صلاح الدين أحمد محمد السيد الطنيحي	19015854
جمال عبد الحميد ناصف نويصر	19015550

Problem statement:

- The Sieve of Eratosthenes algorithm for finding all prime numbers up to any given limit

Pseudo code:

- Take to inputs the original number & 2, 3, 5, 7
- ComputeEratosthenes():
  - For i=2 to Number:
    - Remainder = i% divisor
    - If(remainder == 0)
      - notPrimeNumber.add(i)
  - removeNotPrimeNumber:
    - for i=0 to len(notPrimeNumber):
      - originalList .remove(notPrimeNumber[i])

➢ Sample runs:

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
1
Enter number: 400
[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, 121, 127, 131, 137, 139, 14
3, 149, 151, 157, 163, 167, 169, 173, 179, 181, 187, 191, 193, 197, 199, 209, 211, 221, 223, 227, 229, 233, 239, 241, 247, 251, 253, 257, 263, 269, 2
71, 277, 281, 283, 289, 293, 299, 307, 311, 313, 317, 319, 323, 331, 337, 341, 347, 349, 353, 359, 361, 367, 373, 377, 379, 383, 389, 391, 397]
PS F:\First Semester\Discrete\python\2> & "C:/Users/Gamal Abdul Hameed/AppData/Local/Programs/Python/Python39/python.exe" "f:/First Semester/Discrete
/python/2/Aaaaaaaaaaaaaaaaa.py"
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
1
Enter number: 1000
[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, 121, 127, 131, 137, 139, 14
3, 149, 151, 157, 163, 167, 169, 173, 179, 181, 187, 191, 193, 197, 199, 209, 211, 221, 223, 227, 229, 233, 239, 241, 247, 251, 253, 257, 263, 269, 2
71, 277, 281, 283, 289, 293, 299, 307, 311, 313, 317, 319, 323, 331, 337, 341, 347, 349, 353, 359, 361, 367, 373, 377, 379, 383, 389, 391, 397, 401,
403, 407, 409, 419, 421, 431, 433, 437, 439, 443, 449, 451, 457, 461, 463, 467, 473, 479, 481, 487, 491, 493, 499, 503, 509, 517, 521, 523, 527, 529,
 533, 541, 547, 551, 557, 559, 563, 569, 571, 577, 583, 587, 589, 593, 599, 601, 607, 611, 613, 617, 619, 629, 631, 641, 643, 647, 649, 653, 659, 661
, 667, 671, 673, 677, 683, 689, 691, 697, 701, 703, 709, 713, 719, 727, 731, 733, 737, 739, 743, 751, 757, 761, 767, 769, 773, 779, 781, 787, 793, 79
7, 799, 803, 809, 811, 817, 821, 823, 827, 829, 839, 841, 851, 853, 857, 859, 863, 869, 871, 877, 881, 883, 887, 893, 899, 901, 907, 911, 913, 919, 9
23, 929, 937, 941, 943, 947, 949, 953, 961, 967, 971, 977, 979, 983, 989, 991, 997]
```

➢ Problem Statement:

- Trial Division algorithm for integer factorization

➢ Pseudo code:

- First get prime numbers under the square root of the specified Number using sieve of Eratosthenes algorithm.
- Trial Division class inherit sieve of Eratosthenes
- Counter=0
- While(counter <len(primeNumbers))
  - DivisionResult = specifiedNumber %primeNumbers[counter]
    - If(DivisionResult == 0)
      - Result.add(primeNumbers[counter])
      - inputNumber /= primeNumbers[counter]
    - else:
      - counter += 1

➢ Sample runs:

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
2
Enter number: 13468
[2, 2, 7, 13, 37]
PS F:\First Semester\Discrete\pyth
/python/2/Aaaaaaaaaaaaaaaaa.py"
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
2
Enter number: 637948
[2, 2, 43, 3709]
```

