1.1. Escribir una especificación informal para los distintos métodos de ordenación que conozcas. La especificación debe ser genérica, es decir trabajar con elementos de cualquier tipo. ¿En qué se diferencia la especificación de los distintos métodos?

1.2. Desarrollar una especificación informal genérica para el TAD árbol binario. Incluir operaciones para crear y modificar el árbol.

1.3. Suponiendo que tenemos una implementación del tipo árbol binario, ¿sería posible comprobar su validez utilizando especificaciones? ¿De qué tipo? ¿Nos valdría la desarrollada en el ejercicio anterior?

1.4. (TG 2.1) Desarrollar la especificación formal del TDA conjunto de tipo T (con las secciones **Nombre**, **Conjuntos**, **Sintaxis** y **Semántica**), por el método axiomático o algebraico. La especificación debe ser completa, todas las operaciones deben estar definidas en la parte de semántica. Incluir las operaciones: Vacío, EsVacío, Inserta, Suprime, Miembro, Unión, Intersección, Diferencia y Cardinalidad (número de elementos del conjunto).

1.5. (TG 2.3) Partiendo de la especificación formal para el TAD de los números naturales vista en clase, añadir a la especificación las operaciones: **predecesor**, **producto** y **potencia**. Tener en cuenta que pueden ocurrir casos donde el resultado no sea un número natural.

1.6. (TG 2.4) Con la especificación anterior, comprobar el resultado que se obtiene para las siguientes expresiones:
   a) igual (predecesor (sucesor (cero)), sucesor (predecesor (cero)))
   b) producto (sucesor (sucesor (cero)), sucesor (sucesor (cero)))

1.7. Evaluar el resultado de las siguientes expresiones, usando la especificación formal por el método axiomático del TAD Pila[T]:
   a) tope (pop (push (a, push (b, push (b, crearPila)))))
   b) esVacía (pop (push (a, pop (push (x, crearPila)))))
   c) esVacía (push (4, pop (crearPila)))

1.8. (EX) Para una aplicación que utiliza conjuntos de palabras, representados mediante listas, definimos la especificación formal del TDA **Conjunto_palabras** por el método axiomático. Indicar qué partes de la especificación formal del TDA deben cambiar si en otra aplicación queremos representar los conjuntos mediante otra estructura.

1.9. (TG 2.6) Proponer algún modelo subyacente adecuado, para escribir la especificación formal por el método constructivo del TAD Conjunto[Elemento]. Escribir la parte de semántica para las operaciones del ejercicio 4.

1.10. ¿Qué diferencias existen entre un TAD mutable y uno no mutable? ¿Es posible desarrollar una especificación formal algebraica de un TAD mutable? ¿Por qué? ¿Y si la especificación formal es constructiva u operacional?

1.11. (EX) En la especificación formal del tipo **Array(elemento)**, tenemos una operación **Ordena: A → A**, que dado un array, devuelve sus elementos ordenados de mayor a menor. Escribir la parte de semántica para esta operación, por el método constructivo u operacional. Suponer que existen otras operaciones del tipo:

Tamaño: Array → Entero
ObtenValor: Array x Entero → Elemento
PonValor: Array x Entero x Elemento → Array
MayorQue: Elemento x Elemento → Booleano

1.12. Repetir el ejercicio anterior utilizando especificaciones formales axiomáticas, y suponiendo las operaciones que creas convenientes. ¿Cómo es representado el hecho de que el resultado es un array ordenado?

1.13. (EX) Para desarrollar la especificación formal del TAD grafo dirigido y etiquetado disponemos de los siguientes conjuntos:

G	Conjunto de grafos dirigidos y etiquetados
M	Conjunto de nodos de los grafos
E	Conjunto de valores de las etiquetas en las aristas
N	Conjunto de naturales
B	Conjunto de booleanos
U	Conjunto de mensajes de error

Escribir la parte de sintaxis correspondiente a las operaciones que son los constructores del tipo. Poner también la sintaxis de alguna operación de modificación y otra de consulta sobre el grafo.

