CSE 4094

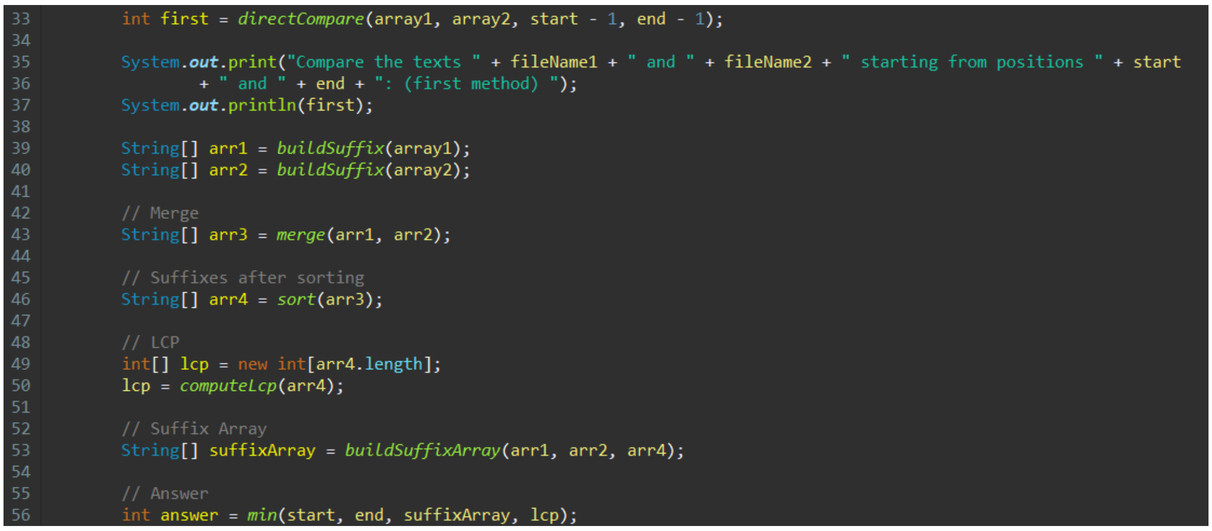
Special Topics in Computer Engineering

Advanced Data Structures

Project 1

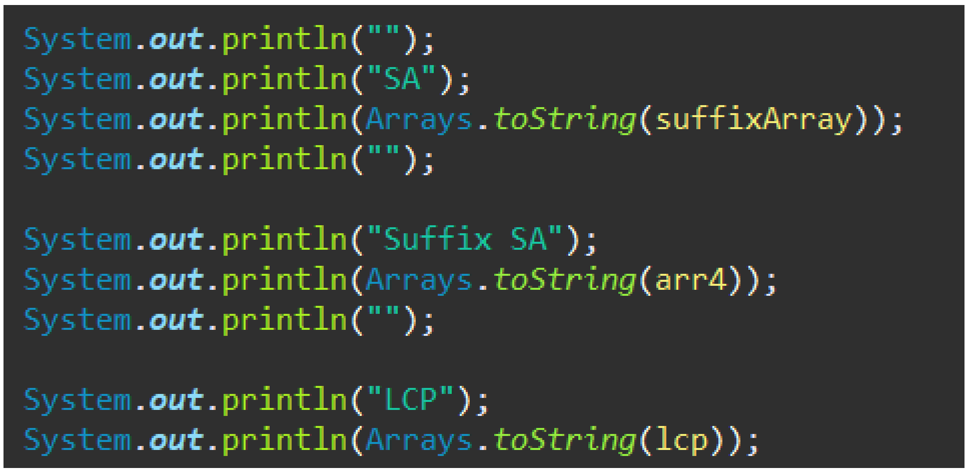
Ahmet Hamza Demir 150116025
Berk Yıldız 150117052

Main Function:



```java
        int first = directCompare(array1, array2, start - 1, end - 1);

        System.out.print("Compare the texts " + fileName1 + " and " + fileName2 + " starting from positions " + start
                + " and " + end + ": (first method) ");
        System.out.println(first);

        String[] arr1 = buildSuffix(array1);
        String[] arr2 = buildSuffix(array2);

        // Merge
        String[] arr3 = merge(arr1, arr2);

        // Suffixes after sorting
        String[] arr4 = sort(arr3);

        // LCP
        int[] lcp = new int[arr4.length];
        lcp = computeLcp(arr4);

        // Suffix Array
        String[] suffixArray = buildSuffixArray(arr1, arr2, arr4);

        // Answer
        int answer = min(start, end, suffixArray, lcp);
```

Here, we call our functions. (We call directCompare function by reducing i and j parameters by one because our index value starts from zero not one.)

