

Check if two Strings are anagrams of each other

```
1  /*
2  * Check if two Strings are anagrams of each other
3  Problem Statement: Given two strings, check if two strings are anagrams of each other or not.
4
5  Examples:
6
7  Example 1:
8  Input: CAT, ACT
9  Output: true
10 Explanation: Since the count of every letter of both strings are equal.
11
12 Example 2:
13 Input: RULES, LESRT
14 Output: false
15 Explanation: Since the count of U and T is not equal in both strings.
16 */
17
```

```
22 public class Valid_Anagram {
23     Codeium: Refactor | Explain | Generate Javadoc | X
24     public static String SortString(String str){
25         char c[] = str.toCharArray();
26         Arrays.sort(c);
27         return new String(c);
28     }
29     Codeium: Refactor | Explain | Generate Javadoc | X
30     public static boolean isAnagram(String s, String t) {
31         /*
32          * BruteForce approach: Time complexity:  $O(N^2)$  & Space complexity:  $O(N)$ .
33          * if(s.length() != t.length()){
34              return false;
35          }
36          HashMap<Character, Integer> map = new HashMap<>();
37          for(int i = 0; i < s.length(); i++){
38              if(!map.containsKey(s.charAt(i))){
39                  map.put(s.charAt(i), 1);
40              }
41              else{
42                  map.put(s.charAt(i), map.get(s.charAt(i)) + 1);
43              }
44          }
45          int count = 0;
46          for(int i = 0; i < s.length(); i++){
47              char c = s.charAt(i);
48              count = 0;
49              for(int j = 0; j < t.length(); j++){
50                  if(c == t.charAt(j)){
51                      count++;
52                  }
53              }
54              if(map.get(c) != count){
55                  return false;
56              }
57          }
58          return true;
59          */
60     }
61 }
```

```

58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105

/*
 * 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 complexity: O(1).

```

```

104         // Optimized Approach: Time complexity: O(N) & Space compelxity: O(1).
105
106         if (s.length() != t.length())return false;
107
108         int[] freq = new int[26];
109         for (int i = 0; i < s.length(); i++) {
110             freq[s.charAt(i) - 'a']++;
111         }
112         for (int i = 0; i < t.length(); i++) {
113             freq[t.charAt(i) - 'a']--;
114         }
115         for (int i = 0; i < 26; i++) {
116             if (freq[i] != 0)
117                 return false;
118         }
119         return true;
120     }
121     Codeium: Refactor | Explain | Generate Javadoc | ✕
122     public static void main(String[] args) {
123         String s = "anagram", t = "nagaram";
124         System.out.println(isAnagram(s, t));
125     }
126

```

```

> Task :Valid_Anagram.main()
true

```

check_whether_one_string_is_a_rotation_of_another

```

1  < /*
2  <  * Given two strings s and goal, return true if and only if s can become goal after some number of shifts on s.
3
4  <  A shift on s consists of moving the leftmost character of s to the rightmost position.
5
6  <  For example, if s = "abcde", then it will be "bcdea" after one shift.
7
8
9  <  Example 1:
10
11 <  Input: s = "abcde", goal = "cdeab"
12 <  Output: true
13 <  Example 2:
14
15 <  Input: s = "abcde", goal = "abced"
16 <  Output: false
17 <  */
18

```

Codeium: Refactor | Explain

```

20 public class check_whether_one_string_is_a_rotation_of_another {
    Codeium: Refactor | Explain | Generate Javadoc | X
21     public static boolean rotateString(String s, String goal) {
22         /*
23          BruteForce Appraoch: Time complexity:  $O(N^2)$  & Space complexity:  $O(1)$ 
24
25          if(s.length() != goal.length()){
26              return false;
27          }
28          for(int i = 0; i < s.length(); i++){
29              char lastIdx = goal.charAt(goal.length() - 1);
30              for(int j = goal.length() - 2; j >= 0; j--){
31                  char c = goal.charAt(j);
32                  goal = goal.substring(0, j + 1) + c + goal.substring(j + 2);
33              }
34              goal = goal.substring(0, 0) + lastIdx + goal.substring(1);
35              boolean flag = true;
36              for(int j = 0; j < goal.length(); j++){
37                  if(s.charAt(j) != goal.charAt(j)){
38                      flag = false;
39                  }
40              }
41              if(flag){
42                  return true;
43              }
44          }
45          return false;
46         */
47
48         // optimized Appraoch: Time complexity:  $O(1)$  & Space complexity:  $O(1)$ 
49
50         if(s.length() == goal.length() && (s + s).contains(goal)){
51             return true;
52         }
53         return false;
54     }
    Codeium: Refactor | Explain | Generate Javadoc | X
55     public static void main(String[] args) {
56         String s = "abcde", goal = "cdeab";
57         System.out.println(rotateString(s, goal));
58     }
59 }
60

```

> Task :check_whether_one_string_is_a_rotation_of_another.main()
true

Isomorphic_Strings

```
1  /*
2  * Given two strings s and t, determine if they are isomorphic.
3
4  Two strings s and t are isomorphic if the characters in s can be replaced to get t.
5
6  All occurrences of a character must be replaced with another character while preserving the order of characters.
7  No two characters may map to the same character, but a character may map to itself.
8
9
10
11 Example 1:
12
13 Input: s = "egg", t = "add"
14 Output: true
15 Example 2:
16
17 Input: s = "foo", t = "bar"
18 Output: false
19 Example 3:
20
21 Input: s = "paper", t = "title"
22 Output: true
23 */
24
```

```

26 import java.util.HashMap;
27
Codeium: Refactor | Explain
28 public class Isomorphic_Strings {
Codeium: Refactor | Explain | Generate Javadoc | X
29     public static boolean isIsomorphic(String s, String t) {
30         if(s.length() != t.length()){
31             return false;
32         }
33         HashMap<Character, Character> map1 = new HashMap<>();
34         HashMap<Character, Boolean> map2 = new HashMap<>();
35         for(int i = 0; i < s.length(); i++){
36             char ch1 = s.charAt(i);
37             char ch2 = t.charAt(i);
38
39             if(map1.containsKey(ch1)){
40                 if(map1.get(ch1) != ch2){
41                     return false;
42                 }
43             }
44             else{
45                 if(map2.containsKey(ch2)){
46                     return false;
47                 }
48                 else{
49                     map1.put(ch1, ch2);
50                     map2.put(ch2, true);
51                 }
52             }
53         }
54         return true;
55     }
Codeium: Refactor | Explain | Generate Javadoc | X
56     public static void main(String[] args) {
57         String s = "egg";
58         String t = "add";
59         System.out.println(isIsomorphic(s, t));
60     }
61 }
62

