Hettingley Sorting Comp

Conner Hettinger and Nik Bailey

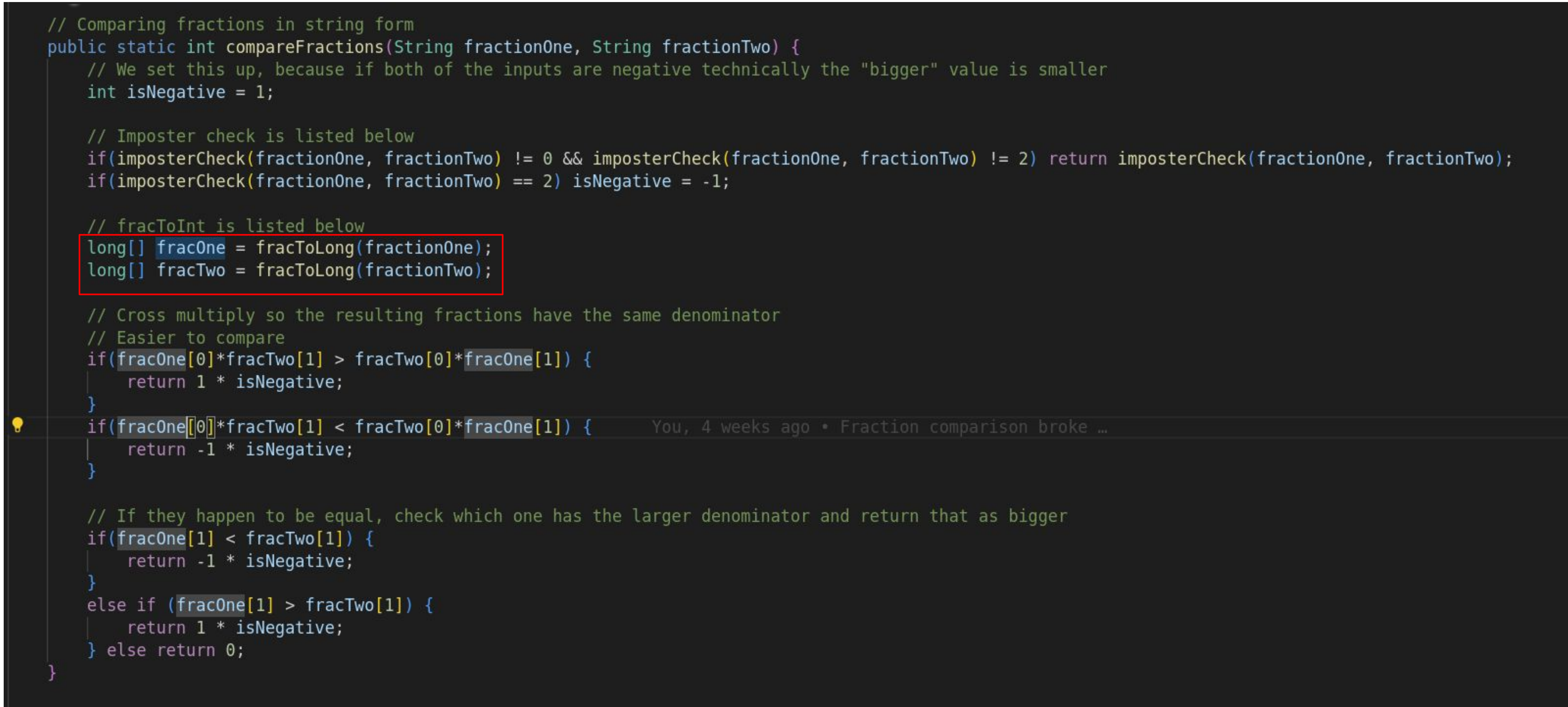
Group 2

Times:

1076	16933
1090	16660
1053	16568
Median: 1076.0	16660.0

Score:
Sum of places is 9
Sum of medians is 17736.0
Took 4th place

Correctness Issue

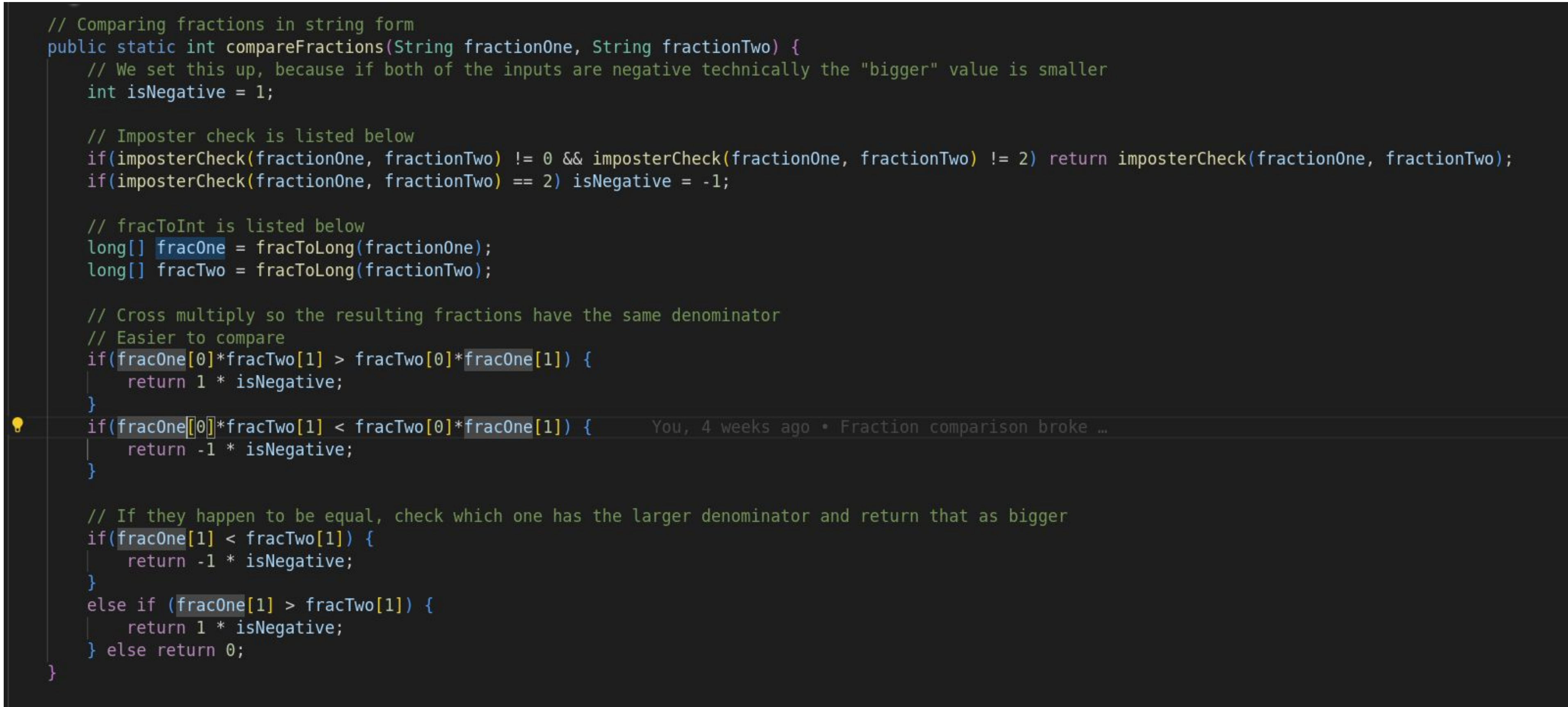


```java
// Comparing fractions in string form
public static int compareFractions(String fractionOne, String fractionTwo) {
    // We set this up, because if both of the inputs are negative technically the "bigger" value is smaller
    int isNegative = 1;

    // Imposter check is listed below
    if(imposterCheck(fractionOne, fractionTwo) != 0 && imposterCheck(fractionOne, fractionTwo) != 2) return imposterCheck(fractionOne, fractionTwo);
    if(imposterCheck(fractionOne, fractionTwo) == 2) isNegative = -1;

    // fracToInt is listed below
    long[] fracOne = fracToLong(fractionOne);
    long[] fracTwo = fracToLong(fractionTwo);

    // Cross multiply so the resulting fractions have the same denominator
    // Easier to compare
    if(fracOne[0]*fracTwo[1] > fracTwo[0]*fracOne[1]) {
        return 1 * isNegative;
    }
    if(fracOne[0]*fracTwo[1] < fracTwo[0]*fracOne[1]) {       You, 4 weeks ago • Fraction comparison broke …
        return -1 * isNegative;
    }

    // If they happen to be equal, check which one has the larger denominator and return that as bigger
    if(fracOne[1] < fracTwo[1]) {
        return -1 * isNegative;
    }
    else if (fracOne[1] > fracTwo[1]) {
        return 1 * isNegative;
    } else return 0;
}
```

