static int partition(int[] a, int l, int r)
        {
                int i = l - 1, j = r, temp = 0;
                while (true)
                {
                        while (a[++i] < a[r]);

                        while (a[--j] > a[r])
                                if (j == i)
                                        break;

                        if (i >= j)
                                break;

                        temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                }
                if (i != r)
                {
                        temp = a[i];
                        a[i] = a[r];
                        a[r] = temp;
                }
                return i;
        }

        public static void sort(int[] a, int low, int high)
        {
                if (low >= high)
                        return;
                int p = partition(a, low, high);
                sort(a, low, p - 1);
                sort(a, p + 1, high);
        }

        public static void main(String[] args)
        {
                int a[] = new int[1000];
                Random r = new Random();
                for (int i = 0; i < 1000; i++)
```

```
            a[i] = r.nextInt(1000);
        System.out.println(Arrays.toString(a));
        sort(a, 0, 999);
        System.out.println(Arrays.toString(a));
    }

}
```

```java
package problems;

import java.util.Stack;

public class RemoveMatchingPairs
{

        boolean isUpper(char c)
        {
                return (c >= ’A’ && c <= ’Z’);
        }

        boolean isLower(char c)
        {
                return (c >= ’a’ && c <= ’z’);
        }

        boolean equalsIgnoreCase(char upper, char lower)
        {
                return (lower - upper == 32);
        }

        int findMatchingPair(String input)
        {
                Stack<Character> st = new Stack<Character>();
                int len = input.length();
                int retValue = -1;
                for (int i = 0; i < len; i++)
                {
                        char c = input.charAt(i);
                        if (isUpper(c))
                        {
                                st.push(c);
                        }
                        else
                        {
                                if (!st.empty() && equalsIgnoreCase(st.peek(), c))
                                {
                                        retValue = i;
                                        st.pop();
                                }
                                else
                                {
                                        return retValue;
                                }
                        }
                }
                return retValue;
        }

        public static void main(String[] args)
        {

        }

}
```

```java
package problems;

public class RLE
{
        public static String compressBad(String str)
        {
                String mystr = "";
                char last = str.charAt(0);
                int count = 1;
                for (int i = 1; i < str.length(); i++)
                {
                        if (str.charAt(i) == last)
                        {
                                count++;
                        }
                        else
                        {
                                mystr += last + "" + count;
                                last = str.charAt(i);
                                count = 1;
                        }
                }
                return mystr + last + count;
        }

        public static void main(String[] args)
        {
                String str = "abcd";
                String str2 = compressBad(str);
                System.out.println("New String (len = " + str2.length() + "): " + str2);
        }

}
```

```java
package problems;

public class Roate90Degree
{
        static int[][] rotateExtraSpace(int[][] a)
        {
                int m = a.length, n = a[0].length;
                int[][] b = new int[n][m];
                for (int i = 0; i < m; i++)
                        for (int j = 0; j < n; j++)
                                b[j][m - i - 1] = a[i][j];
                return b;
        }

        // can be done only for sq matrix. ratate one number at a time
        static void rotateInPlace(int[][] a)
        {
                if (a.length == 0 || a.length != a[0].length)
                        return;
                int n = a.length;
                for (int i = 0; i < n / 2; i++)
                {
                        for (int j = i; j < n - 1 - i; j++)
                        {
                                int temp = a[i][j];
                                a[i][j] = a[n - 1 - j][i];
                                a[n - 1 - j][i] = a[n - 1 - i][n - 1 - j];
                                a[n - 1 - i][n - 1 - j] = a[j][n - 1 - i];
                                a[j][n - 1 - i] = temp;
                        }
                }
        }

        static void printMatrix(int[][] a)
        {
                for (int i = 0; i < a.length; i++)
                {
                        for (int j = 0; j < a[0].length; j++)
                                System.out.print(a[i][j] + " ");
                        System.out.println();
                }
        }

        public static void main(String[] args)
        {
                int[][] a = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }, { 9, 10, 11, 12 } };
                printMatrix(rotateExtraSpace(a));
                rotateInPlace(a);
                System.out.println();
                printMatrix(a);
        }

