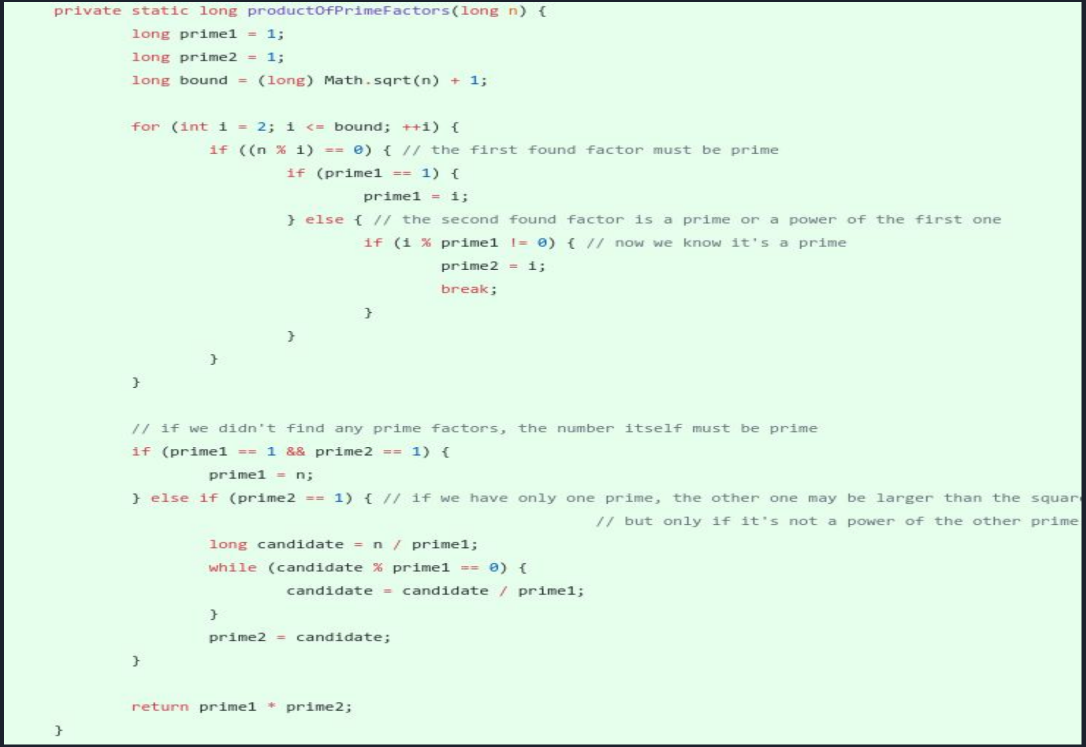
# Sorting Competition: Group13, 8th Place

Philip Blaskowski and Adam Casey

# The Algorithm

- We focused on improving the speed at which we find prime factors.
    - At first we thought that identifying if N is prime would increase the speed of the algorithm so we used BigInteger’s isProbablePrime to do this.
    - If N wasn’t prime we just used a modified version of Group 0’s old productOfPrimeFactors()!
    - Then we use Arrays.sort() to sort our program

