Primes

http://bit.ly/VTProgPrimes

What are Prime Numbers?

A prime number is a number with no divisors other than 1 and itself. They have many interesting mathematical properties, and show up in a few different areas as such.

Checking Primality

How can we go about checking if a number is prime?

```
public boolean isPrimeSuperSlow(int N) {
    for (int i = 2; i < N; i++) {
        if (N % i == 0) {
            return false;
        }
    }
    return true;
}
```

```
public boolean isPrimeSlow(int N) {
    for (int i = 2; i < N/2; i++) {
        if (N % i == 0) {
            return false;
        }
    }
    return true;
}
```

Factor Pairs

All factors of a number come in pairs, we refer to these as factor pairs. It is exactly how most people factor a number in their head. For example:

24: (1, 24), (2, 12), (3, 8), (4, 6)

36: (1, 36), (2, 18), (3, 12), (4, 9), (6, 6)

Factor Pairs

24:

1 2 3 4 | 6 8 12 24

36:

1 2 3 4 6 9 12 18 36

```
public boolean isPrime(int N) {
    for (int i = 2; i*i < N; i++) {
        if (N % i == 0) {
            return false;
        }
    }
    return true;
}
```

Finding Factors

Sometimes we will want to find the factors of a given number, we can use a similar technique as we just showed to do this.

```
public ArrayList<Integer> factors(int N) {
    ArrayList<Integer> factors = new ArrayList<Integer>();
    for (int i = 1; i*i <= N; i++) {
        if (N%i == 0) {
            factors.add(i);
            if (N/i != i) {
                factors.add(N/i);
            }
        }
    }
    return factors;
}
```

Prime Factorization

The prime factorization of a number is another important mathematical concept. Every number has a unique set of primes that multiply into it. Examples:

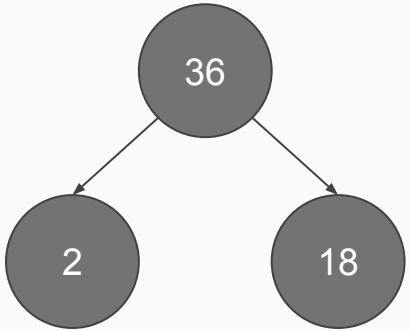
24: $2 * 2 * 2 * 3$ or $2^3 * 3^1$

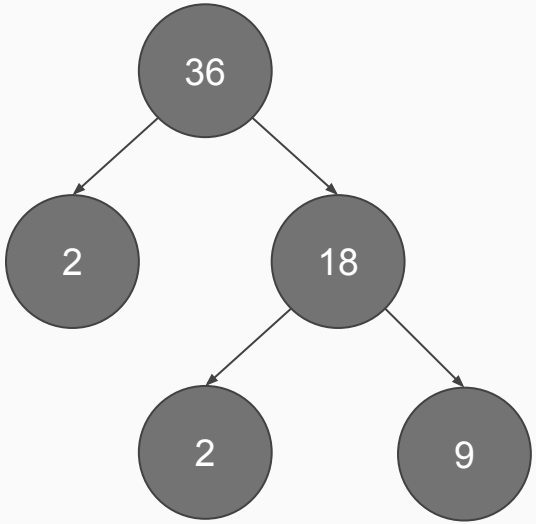
36: $2 * 2 * 3 * 3$ or $2^2 * 3^2$

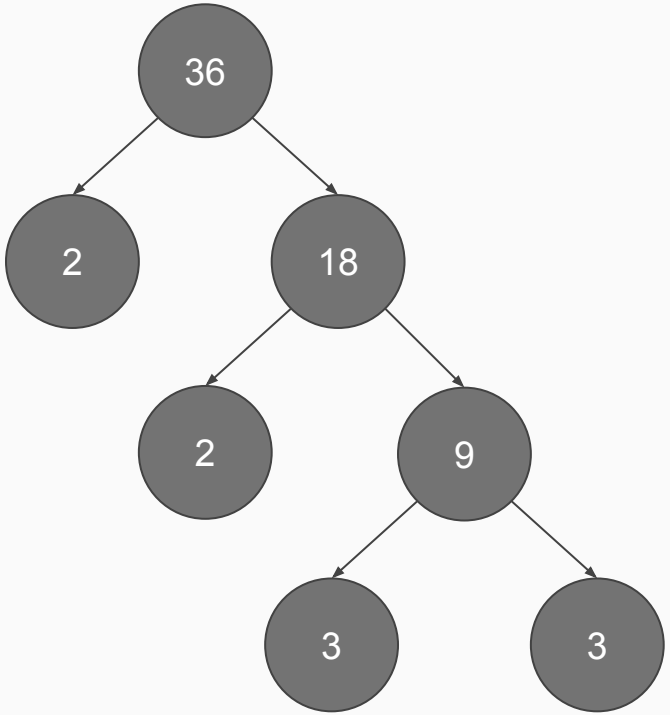
Important considerations

- A number can have a prime factor that is larger than the square root of the number, but only 1 such prime factor
- We can break the problem down by using the concept of factorization trees


