Lab 1

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Write a program of Euclid's algorithm in C++

Euclid's algorithm:

1. Start
2. Input: Get a positive integer `a and b` from the user, where `a` is less than `b`.
3. `a mod b = R`.
4. Let `a = b` and `b = R`.
5. Repeat Steps 3 and 4 until `a mod b` is greater than 0.
6. `GCD = b`.
7. End

Implementation in C++

```
#include<stdio.h>
#include<math.h>
using namespace std;

int gcd(int a, int b) {
    if (a==0) return b;
    if (b % a == 0) return a;
    return gcd(b % a, a);
}

int main() {
    int a, b;
    if (a > b) {
        int temp = a;
        a = b;
        b = temp;
    }
    a = abs(a); b = abs(b);
    printf("Please␣Enter␣two␣numbers␣(a␣&␣b)␣:␣");
    scanf("%d␣%d", &a, &b);
    printf("gcd(%d,␣%d)␣=␣%d", a, b, gcd(a, b));
    return 0;
}
```

Time Complexity: $\mathbf{O}(\log \min(a, b))$

```
input : a = 16, b = 34
output : gcd(16, 34) = 2

input : a = 10, b = 121
output : gcd(10, 121) = 1
```

Problem: Write a program of Extended Euclid's algorithm in C++

The extended Euclidean algorithm is an extension of Euclid's algorithm for finding the greatest common divisor (GCD) of two numbers. It not only finds the GCD of two numbers but also finds integers x and y such that $ax + by = gcd(a, b)$.

The steps for the extended Euclidean algorithm are as follows:

1. Start
2. Input: two positive integers a and b.
3. If b is equal to 0, then a is the GCD and $x = 1$ and $y = 0$. The algorithm terminates. go to step 8.
4. If b is not equal to 0, we calculate the remainder r of a divided by b, and find the values of $x1$ and $y1$ such that $bx1 + (a - b * r/b) * y1 = gcd(b, r)$.
5. We set $x = y1$ and $y = x1 - floor(r/b) * y1$.
6. We replace a with b, b with r, and repeat steps 3 to 5 until b is equal to 0.
7. The final values of x and y are the solutions to the equation $ax + by = gcd(a, b)$.
8. End

Implementation in C++

```
1 #include<stdio.h>
2 using namespace std;
3
4 int gcd(int a, int b, int &x, int &y) {
5     if (a == 0) {
6         x = 1;
7         y = 0;
8         return b;
9     }
10    int x1, y1;
11    int d = gcd(b % a, a, x1, y1);
12    x = y1;
13    y = x1 - y1 * (b / a);
14    return d;
```

```
15 }
16
17 int main() {
18     int a, b, x, y;
19     bool f_a = false, f_b = false;
20     printf("Please␣Enter␣two␣numbers␣(a␣&␣b)␣:␣");
21     scanf("%d␣%d", &a, &b);
22
23     if (a < 0) f_a = true, a = -a;
24     if (b < 0) f_b = true, b = -b;
25
26     if (a>b) {
27         int temp = a;
28         a = b;
29         b = temp;
30     }
31
32     printf("gcd(%d,␣%d)␣=␣%d\n", a, b, gcd(a, b, x, y));
33
34     if (f_a) x = -x;
35     if (f_b) y = -y;
36     printf("x␣=␣%d,␣y␣=␣%d\n", y, x);
37
38     return 0;
39 }
```

Time Complexity: O($\log \min(a, b)$**)**

```
input : a = 16, b = 34
output : gcd(16, 34) = 2, x = -2, y = 1

input : a = 60, b = 48
output : gcd(60, 48) = gcd(48, 60) = 12, x = 1, y = -1

input : a = 10, b = 121
output : gcd(10, 121) = 1, x = -12, y = 1
```

Lab 2

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Write a program to check a given number is prime or not in C++

Algorithm to Check Prime Number:

1. Start
2. Input: Get a positive integer `input_number` from the user.
3. If `input_number` is less than or equal to 1, go to step 11.
4. If `input_number` is equal to 2, go to step 10.
5. If `input_number` is even, go to step 9.
6. Set `is_prime` to true.
7. For `i` from 3 to the square root of `input_number`:
   (a) If `input_number` is divisible evenly by `i` (`input_number % i == 0`), set `is_prime` to false and go to step 9.
   (b) Else set `i := i + 2`.
8. If `is_prime` is true, go to step 10.
9. Output: Print "The number is not prime" and go to step 12.
10. Output: Print "The number is prime" and go to step 12.
11. Output: Print "The number is not prime (since it's less than or equal to 1)" and go to step 12.
12. End

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

bool isPrime(int n) {
    if (n<=1) return 0;
    else if (n==2) return 1;
    else if ((n&1)==0) return 0;
    else {
        int i = 3;
        while (i*i <= n) {
            if (n % i == 0) return 0;
```

```
12              i += 2;
13          }
14          return true;
15      }
16 }
17
18 int main() {
19      int t = 10;
20      while (t--) {
21          int N;
22          cout<<"Please␣Enter␣an␣Number␣(N):␣";
23          cin>>N;
24
25          if (isPrime(N)) cout<<N<<"␣is␣a␣prime␣number."<<endl;
26          else cout<<N<< "␣is␣not␣a␣prime␣number."<<endl;
27      }
28      return 0;
29 }
30 }
```

Time Complexity: O($\sqrt{N}$)

```
input : 61
output : PRIME NUMBER

input : 33
output : NOT A PRIME NUMBER

input : 121
output : NOT A sPRIME NUMBER
```

Problem: Write a program to find prime factorization of a number in C++

Algorithm for Prime Factorization:

1. Start
2. Input: Get a positive integer n from the user.
3. While n is divisible by 2, print 2 and divide n by 2.
4. After step 3, n must be odd. Now start a loop from $i = 3$ to the square root of n. While i divides n, print i and divide n by i, increment i by 2, and continue.
5. If n is a prime number and is greater than 2, then n will not become 1 by the above two steps. So print n if it is greater than 2.
6. End

Example:
Let's find the prime factorization of the number 84 using the provided algorithm.

1. Start
2. Input: $n = 84$
3. While n is divisible by 2, print 2 and divide n by 2. $(84 \div 2 = 42)$
4. While n is divisible by 2, print 2 and divide n by 2. $(42 \div 2 = 21)$
5. While n is divisible by 3, print 3 and divide n by 3. $(21 \div 3 = 7)$
6. We have reached a prime number (7).
7. The prime factorization of 84 is $2 \times 2 \times 3 \times 7$.
8. So, $84 = 2^2 \times 3 \times 7$.
9. End

Implementation in C++

