Check if two Strings are anagrams of each other

```java
/*
 * Check if two Strings are anagrams of each other
   Problem Statement: Given two strings, check if two strings are anagrams of each other or not.

   Examples:

   Example 1:
   Input: CAT, ACT
   Output: true
   Explanation: Since the count of every letter of both strings are equal.

   Example 2:
   Input: RULES, LESRT
   Output: false
   Explanation: Since the count of U and T  is not equal in both strings.
*/


public class Valid_Anagram {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static String SortString(String str){
        char c[] = str.toCharArray();
        Arrays.sort(c);
        return new String(c);
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static boolean isAnagram(String s, String t) {
        /*
         * BruteForce approach: Time complexity: O(N^2) & Space complexity: O(N).
         * if(s.length() != t.length()){
                return false;
           }
           HashMap<Character, Integer> map = new HashMap<>();
           for(int i = 0; i < s.length(); i++){
               if(!map.containsKey(s.charAt(i))){
                   map.put(s.charAt(i), 1);
               }
               else{
                   map.put(s.charAt(i), map.get(s.charAt(i)) + 1);
               }
           }
           int count = 0;
           for(int i = 0; i < s.length(); i++){
               char c = s.charAt(i);
               count = 0;
               for(int j = 0; j < t.length(); j++){
                   if(c == t.charAt(j)){
                       count++;
                   }
               }
               if(map.get(c) != count){
                   return false;
               }
           }
           return true;
        */
```

```
        /*
         * Solution 2: Time complexity: O(NlogN) & Space complexity: O(1).
         * if(s.length() != t.length()){
             return false;
           }
           s = SortString(s);
           t = SortString(t);

           for(int i = 0; i < s.length(); i++){
              if(s.charAt(i) != t.charAt(i)){
                 return false;
              }
           }
           return true;
        */

        /*

           // Solution 2: Time complexity: O(N) & Space complexity: O(1).

           if(s.length() != t.length())return false;

           HashMap<Character, Integer> map = new HashMap<>();
           for(int i = 0; i < s.length(); i++){
              if(map.containsKey(s.charAt(i))){
                 map.put(s.charAt(i), map.get(s.charAt(i)) + 1);
              }
              else{
                 map.put(s.charAt(i), 1);
              }
           }
           for(int i = 0; i < t.length(); i++){
              if(!map.containsKey(t.charAt(i))){
                 return false;
              }
              else if(map.get(t.charAt(i)) == 1){
                 map.remove(t.charAt(i));
              }
              else{
                 map.put(t.charAt(i), map.get(t.charAt(i)) - 1);
              }
           }
           return map.size() == 0;
        */

        // Optimized Approach: Time complexity: O(N) & Space compelxity: O(1).

```

```
        // Optimized Approach: Time complexity: O(N) & Space compelxity: O(1).

        if (s.length() != t.length())return false;

        int[] freq = new int[26];
        for (int i = 0; i < s.length(); i++) {
          freq[s.charAt(i) - 'a']++;
        }
        for (int i = 0; i < t.length(); i++) {
          freq[t.charAt(i) - 'a']--;
        }
        for (int i = 0; i < 26; i++) {
          if (freq[i] != 0)
            return false;
        }
        return true;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String s = "anagram", t = "nagaram";
        System.out.println(isAnagram(s, t));
    }
}

```

```
> Task :Valid_Anagram.main()
true
```

check_whether_one_string_is_a_rotation_of_another

```
/*
 * Given two strings s and goal, return true if and only if s can become goal after some number of shifts on s.

   A shift on s consists of moving the leftmost character of s to the rightmost position.

   For example, if s = "abcde", then it will be "bcdea" after one shift.


   Example 1:

   Input: s = "abcde", goal = "cdeab"
   Output: true
   Example 2:

   Input: s = "abcde", goal = "abced"
   Output: false
*/

