1.Program for find Duplicate Sub trees.

```
import java.util.*;

class TreeNodeDup {
    int val;
    TreeNodeDup left;
    TreeNodeDup right;

    TreeNodeDup(int val) {
        this.val = val;
    }
}

public class DuplicateSubtrees {
    public static List<TreeNodeDup> findDuplicateSubtrees(TreeNodeDup root) {
        List<TreeNodeDup> result = new ArrayList<>();
        Map<String, Integer> subtreeCount = new HashMap<>();
        traverse(root, subtreeCount, result);
        return result;
    }

    private static String traverse(TreeNodeDup node, Map<String, Integer>
subtreeCount, List<TreeNodeDup> result) {
        if (node == null) {
            return null;
        }
        // Recursively traverse left and right subtrees
        String left = traverse(node.left, subtreeCount, result);
        String right = traverse(node.right, subtreeCount, result);
       // Construct subtree representation
        String subtree = node.val + "," + left + "," + right;
        subtreeCount.put(subtree, subtreeCount.getOrDefault(subtree, 0) + 1);

        // If subtree is duplicate, add to result list
        if (subtreeCount.get(subtree) == 2) {
            result.add(node);
        }

        return subtree;
    }

    public static void main(String[] args) {
        TreeNodeDup root = new TreeNodeDup(1);
        root.left = new TreeNodeDup(2);
        root.right = new TreeNodeDup(3);
        root.left.left = new TreeNodeDup(4);
        root.right.left = new TreeNodeDup(2);
        root.right.right = new TreeNodeDup(4);
        root.right.left.left = new TreeNodeDup(4);
        System.out.println("Tree with Duplicate value :");
        printTree(root);
        System.out.println();
        List<TreeNodeDup> duplicates = findDuplicateSubtrees(root);
        System.out.println("Duplicate subtrees:");
```

```java
        for (TreeNodeDup node : duplicates) {
            printTree(node);
            System.out.println();
        }
    }
    // Helper function to print a tree
    private static void printTree(TreeNodeDup node) {
        if (node == null) {
            return;
        }
        System.out.print(node.val + " ");
        printTree(node.left);
        printTree(node.right);
    }
}
```

OUTPUT

```
Tree with Duplicate value:

1 2 4 3 2 4 4

Duplicate subtrees:

4

2 4
```

2.Program for Insert into Binary Search Tree.

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
    }
}

public class InsertIntoBST {
    public static TreeNode insertIntoBST(TreeNode root, int val) {
        // If the root is null, create a new node with the given value
        if (root == null) {
            return new TreeNode(val);
        }

        // If the value is less than the current node's value, insert into
the left subtree
        if (val < root.val) {
            root.left = insertIntoBST(root.left, val);
        }
        // If the value is greater than or equal to the current node's value,
insert into the right subtree
        else {
            root.right = insertIntoBST(root.right, val);
        }

        return root;
    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(4);
        root.left = new TreeNode(2);
        root.right = new TreeNode(7);
        root.left.left = new TreeNode(1);
        root.left.right = new TreeNode(3);
        System.out.println("Tree : ");
        printTree(root, 0);
        // Value to be inserted
        int value = 5;

        // Insert the value into the BST
        TreeNode newRoot =insertIntoBST(root, value);
        System.out.println("Insert New Node :");
        // Print the updated tree
        printTree(newRoot, 0);
    }

    // Helper function to print the tree in-order
    private static void printTree(TreeNode node , int level) {
        if (node == null) {
            return;
        }
        printTree(node.right, level + 1);

        for (int i = 0; i < level; i++) {
            System.out.print("    ");
        }
        System.out.println(node.val);

        printTree(node.left, level + 1);
      }
}
```

OUTPUT

```
Tree

  7

4

    3

  2

    1

Insert New Node

  7

    5

4

    3

  2

    1
```

3.Program for Longest Word in Dictionary

```
public class LongestWordInDictionary {

        public static String longestWord(String[] words) {
            // Sort the words array
            Arrays.sort(words);