```
1
2 #include<bits/stdc++.h>
3 using namespace std;
4
5 bool isPrime(int n) {
6     if (n<=1) return 0;
7     else if (n==2) return 1;
8     else if ((n&1)==0) return 0;
9     else {
10         int i = 3;
11         while (i*i <= n) {
12             if (n % i == 0) return 0;
13             i += 2;
14         }
15         return true;
16     }
17 }
18
19 int main() {
20     map<int, int> factors;
21
22     int N;
23     cout<<"Please␣Enter␣an␣Number␣(N):␣";
24     cin>>N;
25     if (N==1) {
26         cout<<N<<"␣can␣not␣be␣factorized␣because␣it␣is␣neither␣a␣prime␣numbe
27         return 0;
28     }
```

```
29      if (isPrime(N)) {
30          cout<<N<<" is a prime number. Therefore can not be factorized."<<endl
31          return 0;
32      }
33
34      while ((N&1) == 0) {
35          factors[2]++;
36          N>>=1;
37      }
38
39      int i =3;
40      while (i <= N && !isPrime(N)) {
41          if (N % i == 0) {
42              factors[i]++;
43              N /= i;
44              continue;
45          }
46          i += 2;
47      }
48      if (N>1) factors[N] = 1;
49
50      string ans = "";
51      for (auto x: factors) {
52          ans += ("(" + to_string(x.first) + "^" + to_string(x.second) + ") * "
53      }
54
55      ans.pop_back(); ans.pop_back();
56
57      cout<<ans<<endl;
58      return 0;
59  }
```

Time Complexity: O($\sqrt{N}$)

input : 84
output : $2^2 \times 3 \times 7$

Lab 3

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

RSA Algorithm.

In 1977, Ronald Rivest, Adi Shamir and Leonard Adleman (cf. Fig. 7.1) proposed a asymmetric cryptographic scheme RSA. There are many applications for RSA, but in practice it is most often used for: encryption of small pieces of data, especially for key transport digital signatures for digital certificates on the Internet. RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

RSA Key Generation:

Output: Public key: $k_{pub} = (n, e)$ and private key: $k_{pr} = (d)$

1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n) - 1\}$ such that $\gcd(e, \Phi(n)) = 1$.
5. Compute the private key d such that $d \cdot e \equiv 1 \pmod{\Phi(n)}$.

Encryption and Decryption: RSA encryption and decryption is done in the integer ring Zn. RSA encrypts plaintexts x, where we consider the bit string representing x to be an element in Zn = 0,1,...,n1. As a consequence the binary value of the plaintext x must be less than n. The same holds for the ciphertext. Encryption with the public key and decryption with the private key are as shown below:

RSA Encryption Given the public key $(n, e) = k_{pub}$ and the plaintext x, the encryption function is:

$$y = e_{k_{pub}}(x) \equiv x^e n$$

where $x, y \in Z_n$.

RSA Decryption Given the private key $d = k_{pr}$ and the ciphertext y, the decryption function is:

$$x = d_{k_{pr}}(y) \equiv y^d n$$

where $x, y \in Z_n$.

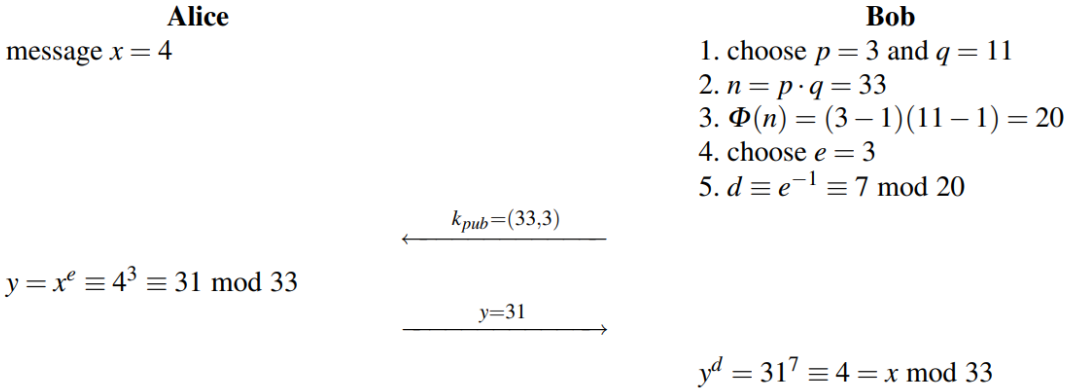

Figure 1: RSA Algorithm

Example 1:
Choose $p = 61$ *and* $q = 53$.
Compute $n = p \cdot q = 61 \cdot 53 = 3233$.
Compute $\Phi(n) = (p-1)(q-1) = 3120$.
Select $e = 17$ *such that* $\gcd(e, \Phi(n)) = 1$.
Compute d *such that* $d \cdot e \equiv 1 \pmod{\Phi(n)}$.
Public key: $k_{pub} = (n, e) = (3233, 17)$.
Private key: $k_{pr} = (d) = (2753)$.
Given plaintext $x = 42$, *compute* y *using the encryption function.*
Computed ciphertext $y = 2557$, *Decrypt* y *to obtain the original plaintext* x.

Example 2:
Choose $p = 71$ *and* $q = 37$.
Compute $n = p \cdot q = 71 \cdot 37 = 2627$.
Compute $\Phi(n) = (p-1)(q-1) = 2520$.
Select $e = 11$ *such that* $\gcd(e, \Phi(n)) = 1$.
Compute d *such that* $d \cdot e \equiv 1 \pmod{\Phi(n)}$.
Public key: $k_{pub} = (n, e) = (2627, 11)$.
Private key: $k_{pr} = (d) = (2291)$.
Given plaintext $x = 128$, *compute* y *using the encryption function.*
Computed ciphertext $y = 2400$, *Decrypt* y *to obtain the original plaintext* x.

Implementation in C++

```
#include<bits/stdc++.h>
using namespace std;

bool isPrime(long long n) {
    if (n<=1) return 0;
    else if (n==2) return 1;
    else if ((n&1)==0) return 0;
    else {
        int i = 3;
```

