

Apellidos, Nombre:

DNI:

## Examen PED abril 2014

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación <i>palindromo</i> (sobre un vector) vista en clase es la siguiente:  Var v: vector; i,x: natural; <i>palindromo(crear_vector()) = VERDADERO</i> <i>palindromo(asig(v,i,x)) = si i&lt;= 50</i> entonces si recu(v,100-i+1) == x entonces palindromo(asig(v,i,x)) sino FALSO sino VERDADERO	<input type="checkbox"/>	<input type="checkbox"/>	1    F
Una operación del TAD X que tenga la sintaxis Crear() →X es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2    V
En C++, al hacer layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	<input type="checkbox"/>	<input type="checkbox"/>	3    F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4    V
El algoritmo de intercambio directo o burbuja estudiado en clase (ordenación de los elementos de un vector) tiene una complejidad promedio de $\Theta(n^2)$ , siendo n el número de elementos del vector.	<input type="checkbox"/>	<input type="checkbox"/>	5    V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.	<input type="checkbox"/>	<input type="checkbox"/>	6    V
La complejidad temporal en el peor caso para la inserción de un elemento en una lista ordenada y en otra no ordenada, que no permiten elementos repetidos, siempre es lineal con el número de elementos en ambos casos.	<input type="checkbox"/>	<input type="checkbox"/>	7    V
El tipo de datos vector (visto en clase) se define como un conjunto en el que sus componentes ocupan posiciones consecutivas de memoria.	<input type="checkbox"/>	<input type="checkbox"/>	8    F
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:  Var i,d:arbin; x:item; <i>nodos(crear_arbin())=0</i> <i>nodos(eraizar(i,x,d))=nodos(i)+nodos(d)</i>	<input type="checkbox"/>	<input type="checkbox"/>	9    F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/>	10    V
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	11    V
A los árboles generales también se les llama árboles multicamino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	12    F
Las rotaciones que hay que realizar en los árboles AVL para mantenerlos balanceados tienen un coste temporal (en su peor caso) lineal con respecto al número de items del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	13    F
El grado de los árboles AVL puede ser +1, 0 ó -1.	<input type="checkbox"/>	<input type="checkbox"/>	14    F

Apellidos, Nombre:  
DNI:

## Examen PED marzo 2015

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En la especificación algebraica, las operaciones constructoras se clasifican en generadoras y modificadoras.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes:  VAR x, y: natural; suma(x, cero) = x suma(cero, x) = x suma(x, suc(y)) = suma(x, y)	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dentro de la especificación algebraica de los números naturales, definimos la sintaxis de la función F como: F: natural $\rightarrow$ BOOL, y su semántica como: F(cero)=TRUE, F(suc(cero))=FALSE, F(suc(suc(x)))=F(x). Para el número natural x=35, la función F devolvería FALSE. Nota: se asume que x=35 es la forma simplificada de indicar x=suc(suc(.....suc(cero).....))).	<input type="checkbox"/>	<input type="checkbox"/>	3	V
Todo árbol binario de altura 9 y 511 nodos es un árbol binario lleno y además es árbol binario de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	4	F
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:	<input type="checkbox"/>	<input type="checkbox"/>	5	F
TPosicion TLista::Primera()	<input type="checkbox"/>	<input type="checkbox"/>	6	F
{ TPosicion p; p.pos = primero; return p; }	<input type="checkbox"/>	<input type="checkbox"/>	7	V
En el método Primera, se invoca de forma implícita a los constructores de TPosicion y TLista.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
En C++, si la variable p es un puntero a un objeto, entonces la expresión p.f() es sintácticamente correcta.	<input type="checkbox"/>	<input type="checkbox"/>	9	V
La complejidad temporal del siguiente fragmento de código es O(n)	<input type="checkbox"/>	<input type="checkbox"/>	10	V
int i, j, n, sum; for (i = 4; i < n; i++) { for (j = i-3, sum = a[i-4]; j <= i; j++) sum += a[j]; cout << "La suma del subarray " << i-4 << " es " << sum << endl; }	<input type="checkbox"/>	<input type="checkbox"/>	11	V
La mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo "n" la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	12	F
Es posible reconstruir un único árbol binario de búsqueda a partir de su recorrido en postorden	<input type="checkbox"/>	<input type="checkbox"/>	13	V
El máximo número de nodos en un nivel i-1 de un árbol binario es $2^{i-2}$ , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	14	F
Un camino en un árbol es una secuencia $a_1, \dots, a_s$ de árboles tal que para todo $i \in \{1, \dots, s-1\}$ , $a_{i+1}$ es subárbol de $a_i$ .				
El grado de un árbol es el máximo nivel que pueden tener sus subárboles.				
La operación desencolar vista en clase es la siguiente:				
<b>VAR</b> c: cola, x: item; desencolar( crear( ) ) = crear( ) <b>si</b> esvacia( c ) <b>entonces</b> desencolar( encolar( c, x ) ) = crear( ) <b>si no</b> desencolar( encolar( c, x ) ) = encolar( desencolar( c ), x )				
El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda siempre se encuentra en la raíz.				

**Apellidos, Nombre:**

**DNI:**

# **Examen PED junio 2018**

## **Modalidad 0**

**Normas:**

- Tiempo para efectuar el test: **25 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual. Este test vale 2 puntos (sobre un total de 10 de la nota de Teoría).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
Utilizando la especificación algebraica de los números naturales vista en clase, la sintaxis y la semántica de la operación de <i>división</i> que calcula el cociente de dos números naturales es la siguiente (Nota: se asume que para <i>resta</i> y <i>división</i> el primer argumento es mayor que el segundo y que el cociente de la división es entero y exacto): <i>resta, division</i> : natural , natural → natural VAR x, y: natural; $resta(x, cero) = x$ $resta(suc(x), suc(y)) = resta(x, y)$ $division(x, cero) = error\_natural$ $division(x, x) = suc(cero)$ $division(x, y) = division(resta(x, y), y)$	<input type="checkbox"/>	<input type="checkbox"/>	1 F
En C++, un destructor devuelve 0 si el objeto se destruye correctamente y un valor distinto de 0 en caso contrario.	<input type="checkbox"/>	<input type="checkbox"/>	2 F
La función de búsqueda BINARIA de un elemento en un vector desordenado tiene una complejidad temporal $O(\log_2 n)$ .	<input type="checkbox"/>	<input type="checkbox"/>	3 F
La operación <i>subconjunto_impares</i> que se aplica sobre un conjunto de números naturales, definida a continuación, devuelve el subconjunto formado por los números impares que existen en el conjunto original (Nota: Se asume que está definida la operación MOD para calcular el resto de la división entera): <i>subconjunto_impares</i> : conjunto → conjunto Var C: conjunto; x: natural; $subconjunto\_impares(crear\_conjunto()) = crear\_conjunto()$ $subconjunto\_impares(Insertar(C, x)) =$ $\quad \text{si } (x \text{ MOD } 2 == suc(cero)) \text{ entonces Insertar}(subconjunto\_impares(C), x)$ $\quad \text{si no } subconjunto\_impares(C)$	<input type="checkbox"/>	<input type="checkbox"/>	4 V
Se puede reconstruir un único árbol binario de búsqueda teniendo su recorrido en preorden.	<input type="checkbox"/>	<input type="checkbox"/>	5 V
Un árbol binario de búsqueda con 3 elementos siempre será un árbol completo.	<input type="checkbox"/>	<input type="checkbox"/>	6 F
Un árbol binario de búsqueda equilibrado respecto a la altura tiene una complejidad temporal en su peor caso en la operación de búsqueda de $O(\log_2(n))$ , con $n$ el número de elementos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	7 V
En la operación de borrado de un elemento en un árbol AVL, si se realiza una rotación doble siempre decrece la altura del subárbol sobre el que se ha realizado la rotación.	<input type="checkbox"/>	<input type="checkbox"/>	8 V
La altura de un árbol 2-3 únicamente crece cuando se inserta un elemento y todos los nodos del árbol son 3-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	9 F
Dado un árbol 2-3 con $n$ ítems con todos sus nodos del tipo 3-nodo: la complejidad temporal en su peor caso de la operación de búsqueda de un ítem es $O(\log_3(n+1))$ .	<input type="checkbox"/>	<input type="checkbox"/>	10 V
Todo árbol binario completo es un árbol 2-3-4 donde todos sus nodos son 2-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	11 F
En el borrado de un elemento en un árbol 2-3-4 sólo se reduce la altura del árbol cuando $p, q$ y $r$ son 2-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
La complejidad temporal en su peor caso de la diferencia de dos conjuntos de cardinalidad “n”, representados como listas ordenadas, es de $O(n)$ .	<input type="checkbox"/>	<input type="checkbox"/>	13 V
La dispersión abierta elimina el problema del clustering secundario.	<input type="checkbox"/>	<input type="checkbox"/>	14 V
Cuando utilizamos una tabla de dispersión abierta de tamaño B, el número máximo de elementos que se pueden almacenar está limitado a B elementos.	<input type="checkbox"/>	<input type="checkbox"/>	15 F
La definición de un Heap Mínimo indica que ha de ser un árbol binario que además es árbol mínimo.	<input type="checkbox"/>	<input type="checkbox"/>	16 F
Una aplicación de los Grafos Acíclicos Dirigidos es la representación de órdenes parciales.	<input type="checkbox"/>	<input type="checkbox"/>	17 V

# Examen PED junio 2018

Normas:

- ♦ Tiempo para efectuar el examen: **2 horas**
- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
- Cada pregunta se escribirá en hojas diferentes.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Cada pregunta vale 1,5 puntos (sobre un total de 10 de la nota de Teoría).**
- Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.

**1.** Definir la sintaxis y la semántica de la operación *completo* que actúa sobre un árbol binario de números naturales e indica si es un árbol binario completo.

*Nota: se pueden utilizar todas las operaciones definidas en clase para el TAD arbin y se asume que está definido el TAD de los números naturales con las operaciones de suma, resta, multiplicación y división.*

**2. a)** Indica cuál de las siguientes representaciones mediante un vector podría corresponder a un HEAP o a un DEAP y justifica tu respuesta. (Las etiquetas A1, A2, ... son números enteros).

a) 

A1	A2	A3	A4	A5
----	----	----	----	----

b) 

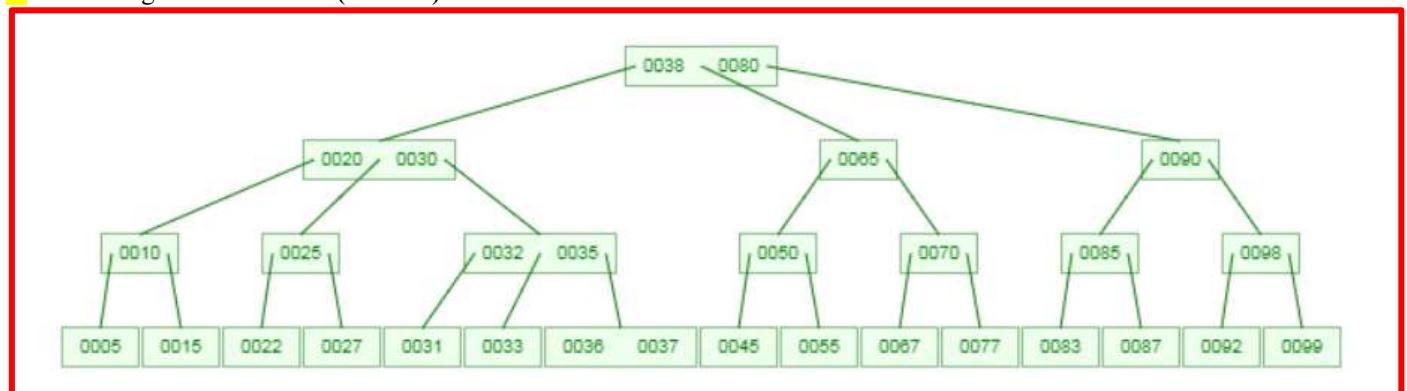
A1		A2		A3
----	--	----	--	----

c) 

	A1	A2	A3	A4
--	----	----	----	----

**b)** Inserta en un DEAP inicialmente vacío los números del 1 al 10 de menor a mayor. Obtén su representación mediante un vector. Tras la inserción elimina el máximo indicando todos los pasos necesarios y obtén su representación mediante un vector.

**3. Dado el siguiente árbol 2-3 (árbol A):**



**a)** Realiza el borrado del ítem **80** del árbol A, indicando número y tipo de transformaciones utilizadas. Criterio 1: Si el ítem a borrar no está en una hoja, sustituir por el mayor de la izquierda. Criterio 2: Consultar el hermano de la izquierda. (NOTA: dibujar un árbol después de cada transformación realizada –combinación o rotación–).

**b)** Del árbol original A, realizar un recorrido por niveles. Según ese recorrido, insertar en un árbol 2-3-4 inicialmente vacío los 14 primeros elementos del recorrido por niveles realizado previamente. (NOTA: dibujar el estado del árbol en estos puntos: Después de las 8 primeras inserciones; después de las siguientes 6 inserciones).

**4. Dada la siguiente función que ordena un vector:**

```

ORD (a:vector[natural]; n: natural)
var i,j: entero; fvar
comienzo
  para i:=2 hasta n hacer
    para j:=n hasta i hacer
      si a[j]<a[j-1] entonces
        SWAP(a[j],a[j-1])
      fsi
    fpara
  fpara
fin
  
```

**a)** Obtened razonadamente el **polinomio o función** que expresa la complejidad **temporal** (número de pasos) en su caso **mejor y peor**.