1.14. (TG 2.8) Decir qué resultado se obtiene para las siguientes expresiones (mostrando los pasos para la obtención del resultado), utilizando la especificación por el método constructivo del TAD Cola(elemento):

a) frente (inserta (4, inserta (5, crearCola)))
b) esVacíaCola (resto (inserta (a, crearCola)))
c) inserta (frente (crearCola), crearCola)

1.15. ¿Qué ocurre en una especificación formal por el método constructivo si en una expresión no se cumple la precondición? ¿Puede ocurrir algo parecido en una especificación por el método algebraico?

1.16. (EX) En la especificación formal del TAD GrafoNoDirigido, por el método algebraico, tenemos definidos los siguientes constructores del tipo:

GrafoVacío: → G *// Crea un grafo vacío*
InsertaArista: $GxN_1xN_2 \rightarrow G$ *// Añade la arista (N_1, N_2) al grafo*

Mostrar las fórmulas que deberían aparecer en la parte semántica para la siguiente operación de consulta, que comprueba si la arista (N_1, N_2) pertenece al grafo o no:

ExisteArista: $GxN_1xN_2 \rightarrow B$

Se suponen los conjuntos:
G: Conjunto de grafos no dirigidos
N: Conjunto de nodos ($N=N_1=N_2$)
B: Conjunto de valores booleanos= {true, false}

1.17. (EX) Sea un TAD cualquiera A, definido mediante una especificación formal algebraica. ¿Qué propiedad cumplen las operaciones que son los constructores del tipo? ¿Qué otros tipos de operaciones pueden existir?

1.18. (EX) Suponiendo la especificación formal algebraica vista en clase para el TAD **Entero** (con las operaciones: *cero*, *sucesor*, *esCero*, *igual*, *suma*), escribir la sintaxis y la semántica correspondientes a una operación *esMenor*, que comprueba si un entero es menor (estrictamente) que otro.

1.19. (EX M01) Considerar la siguiente especificación formal algebraica del TDA **CMT** (contador módulo 3). Las operaciones definidas son: **Nuevo: → CMT**; **Inc: CMT → CMT**; **Dec: CMT → CMT**.
Los axiomas de la parte de semántica son los siguientes: $\forall$ c $\in$ CMT

1) Inc (Dec (c)) = c
2) Dec (Inc (c)) = c
3) Inc (Inc (Inc (c))) = c
4) Dec (Dec (Dec (c))) = c

Demostrar, usando los axiomas de la especificación formal (indicar el número de los que se usen), que se cumple la siguiente propiedad:
Dec (Nuevo) = Inc (Inc (Nuevo))

1.20. (EX S01) Dado el siguiente trozo de la especificación formal del TDA grafo de naturales:

NOMBRE
Grafo de naturales
CONJUNTOS
G conjunto de grafos de naturales
N conjunto de números naturales
B conjunto de valores booleanos {true, false}
SINTAXIS
vacio: → G
esvacio: G → B
insertarnodo: G x N → G
SEMÁNTICA
$\forall$ g $\in$ G, $\forall$ n $\in$ N
1. esvacio (vacio) = true
2. esvacio (insertarnodo (vacio, n)) = false
3. esvacio (insertarnodo (g, n)) = esvacio (g)
4. insertarnodo (vacio, vacio) = vacio
5. insertarnodo (g, n) = insertarnodo (n, g)

corregir la parte **SEMÁNTICA** de la especificación, para que se corresponda con el concepto usual de grafo (teniendo en cuenta que la especificación no es completa). Para las reglas eliminadas o modificadas, decir (en 1 línea como máximo) la razón por la que están mal.

1.21. (TG 2.2) A la especificación del ejercicio 4 añadir las operaciones **máximo(C)** y **mínimo(C)** (para calcular el mayor y el menor elemento de un conjunto) y **sucesor(C, n)** y **predecesor(C, n)** (para calcular el elemento de **C** más próximo a **n**, por arriba o por abajo, respectivamente).