```
10          while (i*i <= n) {
11              if (n % i == 0) return 0;
12              i += 2;
13          }
14          return true;
15      }
16  }
17
18  long long gcd(long long a, long long b, long long &x, long long &y) {
19      if (a == 0) {
20          x = 0;
21          y = 1;
22          return b;
23      }
24
25      long long x1, y1;
26      long long g = gcd(b % a, a, x1, y1);
27
28      x = y1 - (b / a) * x1;
29      y = x1;
30
31      return g;
32  }
33
34  long long inverse(long long e, long long phi) {
35      long long x, y;
36      long long g = gcd(e, phi, x, y);
37
38      if (g != 1) {
39          cerr << "Inverse␣doesn't␣exist\n";
40          return -1;
41      } else {
42          // Ensure 'x' is positive
43          return (x % phi + phi) % phi;
44      }
45  }
46
47  long long mod_pow(long long base, long long exponent, long long modulus) {
48      long long result = 1;
49
50      while (exponent > 0) {
51          if (exponent & 1)
52              result = (result * base) % modulus;
53
54          exponent >>= 1;
55          base = (base * base) % modulus;
56      }
57
58      return result;
59  }
60
```

```
61
62 int main() {
63     long long P, Q;
64     cout<<"Please␣Enter␣the␣value␣of␣P:␣";
65     cin>>P;
66     cout<<"Please␣Enter␣the␣value␣of␣Q:␣";
67     cin>>Q;
68     while (!isPrime(P)) { // O(sqrt(P))
69         cout<<P<<"␣is␣not␣prime.␣Please␣enter␣prime␣number(P):␣";
70         cin>>P;
71     }
72     while (!isPrime(Q)) { // O(sqrt(Q))
73         cout<<Q<<"␣is␣not␣prime.␣Please␣enter␣prime␣number(Q):␣";
74         cin>>Q;
75     }
76     long long N = P * Q;
77     long long phi = (P - 1) * (Q - 1);
78
79     cout<<"Value␣of␣phi␣is:␣"<<phi<<endl;
80
81     long long e;
82     cout<<"Please␣Enter␣the␣value␣of␣e␣(2-"<<phi-1<<")␣that␣are␣coprime␣to␣"
83     cin>>e;
84
85     while (e < phi) {  // O(1)
86         // e must be co-prime to phi and smaller than phi.
87         if (gcd(e, phi) == 1)
88             break;
89         else {
90             e = (e + 1) % phi;
91             if (e == 1) e++;
92         }
93     }
94
95     long long d = inverse(e, phi);
96
97     cout<<"Value␣of␣e␣is␣:␣"<<e<<endl;
98     cout<<"Value␣of␣d␣is␣:␣"<<d<<endl;
99
100    long long m;
101    cout<<"Please␣Enter␣the␣value␣of␣m:␣";
102    cin>>m;
103
104    // Encryption c = (msg ^ e) % n
105    long long c = mod_pow(m, e, N);  // O(log(e))
106    cout<<"\nCipher␣Text␣=␣" << c;
107
108    // Decryption m = (c ^ d) % n
109    long long msg = mod_pow(c, d, N); // O(log(e))
110    cout<<"\nOriginal␣Message␣Sent␣=␣" << msg;
111
```

```
112     return 0;
113 }
```

```
Please Enter the value of P: 61
Please Enter the value of Q: 53
Value of phi is: 3120
Please Enter the value of e (2-3119) that are coprime to 3120 : 17
Value of e is : 17
Value of d is : 2753
Please Enter the value of m: 42
Cipher Text = 2557
Original Message Sent = 42
```

```
Please Enter the value of P: 71
Please Enter the value of Q: 37
Value of phi is: 2520
Please Enter the value of e (2-2519) that are coprime to 2520 : 11
Value of e is : 11
Value of d is : 2291
Please Enter the value of m: 128
Cipher Text = 2400
Original Message Sent = 128
```

Lab 4

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Given three positive integers A, B, and N, which represent a linear congruence of the form AX=B (mod N), the task is to print all possible values of X (mod N) i.e in the range [0, N-1] that satisfies this equation. If there is no solution, print -1.

Solution:
To find all possible values of $X \pmod{N}$ satisfying the linear congruence $AX \equiv B \pmod{N}$, where A, B, and N are integers. we can use the extended Euclidean algorithm. The algorithm will help to find the modular inverse of A modulo N, and then we can use it to find a particular solution. After that, you can use the period of the solution to generate all possible solutions.

Given the linear congruence:

$$AX \equiv B \pmod{N}$$

1. **Existence of Solution:** The linear congruence has a solution if and only if B is divisible by the greatest common divisor (gcd) of A and N. If $\gcd(A, N)$ does not divide B, then there is no solution.

Theorem 1. If $AX \equiv B \pmod{N}$ has a solution, then B is divisible by $\gcd(A, N)$.

Proof. Assume that $AX \equiv B \pmod{N}$ has a solution. This means there exists an integer X_0 such that $AX_0 \equiv B \pmod{N}$.
We can express this congruence as $AX_0 - B = kN$ for some integer k. Rearranging, we get $AX_0 - kN = B$.
Now, let $d = \gcd(A, N)$. Since d divides both A and N, it must divide AX_0 as well. Therefore, d must also divide $AX_0 - kN$.
This implies that d divides B, because $B = AX_0 - kN$.
Hence, we have shown that if $AX \equiv B \pmod{N}$ has a solution, then B is divisible by $\gcd(A, N)$. □

2. **Finding Modular Inverse:** If $\gcd(A, N)$ divides B, then a solution exists, and we can proceed to find the modular inverse of A modulo N. The modular inverse A^{-1} exists if A and N are relatively prime (i.e., $\gcd(A, N) = 1$).
Using the extended Euclidean algorithm, we find integers x and y such that $Ax + Ny = \gcd(A, N)$. If $\gcd(A, N) = 1$, then $Ax + Ny = 1$, and x is the modular inverse of A modulo N.

3. **Particular Solution:** Once we have the modular inverse A^{-1}, we can find a particular solution X_0 using the formula:

$$X_0 \equiv A^{-1} \cdot B \pmod{N}$$

4. **General Solution:** The general solution is then given by:

$$X \equiv (X_0 + k \cdot \frac{N}{\gcd(A, N)}) \pmod{N}$$

where k is an integer, and $\frac{N}{\gcd(A,N)}$ is the period of the solution. This expression ensures that all solutions are considered and lie in the range $[0, N-1]$.

Implementation in C++

