Group stats

- Group 1
- Will and Alwin
- First dataset:
  - Median time: 254334
  - Place: 8th
- Second dataset
  - Median time: 6263
  - Place: 8th
- Third dataset
  - Median time: ???
  - Place: 5th???
- No correctness issues

Algorithm structure

- Pair class implements the comparable interface allowing us to find the greater sum of primes or, if those are equal, the greater number.
- Construct an array of pairs called digitpair and sort it using java's built in mergesort
- For some reason after this we still have a few errors in reverse sorted portions:/
- Use an insertion-sort-esque algorithm to fix these errors
- Sum of prime factors of n is found by looping through the numbers less than n, checking whether n is divisible by each and whether each are prime, and if both are true, adding each to a sum

Algorithm efficiency

- Our theoretical worst case efficiency is nlogn because of our use of mergesort on the array with no change to the comparator.
- The efficiency of our getSumPrimeFactors method is n * num_size.
  - Let’s say n is 100 and num_size is 100. Then the efficiency of this method is n^2. Thus, a large num_size can decrease our overall efficiency drastically

What worked and what didn’t

- How we arrived at our algorithm
  - We severely over thought the problem
  - the user defined pair class overcomplicated how we were storing data
  - A few functions that introduced unneeded passes through the given data i.e our creation of pairs and sorting of identical sums

Future steps if given more time

- Get rid of pairs entirely
- Implement a hashmap to store the sum primes with the number as it’s key
  - This allows us to faster return the prime of repeat numbers
  - The memory usage to do this may not be worth it however
- Implement a better way of dealing with repeats, this would probably be changing the comparator or the compareTo of a custom class.