# Lab Documentation

## Problem context

A company needs to improve the security of their systems, and to do this they need to implement encryption algorithms, which may use prime numbers to aid in the encryption process.

## Developing a solution:

A solution for this problem can be obtained using the 7 Step Method for Engineering

### Step 1: Identifying the problem

Identifying the symptoms and problems:

- The company needs to improve their system's security.
- In order to improve their system's security the company needs to develop encryption algorithms.
- The encryption algorithms may use prime numbers.

Defining the problem:

- The company needs a program that generates prime numbers

### Step 2: Research

| **Name** | FR1: Identify prime numbers |
|---|---|
| **Summary** | The system allows identifying (Mark) all the prime numbers up to n, where n is greater than 0 |
| **Input** | The **max value** (n) |
| **Output** | All the prime numbers up to n |

| **Name** | FR2: Squarest matrix as possible |
|---|---|
| **Summary** | The system allows creating a matrix the squarest as possible with all numbers up to n (Included). |
| **Input** | The **max value** (n) |
| **Output** | A square matrix |

*Primes*
Prime numbers are positive integers that are divisible only by 1 and themselves. Positive integers that are not prime are called composite numbers; but 1 is a special case and is not considered a prime or a composite number.

<u>Algorithms to find prime numbers:</u>

*Naive Algorithm*
A simple algorithm to find out if an integer (n) is prime is to use the modulo operator (or divide manually if you do not have a computer) to evaluate if the number is divisible by a number other than 1 and itself.

*Naive Algorithm Improved*
A simple algorithm to find out if an integer (n) is prime is to use the modulo operator (or divide manually if you do not have a computer) to evaluate if the number is divisible by a number other than 1 and itself. Furthermore, multiples of 5 and 2 (other than 5 and 2 themselves) are ignored, since it is already known that these numbers are not primes.

*Sieve of Eratosthenes*
An algorithm that originated in ancient Greece. It consists of taking a list of integers less than a number n, and taking a number p (starting from 2), and marking every multiple of that number p starting from p squared. It has a time complexity of O(nloglogn).

*Sieve of Sundaram*
An algorithm discovered by Indian mathematician S. P. Sundaram in 1934. This algorithm finds all the prime numbers up to a specified number n. In a list of 1 to n numbers, remove all the numbers of the form *i* + *j* + 2ij and the remaining numbers are doubled and incremented by one (2*x* + 1), giving a list of the odd prime numbers (all primes except 2) below 2n + 2

*Probabilistic algorithms*
Are much faster than deterministic algorithms, but the number is not always a prime. These algorithms are actually used very frequently in cryptography and systems security.

*Sieve of Atkin*
An algorithm that uses module 60 and polynomial equations to find prime numbers up to a limit.

*Euler-Mascheroni Constant*
The Euler-Mascheroni constant suggests that a partial harmonic series is approximately a logarithmic function. In code, it means that an harmonic time complexity should be represented as $\Theta(log n)$.

Sources:

Weisstein, Eric W. "Prime Number." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/PrimeNumber.html

Analysis of Different Methods to find Prime Number in Python. Rishabh Bansal. GeekforGeeks.
https://www.geeksforgeeks.org/analysis-different-methods-find-prime-number-python/

Sieve of Eratosthenes. Abhinav Priyadarshi. GeeksforGeeks.
https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Sieve of Sundaram to print all primes smaller than n. Anuj Rathore. GeeksforGeeks
https://www.geeksforgeeks.org/sieve-sundaram-print-primes-smaller-n/

How To Generate Big Prime Numbers - Miller-rabin
Antoine Prudhomme .
https://medium.com/@prudywsh/how-to-generate-big-prime-numbers-miller-rabin-49e6e6af32fb

Time Complexity: Sieve Of Eratosthenes
Chen Felix -
https://medium.com/@chenfelix/time-complexity-sieve-of-eratosthenes-fb0184da81d

StackExchange Math Community:
https://math.stackexchange.com/questions/306371/simple-proof-of-showing-the-harmonic-number-h-n-theta-log-n

## Step 3: Creative solutions

To find the creative solutions a technique called “Brainstorming” will be used.

Alternative #1
The programs looks for a list of prime numbers on the internet or on an external file and uses it to identify prime numbers in the program

Alternative #2
Implement an algorithm based on the sieve of Erathostenes (described in Step 2 of this document)

Alternative #3
Implement an algorithm based on the sieve of Sundaram (described in Step 2 of this document)

Alternative #4
Use the naive algorithm (described in Step 2 of this document) to find prime numbers.

Alternative #5
Import a library or an API to the program that already has a built-in method for checking whether a number is prime or has a method that can generate a list of prime numbers below a given integer.

Alternative #6
Use a probabilistic algorithm to quickly generate large prime numbers, taking a preselected random number of the desired length, and applying a Femant test and Miller-Rabin primality test (This is the standard way to generate large prime number on cryptography).

Alternative #7
Use an algorithm based on the sieve of Atkin (described in Step 2 of this document)

Alternative #8
Use a modified version of the naive algorithm (described in Step 2 of this document) that skips even numbers (other than 2) and numbers that end in 5 (other than 5).

## Step 4: Transition from ideas to Preliminary Design

Alternative #1
This alternative is not very secure because these external files might be manipulated or tampered with by a third party. Therefore we must discard this alternative unfortunately.

Alternative #2
This algorithm might be a very good alternative to use in the program because the algorithm is:

- Safe
- Reliable
- Relatively fast

Alternative #3
it's a good alternative since the algorithm guarantees always finding the prime numbers up to n. The algorithm is also secure and reliable.

