1.

3.9.a. Using lexicographical order with $z > y > x$, we have

$$\begin{aligned} \text{LT}(y - x^2) &= y, \\ \text{LT}(z - x^3) &= z. \end{aligned}$$

Since every term in r must not be divisible by the leading terms of any of the divisors, this means y, z does not appear in any terms, and r is a polynomial in x.

3.9.b.

$$z^2 - x^4 y = (t^3)^2 - (t)^4 (t^2) = t^6 - t^6 = 0.$$

3.9.c.

$$z^2 - x^4 y = (-x^4)(y - x^2) + (z + x^3)(z - x^3).$$

The code below is an implementation of the division algorithm in JavaScript and is also available at
https://gist.github.com/pillowfication/81b0cba89fae5839265a8761750d364d.

```
const ALPHABET = 'abcdefghijklmnopqrstuvwxyz'

function gcd (a, b) {
  return b === 0 ? a : b > a ? gcd(a, b % a) : gcd(b, a % b)
}

class Fraction {
  constructor (str) {
    if (Array.isArray(str)) {
      return this.simplify(...str)
    }

    if (!/[0-9]/.test(str)) {
      this.simplify(+`${str}1`, 1) // Parses '', '+', '-' into 1, 1, -1
    } else {
      this.simplify(...(/\//.test(str) ? str : `${str}/1`).split('/').map(Number))
    }
  }

  add (frac) {
    return this.simplify(this.p * frac.q + this.q * frac.p, this.q * frac.q)
  }

  times (frac) {
    return this.simplify(this.p * frac.p, this.q * frac.q)
  }

  divide (frac) {
    return this.simplify(this.p * frac.q, this.q * frac.p)
  }

  equals (frac) {
```

```
    return this.p * frac.q === this.q * frac.p
  }

  simplify (p = this.p, q = this.q) {
    if (p === 0) {
      [this.p, this.q] = [0, 1]
    } else {
      if (q < 0) {
        [p, q] = [-p, -q]
      }
      const d = gcd(Math.abs(p), q)
      ;[this.p, this.q] = [p / d, q / d]
    }
    return this
  }

  toString () {
    return this.q === 1 ? `${this.p}` : `${this.p}/${this.q}`
  }

  clone () {
    return new Fraction([this.p, this.q])
  }
}

class Monomial {
  constructor (str) {
    if (Array.isArray(str)) {
      [this.coeff, this.power] = str
      return
    }

    //              (           coeff             )(        powers         )|( constant  )
    const regex = /^([+-]?(?:[0-9](?:\/[0-9]+)?)?)((?:[a-z](?:\^[0-9]+)?)+)|([+-]?[0-9]+)$/
    const [, coeff, powers, constant] = str.match(regex)
    if (constant !== undefined) {
      this.coeff = new Fraction(constant)
      this.power = Array(ALPHABET.length).fill(0)
    } else {
      this.coeff = new Fraction(coeff)
      this.power = (() => {
        const power = Array(ALPHABET.length).fill(0)
        for (const [, varName, exp] of powers.matchAll(/([a-z])(?:\^([0-9]+))?/g)) {
          power[ALPHABET.indexOf(varName)] = exp === undefined ? 1 : +exp
        }
        return power
      })()
    }
  }

  times (mono) {
    this.coeff.times(mono.coeff)
    this.power = this.power.map((p, i) => p + mono.power[i])
    return this
  }

  divide (mono) {
    this.coeff.divide(mono.coeff)
    this.power = this.power.map((p, i) => p - mono.power[i])
    return this
  }

  clone () {
    return new Monomial([this.coeff.clone(), this.power.slice()])
  }
}

class Polynomial {
  constructor (str) {
    if (Array.isArray(str)) {
      this.monomials = str
      return this.simplify()
    }

    this.monomials = [...str.matchAll(/[+-]?([0-9](\/[0-9]+)?)?([a-z](\^[0-9]+)?)+|[+-]?[0-9]+/g)]
```

```
      .map(match => new Monomial(match[0]))
    this.simplify()
  }

  add (poly) {
    this.monomials = this.monomials.concat(poly.monomials)
    return this.simplify()
  }

  times (poly) {
    this.monomials = poly.monomials
      .map(polyMono => this.monomials.map(thisMono => thisMono.clone().times(polyMono)))
      .reduce((acc, curr) => acc.add(new Polynomial(curr)), new Polynomial('0'))
      .monomials
    return this
  }

  sort (order) {
    this.monomials.sort(order)
    return this
  }

  simplify () {
    // Aggregate like powers and remove zeroes
    for (let i = 0; i < this.monomials.length; ++i) {
      const curr = this.monomials[i]
      for (let j = i + 1; j < this.monomials.length; ++j) {
        if (curr.power.join(',') === this.monomials[j].power.join(',')) {
          curr.coeff.add(this.monomials[j].coeff)
          this.monomials.splice(j--, 1)
        }
      }
      if (curr.coeff.p === 0) {
        this.monomials.splice(i--, 1)
      }
    }
    return this
  }

  toString () {
    if (this.monomials.length === 0) {
      return '0'
    } else {
      return this.monomials.map(({ coeff, power }, i) => (
        (coeff.equals(new Fraction('1'))
          ? (i === 0 ? '' : '+')
          : coeff.equals(new Fraction('-1')) ? '-' : `+${coeff.toString()}`) +
        power.reduce((acc, curr, i) =>
          curr > 0 ? curr > 1 ? acc + `${ALPHABET[i]}^${curr}` : acc + ALPHABET[i] : acc
        , '')
      )).join('')
    }
  }

  clone () {
    return new Polynomial(this.monomials.map(mono => mono.clone()))
  }
}

function divides (a, b) {
  for (let i = 0; i < ALPHABET.length; ++i) {
    if (a.power[i] > b.power[i]) {
      return false
    }
  }
  return true
}

function polynomialDivision (poly, f, order = LEX) {
  poly = (new Polynomial(poly)).sort(order)
  f = f.map(f => (new Polynomial(f)).sort(order))

  const q = Array(f.length).fill().map(_ => new Polynomial('0'))
  const r = new Polynomial('0')
  const p = poly.clone()
```

```
  while (p.monomials.length > 0) {
    let i = 0
    let divisionoccurred = false
    while (i < f.length && divisionoccurred === false) {
      const LT_p = p.monomials[0] // eslint-disable-line camelcase
      const LT_fi = f[i].monomials[0] // eslint-disable-line camelcase
      if (divides(LT_fi, LT_p)) {
        q[i].add(new Polynomial([LT_p.clone().divide(LT_fi)]))
        p.add((new Polynomial('-1')).times(new Polynomial([LT_p.clone().divide(LT_fi)])).times(f[i]))
        divisionoccurred = true
      } else {
        i = i + 1
      }
    }
    if (divisionoccurred === false) {
      r.add(new Polynomial([p.monomials[0]]))
      p.add((new Polynomial('-1')).times(new Polynomial([p.monomials[0]])))
    }
  }
  return { q: q.map(q => q.sort(order).toString()), r: r.sort(order).toString() }
}

// Standard sorting functions
function LEX (a, b) {
  for (let i = 0; i < ALPHABET.length; ++i) {
    if (a.power[i] !== b.power[i]) {
      return b.power[i] - a.power[i]
    }
  }
  return 0
}

function GRLEX (a, b) {
  const aSum = a.power.reduce((a, c) => a + c, 0)
  const bSum = b.power.reduce((a, c) => a + c, 0)
  return aSum === bSum ? LEX(a, b) : bSum - aSum
}

//
//
//

console.log(polynomialDivision('z^2-x^4y', ['y-x^2', 'z-x^3'], (a, b) => -LEX(a, b)))
// > { q: [ '-x^4', 'z+x^3' ], r: '0' }
```