36

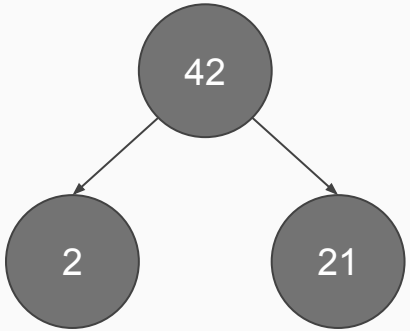

36
2
18

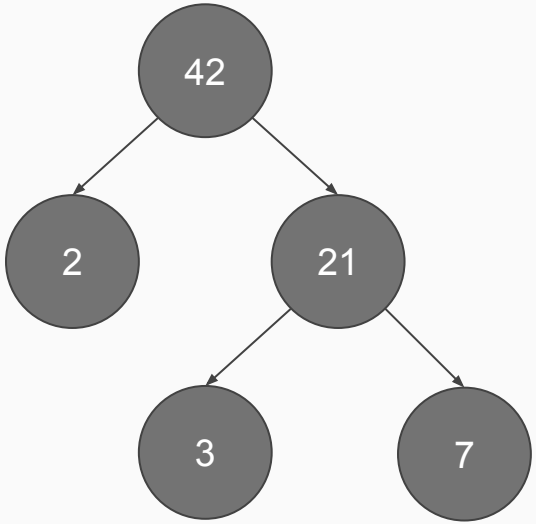

36
2
18
2
9



Factorization Tree - 36


42


42
2
21



Factorization Tree - 42

```
public ArrayList<Integer> primeFactorization(int N) {
    ArrayList<Integer> primeFacs = new ArrayList<Integer>();
    for (int i = 2; i*i <= N; i++) {
        if(N%i == 0 && isPrime(i)) {
            while (N % i == 0) {
                primeFacs.add(i);
                N /= i;
            }
        }
    }
    if (N > 1) {
        primeFacs.add(N);
    }
    return primeFacs;
}
```

Generating Primes

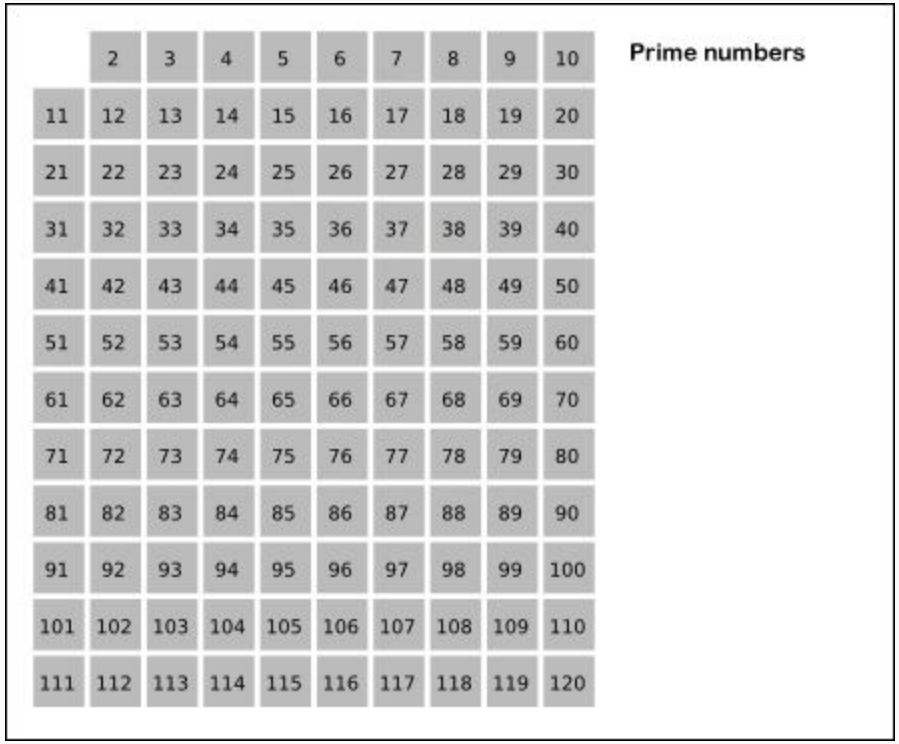
How would you go about generating a list of all the prime numbers up to a certain value? For example:

```
primes(30):
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29
```

```
public ArrayList<Integer> primes(int N) {
    ArrayList<Integer> primes = new ArrayList<Integer>();
    for (int i = 0; i <= N; i++) {
        if (isPrime(i)) {
            primes.add(i);
        }
    }
    return primes;
}
```

Sieve of Eratosthenes

We know that every number that is not prime must be divisible by a prime number that is less than its square root. We can use this knowledge to “mark off” numbers that can not be prime, and then take the rest as they are guaranteed to be prime.



Sieve of Eratosthenes

```java
public ArrayList<Integer> primes(int N) {
    boolean[] isPrime = new boolean[N];
    Arrays.fill(isPrime, true);
    isPrime[0] = isPrime[1] = false;
    for (int i = 2; i*i < N; i++) {
        if (isPrime[i]) {
            for (int j = i*i; j <= N; j+=i) {
                isPrime[j] = false;
            }
        }
    }
    ArrayList<Integer> primes = new ArrayList<Integer>();
    for (int i = 0; i < N; i++) {
        if (isPrime[i]) {
            primes.add(i);
        }
    }
    return primes;
}
```

Generating Primes - Using Sieve of Eratosthenes - O(N*log(log(N)))

Problems

- https://projecteuler.net/problem=3
- https://projecteuler.net/problem=7
- https://projecteuler.net/problem=10
- https://projecteuler.net/problem=27
- https://projecteuler.net/problem=35
- https://projecteuler.net/problem=37
- https://projecteuler.net/problem=41
- https://projecteuler.net/problem=50