            // Create a map to store word
            HashMap<String, Integer> map = new HashMap<>();
            for (int i = 0; i < words.length; i++) {
                map.put(words[i], map.getOrDefault(words[i], 0) + 1);
            }

             String result = "";

            // Iterate through the sorted words in reverse order
            for (int i = words.length - 1; i >= 0; i--) {
                if (words[i].length() < result.length()) {
                    continue;
                } else {
                    String word = "";
                    boolean flag = false;

                    // Check if the current word can be formed by deleting characters
                    for (int j = 0; j < words[i].length(); j++) {
                        word += words[i].charAt(j);
                        if (!map.containsKey(word)) {
                            flag = true;
                            break;
                        }
                    }
                    // If the current word can be formed, update the result
                    if (!flag) {
                        result = words[i];
                    }
                }
            }

            return result;
        }

       public static void main(String[] args) {
            String[] words = {"w", "wo", "wor", "worl", "world"};
            String longestWord = longestWord(words);
            System.out.println("Longest word : " + longestWord);
        }
}
```

OUTPUT

```
Longest word : world
```

4.Program for Increasing Order Search Tree

```java
class TreeNodes {
    int val;
    TreeNodes left;
    TreeNodes right;

    TreeNodes(int val) {
        this.val = val;
    }
}

public class IncreasingOrderSearchTree {
    TreeNodes current;

    public TreeNodes increasingBST(TreeNodes root) {
       TreeNodes treenode = new TreeNodes(0);
       current = treenode;
       inOrderTraversal(root);
       return treenode.right;

    }

    private void inOrderTraversal(TreeNodes node) {
        if (node == null) {
            return;
        }

        inOrderTraversal(node.left);

        // Modify the current node's pointers to create the new tree
        node.left = null;
        current.right = node;
        current = node;

        inOrderTraversal(node.right);
    }

    public static void main(String[] args) {
        TreeNodes root = new TreeNodes(5);
        root.left = new TreeNodes(3);
        root.right = new TreeNodes(6);
        root.left.left = new TreeNodes(2);
        root.left.right = new TreeNodes(4);
        root.left.left.left = new TreeNodes(1);
        root.right.right = new TreeNodes(8);
        root.right.right.left = new TreeNodes(7);
        root.right.right.right = new TreeNodes(9);

        IncreasingOrderSearchTree trees = new IncreasingOrderSearchTree();
        TreeNodes newRoot = trees.increasingBST(root);

        System.out.println("In-order traversal of the new tree:");
        printTree(newRoot);
    }

    // Helper function to print the tree in-order
    private static void printTree(TreeNodes node) {
        if (node == null) {
            return;
        }
        printTree(node.left);
        System.out.print(node.val + " ");
        printTree(node.right);
    }

}
```

OUTPUT

```
In-order traversal of the new tree:

1 2 3 4 5 6 7 8 9
```

5.Program for Univalued Binary Tree

```
class TreeNodeu {
    int val;
    TreeNodeu left;
    TreeNodeu right;
    TreeNodeu(int val) {
        this.val = val;
    }
}

public class UnivaluedBinaryTree {
    public static boolean isUnivalTree(TreeNodeu root) {
        if (root == null) {
            return true;
        }
        // Check if the left child exists and has a different value
        if (root.left != null && root.left.val != root.val) {
            return false;
        }
        // Check if the right child exists and has a different value
        if (root.right != null && root.right.val != root.val) {
            return false;
        }
        // Recursively check the left and right subtrees
        return isUnivalTree(root.left) && isUnivalTree(root.right);
    }

    public static void main(String[] args) {
        TreeNodeu root = new TreeNodeu(1);
        root.left = new TreeNodeu(1);
        root.right = new TreeNodeu(1);
        root.left.left = new TreeNodeu(1);
        root.left.right = new TreeNodeu(1);
        root.right.right = new TreeNodeu(1);
        boolean isUnivalued = isUnivalTree(root);
        if (isUnivalued) {
            System.out.println("The binary tree is univalued.");
        } else {
            System.out.println("The binary tree is not univalued.");
        }
    }
}
```

OUTPUT