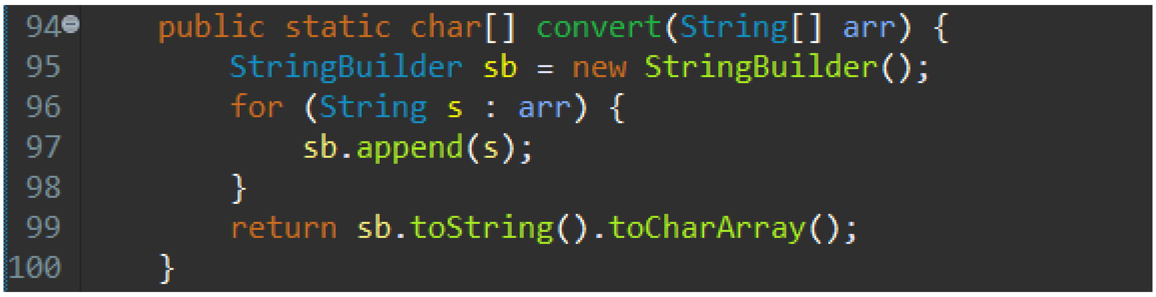


```java
System.out.println("");
System.out.println("SA");
System.out.println(Arrays.toString(suffixArray));
System.out.println("");

System.out.println("Suffix SA");
System.out.println(Arrays.toString(arr4));
System.out.println("");

System.out.println("LCP");
System.out.println(Arrays.toString(lcp));
```

This part is for console output.

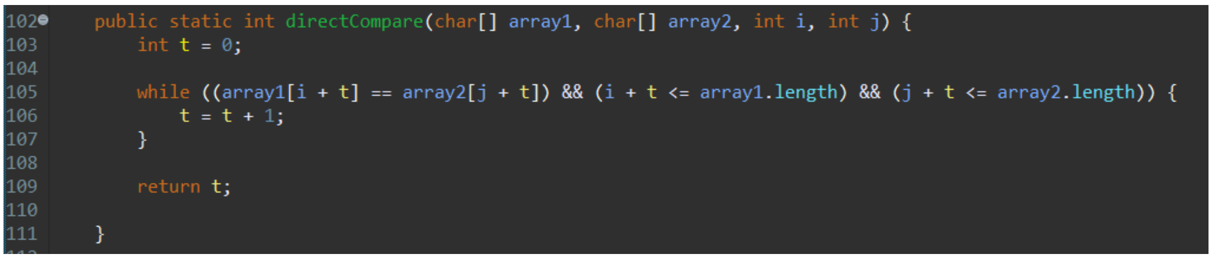
Functions:

<u>Convert function:</u> It converts string to char array.



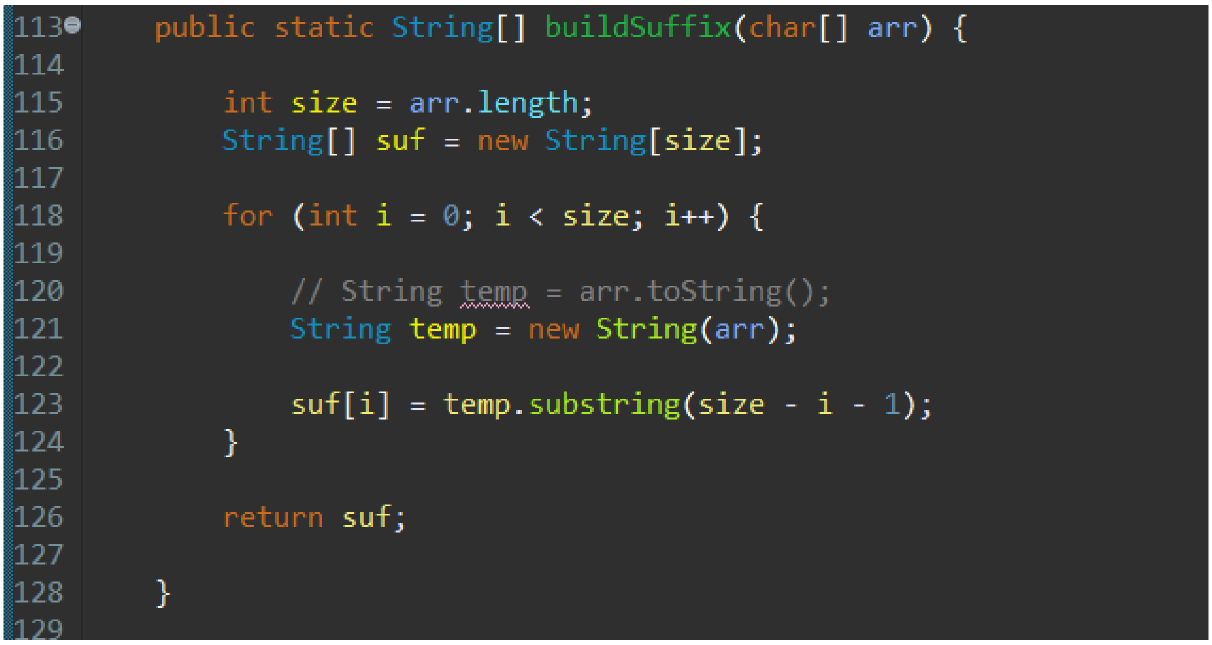
```java
    public static char[] convert(String[] arr) {
        StringBuilder sb = new StringBuilder();
        for (String s : arr) {
            sb.append(s);
        }
        return sb.toString().toCharArray();
    }
```

