```
b = 11   e = 12354   p = 7919   q = 5153   n = 40806607

------------ Right to left ---------------

Answer = 31057402
Time = 1064 nano Seconds

------------ Left to right ----------------

Answer = 31057402
Time = 1421 nano Seconds

 ---------------- CRT --------------------

Answer = 31057402
Time = 6363 nano Seconds
```

As shown in the results the fastest algorithm in my implementation is Right to left modular exponentiation then Left to right then CRT.

Time calculated is the average of running the code 50000 times.

```cpp
#include <iostream>
#include <chrono>

using namespace std;
typedef long long   ll;

ll Right_to_Left_Binary_Modular_Exponentiation(ll b, ll e , ll n) {
    ll a = 1;
    ll s = b;
    int number_of_bits = (int)log2(e) + 1;
    for(int i = 0 ; i< number_of_bits; i++)
    {
        if ( e >> i & 1)
            a = (a * s) % n;

        s = (s * s) % n;
    }
    return a;
}

ll Left_to_Right_Binary_Modular_Exponentiation(ll b, ll e, ll n) {

    ll s = b;
    int number_of_bits = (int)log2(e) + 1;
    for (int i = number_of_bits - 2 ; i >=0; i--)
    {
        s = (s * s) % n;
        if (e >> i & 1)
            s = (s * b) % n;
    }
    return s;
}

```

```cpp
ll Extended_GCD(ll a, ll b, ll& x, ll& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }

    ll x1, y1, gcd = Extended_GCD(b, a % b, x1, y1);
    x = y1;
    lldiv_t div = std::lldiv(a, b);
    y = x1 - div.quot * y1;
    return gcd;
}

ll Modulare_Inverse(ll a, ll n) {
    // ax cong 1 (mod n)
    ll x, y;
    Extended_GCD(a, n, x, y);
    if (x < 0)
        x += n;
    return x;
}

ll CRT(ll b, ll e,ll p , ll q , ll n) {

    ll yp = Left_to_Right_Binary_Modular_Exponentiation(b, e, p);
    ll yq = Left_to_Right_Binary_Modular_Exponentiation(b, e, q);
    ll xq = Modulare_Inverse(q, p);
    ll xp = Modulare_Inverse(p, q);
    ll y = (yp * q * xq + yq * p * xp) % n;
    return y;
}
```

```cpp
int main()
{
    ll b = 11, e = 12354, p = 7919, q = 5153, n = 40806607;
    ll ans1, ans2, ans3, time1 = 0 , time2 = 0, time3 = 0;
    for (int i = 0; i < 50000; i++) {

        auto start_time = chrono::steady_clock::now();
        ans1 = Right_to_Left_Binary_Modular_Exponentiation(b, e, n);
        auto end_time = chrono::steady_clock::now();
        time1 += chrono::duration_cast<chrono::nanoseconds>(end_time - start_time).count();

        start_time = end_time;
        ans2 = Left_to_Right_Binary_Modular_Exponentiation(b, e, n);
        end_time = chrono::steady_clock::now();
        time2 += chrono::duration_cast<chrono::nanoseconds>(end_time - start_time).count();

        start_time = end_time;
        ans3 = CRT(b, e, p, q, n);
        end_time = chrono::steady_clock::now();
        time3 += chrono::duration_cast<chrono::nanoseconds>(end_time - start_time).count();

    }
    time1 = time1 / 50000;
    time2 = time2 / 50000;
    time3 = time3 / 50000;
```

```
    cout << "b = " << b << "    e = " << e;
    cout << "    p = " << p << "    q = " << q << "    n = " << n << endl << endl;

    cout << "------------- Right to left ---------------- " << endl << endl;
    cout << "Answer = " << ans1 << endl;
    cout <<"Time = " << time1 << endl << endl;

    cout << "------------ Left to right ----------------- " << endl << endl;
    cout << "Answer = " << ans2 << endl;
    cout << "Time = " << time2 << endl << endl;

    cout << " ----------------- CRT --------------------- " << endl << endl;
    cout << "Answer = " << ans3 << endl;
    cout << "Time = " << time3 << endl << endl;

    return 0;
}

```