Group 11

Joe Walbran

Overview

- Times: 69 and 93
- Final Score: 162.0 2nd place
- Uses even-odd split mergesort // allows for sorting to be performed slightly faster
  - Stores data in self made class Data
- Longest Repeated Substring
  - Bitwise operators >> <<

```
private static class Data {
  public int value;
  private Integer lengthSubstring = null;

  Data(int value) {
    this.value = value;
  }

  public int getLengthLongestRepeatedSubstring() {
    if (lengthSubstring == null) {
      lengthSubstring = Helper11.lengthLongestRepeatedSubstring(value);
    }
    return lengthSubstring;
  }
```

Algorithm

Time Complexity:

- Even odd Split Mergesort: O(n*log(n))
  - This manages to save a little bit of time by saving some steps
  - Compartor in mergesort ties everything together
- CountBits: O(1)
  - Integer.countBits()
- Find Longest Repeating substring O(m)

n	odd-even mergesort	bitonic sort	shellsort
4	5	6	6
16	63	80	83
64	543	672	724
256	3839	4608	5106
1024	24063	28160	31915

Changes from group 0

This code directly implements even-odd split mergesort with a binary comparator which is similar to Group0;

Faster implementations of Counting bits and finding the longest repeating substring

Designated class for values from file

Longest Repeated Substring

- Split data to be looked at
- Two for loops
- Jump out label using :
- Bitwise right >> left operator <<
- 1 << n times
- 00001010 >> 00000010
- 100 & 101 = 100

```
    public static int lengthLongestRepeatedSubstring(int number) {
int binaryLength = binaryLength(number);
            int longestSubstringLengthSoFar = 0;
            // iterate over possible lengths
            // the longest length is length/2 (rounded down) since they are non-overlapping
            for (int n = 1; n <= binaryLength / 2; ++n) {
  // An integer with the lower n bits set.
  int mask = (1 << n) - 1;

  int upperBoundForI = binaryLength - 2 * n + 1;
  int upperBoundForJ = binaryLength - n + 1;

                    boolean found = false;
                    // first index (the first index of the first copy):
  lookingForSubstringOfLengthN:
                    for (int i = 0; i < upperBoundForI; i++) {
                            // second index (substrings are non-overlapping):
                            for (int j = i + n; j < upperBoundForJ; j++) {
      if (
        ((number >> i) & mask)
        == ((number >> j) & mask)
      ) {
                                            found = true;
        break lookingForSubstringOfLengthN;
                                    }
                            }
                    }

  if (found) {
                            longestSubstringLengthSoFar = n;
                    } else {
                            return longestSubstringLengthSoFar;
                    }
            }

            return longestSubstringLengthSoFar;
    }
```