-> nth Root using Binary Search

-> LOGIC

Suppose we have to find the sq. root of 4.

Then in Binary Search our search space will be from 1 to 4

1 2 3 4

(Now, we will apply Binary search and the condition will be if MID x MID < x then LO = MID ELSE, HI = MID) {x = 4 here}

And if we want decimal places as well then our search space will increase e.g:

For x = 2 if we want 1.4 as ans then our search space will be:

1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2

It's time complexity will be:

$p \times \log(N \times 10^d)$ (For pth root of number N with d decimal places)

CODE

```
DOUBLE EPS = 1e-7; // precision of our ans.

-> function to multiply our no till nth root.
DOUBLE MULTIPLY (DOUBLE MID, INT N)
{
    DOUBLE ANS = 1;
    FOR (INT i=0; i< N; i++)
    {
        ANS *= MID;
    }
    RETURN ANS;
}

VOID NTH_BINARY (DOUBLE x, INT N)
{
    DOUBLE LO = 1, HI = x, MID;
    -> eg: here, EPS = 0.00000001, so,
      it will determine the accuracy
    WHILE (HI - LO > EPS)
    {
        MID = (HI + LO) / 2;
        IF (MULTIPLY (MID, N) < x)
        {
            LO = MID;
        }
        ELSE
        {
            HI = MID;
        }
```

```
}
```

→ Now we can print LO or HI any, they both will give same ans till given precision of 10^{-7} here.

```
COUT << LO << " " << HI << "\n";
```

→ Finding n^{th} root using inbuilt POW function.

```
COUT << POW(x, 1.0/n) << "\n";
}

INT MAIN()
{
```

→ no. for which we have to find the root of

```
DOUBLE x;
```

→ value of n in the n^{th} root

```
INT N;
CIN >> x >> N;
NTH_BINARY(x, N);
RETURN 0;
}
```

NOTE: Predicate function → A function which returns TRUE/FALSE.

Binary search is applied on Monotonic predicate function. e.g.:

F F F T T T

T T T F F F