```

> Task :Isomorphic_Strings.main()

true

Largest_Odd_Number_in_String

```
1  /*
2  * You are given a string num, representing a large integer. Return the largest-valued odd integer
3  * (as a string) that is a
4  * non-empty substring of num, or an empty string "" if no odd integer exists.
5
6  * A substring is a contiguous sequence of characters within a string.
7
8  * Example 1:
9
10 * Input: num = "52"
11 * Output: "5"
12 * Explanation: The only non-empty substrings are "5", "2", and "52". "5" is the only odd number.
13 * Example 2:
14
15 * Input: num = "4206"
16 * Output: ""
17 * Explanation: There are no odd numbers in "4206".
18 * Example 3:
19
20 * Input: num = "35427"
21 * Output: "35427"
22 * Explanation: "35427" is already an odd number.
23 */
24
25 Codeium: Refactor | Explain
26 public class LargestOddNumberInString {
27     Codeium: Refactor | Explain | Generate Javadoc | X
28     public static String largestOddNumber(String nums) {
29         // Time complexity: O(N) & Space complexity: O(1).
30         String ans = "";
31         for(int i = nums.length() - 1; i >= 0; i--){
32             char c = nums.charAt(i);
33             if(c % 2 != 0){
34                 ans = nums.substring(0, i + 1);
35                 return ans;
36             }
37         }
38         return "";
39     }
40
41     Codeium: Refactor | Explain | Generate Javadoc | X
42     public static void main(String[] args) {
43         String str = "52";
44         System.out.println(largestOddNumber(str));
45     }
46 }
```

> Task :Largest_Odd_Number_in_String.main()

5

longestCommonPrefix

19

Codeium: Refactor | Explain

```

20 public class Longest_Common_Prefix {
    Codeium: Refactor | Explain | Generate Javadoc | X
21     public String longestCommonPrefix(String[] strs) {
22         /*
23          * BruteForce Approach: Time complexity: O(firstStringLength * N) & Space complexity: O(1).
24          * String str = strs[0];
25          * String finalAns = "";
26          * for(int i = 0; i < str.length(); i++){
27              for(int j = 0; j < strs.length; j++){
28                  String curr = strs[j];
29                  if(i + 1 > curr.length() || str.charAt(i) != curr.charAt(i)){
30                      return finalAns;
31                  }
32              }
33              finalAns += str.charAt(i);
34          }
35          return finalAns;
36          */
37
38          // Optimized: Used StringBuilder
39
40          String str = strs[0];
41          StringBuilder finalAns = new StringBuilder();
42          for(int i = 0; i < str.length(); i++){
43              for(int j = 0; j < strs.length; j++){
44                  String curr = strs[j];
45                  if(i + 1 > curr.length() || str.charAt(i) != curr.charAt(i)){
46                      return finalAns.toString();
47                  }
48              }
49              finalAns.append(str.charAt(i));
50          }
51          return finalAns.toString();
52      }
    Codeium: Refactor | Explain | Generate Javadoc | X
53     public static void main(String[] args) {
54         String[] strs = {"flower", "flow", "flight"};
55         System.out.println(new
56             Longest_Common_Prefix().longestCommonPrefix(strs));
57     }
58 }
59 }
60

```

> Task :Longest_Common_Prefix.main()

fl

Remove_Outmost_Parentheses


```

16 Example 1:
17
18 Input: s = "(()())()"
19 Output: "()()"
20 Explanation:
21 The input string is "(()())()", with primitive decomposition "(()())" + "()".
22 After removing outer parentheses of each part, this is "()()" + "()" = "()()()".
23
24 Example 2:
25
26 Input: s = "(()())()()()()"
27 Output: "()()()()()"
28 Explanation:
29 The input string is "(()())()()()()", with primitive decomposition "(()())" + "()" + "(()())".
30 After removing outer parentheses of each part, this is "()()" + "()" + "()" = "()()()()()".
31
32 Example 3:
33
34 Input: s = "()"
35 Output: ""
36 Explanation:
37 The input string is "()", with primitive decomposition "()" + "".
38 After removing outer parentheses of each part, this is "" + "" = "".
39 */
40

```

```

42 public class Remove_Outermost_Parentheses {
43     Codeium: Refactor | Explain | Generate Javadoc | X
44     public static String removeOuterParentheses(String str) {
45         /*
46          Time complexity:  $O(N^2)$  & Space complexity:  $O(1)$ .
47           $N^2$  due to every time creating a new string object
48
49          String ans = "";
50          int j = 0;
51          for(int i = 0; i < str.length() - 1; i++){
52              if(str.charAt(i) == ')'){
53                  j--;
54              }
55              if(j != 0){
56                  ans += str.charAt(i);
57              }
58              if(str.charAt(i) == '('){
59                  j++;
60              }
61          }
62          return ans;
63          */
64
65          // Time complexity:  $O(N)$  & Space complexity:  $O(1)$ .
66
67          StringBuilder ans = new StringBuilder();
68          int j = 0;
69          for(int i = 0; i < str.length() - 1; i++){
70              if(str.charAt(i) == ')'){
71                  j--;
72              }
73              if(j != 0){
74                  ans.append(str.charAt(i));
75              }
76              if(str.charAt(i) == '('){
77                  j++;
78              }
79          }
80          return ans.toString();
81     }
82     Codeium: Refactor | Explain | Generate Javadoc | X
83     public static void main(String[] args) {
84         String str = "(()())(()())";
85         System.out.println(removeOuterParentheses(str));
86     }
87 }

```

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

Reverse Words in a String

```
1  /*
2  * Reverse Words in a String
3  * Problem Statement: Given a string s, reverse the words of the string.
4
5  * Examples:
6
7  * Example 1:
8  * Input: s="this is an amazing program"
9  * Output: "program amazing an is this"
10
11 * Example 2:
12 * Input: s="This is decent"
13 * Output: "decent is This"
14 */
```

Codeium: Refactor | Explain

```
17 public class Reverse_Words_in_a_String {
18     Codeium: Refactor | Explain | Generate Javadoc | X
19     public static String reverseWords(String str) {
20         String ans = "";
21         int i = str.length() - 1;
22         while(i >= 0){
23             while(i >= 0 && str.charAt(i) == ' '){
24                 i--;
25             }
26             int j = i;
27             if(i < 0){
28                 break;
29             }
30             while(i >= 0 && str.charAt(i) != ' '){
31                 i--;
32             }
33             if(ans.isEmpty()){
34                 ans = ans.concat(str.substring(i+1, j+1));
35             }
36             else{
37                 ans = ans.concat(" " + str.substring(i+1, j+1));
38             }
39         }
40         return ans;
41     }
42     Codeium: Refactor | Explain | Generate Javadoc | X
43     public static void main(String[] args) {
44         String str = "the sky is blue";
45         System.out.println(reverseWords(str));
46     }
47 }
```

