Question 4 (Part A):

```
> restart:
> maple_f_z := Interp([1,2,3,4],[0,5,5,0],z) mod 7;
```

$$maple\_f\_z := z^2 + 2z + 4 \quad (1)$$

```
> printf("\nInputs:\n"):
  x := [1,2,3,4];
  y := [0,5,5,0];
  p := 7;

Inputs:
```

$$x := [1, 2, 3, 4]$$
$$y := [0, 5, 5, 0]$$
$$p := 7 \quad (2)$$

Step 1:

```
> (* Expanding the product M(z) *)

  m_z := 1:
  for i from 1 to nops(x) do
      m_z := m_z * (z - x[i]):
  od:
  m_z := expand(m_z) mod 7;
```

$$m\_z := z^4 + 4z^3 + 6z + 3 \quad (3)$$

Step 2:

```
> (* Solving L_i(z) = M(z) / (z - x_i) for i = 1 to n *)

  l_z_i := Array(1..nops(x)):
  for i from 1 to nops(x) do
      l_z_i[i] := expand(Divide(m_z, z - x[i], 'q')) mod 7:
      l_z_i[i] := q:
      printf("\nL[%a] = %a.\n", i, l_z_i[i]):
  od:

L[1] = z^3+5*z^2+5*z+4.

L[2] = z^3+6*z^2+5*z+2.

L[3] = z^3+6.
```

```
L[4] = z^3+z^2+4*z+1.
```

Step 3:

```
> (* Set alpha_i := L_i(x_i) *)

  alpha := Array(1..nops(x)):
  for i from 1 to nops(x) do
      alpha[i] := eval(l_z_i[i], z = x[i]) mod 7:
      printf("\nalpha[%a] = %a.\n", i, alpha[i]):
  od:

alpha[1] = 1.

alpha[2] = 2.

alpha[3] = 5.

alpha[4] = 6.
```

Step 4:

```
> (* Set beta_i := y_i * inverse(alpha_i) *)

  beta := Array(1..nops(x)):
  for i from 1 to nops(x) do
      beta[i] := y[i] * (1/alpha[i]) mod 7:
      printf("\nbeta[%a] = %a.\n", i, beta[i]):
  od:

beta[1] = 0.

beta[2] = 6.

beta[3] = 1.

beta[4] = 0.
```

Step 5:

```
> (* Set f = sum(beta_i * L_i(z)) for i from 1 to n *)

  interp_poly := 0:
  for i from 1 to nops(x) do:
      interp_poly := interp_poly + (beta[i] * l_z_i[i]) mod 7:
```

```
od:

unique_interp_poly := interp_poly;
```

$$unique\_interp\_poly := z^2 + 2\,z + 4 \quad \textbf{(4)}$$

Question 4 (Part B):

```
> lag_interpolation := proc(x_val, y_val, id_var, prime_p)

  local x, y, z, p, m_z, l_z, i, n, a, b, f_z, int_pol:
  x := x_val:
  y := y_val:
  z := id_var:
  p := prime_p:
  m_z := z - x[1] mod p:
  n := nops(x):
  l_z := Array(1..n):
  a := Array(1..n):
  b := Array(1..n):

  for i from 2 to n do
      m_z := expand(m_z * (z - x[i])) mod p:
  od:

  for i from 1 to n do
      l_z[i] := expand(Divide(m_z, z - x[i], 'q')) mod p:
      l_z[i] := q:
      printf("\nL[%a] = %a.\n", i, l_z[i]):
  od:

  for i from 1 to n do
      a[i] := eval(l_z[i], z = x[i]) mod p:
  od:

  for i from 1 to n do
      b[i] := y[i] * (1/a[i]) mod p:
  od:

  f_z := 0:
  for i from 1 to n do:
      f_z := f_z + (b[i] * l_z[i]) mod p:
  od:

  int_pol := f_z:
```

```
  return printf("\nInterpolating Polynomial: %a.", int_pol):
  end proc:
> lag_interpolation([1,2,3,4], [0,5,5,0], z, 7);b :

L[1] = z^3+5*z^2+5*z+4.

L[2] = z^3+6*z^2+5*z+2.

L[3] = z^3+6.

L[4] = z^3+z^2+4*z+1.

Interpolating Polynomial: z^2+2*z+4.
```