<u>Direct Compare Function:</u> For the query $LCE_{A,B}(i, j)$, compare the texts A and B starting from positions i and j, directly. W



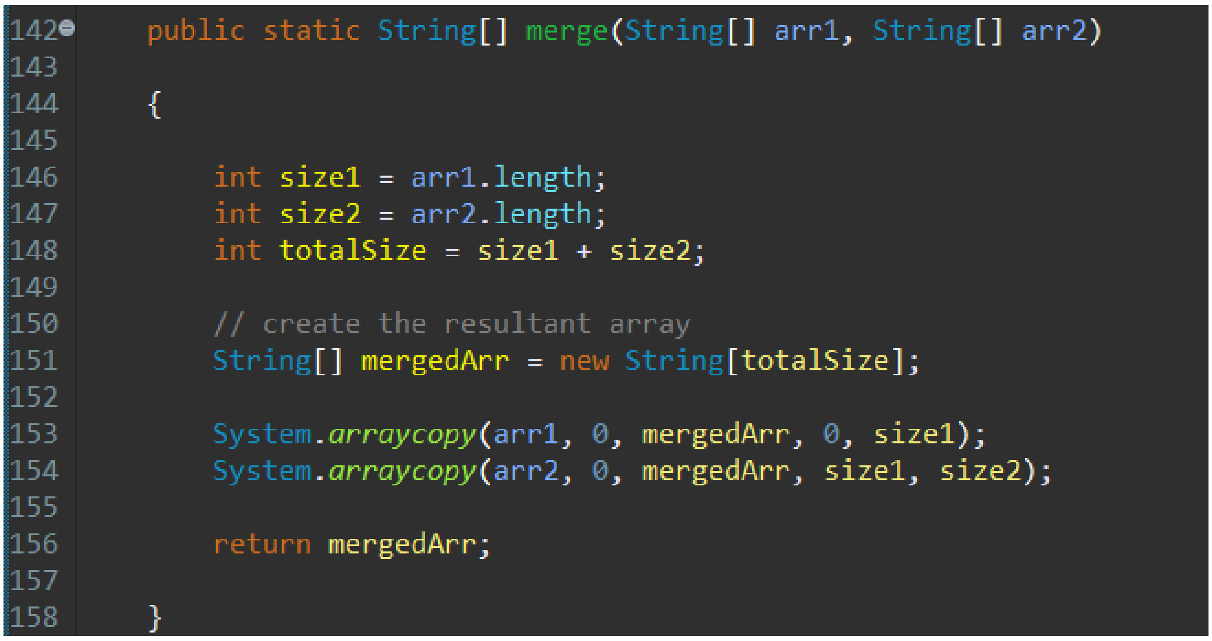
```java
    public static int directCompare(char[] array1, char[] array2, int i, int j) {
        int t = 0;

        while ((array1[i + t] == array2[j + t]) && (i + t <= array1.length) && (j + t <= array2.length)) {
            t = t + 1;
        }

        return t;

    }
```

<u>Build Suffix Array:</u> Building suffix array before sorting.