Alternative #4
This algorithm is easy to understand and implement. Furthermore, it always allows finding all the primes number up to n for these reasons this is a really good alternative.

Alternative #5
The algorithm that the library has might not work properly, we would have to test the algorithm to be sure that the algorithm actually works. This could take too much time, for this, we considered this alternative as unviable.

Alternative #6

This alternative might not be viable because:

- These kinds of algorithms are too difficult for junior programmers
- The numbers given aren’t always primes
- These algorithms are only ideal when finding large prime numbers
- It wouldn't always work for finding all primes numbers up to n (due to its probabilistic property)

Alternative #7

Using the sieve of Atkin formulas to find prime numbers is a good solution for our problem and its even faster than the sieve of Erathostenes. However , the algorithm is too complex for junior programmers who do not have too much experience in the field of cryptography. In addition, the code is difficult to read because it might get disorganized. We must therefore discard this alternative.

Alternative #8

This algorithm, in a similar manner to alternative #2, might be a very good alternative to use in the program because the algorithm is:

- Safe
- Reliable
- Relatively fast

## Step 5: Evaluation and selection of the best solution

Criterion A: The speed of the algorithm.

- [3] The solution is the fastest discovered in history
- [2] The solution is relatively fast
- [1] The solution is extremely slow

Criterion B: The solution is reliable.

- [3] - The solution always identifies the prime numbers
- [2] - The solution sometimes identifies the prime numbers
- [1] - The solution never identifies the prime numbers

Criterion C: Difficulty of implementation. The algorithm is:

- [3] - The algorithm is easy to understand and implement
- [2] - The algorithm is moderately hard to understand and implement
- [1] - The algorithm is very hard to understand and implements

Criterion D: The security and possibility of exploitation of the algorithm.

- [3] - The algorithm is secure and nearly impossible to exploit
- [2] - The algorithm is relatively secure and cannot be easily exploited
- [1] - The algorithm is insecure and buggy/easy to exploit

| Algorithm | Criterion A | Criterion B | Criterion C | Criterion D | Total |
|---|---|---|---|---|---|
| Sieve of Eratosthenes | 2 | 3 | 3 | 3 | 11 |
| Sieve of Sundaram | 2 | 3 | 2 | 3 | 10 |
| Naive algorithm | 2 | 3 | 3 | 3 | 11 |
| Modified naive algorithm (Odd-5) | 2 | 3 | 3 | 3 | 11 |

Justification:

The sieve of Eratosthenes is reliable, relatively fast, not very difficult to understand and implement, and does not have any apparent bugs or security concerns.

The naive  algorithm and the modified naive algorithm are also reliable, relatively fast, not very difficult to understand and implement, and do not have any apparent bugs or security concerns.

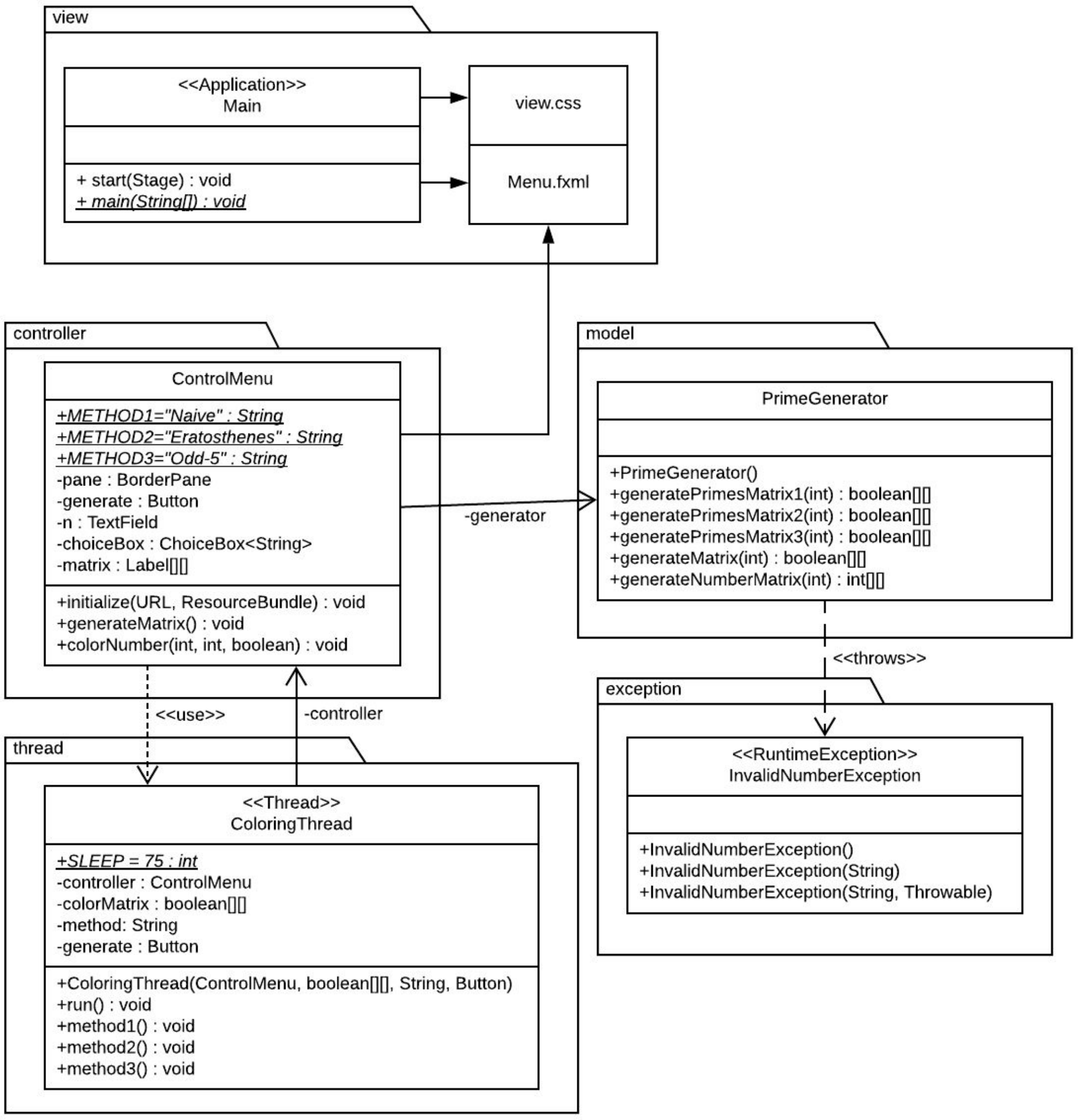
The sieve of Sundaram is also reliable, fast and secure, but it might be difficult to understand and implement.

