Group 11

Shawn Saliyev and Brian Caravantes
3rd place overall(1st in class)
Final sum of medians: 1776.0ms
Sorted Correctly

Big Picture of Algorithm

- Sort with Arrays.sort(String[] a, Comparator c)
- Uses inner PrimesComparator class to compare the data

Details of Primes Comparator class

- productOfPrimeFactors(long n)
  - To speed up the process of finding product of prime factors we use various math properties.

```
if(n%2==0){
    if(n%10==0){
        if(n%3==0){
            return 6;
        }else if(n%5==0){
            return 10;
        }
    }

    if((n%3)==0)return 6;
```

- compare(String s1, String s2)
  - Uses HashMap to store product of primes that were just computed
  - Before we compare two numbers we check if we already calculated the product of prime factors of these numbers,
  - if not we call productOfPrimeFactors method and then store result in map

Worst case efficiency

Theoretical Efficiency: Arrays.sort(...) uses a stable, adaptive, iterative mergesort where the worst case is Θ (nlog(n))

Expected Efficiency: Θ(nlog(n)) when the data is randomly ordered and a little bit less than Θ(nlog(n)) when data is sorted.

Final Thoughts

- What worked:
  - One interesting feature is the way we traverse through prime numbers
  - Storing the products of prime factors gives us the significant rise in terms of performance.
- What didn’t work: Towards the final round we tried to apply Pollard Rho algorithm to calculate prime factors but couldn’t get it to work :(
- What we would have done differently is changing our sorting algorithm to a dual pivot quicksort that sorts on an array of longs.