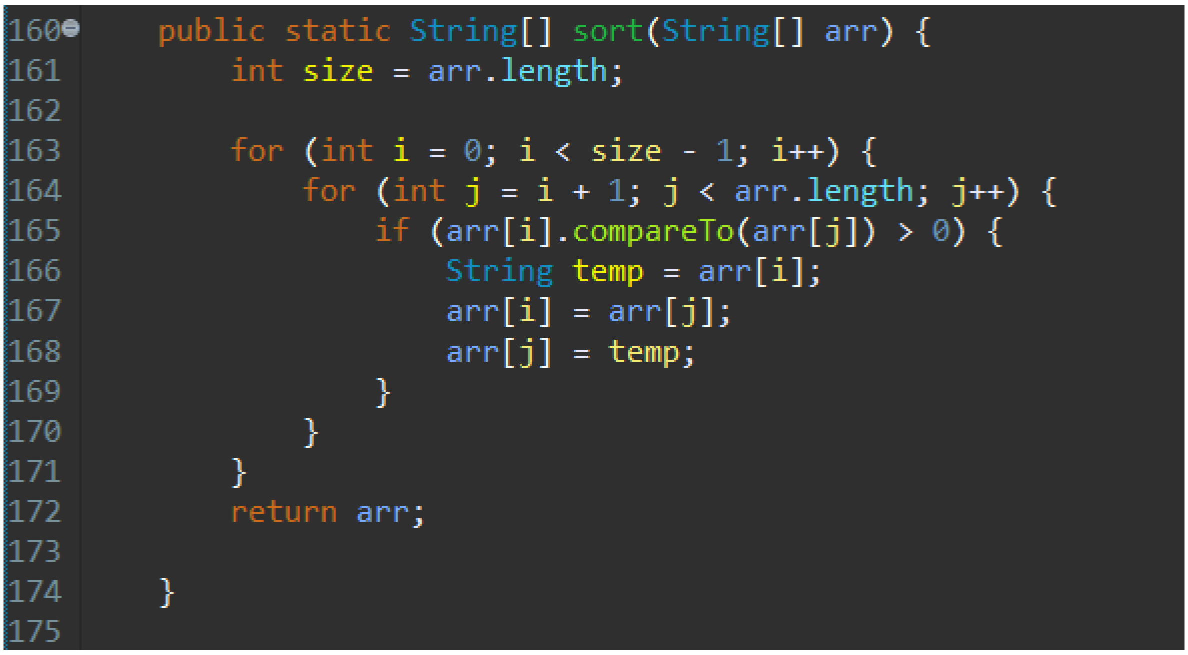
```java
    public static String[] buildSuffix(char[] arr) {

        int size = arr.length;
        String[] suf = new String[size];

        for (int i = 0; i < size; i++) {

            // String temp = arr.toString();
            String temp = new String(arr);

            suf[i] = temp.substring(size - i - 1);
        }

        return suf;

    }
```

Merge Function: Merge two arrays.