**b)** Obtened razonadamente la **cota superior** de complejidad **temporal** en su caso **mejor y peor**.

**c)** Obtened razonadamente la cota superior de complejidad **espacial** en su caso **mejor y peor**.

## Examen PED junio 2018. Soluciones

### 1. Sintaxis:

completo(arbin)  $\rightarrow$  bool

Semántica:

Var i, d: arbin; x, y: natural;

$[enraizar(i,x,d) \rightarrow E(i,x,d)]$

completo (crear\_arbin() ) = TRUE

completo(E(crear\_arbin(),x,crear\_arbin()) ) = TRUE

completo(E(E(crear\_arbin(),x,crear\_arbin()), y, crear\_arbin()) ) = TRUE

completo(E(i,x,d)) =

si (altura(i) – altura(d) == cero) O (altura(i) – altura(d) == suc(cero))

entonces completo (i) Y completo(d)

sino FALSE

### 2.

a) Podría ser un HEAP máximo (mínimo) porque el vector representa un árbol binario completo y además debe cumplir que A1 es mayor/menor que A2, A3, A4 y A5 y debe cumplir que A2 es mayor/menor que A4 y A5. Además, todas las etiquetas deben ser diferentes. No representa un DEAP porque la raíz no está vacía. b) No representa un HEAP ni un DEAP porque no es un árbol binario completo. c) No representa un HEAP porque la raíz está vacía. Puede representar un DEAP porque la raíz es vacía y es un árbol binario completo. Y además debe cumplir que A1, A3, A4 son menores que A2 y debe cumplir que A1 es menor que A3 y A4. Además, todas las etiquetas deben ser diferentes.

b)

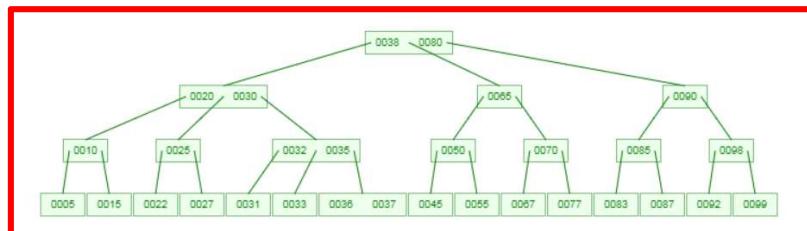
	1	10	2	3	7	9	4	6	5	8
--	---	----	---	---	---	---	---	---	---	---

	1	9	2	3	7	8	4	6	5	
--	---	---	---	---	---	---	---	---	---	--

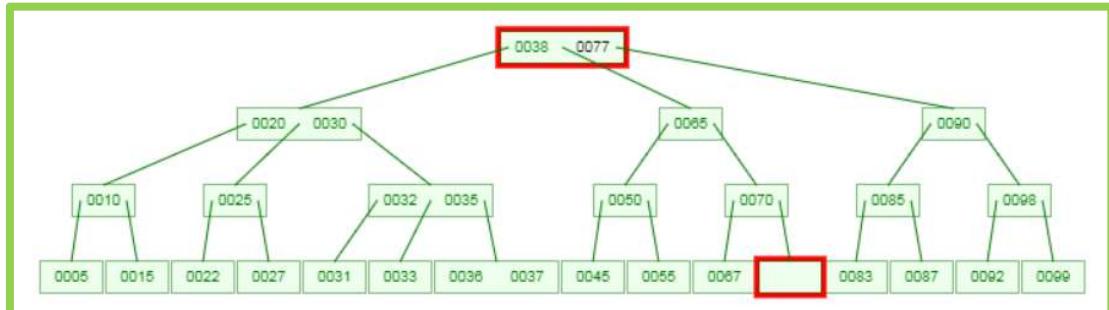
### 3.

A)

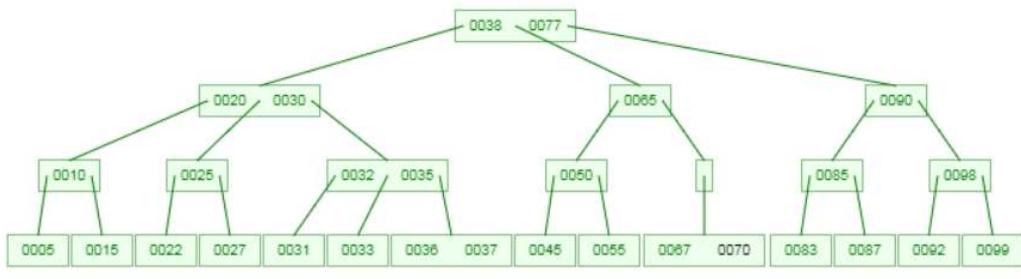
**INICIAL**



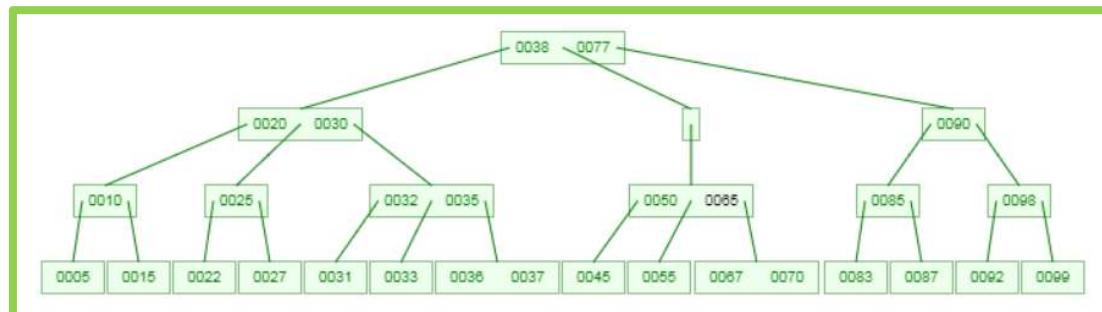
SUSTITUCION



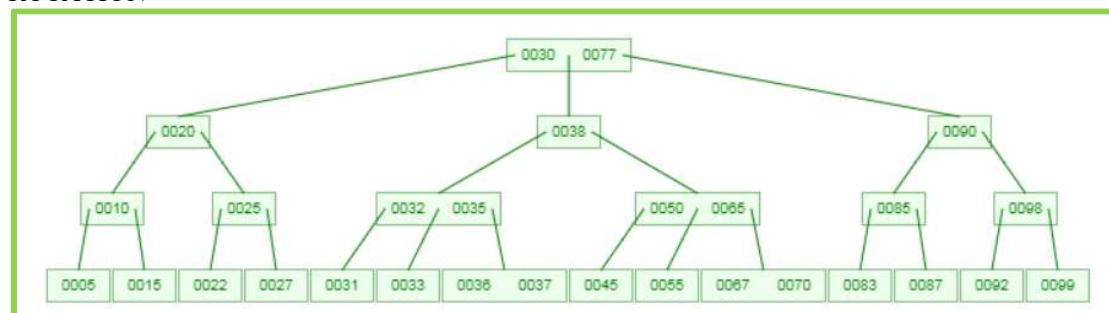
COMBINACION 1



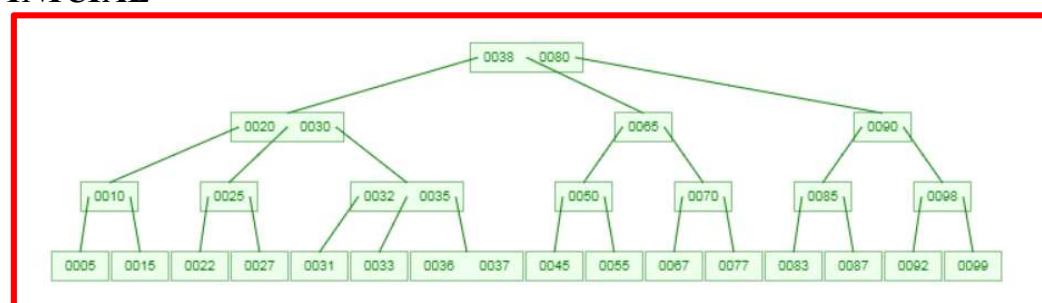
COMBINACION 2



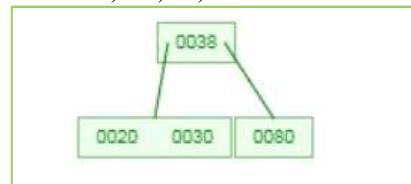
ROTACION



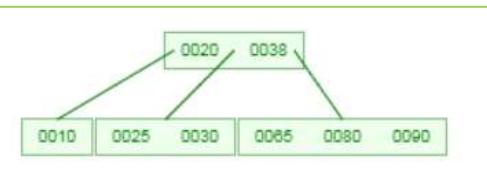
**B)**  
**INICIAL**



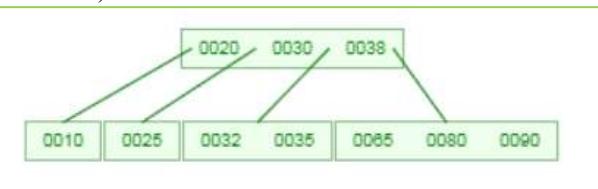
TRAS 38, 80 , 20, 30:



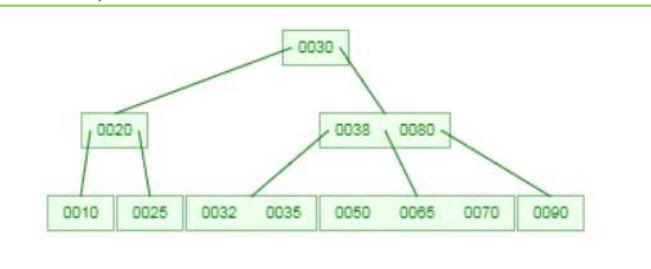
TRAS 65, 90 , 10, 25:



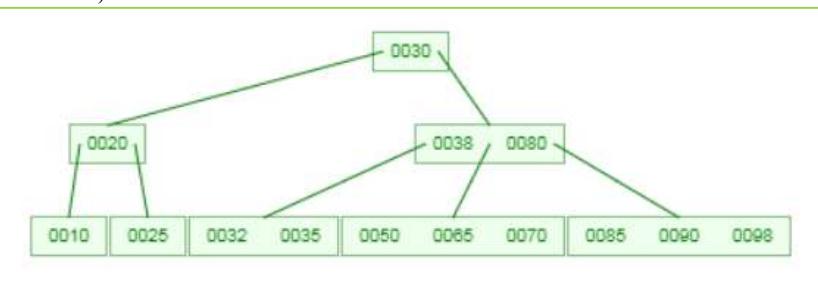
TRAS 32, 35:



TRAS 50, 70:



TRAS 85, 98:



4.

- a) Caso mejor: vector ordenado ascendentemente, por lo que en el **si** no entra nunca, pero sí representaría un coste de 1 por la comprobación de la condición del **si**.

$$\sum_{i=2..n} \sum_{j=n..i} 1 = \sum_{i=2..n} (n - i + 1) = (n-1) + (n-2) + \dots + 1 = \sum_{i=1..n-1} i = (n-1) * n / 2 \in \Omega(n^2)$$

Caso peor: vector ordenado descendente, por lo que en el **si** entra en cada iteración del bucle. De todos modos, el polinomio coincidiría con el expresado para el caso mejor, ya que el coste del **si** sería de 1.

- b) Caso mejor  $\in \Omega(n^2)$ . Caso peor  $\in O(n^2)$ . En ambos casos coincide al ser  $n^2$  el monomio de mayor grado de su polinomio correspondiente.

- c)  $\Omega(n) = O(n)$  Ya que en ambos casos se necesitará un solo vector, concretamente el que la función recibe como parámetro de tamaño **n**.

**Apellidos, Nombre:**

**DNI:**

# **Examen PED junio 2019**

## **Modalidad 0**

**Normas:**

- Tiempo para efectuar el test: **27 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual. Este test vale 2 puntos (sobre un total de 10 de la nota de Teoría).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

		<b>V</b>	<b>F</b>	
<input type="checkbox"/>	<input type="checkbox"/>	1	V	
<input type="checkbox"/>	<input type="checkbox"/>	2	F	
<input type="checkbox"/>	<input type="checkbox"/>	3	F	
<input type="checkbox"/>	<input type="checkbox"/>	4	V	
<input type="checkbox"/>	<input type="checkbox"/>	5	V	
<input type="checkbox"/>	<input type="checkbox"/>	6	V	
<input type="checkbox"/>	<input type="checkbox"/>	7	V	
<input type="checkbox"/>	<input type="checkbox"/>	8	F	
<input type="checkbox"/>	<input type="checkbox"/>	9	V	
<input type="checkbox"/>	<input type="checkbox"/>	10	F	
<input type="checkbox"/>	<input type="checkbox"/>	11	V	
<input type="checkbox"/>	<input type="checkbox"/>	12	V	
<input type="checkbox"/>	<input type="checkbox"/>	13	V	
<input type="checkbox"/>	<input type="checkbox"/>	14	V	
<input type="checkbox"/>	<input type="checkbox"/>	15	F	
<input type="checkbox"/>	<input type="checkbox"/>	16	F	
<input type="checkbox"/>	<input type="checkbox"/>	17	F	
<input type="checkbox"/>	<input type="checkbox"/>	18	V	
<input type="checkbox"/>	<input type="checkbox"/>	19	F	
<input type="checkbox"/>	<input type="checkbox"/>	20	V	

En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras.

La operación *base*, vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente:  
**base(pila) -> item**

```
Var p: pila; x: item;
base(crear()) = error_item()
base(apilar(crear(),x)) = x
base(apilar(p,x)) = base(desapilar(p))
```

La función de búsqueda BINARIA de un elemento en una lista ordenada enlazada tiene una complejidad temporal  $O(\log_2 n)$ .

crear\_pila(), apilar(pila,item) y desapilar(pila) son operaciones constructoras del tipo pila.

El mínimo número de nodos que ha de tener un árbol binario de altura 4 para ser equilibrado respecto a la altura es 7

En los árboles binarios de búsqueda, en el borrado de un elemento que tiene dos hijos, la siguiente ecuación de la especificación algebraica vista en clase indica que dicho elemento se sustituye por el menor del subárbol derecho:

si ( $y==x$ ) y no esvacío( d ) y no esvacío( i ) entonces  
borrar( enraizar( i, x, d ), y ) =  
enraizar( i, min( d ), borrar( d, min( d ) ) ) fsi  
min( crea\_arbin() ) = error\_item()  
si esvacío( i ) entonces min( enraizar( i, x, d ) ) = x  
si no min( enraizar( i, x, d ) ) = min( i ) fsi

Dados los recorridos de preorder, postorden y niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.

Dado cualquier ítem de un árbol AVL, el mayor del subárbol izquierdo de ese ítem siempre será un nodo hoja.

Dado cualquier ítem de un árbol 2-3, el mayor del subárbol izquierdo de ese ítem siempre será el ítem anterior en el recorrido en inorden de dicho árbol.

Dado un árbol 2-3 de altura  $h$  con  $n$  items con todos sus nodos del tipo 2-Nodo: la complejidad temporal de la operación de búsqueda de un ítem es  $O(\log_3 n)$ .

Si  $h$  es la altura de un árbol 2-3-4 con  $n$  elementos se cumple que:  $\log_4(n+1) \leq h \leq \log_2(n+1)$ .

En un árbol 2-3-4 todos los nodos no hoja, tienen al menos dos hijos.

En los conjuntos representados como listas ordenadas, la complejidad temporal de la operación “diferencia de conjuntos” de cardinalidad “n” es de  $O(n)$ .