```

```java
Codeium: Refactor | Explain
public class check_whether_one_string_is_a_rotation_of_another {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static boolean rotateString(String s, String goal) {
        /*
           BruteForce Appraoch: Time complexity: O(N^2) & Space complexity: O(1)

           if(s.length() != goal.length()){
               return false;
           }
           for(int i = 0; i < s.length(); i++){
               char lastIdx = goal.charAt(goal.length() - 1);
               for(int j = goal.length() - 2; j >= 0; j--){
                   char c = goal.charAt(j);
                   goal = goal.substring(0, j + 1) + c + goal.substring(j + 2);
               }
               goal = goal.substring(0, 0) + lastIdx + goal.substring(1);
               boolean flag = true;
               for(int j = 0; j < goal.length(); j++){
                   if(s.charAt(j) != goal.charAt(j)){
                       flag = false;
                   }
               }
               if(flag){
                   return true;
               }
           }
           return false;
        */

        // optimized Appraoch: Time complexity: O(1) & Space complexity: O(1)

        if(s.length() == goal.length() && (s + s).contains(goal)){
            return true;
        }
        return false;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String s = "abcde", goal = "cdeab";
        System.out.println(rotateString(s, goal));
    }
}

```

```
> Task :check_whether_one_string_is_a_rotation_of_another.main()
true
```

Isomorphic_Strings

```
/*
 * Given two strings s and t, determine if they are isomorphic.

   Two strings s and t are isomorphic if the characters in s can be replaced to get t.

   All occurrences of a character must be replaced with another character while preserving the order of characters.
   No two characters may map to the same character, but a character may map to itself.



   Example 1:

   Input: s = "egg", t = "add"
   Output: true
   Example 2:

   Input: s = "foo", t = "bar"
   Output: false
   Example 3:

   Input: s = "paper", t = "title"
   Output: true
*/
```

```java
import java.util.HashMap;

Codeium: Refactor | Explain
public class Isomorphic_Strings {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static boolean isIsomorphic(String s, String t) {
        if(s.length() != t.length()){
            return false;
        }
        HashMap<Character, Character> map1 = new HashMap<>();
        HashMap<Character, Boolean> map2 = new HashMap<>();
        for(int i = 0; i < s.length(); i++){
            char ch1 = s.charAt(i);
            char ch2 = t.charAt(i);

            if(map1.containsKey(ch1)){
               if(map1.get(ch1) != ch2){
                  return false;
               }
            }
            else{
                if(map2.containsKey(ch2)){
                    return false;
                }
                else{
                    map1.put(ch1, ch2);
                    map2.put(ch2, true);
                }
            }
        }
        return true;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String s = "egg";
        String t = "add";
        System.out.println(isIsomorphic(s, t));
    }
}

```

```
> Task :Isomorphic_Strings.main()
true
```

Largest_Odd_Number_in_String

```java
/*
 * You are given a string num, representing a large integer. Return the largest-valued odd integer
  (as a string) that is a
  non-empty substring of num, or an empty string "" if no odd integer exists.

   A substring is a contiguous sequence of characters within a string.

   Example 1:

   Input: num = "52"
   Output: "5"
   Explanation: The only non-empty substrings are "5", "2", and "52". "5" is the only odd number.
   Example 2:

   Input: num = "4206"
   Output: ""
   Explanation: There are no odd numbers in "4206".
   Example 3:

   Input: num = "35427"
   Output: "35427"
   Explanation: "35427" is already an odd number.
*/

Codeium: Refactor | Explain
public class Largest_Odd_Number_in_String {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static String largestOddNumber(String nums) {
        // Time complexity: O(N) & Space complexity: O(1).
        String ans = "";
        for(int i = nums.length() - 1; i >= 0; i--){
            char c = nums.charAt(i);
            if(c % 2 != 0){
                ans = nums.substring(0, i + 1);
                return ans;
            }
        }
        return "";
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String str = "52";
        System.out.println(largestOddNumber(str));
    }
}

```

> Task :Largest_Odd_Number_in_String.main()