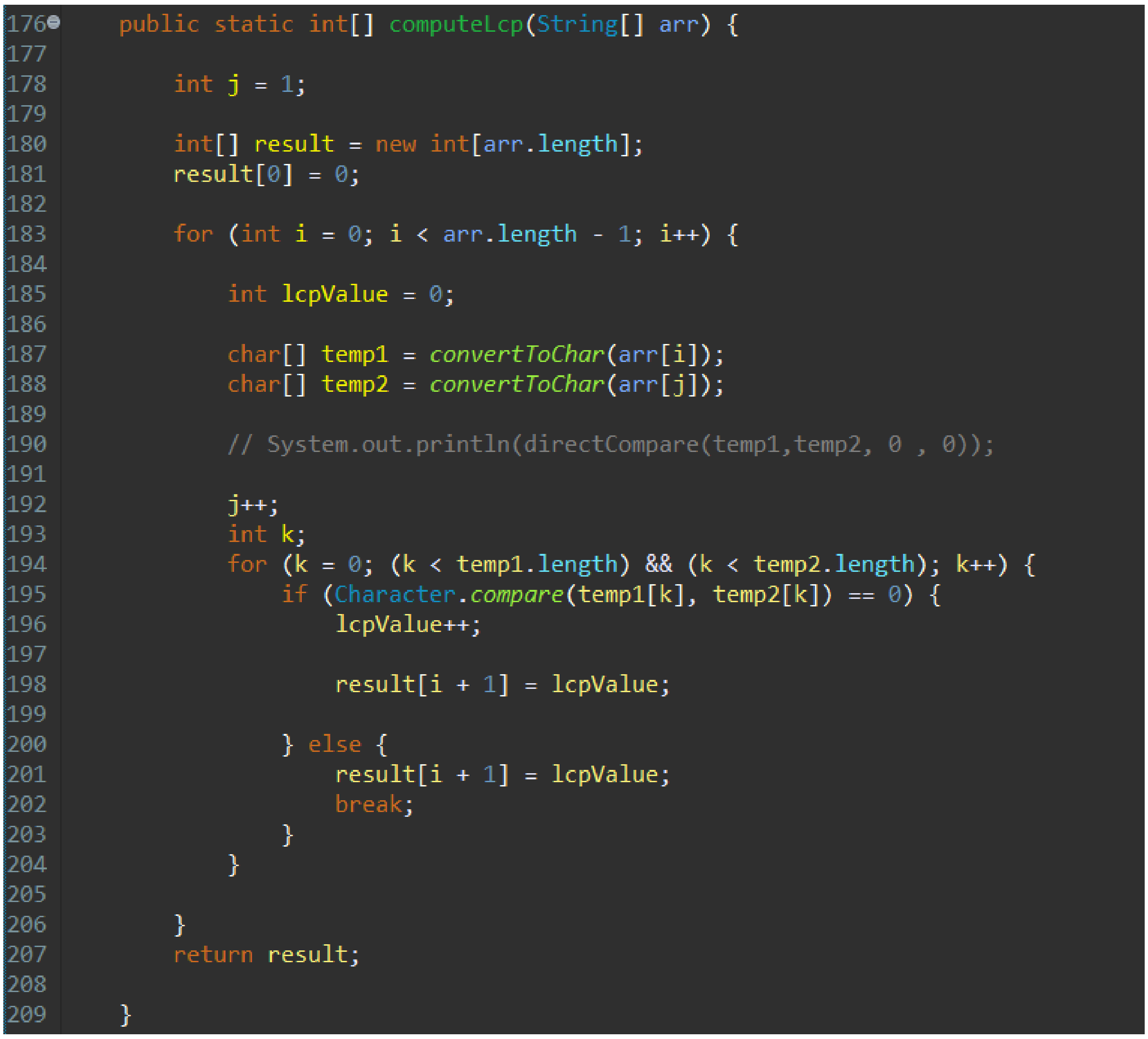
```java
    public static String[] merge(String[] arr1, String[] arr2)

    {

        int size1 = arr1.length;
        int size2 = arr2.length;
        int totalSize = size1 + size2;

        // create the resultant array
        String[] mergedArr = new String[totalSize];

        System.arraycopy(arr1, 0, mergedArr, 0, size1);
        System.arraycopy(arr2, 0, mergedArr, size1, size2);

        return mergedArr;

    }
```

Sort Function: Sorting the suffixes that created before in the “build suffix array” method.