```cpp
#include <bits/stdc++.h>
using namespace std;

// calculate the greatest common divisor (GCD) using Euclid's Algorithm
long long gcd(long long a, long long b, long long &x, long long &y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
    }
    long long x1, y1;
    long long g = gcd(b % a, a, x1, y1);

    x = y1 - (b / a) * x1;
    y = x1;

    return g;
}

// find the modular inverse of 'a' modulo 'm'
long long inverse(long long a, long long m) {
    long long x, y;
    long long g = gcd(a, m, x, y);

    if (g != 1)  return -1; // Modular inverse doesn't exist
    else return (x % m + m) % m; // Ensure 'x' is positive
}

// solve the linear congruence AX     B (mod N)
void solve_linear_congruence(long long A, long long B, long long N) {
    long long A_inv = inverse(A, N);  // modular inverse of A modulo N

    if (A_inv == -1) {
        cout << -1 ; // No solution
        return;
    }

    // Find a particular solution X0 using the modular inverse
    long long X0 = (A_inv * B) % N;

    // Print all possible solutions in the range [0, N-1]
```

```
42      set<long long> X;
43      for (long long k = 0; k < N; ++k)
44          X.insert((X0 + k * (N / gcd(A, N))) % N);
45
46      for (auto x: X) cout<<x<<"␣";
47      cout << endl;
48  }
49
50  int main() {
51      long long A, B, N;
52      cout << "Enter␣values␣for␣A,␣B,␣and␣N:␣";
53      cin >> A >> B >> N;
54
55      solve_linear_congruence(A, B, N);
56
57      return 0;
58  }
```

```
Enter values for A, B, and N: 3 2 7

Solutions: 3
```

```
Enter values for A, B, and N: 101 21 6564

Solutions: 1365
```

```
Enter values for A, B, and N: 5 7 10

Solutions: -1
```

Lab 5

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Suppose you have a sample space representing the number of heads obtained when tossing n fair coins and biased coins. For the biased coin, the probability of getting a head p(head)) is 0.3. Compute the probability distributions for both the fair and biased coins regarding the number of heads, and then calculate the statistical distance between these two distributions.

Solution:
Let X represent the number of heads obtained when tossing n coins. The probability mass function (PMF) for X can be represented by the binomial distribution:

$$P(X = k) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}$$

Where:

- n is the number of coin tosses,
- k is the number of heads obtained,
- p is the probability of getting a head.

For the fair coin:

$$p_{\text{fair}} = 0.5$$

For the biased coin:

$$p_{\text{biased}} = 0.3$$

Now, the probability distributions for the fair coin:

$$P_{\text{fair}}(X = k) = \binom{n}{k} \cdot 0.5^k \cdot 0.5^{n-k}$$

The probability distributions for the biased coin:

$$P_{\text{biased}}(X = k) = \binom{n}{k} \cdot 0.3^k \cdot 0.7^{n-k}$$

Now, the statistical distance between two probability distributions P and Q is defined as:

$$\Delta(P, Q) = \frac{1}{2} \sum_x |P(x) - Q(x)|$$

Where x ranges over all possible values.

Let's calculate the statistical distance between the fair and biased coin distributions.

$$\Delta(P_{\text{fair}}, P_{\text{biased}}) = \frac{1}{2} \sum_{k=0}^{n} |P_{\text{fair}}(X = k) - P_{\text{biased}}(X = k)|$$

$$\Delta(P_{\text{fair}}, P_{\text{biased}}) = \frac{1}{2} \sum_{k=0}^{n} \left| \binom{n}{k} \cdot 0.5^k \cdot 0.5^{n-k} - \binom{n}{k} \cdot 0.3^k \cdot 0.7^{n-k} \right|$$

For $\mathbf{n = 5}$

Fair Coin Probability Distribution:

$$P_{\text{fair}}(0 \text{ heads}) = 0.03125$$
$$P_{\text{fair}}(1 \text{ heads}) = 0.15625$$
$$P_{\text{fair}}(2 \text{ heads}) = 0.3125$$
$$P_{\text{fair}}(3 \text{ heads}) = 0.3125$$
$$P_{\text{fair}}(4 \text{ heads}) = 0.15625$$
$$P_{\text{fair}}(5 \text{ heads}) = 0.03125$$

Biased Coin Probability Distribution:

$$P_{\text{biased}}(0 \text{ heads}) = 0.16807$$
$$P_{\text{biased}}(1 \text{ heads}) = 0.36015$$
$$P_{\text{biased}}(2 \text{ heads}) = 0.3087$$
$$P_{\text{biased}}(3 \text{ heads}) = 0.1323$$
$$P_{\text{biased}}(4 \text{ heads}) = 0.02835$$
$$P_{\text{biased}}(5 \text{ heads}) = 0.00243$$

Statistical Distance between Distributions: 0.34072

For $\mathbf{n = 10}$

Fair Coin Probability Distribution:

$$P(0 \text{ heads}) = 0.0078125$$
$$P(1 \text{ heads}) = 0.0546875$$
$$P(2 \text{ heads}) = 0.164062$$
$$P(3 \text{ heads}) = 0.273438$$
$$P(4 \text{ heads}) = 0.273438$$
$$P(5 \text{ heads}) = 0.164062$$
$$P(6 \text{ heads}) = 0.0546875$$
$$P(7 \text{ heads}) = 0.0078125$$

Biased Coin Probability Distribution:

$$P(0 \text{ heads}) = 0.0823543$$
$$P(1 \text{ heads}) = 0.247063$$
$$P(2 \text{ heads}) = 0.317652$$
$$P(3 \text{ heads}) = 0.226894$$
$$P(4 \text{ heads}) = 0.0972405$$
$$P(5 \text{ heads}) = 0.0250047$$
$$P(6 \text{ heads}) = 0.0035721$$
$$P(7 \text{ heads}) = 0.0002187$$

Statistical Distance between Distributions: 0.420507

Implementation in C++

