Background

In this problem, we will consider different representations for the “IntPoly” ADT, which represents a polynomial with integer coefficients. To do this, we will use the “IntTerm” ADT below, which represents a monomial (a polynomial with one term) with an integer coefficient.

Here is a partial implementation of these ADTs:

```
/**
 * An IntTerm is a mutable representation of a monomial with an integer
 * coefficient, i.e., something of the form A x^b, where A and b are integers
 * satisfying A != 0 and b >= 0. A is called the coefficient of this term,
 * while b is called the exponent.
 */
public class IntTerm {

  // RI: this.coeff != 0 and this.expt >= 0.
  // AF(this) = a monomial Ax^b, where A = this.coeff and B = this.expt.
  private int coeff;
  private int expt;

 ... methods including getCoeff() and getExpt() ...
}


// ... code continues on the next page ...
```

```
/**
 * An IntPoly is a mutable representation of a polynomial with integer
 * coefficients. A polynomial is a sum of monomials.
 */
public class IntPoly {

   // RI: ?? (You will give possible answers for this in the next page.)
   // AF(this) = the sum of all IntTerms in the list this.terms.
   private List<IntTerm> terms;

   /**
    * @param expt the exponent of the term being observed
    * @spec.requires expt >= 0
    * @return the coefficient of the term with exponent “expt”
    *         or 0 if no such term
    */
   public int coeff(int expt) { … }

   /**
    * @return the biggest exponent with a nonzero coefficient
    *         or 0 if this = 0
    */
   public int degree() { … }

   /**
    * @return the number of nonzero coefficients
    */
   public int size() { … }

   /**
    * @return a list containing all the terms with nonzero coefficients
    */
   public List<IntTerm> getTerms() { … }

   /**
    * @param t A monomial to be added to this polynomial
    * @modifies this
    * @spec.effects Adds t to the polynomial
    */
   public void addTerm(IntTerm t) { … }


   ...
}
```

Representation Invariants

There are multiple possible representation invariants we could write for IntPoly. Given the following versions, write a suitable representation invariant (RI) for the ADT:

a) No restrictions on the IntTerms in the list “terms” (except, as always, no nulls).

b) A version where there can be no two terms with the same exponent.

c) A version where the terms are ordered by exponent, in strictly descending order (i.e., with adjacent exponents satisfying “>” not “>=”).

Concrete Representations

For each of the following concrete examples of *this.terms*, a list of IntTerms stored in an IntPoly, note which representation invariants above (a,b,c) are satisfied by it. Then, write which of these examples correspond to an equal abstract state (meaning they represent the same polynomial, if they represent a valid polynomial).

Here, we will use the pair (a, b) as shorthand for the monomial Ax^b.

A. [(5, 4), (1, 0)]

B. [(5, 4), (2, 0), (-1, 0)]

C. [(3, 2), (2, 3)]

D. [null, (1, 0), (5, 4)]

E. [(1, 3), (1, 3), (2, 2), (1, 2)]

F. [(1, 0), (2, 4), (3, 4)]

Choosing Representation Invariants

When implementing methods of an ADT, certain invariants can make the coding easier, while some may lead to more complicated solutions. For each method in the IntPoly ADT, state which RIs above (a,b,c) would make implementing that method easiest. Include a short explanation for each method. Here, multiple answers may be possible, so you will be graded based on your explanation.

Here are the methods included with IntPoly: “coeff”, “degree”, “size”, “getTerms”, and “addTerm”.

Representation Exposure

Which methods of IntPoly have a *potential risk* of representation exposure? For each such method, what does the implementation need to do to ensure representation exposure does not happen?

In our specification, we denoted IntTerm is a mutable class. If we were to change it to an immutable class, how would your answer to the question above change? Would all methods you mentioned still have a potential risk of representation exposure? What does the implementation need to do now to ensure representation exposure does not happen?