}
```

```java
package problems;

import java.util.Arrays;

public class RodCutting
{
        static int cutRod(int price[], int n)
        {
                int max[] = new int[n + 1];
                max[1] = price[0];
                for (int i = 2; i <= n; i++)
                {
                        int max_val = Integer.MIN_VALUE;
                        for (int j = 0; j < i; j++)
                                max_val = Math.max(max_val, price[j] + max[i - j - 1]);
                        max[i] = max_val;
                }
                System.out.println(Arrays.toString(max));
                return max[n];
        }

        public static void main(String args[])
        {
                int arr[] = new int[] { 3, 7, 8, 9, 10, 17, 17, 20 };
                int size = arr.length;
                System.out.println("Maximum Obtainable Value is " + cutRod(arr, size));

        }
}
```

```java
package problems;

public class RotatedArray
{
        public static int findMin(int[] nums)
        {
                if (nums == null || nums.length == 0)
                        return -1;

                int low = 0, high = nums.length - 1;

                if (nums[low] < nums[high])
                        return nums[low];

                while (low < high)
                {
                        int mid = low + (high - low) / 2;

                        if (mid < high && nums[mid + 1] < nums[mid])
                                return nums[mid + 1];

                        if ((mid > low) && (nums[mid] < nums[mid - 1]))
                                return nums[mid];

                        if (nums[low] < nums[mid])
                                low = mid + 1;

                        else
                                high = mid - 1;
                }

                return nums[low];
        }

        static int find(int[] nums, int target)
        {
                if (nums == null || nums.length == 0)
                        return -1;

                int low = 0, high = nums.length - 1;
                while (low <= high)
                {
                        int mid = low + (high - low) / 2;
                        if (nums[mid] == target)
                                return mid;

                        if (nums[low] <= nums[mid])
                        {
                                if (target >= nums[low] && target < nums[mid])
                                        high = mid - 1;
                                else
                                        low = mid + 1;
                        }
                        else
                        {
                                if (target > nums[mid] && target <= nums[high])
                                        low = mid + 1;
                                else
                                        high = mid - 1;
                        }
                }
                return -1;
        }

        public static void main(String[] args)
        {
                int[] nums = { 1 };
                // int[] nums = { 7, 8, 8, 1, 2, 4, 5, 6, 7 };
                // int[] nums = { 7, 7, 7, 7, 7, 7, 7 };
                System.out.println(findMin(nums));

        }
}
```

```java
package problems;

import java.util.Collections;
import java.util.PriorityQueue;
import java.util.Scanner;

public class RunningMedian
{
        public static void main(String args[]) throws Exception
        {
                Scanner sc = new Scanner(System.in);
                int n = sc.nextInt(), a = 0;
                PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>(n / 2 + 1);
                PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>((n / 2 + 1), Collections.reverseOrder());
                double median = 0;
                for (int i = 0; i < n; i++)
                {
                        a = sc.nextInt();
                        if (a > median)
                        {
                                if (maxHeap.size() < minHeap.size())
                                        maxHeap.add(minHeap.poll());
                                minHeap.add(a);
                                median = minHeap.size() == maxHeap.size() ? (minHeap.peek() + maxHeap.peek()) / 2.0 : minHeap.peek();
                        }
                        else
                        {
                                if (minHeap.size() < maxHeap.size())
                                        minHeap.add(maxHeap.poll());
                                maxHeap.add(a);
                                median = minHeap.size() == maxHeap.size() ? (minHeap.peek() + maxHeap.peek()) / 2.0 : maxHeap.peek();
                        }
                        System.out.println(median);
                }
                sc.close();
        }
}
```

```java
package problems;

import java.util.Scanner;

/* Class SBBSTNode */
class SBBSTNode
{
        SBBSTNode left, right;
        int data;
        int height;

        /* Constructor */
        public SBBSTNode()
        {
                left = null;
                right = null;
                data = 0;
                height = 0;
        }

        /* Constructor */
        public SBBSTNode(int n)
        {
                left = null;
                right = null;
                data = n;
                height = 0;
        }
}

public class SelfBalancingBST
{

        private SBBSTNode root;

        public SelfBalancingBST()
        {
                root = null;
        }

        /* Function to check if tree is empty */
        public boolean isEmpty()
        {
                return root == null;
        }

        /* Make the tree logically empty */
        public void clear()
        {
                root = null;
        }

        /* Function to insert data */
        public void insert(int data)
        {
                root = insert(data, root);
        }

        /* Function to get height of node */
        private int height(SBBSTNode t)
        {
                return t == null ? -1 : t.height;
        }

        /* Function to max of left/right node */
        private int max(int lhs, int rhs)
        {
                return lhs > rhs ? lhs : rhs;
        }

