For NP, we defined it as the class of decision problems for each of which there exists a polynomial-time non-deterministic algorithm.

ProdTwoPrimes(n):

> **if** prime_checking(n) **then return** false
>
> prod ← false
>
> Non-deterministically pick x and y ∈ {2, …, n-1}
>
> **If** n = x * y **and** prime_checking(x) **and** prime_checking(y) **then** prod ← true
>
> **return** prod

Let ProdTwoPrimes be the algorithm for the decision problem "given integer n ≥ 2, is n the product of exactly two primes", since prime_checking is in **P**, the algorithm is a polynomial-time non-deterministic algorithm, hence ProdTwoPrimes is in **NP**.

Now, by tweaking this algorithm a little bit, invert its output,

Co_ProdTwoPrimes(n):

> **if** prime_checking(n) **then return** false
>
> prod ← false
>
> Non-deterministically pick x and y ∈ {2, …, n-1}
>
> **If** n = x * y **and** prime_checking(x) **and** prime_checking(y) **then** prod ← true
>
> **return not** prod

This makes the algorithm Co_ProdTwoPrimes to output exactly the opposite output of ProdTwoPrimes. And since prime_checking is in **P**, this algorithm is a polynomial-time non-deterministic algorithm, hence it's also in **NP**. Since both the algorithms are in **NP**, the problem ProdTwoPrimes is in **NP** ∩ **co-NP**.