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

Count_number_of_substrings

```
1  /*
2  * Given a string of lowercase alphabets, count all possible substrings (not necessarily distinct)
3  * that have exactly k distinct characters.
4
5
6  Example 1:
7
8  Input:
9  S = "aba", K = 2
10 Output:
11 3
12 Explanation:
13 The substrings are:
14 "ab", "ba" and "aba".
15
16 Example 2:
17
18 Input:
19 S = "abaaca", K = 1
20 Output:
21 7
22 Explanation:
23 The substrings are:
24 "a", "b", "a", "aa", "a", "c", "a".
25 */
26
```

```
30 public class Count_number_of_substrings {
31     // Time complexity: O(N^2) & Space complexity: O(1).
32     public static int substrCount (String str, int k) {
33
34         int result = 0 ;
35         int n = str.length();
36         int cnt[] = new int[26];
37         for(int i = 0; i < n; i++){
38             int dist_count = 0 ;
39             Arrays.fill(cnt, 0);
40             for (int j = i; j < n ; j++ ){
41                 if(cnt[str.charAt(j) - 'a'] == 0){
42                     dist_count++;
43                 }
44                 cnt[str.charAt(j) - 'a']++;
45                 if(dist_count == k){
46                     result++;
47                 }
48             }
49         }
50         return result;
51     }
52
53     public static void main(String[] args) {
54         String s = "abaaca";
55         int k = 1;
56         System.out.println(substrCount(s, k));
57     }
58 }
```

> Task :Count_number_of_substrings.main()