1.22. Usando la especificación anterior, escribir la expresión correspondiente a crear un conjunto con los elementos {12, 5, 2}. Sobre ese conjunto, aplicar las operaciones **máximo**, **mínimo**, **sucesor (C, 22)**, **predecesor (C, 6)** y deducir el resultado usando los axiomas.

1.23. (EX) Dada una especificación formal algebraica del tipo **árbol binario** que contenga las operaciones **crear_arbol**, **construir** (dados dos árboles y un entero, devuelve un árbol cuya raíz contiene el entero y tiene como hijos izquierdo y derecho los dos árboles pasados como argumentos), **hijo_izq**, **hijo_der**, **elemento** (dado un árbol, devuelve el entero que contiene la raíz) y **es_vacío**, añadir una operación **espejo** que devuelva un árbol binario que sea igual a la imagen reflejada en un espejo respecto al eje de simetría vertical del árbol original. **Ojo:** el ejercicio es mucho más fácil de lo que parece.

1.24. (EX) Queremos construir una especificación formal del tipo **Lista[Elementos]** que contenga la operación **invertirLista**. Escribir la sintaxis de las operaciones incluidas en la especificación y escribir los axiomas referentes a la operación **invertirLista**.

1.25. Construir la especificación formal algebraica del TDA **Persona**, que está formada por un registro con por tres campos: **nombre** (de tipo cadena), **dirección** (de tipo cadena) y **teléfono** (de tipo entero). Incluir operaciones de consulta y modificación de cada uno de ellos. Mostrar varias expresiones de ejemplo, donde se establezcan varios valores y luego se consulten algunos de ellos. ¿Cuáles son los constructores del tipo?

1.26. (EX D01) Construir una especificación formal axiomática para el TAD **Array**[**E]**. Los índices del array son enteros y el rango de índices del array no está limitado. Las operaciones sobre el TAD son: *Nuevo*, *PonValor*, *ObtenValor* (para escribir o leer un valor en una posición del array, respectivamente) y *Maximo* (para obtener el índice máximo de array). Adoptar las decisiones que creas adecuadas para los posibles casos de error. ¿Cuáles son los constructores del tipo?

1.27. (EX M02) Tenemos definido el TAD **Pantano**, por el método axiomático, con las operaciones:

Nuevo: N → Pantano
Devuelve un pantano con capacidad máxima N y cantidad actual de agua 0

Llenar: Pantano → Pantano
Pone la cantidad actual de agua del pantano al valor de capacidad máxima

Cantidad: Pantano → N
Devuelve la cantidad actual de agua del pantano
Transvasar: Pantano x N → Pantano ∪ Error
Decrementa la cantidad actual en N, siempre que sea posible

Añadimos la operación **Ocupación: Pantano → R** (que devuelve el porcentaje de ocupación actual del pantano). Mostrar los nuevos axiomas que aparecen al incluir esta operación. Si se cree conveniente se pueden añadir otras operaciones.

1.28. (EX S02) Queremos definir el TAD genérico **GrafoDirigido[Elemento]** por el método axiomático, con las operaciones: **GrafoVacio** (crea un nuevo grafo sin vértices), **InsArista** (añade una arista al grafo, con los dos vértices pasados como parámetros), **ExisteArista** (comprueba si existe una arista en el grafo) y **GradoEntrada** (devuelve el grado de entrada de un vértice dado). Escribir la especificación formal axiomática del tipo (con las partes: *Nombre, Conjuntos, Sintaxis* y *Semántica*). Tener en cuenta lo siguiente:

- Los vértices son del tipo genérico **Elemento**.
- No existe una operación para insertar vértices, ya que se supone que son añadidos implícitamente en la operación de insertar aristas (si no existen ya).
- Si se inserta una arista que ya existe, no ocurre nada ya que sólo puede existir una arista entre un par de vértices.

