Group 3

Zeke Krug, Ben Burgess

Score/Times

No correctness issues reported.

	Data set 1	Data set 2
Run 1	3041	5603
Run 2	3075	5581
Run 3	3061	5602

Description of Algorithms

- bitCount() - Changed the way that we find the number of 1s in an integer's binary representation to the built in Java method, bitCount(); instead of iterating through the string and counting the number of 1s
- Longest Repeating Substring - Instead of iterating over the string of the binary representation of an integer with 4 for loops, we used a 2D array.
- Added an if statement to only execute the latter two checks when necessary.

LRS (non-overlapping)

```
public static int LNRNOS(String str) {
        int n = str.length();
        int dp[][] = new int[n + 1][n + 1];
        int max = 0, index = 0;
        for (int i = 1; i <= n; ++i) {
            for (int j = i + 1; j <= n; ++j) {
                if (str.charAt(i - 1) == str.charAt(j - 1) && j - i > dp[i - 1][j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                    if (max < dp[i][j]) {
                        max = dp[i][j];
                        //save last index of substring
                        index = Math.max(i, index);
                    }
                } else
                    dp[i][j] = 0;
            }
        }
        int length = str.substring(index - max, index).length();
        return length;
    }
```

Efficiency

N elements

- 2D array (Longest repeating substring) = O(n^3)
- TimSort = O(nlogn)
    - Memory Efficiency = O(n)

Worst Case: Polynomial time O(n^3)

- Due to comparator, in the worst case, having to compute the longest repeating substring (O(n^2)) every comparison

M length string

- Number of 1's found in constant time
- LRS gives O(m^2) efficiency using O(m^2) memory for n strings of m length

Worst case: Quadratic time O(m^2) with O(m^2) memory used

Data Storage

- TimSort allocates additional memory as it is a hybrid of mergesort
- Our algorithm that finds the length of the longest repeating substring creates a 2D array (dependent on the length of the string)

Other things we found/tried/considered

- Suffix Tree
  - To find longest repeating substring
- Bucket sort
  - We talked about bucket-sort, since we knew the distribution of the data.

Sources

Base code LRS from
https://iq.opengenus.org/longest-repeating-non-overlapping-substring/

YouTube Video used to understand the algorithm: Longest Non-Overlapping Repeating Substring - Amazon Coding Interview Question | Dynamic Programming - YouTube

Questions?