        /* Function to insert data recursively */
        private SBBSTNode insert(int x, SBBSTNode t)
        {
                if (t == null)
                        t = new SBBSTNode(x);
                else if (x <= t.data)
                {
                        t.left = insert(x, t.left);
                        if (height(t.left) - height(t.right) == 2)
                                if (x < t.left.data)
                                        t = rotateWithLeftChild(t);
                                else
                                        t = doubleWithLeftChild(t);
                }
                else
                {
                        t.right = insert(x, t.right);
                        if (height(t.right) - height(t.left) == 2)
                                if (x > t.right.data)
                                        t = rotateWithRightChild(t);
                                else
                                        t = doubleWithRightChild(t);
                }
                t.height = max(height(t.left), height(t.right)) + 1;
                return t;
        }

        /* Rotate binary tree node with left child */
        private SBBSTNode rotateWithLeftChild(SBBSTNode k)
        {
                SBBSTNode k1 = k.left;
                k.left = k1.right;
                k1.right = k;
                k.height = max(height(k.left), height(k.right)) + 1;
                k1.height = max(height(k1.left), k.height) + 1;
                return k1;
        }

        /* Rotate binary tree node with right child */
        private SBBSTNode rotateWithRightChild(SBBSTNode k)
        {
                SBBSTNode k1 = k.right;
                k.right = k1.left;
                k1.left = k;
                k.height = max(height(k.left), height(k.right)) + 1;
                k1.height = max(height(k1.right), k.height) + 1;
                return k1;
        }

        private SBBSTNode doubleWithLeftChild(SBBSTNode k)
        {
                System.out.println("doubleWithLeftChild");
                k.left = rotateWithRightChild(k.left);
                return rotateWithLeftChild(k);
        }

        private SBBSTNode doubleWithRightChild(SBBSTNode k)
        {
                System.out.println("doubleWithRightChild");
                k.right = rotateWithLeftChild(k.right);
                return rotateWithRightChild(k);
        }

        /* Functions to count number of nodes */
        public int countNodes()
        {
                return countNodes(root);
        }

        private int countNodes(SBBSTNode r)
        {
                if (r == null)
                        return 0;
                else
                {
                        int l = 1;
                        l += countNodes(r.left);
                        l += countNodes(r.right);
                        return l;
                }
        }

        /* Functions to search for an element */
        public boolean search(int val)
        {
                return search(root, val);
        }

        private boolean search(SBBSTNode r, int val)
        {
                boolean found = false;
                while ((r != null) && !found)
                {
                        int rval = r.data;
                        if (val < rval)
                                r = r.left;
                        else if (val > rval)
                                r = r.right;
                        else
                        {
                                found = true;
                                break;
                        }
                        found = search(r, val);
                }
                return found;
        }

        /* Function for inorder traversal */
        public void inorder()
        {
                inorder(root);
        }

        private void inorder(SBBSTNode r)
        {
                if (r != null)
                {
                        inorder(r.left);
                        System.out.print(r.data + " ");
                        inorder(r.right);
                }
        }

        /* Function for preorder traversal */
        public void preorder()
        {
                preorder(root);
        }

        private void preorder(SBBSTNode r)
        {
                if (r != null)
                {
                        System.out.print(r.data + " ");
                        preorder(r.left);
                        preorder(r.right);
                }
        }

        /* Function for postorder traversal */
        public void postorder()
        {
                postorder(root);
        }

        private void postorder(SBBSTNode r)
        {
                if (r != null)
                {
                        postorder(r.left);
                        postorder(r.right);
                        System.out.print(r.data + " ");
                }
        }

        public static void main(String[] args)
        {
                Scanner scan = new Scanner(System.in);
                SelfBalancingBST sbbst = new SelfBalancingBST();
                char ch;

                do
                {
                        System.out.println("\nSelfBalancingBST Operations\n");
                        System.out.println("1. insert ");
                        System.out.println("2. search");
                        System.out.println("3. count nodes");
                        System.out.println("4. check empty");
                        System.out.println("5. clear tree");