5

longestCommonPrefix

```java
Codeium: Refactor | Explain
public class Longest_Common_Prefix {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public String longestCommonPrefix(String[] strs) {
        /*
         * BruteForce Approach: Time complexity: O(firstStringLength * N) & Space complexity: O(1).
         * String str = strs[0];
           String finalAns = "";
           for(int i = 0; i < str.length(); i++){
               for(int j = 0; j < strs.length; j++){
                   String curr = strs[j];
                   if(i + 1> curr.length() || str.charAt(i) != curr.charAt(i)){
                       return finalAns;
                   }
               }
               finalAns += str.charAt(i);
           }
           return finalAns;
        */

        // Optimized: Used StringBuilder

        String str = strs[0];
        StringBuilder finalAns = new StringBuilder();
        for(int i = 0; i < str.length(); i++){
            for(int j = 0; j < strs.length; j++){
                String curr = strs[j];
                if(i + 1> curr.length() || str.charAt(i) != curr.charAt(i)){
                    return finalAns.toString();
                }
            }
            finalAns.append(str.charAt(i));
        }
        return finalAns.toString();
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String[] strs = {"flower","flow","flight"};
        System.out.println(new
                Longest_Common_Prefix().longestCommonPrefix(strs));

    }
}
```

```
> Task :Longest_Common_Prefix.main()
fl
```

Remove_Outermost_Parentheses

```
    Example 1:

    Input: s = "(()())(())"
    Output: "()()()"
    Explanation:
    The input string is "(()())(())", with primitive decomposition "(()())" + "(())".
    After removing outer parentheses of each part, this is "()()" + "()" = "()()()".

    Example 2:

    Input: s = "(()())(())(()(()))"
    Output: "()()()()(())"
    Explanation:
    The input string is "(()())(())(()(()))", with primitive decomposition "(()())" + "(())" + "(()(()))".
    After removing outer parentheses of each part, this is "()()" + "()" + "()(())" = "()()()()(())".

    Example 3:

    Input: s = "()()"
    Output: ""
    Explanation:
    The input string is "()()", with primitive decomposition "()" + "()".
    After removing outer parentheses of each part, this is "" + "" = "".
*/
```

```java
public class Remove_Outermost_Parentheses {
    public static String removeOuterParentheses(String str) {
        /*
           Time complexity: O(N^2) & Space complexity: O(1).
           N^2 due to every time creating a new string object

           String ans = "";
           int j = 0;
           for(int i = 0; i < str.length() - 1; i++){
               if(str.charAt(i) == ')'){
                   j--;
               }
               if(j != 0){
                   ans += str.charAt(i);
               }
               if(str.charAt(i) == '('){
                   j++;
               }
           }
           return ans;
        */

        // Time complexity: O(N) & Space complexity: O(1).

        StringBuilder ans = new StringBuilder();
        int j = 0;
        for(int i = 0; i < str.length() - 1; i++){
            if(str.charAt(i) == ')'){
                j--;
            }
            if(j != 0){
                ans.append(str.charAt(i));
            }
            if(str.charAt(i) == '('){
                j++;
            }
        }
        return ans.toString();
    }
    public static void main(String[] args) {
        String str = "((()())(()))";
        System.out.println(removeOuterParentheses(str));
    }
}
```

```
> Task :Remove_Outermost_Parentheses.main()
()()()
```

Reverse Words in a String

```java
/*
 * Reverse Words in a String
   Problem Statement: Given a string s, reverse the words of the string.

   Examples:

   Example 1:
   Input: s=”this is an amazing program”
   Output: “program amazing an is this”

   Example 2:
   Input: s=”This is decent”
   Output: “decent is This”
*/


Codeium: Refactor | Explain
public class Reverse_Words_in_a_String {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static String reverseWords(String str) {
        String ans = "";
        int i = str.length() - 1;
        while(i >= 0){
            while(i >= 0 && str.charAt(i) == ' '){
                i--;
            }
            int j = i;
            if(i < 0){
                break;
            }
            while(i >= 0 && str.charAt(i) != ' '){
                i--;
            }
            if(ans.isEmpty()){
                ans = ans.concat(str.substring(i+1, j+1));
            }
            else{
                ans = ans.concat(" " + str.substring(i+1, j+1));
            }
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String str = "the sky is blue";
        System.out.println(reverseWords(str));
    }
}