What changed from Group 0

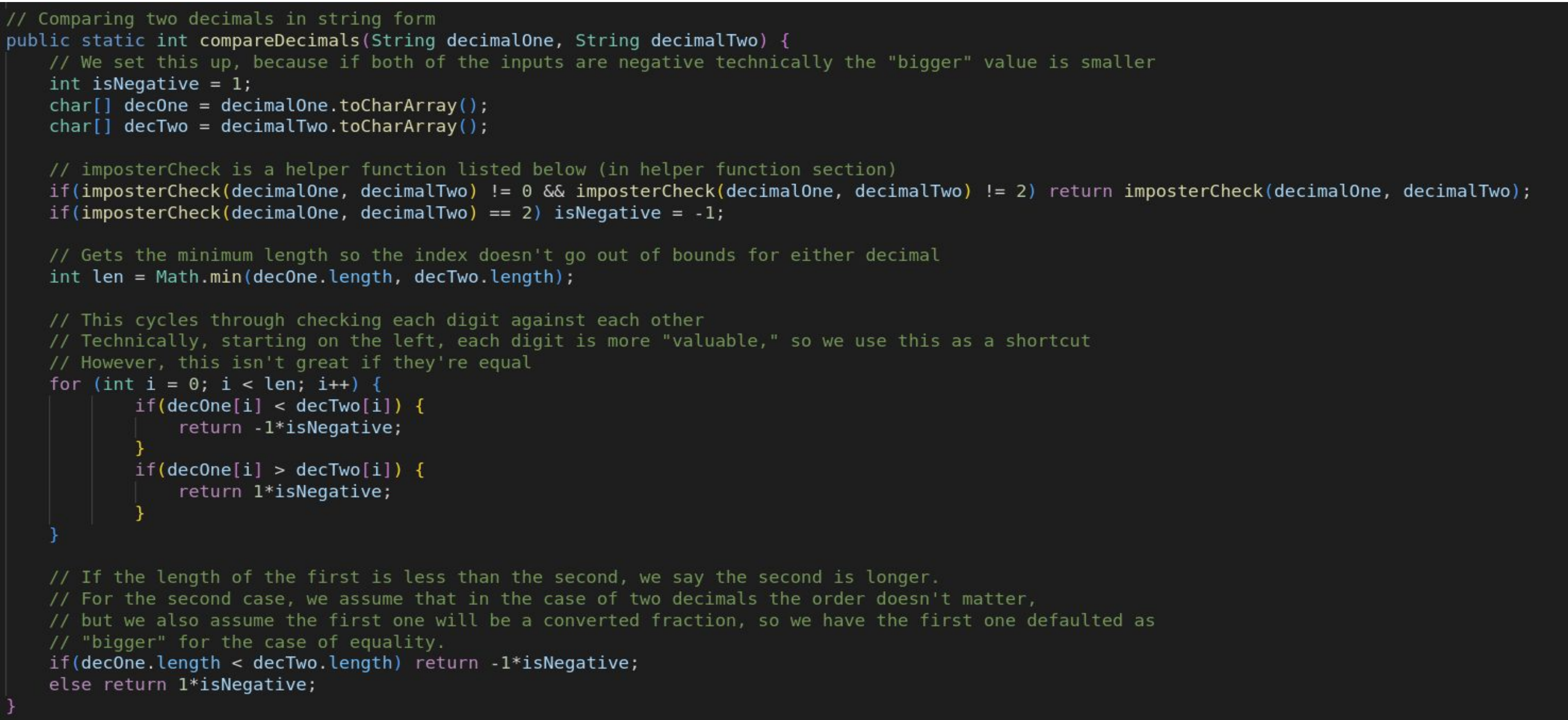
- Comparison methods
  - Compare decimals
  - Compare fractions
  - Helper Methods