7

Implement_Atoi

```
1  /*
2  Q. String to Integer (atoi) (leetcode 8).
3
4  Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).
5
6  The algorithm for myAtoi(string s) is as follows:
7
8  Read in and ignore any leading whitespace.
9  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
10 if the final result is negative or
11 positive respectively. Assume the result is positive if neither is present.
12 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.
13 Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the
14 sign as necessary (from step 2).
15 If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in
16 the range. Specifically,
17   integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
18 Return the integer as the final result.
19 Note:
20
21 Only the space character ' ' is considered a whitespace character.
22 Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.
23
24
25 Example 1:
26
27 Input: s = "42"
28 Output: 42
29 Explanation: The underlined characters are what is read in, the caret is the current reader position.
30 Step 1: "42" (no characters read because there is no leading whitespace)
31   |   |
32   |   ^
33   |   |
34   |   |
35   |   |
36   |   |
37   |   |
38   |   |
39   |   |
40   |   |
41   |   |
42   |   |
43   |   |
44   |   |
45   |   |
46   |   |
47   |   |
48   |   |
49   |   |
50   |   |
51   |   |
52   |   |
53   |   |
54   |   |
55   |   |
56   |   |
57   |   |
58   |   |
59   |   |
60   |   |
61   |   |
62   |   |
63   |   |
64   |   |
65   |   |
66   |   |
67   |   |
68   |   |
69   |   |
70   |   |
71   |   |
72   |   |
73   |   |
74   |   |
75   |   |
76   |   |
77   |   |
78   |   |
79   |   |
80   |   |
81   |   |
82   |   |
83   |   |
84   |   |
85   |   |
86   |   |
87   |   |
88   |   |
89   |   |
90   |   |
91   |   |
92   |   |
93   |   |
94   |   |
95   |   |
96   |   |
97   |   |
98   |   |
99   |   |
100  |   |
101  |   |
102  |   |
103  |   |
104  |   |
105  |   |
106  |   |
107  |   |
108  |   |
109  |   |
110  |   |
111  |   |
112  |   |
113  |   |
114  |   |
115  |   |
116  |   |
117  |   |
118  |   |
119  |   |
120  |   |
121  |   |
122  |   |
123  |   |
124  |   |
125  |   |
126  |   |
127  |   |
128  |   |
129  |   |
130  |   |
131  |   |
132  |   |
133  |   |
134  |   |
135  |   |
136  |   |
137  |   |
138  |   |
139  |   |
140  |   |
141  |   |
142  |   |
143  |   |
144  |   |
145  |   |
146  |   |
147  |   |
148  |   |
149  |   |
150  |   |
151  |   |
152  |   |
153  |   |
154  |   |
155  |   |
156  |   |
157  |   |
158  |   |
159  |   |
160  |   |
161  |   |
162  |   |
163  |   |
164  |   |
165  |   |
166  |   |
167  |   |
168  |   |
169  |   |
170  |   |
171  |   |
172  |   |
173  |   |
174  |   |
175  |   |
176  |   |
177  |   |
178  |   |
179  |   |
180  |   |
181  |   |
182  |   |
183  |   |
184  |   |
185  |   |
186  |   |
187  |   |
188  |   |
189  |   |
190  |   |
191  |   |
192  |   |
193  |   |
194  |   |
195  |   |
196  |   |
197  |   |
198  |   |
199  |   |
200  |   |
201  |   |
202  |   |
203  |   |
204  |   |
205  |   |
206  |   |
207  |   |
208  |   |
209  |   |
210  |   |
211  |   |
212  |   |
213  |   |
214  |   |
215  |   |
216  |   |
217  |   |
218  |   |
219  |   |
220  |   |
221  |   |
222  |   |
223  |   |
224  |   |
225  |   |
226  |   |
227  |   |
228  |   |
229  |   |
230  |   |
231  |   |
232  |   |
233  |   |
234  |   |
235  |   |
236  |   |
237  |   |
238  |   |
239  |   |
240  |   |
241  |   |
242  |   |
243  |   |
244  |   |
245  |   |
246  |   |
247  |   |
248  |   |
249  |   |
250  |   |
251  |   |
252  |   |
253  |   |
254  |   |
255  |   |
256  |   |
257  |   |
258  |   |
259  |   |
260  |   |
261  |   |
262  |   |
263  |   |
264  |   |
265  |   |
266  |   |
267  |   |
268  |   |
269  |   |
270  |   |
271  |   |
272  |   |
273  |   |
274  |   |
275  |   |
276  |   |
277  |   |
278  |   |
279  |   |
280  |   |
281  |   |
282  |   |
283  |   |
284  |   |
285  |   |
286  |   |
287  |   |
288  |   |
289  |   |
290  |   |
291  |   |
292  |   |
293  |   |
294  |   |
295  |   |
296  |   |
297  |   |
298  |   |
299  |   |
300  |   |
301  |   |
302  |   |
303  |   |
304  |   |
305  |   |
306  |   |
307  |   |
308  |   |
309  |   |
310  |   |
311  |   |
312  |   |
313  |   |
314  |   |
315  |   |
316  |   |
317  |   |
318  |   |
319  |   |
320  |   |
321  |   |
322  |   |
323  |   |
324  |   |
325  |   |
326  |   |
327  |   |
328  |   |
329  |   |
330  |   |
331  |   |
332  |   |
333  |   |
334  |   |
335  |   |
336  |   |
337  |   |
338  |   |
339  |   |
340  |   |
341  |   |
342  |   |
343  |   |
344  |   |
345  |   |
346  |   |
347  |   |
348  |   |
349  |   |
350  |   |
351  |   |
352  |   |
353  |   |
354  |   |
355  |   |
356  |   |
357  |   |
358  |   |
359  |   |
360  |   |
361  |   |
362  |   |
363  |   |
364  |   |
365  |   |
366  |   |
367  |   |
368  |   |
369  |   |
370  |   |
371  |   |
372  |   |
373  |   |
374  |   |
375  |   |
376  |   |
377  |   |
378  |   |
379  |   |
380  |   |
381  |   |
382  |   |
383  |   |
384  |   |
385  |   |
386  |   |
387  |   |
388  |   |
389  |   |
390  |   |
391  |   |
392  |   |
393  |   |
394  |   |
395  |   |
396  |   |
397  |   |
398  |   |
399  |   |
400  |   |
401  |   |
402  |   |
403  |   |
404  |   |
405  |   |
406  |   |
407  |   |
408  |   |
409  |   |
410  |   |
411  |   |
412  |   |
413  |   |
414  |   |
415  |   |
416  |   |
417  |   |
418  |   |
419  |   |
420  |   |
421  |   |
422  |   |
423  |   |
424  |   |
425  |   |
426  |   |
427  |   |
428  |   |
429  |   |
430  |   |
431  |   |
432  |   |
433  |   |
434  |   |
435  |   |
436  |   |
437  |   |
438  |   |
439  |   |
440  |   |
441  |   |
442  |   |
443  |   |
444  |   |
445  |   |
446  |   |
447  |   |
448  |   |
449  |   |
450  |   |
451  |   |
452  |   |
453  |   |
454  |   |
455  |   |
456  |   |
457  |   |
458  |   |
459  |   |
460  |   |
461  |   |
462  |   |
463  |   |
464  |   |
465  |   |
466  |   |
467  |   |
468  |   |
469  |   |
470  |   |
471  |   |
472  |   |
473  |   |
474  |   |
475  |   |
476  |   |
477  |   |
478  |   |
479  |   |
480  |   |
481  |   |
482  |   |
483  |   |
484  |   |
485  |   |
486  |   |
487  |   |
488  |   |
489  |   |
490  |   |
491  |   |
492  |   |
493  |   |
494  |   |
495  |   |
496  |   |
497  |   |
498  |   |
499  |   |
500  |   |
501  |   |