Final decision:

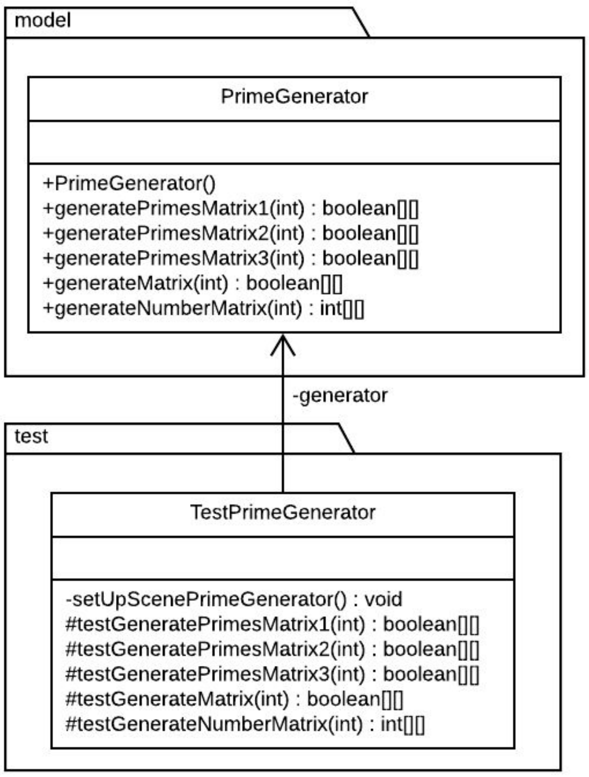
It is easy to see that all of the algorithms are good options, however we must only choose three. Thus we choose: The sieve of Eratosthenes, naive algorithm, and modified naive algorithm (henceforth named Odd-5).

## Step 6: Preparing the report and specifications

**Class Diagram:**



view
<<Application>>
Main
+ start(Stage) : void
+ main(String[]) : void
view.css
Menu.fxml
controller
ControlMenu
+METHOD1="Naive" : String
+METHOD2="Eratosthenes" : String
+METHOD3="Odd-5" : String
-pane : BorderPane
-generate : Button
-n : TextField
-choiceBox : ChoiceBox<String>
-matrix : Label[][]
+initialize(URL, ResourceBundle) : void
+generateMatrix() : void
+colorNumber(int, int, boolean) : void
-generator
model
PrimeGenerator
+PrimeGenerator()
+generatePrimesMatrix1(int) : boolean[][]
+generatePrimesMatrix2(int) : boolean[][]
+generatePrimesMatrix3(int) : boolean[][]
+generateMatrix(int) : boolean[][]
+generateNumberMatrix(int) : int[][]
<<throws>>
exception
<<RuntimeException>>
InvalidNumberException
+InvalidNumberException()
+InvalidNumberException(String)
+InvalidNumberException(String, Throwable)
<<use>>
-controller
thread
<<Thread>>
ColoringThread
+SLEEP = 75 : int
-controller : ControlMenu
-colorMatrix : boolean[][]
-method: String
-generate : Button
+ColoringThread(ControlMenu, boolean[][], String, Button)
+run() : void
+method1() : void
+method2() : void
+method3() : void


model
PrimeGenerator
+PrimeGenerator()
+generatePrimesMatrix1(int) : boolean[][]
+generatePrimesMatrix2(int) : boolean[][]
+generatePrimesMatrix3(int) : boolean[][]
+generateMatrix(int) : boolean[][]
+generateNumberMatrix(int) : int[][]
-generator
test
TestPrimeGenerator
-setUpScenePrimeGenerator() : void
#testGeneratePrimesMatrix1(int) : boolean[][]
#testGeneratePrimesMatrix2(int) : boolean[][]
#testGeneratePrimesMatrix3(int) : boolean[][]
#testGenerateMatrix(int) : boolean[][]
#testGenerateNumberMatrix(int) : int[][]