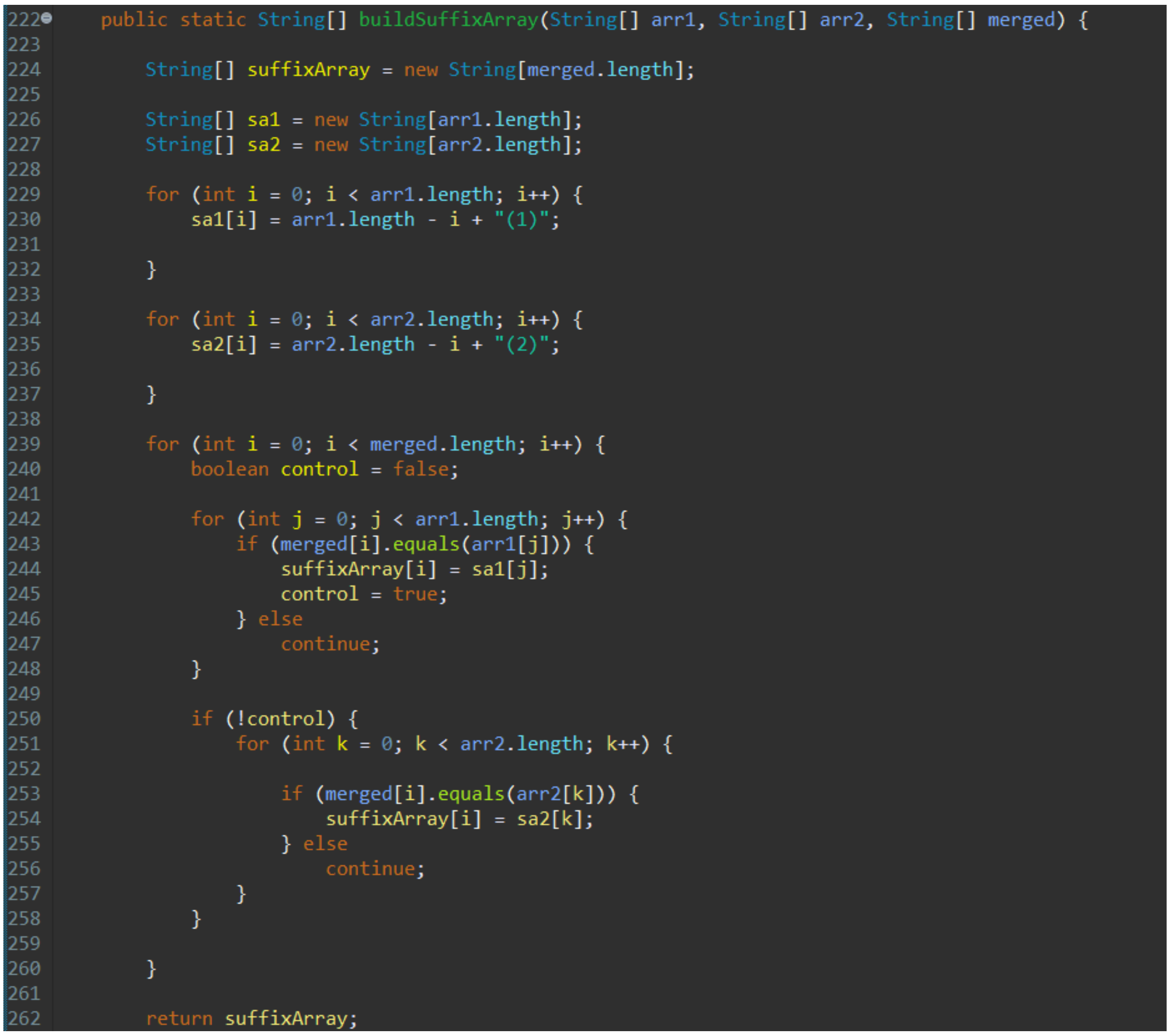
```java
    public static String[] sort(String[] arr) {
        int size = arr.length;

        for (int i = 0; i < size - 1; i++) {
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[i].compareTo(arr[j]) > 0) {
                    String temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
        }
        return arr;

    }
```

Compute LCP Function: Comparing every two consecutive elements in the merged array to compute their LCP values.