502  |   |
503  |   |
504  |   |
505  |   |
506  |   |
507  |   |
508  |   |
509  |   |
510  |   |
511  |   |
512  |   |
513  |   |
514  |   |
515  |   |
516  |   |
517  |   |
518  |   |
519  |   |
520  |   |
521  |   |
522  |   |
523  |   |
524  |   |
525  |   |
526  |   |
527  |   |
528  |   |
529  |   |
530  |   |
531  |   |
532  |   |
533  |   |
534  |   |
535  |   |
536  |   |
537  |   |
538  |   |
539  |   |
540  |   |
541  |   |
542  |   |
543  |   |
544  |   |
545  |   |
546  |   |
547  |   |
548  |   |
549  |   |
550  |   |
551  |   |
552  |   |
553  |   |
554  |   |
555  |   |
556  |   |
557  |   |
558  |   |
559  |   |
560  |   |
561  |   |
562  |   |
563  |   |
564  |   |
565  |   |
566  |   |
567  |   |
568  |   |
569  |   |
570  |   |
571  |   |
572  |   |
573  |   |
574  |   |
575  |   |
576  |   |
577  |   |
578  |   |
579  |   |
580  |   |
581  |   |
582  |   |
583  |   |
584  |   |
585  |   |
586  |   |
587  |   |
588  |   |
589  |   |
590  |   |
591  |   |
592  |   |
593  |   |
594  |   |
595  |   |
596  |   |
597  |   |
598  |   |
599  |   |
600  |   |
601  |   |
602  |   |
603  |   |
604  |   |
605  |   |
606  |   |
607  |   |
608  |   |
609  |   |
610  |   |
611  |   |
612  |   |
613  |   |
614  |   |
615  |   |
616  |   |
617  |   |
618  |   |
619  |   |
620  |   |
621  |   |
622  |   |
623  |   |
624  |   |
625  |   |
626  |   |
627  |   |
628  |   |
629  |   |
630  |   |
631  |   |
632  |   |
633  |   |
634  |   |
635  |   |
636  |   |
637  |   |
638  |   |
639  |   |
640  |   |
641  |   |
642  |   |
643  |   |
644  |   |
645  |   |
646  |   |
647  |   |
648  |   |
649  |   |
650  |   |
651  |   |
652  |   |
653  |   |
654  |   |
655  |   |
656  |   |
657  |   |
658  |   |
659  |   |
660  |   |
661  |   |
662  |   |
663  |   |
664  |   |
665  |   |
666  |   |
667  |   |
668  |   |
669  |   |
670  |   |
671  |   |
672  |   |
673  |   |
674  |   |
675  |   |
676  |   |
677  |   |
678  |   |
679  |   |
680  |   |
681  |   |
682  |   |
683  |   |
684  |   |
685  |   |
686  |   |
687  |   |
688  |   |
689  |   |
690  |   |
691  |   |
692  |   |
693  |   |
694  |   |
695  |   |
696  |   |
697  |   |
698  |   |
699  |   |
700  |   |
701  |   |
702  |   |
703  |   |
704  |   |
705  |   |
706  |   |
707  |   |
708  |   |
709  |   |
710  |   |
711  |   |
712  |   |
713  |   |
714  |   |
715  |   |
716  |   |
717  |   |
718  |   |
719  |   |
720  |   |
721  |   |
722  |   |
723  |   |
724  |   |
725  |   |
726  |   |
727  |   |
728  |   |
729  |   |
730  |   |
731  |   |
732  |   |
733  |   |
734  |   |
735  |   |
736  |   |
737  |   |
738  |   |
739  |   |
740  |   |
741  |   |
742  |   |
743  |   |
744  |   |
745  |   |
746  |   |
747  |   |
748  |   |
749  |   |
750  |   |
751  |   |
752  |   |
753  |   |
754  |   |
755  |   |
756  |   |
757  |   |
758  |   |
759  |   |
760  |   |
761  |   |
762  |   |
763  |   |
764  |   |
765  |   |
766  |   |
767  |   |
768  |   |
769  |   |
770  |   |
771  |   |
772  |   |
773  |   |
774  |   |
775  |   |
776  |   |
777  |   |
778  |   |
779  |   |
780  |   |
781  |   |
782  |   |
783  |   |
784  |   |
785  |   |
786  |   |
787  |   |
788  |   |
789  |   |
790  |   |
791  |   |
792  |   |
793  |   |
794  |   |
795  |   |
796  |   |
797  |   |
798  |   |
799  |   |
800  |   |
801  |   |
802  |   |
803  |   |
804  |   |
805  |   |
806  |   |
807  |   |
808  |   |
809  |   |
810  |   |
811  |   |
812  |   |
813  |   |
814  |   |
815  |   |
816  |   |
817  |   |
818  |   |
819  |   |
820  |   |
821  |   |
822  |   |
823  |   |
824  |   |
825  |   |
826  |   |
827  |   |
828  |   |
829  |   |
830  |   |
831  |   |
832  |   |
833  |   |
834  |   |
835  |   |
836  |   |
837  |   |
838  |   |
839  |   |
840  |   |
841  |   |
842  |   |
843  |   |
844  |   |
845  |   |
846  |   |
847  |   |
848  |   |
849  |   |
850  |   |
851  |   |
852  |   |
853  |   |
854  |   |
855  |   |
856  |   |
857  |   |
858  |   |
859  |   |
860  |   |
861  |   |
862  |   |
863  |   |
864  |   |
865  |   |
866  |   |
867  |   |
868  |   |
869  |   |
870  |   |
871  |   |
872  |   |
873  |   |
874  |   |
875  |   |
876  |   |
877  |   |
878  |   |
879  |   |
880  |   |
881  |   |
882  |   |
883  |   |
884  |   |
885  |   |
886  |   |
887  |   |
888  |   |
889  |   |
890  |   |
891  |   |
892  |   |
893  |   |
894  |   |
895  |   |
896  |   |
897  |   |
898  |   |
899  |   |
900  |   |
901  |   |
902  |   |
903  |   |
904  |   |
905  |   |
906  |   |
907  |   |
908  |   |
909  |   |
910  |   |
911  |   |
912  |   |
913  |   |
914  |   |
915  |   |
916  |   |
917  |   |
918  |   |
919  |   |
920  |   |
921  |   |
922  |   |
923  |   |
924  |   |
925  |   |
926  |   |
927  |   |
928  |   |
929  |   |
930  |   |
931  |   |
932  |   |
933  |   |
934  |   |
935  |   |
936  |   |
937  |   |
938  |   |
939  |   |
940  |   |
941  |   |
942  |   |
943  |   |
944  |   |
945  |   |
946  |   |
947  |   |
948  |   |
949  |   |
950  |   |
951  |   |
952  |   |
953  |   |
954  |   |
955  |   |
956  |   |
957  |   |
958  |   |
959  |   |
960  |   |
961  |   |
962  |   |
963  |   |
964  |   |
965  |   |
966  |   |
967  |   |
968  |   |
969  |   |
970  |   |
971  |   |
972  |   |
973  |   |
974  |   |
975  |   |
976  |   |
977  |   |
978  |   |
979  |   |
980  |   |
981  |   |
982  |   |
983  |   |
984  |   |
985  |   |
986  |   |
987  |   |
988  |   |
989  |   |
990  |   |
991  |   |
992  |   |
993  |   |
994  |   |
995  |   |
996  |   |
997  |   |
998  |   |
999  |   |
1000 |   |
1001 |   |
1002 |   |
1003 |   |
1004 |   |
1005 |   |
1006 |   |
1007 |   |
1008 |   |
1009 |   |
1010 |   |
1011 |   |
1012 |   |
1013 |   |
1014 |   |
1015 |   |
1016 |   |
1017 |   |
1018 |   |
1019 |   |
1020 |   |
1021 |   |
1022 |   |
1023 |   |
1024 |   |
1025 |   |
1026 |   |
1027 |   |
1028 |   |
1029 |   |
1030 |   |
1031 |   |
1032 |   |
1033 |   |
1034 |   |
1035 |   |
1036 |   |
1037 |   |
1038 |   |
1039 |   |
1040 |   |
1041 |   |
1042 |   |
1043 |   |
1044 |   |
1045 |   |
1046 |   |
1047 |   |
1048 |   |
1049 |   |
1050 |   |
1051 |   |
1052 |   |
1053 |   |
1054 |   |
1055 |   |
1056 |   |
1057 |   |
1058 |   |
1059 |   |
1060 |   |
1061 |   |
1062 |   |
1063 |   |
1064 |   |
1065 |   |
1066 |   |
1067 |   |
1068 |   |
1069 |   |
1070 |   |
1071 |   |
1072 |   |
1073 |   |
1074 |   |
1075 |   |