compare Fractions



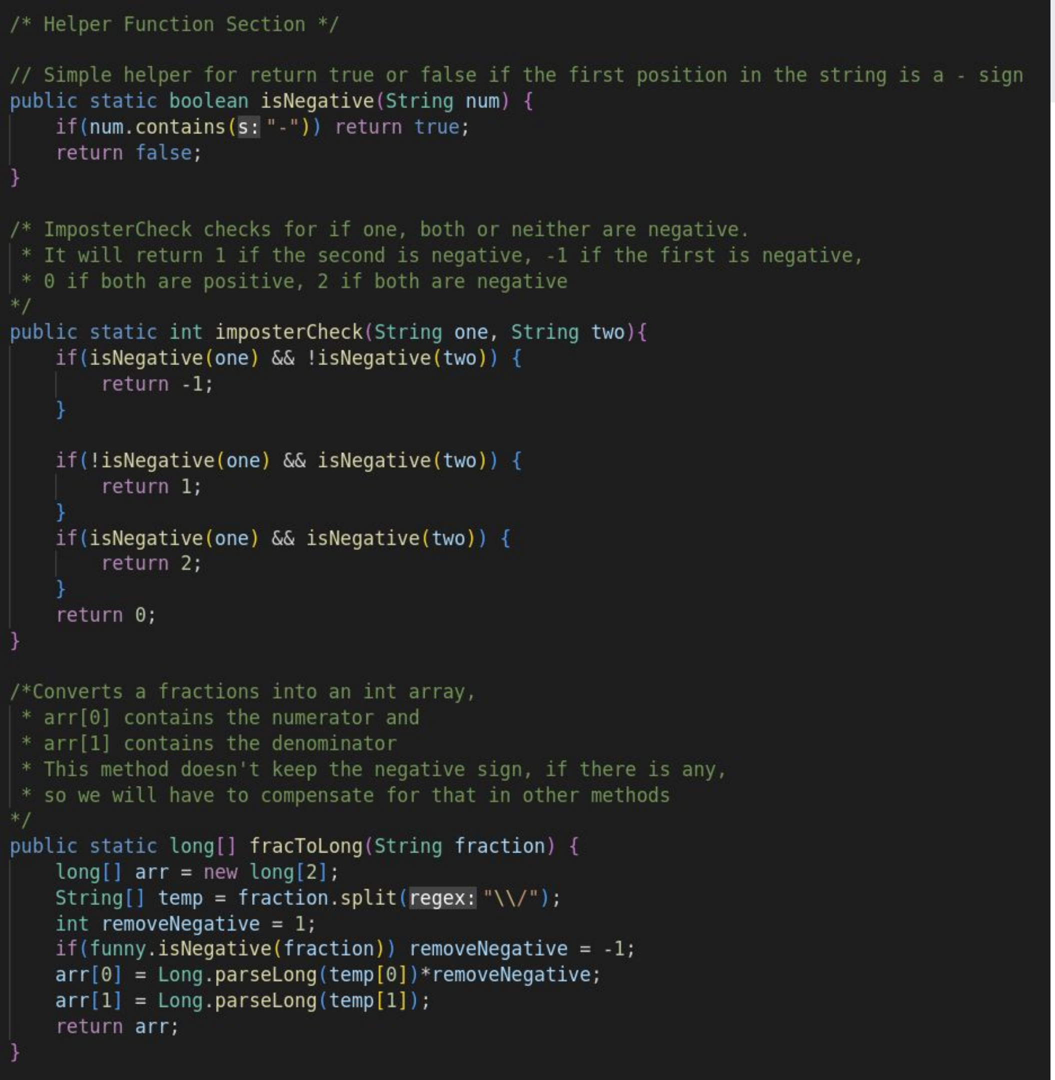
```java
// Comparing fractions in string form
public static int compareFractions(String fractionOne, String fractionTwo) {
    // We set this up, because if both of the inputs are negative technically the "bigger" value is smaller
    int isNegative = 1;

    // Imposter check is listed below
    if(imposterCheck(fractionOne, fractionTwo) != 0 && imposterCheck(fractionOne, fractionTwo) != 2) return imposterCheck(fractionOne, fractionTwo);
    if(imposterCheck(fractionOne, fractionTwo) == 2) isNegative = -1;

    // fracToInt is listed below
    long[] fracOne = fracToLong(fractionOne);
    long[] fracTwo = fracToLong(fractionTwo);

    // Cross multiply so the resulting fractions have the same denominator
    // Easier to compare
    if(fracOne[0]*fracTwo[1] > fracTwo[0]*fracOne[1]) {
        return 1 * isNegative;
    }
    if(fracOne[0]*fracTwo[1] < fracTwo[0]*fracOne[1]) {      You, 4 weeks ago • Fraction comparison broke …
        return -1 * isNegative;
    }

    // If they happen to be equal, check which one has the larger denominator and return that as bigger
    if(fracOne[1] < fracTwo[1]) {
        return -1 * isNegative;
    }
    else if (fracOne[1] > fracTwo[1]) {
        return 1 * isNegative;
    } else return 0;
}
```