2.

3.6. Let

$$g = 3x \cdot f_1 - 2y \cdot f_2 = -3x^2 + 2y^2 + 2y.$$

None of the terms in g are divisible by the leading terms of f_1, f_2, so the result of r after the division algorithm is g itself.

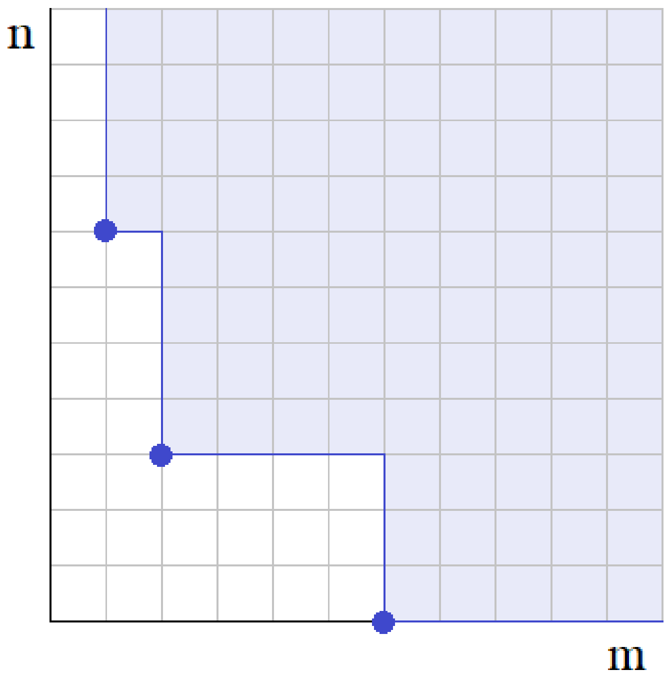
3.

5.2. Note that xy divides both $\mathrm{LT}(f_1)$ and $\mathrm{LT}(f_2)$, so xy divides every element of $\langle \mathrm{LT}(f_1), \mathrm{LT}(f_2)\rangle$. But xy does not divide $\mathrm{LT}(g) = -3x^2$. Thus

$$\mathrm{LT}(g) \in \mathrm{LT}(I) \not\subseteq \langle \mathrm{LT}(f_1), \mathrm{LT}(f_2)\rangle.$$

4.

4.3.a.



4.3.b. The terms in the remainder must not be elements of I.

5.

5.17.a. Let $f_1 = x^2 - y$ and $f_2 = y + x^2 - 4$. Then

$$\begin{aligned} x^2 - y &= f_1, \\ x^2 - 2 &= \tfrac{1}{2} f_1 + \tfrac{1}{2} f_2, \end{aligned}$$

so $\langle x^2 - y, x^2 - 2 \rangle \subseteq I$. Similarly,

$$\begin{aligned} f_1 &= x^2 - y, \\ f_2 &= -1 \cdot (x^2 - y) + 2 \cdot (x^2 - 2), \end{aligned}$$

so $I \subseteq \langle x^2 - y, x^2 - 2 \rangle$.

5.17.b. Let $(x, y) \in \mathbf{V}(I)$. Since $x^2 - 2 \in I$, this means either $x = \sqrt{2}$ or $x = -\sqrt{2}$. And since $x^2 - y \in I$, then $y = x^2 = 2$. Thus $\mathbf{V}(I) \subseteq \{(\pm\sqrt{2}, 2)\}$, and because $\{x^2 - y, x^2 - 2\}$ is a basis for I, $\mathbf{V}(I) = \{(\pm\sqrt{2}, 2)\}$.