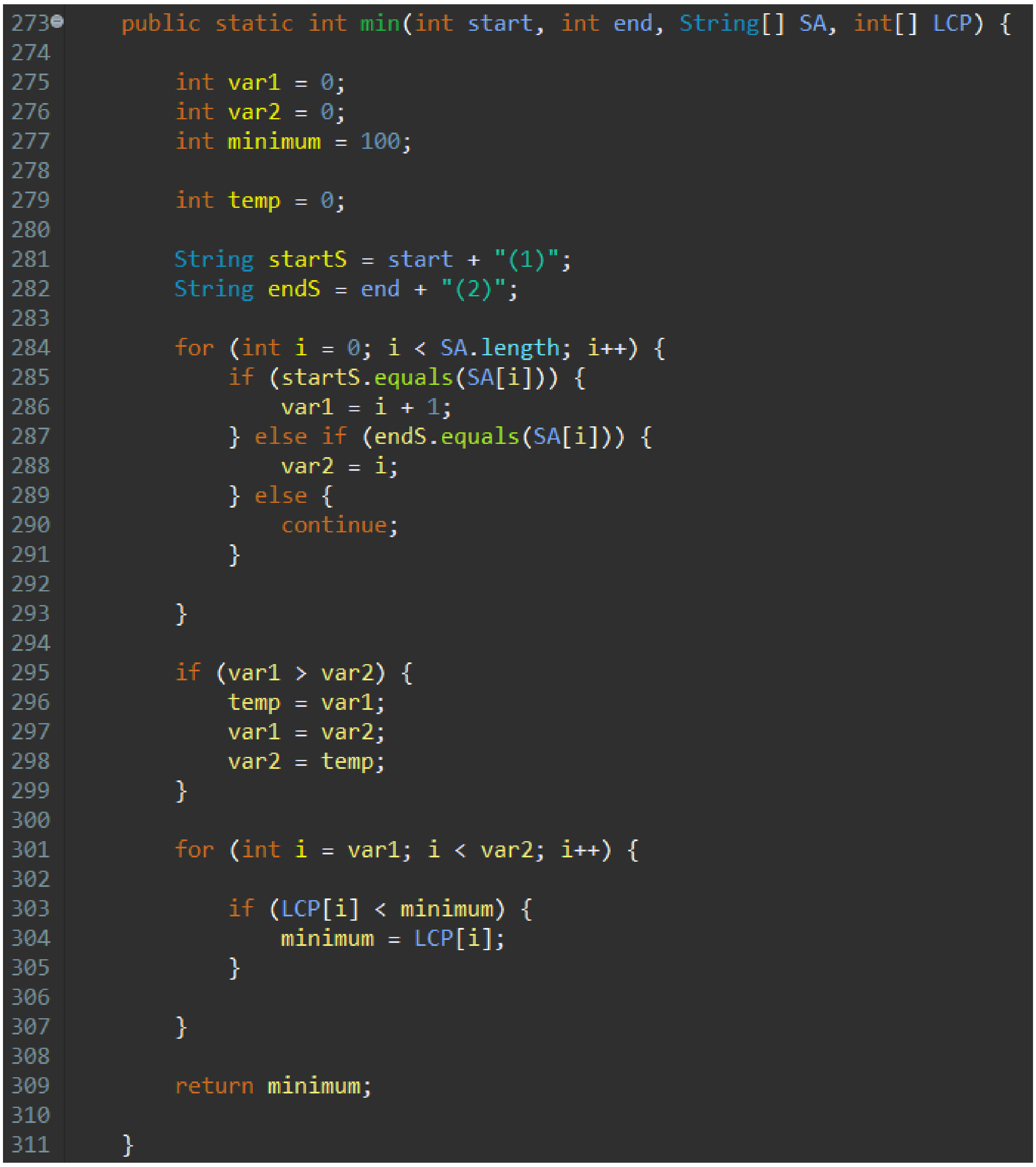
```java
    public static int[] computeLcp(String[] arr) {

        int j = 1;

        int[] result = new int[arr.length];
        result[0] = 0;

        for (int i = 0; i < arr.length - 1; i++) {

            int lcpValue = 0;

            char[] temp1 = convertToChar(arr[i]);
            char[] temp2 = convertToChar(arr[j]);

            // System.out.println(directCompare(temp1,temp2, 0 , 0));

            j++;
            int k;
            for (k = 0; (k < temp1.length) && (k < temp2.length); k++) {
                if (Character.compare(temp1[k], temp2[k]) == 0) {
                    lcpValue++;

                    result[i + 1] = lcpValue;

                } else {
                    result[i + 1] = lcpValue;
                    break;
                }
            }

        }
        return result;

    }
```

Build Suffix array: Matching every suffix in the merged array with the original positions in their own array.