```
The binary tree is univalued.
```

6.Program for Day of the Year

```
public class DayOfYear {
    public int dayOfYear(String date) {
        int[] daysInMonth = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 
31};
        String[] datePart = date.split("-");

        int year = Integer.parseInt(datePart[0]);
        int month = Integer.parseInt(datePart[1]);
        int day = Integer.parseInt(datePart[2]);

        // Check if the given year is a leap year
        boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 
400 == 0);
        int dayCount = day;

        // Calculate the day count by adding days of previous months
        for (int i = 1; i < month; i++) {
            dayCount += daysInMonth[i];
        }

        // If it's a leap year and after February, add an extra day
        if (isLeapYear && month > 2) {
            dayCount++;
        }

        return dayCount;
    }

    public static void main(String[] args) {
        DayOfYear dy = new DayOfYear();
        String date = "2023-08-02";
        int dayNumber = dy.dayOfYear(date);

        System.out.println("Day of the year for " + date + " is: " + 
dayNumber);
    }
}
```

OUTPUT

```
Day of the year for 2023-08-02 is: 214
```

7.Program for Day of the Week

```java
public class DayOfWeek {
    public String dayOfWeek(int day, int month, int year) {
        // Array to store days of the week
        String[] daysOfWeek = {"Saturday", "Sunday", "Monday", "Tuesday", 
"Wednesday", "Thursday", "Friday"};

        // Adjust month and year for January and February
        if (month < 3) {
            month += 12;
            year--;
        }

        // Calculate k and j
        int k = year % 100; // Last two digits of the year
        int j = year / 100; // First two digits of the year

        // Calculate the day of the week index
        int dayOfWeekIndex = (day + 13 * (month + 1) / 5 + k + k / 4 + j / 4 + 5 * j) 
% 7;

        return daysOfWeek[dayOfWeekIndex];
    }

    public static void main(String[] args) {
        DayOfWeek dw = new DayOfWeek();
        int day = 28;
        int month = 8;
        int year = 2023;

        String dayOfWeek = dw.dayOfWeek(day, month, year);

        System.out.println("Day of the week for " + day + "-" + month + "-" + year + 
" is: " + dayOfWeek);
    }
}
```

OUTPUT

```
Day of the week for 28-8-2023 is: Monday
```

8.Program for Check If a Word Occurs As a Prefix of Any Word in a Sentence

```java
public class PrefixCheck {
    public static int isPrefixOfWord(String sentence, String searchWord) {
        String[] words = sentence.split(" ");

        for (int i = 0; i < words.length; i++) {
            if (words[i].startsWith(searchWord)) {
                return i + 1; // Return 1-based index
            }
        }

        return -1; // Return -1 if not found
    }

    public static void main(String[] args) {
        String sentence = "i love eating burger";
        String searchWord = "love";

        int index = isPrefixOfWord(sentence, searchWord);

        if (index != -1) {
            System.out.println("The word (" + searchWord + ") set at " + index);
        } else {
            System.out.println("The word (" + searchWord + ") does not occur .");
        }
    }
}
```

OUTPUT

```
The word (love) set at 2
```

9.Program for Minimum Insertions to Balance a Parentheses String

```
import java.util.Stack;

public class MinInsertionsToBalanceParentheses {

    public static int minAddToMakeValid(String s) {
      Stack<Character> stack = new Stack<>();
      int insertions = 0;

      for (char c : s.toCharArray()) {
        if (c == '(') {
          stack.push(c);
        } else if (c == ')') {
          if (!stack.isEmpty() && stack.peek() == '(') {
            stack.pop(); // Matched a pair
          } else {
            insertions++; // Need to insert an opening parenthesis
          }
        }
      }