```cpp
#include <bits/stdc++.h>
using namespace std;

double pow_(double base, long long exponent) {
    double result = 1;

    while (exponent > 0) {
        if (exponent & 1)
            result = result * base;

        exponent >>= 1;
        base = base * base;
    }

    return result;
}

// Function to calculate binomial coefficient (n choose k)
long long binCoeff(long long n, long long k) {
   long long C[k+1];
   memset(C, 0, sizeof(C));
   C[0] = 1;
   for (long long i = 1; i <= n; i++) {
      for (int j = min(i, k); j > 0; j--)
         C[j] = C[j] + C[j-1];
   }
   return C[k];
}


double fairCoinProbability(long long n, long long k) {
    double p_head = 0.5;
    return binCoeff(n, k) * pow_(p_head, k) * pow_(1 - p_head, n - k);
}

double biasedCoinProbability(long long n, long long k) {
    double p_head_bias = 0.3;
    return binCoeff(n, k) * pow_(p_head_bias, k) * pow_(1-p_head_bias, n-k);
}

double statisticalDistance(vector<double>& dist1, vector<double>& dist2) {
    double distance = 0.0;
    for (size_t i = 0; i < dist1.size(); ++i) {
        distance += 0.5 * fabs(dist1[i] - dist2[i]);
    }
    return distance;
}


```

```
50 int main() {
51     long long n = 7; // Number of coin tosses
52 
53     // Calculate probability distributions for fair and biased coins
54     vector<double> fairDistrib(n + 1, 0.0);
55     vector<double> biasedDistrib(n + 1, 0.0);
56 
57     for (long long k = 0; k <= n; ++k) {
58         fairDistrib[k] = fairCoinProbability(n, k);
59         biasedDistrib[k] = biasedCoinProbability(n, k);
60     }
61 
62     // Output probability distributions
63     cout << "Fair␣Coin␣Probability␣Distribution:\n";
64     for (long long k = 0; k <= n; ++k)
65         cout << "P(" << k << "␣heads)␣=␣" << fairDistrib[k] << "\n";
66 
67     cout << "\nBiased␣Coin␣Probability␣Distribution:\n";
68     for (long long k = 0; k <= n; ++k)
69         cout << "P(" << k << "␣heads)␣=␣" << biasedDistrib[k] << "\n";
70 
71     // Calculate and output the statistical distance
72     double distance = statisticalDistance(fairDistrib, biasedDistrib);
73     cout << "\nStatistical␣Distance␣between␣Distributions:␣" << distance;
74 
75     return 0;
76 }
```

Lab 6

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Suppose you are playing a game where you roll a fair six-sided die. Let X be the random variable representing the outcome of the die roll.

1. Compute the expectation E[X] of the random variable X.
2. Compute the expectation E[X^2] of the random variable X.
3. Using the formula for variance, calculate the variance of X.

Solution:

Random Variable (X): A random variable is a quantity whose possible values depend on the outcome of a random process, each value having a certain probability.

Expectation $E[X]$: The expectation, denoted as $E[X]$, signifies the average value of the random variable X across multiple trials. It's determined by multiplying each potential value of X by its probability and summing these products.

$$E[X] = \sum_{i=1}^{n} x_i \cdot P(X = x_i)$$

Expectation of Squared Outcome $E[X^2]$: We can find the expectation of the squared outcome of X, indicating its average value squared over numerous trials.

$$E[X^2] = \sum_{i=1}^{n} x_i^2 \cdot P(X = x_i)$$

Variance (Var$[X]$): Variance quantifies the spread of values of a random variable around its mean (expected value). It's computed by averaging the squared deviations of each value from the mean.

$$\text{Var}[X] = E[X^2] - (E[X])^2$$

Example: Consider a fair six-sided die, where each face has an equal probability of $\frac{1}{6}$. Let X be the random variable representing the outcome of rolling the die.

$$E[X] = \frac{1}{6}(1) + \frac{1}{6}(2) + \frac{1}{6}(3) + \frac{1}{6}(4) + \frac{1}{6}(5) + \frac{1}{6}(6) = 3.5$$

$$E[X^2] = \frac{1}{6}(1^2) + \frac{1}{6}(2^2) + \frac{1}{6}(3^2) + \frac{1}{6}(4^2) + \frac{1}{6}(5^2) + \frac{1}{6}(6^2) = 15.17$$

$$\text{Var}[X] = E[X^2] - (E[X])^2 = 15.17 - (3.5)^2 = 2.92$$

Implementation in C++

```cpp
#include <iostream>
#include <cmath>
using namespace std;

class Die {
private:
    int sides;

public:
    Die(int sides) : sides(sides) {}

    double expectation() const {
        return (1.0 + sides) / 2.0;
    }

    double expectationSquared() const {
        double sum = 0.0;
        for (int i = 1; i <= sides; ++i)
            sum += pow(i, 2);
        return sum / sides;
    }

    double variance() const {
        double ex = expectation();
        double exSquared = expectationSquared();
        return exSquared - pow(ex, 2);
    }
};

int main() {
    int sides = 10; // Number of sides on the die
    // Create a Die object
    Die die(sides);

    // Calculate and print expectation E[X]
    cout << "Expectation E[X]: " << die.expectation() << endl;
    // Calculate and print expectation E[X^2]
    cout << "Expectation E[X^2]: " << die.expectationSquared() << endl;
    // Calculate and print variance of X
    cout << "Variance of X: " << die.variance() << endl;
    return 0;
}
```

Time Complexity: $O(n)$

Output:

```
Expectation E[X]: 3.5
Expectation E[X^2]: 15.1667
Variance of X: 2.91667
```

Lab 7

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Consider a bin1 full of n red, n green and n yellow balls and another empty bin2. Now x balls are uniformly chosen from bin1 and placed in bin2.What is the probability that bin 2 now has at least y green balls. Write a program that takes n,x,y as inputs and outputs the upperbounds of the probabilty according to Markov, Cheybyshev and Chernoff (with different choice of Δ)

Solution:

Markov's Inequality: Markov's inequality provides an upper bound on the probability that a non-negative random variable X is greater than or equal to a specified positive constant. It states that for any non-negative random variable X and any $a > 0$, the probability that X is at least a times its mean $E[X]$ is at most the reciprocal of a.

$$P(X \geq a) \leq \frac{E[X]}{a}$$

Chebyshev's Inequality: Chebyshev's inequality provides an upper bound on the probability that a random variable X deviates from its mean by more than a specified number of standard deviations. It states that for any random variable X with finite mean μ and finite non-zero variance σ^2, and any $k > 0$, the probability that X deviates from its mean by more than k standard deviations is at most $1/k^2$.

$$P(|X - \mu| \geq k) \leq \frac{\sigma}{k^2}$$

Chernoff Bounds: Chernoff bounds offer exponential tail bounds on the sum of independent random variables. In our case, we can utilize Chernoff bounds to establish an upper limit on the likelihood that the number of green balls in bin2 surpasses a predetermined threshold

$$P(X \geq (1+\delta)E[X]) \leq \frac{e^{-\delta E[X]}}{(1+\delta)^{(1+\delta)E[X]}}$$

Example: Let X be a random variable representing the outcome of rolling a fair six-sided die. The possible outcomes are $\{1, 2, 3, 4, 5, 6\}$, each with probability $\frac{1}{6}$.

$$E[X] = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = \frac{21}{6} = 3.5$$

$$E[X^2] = \frac{1}{6}(1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2) = \frac{91}{6}$$

$$\text{Var}[X] = \frac{91}{6} - \left(\frac{7}{2}\right)^2 = \frac{35}{12} \approx 2.9167$$

Markov's Inequality

$$P(X \geq 10) \leq \frac{E[X]}{10} = \frac{3.5}{10} = 0.35$$

Chebyshev's Inequality

$$P(|X - 3.5| \geq 10) \leq \frac{\text{Var}[X]}{10^2} = \frac{2.9167}{10^2} \approx 0.029167$$

Chernoff Bounds

$$P(X \geq (1+\delta)E[X]) \leq \frac{e^{-\delta E[X]}}{(1+\delta)^{(1+\delta)E[X]}}$$

Using $\delta = 1$:

$$P(X \geq 7) \leq \frac{e^{-3.5}}{(1+1)^{2*3.5}} = \frac{e^{-3.5}}{2^7} \approx 0.000235917$$

Implementation in C++

```
#include <iostream>
#include <cmath>
using namespace std;


class Bounds {
public:
    double markovBound(int n, int x, int y) {
        double mean = static_cast<double>(x * n) / 3;
        return static_cast<double>(n) / (y * mean);
    }
    double chebyshevBound(int n, int x, int y) {
        double variance = static_cast<double>(x * n) * (2 * n) / 9;
        double mean = static_cast<double>(x * n) / 3;
        double stdDev = sqrt(variance);
        double k = (mean - static_cast<double>(y * n)) / stdDev;
        return 1 / (1 + pow(k, 2));
    }

    double chernoffBound(int n, int x, int y, double delta) {
        double mean = static_cast<double>(x * n) / 3;
        double k = (mean - static_cast<double>(y * n)) / mean;
        return exp(-k * delta) / pow(1 + delta, (1 + delta)*mean);
    }
};

int main() {
    int n, x, y;
    double delta;

    cout << "Enter the value of n, x, y: ";
    cin >> n >> x >> y;

    cout << "Enter the value of delta (for Chernoff bound): ";
    cin >> delta;
```

```
36
37      Bounds bounds;
38
39      cout<<"Markov's␣Upper␣Bound:␣"<<bounds.markovBound(n, x, y)<<endl;
40      cout<<"Chebyshev's␣Upper␣Bound:␣"<<bounds.chebyshevBound(n, x, y)<<endl;
41      cout<<"Chernoff's␣Upper␣Bound␣with␣delta␣=␣"<<delta<<":␣"<<bounds.cherno
42
43      return 0;
44  }
```

Time Complexity: $O(1)$, given that we already know $E[X]$ and $Var[X]$

Output:

```
Enter the value of n, x, y: 10 5 3
Enter the value of delta (for Chernoff bound): 0.1
Markov's Upper Bound: 5.55556
Chebyshev's Upper Bound: 12.3457
Chernoff's Upper Bound with delta = 0.1: 0.0329086
```

Lab 8

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Take a prime p as input and write functions to implement field operations of Z_p

Solution:

The finite field Z_p, where p is a prime number, consists of integers modulo p. It is denoted by Z_p and contains p elements, known as residues, obtained by dividing integers by p and taking the remainder.
Mathematically, the set Z_p is defined as:

$$Z_p = \{0, 1, 2, \ldots, p-1\}$$

Addition, subtraction, multiplication, and division in Z_p are defined modulo p. For any two integers a and b in Z_p, the operations are computed as follows:

- **Addition**: The sum $a + b$ is calculated modulo p as $(a + b) \mod p$.
- **Subtraction**: The difference $a - b$ is calculated modulo p as $(a - b) \mod p$.
- **Multiplication**: The product $a \times b$ is calculated modulo p as $(a \times b) \mod p$.
- **Division**: The quotient $\frac{b}{a}$ is calculated as the modular multiplicative inverse of a modulo p, denoted as a^{-1}, multiplied by b. Mathematically, $\frac{b}{a}$ is represented as $(b \times a^{-1}) \mod p$.

Example: Let's consider the finite field Z_7, where $p = 7$. Perform addition, subtraction, multiplication, and division for the following integers:

$$a = 2, \qquad b = 3$$

Addition

$$a + b = (2 + 3) \mod 7 = 5 \mod 7 = 5$$

Subtraction

$$a - b = (2 - 3) \mod 7 = -1 \mod 7 = 6$$

Multiplication

$$a \times c = (2 \times 3) \mod 7 = 8 \mod 7 = 6$$

Division To find $\frac{a}{b}$, we need to calculate the modular multiplicative inverse of b modulo 7. Since $b = 3$, we find b^{-1} such that $3 \times b^{-1} \equiv 1 \mod 7$. In this case, $b^{-1} = 5$ because $6 \times 6 \equiv 1 \mod 7$.

$$\frac{a}{b} = (2 \times 5) \mod 7 = 10 \mod 7 = 3$$

Implementation in C++

```cpp
#include <iostream>
using namespace std;

class ModularArithmetic {
private:
    int p;

    // Function to perform modulo operation
    int mod(int a, int b) {
        int result = a % b;
        if (result < 0) result += b;

        return result;
    }

public:
    ModularArithmetic(int modulus) : p(modulus) {}

    // Function to calculate modular exponentiation (a^b mod p)
    int modExp(int a, int b) {
        if (b == 0) return 1;
        long long int temp = modExp(a, b / 2);
        long long int result = (temp * temp) % p;
        if (b % 2 == 1) result = (result * a) % p;

        return static_cast<int>(result);
    }

    // Function to calculate modular inverse (a^(-1) mod p)
    int modInverse(int a) {
        return modExp(a, p - 2);
    }