La complejidad temporal de la búsqueda de un elemento en un conjunto de cardinalidad “n”, representado como una lista, es  $O(n)$ .

El factor de carga de la dispersión abierta siempre está entre 0 y 1.

En la dispersión cerrada sólo se producen colisiones entre claves sinónimas.

El máximo de un Deep de  $n$  nodos representado como un vector de  $n$  posiciones (numeradas de 1 a  $n$ ) estará siempre en la casilla número 1.

La complejidad temporal de la inserción de un elemento en un Heap de altura  $h$  será de  $O(h)$ .

El resultado del recorrido DFS (en profundidad) sobre un digrafo siempre es un camino.

Al representar un grafo dirigido de  $N$  vértices y  $K$  aristas con una lista de adyacencia, la operación que halla la adyacencia de salida de un vértice, tiene una complejidad temporal de  $O(N)$ .

# Examen PED junio 2019

Normas: • Tiempo para efectuar el examen: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
- **Cada pregunta se escribirá en hojas diferentes.**
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con **lápiz**, siempre que sea legible
- **Cada pregunta vale 1,5 puntos (sobre un total de 10 de la nota de Teoría).**
- **Las fechas de "Publicación de notas" y "Revisión del examen teórico" se publicarán en el Campus Virtual.**

**1.** Utilizando exclusivamente las operaciones constructoras generadoras del tipo lista, definir la sintaxis y la semántica de la operación *examen* que actúa sobre una lista ordenada ascendente de números naturales que permite elementos repetidos y devuelve una lista en la que se han borrado todas las ocurrencias de un ítem especificado. **La operación debe ser lo más eficiente posible, reduciendo al máximo el número de llamadas recursivas.**

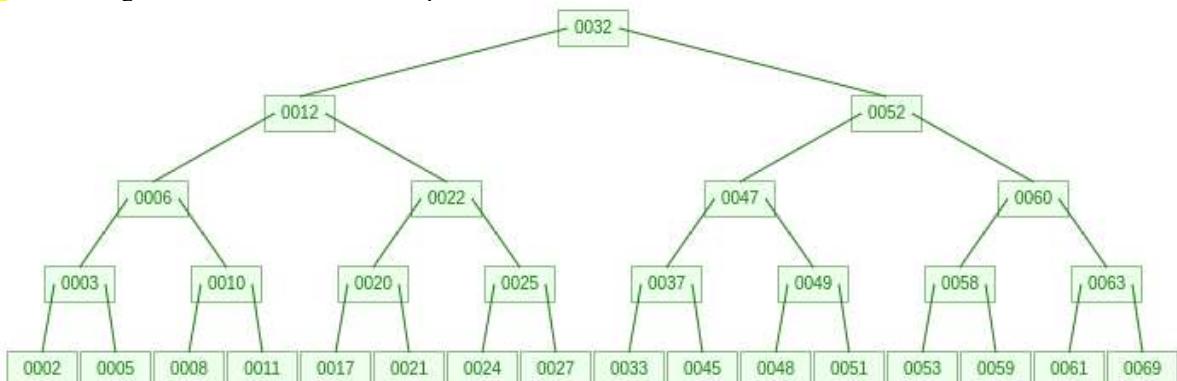
Nota: se asume que está definido el TAD de los números naturales con todas las operaciones aritméticas.

**2. a)** Insertar en una tabla de dispersión (Hash) de tamaño  $B=11$ , sin reestructurar la misma, calculando el NÚMERO DE INTENTOS requerido, con REDISPERSIÓN CERRADA - 2ª FUNCIÓN HASH (**0,5 puntos**) y REDISPERSIÓN ABIERTA (**0,5 puntos**) los siguientes elementos: 10, 5, 35, 24, 16, 4, 3, 7, 8, 47, 6.

**b)** Define y explica la fórmula del factor de carga que sirve para reestructurar la tabla HASH CERRADA dependiendo del número de elementos insertados (**0,1 puntos**). ¿A partir de la inserción de qué elemento se recomienda reestructurar la tabla de HASH CERRADO? Explícalo. ¿Qué tamaño debería tener esa tabla tras la reestructuración (de acuerdo con los requisitos del HASH CERRADO - 2ª FUNCIÓN HASH)? Explícalo (**0,15 puntos**)

**c)** En la tabla HASH ABIERTA, considerando "N" la máxima longitud de las listas de la tabla, calcular las complejidades temporales en su caso peor de las siguientes operaciones: la BUSQUEDA y la INSERCIÓN de 1 elemento. Explícalo. (**0,25 puntos**)

**3.** Dado el siguiente árbol binario de búsqueda:



**a)** Obtener su recorrido inorden (**0,3 puntos**)

**b)** Queremos obtener el mayor elemento de un árbol binario de búsqueda a partir de su recorrido inorden, que se encuentra almacenado en una lista doblemente enlazada como la implementada en el cuadernillo 1. ¿Cuál es la cota de complejidad en el peor caso del algoritmo más eficiente posible para obtener dicho elemento a partir de la lista? Justifica tu respuesta. Nota: a la hora de calcular la complejidad, debe asumirse que el recorrido ya se encuentra calculado y almacenado en la lista (**0,3 puntos**)

**c)** A partir del árbol original mostrado, asumiendo que es un árbol 2-3, inserta la etiqueta 23. Después, borra las etiquetas 11 y 32, en este orden. Si el nodo a borrar tiene dos hijos, sustituir por el mayor de la izquierda; si se puede consultar a dos hermanos, consultar al hermano de la izquierda (**0,45 puntos**)

**d)** A partir del árbol original mostrado, asumiendo que es un árbol 2-3-4, inserta la etiqueta 23. Después, borra la raíz del árbol. Si el nodo a borrar tiene dos hijos, sustituir por el mayor de la izquierda; si se puede consultar a dos hermanos, consultar al hermano de la izquierda (**0,45 puntos**)

**4.** Obtened RAZONADAMENTE el polinomio/funció que expresa la complejidad temporal (número de pasos) y su complejidad asintótica, todos ellos en su caso mejor y peor para cada uno de los 3 siguientes fragmentos de códigos:

<b>a)</b>	<pre>for(i = 1; i &lt; n; i++) a[i-1] = a[i]; for(j = 0; j &lt; (n - 1); j++) a[j] = a[j+1];</pre>
<b>b)</b>	<pre>for(i = 0; i &lt;= n; i++) {     for(j = 1, sum = a[0]; j &lt;= i; j++)         sum += a[j]; }</pre>
<b>c)</b>	<pre>for(i = 0, length = 1; i &lt; n-1; i++) {     for(i1 = i2 = k = i; k &lt; n-1 &amp;&amp; a[k] &lt; a[k+1]; k++, i2++);     if(length &lt; i2 - i1 + 1)         length = i2 - i1 + 1; }</pre>

## Examen PED junio 2019. Soluciones

1.

Sintaxis:

examen(lista, natural) → lista      **(0,1 puntos)**

Semántica:

Var L: lista; ítem, x: natural;

*[InsertarCabeza(L,x) → IC(L,x)]*

examen(crearLista(), ítem)=crearLista()      **(0,1 puntos)**

examen(IC(L,x), ítem) =

    si ( $x > \text{ítem}$ ) entonces IC(L,x)

        sino si ( $x < \text{ítem}$ ) entonces IC(examen(L,ítem), x)

            sino examen(L,ítem)

**(0,7 puntos)**

**(0,3 puntos)**

**(0,3 puntos)**

2.

a) **(1 punto)**

**HASH CERRADO (0,50 puntos)**

10, 5, 35, 24, 16, 4, 3, 7, 8, 47, 6.       $B=11$

$H(x)=x \bmod 11$

$k(x) = (x \bmod 10) + 1$

0	47	$H(47) = 3$ ; $(k(47) = 8)$ $h1(47) = (3+8) \bmod 11 = 0$ (2 <sup>a</sup> )
1	16	$H(16) = 5$ ; $(k(16) = 7)$ $h1(16) = (5+7) \bmod 11 = 1$ (2 <sup>a</sup> )
2	35	$H(35) = 2$ (1 <sup>a</sup> )
3	3	$H(3) = 3$ (1 <sup>a</sup> )
4	4	$H(4) = 4$ (1 <sup>a</sup> )
5	5	$H(5) = 5$ (1 <sup>a</sup> )
6	6	$H(6) = 6$ (1 <sup>a</sup> )
7	24	$H(24) = 2$ ; $(k(24) = 5)$ $h1(24) = (2+5) \bmod 11 = 7$ (2 <sup>a</sup> )
8	8	$H(8) = 8$ (1 <sup>a</sup> )
9	7	$H(7) = 7$ ; $(k(7) = 8)$ $h3(7) = (1+8) \bmod 11 = 9$ (4 <sup>a</sup> )
10	10	$H(10) = 10$ (1 <sup>a</sup> )

**17 INTENTOS**

$H(24) = 2$  ;       $(k(24) = 5)$        $h1(24) = (2+5) \bmod 11 = 7$  (2<sup>a</sup>)

$H(16) = 5$  ;       $(k(16) = 7)$        $h1(16) = (5+7) \bmod 11 = 1$  (2<sup>a</sup>)

$H(7) = 7$  ;       $(k(7) = 8)$   $h1(7) = (7+8) \bmod 11 = 4$  (2<sup>a</sup>)

$(k(7) = 8)$   $h2(7) = (4+8) \bmod 11 = 1$  (3<sup>a</sup>)

$(k(7) = 8)$   $h3(7) = (1+8) \bmod 11 = 9$  (4<sup>a</sup>)

$H(47) = 3$  ;       $(k(47) = 8)$        $h1(47) = (3+8) \bmod 11 = 0$  (2<sup>a</sup>)

**HASH ABIERTO (0,50 puntos)**

10, 5, 35, 24, 16, 4, 3, 7, 8, 47, 6.       $B=11$

$H(x)=x \bmod 11$

0		
1		
2	<b>35</b> (1 <sup>a</sup> )	<b>24</b> (2 <sup>a</sup> )
3	<b>3</b> (1 <sup>a</sup> )	<b>47</b> (2 <sup>a</sup> )
4	<b>4</b> (1 <sup>a</sup> )	

5	5 (1 <sup>a</sup> )	16 (2 <sup>a</sup> )
6	6 (1 <sup>a</sup> )	
7	7 (1 <sup>a</sup> )	
8	8 (1 <sup>a</sup> )	
9		
10	10 (1 <sup>a</sup> )	

## 14 INTENTOS

b) (0,50 puntos)

$$\alpha = \frac{n}{|B|}$$

alfa = "Factor de carga" → indica el nivel de ocupación de la tabla HASH . En HASH Cerrado →  $0 \leq \alpha \leq 1$ .

n = num. de elementos a insertar

B = tamaño de la tabla

(0,10 puntos)

HASH Cerrado reestructura cuando  $\alpha \geq 0,9 \rightarrow$  Por tanto, cuando  $N \geq 0,9 * B$

Reestructurar si  $N \geq 0,9 * B \rightarrow$  Reestructurar si  $N \geq 0,9 * 11 \rightarrow$  Reestructurar si  $N \geq 9,9$

→ Reestructurar a partir del elemento 10º inclusive → El elemento 10º es el 47

→ Antes de insertar el 47, hay que reestructurar.

(0,20 puntos)

Reestructurar a tamaño TEORICO = 22

→ Como NO es primo, hay que reestructurar al TAMAÑO del N° primo más cercano = 23  
(PRIMOS: 2 3 5 7 11 13 17 19 23 29)

(0,20 puntos)

3.

a

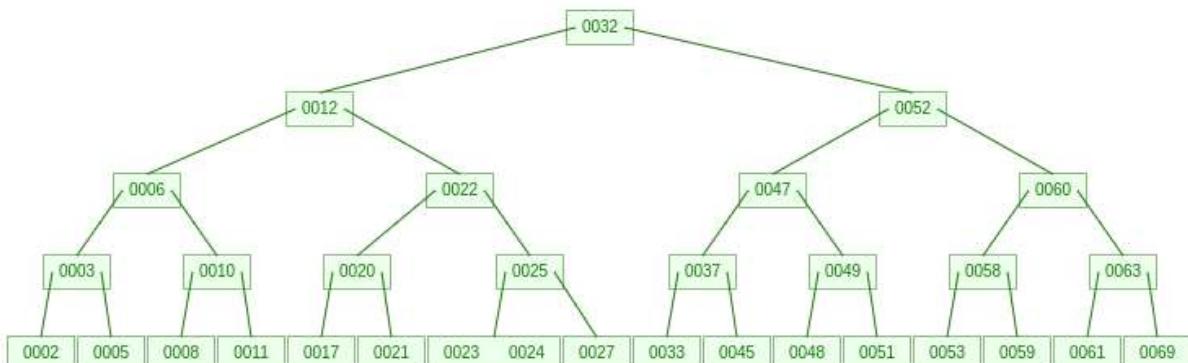
2,3,5,6,8,10,11,12,17,20,21,22,24,25,27,32,33,37,45,47,48,49,51,52,53,58,59,60,61,63,69

b

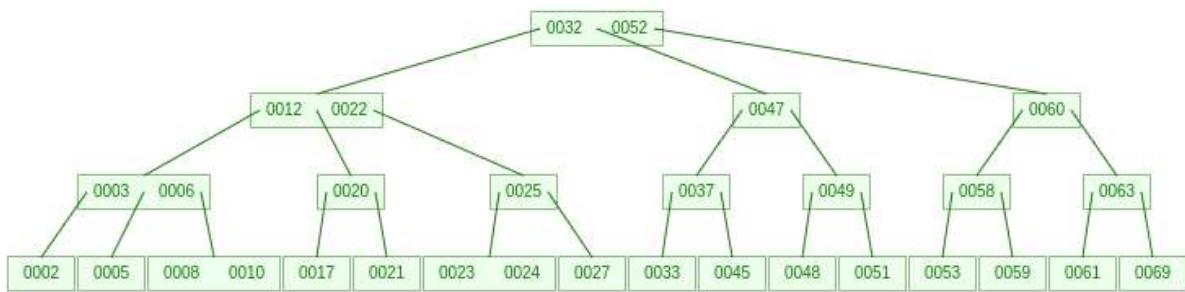
El recorrido inorden de un árbol binario de búsqueda devuelve las etiquetas ordenadas de menor a mayor. Por tanto, el mayor elemento será el que ocupa la última posición de la lista. Como en la lista doblemente enlazada del cuadernillo 1 hay un puntero su último elemento, el coste temporal no depende del número de elementos que haya en el árbol. Por tanto, el algoritmo tiene complejidad constante y su coste temporal pertenece a O(1).

c

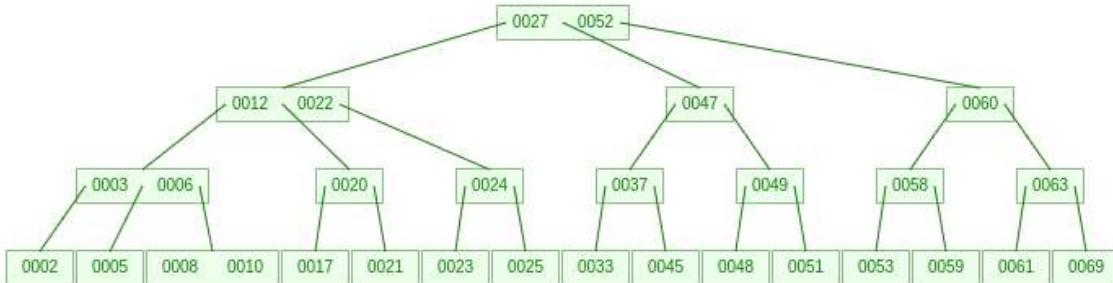
Inserción 23. No se produce ninguna rotación ni combinación.