                        int choice = scan.nextInt();
                        switch (choice)
                        {
                                case 1:
                                        System.out.println("Enter integer element to insert");
                                        sbbst.insert(scan.nextInt());
                                        break;
                                case 2:
                                        System.out.println("Enter integer element to search");
                                        System.out.println("Search result : " + sbbst.search(scan.nextInt()));
                                        break;
                                case 3:
                                        System.out.println("Nodes = " + sbbst.countNodes());
                                        break;
                                case 4:
                                        System.out.println("Empty status = " + sbbst.isEmpty());
                                        break;
                                case 5:
                                        System.out.println("\nTree Cleared");
                                        sbbst.clear();
                                        break;
                                default:
                                        System.out.println("Wrong Entry \n ");
                                        break;
                        }
                        /* Display tree */
                        System.out.print("\nPost order : ");
                        sbbst.postorder();
                        System.out.print("\nPre order : ");
                        sbbst.preorder();
                        System.out.print("\nIn order : ");
                        sbbst.inorder();

                        System.out.println("\nDo you want to continue (Type y or n) \n");
                        ch = scan.next().charAt(0);
                } while (ch == 'Y' || ch == 'y');
                scan.close();
        }
}
```

```java
package problems;

import java.util.LinkedList;
import java.util.Queue;

class MazeNode
{
        int x;
        int y;

        public MazeNode(int a, int b)
        {
                x = a;
                y = b;
        }
}

class QueueNode
{
        MazeNode a;
        int dist;

        public QueueNode(MazeNode a, int dist)
        {
                this.a = a;
                this.dist = dist;
        }
}

public class ShortestPathBinaryMaze
{
        static boolean isValid(int x, int y, int rows, int cols)
        {
                if (x >= 0 && y >= 0 && x < rows && y < cols)
                        return true;
                return false;
        }

        static int shortestPath(int[][] maze, MazeNode src, MazeNode dst)
        {
                if (maze[src.x][src.y] == 0 || maze[dst.x][dst.y] == 0)
                        return -1;

                int rows = maze.length, cols = maze[0].length;
                int adjX[] = {-1, 0, 1, 0}, adjY[] = {0, -1, 0, 1};
                Queue<QueueNode> q = new LinkedList<QueueNode>();
                boolean[][] visited = new boolean[rows][cols];
                q.offer(new QueueNode(src, 0));
                visited[src.x][src.y] = true;
                while (!q.isEmpty())
                {
                        QueueNode temp = q.poll();
                        if (temp.a.x == dst.x && temp.a.y == dst.y)
                                return temp.dist;

                        for (int i = 0; i < 4; i++)
                        {
                                int tempX = temp.a.x + adjX[i];
                                int tempY = temp.a.y + adjY[i];

                                if (isValid(tempX, tempY, rows, cols) && !visited[tempX][tempY] && maze[tempX][tempY] > 0)
                                {
                                        q.offer(new QueueNode(new MazeNode(tempX, tempY), temp.dist + 1));
                                        visited[tempX][tempY] = true;
                                }
                        }
                }
                return -1;
        }

        public static void main(String[] args)
        {
                int maze[][] =
                {
                        { 1, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
                        { 1, 0, 1, 0, 1, 1, 1, 0, 1, 1 },
                        { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
                        { 0, 0, 1, 1, 1, 0, 0, 0, 0, 1 },
                        { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
                        { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },
                        { 1, 0, 0, 0, 0, 0, 0, 0, 0, 1 },
                        { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },
                        { 1, 1, 0, 0, 0, 0, 1, 0, 0, 1 }
                };
                System.out.println(shortestPath(maze, new MazeNode(0, 0), new MazeNode(0, 6)));
        }
}
```

```java
package problems;

import java.util.Arrays;

public class SlidingWindowSum
{
        static int[] slidingWindowSum(int a[], int k)
        {
                if (a == null || a.length == 0)
                        return null;
                int win[] = new int[k];
                int sum = 0, i = 0, n = a.length, j = 0;
                int res[] = new int[n - k + 1];
                for (i = 0; i < n; i++)
                {
                        sum -= win[i % k];
                        win[i % k] = a[i];
                        sum += win[i % k];
                        if (i >= k - 1)
                                res[j++] = sum;
                }
                if (i < k)
                        res[j++] = sum;
                return res;
        }

        public static void main(String[] args)
        {
                int[] a = { 1, 3, -1 };
                System.out.println(Arrays.toString(slidingWindowSum(a, 3)));
        }
}
```

```
package problems;

public class SpiralPrint
{
        static void spiralPrint(int a[][])
        {
                int i, rowStart = 0, colStart = 0, rowEnd = a.length, colEnd = a[0].length;

                while (rowStart < rowEnd && colStart < colEnd)
                {
                        for (i = colStart; i < colEnd; ++i)
                                System.out.print(a[rowStart][i] + " ");
                        rowStart++;