1.29. Construir una especificación formal axiomática para el TAD **ArbolBinario[Elemento]**. El tipo debe contener las operaciones: *CrearArbol*, *Construir* (dados dos árboles *I* y *D*, y un elemento *x*, devuelve un nuevo árbol donde *x* es la raíz y los subárboles izquierdo y derecho son *I* y *D*, respectivamente), *Raíz* (devuelve la raíz), *HijoIzq* (devuelve el hijo izquierdo), *HijoDer* y *EsVacío*.

1.30. (EX) Partimos de las especificaciones formales algebraicas del TAD **Lista[Elemento]**, vista en clase, y **ArbolBinario[Elemento]**, del ejercicio anterior. Añadir a este último las operaciones *OrdenPrevio*, *OrdenSimétrico* y *OrdenPosterior*, que dado un árbol como argumento, devuelven una lista que contiene todos los elementos del árbol ordenados según su recorrido en orden previo, simétrico y posterior, respectivamente.

1.31. (EX D02) Definimos el TAD **ColaCíclica[Elemento]** como una cola FIFO usual (primero en entrar, primero en salir), pero en la cual al sacar un elemento de la cabeza, se vuelve a meter en la cola en la última posición. Las operaciones del tipo son: **Crear**, **Meter** (mete un elemento en la última posición de la cola), **Cabeza** (devuelve el elemento que está en la cabeza de la cola) y **Avanzar** (el elemento de la cabeza pasa al final de la cola). Escribir una especificación formal axiomática del tipo **ColaCíclico[Elemento]**. Si se cree conveniente se pueden incluir otras operaciones (en cuyo caso se deberán especificar también).

1.32. (TG 2.7, EX M03) A la definición formal del TAD **Conjunto[T]** (con las operaciones *Vacío, EsVacío, Inserta, Miembro, Suprime, Unión, Intersección*, etc.) queremos añadir las operaciones: **EsSubcjto**, **EsSubPropio** y **EsIgual**, que dados dos conjuntos comprueban si uno es subconjunto del otro, si uno es subconjunto propio del otro o si ambos son iguales, respectivamente. Escribir la sintaxis y la

semántica de las operaciones, usando alguna de las técnicas formales de especificación vistas en clase.

1.33. (EX S03) Definimos el TAD **Sensor** con las siguientes operaciones: *Crear*, *Más*, *Menos* y *Estado*. La operación *Crear* recibe un entero, que es el "tope" del sensor, y devuelve un sensor nuevo. Las operaciones *Más* y *Menos* reciben un sensor y devuelven otro sensor. La operación *Estado* es de consulta: devuelve ON si el número de veces que se ha aplicado *Más*, menos el número de veces que se ha aplicado *Menos* es mayor o igual que el tope del sensor; en caso contrario devuelve OFF. Escribir una especificación formal, algebraica o constructiva, del tipo **Sensor**.

1.34. (EX M04, EX J04 y EX S04) Suponer que tenemos las especificaciones formales algebraicas de los TAD **Natural**, **Booleano** y **ArbolBinario[T]**, con las siguientes operaciones:

- **N=Natural**: *cero*, *sucesor*, *esCero*, *suma*, *resta*.
- **L=Lista[T]**: *vacía*, *insertar* (dada una lista y un valor de tipo **T**, lo inserta en la primera posición), *esVacía*, *concatenar* (concatena dos listas).
- **A=ArbolBinario[T]**: *crear* (devuelve un árbol vacío), *construir* (dados los parámetros *(t, a_1, a_2)*, donde *t* es de tipo **T**, a_1 y a_2 de tipo **A**, crea un árbol en el que *t* es la raíz, y a_1 y a_2 los subárboles izquierdo y derecho, respectivamente).