Borrado 11. Se producen 4 combinaciones:

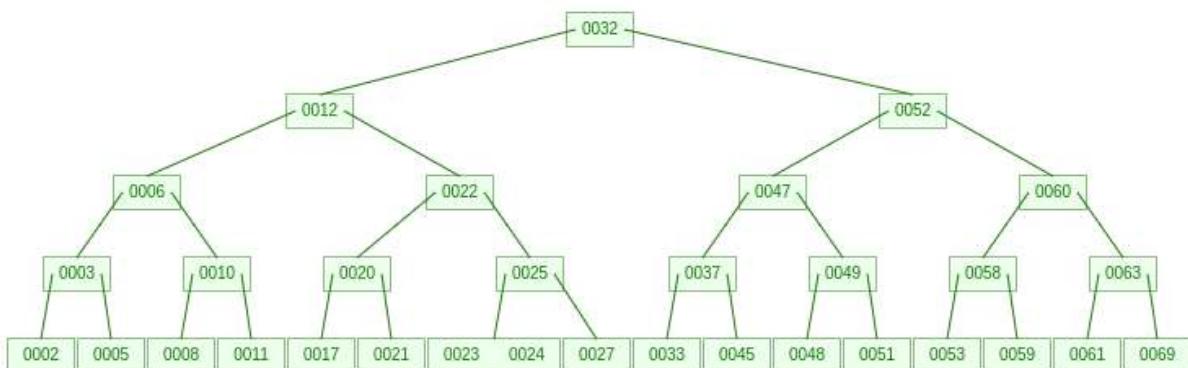


Borrado 32. Se produce 1 rotación.

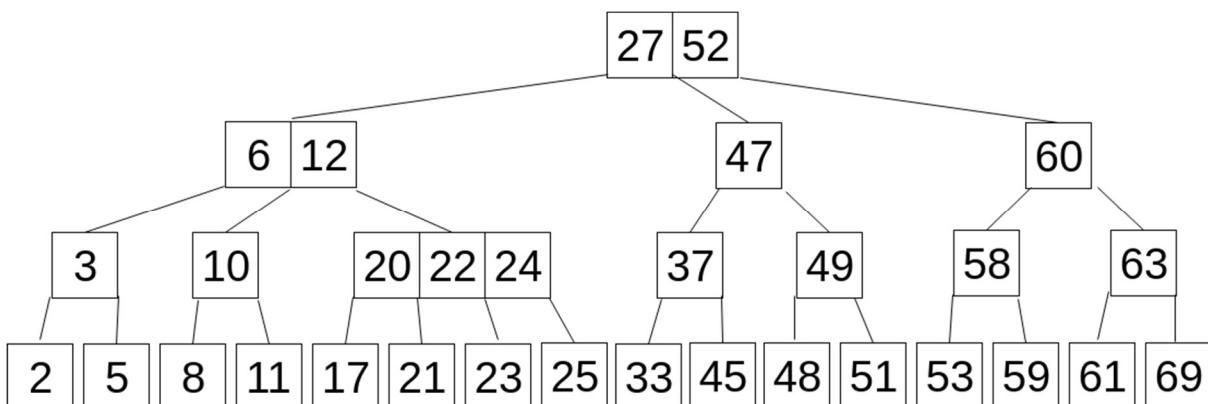


d

Inserción 23. No se produce ninguna rotación ni combinación.



Borrado raíz (32). Se producen 3 combinaciones y 1 rotación.



4.

a) Función de complejidad:

$$\sum_{i=1..n-1} 1 + \sum_{j=0..n-2} 1 = (n-1) - 1 + 1 + (n-2) - 0 + 1 = (n-1) + (n-1) = 2(n-1)$$

Complejidad asintótica:  $\Omega(n)$   $O(n)$

Caso mejor y peor coinciden

b) Función de complejidad:

$$\sum_{i=0..n} (\sum_{j=1..i} 1) = \sum_{i=0..n} (i - 1 + 1) = \sum_{i=0..n} (i) = (n-1) * n / 2$$

Complejidad asintótica:  $\Omega(n^2)$   $O(n^2)$

Caso mejor y peor coinciden

c) CASO MEJOR (Vector ordenado descendentemente)

El primer bucle for ( $i=0$ ,  $length=1$ ;  $i < n-1$ ;  $i++$ ) se repite  $n-2$  veces, ya que, recorre todas las posiciones del vector.

Las repeticiones del segundo bucle vienen determinadas por la condición: for ( $i1=i2=k=i$ ;  $k < n-1$   $&&$   $a[k] < a[k+1]$ ;  $k++$ ,  $i2++$ )

Si el vector está ordenado descendentemente esta condición NUNCA se cumplirá.

Función de complejidad:

$$\sum_{i=0}^{n-2} 1 = n - 2 - 0 + 1 = n - 1$$

Complejidad asintótica:  $\Omega(n)$

CASO PEOR (Vector ordenado ascendentemente)

El primer bucle for ( $i=0$ ,  $length=1$ ;  $i < n-1$ ;  $i++$ ) se repite  $n-2$  veces, ya que, recorre todas las posiciones del vector.

Las repeticiones del segundo bucle vienen determinadas por la condición: for ( $i1=i2=k=i$ ;  $k < n-1$   $&&$   $a[k] < a[k+1]$ ;  $k++$ ,  $i2++$ )

Si el vector está ordenado ascendentemente esta condición SIEMPRE se cumplirá.

$$\sum_{i=0}^{n-2} 1 + \sum_{k=i}^{n-2} 1 = \sum_{i=0}^{n-2} 1 + n - 2 - i + 1 = \frac{(n+2)(n-1)}{2}$$

Complejidad asintótica:  $O(n^2)$

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED enero 2006

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **35 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En la especificación de un TAD, las operaciones auxiliares son visibles para los usuarios.	<input type="checkbox"/>	<input type="checkbox"/>	1. F
Una operación del TAD X que tenga la sintaxis Crear() →X es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2. V
En C++, los miembros <i>protected</i> son privados para el exterior, pero permiten el acceso a las clases derivadas.	<input type="checkbox"/>	<input type="checkbox"/>	3. V
Dadas las clases <i>TDir</i> y <i>TVectorDir</i> :			
class TDir { public: .... private: int e1; char c1; };	class TVectorDir { public: .... private: TDir *vector; int dim; };	TVectorDir::~TVectorDir () { if (v!=NULL) delete v; dim=0; v=NULL; }	
¿Es correcta la implementación del destructor de <i>TVectorDir</i> ?			
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input type="checkbox"/>	5. F
Dado un único recorrido de un árbol binario lleno es posible reconstruir dicho árbol	<input type="checkbox"/>	<input type="checkbox"/>	6. V
Sea el tipo <i>arbin</i> definido en clase. La semántica de la operación <i>nodos</i> es la siguiente:  Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)			
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	8. F
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	9. V
Un árbol completo siempre está balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	10. V
En un árbol AVL siempre que se inserte una etiqueta hay que realizar una rotación	<input type="checkbox"/>	<input type="checkbox"/>	11. F
El coste temporal en el peor caso de la operación de inserción en un árbol 2-3-4 es $\log_2(n+1) \approx \log_2(n)$ siendo "n" el número total de ítems	<input type="checkbox"/>	<input type="checkbox"/>	12. V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	13. V
La semántica de la operación <i>anterior</i> vista en clase es la siguiente:  VAR L1: lista; x: ítem; p: posición; anterior( L1, primera( L1 ) ) = error_posicion(); si p != ultima( L1 ) entonces anterior( L1, siguiente( L1, p ) ) = p anterior( inscabeza( L1, x ), primera( L1 ) ) = primera( inscabeza( L1, x ) )			
Sea el tipo <i>vector</i> definido en clase. La semántica de la operación <i>recu</i> es la siguiente:  Var v:vector; i,j:int; x:ítem; recu(crear_vector(),i)=error_item(); si i<>j entonces recu(asig(v,i,x),j)=recu(v,j) sino recu(asig(v,i,x),j)=TRUE	<input type="checkbox"/>	<input type="checkbox"/>	15. F
<i>crear_pila()</i> , <i>apilar(pila,item)</i> y <i>desapilar(pila)</i> son operaciones constructoras del tipo <i>pila</i> .	<input type="checkbox"/>	<input type="checkbox"/>	16. V
Todo árbol binario de búsqueda es un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	17. F
La semántica de la operación <i>Base</i> que actúa sobre una <i>pila</i> y devuelve el primer elemento apilado es la siguiente:  Base(crear_pila ()) = crear_pila () Base(apilar(crear_pila (), x)) = x Base(apilar(p, x)) = Base(p)			
	<input type="checkbox"/>	<input type="checkbox"/>	18. F

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED febrero 2007

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **35 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V

En C++ y cuando se emplea composición (*layering*), los métodos de la clase derivada pueden acceder a la parte pública de la clase base.

En C++, el puntero *this* no se puede emplear para modificar el objeto al que apunta.

En C++, los constructores se pueden invocar explícitamente cuando el programador lo desee (por ejemplo: *TLista a; a.TLista();*).

En C++, la siguiente declaración es incorrecta: *int& a = I;*

En la escala de complejidades se cumple que  $O(\log n) \subset O(\log \log n)$ .

El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de  $\Omega(1)$ .

Para un vector de naturales, se define la operación *eliminar* que borra las posiciones pares del vector marcándolas con "0" (para calcular el resto de una división, se puede utilizar la operación MOD). La sintaxis y la semántica de la operación *eliminar* es la siguiente:

```

eliminar: vector → vector
Var v:vector; i: entero; x:natural;
eliminar(crear_vector()) = crear_vector()
si (i MOD 2) == 0
entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)
si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x)

```

La semántica de la operación *ultima* vista en clase es la siguiente:

```

VAR L1: lista; x: item;
si esvacia( L1 ) entonces
ultima( inscabeza( L1, x ) = primera( L1 )
si no ultima( inscabeza( L1, x ) = ultima( L1 )

```

Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en preorden con 62 etiquetas.

La sintaxis y semántica de la operación *quita\_hojas*, que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas, es la siguiente:

```

quita_hojas(arbin) → arbin
VAR i, d: arbin; x: item;
quita_hojas(crea_arbin( )) = crea_arbin( )
quita_hojas(enraizar(crea_arbin( ), x, crea_arbin( )) = crea_arbin( )
quita_hojas(enraizar(i, x, d)) =
enraizar(quita_hojas(i), x, quita_hojas(d))

```

Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 2 y 1 hoja es posible reconstruir 2 árboles binarios.

Todo árbol AVL es un árbol 2-3-4

La operación (*DIVIDEHIJODE2 (p, q)*) en la inserción de un elemento en un árbol 2-3-4 puede aumentar la altura del árbol original.

En el algoritmo del borrado de un elemento en un árbol 2-3-4 si *q* es 2-nodo y *r* es 3-nodo hay que hacer una ROTACIÓN.

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED febrero 2008

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **30 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
Dada la sintaxis de la función $IC(lista,item) \rightarrow lista$ , que inserta un elemento a la cabeza de la lista pasada como parámetro y $crear() \rightarrow lista$ , que crea una lista vacía. La siguiente secuencia: $IC(IC(crear()),a),b,c$ , daría como resultado una lista con los elementos en este orden: $a \rightarrow b \rightarrow c$ , donde $a$ es el primer elemento de la lista	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: $F: natural \rightarrow BOOL$ , y su semántica como: $F(cero)=TRUE$ , $F(suc(cero))=FALSE$ , $F(suc(suc(x)))=F(x)$ . Para el número natural $x=35$ , la función F devolvería TRUE.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	<input type="checkbox"/>	<input type="checkbox"/>	5	F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	6	V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución	<input type="checkbox"/>	<input type="checkbox"/>	7	V
La operación BorrarItem, que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i ) = Crear BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces BorrarItem ( L1, i ) sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	8	V
La semántica de la operación obtener en una lista con acceso por posición es la siguiente (IC= InsertarCabeza(Lista, Ítem), p: posición, l1: lista, x: ítem):  obtener(crear(),p)=error_item() si p == primera(IC(l1,x)) entonces obtener(IC(l1,x),p)=x sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)	<input type="checkbox"/>	<input type="checkbox"/>	9	F
El máximo número de nodos en un nivel $i-1$ de un árbol binario es $2^{i-2}$ , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	10	V
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:  Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	11	F
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	12	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	13	V

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED febrero 2008

### Modalidad 1

- Normas:**
- La entrega del test **no** corre convocatoria.
  - \* Tiempo para efectuar el test: **30 minutos**.
  - \* Una pregunta mal contestada elimina una correcta.
  - \* Las soluciones al examen se dejarán en el campus virtual.
  - \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - \* En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	F

El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de  $\Omega(1)$ .  
 El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol  
 Dada la sintaxis de la función  $IC(lista,item) \rightarrow lista$ , que inserta un elemento a la cabeza de la lista pasada como parámetro y  $crear() \rightarrow lista$ , que crea una lista vacía. La siguiente secuencia:  $IC(IC(IC(crear()),a),b),c$ , daría como resultado una lista con los elementos en este orden:  $a \rightarrow b \rightarrow c$ , donde  $a$  es el primer elemento de la lista  
 Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función  $F$  como:  $F: natural \rightarrow BOOL$ , y su semántica como:  $F(cero)=TRUE$ ,  $F(suc(cero))=FALSE$ ,  $F(suc(suc(x)))=F(x)$ . Para el número natural  $x=35$ , la función  $F$  devolvería  $TRUE$ .  
 El máximo número de nodos en un nivel  $i-1$  de un árbol binario es  $2^{i-2}$ ,  $i \geq 2$   
 En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.  
 En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.  
 La operación **BorrarItem**, que borra todas las ocurrencias del item  $i$  que se encuentren en la lista, tiene la siguiente sintaxis y semántica:  
 BorrarItem: LISTA, ITEM  $\rightarrow$  LISTA  
 BorrarItem( Crear, i ) = Crear  
 BorrarItem( IC(L1,j), i ) = si (  $i == j$  ) entonces BorrarItem ( L1, i )  
                               sino IC ( BorrarItem ( L1, i ), j )  
 La semántica de la operación obtener en una lista con acceso por posición es la siguiente (IC=InsertarCabeza(Lista, Ítem), p: posición, l1: lista, x: ítem):  
 obtener(crear(),p)=error\_item()  
 si  $p == \text{primera}(IC(l1,x))$  entonces obtener(IC(l1,x),p)=x  
 sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)  
 La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución  
 La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.  
 Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles  
 Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:  
 Var i,d:arbin; x:item;  
 nodos(crear\_arbin())=0  
 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED febrero 2008

### Modalidad 2

**Normas:** • La entrega del test no corre convocatoria.

\* Tiempo para efectuar el test: **30 minutos**.