**Naive Algorithm:**

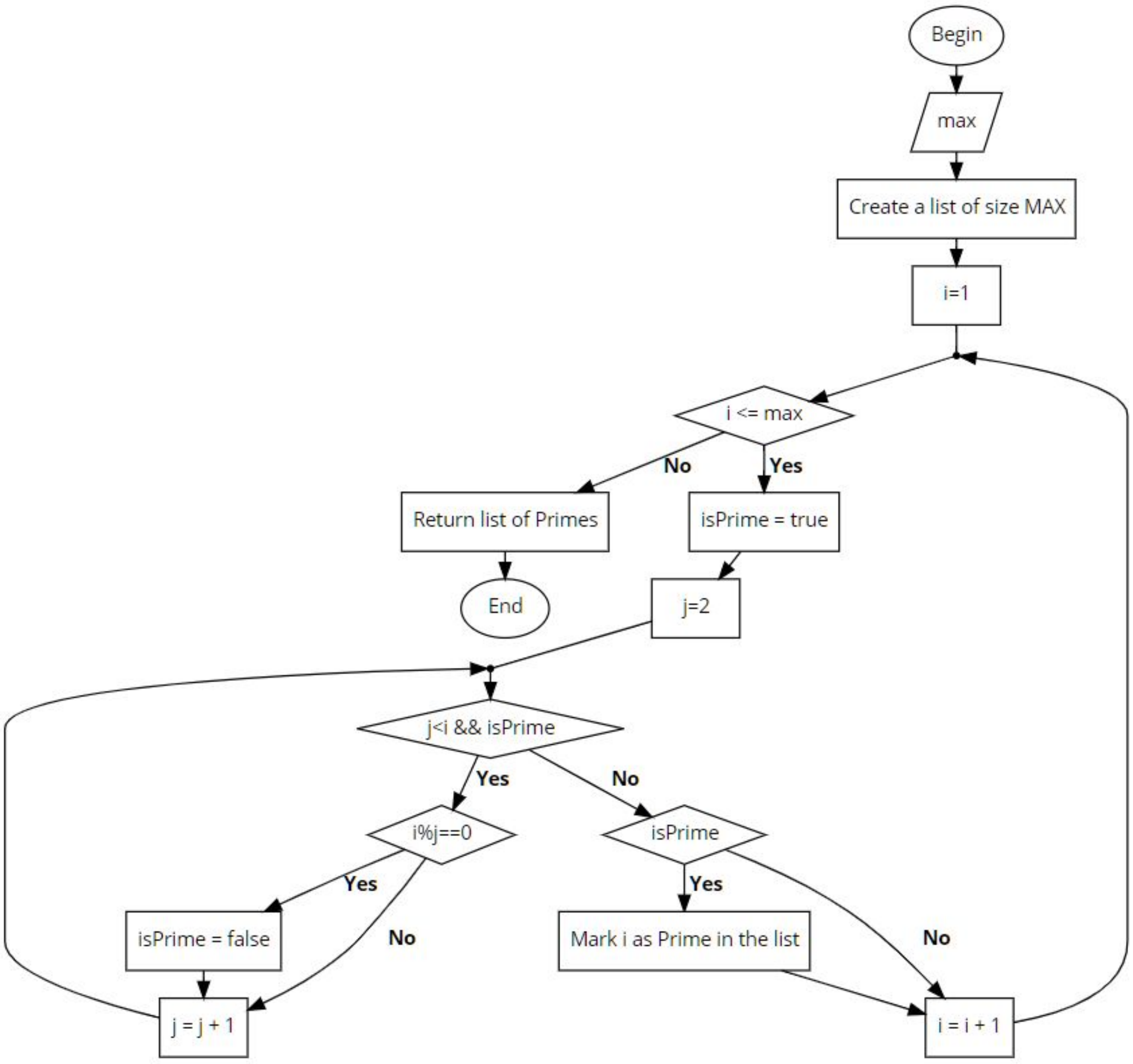
- Pseudocode:

```
Begin
Input: max

Create a matrix of size max
for (i ← 1; i <= max; i ←  i + 1)
        isPrime ←  true
        for ( j ←  2; (j<i) ^ (isPrime); j ←  j + 1)
                if (i%j==0)
                        isPrime ←  false
        if (isPrime)
                Mark i as Prime in the matrix

Output: matrix of Primes
End
```

- Flow diagram:



- <u>Time Complexity:</u>

| # | Algorithm | Cost | Times |
|---|---|---|---|
| 1. | Create a matrix of size max | $C_1$ | $5$ |
| 2. | for (i ← 1; i <= max; i ← i + 1) | $C_2$ | $n+1$ |
| 3. | isPrime ← true | $C_3$ | $n$ |
| 4. | for ( j ← 2; (j<i) ^ (isPrime); j ← j + 1) | $C_4$ | $\frac{n*(n-1)}{2}+1$ |
| 5. | if (i%j==0) | $C_5$ | $\frac{n*(n-1)}{2}+1-n$ |
| 6. | isPrime ← false | $C_6$ | $\frac{n*(n-1)}{2}+1-n$ |
| 7. | if (isPrime) | $C_7$ | $n$ |

| | | | |
|---|---|---|---|
| 8. | calculate row of i | $C_8$ | $n$ |
| 9. | calculate column of i | $C_9$ | $n$ |
| 10. | Mark i as Prime in the matrix | $C_{10}$ | $n$ |
| 11. | return matrix of Primes | $C_{11}$ | 1 |

$5(C_1) + n(C_2) + (C_2) + n(C_3) + \frac{1}{2}n^2(C_4) - \frac{1}{2}n(C_4) + (C_4) + \frac{1}{2}n^2(C_5) - \frac{1}{2}n(C_5) + (C_5) - n(C_5)$
$+ \frac{1}{2}n^2(C_6) - \frac{1}{2}n(C_6) + (C_6) - n(C_6) + n(C_7) + n(C_8) + n(C_9) + n(C_{10}) + (C_{11})$