1076 |   |
1077 |   |
1078 |   |
1079 |   |
1080 |   |
1081 |   |
1082 |   |
1083 |   |
1084 |   |
1085 |   |
1086 |   |
1087 |   |
1088 |   |
1089 |   |
1090 |   |
1091 |   |
1092 |   |
1093 |   |
1094 |   |
1095 |   |
1096 |   |
1097 |   |
1098 |   |
1099 |   |
1100 |   |
1101 |   |
1102 |   |
1103 |   |
1104 |   |
1105 |   |
1106 |   |
1107 |   |
1108 |   |
1109 |   |
1110 |   |
1111 |   |
1112 |   |
1113 |   |
1114 |   |
1115 |   |
1116 |   |
1117 |   |
1118 |   |
1119 |   |
1120 |   |
1121 |   |
1122 |   |
1123 |   |
1124 |   |
1125 |   |
1126 |   |
1127 |   |
1128 |   |
1129 |   |
1130 |   |
1131 |   |
1132 |   |
1133 |   |
1134 |   |
1135 |   |
1136 |   |
1137 |   |
1138 |   |
1139 |   |
1140 |   |
1141 |   |
1142 |   |
1143 |   |
1144 |   |
1145 |   |
1146 |   |
1147 |   |
1148 |   |
1149 |   |
1150 |   |
1151 |   |
1152 |   |
1153 |   |
1154 |   |
1155 |   |
1156 |   |
1157 |   |
1158 |   |
1159 |   |
1160 |   |
1161 |   |
1162 |   |
1163 |   |
1164 |   |
1165 |   |
1166 |   |
1167 |   |
1168 |   |
1169 |   |
1170 |   |
1171 |   |
1172 |   |
1173 |   |
1174 |   |
1175 |   |
1176 |   |
1177 |   |
1178 |   |
1179 |   |
1180 |   |
1181 |   |
1182 |   |
1183 |   |
1184 |   |
1185 |   |
1186 |   |
1187 |   |
1188 |   |
1189 |   |
1190 |   |
1191 |   |
1192 |   |
1193 |   |
1194 |   |
1195 |   |
1196 |   |
1197 |   |
1198 |   |
1199 |   |
1200 |   |
1201 |   |
1202 |   |
1203 |   |
1204 |   |
1205 |   |
1206 |   |
1207 |   |
1208 |   |
1209 |   |
1210 |   |
1211 |   |
1212 |   |
1213 |   |
1214 |   |
1215 |   |
1216 |   |
1217 |   |
1218 |   |
1219 |   |
1220 |   |
1221 |   |
1222 |   |
1223 |   |
1224 |   |
1225 |   |
1226 |   |
1227 |   |
1228 |   |
1229 |   |
1230 |   |
1231 |   |
1232 |   |
1233 |   |
1234 |   |
1235 |   |
1236 |   |
1237 |   |
1238 |   |
1239 |   |
1240 |   |
1241 |   |
1242 |   |
1243 |   |
1244 |   |
1245 |   |
1246 |   |
1247 |   |
1248 |   |
1249 |   |
1250 |   |
1251 |   |
1252 |   |
1253 |   |
1254 |   |
1255 |   |
1256 |   |
1257 |   |
1258 |   |
1259 |   |
1260 |   |
1261 |   |
1262 |   |
1263 |   |
1264 |   |
1265 |   |
1266 |   |
1267 |   |
1268 |   |
1269 |   |
1270 |   |
1271 |   |
1272 |   |
1273 |   |
1274 |   |
1275 |   |
1276 |   |
1277 |   |
1278 |   |
1279 |   |
1280 |   |
1281 |   |
1282 |   |
1283 |   |
1284 |   |
1285 |   |
1286 |   |
1287 |   |
1288 |   |
1289 |   |
1290 |   |
1291 |   |
1292 |   |
1293 |   |
1294 |   |
1295 |   |
1296 |   |
1297 |   |
1298 |   |
1299 |   |
1300 |   |
1301 |   |
1302 |   |
1303 |   |
1304 |   |
1305 |   |
1306 |   |
1307 |   |
1308 |   |
1309 |   |
1310 |   |
1311 |   |
1312 |   |
1313 |   |
1314 |   |
1315 |   |
1316 |   |
1317 |   |
1318 |   |
1319 |   |
1320 |   |
1321 |   |
1322 |   |
1323 |   |
1324 |   |
1325 |   |
1326 |   |
1327 |   |
1328 |   |
1329 |   |
1330 |   |
1331 |   |
1332 |   |
1333 |   |
1334 |   |
1335 |   |
1336 |   |
1337 |   |
1338 |   |
1339 |   |
1340 |   |
1341 |   |
1342 |   |
1343 |   |
1344 |   |
1345 |   |
1346 |   |
1347 |   |
1348 |   |
1349 |   |
1350 |   |
1351 |   |
1352 |   |
1353 |   |
1354 |   |
1355 |   |
1356 |   |
1357 |   |
1358 |   |
1359 |   |
1360 |   |
1361 |   |
1362 |   |
1363 |   |
1364 |   |
1365 |   |
1366 |   |
1367 |   |
1368 |   |
1369 |   |
1370 |   |
1371 |   |
1372 |   |
1373 |   |
1374 |   |
1375 |   |
1376 |   |
1377 |   |
1378 |   |
1379 |   |
1380 |   |
1381 |   |
1382 |   |
1383 |   |
1384 |   |
1385 |   |
1386 |   |
1387 |   |
1388 |   |
1389 |   |
1390 |   |
1391 |   |
1392 |   |
1393 |   |
1394 |   |
1395 |   |
1396 |   |
1397 |   |
1398 |   |
1399 |   |
1400 |   |
1401 |   |
1402 |   |
1403 |   |
1404 |   |
1405 |   |
1406 |   |
1407 |   |
1408 |   |
1409 |   |
1410 |   |
1411 |   |
1412 |   |
1413 |   |
1414 |   |
1415 |   |
1416 |   |
1417 |   |
1418 |   |
1419 |   |
1420 |   |
1421 |   |
1422 |   |
1423 |   |
1424 |   |
1425 |   |
1426 |   |
1427 |   |
1428 |   |
1429 |   |
1430 |   |
1431 |   |
1432 |   |
1433 |   |
1434 |   |
1435 |   |
1436 |   |
1437 |   |
1438 |   |
1439 |   |
1440 |   |
1441 |   |
1442 |   |
1443 |   |
1444 |   |
1445 |   |
1446 |   |
1447 |   |
1448 |   |
1449 |   |
1450 |   |
1451 |   |
1452 |   |
1453 |   |
1454 |   |
1455 |   |
1456 |   |
1457 |   |
1458 |   |
1459 |   |
1460 |   |
1461 |   |
1462 |   |
1463 |   |
1464 |   |
1465 |   |
1466 |   |
1467 |   |
1468 |   |
1469 |   |
1470 |   |
1471 |   |
1472 |   |
1473 |   |
1474 |   |
1475 |   |
1476 |   |
1477 |   |
1478 |   |
1479 |   |
1480 |   |
1481 |   |
1482 |   |
1483 |   |
1484 |   |
1485 |   |
1486 |   |
1487 |   |
1488 |   |
1489 |   |
1490 |   |
1491 |   |
1492 |   |
1493 |   |
1494 |   |
1495 |   |
1496 |   |
1497 |   |
1498 |   |
1499 |   |
1500 |   |
1501 |   |
1502 |   |
1503 |   |
1504 |   |
1505 |   |
1506 |   |
1507 |   |
1508 |   |
1509 |   |
1510 |   |
1511 |   |
1512 |   |
1513 |   |
1514 |   |
1515 |   |
1516 |   |
1517 |   |
1518 |   |
1519
```

```

Codeium: Refactor | Explain
67  public class Implement_Atoi {
    Codeium: Refactor | Explain | Generate Javadoc | X
68      public static int myAtoi(String s) {
69          if(s.length() == 0){
70              return 0;
71          }
72          for(int i = 0; i < s.length(); i++){
73              if(s.charAt(i) != ' '){
74                  s = s.substring(i, s.length());
75                  break;
76              }
77          }
78
79          int max = Integer.MAX_VALUE;
80          int min = Integer.MIN_VALUE;
81
82          long ans = 0;
83          int sign = 1;
84
85          if(s.charAt(0) == '-'){
86              sign = -1;
87          }
88          int i = (s.charAt(0) == '+' || s.charAt(0) == '-') ? 1 : 0;
89          while(i < s.length()){
90              if(s.charAt(i) == '-' || !Character.isDigit(s.charAt(i))){
91                  return sign * (int)(ans);
92              }
93              ans = ans * 10 + (s.charAt(i) - '0');
94              if(sign == -1 && -1 * ans < min){
95                  return min;
96              }
97              if(sign == 1 && ans > max){
98                  return max;
99              }
100             i++;
101
102         }
103         return sign * (int)(ans);
104     }
    Codeium: Refactor | Explain | Generate Javadoc | X
105     public static void main(String[] args) {
106         String s = "-42";
107         System.out.println(myAtoi(s));
108     }
109 }
110