```

```
> Task :Reverse_Words_in_a_String.main()
blue is sky the
```

Count_number_of_substrings

```java
/*
 * Given a string of lowercase alphabets, count all possible substrings (not necessarily distinct)
  that have exactly k distinct characters.


   Example 1:

   Input:
   S = "aba", K = 2
   Output:
   3
   Explanation:
   The substrings are:
   "ab", "ba" and "aba".

   Example 2:

   Input:
   S = "abaaca", K = 1
   Output:
   7
   Explanation:
   The substrings are:
   "a", "b", "a", "aa", "a", "c", "a".
*/


public class Count_number_of_substrings {
    public static int substrCount (String str, int k) {
        // Time complexity: O(N^2) & Space complexity: O(1).

        int result = 0 ;
        int n = str.length();
        int cnt[] = new int[26];
        for(int i = 0; i < n; i++){
            int dist_count = 0 ;
            Arrays.fill(cnt, 0);
            for (int j = i; j < n ; j++ ){
                if(cnt[str.charAt(j) - 'a'] == 0){
                    dist_count++;
                }
                cnt[str.charAt(j) - 'a']++;
                if(dist_count == k){
                    result++;
                }
            }
        }
        return result;
    }
    public static void main(String[] args) {
        String s = "abaaca";
        int k = 1;
        System.out.println(substrCount(s, k));
    }
}
```

```
> Task :Count_number_of_substrings.main()
7
```

Implement_Atoi

```
/*
  Q. String to Integer (atoi) (leetcode 8).

  Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

  The algorithm for myAtoi(string s) is as follows:

  Read in and ignore any leading whitespace.
  Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines
  if the final result is negative or
   positive respectively. Assume the result is positive if neither is present.
  Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
  Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the
   sign as necessary (from step 2).
  If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in
  the range. Specifically,
   integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
  Return the integer as the final result.
  Note:

  Only the space character ' ' is considered a whitespace character.
  Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.


  Example 1:

  Input: s = "42"
  Output: 42
  Explanation: The underlined characters are what is read in, the caret is the current reader position.
  Step 1: "42" (no characters read because there is no leading whitespace)
    ^
  Step 2: "42" (no characters read because there is neither a '-' nor '+')
    ^
  Step 3: "42" ("42" is read in)
      ^
  The parsed integer is 42.
  Since 42 is in the range [-231, 231 - 1], the final result is 42.
  Example 2:

  Input: s = "   -42"
  Output: -42
  Explanation:
  Step 1: "   -42" (leading whitespace is read and ignored)
     ^
  Step 2: "   -42" ('-' is read, so the result should be negative)
      ^
  Step 3: "   -42" ("42" is read in)
       ^
  The parsed integer is -42.
  Since -42 is in the range [-231, 231 - 1], the final result is -42.
  Example 3:
  The parsed integer is -42.
  Since -42 is in the range [-231, 231 - 1], the final result is -42.
  Example 3:

  Input: s = "4193 with words"
  Output: 4193
  Explanation:
  Step 1: "4193 with words" (no characters read because there is no leading whitespace)
    ^
  Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
    ^
  Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
       ^
  The parsed integer is 4193.
  Since 4193 is in the range [-231, 231 - 1], the final result is 4193.
*/
```

```java
Codeium: Refactor | Explain
public class Implement_Atoi {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int myAtoi(String s) {
        if(s.length() == 0){
            return 0;
        }
        for(int i = 0; i < s.length(); i++){
            if(s.charAt(i) != ' '){
                s = s.substring(i, s.length());
                break;
            }
        }

        int max = Integer.MAX_VALUE;
        int min = Integer.MIN_VALUE;

        long ans = 0;
        int sign = 1;

        if(s.charAt(0) == '-'){
            sign = -1;
        }
        int i = (s.charAt(0) == '+' || s.charAt(0) == '-') ? 1 : 0;
        while(i < s.length()){
            if(s.charAt(i) == '-' || !Character.isDigit(s.charAt(i))){
                return sign * (int)(ans);
            }
            ans = ans * 10 + (s.charAt(i) - '0');
            if(sign == -1 && -1 * ans < min){
                return min;
            }
            if(sign == 1 && ans > max){
                return max;
            }
            i++;

        }
        return sign * (int)(ans);
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String s = "   -42";
        System.out.println(myAtoi(s));
    }
}