    // Function to perform addition in Zp
    int add(int a, int b) {
        return mod(a + b, p);
    }

    // Function to perform subtraction in Zp
    int subtract(int a, int b) {
        return mod(a - b, p);
    }

    // Function to perform multiplication in Zp
    int multiply(int a, int b) {
        return mod(a * b, p);
    }

    // Function to perform division in Zp
```

```
50      int divide(int a, int b) {
51          // Division by b is equivalent to multiplication by the modular inve
52          return multiply(a, modInverse(b));
53      }
54  };
55
56  int main() {
57      int p;
58      cout << "Enter␣the␣prime␣number␣p:␣";
59      cin >> p;
60
61      ModularArithmetic Zp(p);
62
63      int a, b;
64      cout << "Enter␣two␣integers␣a␣and␣b:␣";
65      cin >> a >> b;
66
67      // Perform field operations
68      int sum = Zp.add(a, b);
69      int difference = Zp.subtract(a, b);
70      int product = Zp.multiply(a, b);
71      int quotient = Zp.divide(a, b);
72
73      // Output results
74      cout<<"Sum(a␣+␣b)␣mod␣"<<p<<"␣=␣"<<sum<<endl;
75      cout<<"Difference(a␣-␣b)␣mod␣"<<p<<"␣=␣"<<difference<<endl;
76      cout<<"Product(a␣*␣b)␣mod␣"<<p<<"␣=␣"<<product<<endl;
77      cout<<"Quotient(a␣/␣b)␣mod␣"<<p<<"␣=␣"<<quotient<<endl;
78
79      return 0;
80  }
```

Time Complexity: $O(\log \max(a, b))$

Output:

```
Enter the prime number p: 7
Enter two integers a and b: 2 3
Sum(a + b) mod 7 = 5
Difference(a - b) mod 7 = 6
Product(a * b) mod 7 = 6
Quotient(a / b) mod 7 = 3
```

Lab 9

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Write a program that takes p (a prime) as input, an $m \times n$ matrix A, and Z_p, and outputs its Reduced Row Echelon (RRE) form.

Solution:

The Reduced Row Echelon form (RRE) of a matrix is a specific form achieved through a series of row operations. In the RRE form:

1. Each leading entry (pivot) of a row is 1.
2. The leading entry is the only non-zero entry in its column.
3. Rows with all zero entries, if any, are at the bottom of the matrix.
4. Each leading 1 is to the right of the leading 1 of the row above it.

Example: Consider the following matrix:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

To reduce this matrix to its Reduced Row Echelon form (RRE), we'll perform row operations to transform it. Here's the step-by-step process:

1. We'll begin by making the leading entry (pivot) of the first column 1 and eliminating other entries below it.
2. Subtract the first row from the third row:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 0 & 6 & 5 \end{pmatrix}$$

3. Subtracting the first row from the second row yields:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 6 & 2 \\ 0 & 6 & 5 \end{pmatrix}$$

4. To make the leading entry of the third row 1, we'll divide the entire row by 3 (means multipy by $3^{-1} = 5$:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 6 & 2 \\ 0 & 0 & 3 \end{pmatrix}$$

5. Now, to make the leading entry of the second row 1, we'll divide the entire row by 6 (means multipy by $6^{-1} = 6$:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix}$$

6. This is the Reduced Row Echelon form (RRE) of the matrix. Each leading entry (pivot) is 1, and each leading entry is the only non-zero entry in its column.

Implementation in C++

```
#include <iostream>
#include <vector>
using namespace std;

class Matrix {
private:
    vector<vector<int>> data;
    int p; // Prime
    // Function to perform modulo operation
    int mod(int a, int b) {
        int result = a % b;
        if (result < 0) result += b;
        return result;
    }
    // Function to perform subtraction in Zp
    int subtract(int a, int b) {
        return mod(a - b, p);
    }
    int multiply(int a, int b) {
        return mod(a * b, p);
    }
    // Function to calculate modular exponentiation (a^b mod p)
    int modExp(int base, int exponent) {
        long long result = 1;
        while (exponent > 0) {
            if (exponent & 1)
                result = (result * base) % p;
            exponent >>= 1;
            base = (base * base) % p;
        }
        return static_cast<int>(result);;
    }
    // Function to calculate modular inverse (a^(-1) mod p)
    int modInverse(int a) {
        return modExp(a, p - 2);
    }
public:
    Matrix(int m, int n, int prime) : data(m, vector<int>(n)), p(prime) {}

    void setElement(int row, int col, int value) {
        data[row][col] = value % p; // Ensure element belongs to Z_p
    }

    void printMatrix() const {
        for (const auto& row : data) {
            for (int elem : row)
                cout << elem << "␣";
            cout << endl;
        }
    }
```

```
51
52      void rowReduce() {
53          for (int col = 0; col < data.size(); ++col) {
54              // Make the pivot element 1
55              int pivot = data[col][col];
56              int inv = modInverse(pivot);
57              for (int j = col; j < data.size() + 1; ++j)
58                  data[col][j] = multiply(data[col][j], inv);
59
60              for (int row = col+1; row < data.size(); ++row) {
61                  int factor = data[row][col];
62                  for (int j = col; j < data.size() + 1; ++j) {
63                      int prod = multiply(factor, data[col][j]);
64                      data[row][j] = subtract(data[row][j], prod);
65                  }
66              }
67          }
68      }
69  };
70
71  int main() {
72      int p, m, n;;
73      cout << "Enter␣the␣prime␣number␣p:␣"; cin >> p;
74      cout << "Enter␣the␣dimensions␣of␣the␣matrix␣(m␣x␣n):␣"; cin >> m >> n;
75
76      Matrix A(m, n, p); // Create a matrix with prime modulus p
77      cout << "Enter␣the␣elements␣of␣the␣matrix␣A␣(mod␣" << p << "):\n";
78      for (int i = 0; i < m; ++i) {
79          for (int j = 0; j < n; ++j) {
80              int value;
81              cin >> value;
82              A.setElement(i, j, value);
83          }
84      }
85      A.rowReduce(); // Perform row reduction
86      cout << "\nRow␣Reduced␣Echelon␣Form␣of␣matrix␣A:\n";
87      A.printMatrix(); // Output the row-reduced echelon form
88      return 0;
89  }
```

Time Complexity: $O(m \times n)$

Output:

```
Enter the prime number p: 7
Enter the dimensions of the matrix (m x n): 3 3
Enter the elements of the matrix A (mod 7):
1 2 3
2 3 1
1 1 1

Row Reduced Echelon Form of matrix A:
1 2 3
0 1 5
0 0 1
```

Lab 10

Name: Yuvraj Singh
Roll No: 23072021
M.Tech CSE

Problem: Write a program that takes an $m \times n$ matrix A and an $m \times 1$ matrix B over Z_p and outputs whether B belongs to the range space of A along with two solutions for the systems of equations $AX = B$ (if this system has more than one solution), the prime p, m, n, A, B are to be taken as input.

Solution:

Range Space of a Matrix: The range space (or column space) of a matrix A consists of all possible linear combinations of its column vectors. It represents the subspace of the vector space spanned by the column vectors of A.
To check if the vector B belongs to the range space of matrix A, we perform Gaussian elimination on the augmented matrix $[A \,|\, B]$ to obtain its RRE form. If all the non-zero rows of the resulting matrix correspond to non-zero elements in B, then B lies in the range space of A.
Finding Solutions (if multiple): If the system of equations $AX = B$ has multiple solutions, we can provide two solutions by setting $n - 2$ free variables to arbitrary values and solving for the remaining variables using back-substitution.
Example:

Implementation in C++

```
#include <iostream>
#include <vector>
using namespace std;

class ModularArithmetic {
private:
    int p;
    int mod(int a, int b) {
        int result = a % b;
        if (result < 0) result += b;
        return result;
    }

    int modExp(int base, int exponent) {
        long long result = 1;
        while (exponent > 0) {
            if (exponent & 1)
                result = (result * base) % p;
            exponent >>= 1;
            base = (base * base) % p;
        }
        return static_cast<int>(result);;
    }
```

```
24
25      int modInverse(int a) {
26          return modExp(a, p - 2);
27      }
28
29      int add(int a, int b) {
30          return mod(a + b, p);
31      }
32
33      int subtract(int a, int b) {
34          return mod(a - b, p);
35      }
36
37      int multiply(int a, int b) {
38          return mod(a * b, p);
39      }
40
41      int divide(int a, int b) {
42          return multiply(a, modInverse(b));
43      }
44
45  public:
46      ModularArithmetic(int prime) : p(prime) {}
47
48      void gaussianElimination(vector<vector<int>>& A, vector<int>& B) {
49          int m = A.size();
50          int n = A[0].size();
51          int pivot_col = 0;
52          for (int r = 0; r < m; ++r) {
53              if (pivot_col >= n) break;
54              int i = r;
55              while (A[i][pivot_col] == 0) {
56                  ++i;
57                  if (i == m) {
58                      i = r;
59                      ++pivot_col;
60                      if (pivot_col == n) break;
61                  }
62              }
63              swap(A[i], A[r]);
64              int pivot = A[r][pivot_col];
65              for (int j = 0; j < n; ++j)
66                  A[r][j] = divide(A[r][j], pivot);
67
68              B[r] = divide(B[r], pivot);
69              for (int i = 0; i < m; ++i) {
70                  if (i != r) {
71                      int pivot = A[i][pivot_col];
72                      for (int j = 0; j < n; ++j)
73                          A[i][j] = subtract(A[i][j], multiply(pivot, A[r][j]))
74                      B[i] = subtract(B[i], multiply(pivot, B[r]));
```

```
75                  }
76              }
77              ++pivot_col;
78          }
79      }
80
81      bool isInRangeSpace(const vector<vector<int>>& A, const vector<int>& B) {
82          vector<vector<int>> RRE_A = A;
83          vector<int> RRE_B = B;
84          gaussianElimination(RRE_A, RRE_B);
85          for (int i = 0; i < RRE_A.size(); ++i) {
86              bool allZero = true;
87              for (int j = 0; j < RRE_A[i].size(); ++j) {
88                  if (RRE_A[i][j] != 0) {
89                      allZero = false;
90                      break;
91                  }
92              }
93              if (allZero && RRE_B[i] != 0) return false;
94          }
95          return true;
96      }
97
98      void backSubstitution(const vector<vector<int>>& A, const vector<int>& B)
99          int n = A[0].size();
100         vector<int> solution(n);
101         for (int i = n - 1; i >= 0; --i) {
102             solution[i] = B[i];
103             for (int j = i + 1; j < n; ++j) {
104                 solution[i] = subtract(solution[i], multiply(A[i][j], solutio
105             }
106         }
107         cout << "Solutions␣for␣AX␣=␣B:" << endl;
108         for (int i = 0; i < n; ++i) {
109             cout << "x" << i + 1 << "␣=␣" << solution[i] << endl;
110         }
111     }
112 };
113
114 int main() {
115     int p, m, n;
116     cout << "Enter␣the␣prime␣number␣p:␣";
117     cin >> p;
118     cout << "Enter␣the␣dimensions␣of␣matrix␣A␣(m␣x␣n):␣";
119     cin >> m >> n;
120
121     vector<vector<int>> A(m, vector<int>(n));
122     cout << "Enter␣the␣elements␣of␣matrix␣A:" << endl;
123     for (int i = 0; i < m; ++i)
124         for (int j = 0; j < n; ++j)
125             cin >> A[i][j];
```

```
126
127     vector<int> B(m);
128     cout << "Enter␣the␣elements␣of␣matrix␣B:" << endl;
129     for (int i = 0; i < m; ++i) cin >> B[i];
130
131     ModularArithmetic mod(p);
132
133     // Check if B belongs to the range space of A
134     if (mod.isInRangeSpace(A, B)) {
135         cout <<"Matrix␣B␣belongs␣to␣the␣range␣space␣of␣matrix␣A."<<endl;
136         // Perform back substitution and print solutions
137         mod.backSubstitution(A, B);
138     } else {
139         cout <<"Matrix␣B␣does␣not␣belong␣to␣the␣range␣space␣of␣matrix␣A.\n";
140     }
141     return 0
142 }
```

Time Complexity: $O(m \times n^2)$

Output:

```
Enter the prime number p: 5
Enter the dimensions of matrix A (m x n): 3 3
Enter the elements of matrix A:
2 1 2
1 2 2
3 3 1
Enter the elements of matrix B:
1 1  1

Matrix B belongs to the range space of matrix A.

Solutions for AX = B:
x1 = 0
x2 = 4
x3 = 1
```