## SECURITY LAB 3 – 28TH AUGUST 2024.

## ROLL NO: **21CSE1016**. NAME: **Ketan Kunkalikar**.

// Q1. Write a program to find elements of high order in Zn\* where n is a large integer.

#include <bits/stdc++.h>

using namespace std;

// Function to compute gcd

long long gcd(long long a, long long b) {

    while (b != 0) {

        long long temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

// Function to compute (a^b) % mod

long long power(long long a, long long b, long long mod) {

    long long result = 1;

    a = a % mod;

    while (b > 0) {

        if (b % 2 == 1) {

            result = (result \* a) % mod;

        }

        b = b >> 1;

        a = (a \* a) % mod;

    }

    return result;

}

// Function to compute Euler's totient function

long long eulerTotient(long long n) {

    long long result = n;

    for (long long p = 2; p \* p <= n; ++p) {

        if (n % p == 0) {

            while (n % p == 0) {

                n /= p;

            }

            result -= result / p;

        }

    }

    if (n > 1) {

        result -= result / n;

    }

    return result;

}

// Function to find prime factors of a number

vector<long long> primeFactors(long long n) {

    vector<long long> factors;

    for (long long p = 2; p \* p <= n; ++p) {

        if (n % p == 0) {

            factors.push\_back(p);

            while (n % p == 0) {

                n /= p;

            }

        }

    }

    if (n > 1) {

        factors.push\_back(n);

    }

    return factors;

}

// Function to find elements of high order in Zn\*

vector<long long> findHighOrderElements(long long n) {

    vector<long long> highOrderElements;

    long long phi = eulerTotient(n);

    vector<long long> factors = primeFactors(phi);

    for (long long r = 2; r < n; r++) {

        if (gcd(r, n) != 1) continue; // r must be coprime with n

        bool isHighOrder = true;

        for (long long factor : factors) {

            if (power(r, phi / factor, n) == 1) {

                isHighOrder = false;

                break;

            }

        }

        if (isHighOrder) {

            highOrderElements.push\_back(r);

        }

    }

    return highOrderElements;

}

int main() {

    long long n;

    cout << "Enter a large integer n: ";

    cin >> n;

    if (n <= 1) {

        cout << "n must be greater than 1." << "\n";

        return 1;

    }

    vector<long long> highOrderElements = findHighOrderElements(n);

    cout << "Elements of high order in Z" << n << "\* are: ";

    for (long long elem : highOrderElements) {

        cout << elem << " ";

    }

    cout << "\n";

    return 0;

}

## Explanation:

## This C++ program calculates the elements of high order in the multiplicative group of integers modulo `n` (denoted as `Zn\*`). It includes standard libraries and uses the standard namespace. The program defines several functions: `gcd` to compute the greatest common divisor, `power` to compute modular exponentiation, `eulerTotient` to compute Euler's totient function, and `primeFactors` to find the prime factors of a number. The main function, `findHighOrderElements`, uses these helper functions to identify elements in `Zn\*` that have a high order by checking if they are coprime with `n` and ensuring their order is not a divisor of Euler's totient function of `n`. The `main()` function prompts the user to enter a large integer `n`, computes the high order elements, and prints them.

## Output:

## 

// Q2. Write a program to find the list of cyclic group present within a range (Example: 2000 to 3000).

#include <bits/stdc++.h>

using namespace std;

void toBinary(unsigned num, vector<int> &binaryVec) {

    if (num > 1)

        toBinary(num / 2, binaryVec);

    binaryVec.push\_back(num % 2);

}

int modularExponentiation(int base, int exp, int mod) {

    vector<int> binaryExp;

    toBinary(exp, binaryExp);

    reverse(binaryExp.begin(), binaryExp.end());

    int result = (binaryExp[0] == 1) ? base : 1;

    int currentPower = base;

    for (int i = 1; i < binaryExp.size(); i++) {

        currentPower = (currentPower \* currentPower) % mod;

        if (binaryExp[i] == 1) {

            result = (result \* currentPower) % mod;

        }

    }

    return result;

}

int computeGCD(int a, int b) {

    int minVal = min(a, b);

    while (minVal > 0) {

        if (a % minVal == 0 && b % minVal == 0) {

            break;

        }

        minVal--;

    }

    return minVal;

}

bool isCyclicGroup(int num) {

    vector<int> znStar;

    for (int i = 1; i < num; i++) {

        if (computeGCD(i, num) == 1) {

            znStar.push\_back(i);

        }

    }

    int phi = znStar.size();

    vector<int> divisors;

    for (int i = 1; i <= phi; i++) {

        if (phi % i == 0) {

            divisors.push\_back(i);

        }

    }

    unordered\_map<int, vector<int>> orderMap;

    for (int elem : znStar) {

        for (int div : divisors) {

            if (modularExponentiation(elem, div, num) == 1) {

                orderMap[div].push\_back(elem);

                break;

            }

        }

    }

    return !orderMap[phi].empty();

}

int main() {

    vector<int> cyclicGroups;

    int rangeStart, rangeEnd;

    cout << "Enter the start value of the range: " << endl;

    cin >> rangeStart;

    cout << "Enter the end value of the range: " << endl;

    cin >> rangeEnd;

    for (int i = rangeStart; i <= rangeEnd; i++) {

        if (isCyclicGroup(i)) {

            cyclicGroups.push\_back(i);

        }

    }

    for (int group : cyclicGroups) {

        cout << group << " ";

    }

    return 0;

}

Explanation:

This C++ program calculates the order of an element a in the multiplicative group of integers modulo n (denoted as Zn\*). It includes all standard libraries and uses the standard namespace. The main() function prompts the user to input two integers, n and a, and reads these values. It initializes variables to track the remainder, the order k, and the current result. The program then computes the order by repeatedly multiplying the current result by a and taking the modulo n until the result equals 1, incrementing k with each iteration. Finally, it prints the order of a modulo n, which is the smallest positive integer k such that a^k ≡ 1 (mod n).