```

```
> Task :Implement_Atoi.main()
-42
```

Longest_Palindromic_Substring

```
/*
 * Given a string s, return the longest palindromic substring in s.

   A string is called a palindrome string if the reverse of that string is the same as the original string.

   Example 1:

   Input: s = "babad"
   Output: "bab"
   Explanation: "aba" is also a valid answer.
   Example 2:

   Input: s = "cbbd"
   Output: "bb"
*/
```

```java
public class Longest_Palindromic_Substring {
    public static boolean palindrome(String str){
        // This function will be used in first BruteForce Mehtod
        int i = 0;
        int j = str.length() - 1;
        while(i < j){
            if(str.charAt(i) != str.charAt(j)){
                return false;
            }
            i++; j--;
        }
        return true;
    }
    public static String longestPalindrome(String s) {
        /*
         * BruteForce Approach: Time complexity: O(N^3) & Space complexity: O(1).
         * int max = Integer.MIN_VALUE;
           StringBuilder ans = new StringBuilder("temp");
           for(int i = 0; i < s.length(); i++){
               for(int j = i; j < s.length(); j++){
                   boolean possible = palindrome(s.substring(i, j + 1));
                   if(possible){
                       if(max < ((j - i) + 1)){
                           max = ((j - i) + 1);
                           ans.replace(0, ans.length(), s.substring(i, j + 1));
                       }
                   }
               }
           }
           return ans.toString();
        */
        int start = 0;
        int maxLen = 1;
        int l, r;
        for(int i = 0; i < s.length(); i++){
            // even
            l=i-1;
            r=i ;
            while(l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)){
                if(r-l+1>maxLen){
                    maxLen=r-l+1;
                    start=l;
                }
                l-=1;
                r+=1;
            }
```

```
            // odd
            l=i-1;
            r=i+1;
            while(l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)){
                if( r-l+1>maxLen){
                    maxLen=r-l+1;
                    start=l;
                }
                l-=1;
                r+=1;
            }
        }
        return s.substring(start, start + maxLen);
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String s = "babad";
        System.out.println(longestPalindrome(s));
    }
}

```

```
> Task :Longest_Palindromic_Substring.main()
bab
```

Maximum_Nesting_Depth_of_the_Parentheses

```
/*
 * Q. Write a program to Count Maximum Nesting Depth of the Parentheses.
 *
 * Example 1:

   Input: s = "(1+(2*3)+((8)/4))+1"
   Output: 3
   Explanation: Digit 8 is inside of 3 nested parentheses in the string.

   Example 2:

   Input: s = "(1)+((2))+(((3)))"
   Output: 3
*/

// import java.util.ArrayList;

```

```java
Codeium: Refactor | Explain
public class Maximum_Nesting_Depth_of_the_Parentheses {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int maxDepth(String str) {
        /*
         * BruteForce Approach: Time complexity: O(N) & Space complexity: O(N).
         * ArrayList<Integer> temp = new ArrayList<>();
           int j = 0;
           int max = Integer.MIN_VALUE;
           for(int i = 0; i < str.length(); i++){
               if(str.charAt(i) == '('){
                   j++;
                   max = Math.max(max, j);
               }
               if(str.charAt(i) == ')'){
                   j--;
                   if(j == 0){
                       temp.add(max);
                       max = Integer.MIN_VALUE;
                   }
               }
           }
           int ans = 0;
           for(int i = 0; i < temp.size(); i++){
               ans = Math.max(ans, temp.get(i));
           }
           return ans;
        */

        // Optimized Approach: Time complexity: O(N) & Space complexity: O(1).

        int ans = 0, j = 0;
        for(int i = 0; i < str.length(); i++){
            if(str.charAt(i) == '('){
                j++;
            }
            if(str.charAt(i) == ')'){
                j--;
            }
            ans = Math.max(ans, j);
        }
        return ans;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String str = "(1+(2*3)+((8)/4))+1";
        System.out.println(maxDepth(str));
    }
}