$T(n) = n^2(\frac{1}{2}C_4 + \frac{1}{2}C_5 + \frac{1}{2}C_6) + n(C_2 + C_3 - \frac{1}{2}C_4 - \frac{3}{2}C_5 - \frac{3}{2}C_6 + C_7 + C_8 + C_9 + C_{10}) + (5C_1$
$+ C_2 + C_4 + C_5 + C_6 + C_{11})$

$\frac{3}{2}n^2 + \frac{5}{2}n + 10 \leq c\,n^2$
$\frac{3}{2}n^2 + \frac{5}{2}n + 10 \leq \frac{3}{2}n^2 + \frac{5}{2}n^2 + 10n^2 \leq c\,n^2$
$14n^2 \leq c\,n^2$
$14 \leq c$

$14 = c$ $\quad\quad 1 = n_0$
$O(n^2)$

- <u>Space Complexity:</u>

| | |
|---|---|
| **Input**<br>*max* | 1 |
| **Auxiliary**<br>*i*<br>*isPrime*<br>*j*<br>*row*<br>*column* | <br>1<br>1<br>1<br>1<br>1 |
| **Output**<br>*primesMatrix* | $n^2$ |
| **Total** | $n^2 + 6$ |

$n^2 + 6 <= c\,n^2$
$n^2 + 6 <= n^2 + 6n^2 <= c\,n^2$
$7n^2 <= c\,n^2$
$7 <= c$

$7 = c$ $\quad\quad 1 = n_0$
$O(n^2)$

**<u>Odd-5 Algorithm:</u>**

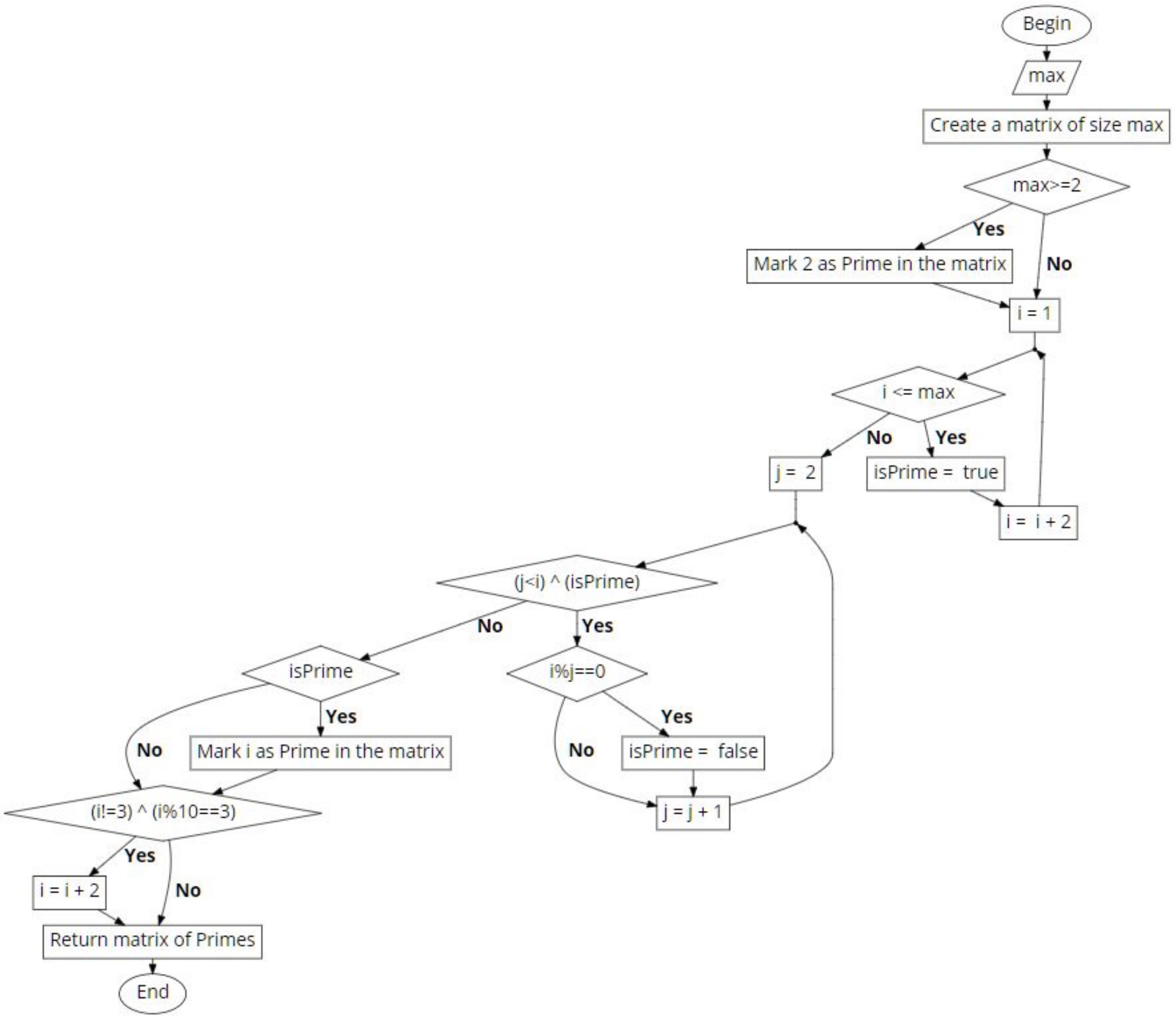
- Pseudocode:

```
Begin
Input: max

Create a matrix of size max
if(max>=2)
        Mark 2 as Prime in the matrix
for (i ← 1; i <= max; i ←  i + 2)
        isPrime ←  true
        for ( j ←  2; (j<i) ^ (isPrime); j ←  j + 1)
                if (i%j==0)
                        isPrime ←  false
        if (isPrime)
                Mark i as Prime in the matrix
        if((i!=3) ^ (i%10==3))
                i ← i + 2

Output: matrix of Primes
End
```