Output:

## 

// Q3. Write a program to find the order of an element in Zn\* where n is a large integer.

#include <bits/stdc++.h>

using namespace std;

int main() {

    long long n, a;

    cout << "Enter n and a: ";

    cin >> n >> a;

    long long remainder = 0;

    long long k = 1;

    long long result = a % n;

    while (result != 1) {

        result = (result \* a) % n;

        k++;

    }

    cout << "Order of " << a << " mod " << n << " is " << k << "\n";

    return 0;

}

## Explanation:

## The main() function prompts the user to enter two integers, n and a, and reads these values. It initializes variables remainder, k, and result, and then computes the order of a modulo n by repeatedly multiplying result by a and taking modulo n until result equals 1. Finally, it prints the order of a modulo n. This program calculates the order of an element a in the multiplicative group of integers modulo n (denoted as Zn\*), where the order is the smallest positive integer k such that a^k ≡ 1 (mod n).

## 

// Q4. Write a program to find the quadratic residue and quadratic nonresidue mod n where n is a large integer.

#include <bits/stdc++.h>

using namespace std;

long long gcd(long long a, long long b) {

    while (b != 0) {

        long long temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

vector<long long> Zn\_star\_generator(long long n) {

    vector<long long> Zn\_star;

    for (long long i = 1; i < n; i++) {

        if (gcd(i, n) == 1) {

            Zn\_star.push\_back(i);

        }

    }

    return Zn\_star;

}

set<long long> quadratic\_residue(long long n, vector<long long> Zn\_star) {

    set<long long> quadratic\_residue;

    for (long long i = 0; i < Zn\_star.size(); i++) {

        long long a = Zn\_star[i];

        long long element = (a \* a) % n;

        if(find(Zn\_star.begin(), Zn\_star.end(), element) != Zn\_star.end()) {

            quadratic\_residue.insert(element);

        }

    }

    return quadratic\_residue;

}

int main() {

    long long n;

    cout << "Enter the number: ";

    cin >> n;

    vector<long long> Znstar = Zn\_star\_generator(n);

    set<long long> quad\_res = quadratic\_residue(n, Znstar);

    cout << "Quadratic Residue: ";

    for(long long i : quad\_res) {

        cout << i << " ";

    }

    cout << "\nQuadratic Non-Residue: ";

    for(long long i : Znstar) {

        if(quad\_res.find(i) == quad\_res.end()) {

            cout << i << " ";

        }

    }

    return 0;

}

## Explanation:

## **gcd(long long a, long long b)**: This function computes the greatest common divisor (GCD) of two numbers a and b using the Euclidean algorithm.

## **Zn\_star\_generator(long long n)**: This function generates a list of integers less than n that are coprime with n. It iterates through all integers from 1 to n-1 and includes those whose GCD with n is 1.

## **quadratic\_residue(long long n, vector<long long> Zn\_star)**: This function finds the quadratic residues modulo n from the set of integers coprime with n (generated by Zn\_star\_generator). It squares each element in Zn\_star and checks if the result is also in Zn\_star, then adds it to the set of quadratic residues.

## **main()**: This function prompts the user to enter a value for n, reads this value, computes the set of integers coprime with n, finds the quadratic residues modulo n, and prints both the quadratic residues and non-residues. Quadratic non-residues are those elements in Zn\_star that are not in the set of quadratic residues.

## 

// Q5. Write a program to find the square root of a modulo n where n is a large integer.

#include <bits/stdc++.h>

using namespace std;

long long gcd(long long a, long long b) {

    while (b != 0) {

        long long temp = b;

        b = a % b;

        a = temp;

    }

    return a;

}

vector<long long> Zn\_star\_generator(long long n) {

    vector<long long> Zn\_star;

    for (long long i = 1; i < n; i++) {

        if (gcd(i, n) == 1) {

            Zn\_star.push\_back(i);

        }

    }

    return Zn\_star;

}

set<long long> square\_root\_mod\_n(long long n, long long a) {

    vector<long long> Zn\_star = Zn\_star\_generator(n);

    set<long long> square\_root;

    for (long long i = 0; i < Zn\_star.size(); i++) {

        long long x = Zn\_star[i];

        if(x \* x % n == a) {

            square\_root.insert(x);

        }

    }

    return square\_root;

}

int main() {

    long long n, a;

    cout << "Enter n and a: ";

    cin >> n >> a;

    vector<long long> Znstar = Zn\_star\_generator(n);

    set<long long> square\_root = square\_root\_mod\_n(n, a);

    cout << "Square Root: ";

    for(long long i : square\_root) {

        cout << i << " ";

    }

    return 0;

}

## Explanation:

## **gcd(a, b)**: This function computes the greatest common divisor (GCD) of two numbers a and b using the Euclidean algorithm.

## **Zn\_star\_generator(n)**: This function generates a list of integers less than n that are coprime with n. It iterates through all integers from 1 to n-1 and includes those whose GCD with n is 1.

## **square\_root\_mod\_n(n, a)**: This function finds all integers x in the set of integers coprime with n (generated by Zn\_star\_generator) such that x squared is congruent to a modulo n. It returns these integers as a set.

## **main**: This function prompts the user to enter values for n and a, reads these values, computes the set of integers coprime with n, finds their square roots modulo n, and prints the results.

## 