```

> Task :Implement_Atoi.main()

-42

Longest_Palindromic_Substring

LeetCode / medium / Longest Palindromic Substring

```
1  /*
2   * Given a string s, return the longest palindromic substring in s.
3
4   A string is called a palindrome string if the reverse of that string is the same as the original string.
5
6   Example 1:
7
8   Input: s = "babad"
9   Output: "bab"
10  Explanation: "aba" is also a valid answer.
11  Example 2:
12
13  Input: s = "cbbd"
14  Output: "bb"
15 */
16
17
```

```

18 public class Longest_Palindromic_Substring {
    Codeium: Refactor | Explain | Generate Javadoc | X
19     public static boolean palindrome(String str){
20         // This function will be used in first BruteForce Mehtod
21         int i = 0;
22         int j = str.length() - 1;
23         while(i < j){
24             if(str.charAt(i) != str.charAt(j)){
25                 return false;
26             }
27             i++; j--;
28         }
29         return true;
30     }
    Codeium: Refactor | Explain | Generate Javadoc | X
31     public static String longestPalindrome(String s) {
32         /*
33          * BruteForce Approach: Time complexity:  $O(N^3)$  & Space complexity:  $O(1)$ .
34          * int max = Integer.MIN_VALUE;
35          * StringBuilder ans = new StringBuilder("temp");
36          * for(int i = 0; i < s.length(); i++){
37              for(int j = i; j < s.length(); j++){
38                  boolean possible = palindrome(s.substring(i, j + 1));
39                  if(possible){
40                      if(max < ((j - i) + 1)){
41                          max = ((j - i) + 1);
42                          ans.replace(0, ans.length(), s.substring(i, j + 1));
43                      }
44                  }
45              }
46          }
47          return ans.toString();
48          */
49         int start = 0;
50         int maxLen = 1;
51         int l, r;
52         for(int i = 0; i < s.length(); i++){
53             // even
54             l=i-1;
55             r=i ;
56             while(l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)){
57                 if(r-l+1>maxLen){
58                     maxLen=r-l+1;
59                     start=l;
60                 }
61                 l-=1;
62                 r+=1;
63             }
64         }
        ..

```



```

64         // odd
65         l=i-1;
66         r=i+1;
67         while(l>=0 && r<s.length() && s.charAt(l)==s.charAt(r)){
68             if( r-l+1>maxLen){
69                 maxLen=r-l+1;
70                 start=l;
71             }
72             l-=1;
73             r+=1;
74         }
75     }
76     return s.substring(start, start + maxLen);
77 }
78 Codeium: Refactor | Explain | Generate Javadoc | ✕
79 public static void main(String[] args) {
80     String s = "babad";
81     System.out.println(longestPalindrome(s));
82 }
83

```

> Task :Longest_Palindromic_Substring.main()

bab

Maximum_Nesting_Depth_of_the_Parentheses

```

1  ✓ /*
2      * Q. Write a program to Count Maximum Nesting Depth of the Parentheses.
3      *
4  ✓  * Example 1:
5
6      Input: s = "(1+(2*3)+((8)/4))+1"
7      Output: 3
8      Explanation: Digit 8 is inside of 3 nested parentheses in the string.
9
10     Example 2:
11
12     Input: s = "(1)+((2))+(((3)))"
13     Output: 3
14 */
15
16 // import java.util.ArrayList;
17
Codeium: Refactor | Explain

```

```

18 public class Maximum_Nesting_Depth_of_the_Parentheses {
19     public static int maxDepth(String str) {
20         /*
21          * BruteForce Approach: Time complexity: O(N) & Space complexity: O(N).
22          * ArrayList<Integer> temp = new ArrayList<>();
23          int j = 0;
24          int max = Integer.MIN_VALUE;
25          for(int i = 0; i < str.length(); i++){
26              if(str.charAt(i) == '('){
27                  j++;
28                  max = Math.max(max, j);
29              }
30              if(str.charAt(i) == ')'){
31                  j--;
32                  if(j == 0){
33                      temp.add(max);
34                      max = Integer.MIN_VALUE;
35                  }
36              }
37          }
38          int ans = 0;
39          for(int i = 0; i < temp.size(); i++){
40              ans = Math.max(ans, temp.get(i));
41          }
42          return ans;
43          */
44
45          // Optimized Approach: Time complexity: O(N) & Space complexity: O(1).
46
47          int ans = 0, j = 0;
48          for(int i = 0; i < str.length(); i++){
49              if(str.charAt(i) == '('){
50                  j++;
51              }
52              if(str.charAt(i) == ')'){
53                  j--;
54              }
55              ans = Math.max(ans, j);
56          }
57          return ans;
58      }
59      public static void main(String[] args) {
60          String str = "(1+(2*3)+((8)/4))+1";
61          System.out.println(maxDepth(str));
62      }
63  }
64

```

> Task :Maximum_Nesting_Depth_of_the_Parentheses.main()

3

Roman_Number_to_Integer_and_vice-versa

```

1  /*
2   * Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.
3
4   Symbol      Value
5   I           1
6   V           5
7   X           10
8   L           50
9   C           100
10  D           500
11  M           1000
12  For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply
13  X + II. The number 27 is written as XXVII, which is XX + V + II.
14
15  Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII.
16  Instead, the number four is written as IV. Because the one is before the five we subtract it making four.
17  The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:
18
19  I can be placed before V (5) and X (10) to make 4 and 9.
20  X can be placed before L (50) and C (100) to make 40 and 90.
21  C can be placed before D (500) and M (1000) to make 400 and 900.
22  Given a roman numeral, convert it to an integer.
23
24
25
26  Example 1:
27
28  Input: s = "III"
29  Output: 3
30  Explanation: III = 3.
31  Example 2:
32
33  Input: s = "LVIII"
34  Output: 58
35  Explanation: L = 50, V = 5, III = 3.
36  Example 3:
37
38  Input: s = "MCMXCIV"
39  Output: 1994
40  Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
41  */
42
43  Codeium: Refactor | Explain | Generate Javadoc | X
44  public class Roman_Number_to_Integer_and_vice versa {
45      public static int romanToInt(String str) {
46          HashMap<Character, Integer> map = new HashMap<>();
47          map.put('I', 1);
48          map.put('V', 5);
49          map.put('X', 10);
50          map.put('L', 50);
51          map.put('C', 100);
52          map.put('D', 500);
53          map.put('M', 1000);
54
55          /*
56           * BruteForce Approach: Time complexity: O(N) & Space complexity: O(1). *
57           * if(str.length() == 1){
58           |     return map.get(str.charAt(0));
59           | }
60           | int num = 0;
61           | for(int i = 0; i < str.length() - 1; i++){
62           |     if((str.charAt(i) == 'I' && (str.charAt(i + 1) == 'X' || str.charAt(i + 1) == 'V')) || (str.charAt(i) == 'X' &&
63           |     (str.charAt(i + 1) == 'L' || str.charAt(i + 1) == 'C')) || (str.charAt(i) == 'C' && (str.charAt(i + 1) == 'D'
64           |     || str.charAt(i + 1) == 'M'))){
65           |         num += map.get(str.charAt(i + 1)) - map.get(str.charAt(i));
66           |         i++;
67           |         if(i == str.length() - 1){
68           |             return num;
69           |         }
70           |     }
71           |     else{
72           |         num += map.get(str.charAt(i));
73           |     }
74           | }
75           | int i = str.length() - 1;
76           | if((str.charAt(i - 1) == 'I' && (str.charAt(i) == 'X' || str.charAt(i) == 'V')) || (str.charAt(i - 1) == 'X'
77           | && (str.charAt(i) == 'L' || str.charAt(i) == 'C')) || (str.charAt(i - 1) == 'C' &&
78           | (str.charAt(i) == 'D' || str.charAt(i) == 'M'))){
79           |     num -= map.get(str.charAt(i - 1));
80           |     num += map.get(str.charAt(i)) - map.get(str.charAt(i - 1));
81           | }
82           | else{
83           |     num += map.get(str.charAt(i));
84           | }
85           | return num;
86           */
87      }
88  }

