-> Prime check and Prime Factorization

-> Brute Force Method to check prime.

-> Time Complexity O(N)

```
BOOL PRIMCHECK (INT n)
{
    BOOL ISPRIME = TRUE;
    -> If n is 1 return false as 1 is a
       non-prime number
    IF (n == 1)
    {
        RETURN FALSE;
    }

    FOR (INT i=2; i<n; i++)
    {
        IF (n % i == 0)
        {
            RETURN FALSE;
        }
    }
    RETURN TRUE;
}
```

-> But we can observe a pattern here, just like in finding divisors, we could use n till SQRT(n), here we can do the same, because if a number is divisible by a number coming before SQRT(n), it will automatically be divisible by numbers coming afterwards.

-> eg: For n = 36

-> 2 → 36/2 == 18 (so if a number is divisible by 2, it will automatically be divisible by 18)

-> 3 → 36/3 == 12

-> 4 → 36/4 == 9

-> 6 -> 36/6 == 6

NOTE: ~~Int~~
-> Square Root Method to check Prime
-> Time Complexity O(SQRT(n)) OR $O(\sqrt{n})$

```
BOOL SQRTPRIMECHK (INT n)
{
    BOOL ISPRIME = TRUE;
    -> If n is 1 returning false as 1 is
       a non prime number.
    IF (n == 1)
    {
        RETURN FALSE;
    }

    FOR (INT i = 2; i * i <= n; i++)
    {
        IF (n % i == 0)
        {
            RETURN FALSE;
        }
    }
    RETURN TRUE;
}
```

-> Prime Factors :

Approach: First we have to find the smallest prime number of a given n, and then we keep on dividing that

n with that prime number, until its is no more divisible, now we have to find this new n's prime number, and follow the same process until n becomes 1.

-> Brute Force Approach
-> Time Complexity O(n)

```
INT n;
CIN n;
VECTOR < INT > PRIME_FACT_B;
FOR ( INT i=2;  i ≤ n;  i++ )
{
    WHILE ( n %. i == 0)
    {
        PRIME_FACT_B . PB (i);
        n /= i ;
    }
}
```

-> As in prime factorization, we always consider the smallest possible prime number which is divisor of n, so, we use n till SQRT(n), because if a number is divisible by a number coming before SQRT(n), it will automatically be divisible by numbers coming afterwards.
This will work fine until we have n composite, but ASA we get n a prime no, we have to apply an if check, bc if n = 3, then SQRT(3) condition will

not work.

-> Square Root Method for prime factorization

-> Time complexity O(SQRT(n)) or $O(\sqrt{n})$

```
INT n;
CIN >> n;
VECTOR<INT> PRIME_FACT_S;
FOR (INT i=2; i* i <= n; i++)
{
    WHILE (n % i == 0)
    {
        PRIME_FACT_S.PB(i);
        n /= i;
    }
}
IF (n > 1)
{
    PRIME_FACT_S.PB(n);
}
```