FFT Algorithm

NOTE: Procedure FFT is presented here in pseudo-code, for a generic field F in which it is possible to define ω, a primitive n-th root of unity.

```
procedure FFT (A, n, w)

    # Preconditions:
    #   A is a Vector of length n;
    #   n is a power of 2;
    #   w is a primitive n-th root of unity.
    #
    # The Vector A represents the polynomial
    #   a(z)  =  A[1] + A[2]*z + ... + A[n]*z^(n-1) .
    #
    # The value returned is a Vector of the values
    #    [ a(1), a(w), a(w^2), ... , a(w^(n-1)) ]
    # computed via a recursive FFT algorithm.

    if n = 1 then
        return A
    else
        Aeven <-- Vector( [A[1], A[3], ..., A[n-1]] )
        Aodd  <-- Vector( [A[2], A[4], ..., A[n]] )

        Veven <-- FFT( Aeven, n/2, w^2 )
        Vodd <-- FFT( Aodd, n/2, w^2 )

        V <-- Vector(n)  # Define a Vector of length n
        for i from 1 to n/2 do
            V[i] <-- Veven[i] + w^(i-1)*Vodd[i]
            V[n/2 + i] <-- Veven[i] - w^(i-1)*Vodd[i]
        end do
        return V
    end if
end procedure
```

Additional comments

- For computational efficiency, in the for-loop build up the powers of `w` using just one multiplication each pass through the loop. Similarly for the recursive FFT calls, `w^2` should be computed only once.
- If the computation is in the field $\mathbf{Z}_p$, each arithmetic operation will be performed `mod p`. In Maple, globally assign `` `mod` := 'mods' `` to get "symmetric representation" (see the help page `?mod` in Maple).