```

```

91 // Same Approach in a little Optimized way. Time complexity: O(N) & Space complexity: O(1).
92
93 int num = map.get(str.charAt(str.length() - 1));
94 for(int i = str.length() - 2; i >= 0; i--){
95     if(map.get(str.charAt(i)) < map.get(str.charAt(i + 1))){
96         num -= map.get(str.charAt(i));
97     }
98     else{
99         num += map.get(str.charAt(i));
100     }
101 }
102 return num;
103
104 Codeium: Refactor | Explain | Generate Javadoc | X
105 public static void main(String[] args) {
106     String str = "MCMXCIV";
107     System.out.println(romanToInt(str));
108 }
109

```

> Task :Roman_Number_to_Integer_and_vice_versa.main()
1994

Sort_Characters_by_frequency

```

1  /*
2  * Given a string s, sort it in decreasing order based on the frequency of the characters.
3  * The frequency of a character is the number of times it appears in the string.
4
5  * Return the sorted string. If there are multiple answers, return any of them.
6
7  * Example 1:
8
9  * Input: s = "tree"
10 * Output: "eert"
11 * Explanation: 'e' appears twice while 'r' and 't' both appear once.
12 * So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
13
14 * Example 2:
15
16 * Input: s = "cccaaa"
17 * Output: "aaaccc"
18 * Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.
19 * Note that "cacaca" is incorrect, as the same characters must be together.
20
21 * Example 3:
22
23 * Input: s = "Aabb"
24 * Output: "bbAa"
25 * Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
26 * Note that 'A' and 'a' are treated as two different characters.
27 */
28

```

Codeium: Refactor | Explain

```

35 public class Sort_Characters_by_frequency {
    Codeium: Refactor | Explain | Generate Javadoc | X
36     public static String frequencySort(String s) {
37         HashMap<Character, Integer> map = new HashMap<>();
38         for(int i = 0; i < s.length(); i++){
39             if(map.containsKey(s.charAt(i))){
40                 map.put(s.charAt(i), map.get(s.charAt(i)) + 1);
41             }
42             else{
43                 map.put(s.charAt(i), 1);
44             }
45         }
46         List<Character>[] bucket = new List[s.length() + 1];
47         for(Character key : map.keySet()){
48             int frequency = map.get(key);
49             if(bucket[frequency] == null){
50                 bucket[frequency] = new ArrayList<>();
51             }
52             bucket[frequency].add(key);
53         }
54         StringBuilder sb = new StringBuilder();
55         for(int i = bucket.length - 1; i >= 0; i--){
56             if(bucket[i] != null){
57                 for(char c : bucket[i]){
58                     for(int j = 0; j < map.get(c); j++){
59                         sb.append(c);
60                     }
61                 }
62             }
63         }
64         return sb.toString();
65     }
    Codeium: Refactor | Explain | Generate Javadoc | X
66     public static void main(String[] args) {
67         String s = "tree";
68         System.out.println(frequencySort(s));
69     }
70 }
71

```

> Task :Sort_Characters_by_frequency.main()
eert

Sum_of_Beauty_of_All_Substrings

```

1  /*
2   the beauty of a string is the difference in frequencies between the most frequent and least frequent characters.
3
4   For example, the beauty of "abaacc" is 3 - 1 = 2.
5   Given a string s, return the sum of beauty of all of its substrings.
6
7   Example 1:
8
9   Input: s = "aabcb"
10  Output: 5
11  Explanation: The substrings with non-zero beauty are ["aab","aabc","aabcb","abcb","bcb"], each with beauty equal to 1.
12
13  Example 2:
14
15  Input: s = "aabcbaa"
16  Output: 17
17  */

```

```

24  public class Sum_of_Beauty_of_All_Substrings {
25      public static int findBeauty2(int[] cnt){
26          int max = -1;
27          int min = Integer.MAX_VALUE;
28          for(int i = 0; i < 26; i++){
29              max = Math.max(max, cnt[i]);
30              if(cnt[i] >= 1){
31                  min = Math.min(min, cnt[i]);
32              }
33          }
34          return max - min;
35      }
36      public static int findBeauty(String str){
37          if(str.length() == 0){
38              return 0;
39          }
40          HashMap<Character, Integer> map = new HashMap<>();
41          for(int i = 0; i < str.length(); i++){
42              if(map.containsKey(str.charAt(i))){
43                  map.put(str.charAt(i), map.get(str.charAt(i)) + 1);
44              }
45              else{
46                  map.put(str.charAt(i), 1);
47              }
48          }
49          int max = Integer.MIN_VALUE;
50          int min = Integer.MAX_VALUE;
51          for(Map.Entry<Character, Integer> entry: map.entrySet()){
52              max = Math.max(entry.getValue(), max);
53              min = Math.min(entry.getValue(), min);
54          }
55          return max - min;
56      }

```

```

57 public static int beautySum(String s) {
58     /*
59      * BruteForce Appraoch: Time complexity:  $O(N^3)$  & Space complexity:  $O(N)$ .
60      *
61      * int beauty = 0;
62      * for(int i = 0; i < s.length(); i++){
63      *     for(int j = i; j < s.length(); j++){
64      *         beauty += findBeauty(s.substring(i, j + 1));
65      *     }
66      * }
67      * return beauty;
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 = "aabcba";
83     System.out.println(beautySum(s));
84 }
85 }

```

> Task :Sum_of_Beauty_of_All_Substrings.main()

5