- Flow diagram:



- <u>Time Complexity:</u>

| # | Algorithm | Cost | Times |
|---|---|---|---|
| 1. | Create a matrix of size max | $C_1$ | 5 |
| 2. | if(max>=2) | $C_2$ | 1 |
| 3. | Mark 2 as Prime in the matrix | $C_3$ | 1 |
| 4. | for (i ← 1; i <= max; i ← i + 2) | $C_4$ | $\frac{2}{5}n + 1$ |
| 5. | isPrime ← true | $C_5$ | $\frac{2}{5}n$ |
| 6. | for ( j ← 2; (j<i) ^ (isPrime); j ← j + 1) | $C_6$ | $\frac{\frac{2}{5}n(\frac{2}{5}n-1)}{2} + \frac{2}{5}$ |

| | | | |
|---|---|---|---|
| 7. | if (i%j==0) | $C_7$ | $\frac{\frac{2}{5}n(\frac{2}{5}n-1)}{2} + \frac{2}{5} - \frac{2}{5}n$ |
| 8. | isPrime ← false | $C_8$ | $\frac{\frac{2}{5}n(\frac{2}{5}n-1)}{2} + \frac{2}{5} - \frac{2}{5}n$ |
| 9. | if (isPrime) | $C_9$ | $\frac{2}{5}n$ |
| 10. | calculate row of i | $C_{10}$ | $\frac{2}{5}n$ |
| 11. | calculate column of i | $C_{11}$ | $\frac{2}{5}n$ |
| 12. | Mark i as Prime in the matrix | $C_{12}$ | $\frac{2}{5}n$ |
| 13. | if((i!=3) ^ (i%10==3)) | $C_{13}$ | $\frac{2}{5}n$ |
| 14. | i ← i + 2 | $C_{14}$ | $\frac{2}{5}n$ |
| 15. | return matrix of Primes | $C_{15}$ | 1 |

$$5(C_1) + (C_2) + (C_3) + \frac{2}{5}n(C_4) + (C_4) + \frac{2}{5}n(C_5) + \frac{2}{25}n^2(C_6) - \frac{3}{5}n(C_6) + \frac{2}{5}(C_6) + \frac{2}{25}n^2(C_7) - n(C_7) + \frac{2}{5}(C_7)$$
$$+ \frac{2}{25}n^2(C_8) - n(C_8) + \frac{2}{5}(C_8) + \frac{2}{5}n(C_9) + \frac{2}{5}n(C_{10}) + \frac{2}{5}n(C_{11}) + \frac{2}{5}n(C_{12}) + \frac{2}{5}n(C_{13}) + \frac{2}{5}n(C_{14}) + (C_{15})$$

$$T(n) = n^2(\frac{2}{25}C_6 + \frac{2}{25}C_7 + \frac{2}{25}C_8) + n(\frac{2}{5}C_4 + \frac{2}{5}C_5 - \frac{3}{5}C_6 - C_7 - C_8 + \frac{2}{5}C_9 + \frac{2}{5}C_{10} + \frac{2}{5}C_{11} + \frac{2}{5}C_{12} + \frac{2}{5}C_{13}$$
$$+ \frac{2}{5}C_{14}) + (5C_1 + C_2 + C_3 + C_4 + \frac{2}{5}C_6 + \frac{2}{5}C_7 + \frac{2}{5}C_8 + C_{15})$$

$$\frac{6}{25}n^2 + \frac{3}{5}n + \frac{51}{5} \le c\,n^2$$
$$\frac{6}{25}n^2 + \frac{3}{5}n + \frac{51}{5} \le \frac{6}{25}n^2 + \frac{3}{5}n^2 + \frac{51}{5}n^2 \le c\,n^2$$
$$\frac{276}{25}n^2 \le c\,n^2$$
$$\frac{276}{25} \le c$$

$\frac{276}{25} = c$ $\qquad\qquad$ $1 = n_0$
$O(n^2)$

- <u>Space Complexity:</u>

| | |
|---|---|
| **Input**<br>*max* | <br>1 |
| **Auxiliary**<br>*i*<br>*isPrime*<br>*j*<br>*row* | <br>1<br>1<br>1<br>1 |

|  |  |
|---|---|
| *column* | 1 |
| **Output** *primesMatrix* | $n^2$ |
| **Total** | $n^2 + 6$ |