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
2
Enter number: 97
[97]
PS F:\First Semester\Discrete\pyth
/python/2/Aaaaaaaaaaaaaaaaa.py"
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
2
Enter number: 140
[2, 2, 5, 7]
```

➢ Problem statement:

- Chinese remainder theorem that takes as input m1, m2, m3, …., mn that are pairwise relatively prime and (a1, a2, …., an) and calculates x

➢ Pseudo code:

- ComputeCapital_M:
  - M = 1
  - For i=0 to numberofEquations-1:
    - M *= inputMatrix[i][1]
- Compute-Mi:
  - For i=0 to numberOfEquations-1
    - inputMatrix[i][2] = M/inputMatrix[i][1]
- chineseRemainderAlgorritm:
  - for i=0 to len(inputMatrix):

        - remainder = inputMatrix[i][2]/ inputMatrix[i][1]
        - if(remainder == 1):
            - inputMatrix[i][3] = 1
        - else:
            - flag = True
            - counter = 2
            - while(flag):
                - temp = (counter*remainder)%inputMatrix[i][1]
                - if(temp == 1):
                    - flag = False
                    - inputMatrix[i][3] = counter
                - counter += 1
- computeResult:
    - result = 0
    - for i=0 to numberOfEquations-1
        - result += inputMatrix[i][0]*inputMatrix[i][2]*inputMatrix[i][3]
    - print(result % M)

➢ Sample runs:

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
4
Number of Equations: 3
Equation form x≡b(mod m)
Params of Eqn  1 :
b: 2
m: 3
Params of Eqn  2 :
b: 3
m: 5
Params of Eqn  3 :
b: 2
m: 7
X = 23 + 105n
```

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
4
Number of Equations: 3
Equation form x≡b(mod m)
Params of Eqn  1 :
b: 1
m: 5
Params of Eqn  2 :
b: 2
m: 6
Params of Eqn  3 :
b: 3
m: 7
X = 206 + 210n
```

➢ Problem statement:

- The Extended Euclidean algorithm that finds the greatest common divisor d of two positive integers a and b. And the Bezout's coefficients s and t (d = s a + t b).
- List of lists

➢ Pseudo code:

```
function initializeVariables():
        A = list()  # 2D Array of the result table
        counter = 0  # Index of the row in the table
        INDEX = 0   # Index of the 1st column in the table (j)
        RJ = 1      # Index of the 2nd column in the table (r(j))
        RJ1 = 2     # Index of the 3rd column in the table (r(j + 1))
        QJ1 = 3     # Index of the 4th column in the table (q(j + 1))
        RJ2 = 4     # Index of the 5th column in the table (r(j + 2))
        SJ = 5      # Index of the 6th column in the table (s(j))
        TJ = 6      # Index of the 7th column in the table (t(j))
        NumberOfColumns = 7
        flag = True  # false if one number is multiple of the other
End initializeVariables

function takeInput():
        A.append(list(range(NumberOfColumns)))
        Scan first number (a)
        Scan second number (b)
End takeInput
```

```
# put the values of the first row
function setFirstRow():
        A[counter][INDEX] = counter  # Number of the row

        A[counter][RJ] = a  # r0 = a
        A[counter][RJ1] = b  # r1 = b

        # q(j + 1) = floor(a / b)
        A[counter][QJ1] = int(A[counter][RJ] / A[counter][RJ1])

        # r(j + 2) = a mod b
        A[counter][RJ2] = A[counter][RJ] % A[counter][RJ1]

        # Bezout's coefficients are 1 & 0
        A[counter][SJ] = 1
        A[counter][TJ] = 0
        remainder = A[counter][RJ2]  # the remainder
End setFirstRow

# Check if one number is multiple of the other
function is_B_factor_of_A():
        Begin if(remainder == 0):
                flag = False  # finish the program
                gcd = b  # The GCD of the two numbers
                # Add another row to get the Bezout's coefficients
                counter = counter + 1
                A.append(list(range(NumberOfColumns)))
                A[counter][INDEX] = counter
                A[counter][RJ] = ''
                A[counter][RJ1] = ''
                A[counter][QJ1] = ''
                A[counter][RJ2] = ''
```

```
            A[counter][SJ] = 0
            A[counter][TJ] = 1

            # Bezout's coefficients are 0 & 1
            s = 0
            t = 1
        End if
End is_B_factor_of_A

# put the values of the second row
function setSecondRow():
        # if the two numbers are not multiple of each other
        Begin if(remainder != 0):
            # Add a new row
            counter = counter + 1
            A.append(list(range(NumberOfColumns)))

