**Lab 3 Report**

**Course:** Discrete Mathematics

**Team members:**

**Student Name:** Ehab Yasser Mahmoud Mohamed Al Kerm

**ID:** 20010382

**Student Name:** Rowaina Abdelnaser Mohamed Sabra Abouzaid

**ID:** 20010605

**Report Content:**

* [**Problem statement**](#_Problem_statement:)
* [**Used data structures**](#_Used_data_structures:)
* [**Algorithms used**](#_Algorithms_used:)
* [**Code Snippets**](#_Code_Snippets:)
* [**Important assumptions and details**](#_Important_assumptions_and)
* **Design decisions**
* **Sample runs**

# 

# Problem statement:

# Question 1:

# Implement sieve of Eratosthenes algorithm for finding all prime numbers up to any given limit.

**Question 2:**

Implement Trial Division algorithm for integer factorization.

**Question 3**:

Implement the extended Euclidean algorithm that finds the greatest common divisor d of two positive integers a and b. In addition, it outputs Bezout’s coefficients s and t such that d = s a + t b

**Question 4:**

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

**Question 5:**

Implement Miller’s test (a probabilistic primality test).

# Used data structures:

**Problem 1:**

* Array of Booleans

**Problem 2:**

* Array of Booleans
* Vector of Integers

**Problem 3:**

* Only integers and pointers are used

**Problem 4:**

* Array of integers

**Problem 5:**

* All data types used are primitive

# Algorithms used:

**Problem 1:**

Diagram

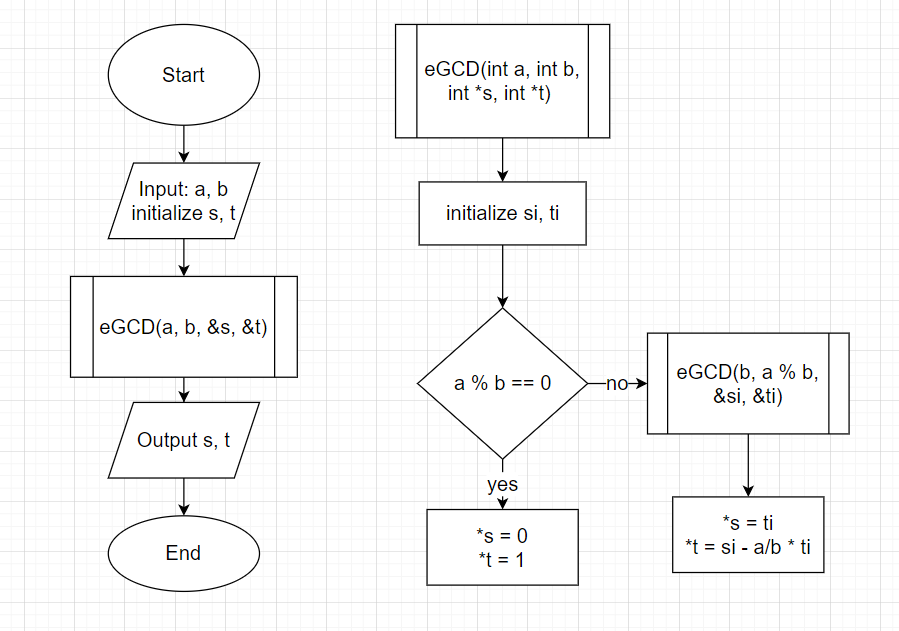
Description automatically generated

**Problem 2:**

Diagram

Description automatically generated

**Problem 3:**



**Problem 4:**

**Diagram

Description automatically generated**

# Problem 5:

Main function and miller test function algorithms:

# Diagram Description automatically generated

Single test and fast modular exponentiation functions algorithms

# Diagram Description automatically generated

# Code Snippets:

**Problem 1:**

Function implementing sieve algorithm

**Text

Description automatically generated**

**Problem 2:**

Function implementing trial division for finding prime factors of a number

**Text

Description automatically generated**

**Problem 3:**

Function finds Bezouts’ coefficients using extended Euclidean Algorithm

**Text

Description automatically generated**

**Problem 4:**

Function solves a system of congruences

**Text

Description automatically generated**

**Problem 5:**

Function finds modular exponentiation using fast modular exponentiation algorithm

**Text

Description automatically generated**

# Function tests a single base using the Miller-Rabin probabilistic primality test Text Description automatically generated

Function that tests whether n is probably prime or composite with the Miller-Rabin test.