Escribir la semántica de las siguientes operaciones:

a) **Nivel: A x N → L**
Nivel(a, n) devuelve una lista con los elementos del árbol *a* que se encuentren a nivel *n*. Se supone que la raíz del árbol tiene nivel 0, los hijos nivel 1, los nietos 2, ...

b) **RSD: A → A**
RSD(a) devuelve el árbol resultante de aplicar una rotación simple a la derecha de la raíz de *a*. En caso de que no sea posible debe devolver *"Error"*.

c) **NumHojas: A → N**
NumHojas(a) devuelve el número de hojas del árbol *a*.

d) **esABB: A → Booleano**
esABB(a) devuelve TRUE si el árbol *a* es un árbol binario de búsqueda y FALSE en caso contrario. Se supone que disponemos de la operación "<" para comparar dos elementos de tipo **T**.

e) **peorAVL: N x T → A**
peorAVL(n, t) devuelve el peor caso de AVL para altura *n* (es decir, un árbol AVL con el menor número de nodos posible para altura *n*) donde todos los nodos contienen el valor *t*.

f) **cuentaVeces: A x T→ N**
cuentaVeces(a, t) devuelve un natural que indica el número de veces que aparece el elemento *t* en el árbol *a*. Se supone que disponemos de la operación **esIgual** para comparar dos elementos de tipo **T**.

g) **quitaRaíz: A → A**
quitaRaíz(a) devuelve un árbol con los mismos elementos que el árbol *a*, excepto el de la raíz, que es eliminado. La nueva raíz puede ser cualquiera de los otros elementos de *a*.

1.35. (EX M05) Escribir una especificación formal, usando el método axiomático, del TAD **Tabla de dispersión abierta**, en la que se almacenarán elementos que son pares de (*clave*, *valor*), con las siguientes operaciones:

a. **Crear**: crea una tabla con un determinado número de cubetas mayor que cero.
b. **Insertar**: mete un elemento en la tabla.
c. **EsVacia**: comprueba si la tabla está vacía.
d. **Tamaño**: devuelve el número de cubetas de la tabla.
e. **Cuántos**: indica cuántos elementos hay en la tabla.
f. **Consulta**: devuelve el valor asociado a la clave de un elemento en una tabla.
g. **Suprimir**: elimina un elemento (*clave, valor*) de la tabla, dada la clave.

Hay que tener en cuenta que la inserción de un elemento cuya clave ya está equivale a sustituir su valor por el nuevo, y que si eliminamos un elemento, su clave ya no estará en la tabla ni ningún valor asociado a ella. Se supone la especificación del TAD **Natural** vista en clase. Escribir las 4 partes de la especificación del TAD **Tabla de dispersión** (nombre, conjuntos, sintaxis y semántica).

1.36. (EX J05) Queremos especificar formalmente el TAD **Carro** que representa un carro de la compra en el que podemos meter productos, sacarlos, consultar cuántos llevamos y saber el precio de la compra. Suponer que tenemos ya definidos los TAD **Natural**, **Booleano** y **Producto**, con las siguientes operaciones:

- **N=Natural**: *cero*, *sucesor*, *esCero*, *esMenor*, *suma*, *resta*, *multiplica*.
- **P=Producto**: *esIgual* (comprueba si dos productos son iguales o no), *precio* (dado un producto, devuelve un natural que será su precio). Este TAD tendrá constructores constantes como *manzanas*, *peras*, *patatas*, etc.

Las operaciones del TAD **Carro** son las siguientes: **carroVacío** (devuelve un carro sin productos), **meter** (recibe un carro, un producto y un natural que indica el número de veces que se mete ese producto), **sacar** (dado un carro, un producto y un natural, saca el producto el número indicado de veces), **cuántos** (dado un carro y un producto, devuelve un natural que indica el número de veces que está ese producto), **precio** (dado un carro, calcula el coste total de la compra), **cajaRápida** (dado un carro, devuelve un booleano indicando si el número de productos distintos es menor que 5). Escribir la sintaxis y la semántica del TAD.