            # Number of the row
            A[counter][INDEX] = counter

            # get the values of r(j) & r(j + 1) from the previous row
            A[counter][RJ] = A[counter - 1][RJ1]
            A[counter][RJ1] = A[counter - 1][RJ2]

            # q(j + 1) = floor(r(j) / r(j + 1))
            A[counter][QJ1] = A[counter][RJ] // A[counter][RJ1]

            # r(j + 2) = r(j) % r(j + 1)
            A[counter][RJ2] = A[counter][RJ] % A[counter][RJ1]

            # Bezout's coefficients are 0 & 1
            A[counter][SJ] = 0
```

```
        A[counter][TJ] = 1
        remainder = A[counter][RJ2]  # the remainder

    End if

End setSecondRow


# put the values of the other rows

function setOtherRows():

    # the same as the second row except the Bezout's coefficients

    Begin while(remainder != 0):

        counter = counter + 1

        A.append(list(range(NumberOfColumns)))

        A[counter][INDEX] = counter

        A[counter][RJ] = A[counter - 1][RJ1]

        A[counter][RJ1] = A[counter - 1][RJ2]

        A[counter][QJ1] = A[counter][RJ] / A[counter][RJ1]

        A[counter][RJ2] = A[counter][RJ] mod A[counter][RJ1]


        # Bezout's coefficients

        # s(j) = s(j - 2) - q(j - 1) * s(j - 1)

        A[counter][SJ] = A[counter - 2][SJ] - A[counter - 2][QJ1] *
        A[counter - 1][SJ]


        # t(j) = t(j − 2) − q(j − 1) * t(j − 1)
```

```
            A[counter][TJ] = A[counter - 2][TJ] - A[counter - 2][QJ1] *

            A[counter - 1][TJ]


            remainder = A[counter][RJ2]  # the remainder

        End while

    End setOtherRows


    function getBezoutCoffiecients():

    Begin if(flag == True):

        gcd = A[counter][RJ1]  # GCD of the two numbers


        # Add another row to get the Bezout's coefficients

        counter = counter + 1

        A.append(list(range(NumberOfColumns)))

        A[counter][INDEX] = counter

        A[counter][RJ] = ''

        A[counter][RJ1] = ''

        A[counter][QJ1] = ''

        A[counter][RJ2] = ''

        A[counter][SJ] = A[counter - 2][SJ] - A[counter - 2][QJ1] *
        A[counter - 1][SJ]

        A[counter][TJ] = A[counter - 2][TJ] - A[counter - 2][QJ1] *
```

```
        A[counter - 1][TJ]


        # Bezout’s coefficients
        s = A[counter][SJ]
        t = A[counter][TJ]
    function printResults():
        print("GCD =", gcd)
        # GCD = s * a + t * b
        print("Bezout’s coefficients:")
        print(str(gcd) + " = " + str(s) + " * " + str(a) + " + " + str(t) + " * " + str(b))
        print(A)
    End printResults
```

➢ Sample runs:

```
Enter the First Number: 252
Enter the Second Number: 198
GCD = 18
Bezout's coefficients:
18 = 4 * 252 + -5 * 198
+-----+------+--------+--------+--------+------+------+
|   j | r_j  | r_j+1  | q_j+1  | r_j+2  |  s_j |  t_j |
+=====+======+========+========+========+======+======+
|   0 | 252  | 198    | 1      | 54     |    1 |    0 |
+-----+------+--------+--------+--------+------+------+
|   1 | 198  | 54     | 3      | 36     |    0 |    1 |
+-----+------+--------+--------+--------+------+------+
|   2 | 54   | 36     | 1      | 18     |    1 |   -1 |
+-----+------+--------+--------+--------+------+------+
|   3 | 36   | 18     | 2      | 0      |   -3 |    4 |
+-----+------+--------+--------+--------+------+------+
|   4 |      |        |        |        |    4 |   -5 |
+-----+------+--------+--------+--------+------+------+
```