\* Una pregunta mal contestada elimina una correcta.

\* Las soluciones al examen se dejarán en el campus virtual.

\* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**

\* En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: F: natural $\rightarrow$ BOOL, y su semántica como: F(cero)=TRUE, F(suc(cero))=FALSE, F(suc(suc(x)))=F(x). Para el número natural x=35, la función F devolvería TRUE.	<input type="checkbox"/>	<input type="checkbox"/>	1 F
La operación BorrarItem, que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i ) = Crear BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces BorrarItem (L1, i ) sino IC ( BorrarItem (L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	2 V
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	3 V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución	<input type="checkbox"/>	<input type="checkbox"/>	4 V
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	5 V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	<input type="checkbox"/>	<input type="checkbox"/>	6 F
El máximo número de nodos en un nivel $i-1$ de un árbol binario es $2^{i-2}$ , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:  Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	8 F
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	9 V
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	10 V
La semántica de la operación obtener en una lista con acceso por posición es la siguiente (IC=InsertarCabeza(Lista, Ítem), p: posición, l1: lista, x: ítem):  obtener(crear(),p)=error_item() si p == primera(IC(l1,x)) entonces obtener(IC(l1,x),p)=x sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)	<input type="checkbox"/>	<input type="checkbox"/>	11 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la sintaxis de la función $IC(lista,item) \rightarrow lista$ , que inserta un elemento a la cabeza de la lista pasada como parámetro y $crear() \rightarrow lista$ , que crea una lista vacía. La siguiente secuencia: $IC(IC(IC(crear(),a),b),c)$ , daría como resultado una lista con los elementos en este orden: $a \rightarrow b \rightarrow c$ , donde a es el primer elemento de la lista	<input type="checkbox"/>	<input type="checkbox"/>	13 F

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED febrero 2009

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **25 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En C++, al declarar una clase "A" como AMIGA de otra clase "B" , todas las funciones miembro de "B" automáticamente pasan a ser funciones AMIGAS de "A"	<input type="checkbox"/>	<input type="checkbox"/>	1	F
En C++, si una clase "B" se construye por composición (layering) , a partir de otra clase "A" , definiendo un objeto miembro de la clase "A" en su parte privada, al invocar al constructor de "B" se invoca antes al constructor de "A" y luego al de "B"	<input type="checkbox"/>	<input type="checkbox"/>	2	V
Las funciones y clases amigas se tienen que declarar en la parte pública de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
Para el siguiente algoritmo, la complejidad temporal en su peor caso sería O(1):	<input type="checkbox"/>	<input type="checkbox"/>	4	V
<pre>for (i=0; i&lt;100; i++)     for (j=0; j&lt;100; j++)         if (v[i]&lt;v[j]) v[i]=v[j];         else v[j]=v[i];</pre>	<input type="checkbox"/>	<input type="checkbox"/>	5	V
La complejidad temporal en su peor caso del siguiente fragmento de código es O(n)	<input type="checkbox"/>	<input type="checkbox"/>	6	V
<pre>int i, j, n, sum; for (i = 4; i &lt; n; i++) {     for (j = i-3; sum = a[i-4]; j &lt;= i; j++) sum += a[j];     cout &lt;&lt; "La suma del subarray " &lt;&lt; i-4 &lt;&lt; " es " &lt;&lt; sum &lt;&lt; endl; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	7	F
La semántica de la operación esvacíapos del tipo vector vista en clase es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/>	8	F
<pre>VAR v: vector; i, j: int; x: item; esvacíapos( crear( ), i ) = CIERTO esvacíapos( asig( v, i, x ), j ) si ( i == j ) entonces FALSO si no esvacíapos( v, j ) fsi</pre>	<input type="checkbox"/>	<input type="checkbox"/>	9	F
La semántica de la operación anterior vista en clase es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/>		
<pre>VAR L1: lista; x: item; p: posicion; anterior( L1, primera( L1 ) ) = error_posicion( ); si p != ultima( L1 ) entonces anterior( L1, siguiente( L1, p ) ) = p anterior( inscabeza( L1, x ), primera( L1 ) ) = L1</pre>	<input type="checkbox"/>	<input type="checkbox"/>		
La sintaxis y semántica de la operación simétricos, que comprueba que 2 árboles binarios son simétricos, es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/>		
<pre>simétricos(arbin, arbin) → bool VAR i1, d1, i2, d2: arbin; x, y: item; simétricos(enraizar(i1, x, d1), crea_arbin( )) = FALSO simétricos(crea_arbin( ), enraizar(i1, x, d1)) = FALSO simétricos(enraizar(i1, x, d1), enraizar(i2, y, d2)) =     si (x == y) entonces (simétricos(i1, d2) &amp; simétricos (d1, i2))     sino FALSO</pre>	<input type="checkbox"/>	<input type="checkbox"/>		
Un árbol binario completo es un AVL	<input type="checkbox"/>	<input type="checkbox"/>		

Apellidos, Nombre:  
DNI:

## Examen PED enero 2010

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **15 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	1	V
<input type="checkbox"/>	<input type="checkbox"/>		
En C++, el valor de la variable q al finalizar este fragmento de código es 11:			
<pre>int q = 0; int k = 5; do {     q += k;     k++; } while(q &lt; 7);</pre>			
<input type="checkbox"/>	<input type="checkbox"/>	2	V
La complejidad temporal (en su caso mejor) del siguiente fragmento de código es $\Omega(n)$			
<pre>int i, length, n, i1, i2, k; for (i = 0, length = 1; i &lt; n-1; i++) {     for (i1 = i2 = k = i; k &lt; n-1 &amp;&amp; a[k] &lt; a[k+1]; k++, i2++);     if (length &lt; i2 - i1 + 1) length = i2 - i1 + 1; }</pre>			
<input type="checkbox"/>	<input type="checkbox"/>	3	F
La semántica de la operación insertar del tipo lista vista en clase es la siguiente:			
<pre>VAR L1: lista; x,y: item; p: posicion;  insertar( crear( ), p, x ) = crear( ) si p == primera( inscabeza( L1, x ) ) entonces     insertar( inscabeza( L1, x ), p, y ) = inscabeza( inscabeza( L1, x ), y )     si no insertar( inscabeza( L1, x ), p, y ) = inscabeza( insertar( L1, p, y ), x )</pre>			
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	F
El grado de un árbol es el grado mínimo de todos los nodos de ese árbol			
El siguiente árbol es binario de búsqueda			
<pre> graph TD     g((g)) --&gt; b((b))     g --&gt; m((m))     b --&gt; f((f))     m --&gt; j((j))     m --&gt; p((p))     j --&gt; k((k))     j --- null   </pre>			
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5  ] el elemento 5 es el hijo izquierda del elemento 8			
En el algoritmo de borrado de un elemento de un árbol AVL, tenemos que actualizar los factores de equilibrio de todos los nodos que han intervenido en la búsqueda del elemento a borrar			
En el algoritmo del borrado de un elemento en un árbol 2-3-4 siempre que q sea 2-nodo hay que hacer una reestructuración.			
El árbol 2-3-4 no vacío tiene como mínimo dos claves en cada nodo			
La operación <i>BorrarItem</i> , que borra todas las ocurrencias del item <i>i</i> que se encuentren en la lista, tiene la siguiente sintaxis y semántica:			
<i>BorrarItem</i> : LISTA, ITEM -> LISTA <i>BorrarItem</i> ( Crear, i) = Crear <i>BorrarItem</i> ( IC(L1,j), i) = si ( i == j ) entonces <i>BorrarItem</i> (L1, i ) sino IC ( <i>BorrarItem</i> (L1, i ), j )			

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 0

Normas:

- Tiempo para efectuar el test: **25 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	V
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 1

**Normas:**

- \* Tiempo para efectuar el test: **25 minutos**.
- \* Una pregunta mal contestada elimina una correcta.
- \* Las soluciones al examen se dejarán en el campus virtual.
- \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- \* En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Sea un vector de números naturales. La operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con “0”, vista en clase, se define así:  eliminar: vector -> vector  Var v:vector; i: entero; x:natural; eliminar(crear()) = crear() si (i MOD 2) == 0 entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x) si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)	<input type="checkbox"/>	<input type="checkbox"/>	2 F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras. Diremos que se trata de una operación derivada.	<input type="checkbox"/>	<input type="checkbox"/>	4 V
En la escala de complejidades, la mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo “n” la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	5 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La operación <i>BorrarItem</i> tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) =      Crear BorrarItem( IC(L1,j), i) =    si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista.	<input type="checkbox"/>	<input type="checkbox"/>	8 F
La operación <i>base</i> , vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente:  base(pila) -> item  Var p: pila; x: item; base(crear()) = error() base(apilar(crear(),x)) = x base(apilar(p,x)) = base(desapilar(p))	<input type="checkbox"/>	<input type="checkbox"/>	9 V
El número mínimo de nodos que tiene un árbol AVL de altura 4 es 7.	<input type="checkbox"/>	<input type="checkbox"/>	10 F
El grado de un nodo es el número máximo de items asociados a dicho nodo.	<input type="checkbox"/>	<input type="checkbox"/>	11 V
El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5  ] el elemento 5 es el hijo izquierda del elemento 8.	<input type="checkbox"/>	<input type="checkbox"/>	13 F
Cuando se realiza un borrado en un árbol AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	14 F
Dado un único recorrido de un árbol binario, es posible reconstruir dicho árbol.			

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013 Modalidad 2

Normas:

- \* Tiempo para efectuar el test: **25 minutos**.
- \* Una pregunta mal contestada elimina una correcta.
- \* Las soluciones al examen se dejarán en el campus virtual.
- \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- \* En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación <i>base</i> , vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente: base(pila) -> item Var p: pila; x: item; base(crear()) = error() base(apilar(crear(),x)) = x base(apilar(p,x)) = base(desapilar(p))	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Sea un vector de números naturales. La operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con “0”, vista en clase, se define así: eliminar: vector -> vector Var v:vector; i: entero; x:natural; eliminar(crear()) = crear() si (i MOD 2) == 0 entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x) si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)	<input type="checkbox"/>	<input type="checkbox"/>	2 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4 F
En la escala de complejidades, la mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo “n” la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	5 F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La operación <i>BorrarItem</i> tiene la siguiente sintaxis y semántica: BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) =      Crear BorrarItem( IC(L1,j), i) =    si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista.	<input type="checkbox"/>	<input type="checkbox"/>	8 V
En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras. Diremos que se trata de una operación derivada.	<input type="checkbox"/>	<input type="checkbox"/>	9 F
Dado un único recorrido de un árbol binario, es posible reconstruir dicho árbol.	<input type="checkbox"/>	<input type="checkbox"/>	10 F
Cuando se realiza un borrado en un árbol AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	11 V
El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5      ] el elemento 5 es el hijo izquierda del elemento 8.	<input type="checkbox"/>	<input type="checkbox"/>	13 F
El grado de un nodo es el número máximo de items asociados a dicho nodo.	<input type="checkbox"/>	<input type="checkbox"/>	14 V

Apellidos, Nombre:

DNI:

## Examen PED abril 2014

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación <i>palindromo</i> (sobre un vector) vista en clase es la siguiente:  Var v: vector; i,x: natural; <i>palindromo(crear_vector()) = VERDADERO</i> <i>palindromo(asig(v,i,x)) = si i&lt;= 50</i> entonces si recu(v,100-i+1) == x entonces palindromo(asig(v,i,x)) sino FALSO sino VERDADERO	<input type="checkbox"/>	<input type="checkbox"/>	1    F
Una operación del TAD X que tenga la sintaxis Crear() →X es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2    V
En C++, al hacer layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	<input type="checkbox"/>	<input type="checkbox"/>	3    F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4    V
El algoritmo de intercambio directo o burbuja estudiado en clase (ordenación de los elementos de un vector) tiene una complejidad promedio de $\Theta(n^2)$ , siendo n el número de elementos del vector.	<input type="checkbox"/>	<input type="checkbox"/>	5    V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.	<input type="checkbox"/>	<input type="checkbox"/>	6    V
La complejidad temporal en el peor caso para la inserción de un elemento en una lista ordenada y en otra no ordenada, que no permiten elementos repetidos, siempre es lineal con el número de elementos en ambos casos.	<input type="checkbox"/>	<input type="checkbox"/>	7    V
El tipo de datos vector (visto en clase) se define como un conjunto en el que sus componentes ocupan posiciones consecutivas de memoria.	<input type="checkbox"/>	<input type="checkbox"/>	8    F
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:  Var i,d:arbin; x:item; <i>nodos(crear_arbin())=0</i> <i>nodos(eraizar(i,x,d))=nodos(i)+nodos(d)</i>	<input type="checkbox"/>	<input type="checkbox"/>	9    F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/>	10    V
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	11    V
A los árboles generales también se les llama árboles multicamino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	12    F
Las rotaciones que hay que realizar en los árboles AVL para mantenerlos balanceados tienen un coste temporal (en su peor caso) lineal con respecto al número de items del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	13    F
El grado de los árboles AVL puede ser +1, 0 ó -1.	<input type="checkbox"/>	<input type="checkbox"/>	14    F

Apellidos, Nombre:  
DNI:

## Examen PED marzo 2015

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
En la especificación algebraica, las operaciones constructoras se clasifican en generadoras y modificadoras.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes:  VAR x, y: natural; suma(x, cero) = x suma(cero, x) = x suma(x, suc(y)) = suma(x, y)	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dentro de la especificación algebraica de los números naturales, definimos la sintaxis de la función F como: F: natural $\rightarrow$ BOOL, y su semántica como: F(cero)=TRUE, F(suc(cero))=FALSE, F(suc(suc(x)))=F(x). Para el número natural x=35, la función F devolvería FALSE. Nota: se asume que x=35 es la forma simplificada de indicar x=suc(suc(...suc(cero)...))).	<input type="checkbox"/>	<input type="checkbox"/>	3	V
Todo árbol binario de altura 9 y 511 nodos es un árbol binario lleno y además es árbol binario de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	4	F
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:	<input type="checkbox"/>	<input type="checkbox"/>	5	F
TPosicion TLista::Primera()	<input type="checkbox"/>	<input type="checkbox"/>	6	F
{ TPosicion p; p.pos = primero; return p; }	<input type="checkbox"/>	<input type="checkbox"/>	7	V
En el método Primera, se invoca de forma implícita a los constructores de TPosicion y TLista.	<input type="checkbox"/>	<input type="checkbox"/>	8	F
En C++, si la variable p es un puntero a un objeto, entonces la expresión p.f() es sintácticamente correcta.	<input type="checkbox"/>	<input type="checkbox"/>	9	V
La complejidad temporal del siguiente fragmento de código es O(n)	<input type="checkbox"/>	<input type="checkbox"/>	10	V
int i, j, n, sum; for (i = 4; i < n; i++) { for (j = i-3, sum = a[i-4]; j <= i; j++) sum += a[j]; cout << "La suma del subarray " << i-4 << " es " << sum << endl; }	<input type="checkbox"/>	<input type="checkbox"/>	11	V
La mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo "n" la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	12	F
Es posible reconstruir un único árbol binario de búsqueda a partir de su recorrido en postorden	<input type="checkbox"/>	<input type="checkbox"/>	13	V
El máximo número de nodos en un nivel i-1 de un árbol binario es $2^{i-2}$ , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	14	F
Un camino en un árbol es una secuencia $a_1, \dots, a_s$ de árboles tal que para todo $i \in \{1, \dots, s-1\}$ , $a_{i+1}$ es subárbol de $a_i$ .				
El grado de un árbol es el máximo nivel que pueden tener sus subárboles.				
La operación desencolar vista en clase es la siguiente:				
<b>VAR</b> c: cola, x: item; desencolar( crear( ) ) = crear( ) <b>si</b> esvacia( c ) <b>entonces</b> desencolar( encolar( c, x ) ) = crear( ) <b>si no</b> desencolar( encolar( c, x ) ) = encolar( desencolar( c ), x )				
El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda siempre se encuentra en la raíz.				

## TESTS DE LOS EXÁMENES

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	V 1.
<input type="checkbox"/>	<input type="checkbox"/>	V 2.
<input type="checkbox"/>	<input type="checkbox"/>	F 3.
<input type="checkbox"/>	<input type="checkbox"/>	F 4.
<input type="checkbox"/>	<input type="checkbox"/>	F 5.
<input type="checkbox"/>	<input type="checkbox"/>	V 6.
<input type="checkbox"/>	<input type="checkbox"/>	V 7.
<input type="checkbox"/>	<input type="checkbox"/>	V 8.
<input type="checkbox"/>	<input type="checkbox"/>	V 9.
<input type="checkbox"/>	<input type="checkbox"/>	V 10.
<input type="checkbox"/>	<input type="checkbox"/>	F 11.
<input type="checkbox"/>	<input type="checkbox"/>	V 12.
<input type="checkbox"/>	<input type="checkbox"/>	F 13.
<input type="checkbox"/>	<input type="checkbox"/>	V 14.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	F	1.
<input type="checkbox"/>	<input type="checkbox"/>	F	2.
<input type="checkbox"/>	<input type="checkbox"/>	F	3.
<input type="checkbox"/>	<input type="checkbox"/>	F	4.
<input type="checkbox"/>	<input type="checkbox"/>	F	5.
<input type="checkbox"/>	<input type="checkbox"/>	V	6.
<input type="checkbox"/>	<input type="checkbox"/>	V	7.
<input type="checkbox"/>	<input type="checkbox"/>	F	8.
<input type="checkbox"/>	<input type="checkbox"/>	F	9.
<input type="checkbox"/>	<input type="checkbox"/>	V	10.
<input type="checkbox"/>	<input type="checkbox"/>	V	11.
<input type="checkbox"/>	<input type="checkbox"/>	V	12.
<input type="checkbox"/>	<input type="checkbox"/>	F	13.
<input type="checkbox"/>	<input type="checkbox"/>	F	14.
<input type="checkbox"/>	<input type="checkbox"/>	V	15.

La operación crear\_pila() es constructora modificadora.

En C++, una forma correcta de copiar una cadena es la siguiente:

```
char a[50] = "Tipos Abstractos de Datos";
char *b;
b = new char[strlen(a)];
strcpy(b, a);
```

El caso peor de la búsqueda es más eficiente en una lista ordenada que en una lista cuyos elementos no están ordenados.

En un árbol binario cada elemento puede tener como máximo dos predecesores.

Si se implementa el algoritmo de ordenación de un vector “heapsort” utilizando un heap máximo los elementos quedan ordenados en el vector de forma descendente.

Dado un recorrido en preorder (RID) de un árbol AVL es posible reconstruir un único árbol AVL.

El número máximo de elementos que se puede almacenar en un árbol 2-3 de altura h es  $3^h - 1$

Al borrar un elemento en un árbol 2-3-4 se puede realizar una operación de DIVIDERAZ.

En un árbol B m-camino de búsqueda con m=16: en cualquier nodo excepto la raíz hay 8 ítems como mínimo.

La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: Ítem):

$$\text{Insertar}(\text{Insertar}(C, x), y) \Leftrightarrow \text{Insertar}(\text{Insertar}(C, y), x)$$

En el TAD Diccionario con dispersión cerrada, con función de redispersión “ $h(x) = (H(x) + k(x)*i) \text{ MOD } B$ ”, con B=6 se puede dar la situación de que en una búsqueda no se acceda a todas las posiciones de la tabla.

El siguiente vector representa un montículo máximo:

10	5	3	1	2
----	---	---	---	---

Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.

La complejidad temporal de la búsqueda en un trie es lineal respecto al número de palabras almacenadas

Un bosque extendido en profundidad de un grafo no dirigido es un grafo acíclico.

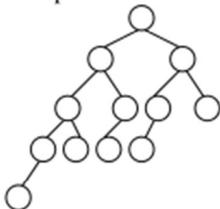
V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1. V
<input type="checkbox"/>	<input type="checkbox"/>	2. F
<input type="checkbox"/>	<input type="checkbox"/>	3. F
<input type="checkbox"/>	<input type="checkbox"/>	4. F
<input type="checkbox"/>	<input type="checkbox"/>	5. F
<input type="checkbox"/>	<input type="checkbox"/>	6. V
<input type="checkbox"/>	<input type="checkbox"/>	7. F
<input type="checkbox"/>	<input type="checkbox"/>	8. F
<input type="checkbox"/>	<input type="checkbox"/>	9. F
<input type="checkbox"/>	<input type="checkbox"/>	10. V
<input type="checkbox"/>	<input type="checkbox"/>	11. V
<input type="checkbox"/>	<input type="checkbox"/>	12. V
<input type="checkbox"/>	<input type="checkbox"/>	13. F
<input type="checkbox"/>	<input type="checkbox"/>	14. F

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1. F
<input type="checkbox"/>	<input type="checkbox"/>	2. F
<input type="checkbox"/>	<input type="checkbox"/>	3. F
<input type="checkbox"/>	<input type="checkbox"/>	4. V
<input type="checkbox"/>	<input type="checkbox"/>	5. V
<input type="checkbox"/>	<input type="checkbox"/>	6. V
<input type="checkbox"/>	<input type="checkbox"/>	7. F
<input type="checkbox"/>	<input type="checkbox"/>	8. F
<input type="checkbox"/>	<input type="checkbox"/>	9. F
<input type="checkbox"/>	<input type="checkbox"/>	10. V
<input type="checkbox"/>	<input type="checkbox"/>	11. V
<input type="checkbox"/>	<input type="checkbox"/>	12. V
<input type="checkbox"/>	<input type="checkbox"/>	13. F
<input type="checkbox"/>	<input type="checkbox"/>	14. F

Siempre que se realiza una rotación DD en el borrado de un elemento en un árbol AVL decrece la altura del subárbol sobre el que se realiza la rotación.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1	F
Un árbol 2-3 es un árbol $m$ -camino de búsqueda con $m=3$ .	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	V
La complejidad temporal (en su peor caso) de buscar un elemento en un vector ordenado utilizando un algoritmo de búsqueda binaria es $\alpha(\log_2)$ [siendo $n$ el número de elementos].	<input type="checkbox"/>	<input checked="" type="checkbox"/>	3	F
Un árbol rojo-negro, en el que no hay ningún enlace rojo, es un árbol binario lleno	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4	V
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $h_i(14)=(28 + 7*i) \text{ MOD } 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	5	V
En el algoritmo HeapSort, después del primer paso de insertar todos los elementos en un Heap representado como un vector, los elementos del mismo quedan ordenados.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	6	F
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de avance y de cruce es un grafo acíclico dirigido.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	7	V
Sea $G=(V,A)$ un grafo dirigido. Diremos que $G''=(V'',A'')$ es un árbol extendido de $G \Leftrightarrow V''=V, A'' \subset A, \forall v \in V'' \Rightarrow \text{grado}_E(v) \leq 1$ .	<input type="checkbox"/>	<input checked="" type="checkbox"/>	8	V
La operación <i>BorrarItem</i> , que borra la primera ocurrencia del ítem $i$ que se encuentre en la lista, tiene la siguiente sintaxis y semántica:	<input type="checkbox"/>	<input checked="" type="checkbox"/>	9	F
	BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) = Crear BorrarItem( IC(L1,j), i) = si ( i == j ) entonces BorrarItem ( L1, i ) sino IC ( BorrarItem ( L1, i ), j )			
Se puede reconstruir un único árbol binario de búsqueda teniendo sus recorridos en preorden y postorden.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	10	V
La estructura de datos árbol aparece porque los elementos que lo constituyen mantienen una estructura jerárquica, obtenida a partir de estructuras lineales, al eliminar el requisito de que cada elemento tiene como máximo un predecesor.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	11	F
Al realizar un recorrido en inorder de un montículo obtenemos una sucesión de claves ordenadas.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	12	F
La función de redispersión en una tabla hash abierta tiene que cumplir como condición para que se recorran todas las posiciones del vector que el valor de B sea primo	<input type="checkbox"/>	<input checked="" type="checkbox"/>	13	F
Un grafo que tiene componentes fuertemente conexas es un grafo necesariamente libre de ciclos	<input type="checkbox"/>	<input checked="" type="checkbox"/>	14	F

V F  
  1. V

El siguiente árbol está balanceado con respecto a la altura

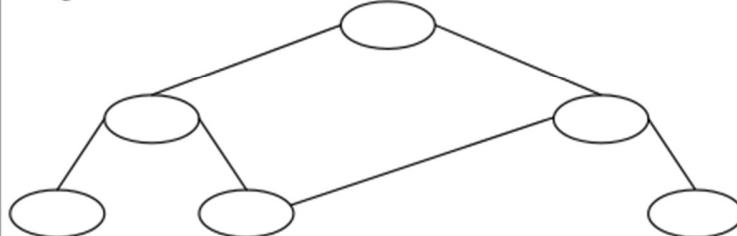


2. V  
  3. F  
  4. F

La siguiente función de C++, `int& Incremento(int valor){valor=valor+5;return valor;}`, devuelve el resultado por referencia.

En las colas, las inserciones y borrados se realizan por el mismo extremo.

La siguiente estructura es un árbol binario:



5. F  
  6. V  
  7. F  
  8. F  
  9. V  
  10. F  
  11. V  
  12. F  
  13. F  
  14. V

Un árbol completo es un árbol completamente equilibrado

Los árboles AVL son árboles balanceados con respecto a la altura de los subárboles.

En un árbol 2-3, la diferencia en número de nodos entre los subárboles de la raíz es como mucho 1.

Un árbol rojo-negro, en el que no hay ningún enlace rojo, es un árbol binario completo.

Un árbol binario de búsqueda con altura 7 y 127 nodos es un árbol B con m=2

En un árbol m-camino de búsqueda, todos los nodos excepto la raíz tienen al menos m/2 hijos.

En la dispersión cerrada se pueden producir colisiones entre claves no sinónimas

En un montículo doble, un elemento "j" del montículo máximo es el simétrico de un único elemento "i" del montículo mínimo.

Un árbol Rojo-Negro cumple las propiedades de un árbol Leftist.

Al representar un grafo no dirigido con una matriz de adyacencia, su diagonal principal (casillas i,i) siempre tendrá valores Falso.

EN C++, la instrucción `cola a = b;` donde `b` es de tipo `cola` invoca al constructor de copia.

1. V  
  2. V

La semántica de la operación `concatena` del tipo cola vista en clase es la siguiente:

```
VAR c, q: cola; x: item;
concatena( c, crear_cola() ) = c
concatena(crear_cola(), c) = c
concatena( c, encolar( q, x ) ) = encolar( concatena( c, q ), x )
```

Se puede reconstruir un árbol binario cualquiera teniendo sus recorridos en preorden e inorden.

En la operación de borrado de un elemento en un árbol AVL, si se realiza una rotación doble siempre decrece la altura del árbol sobre el que se ha realizado la rotación.

El número máximo de elementos que se puede almacenar en un árbol 2-3 de altura  $h$  coincide con el número de elementos que hay en un árbol binario lleno de altura  $h$ .

En un árbol 2-3-4, los nodos pueden tener 1, 2 ó 3 hijos.

Todo árbol Rojo - Negro es un árbol 2-3-4.

Todo árbol B con  $m=4$  es un árbol 2-3

El siguiente vector representa un montículo máximo:

10	5	(Vacío)	1	2
----	---	---------	---	---

3. V  
  4. V  
  5. F  
  6. F  
  7. F  
  8. F  
  9. F

Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de avance es un grafo acíclico dirigido.