```

```
> Task :Maximum_Nesting_Depth_of_the_Parentheses.main()
3
```

Roman_Number_to_Integer_and_vice_versa

```java
/*
 * Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

   Symbol       Value
   I             1
   V             5
   X             10
   L             50
   C             100
   D             500
   M             1000
   For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply
   X + II. The number 27 is written as XXVII, which is XX + V + II.

   Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII.
   Instead, the number four is written as IV. Because the one is before the five we subtract it making four.
   The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

   I can be placed before V (5) and X (10) to make 4 and 9.
   X can be placed before L (50) and C (100) to make 40 and 90.
   C can be placed before D (500) and M (1000) to make 400 and 900.
   Given a roman numeral, convert it to an integer.



   Example 1:

   Input: s = "III"
   Output: 3
   Explanation: III = 3.
   Example 2:

   Input: s = "LVIII"
   Output: 58
   Explanation: L = 50, V= 5, III = 3.
   Example 3:

   Input: s = "MCMXCIV"
   Output: 1994
   Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
*/
Codeium: Refactor | Explain
public class Roman_Number_to_Integer_and_vice_versa {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static int romanToInt(String str) {
        HashMap<Character, Integer> map = new HashMap<>();
        map.put('I', 1);
        map.put('V', 5);
        map.put('X', 10);
        map.put('L', 50);
        map.put('C', 100);
        map.put('D', 500);
        map.put('M', 1000);

        /*
         * BruteForce Approach: Time complexity: O(N) & Space complexity: O(1). *
         * if(str.length() == 1){
              return map.get(str.charAt(0));
            }
            int num = 0;
            for(int i = 0; i < str.length() - 1; i++){
                if((str.charAt(i) == 'I' && (str.charAt(i + 1) == 'X' || str.charAt(i + 1) == 'V'))|| (str.charAt(i) == 'X' &&
                (str.charAt(i + 1) == 'L' || str.charAt(i + 1) == 'C'))|| (str.charAt(i) == 'C' && (str.charAt(i + 1) == 'D'
                 || str.charAt(i + 1) == 'M'))){
                    num += map.get(str.charAt(i + 1)) - map.get(str.charAt(i));
                    i++;
                    if(i == str.length() - 1){
                        return num;
                    }
                }
                else{
                    num+= map.get(str.charAt(i));
                }
            }
            int i = str.length() - 1;
            if((str.charAt(i - 1) == 'I' && (str.charAt(i) == 'X' || str.charAt(i) == 'V'))|| (str.charAt(i - 1) == 'X'
            && (str.charAt(i) == 'L' || str.charAt(i) == 'C'))|| (str.charAt(i - 1) == 'C' &&
            (str.charAt(i) == 'D' || str.charAt(i) == 'M'))){
                num -= map.get(str.charAt(i - 1));
                num += map.get(str.charAt(i)) - map.get(str.charAt(i - 1));
            }
            else{
                num += map.get(str.charAt(i));
            }
            return num;
        */
```

```java
        // Same Approach in a little Optimized way. Time complexity: O(N) & Space complexity: O(1).

        int num = map.get(str.charAt(str.length() - 1));
        for(int i = str.length() - 2; i >= 0; i--){
            if(map.get(str.charAt(i)) < map.get(str.charAt(i + 1))){
                num -= map.get(str.charAt(i));
            }
            else{
                num += map.get(str.charAt(i));
            }
        }
        return num;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String str = "MCMXCIV";
        System.out.println(romanToInt(str));
    }
}

```

```
> Task :Roman_Number_to_Integer_and_vice_versa.main()
1994
```

Sort_Characters_by_frequency

```java
/*
 * Given a string s, sort it in decreasing order based on the frequency of the characters.
 The frequency of a character is the number of times it appears in the string.

   Return the sorted string. If there are multiple answers, return any of them.

   Example 1:

   Input: s = "tree"
   Output: "eert"
   Explanation: 'e' appears twice while 'r' and 't' both appear once.
   So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

   Example 2:

   Input: s = "cccaaa"
   Output: "aaaccc"
   Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.
   Note that "cacaca" is incorrect, as the same characters must be together.

   Example 3:

   Input: s = "Aabb"
   Output: "bbAa"
   Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
   Note that 'A' and 'a' are treated as two different characters.
*/