compareDecimals



```java
// Comparing two decimals in string form
public static int compareDecimals(String decimalOne, String decimalTwo) {
    // We set this up, because if both of the inputs are negative technically the "bigger" value is smaller
    int isNegative = 1;
    char[] decOne = decimalOne.toCharArray();
    char[] decTwo = decimalTwo.toCharArray();

    // imposterCheck is a helper function listed below (in helper function section)
    if(imposterCheck(decimalOne, decimalTwo) != 0 && imposterCheck(decimalOne, decimalTwo) != 2) return imposterCheck(decimalOne, decimalTwo);
    if(imposterCheck(decimalOne, decimalTwo) == 2) isNegative = -1;

    // Gets the minimum length so the index doesn't go out of bounds for either decimal
    int len = Math.min(decOne.length, decTwo.length);

    // This cycles through checking each digit against each other
    // Technically, starting on the left, each digit is more "valuable," so we use this as a shortcut
    // However, this isn't great if they're equal
    for (int i = 0; i < len; i++) {
            if(decOne[i] < decTwo[i]) {
                return -1*isNegative;
            }
            if(decOne[i] > decTwo[i]) {
                return 1*isNegative;
            }
    }

    // If the length of the first is less than the second, we say the second is longer.
    // For the second case, we assume that in the case of two decimals the order doesn't matter,
    // but we also assume the first one will be a converted fraction, so we have the first one defaulted as
    // "bigger" for the case of equality.
    if(decOne.length < decTwo.length) return -1*isNegative;
    else return 1*isNegative;
}
```

Helper Functions



```java
/* Helper Function Section */

// Simple helper for return true or false if the first position in the string is a - sign
public static boolean isNegative(String num) {
    if(num.contains(s: "-")) return true;
    return false;
}

/* ImposterCheck checks for if one, both or neither are negative.
 * It will return 1 if the second is negative, -1 if the first is negative,
 * 0 if both are positive, 2 if both are negative
*/
public static int imposterCheck(String one, String two){
    if(isNegative(one) && !isNegative(two)) {
        return -1;
    }

    if(!isNegative(one) && isNegative(two)) {
        return 1;
    }
    if(isNegative(one) && isNegative(two)) {
        return 2;
    }
    return 0;
}

/*Converts a fractions into an int array,
 * arr[0] contains the numerator and
 * arr[1] contains the denominator
 * This method doesn't keep the negative sign, if there is any,
 * so we will have to compensate for that in other methods
*/
public static long[] fracToLong(String fraction) {
    long[] arr = new long[2];
    String[] temp = fraction.split(regex: "\\/");
    int removeNegative = 1;
    if(funny.isNegative(fraction)) removeNegative = -1;
    arr[0] = Long.parseLong(temp[0])*removeNegative;
    arr[1] = Long.parseLong(temp[1]);
    return arr;
}
```

Runtime Efficiency

- Use timSort, O(n log n)

Data is stored in the original array

- timSort is a hybrid of insertion sort and mergesort

Optional Stuff

How did we get to where we are:

- Idea of comparing sig figs for decimals
- Cross multiply fractions
- Convert fraction to decimal for comparing fractions and decimals
  - Problem: 1/3, 0.3333333333333333333333333333333333333333333333333333333333333
  - timSort transitive issue, tried other sorting methods

What would we do differently: Comparing fractions and decimals, preprocessing