10. V

	<b>V</b>	<b>F</b>
Las operaciones constructoras generadoras de un tipo permiten obtener cualquier valor de dicho tipo.	<input type="checkbox"/>	<input type="checkbox"/> 1. V
En C++, si no se ha implementado la sobrecarga del operador asignación, se invoca automáticamente al constructor de copia.	<input type="checkbox"/>	<input type="checkbox"/> 2. F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/> 3. V
La semántica de la operación nodos del tipo <i>arbin</i> vista en clase es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/> 4. F
<i>VAR i, d: arbin; x: item;</i> <i>nodos( crea_arbin() ) = 0</i> <i>nodos( enraizar(i, x, d) ) = nodos(i) + nodos(d)</i>		
Se puede reconstruir un único árbol binario cualquiera teniendo sus recorridos en preorden y postorden.	<input type="checkbox"/>	<input type="checkbox"/> 5. F
La semántica de la operación <i>recu</i> vista en clase es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/> 6. F
<i>VAR v: vector; i, j: int; x: item;</i> <i>recu( crear_vector( ), i ) = error_item( )</i> <i>recu( asig(v, i, x ), j )</i> <i>si ( i == j ) entonces x</i> <i>sino FALSO</i> <i>fsi</i>		
En un árbol AVL cuyo factor de equilibrio es -2, al insertar un elemento en la rama derecha, el árbol vuelve al estado de equilibrio.	<input type="checkbox"/>	<input type="checkbox"/> 7.
Dado un árbol 2-3 de altura $h$ con $n$ ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 n)$	<input type="checkbox"/>	<input type="checkbox"/> 8. V
Un árbol binario de búsqueda lleno de altura 4 es un árbol 2-3-4, pero no se puede conseguir a partir de un árbol inicialmente vacío y utilizando las operaciones de inserción y borrado de un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/> 9. V
Un grafo no dirigido puede tener aristas que empiecen y acaben en el mismo vértice.	<input type="checkbox"/>	<input type="checkbox"/> 10. F
El siguiente árbol es leftist mínimo:		<input type="checkbox"/> 11. V
<pre> graph TD     1((1)) --- 7((7))     1 --- 3((3))     3 --- 4((4))   </pre>		
Un trie cumple las propiedades de un árbol general.	<input type="checkbox"/>	<input type="checkbox"/> 12. V

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes:			
VAR x, y: natural;			
suma(x, cero) = x			
suma(cero, x) = x			
suma(x, suc(y)) = suma(suc(x), y)			
En C++, la siguiente declaración es INCORRECTA :      const int& a = 1;			
Dadas las clases TDir y TVectorDir con todos sus métodos implementados: constructor, constructor sobrecargado, sobrecarga del corchete, sobrecarga del operador salida (se muestra el contenido de cada posición del vector dejando un espacio), etc.			
Nota: El constructor por defecto crea un vector vacío.			
El constructor a partir de una dimensión, pone todos los elementos a 0.			
<pre>class TDir           class TVectorDir        main() {     public: ....      public: ....          TDir a(1,1);     private: ....     private: ...          TVectorDir v;     int e1;           TDir *vector;         cout&lt;&lt;"v_anter= "&lt;&lt;v&lt;&lt;endl;     int e2;           int longitud;        v[1]=a; };  };  };  cout&lt;&lt;"v_despues= "&lt;&lt;v&lt;&lt;endl; } </pre>			
El resultado obtenido tras la ejecución del <i>main()</i> sería:			
<i>v_anter= 0 0</i>			
<i>v_despues= 1 1</i>			
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.			
La complejidad temporal de un algoritmo depende de la complejidad espacial del mismo.			
La semántica de la operación <i>desencolar</i> vista en clase es la siguiente:			
VAR c: cola, x: item; si esvacia( c ) entonces desencolar( encolar( c, x ) ) = crear_cola () si no desencolar( encolar( c, x ) ) = encolar( desencolar( c ), x )			
Un árbol con un único nodo es un árbol lleno.			
Un árbol con un único nodo tiene un único camino cuya longitud es 1.			
Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.			
En la representación de conjuntos mediante listas, la complejidad espacial es proporcional al tamaño del conjunto representado.			
En un grafo dirigido pueden existir infinitas aristas para un número "n" de vértices.			

1 F

2 F

3 F

4 V

5 F

6 V

7 V

8 F

9 V

10 V

11 F

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	F
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	V

La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.

Dada la sintaxis de la función  $IC(lista,item) \rightarrow lista$ , que inserta un elemento a la cabeza de la lista pasada como parámetro y  $crear() \rightarrow lista$ , que crea una lista vacía. La siguiente secuencia:  $IC(IC(crear(),a),b),c$ , daría como resultado una lista con los elementos en este orden:  $a \rightarrow b \rightarrow c$ , donde  $a$  es el primer elemento de la lista

Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función  $F$  como:  $F: natural \rightarrow BOOL$ , y su semántica como:  $F(cero)=TRUE$ ,  $F(suc(cero))=FALSE$ ,  $F(suc(suc(x)))=F(x)$ . Para el número natural  $x=35$ , la función  $F$  devolvería TRUE.

En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.

En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.

El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de  $\Omega(1)$ .

La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución

La operación BorrarItem, que borra todas las ocurrencias del item  $i$  que se encuentren en la lista, tiene la siguiente sintaxis y semántica:

```

BorrarItem: LISTA, ITEM -> LISTA
BorrarItem( Crear, i) =      Crear
BorrarItem( IC(L1,j), i) =    si ( i == j ) entonces BorrarItem ( L1, i )
                                sino IC ( BorrarItem ( L1, i ), j )

```

La semántica de la operación obtener en una lista con acceso por posición es la siguiente ( $IC=InsertarCabeza(Lista, Ítem)$ ,  $p$ : posición,  $l1$ : lista,  $x$ : ítem):

```

obtener(crear(),p)=error_item()
si p == primera(IC(l1,x)) entonces obtener(IC(l1,x),p)=x
sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)

```

El máximo número de nodos en un nivel  $i-1$  de un árbol binario es  $2^{i-2}$ ,  $i \geq 2$

Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:

```

Var i,d:arbin; x:item;
nodos(crear_arbin())=0
nodos(enraizar(i,x,d))=nodos(i)+nodos(d)

```

El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol

Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles

	<input type="checkbox"/>	<input checked="" type="checkbox"/>	V	F	
Las ecuaciones (vistas en clase) que permiten realizar la multiplicación de números naturales son las siguientes:	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1	V	
<pre>VAR x, y: natural; mult(cero, x) = cero mult(x, cero) = cero mult(suc(y), x) = suma(mult(y, x), x)</pre>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	F	
En C++ y cuando se emplea composición (layering), los métodos de la clase derivada pueden acceder a la parte protegida de la clase base.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	3	F	
Para el siguiente algoritmo, la complejidad sería $O(n^2)$ :	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4	F	
<pre>for (i=0; i&lt;100; i++)   for (j=0; j&lt;100; j++)     if (v[i]&lt;v[j]) v[i] = v[j];     else v[j] = v[i];</pre>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	5	V	
Dados los recorridos de preorder, postorden y niveles de un árbol binario sólo se puede reconstruir un único árbol binario.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	6	F	
Sabiendo que A es un árbol binario de búsqueda completo y dado su recorrido inorder 1,4,6,7,9,12,14,20,23. La secuencia 12,7,20,4,9,14,23,1,6, se corresponde con su recorrido por niveles.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	V	F	
<i>Cuando se borra un elemento en un AVL habrá casos en los que no sea necesario reestructurar</i>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1	F	
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2	V	
<pre>TPosicion TLista::Primera() { TPosicion p;   p.pos = lis;   return p; }</pre>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	3	V	
En la línea resaltada, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	4	F	
En la escala de complejidades se cumple que $O(n \log n) \subset O(n^2)$ .	<input type="checkbox"/>	<input checked="" type="checkbox"/>	5	F	
La operación BorrarItem tiene la siguiente sintaxis y semántica:	<input type="checkbox"/>	<input checked="" type="checkbox"/>	6	V	
BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) = Crear BorrarItem( IC(L1,j), i) = si ( i == j ) entonces L1 sino IC ( BorrarItem (L1, i ), j )	<input type="checkbox"/>	<input checked="" type="checkbox"/>	7	F	
Esta operación borra la primera ocurrencia del item que se encuentra en la lista	<input type="checkbox"/>	<input checked="" type="checkbox"/>	8	F	
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input checked="" type="checkbox"/>	9	F	
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input checked="" type="checkbox"/>	10	F	
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, sólo se va a efectuar una rotación como mucho	<input type="checkbox"/>	<input checked="" type="checkbox"/>	11	F	
Dado un árbol 2-3 con $n$ items con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 n)$	<input type="checkbox"/>	<input checked="" type="checkbox"/>	12	V	
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3	<input type="checkbox"/>	<input checked="" type="checkbox"/>	13	F	
Un árbol rojo-negro es un árbol binario balanceado respecto a la altura	<input type="checkbox"/>	<input checked="" type="checkbox"/>	14	V	
La raíz del árbol B m-camino de búsqueda siempre tiene al menos $m/2$ claves o etiquetas	<input type="checkbox"/>	<input checked="" type="checkbox"/>	15	V	
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
$\text{Eliminar}(\text{Crear}, x) \Leftrightarrow \text{Crear}$ $\text{Eliminar}(\text{Insertar}(C, x), y) \Leftrightarrow$ $si (x == y) entonces C sino Insertar(\text{Eliminar}(C, y), x)$	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $h_i(14)=(28 + 7*i) \text{ MOD } 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
El TAD Cola de Prioridad representado por un montículo, tendrá las siguientes complejidades: $O(1)$ para el borrado, y $O(\log n)$ para la inserción, siendo $n$ el número de elementos.	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
En un árbol binario lleno, el camino mínimo de la raíz (longitud del camino más corto hasta un árbol vacío) es igual a la altura del árbol.	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
La altura máxima de un árbol de búsqueda digital es “n+1”, siendo n el número de bits de la clave.	<input type="checkbox"/>	<input checked="" type="checkbox"/>			

Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: F: natural  $\rightarrow$  BOOL, y su semántica como: F(cero)=TRUE, F(suc(cero))=FALSE, F(suc(suc(x)))=F(x). Para el número natural x=35, la función F devolvería FALSE.

En C++, la memoria que se reserva con new se libera automáticamente por el destructor.

La mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), con "n" como la talla del problema.

La semántica de la operación *sublista* del tipo lista vista en clase es la siguiente:

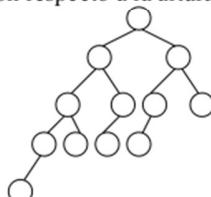
```
VAR L: lista; x, y: item; n: natural; p: posicion;
sublista( L, p, 0 ) = crear()
sublista( crear(), p, n ) = crear()
si p == primera( inscabeza( L, x ) ) entonces
    sublista( inscabeza( L, x ), p, n ) = inscabeza( sublista( L, primera( L ), n ), x )
si no sublista( inscabeza( L, x ), p, n ) = sublista( L, p, n )
```

Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol

Cuando realizamos un recorrido en inorden en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor

Todo árbol binario de búsqueda es un árbol mínimo

El siguiente árbol está balanceado con respecto a la altura



Todo árbol binario de búsqueda es un árbol 2-3.

En un árbol 2-3-4 sólo los nodos hoja y la raíz pueden ser de tipo 2-nodo

En un árbol rojo-negro la búsqueda de una etiqueta dependerá de los colores de los hijos de cada nodo

El árbol 2-3 es un árbol B m-camino de búsqueda con m=2

El TAD Diccionario es un subtipo del TAD Conjunto

En C++, la expresión return \*c; devuelve la dirección de memoria de la variable c.

En un multigrafo pueden existir infinitas aristas para un número "n" de vértices.

La semántica de la operación obtener del tipo lista vista en clase es la siguiente:

VAR L1: lista; x: item; p: posicion;

```
obtener( crear(), p ) = error_item()
si p == primera( inscabeza( L1, x ) ) entonces
    obtener( inscabeza( L1, x ), p ) = x
    si no obtener( inscabeza( L1, x ), p ) = inscabeza( obtener( L1, p ), x )
```

El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo

A los árboles generales también se les llama árboles multicamino de búsqueda

Un árbol binario de búsqueda equilibrado respecto a la altura tiene una complejidad temporal en su peor caso en la búsqueda de  $O(\log_2(n))$ , con n el número de elementos del árbol.

En un árbol 2-3 la altura siempre disminuye si al borrar un elemento se produce una combinación con los elementos de la raíz del árbol

En la operación de borrado de un elemento en un árbol 2-3-4, si hay que realizar reestructuraciones, éstas se realizan desde las hojas hacia la raíz.

Las rotaciones en un árbol Rojo – Negro requieren un cambio de color en los nodos implicados.

Todo árbol binario de búsqueda es un árbol B con m=3.

En la dispersión cerrada puede haber colisiones entre claves sinónimas y no sinónimas.

Un Heap Mínimo es un árbol binario que además es árbol mínimo

En un árbol leftist se cumple que:

$CMIN(x) = 1 + CMIN(HijoDer(x))$  para todo x no vacío y x con dos hijos

V	F
<input type="checkbox"/>	<input type="checkbox"/>

1 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

2 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

3 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

4 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

5 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

6 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

7 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

8 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

9 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

10 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

11 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

12 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

13 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

1 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

2 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

3 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

4 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

5 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

6 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

7 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

8 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

9 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

10 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

11 V

V	F
<input type="checkbox"/>	<input type="checkbox"/>

12 F

V	F
<input type="checkbox"/>	<input type="checkbox"/>

13 V

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V

En C++, la expresión `return &c;` devuelve la dirección de memoria de la variable c.

En C++, una función no puede tener todos sus parámetros con valores por omisión o por defecto.

En la escala de complejidades se cumple que  $O(\log n) \subset O(\log \log n)$ .

La operación BorrarItem, que borra todas las ocurrencias del ítem i que se encuentren en la lista, tiene la siguiente sintaxis y semántica:

```

BorrarItem: LISTA, ITEM -> LISTA
BorrarItem( Crear, i) = Crear
BorrarItem( IC(L1,j), i) = si ( i == j ) entonces BorrarItem ( L1, i )
                           sino IC ( BorrarItem ( L1, i ), j )

```

Un árbol con un único nodo tiene un único camino cuya longitud es 1

En cualquier tipo de datos árbol, cada elemento puede tener varios predecesores, pero como máximo un sucesor.

El siguiente árbol está balanceado con respecto a la altura

Si se inserta un elemento en un árbol 2-3 y todos los nodos que están en el camino desde la raíz a la hoja donde se inserta el elemento son del tipo 3-nodo, la altura del árbol 2-3 resultante crece con respecto al árbol 2-3 original.

En un árbol 2-3-4 de altura 3 donde todos sus nodos son del tipo 3-nodo, el número de elementos total es 27.

En un árbol rojo-negro, el número de enlaces negros ha de ser mayor que el de enlaces rojos.

