Group 2

Richard Lussier, Natasha Zebrev, Audrey LeMeur

Results

- Small Dataset
  - Times of 3832, 3825, 3898
  - Median of 3832
  - 11th place
- Big Dataset
  - Times of 6699, 6730, 6751
  - Median of 6730
  - 11th place

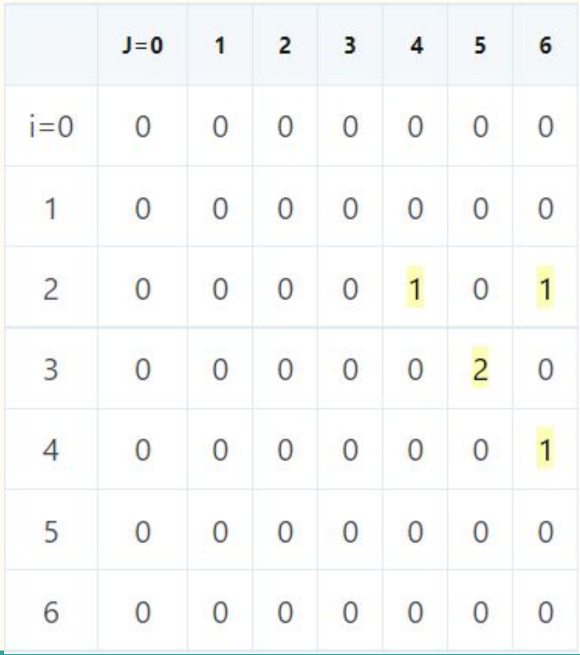
No correctness errors were found

Details of Implementation

- Length of repeated substring was changed to increase speed
- Data is stored as integers and then converted to binary strings for the necessary sorting methods
- No other data structures were used
- Counting number of 1's
    - O(m) time, where m is the length of the binary string
- Longest repeating substring
    - O(m^2) time, where m is the length of the binary string
- Overall, the implementation is O(n) since each element needs to go through each method twice

Longest Repeating Substring

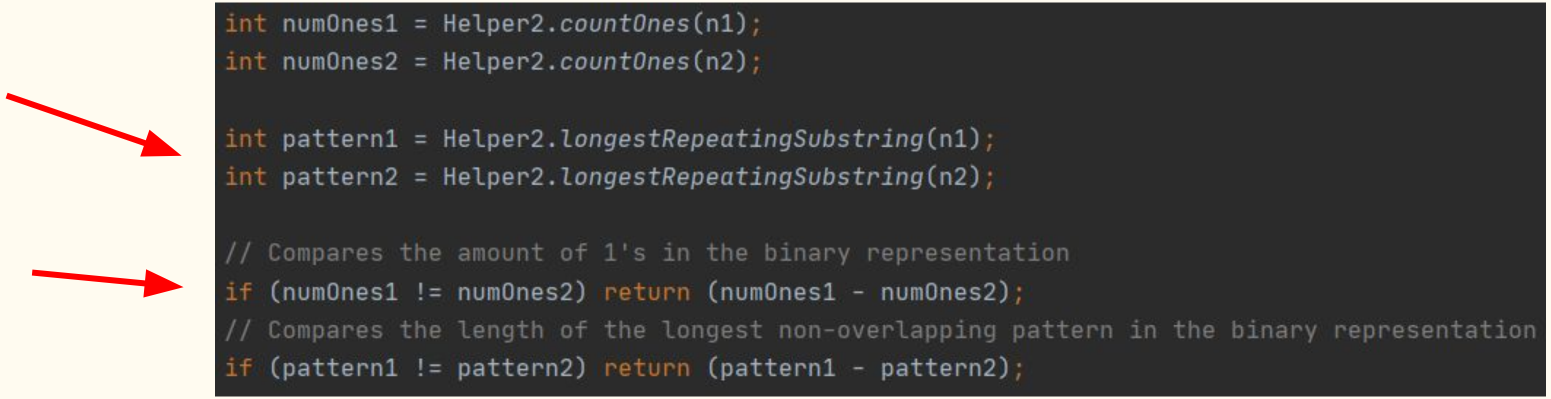
- The only thing we changed in our implementation
- Works by finding every equal character and keeping track of the length of the pattern before each character, then finds the largest value
- https://iq.opengenus.org/longest-repeating-non-overlapping-substring



	J=0	1	2	3	4	5	6
i=0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	1	0	1
3	0	0	0	0	0	2	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

Further Updates

We would switch three of our lines in our compare method so we would limit the number of repeated substring calls we had to do



```
int numOnes1 = Helper2.countOnes(n1);
int numOnes2 = Helper2.countOnes(n2);

int pattern1 = Helper2.longestRepeatingSubstring(n1);
int pattern2 = Helper2.longestRepeatingSubstring(n2);

// Compares the amount of 1's in the binary representation
if (numOnes1 != numOnes2) return (numOnes1 - numOnes2);
// Compares the length of the longest non-overlapping pattern in the binary representation
if (pattern1 != pattern2) return (pattern1 - pattern2);
```