1.37. (EX S05) Suponer que tenemos las especificaciones formales de los TAD **Lista[T]** y **Natural**, con las siguientes operaciones:

- **N=Natural**: *cero*, *sucesor*, *esCero*, *esMenor*, *suma*, *resta*, *multiplica*, *doble*, *mitad*.
- **Lista[T]**: *vacía* (devuelve una lista sin elementos), *inserta* (dada una lista y un elemento, añade ese elemento en la primera posición de la lista).

Queremos añadir las siguientes operaciones al TAD **Lista[T]**: **sacaPares** (dada una lista cualquiera, devuelve otra lista en la que se ha eliminado el 2º elemento, el 4º, el 6º, etc., es decir, todos los elementos en posiciones pares), **primeraMitad** (dada una lista con *k* elementos, devuelve otra lista con los elementos 1º, 2º, ..., hasta el k/2-ésimo), **segundaMitad** (dada una lista con *k* elementos, devuelve otra lista con los elementos k/2+1-ésimo, k/2+2-ésimo, ..., hasta el k-ésimo). Escribir la sintaxis y la semántica de las operaciones añadidas. Se pueden incluir otras operaciones, pero será necesario especificarlas también. Nota: usar *mitad(k)* para obtener *k*/2.

1.38. (EX F06) Partiendo de la especificación del tipo **Natural** vista en clase, escribir una especificación formal del tipo **Matriz_de_naturales** con las siguientes operaciones: **crear** (crea una matriz, inicializada a 0, de tamaño UxV; este tamaño es fijo, por lo que no hace falta incluirlo en la especificación), **asignar** (sustituye el valor de una celda, dada la fila, la columna y el nuevo valor), **obtener** (obtiene el valor de una posición dada de la matriz), **sumar** (suma dos matrices de dimensión UxV), **produtoEscalar** (multiplica cada elemento de la matriz por un natural dado) y **máximoHistórico** (devuelve el máximo valor asignado en algún momento a cualquier posición de la matriz). Se puede suponer la existencia de las operaciones **sumarN** y **multiplicarN** sobre los naturales.

1.39. (EX) Escribir una especificación formal, usando el método axiomático, del TAD **Grafo dirigido**, suponiendo los siguientes tipos y operaciones predefinidos:

- **Natural**: cero, esCero, sucesor, suma, esIgual, resta.
- **Conjunto**: vacío, esVacío, inserta, suprime, esMiembro, cardinalidad, unión, intersección, diferencia, etc.

Los nodos del grafo serán de tipo genérico **T**. Sólo hay una operación para insertar aristas. Se supone que los nodos se insertan al insertar una arista que los contenga. De esta manera, las operaciones que queremos definir son las siguientes:

- **crear**: devuelve un grafo sin nodos.
- **insArista**: dado un par de nodos y un grafo, inserta la arista correspondiente.
- **existeArista**: consulta si existe o no una arista en un grafo.
- **existeNodo**: consulta si existe o no un nodo en un grafo.
- **gradoEntrada**: devuelve el grado de entrada de un nodo del grafo.
- **nodos**: dado un grafo, devuelve un conjunto con los nodos del grafo.
- **raíces**: devuelve un conjunto con todos los nodos que no tienen predecesores en el grafo.

Escribir las 4 partes de la especificación del TAD (nombre, conjuntos, sintaxis y semántica).

1.40. (EX) Escribir una especificación formal, usando el método axiomático, del TAD **lista**, con al menos las siguientes operaciones:

- **colocar**: dada una lista *c*, un natural *n* y un elemento *e*, devolvería la lista resultante de insertar en *c* el elemento *e* en la posición *n* desplazando los siguientes. Si la lista tiene *n* o menos elementos, se inserta por el final.
- **obtener**: dado un natural *n* y una lista, devuelve el elemento que se encuentra en la posición *n* de la lista dada.
- **longitud**: devuelve el número de elementos de una lista.

Añadir los constructores y las operaciones que consideres necesarias, especificándolas de forma adecuada.