Task 7: Primality Tests

To solve programming problems which includes the concept of Prime numbers and its related properties.

a. **Sieve of Eratosthenes** with example
b. **Fermat's Primality Testing** with example
c. **Miller-Rabin Primality** Testing with example

a. Sieve of Eratosthenes with example

The **Sieve of Eratosthenes** is an efficient algorithm to find all prime numbers up to a given number **n**.
Steps:

1. Create a list of numbers from **2 to n**.
2. Start with the first prime (2).
3. Eliminate all multiples of 2 (except 2 itself).
4. Move to the next unmarked number (3) → mark its multiples.
5. Repeat until you reach √n.
6. The remaining unmarked numbers are all primes.

🔹 **Example in C**

```
#include <stdio.h>
#include <stdbool.h>

void sieveOfEratosthenes(int n) {
   bool prime[n+1];   // Boolean array to track primes
   for (int i = 0; i <= n; i++)
      prime[i] = true;  // Assume all numbers are prime initially

   prime[0] = prime[1] = false; // 0 and 1 are not prime

   for (int p = 2; p * p <= n; p++) {
      if (prime[p] == true) {
         // Mark all multiples of p as not prime
         for (int i = p * p; i <= n; i += p)
            prime[i] = false;
      }
   }

   // Print prime numbers
   printf("Prime numbers up to %d are:\n", n);
   for (int i = 2; i <= n; i++) {
      if (prime[i])
         printf("%d ", i);
   }
   printf("\n");
}

int main() {
   int n;
```

```
    printf("Enter the limit: ");
    scanf("%d", &n);

    sieveOfEratosthenes(n);

    return 0;
}
```

🔹 **Example Run**

Input:

Enter the limit: 30

Output:

Prime numbers up to 30 are:

2 3 5 7 11 13 17 19 23 29

B. Fermat's Primality Testing with example

Fermat's Little Theorem:

If **p** is prime and **a** is any integer such that 1 < a < p, then

ap−1≡1(modp)a^{p-1} \equiv 1 \pmod{p}ap−1≡1(modp)

Idea:

- Pick a random number a in range [2, p-2].
- Compute ap−1mod pa^{p-1} \mod pap−1modp.
- If result ≠ 1 → **p is composite**.
- If result = 1 for several random values of a → **p is probably prime**.

⚠️ Note: It's a **probabilistic test**. Carmichael numbers may pass even though they are not prime.

🔹 **Example in C**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function for modular exponentiation (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 & 1)
            result = (result * a) % mod;
        b = b >> 1;   // Divide b by 2
        a = (a * a) % mod;
    }
    return result;
}

// Fermat Primality Test
int isPrimeFermat(int n, int k) {
    if (n <= 1 || n == 4) return 0;
    if (n <= 3) return 1;

    // Try k times
```

```
    for (int i = 0; i < k; i++) {
        int a = 2 + rand() % (n - 4); // random number in [2, n-2]
        if (power(a, n - 1, n) != 1)
            return 0;  // composite
    }
    return 1; // probably prime
}

int main() {
    srand(time(0)); // Seed for random numbers

    int n, k;
    printf("Enter number to test: ");
    scanf("%d", &n);
    printf("Enter number of iterations: ");
    scanf("%d", &k);

    if (isPrimeFermat(n, k))
        printf("%d is probably prime.\n", n);
    else
        printf("%d is composite.\n", n);

    return 0;
}
```

🔹 **Example Run**

Input:
Enter number to test: 97
Enter number of iterations: 5
Output:
97 is probably prime.
Input:
Enter number to test: 91
Enter number of iterations: 5
Output:
91 is composite.

C. Miller-Rabin Primality Testing with example C program

Concept: Miller–Rabin Primality Test

It's a **probabilistic test** but much stronger than Fermat's test.

Idea:

1. Write n−1=2s·dn-1 = 2^s \cdot dn−1=2s·d, where `d` is odd.
2. Pick a random number `a` in `[2, n-2]`.
3. Compute x=admod nx = a^d \mod nx=admodn.
4. If x=1x = 1x=1 or x=n−1x = n-1x=n−1, then continue with next iteration.

5. Otherwise, square `x` repeatedly:
    - If you ever get n−1n-1n−1, then continue.
    - If you never get n−1n-1n−1, then `n` is composite.
6. Repeat the test multiple times with different random `a`.
7. If it passes all, `n` is **probably prime**.

Example C Program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function for modular exponentiation (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 & 1)
            result = (result * a) % mod;
        b = b >> 1;
        a = (a * a) % mod;
    }
    return result;
}

// Miller test for a single base 'a'
int millerTest(long long d, long long n) {
    long long a = 2 + rand() % (n - 4);
    long long x = power(a, d, n);

    if (x == 1 || x == n - 1)
        return 1;

    // Keep squaring x until d becomes n-1
    while (d != n - 1) {
        x = (x * x) % n;
        d *= 2;

        if (x == 1)  return 0;  // composite
        if (x == n - 1) return 1;
    }
    return 0; // composite
}

// Miller-Rabin primality test
int isPrimeMillerRabin(long long n, int k) {
    if (n <= 1 || n == 4) return 0;
    if (n <= 3) return 1;

    // Find d such that n-1 = 2^s * d
    long long d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    // Perform k iterations
    for (int i = 0; i < k; i++) {
```

```
        if (!millerTest(d, n))
            return 0;
    }
    return 1;
}

int main() {
    srand(time(0));
    long long n;
    int k;

    printf("Enter number to test: ");
    scanf("%lld", &n);
    printf("Enter number of iterations: ");
    scanf("%d", &k);

    if (isPrimeMillerRabin(n, k))
        printf("%lld is probably prime.\n", n);
    else
        printf("%lld is composite.\n", n);

    return 0;
}
```

Example Run

Input:

```
Enter number to test: 97
Enter number of iterations: 5
```

Output:

```
97 is probably prime.
```

Input:

```
Enter number to test: 221
Enter number of iterations: 5
```

Output:

```
221 is composite.
```