$n^2 + 6 <= c\,n^2$
$n^2 + 6 <= n^2 + 6n^2 <= c\,n^2$
$7n^2 <= c\,n^2$
$7 <= c$

$7 = c$ $1 = n_0$
$O(n^2)$

**Eratosthenes Algorithm:**

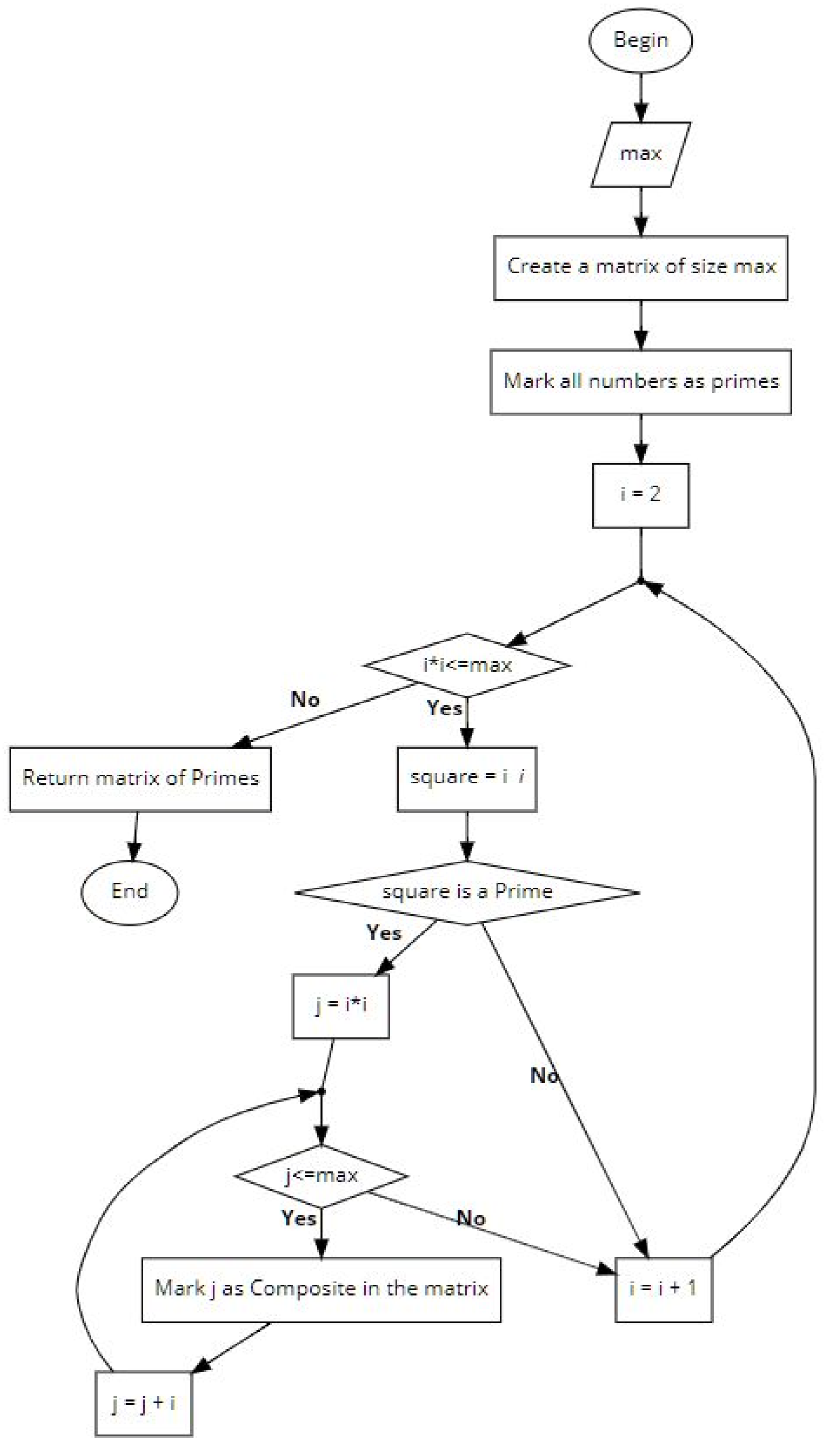
- Pseudocode:

```
Begin
Input: max

Create a matrix of size max
for(row in matrix)
        Mark all numbers as primes in row
for(i← 2; (i*i)<=max; i← i + 1)
        square ← i * i
        if(square is a Prime)
                for(j ← i*i; j<=max ; j ← j + i)
                        Mark j as Composite in the matrix

Output: matrix of Primes
End
```

- Flow diagram:



- <u>Time Complexity:</u>

| **#** | **Algorithm** | **Cost** | **Times** |
|---|---|---|---|
| 1. | Create a matrix of size max | $C_1$ | 5 |

| | | | |
|---|---|---|---|
| 2. | for(row in matrix) | $C_2$ | $\sqrt{n} + 1$ |
| 3. | Mark all numbers as primes in row | $C_3$ | $\sqrt{n}$ |
| 4. | for(i← 2; (i*i)<=max; i← i + 1) | $C_4$ | $\sqrt{n}$ |
| 5. | square ← i * i | $C_5$ | $\sqrt{n} - 1$ |
| 6. | calculate row of square | $C_6$ | $\sqrt{n} - 1$ |
| 7. | calculate column of square | $C_7$ | $\sqrt{n} - 1$ |
| 8. | if(square is a Prime) | $C_8$ | $\sqrt{n} - 1$ |
| 9. | for(j ← i*i; j<=max ; j ← j + i) | $C_9$ | $\sum_{i=2}^{\sqrt{n}} \frac{n}{i} + (2 - i) \equiv$ $nlog(log(n)) + (2(\sqrt{n} - 1)) -$ $\frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1$ |
| 10. | calculate row of j | $C_{10}$ | $nlog(log(n)) + (\sqrt{n} - 1) -$ $\frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1$ |
| 11. | calculate row of j | $C_{11}$ | $nlog(log(n)) + (\sqrt{n} - 1) -$ $\frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1$ |
| 12. | Mark j as Composite in the matrix | $C_{12}$ | $nlog(log(n)) + (\sqrt{n} - 1) -$ $\frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1$ |
| 13. | return matrix of Primes | $C_{13}$ | $1$ |