```

```java
Codeium: Refactor | Explain
public class Sort_Characters_by_frequency {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static String frequencySort(String s) {
        HashMap<Character, Integer> map = new HashMap<>();
        for(int i = 0; i < s.length(); i++){
            if(map.containsKey(s.charAt(i))){
                map.put(s.charAt(i), map.get(s.charAt(i)) + 1);
            }
            else{
                map.put(s.charAt(i), 1);
            }
        }
        List<Character>[] bucket = new List[s.length() + 1];
        for(Character key : map.keySet()){
            int frequency = map.get(key);
            if(bucket[frequency] == null){
                bucket[frequency] = new ArrayList<>();
            }
            bucket[frequency].add(key);
        }
        StringBuilder sb = new StringBuilder();
        for(int i = bucket.length - 1; i >= 0; i--){
            if(bucket[i] != null){
                for(char c : bucket[i]){
                    for(i = 0; i < map.get(c); i++){
                        sb.append(c);
                    }
                }
            }
        }
        return sb.toString();
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        String s = "tree";
        System.out.println(frequencySort(s));
    }
}

```

```
> Task :Sort_Characters_by_frequency.main()
eert
```

Sum_of_Beauty_of_All_Substrings

```java
/*
   he beauty of a string is the difference in frequencies between the most frequent and least frequent characters.

   For example, the beauty of "abaacc" is 3 - 1 = 2.
   Given a string s, return the sum of beauty of all of its substrings.

   Example 1:

   Input: s = "aabcb"
   Output: 5
   Explanation: The substrings with non-zero beauty are ["aab","aabc","aabcb","abcb","bcb"], each with beauty equal to 1.

   Example 2:

   Input: s = "aabcbaa"
   Output: 17
*/
public class Sum_of_Beauty_of_All_Substrings {
    public static int findBeauty2(int[] cnt){
        int max = -1;
        int min = Integer.MAX_VALUE;
        for(int i = 0; i < 26; i++){
            max = Math.max(max, cnt[i]);
            if(cnt[i] >= 1){
                min = Math.min(min, cnt[i]);
            }
        }
        return max - min;
    }
    public static int findBeauty(String str){
        if(str.length() == 0){
            return 0;
        }
        HashMap<Character, Integer> map = new HashMap<>();
        for(int i = 0; i < str.length(); i++){
            if(map.containsKey(str.charAt(i))){
                map.put(str.charAt(i), map.get(str.charAt(i)) + 1);
            }
            else{
                map.put(str.charAt(i), 1);
            }
        }
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;
        for(Map.Entry<Character, Integer> entry: map.entrySet()){
            max = Math.max(entry.getValue(), max);
            min = Math.min(entry.getValue(), min);
        }
        return max - min;
    }
```

```
Codeium: Refactor | Explain | Generate Javadoc | ×
57      public static int beautySum(String s) {
58          /*
59             BruteForce Appraoch: Time complexity: O(N^3) & Space complexity: O(N).
60             int beauty = 0;
61             for(int i = 0; i < s.length(); i++){
62                 for(int j = i; j < s.length(); j++){
63                     beauty += findBeauty(s.substring(i, j + 1));
64                 }
65             }
66             return beauty;
67          */
68
69          // Optimized Appraoch: Time complexity: O(N^2) & Space Complexity: O(1).
70          int beauty = 0;
71          int cnt[] = new int[26];
72          for(int i = 0; i < s.length(); i++){
73              Arrays.fill(cnt, 0);
74              for(int j = i; j < s.length(); j++){
75                  cnt[s.charAt(j) - 'a']++;
76                  beauty += findBeauty2(cnt); // This is using another function
77              }
78          }
79          return beauty;
80      }
Codeium: Refactor | Explain | Generate Javadoc | ×
81      public static void main(String[] args) {
82          String s = "aabcb";
83          System.out.println(beautySum(s));
84      }
85  }
```

```
> Task :Sum_of_Beauty_of_All_Substrings.main()
5
```