Text

Description automatically generated

# Important assumptions and details:

**Problem 1:**

* n: the number we want to find prime numbers less than or equal to it.
* primes: the indexes of array ‘primes’ are the numbers starting from 2 till n.
* Assume that if the entry is true the index is a prime number and if false the index is composite number.
* The second loop with counter j starts from i\*i and not 2 as all numbers less than i\*i were handled in the previous iterations.

**Problem 2:**

* n: the number to be factorized.
* primes: is the same as assumed in problem 1.
* factors: vector of integers storing the factors to be printed.

**Problem 3:**

* **a:**
* **b:**
* **s:**
* **t:**
* **si:**
* **ti:**
* **//simple explanation**

**Problem 4:**

* M:
* m:
* a:
* Mtotal:
* ans
* n:
* s:

**Problem 5:**

* **n:**
* **k:**
* **a:**
* **res:**
* **power:**
* **bit:**
* **exp:**
* **base:**
* **mod:**
* Random number is generated to be at range 2<=a<=n-1 and is n is guaranteed to be at least 5 from millerTest function, so 2+rand() % (n-4) is the calculation to generate this number each iteration.
* //res conditions explanation

**Design decision:**

**Problem 1:**

* **Main function:**
  + Takes the bound to which the sieve algorithm will be implemented.
  + Declares the array of Boolean ’primes’
  + Calls the function ‘sieve(n, primes)’
  + Prints the prime numbers
* **Sieve function:**
  + Parameters: bound n and the array of Boolean.
  + Void function
  + Sets all entries of the mentioned array to true.
  + Sets composite numbered index to false.

**Problem 2:**

* **Main function:**
  + Takes number to be factorized as input from the user.
  + Declares the array of Booleans and vector of factors
  + Cals the function ‘trialDivisionFactorization(n, primes, factors)’
  + Prints the factors created.
* **trialDivisionFactorization function:**
  + Calls the function ‘sieve’ in problem 1, creating the array of Booleans to know the primality of numbers.
  + Test the divisibility of n by prime numbers formed in the previous step.

**Problem 3:**

* **Main function:**
  + Calls function eGCD(a, b, &s, &t)
  + Prints Bezout’s coefficients after returning from the last recursive call.
* **eGCD function:**
  + Recursive function :

base case: if(a%b == 0) then s points to 0 and t points to 1

//Explain your design

**Problem 4:**

* **Main function:**
  + Takes the number of equations n to be solved.
  + Declares the arrays of remainders, /////, and devisors.
  + Takes the values of remainders and devisors.
  + Calls the function ‘solveCongrunce(n, m, M, a)’.
* **solveCongrunces function:**
  + Calculates Mtotal.
  + Calculates the M for each equation.
  + Calculates the inverse of M[i] mod m[i]
  + Inverse is found by calling ‘eGCD(M[i], m[i], &s, &t)’ function, where s is the inverse.
  + Generates the final result.
  + Prints the final result.

**Problem 5:**

* **Main function:**
* **millerTest function:**
* **singleTest function:**
* **fastModularExponentiation function:**

**Sample runs and test cases:**

**Problem 1:**



**Problem 2:**

**Problem 3:**

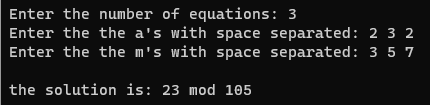


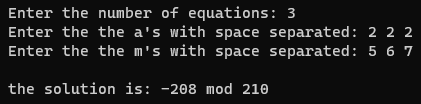


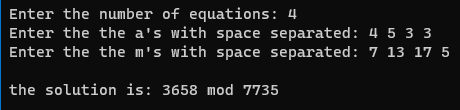




**Problem 4:**







**Problem 5:**