$$C_1 5 + C_2(\sqrt{n} + 1) + C_3\sqrt{n} + C_4\sqrt{n} + C_6(\sqrt{n} - 1) + C_7(\sqrt{n} - 1) + C_8(\sqrt{n} - 1)$$
$$+ C_9(nlog(log(n)) + (2(\sqrt{n} - 1)) - \frac{(\sqrt{n})(\sqrt{n}+1)}{2}) + C_{10}(nlog(log(n)) + (\sqrt{n} - 1) - \frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1)$$
$$+ C_{11}(nlog(log(n)) + (\sqrt{n} - 1) - \frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1) + C_{12}(nlog(log(n)) + (\sqrt{n} - 1) - \frac{(\sqrt{n})(\sqrt{n}+1)}{2} + 1) + C_{13}$$

$$T(n) = nlog(log(n))(C_9 + C_{10} + C_{11} + C_{12}) + n(-\tfrac{1}{2}C_9 - \tfrac{1}{2}C_{10} - \tfrac{1}{2}C_{11} - \tfrac{1}{2}C_{12})$$
$$+ \sqrt{n}\,(C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8 + \tfrac{3}{2}C_9 + \tfrac{1}{2}C_{10} + \tfrac{1}{2}C_{11} + \tfrac{1}{2}C_{12})$$
$$+ (5C_1 + C_2 - C_6 - C_7 - C_8 - 2C_9 - C_{10} - C_{11} - C_{12} + C_{13})$$

$$4nlog(log(n)) - 2n + 10\sqrt{n} + -1 \le c\, nlog(log(n))$$
$$4nlog(log(n)) + 10\sqrt{n} \le c\, nlog(log(n))$$
$$4nlog(log(n)) + 10nlog(log(n)) \le c\, nlog(log(n))$$
$$14nlog(log(n)) \le c\, nlog(log(n))$$

$$14 \le c$$

$14 = c$ $\qquad 1 = n_0$

$O(nlog(log(n)))$

- Space Complexity:

| | |
|---|---|
| **Input**<br>*max* | 1 |
| **Auxiliary**<br>*primesRow*<br>*i*<br>*square*<br>*row*<br>*column*<br>*j* | 1<br>1<br>1<br>1<br>1<br>1 |
| **Output**<br>*primesMatrix* | $n^2$ |
| **Total** | $n^2 + 7$ |

$n^2 + 7 <= c\,n^2$
$n^2 + 7 <= n^2 + 7n^2 <= c\,n^2$
$8n^2 <= c\,n^2$
$8 <= c$

$8 = c$ $\qquad 1 = n_0$
$O(n^2)$

**Unit Tests Design:**

| **Class** | **Method** | **Scene** | **Input** | **Result** |
|---|---|---|---|---|
| PrimeGenerator | generatePrimes Matrix1 | A prime generator. | Generate the primes between 1 to -1. | Impossible to perform this action. |
| | | | Generate the primes between 1 to 1. | Return the matrix with the correct highlighted primes. |
| | | | Generate the primes between 1 to 4. | Return the matrix with the correct highlighted |

| | | | | |
|---|---|---|---|---|
| | | | | primes. |
| | | | Generate the primes between 1 to 8. | Return the matrix with the correct highlighted primes. |
| PrimeGenerator | generatePrimesMatrix2 | A prime generator. | Generate the primes between 1 to -1. | Impossible to perform this action. |
| | | | Generate the primes between 1 to 1. | Return the matrix with the correct highlighted primes. |
| | | | Generate the primes between 1 to 4. | Return the matrix with the correct highlighted primes. |
| | | | Generate the primes between 1 to 8. | Return the matrix with the correct highlighted primes. |
| PrimeGenerator | generatePrimesMatrix3 | A prime generator. | Generate the primes between 1 to -1. | Impossible to perform this action. |
| | | | Generate the primes between 1 to 1. | Return the matrix with the correct highlighted primes. |
| | | | Generate the primes between 1 to 4. | Return the matrix with the correct highlighted primes. |
| | | | Generate the primes between 1 to 8. | Return the matrix with the correct highlighted primes. |

| | | | | |
|---|---|---|---|---|
| PrimeGenerator | generateMatrix | A prime generator. | Generate a matrix with -1 cells. | Impossible to perform this action. |
| | | | Generate a matrix with 1 cells. | Return a 1x1 matrix. |
| | | | Generate a matrix with 4 cells. | Return a 2x2 matrix. |
| | | | Generate a matrix with 8 cells. | Return a 3x3 matrix. |
| PrimeGenerator | generateNumberMatrix | A prime generator. | Generate a matrix with numbers from 1 to -1. | Impossible to perform this action. |
| | | | Generate a matrix with numbers from 1 to 1. | Return the matrix with all the numbers in the appropriate position. |
| | | | Generate a matrix with numbers from 1 to 4. | Return the matrix with all the numbers in the appropriate position. |
| | | | Generate a matrix with numbers from 1 to 8. | Return the matrix with all the numbers in the appropriate position. |

**Github:** https://github.com/Esarac/Primes