# Group 0’s old Code



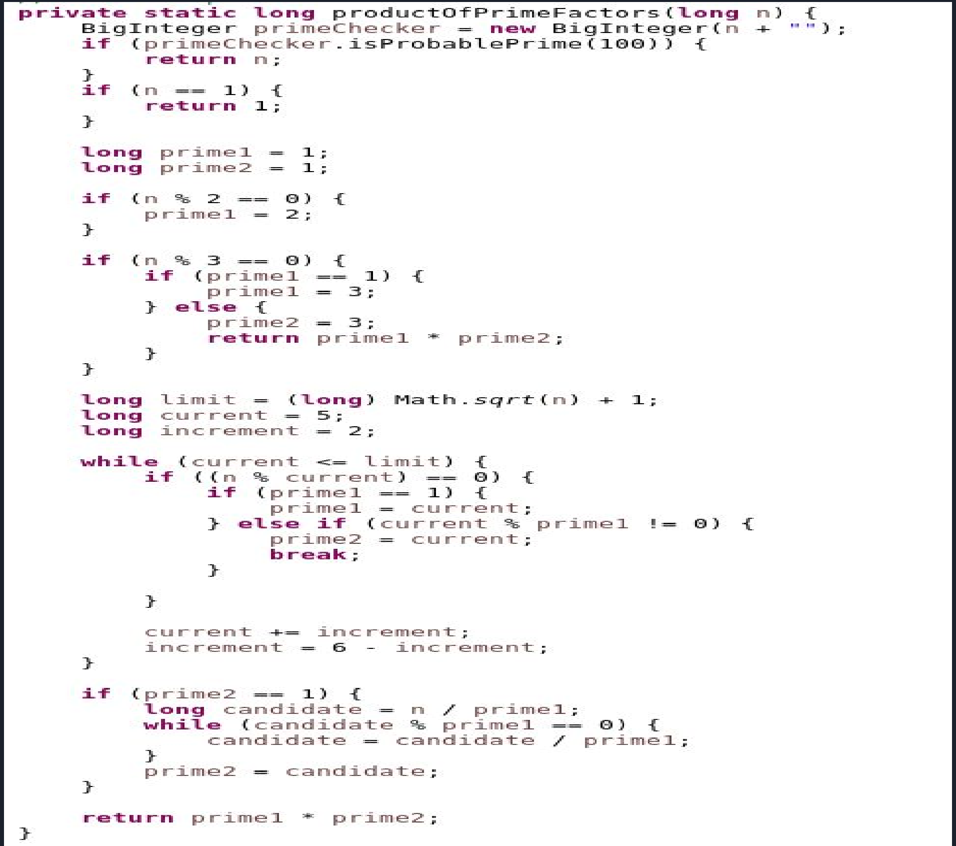
```
private static long productOfPrimeFactors(long n) {
        long prime1 = 1;
        long prime2 = 1;
        long bound = (long) Math.sqrt(n) + 1;

        for (int i = 2; i <= bound; ++i) {
                if ((n % i) == 0) { // the first found factor must be prime
                        if (prime1 == 1) {
                                prime1 = i;
                        } else { // the second found factor is a prime or a power of the first one
                                if (i % prime1 != 0) { // now we know it's a prime
                                        prime2 = i;
                                        break;
                                }
                        }
                }
        }

        // if we didn't find any prime factors, the number itself must be prime
        if (prime1 == 1 && prime2 == 1) {
                prime1 = n;
        } else if (prime2 == 1) { // if we have only one prime, the other one may be larger than the squar
                                                        // but only if it's not a power of the other prime
                long candidate = n / prime1;
                while (candidate % prime1 == 0) {
                        candidate = candidate / prime1;
                }
                prime2 = candidate;
        }

        return prime1 * prime2;
}
```

# Our Code



```
private static long productOfPrimeFactors(long n) {
    BigInteger primeChecker = new BigInteger(n + "");
    if (primeChecker.isProbablePrime(100)) {
        return n;
    }
    if (n == 1) {
        return 1;
    }

    long prime1 = 1;
    long prime2 = 1;

    if (n % 2 == 0) {
        prime1 = 2;
    }

    if (n % 3 == 0) {
        if (prime1 == 1) {
            prime1 = 3;
        } else {
            prime2 = 3;
            return prime1 * prime2;
        }
    }

    long limit = (long) Math.sqrt(n) + 1;
    long current = 5;
    long increment = 2;

    while (current <= limit) {
        if ((n % current) == 0) {
            if (prime1 == 1) {
                prime1 = current;
            } else if (current % prime1 != 0) {
                prime2 = current;
                break;
            }

        }

        current += increment;
        increment = 6 - increment;
    }

    if (prime2 == 1) {
        long candidate = n / prime1;
        while (candidate % prime1 == 0) {
            candidate = candidate / prime1;
        }
        prime2 = candidate;
    }

    return prime1 * prime2;
}
```

# Data Representation

- We didn’t change the original data representation
- We are still taking an array of longs
- Our functions still take longs

# But..... is it correct?

- Our algorithm was evaluated by other classmates and was determined to sort correctly.
- Also, since our algorithm is just a slightly modified version of group0’s old code, which we know sorted correctly, this algorithm should also sort correctly.

# Theoretical stuff

- Since we are using TimSort to sort our data, our worst case for sorting is BigTheta(nlog(n)).
- We believe that the isProbablePrime method causes the biggest slow down in our program and it’s efficiency is unknown to us. Apparently it uses miller-rabin primality testing in some way but it’s efficiency is not mentioned anywhere. So our best estimation of the worst case is BigTheta(nlog(n)). It’s possible it is larger if isProbablePrime is slower but we don’t know that.

# Things we tried

We tried to implement Pollard’s Rho algorithm. We found that it was a lot less painful to just abandon this experiment and move on to something more manageable.

# Our mistakes

Using BigInteger’s isProbablePrime() method with a certainty of 100, was a mistake and probably caused the demise of our program.