Group 6

Max Quintavalle & Collin Beane

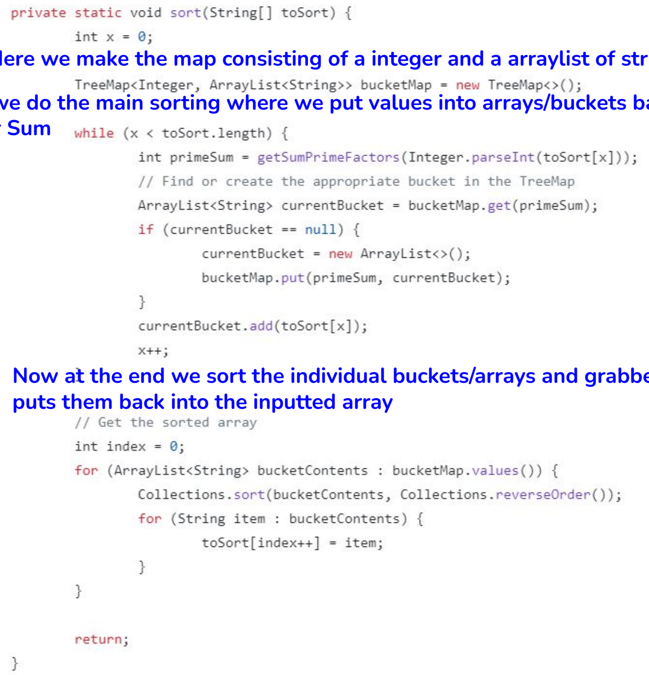
Scores and Times

Prelim 1: (3rd Place)	**Prelim 2: (7th Place)**	**Final: (4th Place)**
Run 1 -	Run 1 -	Run 1 -
1379	62318	14
1365	62317	15
1407	62323	16
Median: 1379.0	Median: 62318.0	Median: 15.0
Run 2 -	Run 2 -	Run 2 -
5928	32831	5
5929	32816	4
5930	32855	8
Median: 5929.0	Median: 32831.0	Median: 5.0
Sum of medians is 7308.0	Sum of medians is 95149.0	Sum of medians is 92.0

No known Issues with correctness at any stage.

Worst/Expected Case: N*log(N)

Sort



```
private static void sort(String[] toSort) {
        int x = 0;
```

Here we make the map consisting of a integer and a arraylist of strings

```
        TreeMap<Integer, ArrayList<String>> bucketMap = new TreeMap<>();
```

Now we do the main sorting where we put values into arrays/buckets based on their Prime Factor Sum

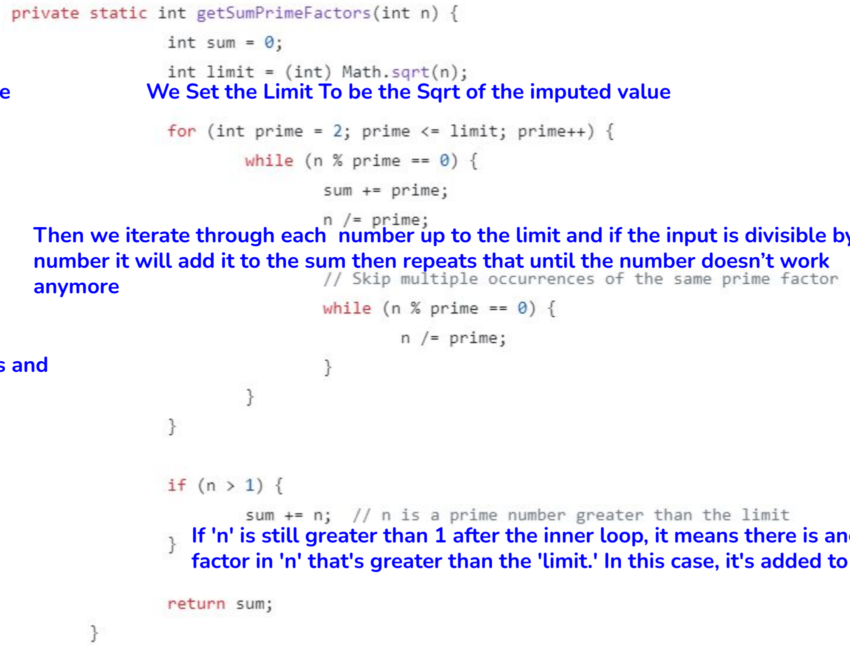
```
        while (x < toSort.length) {
                int primeSum = getSumPrimeFactors(Integer.parseInt(toSort[x]));
                // Find or create the appropriate bucket in the TreeMap
                ArrayList<String> currentBucket = bucketMap.get(primeSum);
                if (currentBucket == null) {
                        currentBucket = new ArrayList<>();
                        bucketMap.put(primeSum, currentBucket);
                }
                currentBucket.add(toSort[x]);
                x++;
```

Now at the end we sort the individual buckets/arrays and grabbed the sortd values and puts them back into the inputted array

```
        // Get the sorted array
        int index = 0;
        for (ArrayList<String> bucketContents : bucketMap.values()) {
                Collections.sort(bucketContents, Collections.reverseOrder());
                for (String item : bucketContents) {
                        toSort[index++] = item;
                }
        }

        return;
}
```

GetSumPrimeFactors



```
private static int getSumPrimeFactors(int n) {
                int sum = 0;
                int limit = (int) Math.sqrt(n);
```

We Set the Limit To be the Sqrt of the imputed value

```
                for (int prime = 2; prime <= limit; prime++) {
                        while (n % prime == 0) {
                                sum += prime;
                                n /= prime;
```

Then we iterate through each number up to the limit and if the input is divisible by the number it will add it to the sum then repeats that until the number doesn’t work anymore

```
                                // Skip multiple occurrences of the same prime factor
                                while (n % prime == 0) {
                                        n /= prime;
                                }
                        }
                }

                if (n > 1) {
                        sum += n;  // n is a prime number greater than the limit
                }
```

If 'n' is still greater than 1 after the inner loop, it means there is another prime factor in 'n' that's greater than the 'limit.' In this case, it's added to the 'sum.'

```
                return sum;
        }
```

Process

- Bucket sort was chosen because each string has a SPF that can be used as a key for a bucket and strings will fall into those buckets
- TreeMap was chosen because it allowed for SPF to be used as a Key, and an arrayList of strings for the values, better than our original bucket object
- Originally, the bucket keys had to be sorted, but using a TreeMap orders automatically by the natural order of its keys
- SPF had to be optimized, a new method was devised, this improvement is the biggest factor in speed from prelim 2 to the final
- If we had more time, we would have liked to insert values into the arrayLists directly where they belonged, eliminating the sorting step at the end