                        for (i = rowStart; i < rowEnd; ++i)
                                System.out.print(a[i][colEnd - 1] + " ");
                        colEnd--;

                        // to prevent printing first row again in case of single row matrix
                        if (rowStart < rowEnd)
                        {
                                for (i = colEnd - 1; i >= colStart; --i)
                                        System.out.print(a[rowEnd - 1][i] + " ");
                                rowEnd--;
                        }
                        // to prevent printing first column again in case of single column
                        // matrix
                        if (colStart < colEnd)
                        {
                                for (i = rowEnd - 1; i >= rowStart; --i)
                                        System.out.print(a[i][colStart] + " ");
                                colStart++;
                        }
                }
        }

        static void spiralPrintReverse(int a[][])
        {
                int sr = a.length / 2, sc = sr, totalCount = (a.length * a[0].length);
                int leftInc = 1, rightInc = 2;
                int count = 0;
                while (count < totalCount)
                {
                        for (int i = 0; i < leftInc; i++)
                        {
                                System.out.print(a[sr][sc--] + " ");
                                count++;
                        }
```

```java
                        for (int i = 0; i < leftInc && count < totalCount; i++)
                        {
                                System.out.print(a[sr--][sc] + " ");
                                count++;
                        }

                        for (int i = 0; i < rightInc && count < totalCount; i++)
                        {
                                System.out.print(a[sr][sc++] + " ");
                                count++;
                        }
                        for (int i = 0; i < rightInc && count < totalCount; i++)
                        {
                                System.out.print(a[sr++][sc] + " ");
                                count++;
                        }
                        leftInc += 2;
                        rightInc += 2;
                }
        }

        public static void main(String[] args)
        {
                int[][] a = { { 1, 2 }, { 1, 2 } };
                spiralPrint(a);
                System.out.println();
                spiralPrintReverse(a);
        }
}
```

```java
package problems;

import java.util.Scanner;

public class StrobogrammaticNumber
{
        public static boolean isStrobogrammatic(String n)
        {
                if (n == null || n.isEmpty())
                        return false;
                int start = 0, end = n.length() - 1;
                while (start <= end)
                {
                        if (n.charAt(start) == n.charAt(end))
                        {
                                if (isStrobo(n.charAt(start)))
                                {
                                        start++;
                                        end--;
                                }
                                else
                                {
                                        return false;
                                }
                        }
                        else
                        {
                                if (n.charAt(start) == '6' && n.charAt(end) == '9' || n.charAt(start) == '9' && n.charAt(end) == '6')
                                {
                                        start++;
                                        end--;
                                }
                                else
                                {
                                        return false;
                                }
                        }
                }
                return true;
        }

        static boolean isStrobo(char c)
        {
                return (c == '8' || c == '0' || c == '1');
        }

        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                String n = sc.next();
                sc.close();
                System.out.println(isStrobogrammatic(n));
        }
}
```

```java
package problems;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class SubPowerSet
{
        static List<List<Integer>> combine(int n, int k)
        {
                List<List<Integer>> result = new ArrayList<List<Integer>>();
                if (k > n || k < 0)
                        return result;

                if (k == 0)
                {
                        result.add(new ArrayList<Integer>());
                        return result;
                }
                result = combine(n - 1, k - 1);
                for (List<Integer> list : result)
                        list.add(n);

                result.addAll(combine(n - 1, k));
                return result;
        }

        public static void main(String[] args)
        {
                List<List<Integer>> list = combine(20, 16);
                for (List<Integer> l : list)
                        System.out.println(Arrays.toString(l.toArray()));
        }

}
```

```java
package problems;

// this program will compile and run w/o any error regardless of Test1 extending Test (javac Test.java)
// java Test - no output
// java Test1 - hello
// Only constraint - filename should match with public class name.
// A java file can have ony one public class, but it can have multiple classes with one main method each
// On compiling such a class multiple class files are created and they can be run separately

public class Test
{

}

class Test1 extends Test
{
        public static void main(String[] args)
        {
                System.out.println("hello");
        }
}
```