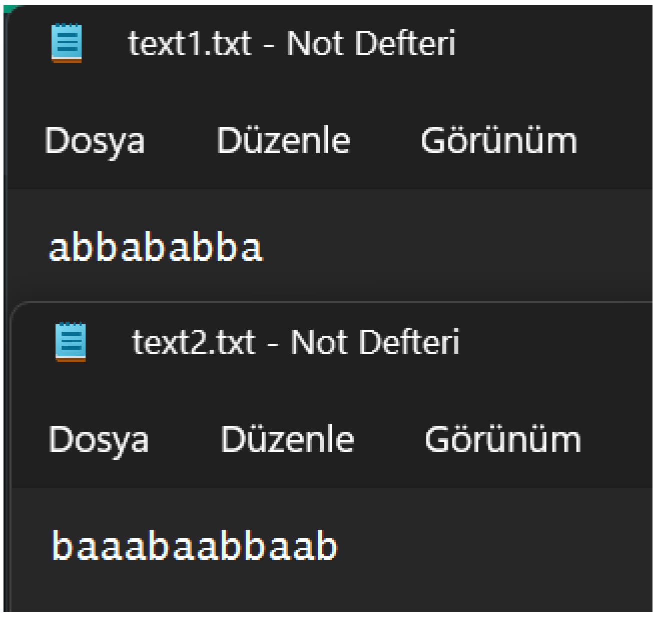
```
    public static String[] buildSuffixArray(String[] arr1, String[] arr2, String[] merged) {

        String[] suffixArray = new String[merged.length];

        String[] sa1 = new String[arr1.length];
        String[] sa2 = new String[arr2.length];

        for (int i = 0; i < arr1.length; i++) {
            sa1[i] = arr1.length - i + "(1)";

        }

        for (int i = 0; i < arr2.length; i++) {
            sa2[i] = arr2.length - i + "(2)";

        }

        for (int i = 0; i < merged.length; i++) {
            boolean control = false;

            for (int j = 0; j < arr1.length; j++) {
                if (merged[i].equals(arr1[j])) {
                    suffixArray[i] = sa1[j];
                    control = true;
                } else
                    continue;
            }

            if (!control) {
                for (int k = 0; k < arr2.length; k++) {

                    if (merged[i].equals(arr2[k])) {
                        suffixArray[i] = sa2[k];
                    } else
                        continue;
                }
            }

        }

        return suffixArray;
```

Min Function: For given two indices, we find their corresponding positions on the suffix array. Then using its index values, we compare LCP array values between these two indexes. We are finding the minimum among them.



```java
    public static int min(int start, int end, String[] SA, int[] LCP) {

        int var1 = 0;
        int var2 = 0;
        int minimum = 100;

        int temp = 0;

        String startS = start + "(1)";
        String endS = end + "(2)";

        for (int i = 0; i < SA.length; i++) {
            if (startS.equals(SA[i])) {
                var1 = i + 1;
            } else if (endS.equals(SA[i])) {
                var2 = i;
            } else {
                continue;
            }

        }

        if (var1 > var2) {
            temp = var1;
            var1 = var2;
            var2 = temp;
        }

        for (int i = var1; i < var2; i++) {

            if (LCP[i] < minimum) {
                minimum = LCP[i];
            }

        }

        return minimum;

    }
```

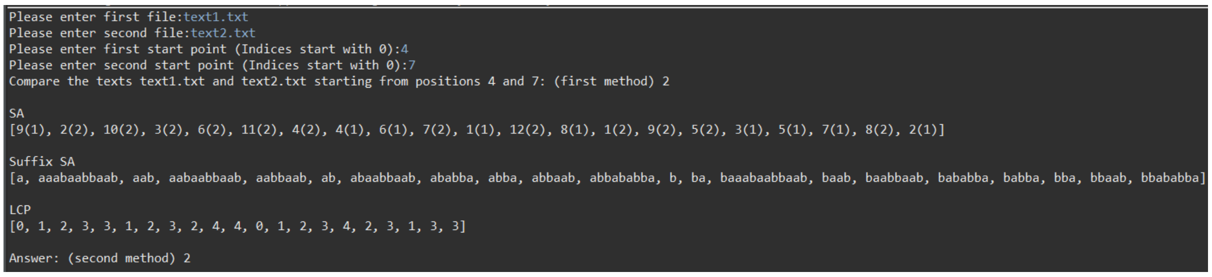
Input Files:



```
abbababba
```

```
baaabaabbaab
```

Example Run:



```
Please enter first file:text1.txt
Please enter second file:text2.txt
Please enter first start point (Indices start with 0):4
Please enter second start point (Indices start with 0):7
Compare the texts text1.txt and text2.txt starting from positions 4 and 7: (first method) 2

SA
[9(1), 2(2), 10(2), 3(2), 6(2), 11(2), 4(2), 4(1), 6(1), 7(2), 1(1), 12(2), 8(1), 1(2), 9(2), 5(2), 3(1), 5(1), 7(1), 8(2), 2(1)]

Suffix SA
[a, aaabaabbaab, aab, aabaabbaab, aabbaab, ab, abaabbaab, ababba, abba, abbaab, abbababba, b, ba, baaabaabbaab, baab, baabbaab, bababba, babba, bba, bbaab, bbababba]

LCP
[0, 1, 2, 3, 3, 1, 2, 3, 2, 4, 4, 0, 1, 2, 3, 4, 2, 3, 1, 3, 3]

Answer: (second method) 2
```