Farrah Rahman
Can publish on site

1. Suppose there are finitely many primes, $p_1, p_2, p_3, \cdots, p_n$.
   Consider the number $p_1 \cdot p_2 \cdot p_3 \cdots p_{n-1} \cdot p_n + 1 + 1$. Since p_1 divides $p_1 \cdot p_2 \cdots p_n$, it follows that $(p_1 \cdot p_2 \cdots p_n + 1)\%p_1 = 1$. The same logic can be extended to show that the remainder of $(p_1 \cdot p_2 \cdots p_n + 1)\%p_1 = 1$ divided by each of the primes p_1 through p_n is 1. So $p_1 \cdot p_2 \cdots p_n + 1$ isn't divisible by any of the known primes. Every composite number has a prime factorization involving prime(s) other than itself. If $p_1 \cdot p_2 \cdots p_n + 1$ is composite, since no primes of its prime factorization have been found it must be that there must exist at least another prime outside of the known primes of the list. If $p_1 \cdot p_2 \cdots p_n + 1$ is prime, then it itself is a prime existing outside of the list. Therefore any finite list of primes is incomplete. In other words, there are infinitely many primes.

2. Not exactly the prompt but I'm studying for an interview and really don't want to do this by hand and this probably demonstrates my understanding anyway
   Output:
   2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139

```java
public List<Integer> computePrimes(int n) {
    List<Integer> retVal = new ArrayList<>();
    int currPrime = 2;
    int[] nums = new int[n+1]; //use indices 2 through n
    int maxPrime = (int)Math.sqrt(n);
    while(currPrime <= maxPrime) {
        for(int i = currPrime*2; i <= n; i += currPrime)
            nums[i] = -1;
        currPrime++;
        while((currPrime <= maxPrime) && (nums[currPrime] == -1))
            currPrime++;
    }
    for(int i = 2; i <= n; i++)
        if(nums[i] == 0)
            retVal.add(i);
    return retVal;
}
```

3. Output:
$3^1 7^1 11^1 13^1$

```java
public List<String> computePrimePowers(int n) {
    List<Integer> possiblePrimes = computePrimes((int)Math.sqrt(n));
    List<String> retVal = new ArrayList<>();
    int remainingNum = n;
    while(remainingNum > 1) {
        int oldRemaining = remainingNum;
        for(Integer prime: possiblePrimes) {
            if(remainingNum % prime == 0) {
                int currPrimePower = 1;
                int power = 0;
                while(remainingNum % (currPrimePower*prime) == 0) {
                    currPrimePower *= prime;
                    power++;
                }
                retVal.add(prime+"^"+power);
                remainingNum /= currPrimePower;
            }
        }
        if(oldRemaining == remainingNum) { //n is prime
            retVal.add(n+"^"+1);
            return retVal;
        }
    }
    return retVal;
}
```

4. Number of Primes:
$e^{100}/100 = 268,811,714,181,610,000,000,000,000,000,000,000,000,000$