      // For each remaining opening parenthesis in the stack, insert a closing 
parenthesis
      insertions += stack.size();

      return insertions;
    }

    public static void main(String[] args) {

      String parenthesesString =   "(()))";

      int minInsertions = minAddToMakeValid(parenthesesString);


      System.out.println("Minimum insertions to balance the parentheses string: 
" + minInsertions);
    }
  }
```

OUTOUT

```
Minimum insertions to balance the parentheses string: 1
```

10.Program for Convert 1D Array Into 2D Array

```java
public class Convert1DArrayTo2DArray {

    public static int[][] convertTo2DArray(int[] nums, int rows, int cols) {
        // Check if the number of elements matches the specified dimensions
        if (nums.length != rows * cols) {
            throw new IllegalArgumentException("Number of elements in 1D array does not match rows * cols");
        }

        // Create a new 2D array
        int[][] result = new int[rows][cols];
        int index = 0;

        // Fill the 2D array with elements from the 1D array
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = nums[index++];
            }
        }

        return result;
    }

    public static void main(String[] args) {

        int[] nums = {1, 2, 3, 4, 5, 6};
        int rows = 2;
        int cols = 3;

        int[][] result = convertTo2DArray(nums, rows, cols);

        System.out.println("Converted 2D array:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

OUTPUT

```
Converted 2D array:

1 2 3

4 5 6
```

11.Program for Vowels of All Substrings

```java
public class VowelsInSubstring {

        public static long countVowels(String word) {
            long lengthword = word.length();
            long result = 0;

            // Loop through the characters of the word
            for (int i = 0; i < lengthword; i++) {
                char x = word.charAt(i);

                // Check if the character is a vowel
                if (x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u') {
                    // Calculate and add the contribution of the current vowel
                    result = result+(i + 1) * (lengthword - i);
                }
                System.out.println(result);
            }

            return result; // Return the sum of occurrences
        }

        public static void main(String[] args) {

            String word = "aba";
            long sum = countVowels(word);

            System.out.println("Word: " + word);
            System.out.println("Sum of vowel occurrences in all substrings: " + sum);
        }
    }
```

OUTPUT

```
Word: aba

Sum of vowel occurrences in all substrings: 6
```

12.Program for Number of Common Factors

```
public class CommonFactors {

    // Method to calculate the greatest common divisor (GCD) of two numbers
    public static int  greatestCommonDivisor (int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }

    // Method to count the number of common factors between two numbers
    public static int countCommonFactors(int a, int b) {
        int gcdValue = greatestCommonDivisor(a, b);
        int count = 0;

        // Count the factors of the GCD
        for (int i = 1; i <= gcdValue; i++) {
            if (gcdValue % i == 0) {
                count++;
            }
        }

        return count;
    }

    public static void main(String[] args) {
        int num1 = 12;
        int num2 = 18;

        int commonFactorsCount = countCommonFactors(num1, num2);

        System.out.println("Number of common factors : (" + num1 + " , " + num2 + ")
: " + commonFactorsCount);
    }
}
```

OUTPUT

```
Number of common factors : (12 , 18) : 4
```

13.Program for Find Closest Number to Zero

```
public class ClosestNumberToZero {

    public static int findClosestToZero(int[] nums) {

        int result = Integer.MAX_VALUE;

        for(int i: nums)

            if(Math.abs(i) < Math.abs(result) || i == Math.abs(result))
            {
                result = i;
            }
        return result;
    }

    public static void main(String[] args) {

        int[] nums = {-9, 5, 3, -2, -4, 8, -7};

        int closestToZero = findClosestToZero(nums);

        System.out.println("Number :");

        for(int i=0;i<nums.length; i++) {

           System.out.print(nums[i]+"  ");
         }

       System.out.println();

        System.out.println("Closest number to zero: " + closestToZero);
    }
}
```

OUTPUT

```
Number :

-9  5  3  -2  -4  8  -7

Closest number to zero: -2
```