```
Enter the First Number: 662
Enter the Second Number: 414
GCD = 2
Bezout's coefficients:
2 = -5 * 662 + 8 * 414
+-----+------+--------+--------+--------+------+------+
|   j | r_j  | r_j+1  | q_j+1  | r_j+2  |  s_j |  t_j |
+=====+======+========+========+========+======+======+
|   0 | 662  | 414    | 1      | 248    |    1 |    0 |
+-----+------+--------+--------+--------+------+------+
|   1 | 414  | 248    | 1      | 166    |    0 |    1 |
+-----+------+--------+--------+--------+------+------+
|   2 | 248  | 166    | 1      | 82     |    1 |   -1 |
+-----+------+--------+--------+--------+------+------+
|   3 | 166  | 82     | 2      | 2      |   -1 |    2 |
+-----+------+--------+--------+--------+------+------+
|   4 | 82   | 2      | 41     | 0      |    2 |   -3 |
+-----+------+--------+--------+--------+------+------+
|   5 |      |        |        |        |   -5 |    8 |
+-----+------+--------+--------+--------+------+------+
```

➢ Problem statement:

- Miller's Test.

➢ Pseudo code:

```
def takeInput():
        scan(n)
        scan (ITERATIONS)
End takeInput

# (x ^ y) % p
function modularPower(x, y):
        ans = 1
        x = x mod n
        Begin while (y > 0):
                Begin if (y & 1):  # If y is odd
                        ans = (ans * x) mod n
                End if

                y = y / 2
                x = (x * x) mod n
        End while
        return ans
End modularPower

# returns false if n is composite
# and true if n is probably prime
function MillerTest():
        # n - 1 = t * (2 ^ s)
        # s >= 0, t > 0 & t is odd

        # Choose a random base in [2, n-2]
```

```
        b = 2 + randint(1, n - 4)

        # x = (b ^ t) % n
        x = function modularPower(b, t)

        # n is probably prime if
        # (b ^ t) mod n = 1 or
        # (b ^ t) mod n = (n - 1)
        Begin if (x == 1 or x == n - 1):
                return True
        End if

        Begin while (t != n - 1):
                x = (x * x) mod n
                t = t * 2

                # n is probably prime if
                # (x ^ 2) % n = (n - 1)
                Begin if (x == n - 1):
                        return True
                End if

                Begin if (x == 1):
                        return False
                End if
        End while

        return False
End MillerTest
```

```
function isPrime():

        Beginif (n == 2 or n == 3):

                return True  # 2 & 3 are Primes

        End if


        Begin if (n == 1 or (n mod 2 == 0)):

                return False  # 1 & even numbers greater than 2 are composites

        End if


        t = n - 1  # n is odd, so t must be even

        Begin while (t mod 2 == 0):  # we need t to be odd

                t = t / 2

        End while


        Begin for iteration = 0 to (ITERATIONS):

                Begin if (MillerTest() == False):

                        return False

                End if

        End for

        return True

End isPrime
```

```
function printResults(self):
    Begin if (self.isPrime()):
        print("Prime")
    End if
    Begin else:
        print("Composite")
    End else
End printResults
```

➢ Sample runs:

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
5
Enter the Number: 99991
Enter Number of Iterations: 10
Prime
```

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
5
Enter the Number: 17437
Enter Number of Iterations:
Composite
```

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
5
Enter the Number: 997
Enter Number of Iterations: 5
Prime
```

```
Enter the number of the Algorithm:
1- Eratosthenes
2- Trial Division
3- Extended_Euclidean
4- Chinese remainder
5- Miller 's Test
5
Enter the Number: 1991
Enter Number of Iterations: 8
Composite
```

Used Data Structures:

- List

Assumptions:

- **In the start, choose the number of the algorithm. If you enter a wrong number, the program will finish printing(“Error! Not supported!!”).**
- **If you enter a negative integer, the program will take the absolute.**
- **If you enter any type rather than integer, the program will finish, and a Value Error will appear.**
- **If you don’t enter a value for iterations, the default value will be 50.**
- **In the Chinese remainder:**
- **Enter the number of equations first.**
- **The input form is: x ≡ b (mod m)**