9/25

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 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 it self multiples
5. Repeat until you reach $\sqrt{n}$
6. The remaining unmarked numbers are all primes

```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;
```

9/25

Task - 7B

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 (mod p) a^(P-1)\equiv 1\p mod (P) ap-1 ≡ 1 (mod P)

Idea:

1. Pick a random number a in range (2, P-2).
2. Compute ap-1 mode pa^(P-1)\mod pap-1 mod P.
3. If result ≠ 1 → P is composite.
4. If result = 1 for several random values of a → P is probably prime.

```
#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]
```

```c
    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;
}
```

2/9/25

Task - 7C

Miller - Rabin Primality Testing with example

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

Algorithm

1. write n-1= 2 s*dn-1 = 2^s\c dot dn-1 = 2 s^d , where d is odd
2. Pick a random number a in (2, n-2).
3. Compute x = a d mod nx = a^d\mod nx = ad mod n.
4. If x = 1x = 1x = 1 or x = n-1x = n-1x = n-1, then continue with next iteration.
5. otherwise, square x repeatedly:
   a. If you ever get n-1 n-1 n-1 then continue.
   b. 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.

```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;
    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;
```

```
    }
    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;
}
```

VEL TECH - CSE	
EX NO	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (3)	3
VIVA VOCE (3)	3
RECORD (4)	4
TOTAL (15)	15
SIGN WITH DATE	

Result :-
Thus the program was executed and verified successfully.