```java
package problems;

import java.util.Stack;

public class TopologicalSort
{
        static void topologicalSortUtil(DirectedGraph g, int v, boolean[] visited, Stack<Integer> stack)
        {
                visited[v] = true;
                for (int n : g.adj[v])
                {
                        if (!visited[n])
                                topologicalSortUtil(g, n, visited, stack);
                }
                stack.push(v);
        }

        static void topologicalSort(DirectedGraph g, int v)
        {
                boolean visited[] = new boolean[v];
                Stack<Integer> stack = new Stack<Integer>();

                for (int i = 0; i < v; i++)
                {
                        if (!visited[i])
                                topologicalSortUtil(g, i, visited, stack);
                }

                while (!stack.isEmpty())
                        System.out.print(stack.pop() + " ");
        }

        public static void main(String[] args)
        {
                DirectedGraph g = new DirectedGraph(6);
                g.addEdge(0, 2);
                g.addEdge(0, 5);
                g.addEdge(1, 5);
                g.addEdge(1, 4);
                g.addEdge(2, 3);
                g.addEdge(3, 4);

                System.out.println("Topological Sort:");
                topologicalSort(g, 6);
        }
}
```

```java
package problems;

public class Triangle
{
        public static void main(String[] args)
        {
                int n = 4;
                for (int i = 0; i < n; i++)
                {
                        for (int j = 1; j < n + i + 1; j++)
                        {
                                if (j < n - i)
                                        System.out.print(" ");
                                else
                                        System.out.print("*");
                        }
                        System.out.println();
                }
        }
}
```

```java
package problems;

class TrieNode
{
        TrieNode[] children;
        boolean isLeaf;

        public TrieNode()
        {
                this.children = new TrieNode[256];
        }

        public void printChildren()
        {
                for (char i = 0; i < 256; i++)
                {
                        if (children[i] == null)
                                continue;
                        if (children[i].isLeaf)
                                System.out.println(i);
                        else
                                System.out.print(i);
                        children[i].printChildren();
                }
        }

        public void printChildren(String prefix)
        {
                for (char i = 0; i < 256; i++)
                {
                        if (children[i] == null)
                                continue;
                        System.out.print(prefix);
                        children[i].printChildren();
                }
        }
}

public class Trie
{
        private TrieNode root;

        public Trie()
        {
                this.root = new TrieNode();
        }

        public TrieNode getRoot()
        {
                return this.root;
        }

        public void printChildren()
        {
                TrieNode t = getRoot();
                for (char i = 0; i < 256; i++)
                {
                        if (t.children[i] == null)
                                continue;
                        System.out.print(i);
                        t.children[i].printChildren();
                        System.out.println();
                }
        }

        // Inserts a word into the trie.
        public void insert(String word)
        {
                TrieNode t = getRoot();
                for (int i = 0; i < word.length(); i++)
                {
                        char c = word.charAt(i);
                        if (t.children[c] == null)
                                t.children[c] = new TrieNode();
                        t = t.children[c];
                }
                t.isLeaf = true;
        }

        // Returns if the word is in the trie.
        public boolean search(String word)
        {
                TrieNode t = searchNode(word);

                if (t != null)
                        return t.isLeaf;
                else
                        return false;
        }

        // Returns if there is any word in the trie
        // that starts with the given prefix.
        public boolean startsWith(String prefix)
        {
                if (searchNode(prefix) == null)
                        return false;
```

```java
                else
                        return true;
        }

        public void autoComplete(String s)
        {
                TrieNode t = searchNode(s);
                if (t.isLeaf)
                        System.out.println(s);
                t.printChildren(s);
        }

        public TrieNode searchNode(String str)
        {
                TrieNode t = getRoot();
                for (int i = 0; i < str.length(); i++)
                {
                        char c = str.charAt(i);
                        if (t.children[c] != null)
                                t = t.children[c];
                        else
                                return null;
                }
                return t;
        }

        public static void main(String args[])
        {
                Trie t = new Trie();
                t.insert("cat");
                t.insert("cater");
                t.insert("base");
                t.insert("basement");
                t.insert("baseline");
                t.printChildren();
                // System.out.println(t.search("cat"));
                // System.out.println(t.search("cate"));
                t.autoComplete("cat");
        }
}
```

```java
package problems;

import java.util.Arrays;

public class WiggleSort
{
        static void wiggleSort(int[] nums)
        {
                for (int i = 1; i < nums.length; i++)
                {
                        int a = nums[i - 1];
                        if ((i % 2 == 1) == (a > nums[i]))
                        {
                                nums[i - 1] = nums[i];
                                nums[i] = a;
                        }
                }
        }

        public static void main(String[] args)
        {
                int[] nums = { 1, 2, 3, 4, 5, 6, 7 };
                wiggleSort(nums);
                System.out.println(Arrays.toString(nums));
        }

}
```