EsVacia: PILA -> BOOLEAN

Si P y Q son pilas:  $Q = \text{EsVacia}(P)$ , es una expresión sintácticamente correcta

En C++, cuando se sobrecarga un operador que no modifica al operando izquierdo (por ejemplo : "+") se debe crear un objeto temporal, que luego el método devuelve por valor

La complejidad temporal (en su caso promedio) del siguiente fragmento de código es  $\Theta(n^2)$

```

int i, length, n, i1, i2, k;
for (i = 0, length = 1; i < n-1; i++) {
    for (i1 = i2 = k = i; k < n-1 && a[k] < a[k+1]; k++, i2++);
    if (length < i2 - i1 + 1) length = i2 - i1 + 1;
}

```

En cualquier tipo de datos lineal cada elemento tiene como máximo un único sucesor y un único predecesor

El máximo número de nodos en un árbol binario de altura  $k-1$  es  $2^k - 1$ ,  $k \geq 1$ .

En la inserción, en el peor de los casos, las rotaciones realizadas en los árboles AVL para mantenerlos balanceados tienen un coste temporal lineal respecto al número de ítems del árbol

El borrado de un elemento en un árbol 2-3 se realiza en las hojas. Se pueden producir reestructuraciones del árbol en el camino de vuelta desde las hojas a la raíz del árbol.

En un árbol 2-3-4 de altura=2 y número de elementos=15, si se insertara un nuevo elemento se tendría que hacer un DIVIDERAIZ y un DIVIDEHIJODE2

Un árbol rojo-negro es un árbol B con  $m=2$

Todo árbol binario de búsqueda es un árbol B con  $m=4$ .

La complejidad temporal en su peor caso de la operación de Unión entre 2 conjuntos con  $m$  elementos cada uno y representados con una lista desordenada es  $O(m^2)$ .

En dos tablas de dispersión cerrada y abierta con tamaños  $B=7$  y  $B=6$  respectivamente, siempre se cumple que el factor de carga en la abierta es mayor que en la cerrada.

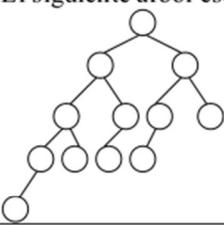
Todo árbol binario que además es árbol mínimo es un Heap Mínimo

La complejidad temporal, en su peor caso, de la operación de PERTENECE de un elemento de tamaño N en un árbol de búsqueda digital es  $O(N+1)$ .

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	V
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	F
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	V

Longitud: LISTA -> NATURAL  
Si L es una lista, a es un ítem de la lista: a = Longitud ( L ) es un uso sintácticamente incorrecto de la operación  
En layering los métodos de la clase derivada pueden acceder a la parte pública de la clase base.  
En C++, el valor de la variable q al finalizar este fragmento de código es 7:  
int q = 0;  
int i;  
for(i = 1; i < 5; i = i + 1)  
if(i != q)  
q += i;  
En la escala de complejidades se cumple que  $O(\log n) \subset O(\log \log n)$ .  
En cualquier tipo de datos lineal cada elemento tiene un único sucesor y varios predecesores  
Un árbol binario completo con n nodos y altura k es un árbol binario lleno para esa misma altura  
El menor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja  
Los árboles AVL son aquellos en los que el número de elementos en los subárboles izquierdo y derecho difieren como mucho en 1  
Un árbol 2-3 es un árbol 2-ario de búsqueda  
El Árbol 2-3-4 no necesita tener como mínimo una clave en cada nodo

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	V

En C++, la instrucción "int &a = 1 ; " daría error de compilación  
En C++, cuando se emplea layering, desde la clase A que contiene un objeto de la clase B siempre se puede acceder a la parte privada del objeto contenido de la clase B  
Paso de programa es una secuencia de operaciones con contenido semántico cuyo coste es dependiente de la talla del problema  
La semántica de la operación cima del tipo pila vista en clase es la siguiente:  
VAR p: pila, e: ítem;  
cima( crear() ) = error()  
cima( apilar( p, e ) ) = cima( p )  
Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol  
En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el mayor del subárbol de la izquierda o por el menor del subárbol de la derecha  
El siguiente árbol está balanceado con respecto a la altura  
  
El borrado de un elemento en un árbol 2-3 se realiza en las hojas. Se pueden producir reestructuraciones del árbol aplicando el algoritmo descendente que empieza en la raíz del árbol y finaliza en las hojas.  
En un árbol 2-3-4 el máximo número de elementos del nivel N es  $3*2^N-1$   
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):  
    Eliminar(Crear, x)  $\Leftrightarrow$  Crear  
    Eliminar(Insertar(C, x), y)  $\Leftrightarrow$   
        si (x == y) entonces C sino Insertar(Eliminar(C, y), x)  
En el TAD Diccionario con dispersión cerrada, los elementos se almacenan en una tabla de tamaño fijo.  
Todo Heap Mínimo cumple las condiciones de ser un árbol binario y un árbol mínimo  
En un multigrafo pueden existir infinitas aristas para un número "n" de vértices.  
En un grafo dirigido con K arcos (el número máximo de arcos en el grafo) y N vértices, una complejidad de  $O(K)$  es equivalente a la complejidad de  $O(N^2)$ .

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	V
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	F

Longitud: LISTA -> NATURAL  
Si L es una lista, a es un ítem de la lista: a = Longitud ( L ) es un uso sintácticamente correcto de la operación Longitud.

Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca:

```
TPosicion Tlista::Primera()
{
    TPosicion p;
    p.pos = lis;
    return p;
}
```

En el método Primera se invoca al constructor y destructor para el objeto TPosicion p.

El algoritmo de intercambio directo o burbuja estudiado en clase (ordenación de los elementos de un vector) tiene una complejidad de  $\Omega(n^2)$ , siendo  $n$  el número de elementos del vector.

La operación BorrarItem, que borra todas las ocurrencias del ítem i que se encuentren en la lista, tiene la siguiente sintaxis y semántica:

```
BorrarItem: LISTA, ITEM -> LISTA
BorrarItem( Crear, i) = Crear
BorrarItem( IC(L1,j), i) = si ( i == j ) entonces BorrarItem ( L1, i )
                           sino IC ( BorrarItem ( L1, i ), j )
```

Existe al menos un árbol binario, que representa los siguientes recorridos: inorder = YXZT, niveles = XTYZ.

El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol.

Un árbol completo siempre está balanceado respecto a la altura.

El grado del árbol 2-3 es 2.

En un árbol 2-3-4 sólo los nodos hoja y la raíz pueden ser de tipo 2-nodo.

En los conjuntos representados como listas no ordenadas, la complejidad temporal de la operación “diferencia de conjuntos” es  $O(n)$ , siendo  $n$  el número de elementos de cada conjunto.

En la dispersión cerrada puede haber colisiones entre claves sinónimas y no sinónimas.

La definición de un Heap Mínimo indica que ha de ser un árbol binario que además es árbol mínimo.

En un grafo dirigido pueden existir infinitas aristas para un número “n” de vértices.

Sea G un grafo no dirigido de n vértices. Si G tiene “n-1” aristas, entonces nunca podría tener un ciclo.

### 1. Introducción: Introducción a los TAD:

Las operaciones constructoras modificadoras permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.	F
Los enriquecimientos forman parte de la definición de un TAD	F
Los enriquecimientos no forman parte de la definición de un TAD	V
EsVacia: PILA -> BOOLEAN. Si P y Q son pilas: Q = EsVacia ( P ), es un uso sintácticamente correcto de la operación	F
Longitud: LISTA -> NATURAL. Si L es una lista, a es un ítem de la lista: a = Longitud ( L ) es un uso sintácticamente incorrecto de la operación	V
Longitud: LISTA -> NATURAL. Si L es una lista, a es un ítem de la lista: a = Longitud ( L ) es un uso sintácticamente correcto de la operación	F
La elección de la estructura de datos que soportará el TAD debe tomarse en la fase de especificación; no en la fase de implementación	F
Una expresión está en forma reducida si contiene operaciones que pertenecen sólo al conjunto de los constructores	V
El calificativo "abstracto" asociado a los tipos de datos hace referencia a que lo que importa es cómo se hacen las cosas y no lo que se hace	F
El tiempo requerido por un algoritmo expresado en función de la talla del problema se llama complejidad espacial del algoritmo	F
Altura: Árbol -> Natural. Si A PERTENECE Árbol, b PERTENECE Ítem, entonces b=Altura (A) es un uso sintácticamente correcto de la operación	F
Las operaciones generadoras son aquellas que permiten generar todos los valores del TAD a especificar	V
Una operación del TAD X que tenga la sintaxis Crear() -> X es una operación constructora generadora.	V
Las operaciones ocultas forman parte de la definición de un TAD	F
Sea el siguiente TAD: MÓDULO NATURALEXAMEN. TIPO natural. OPERACIONES. uno: -> natural; siguiente: natural -> natural. sumar: natural natural -> natural. FMÓDULO. Si N es un natural: N = sumar(uno,siguiente(uno)) es un uso sintácticamente incorrecto de la operación sumar.	F
Una operación consultora devuelve un valor del tipo definido.	F
Para el tratamiento de errores en la especificación, se añaden funciones constantes que devuelven un valor del tipo que causa el error.	V

Para definir la semántica de una operación de un tipo de datos sólo se pueden utilizar las operaciones generadoras constructoras

Si no definimos la semántica de un tipo, las ecuaciones: siguiente(siguiente(uno)) y sumar(uno,siguiente(uno)) denotan diferentes valores

Las operaciones auxiliares de un TAD también se les llama ocultas o privadas

En la escala de complejidades, la complejidad logarítmica es menor que la lineal.

Dada una especificación de un TAD, sólo existe una implementación posible

Las operaciones constructoras modificadoras permiten generar, por aplicaciones sucesivas, todos los valores del TAD a especificar.

Las operaciones constructoras modificadoras permiten obtener cualquier valor del tipo.

### 2. Introducción: C++

En layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.

La sobrecarga del operador corchete tiene que definirse de la siguiente forma para que pueda aparecer a ambos lados de una asignación: Títem operator[] (int i);

La sobrecarga del operador corchete tiene que definirse de la siguiente forma para que pueda aparecer a ambos lados de una asignación: Títem& operator[] (int i);

Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:

```
TPosicion TLista::Primera() { TPosicion p; p.pos = lis; return p; } class TLista { public: ... private: TNodo *lis; }
```

En el método Primera, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.

En layering los métodos de la clase derivada pueden acceder a la parte pública de la clase base.

En herencia pública, la parte privada de la clase base es accesible desde los métodos de la clase derivada.

Para que se ejecute el constructor de copia de la clase base, éste debe ser invocado explícitamente en la fase de inicialización de la clase derivada

En herencia privada los métodos de la clase derivada pueden acceder a la parte pública de la clase base.

Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca:  TPosicion Tlista::Primera( ) { TPosicion p; p.pos = lis; return p; } class Tlista { public: ... private: TNodo *lis; }	V	La operación BorrarItem, que borra todas las ocurrencias del item i que se encuentren en la lista, tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA. BorrarItem( Crear, i ) = Crear. BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces BorrarItem (L1, i ). sino IC ( BorrarItem (L1, i ), j )	V
En el método Primera se invoca al constructor y destructor para el objeto TPosicion p.			
Si definimos un método en la parte privada de una clase, éste será accesible desde los métodos de la propia clase y desde todas sus clases derivadas.	F	La complejidad temporal de obtener un elemento en un vector ordenado o en una lista ordenada es la misma	F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	V	La complejidad temporal de la operación desapilar utilizando vectores ( con un índice que indica la cima ) o listas es la misma	V
En layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	F	La complejidad temporal para la inserción de un elemento en una lista ordenada y en otra no ordenada siempre es lineal con el número de elementos en ambos casos	F
Sea el método Primera perteneciente a la clase Tlista que devuelve la primera posición de la lista que lo invoca:  TPosicion Tlista::Primera( ) { TPosicion p; p.pos = lis; return p; } class Tlista { public: ... private: TNodo *lis; }	F	En cualquier tipo de datos lineal cada elemento tiene un único sucesor y un único predecesor	V
En el método Primera, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.		En cualquier tipo de datos lineal cada elemento tiene un único sucesor y varios predecesores	F
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	F	El tipo de datos vector se define como un conjunto en el que sus componentes ocupan posiciones consecutivas de memoria	F

### 3. Tipos lineales: Vector, Lista, Pila y Cola.

La operación BorrarItem tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA. BorrarItem( Crear, i ) = Crear. BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces L1. sino IC ( BorrarItem (L1, i ), j ).	F	La complejidad temporal en el peor caso de insertar un elemento en un vector ordenado o en una lista ordenada es la misma	V
Esta operación borra todas las ocurrencias del item que se encuentra en la lista		La complejidad espacial de una pila utilizando estructuras dinámicas es constante respecto al tamaño de la pila	F
La operación BorrarItem tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA. BorrarItem( Crear, i ) = Crear. BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces L1. sino IC ( BorrarItem (L1, i ), j ).	V	La operación de lista: Longitud: (LISTA) --> NATURAL es una operación consultora	V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista		Es posible obtener una representación enlazada de una cola utilizando un único puntero que apuntará al fondo de la cola.	V
		En una lista se establece un orden secuencial a partir de las posiciones que ocupan sus elementos	V
		En una cola representada a partir de una lista enlazada simple con un único puntero al principio de la lista (cabeza de la cola), todas las operaciones de la cola (Cabeza, Encolar, Desencolar y EsVacia) se realizan en tiempo constante.	F
		UNA LISTA ES UNA SECUENCIA DE CERO O MÁS ELEMENTOS DE CUALQUIER TIPO DE LA FORMA: EP, ESIG(P), ...	F
		Una lista de acceso por posiciones es un secuencia de cero o mas elementos que pueden ser de distinto tipo	F

La operación de insertar un elemento en una lista ordenada tiene el mismo coste que la de insertar un elemento en un vector ordenado	V	Si en un árbol binario representado secuencialmente tenemos el nodo padre en la posición 5, sus hijos izquierdo y derecho se encuentran, respectivamente, en las posiciones 6 y 7	F
LA COMPLEJIDAD TEMPORAL DE LA OPERACIÓN APILAR EN UNA PILA SIEMPRE ES O(1).	F	La estructura de datos árbol aparece porque los elementos que lo constituyen mantienen una estructura jerárquica, obtenida a partir de estructuras lineales, al eliminar el requisito de que cada elemento tiene como máximo un sucesor y un predecesor.	F
LA COMPLEJIDAD TEMPORAL DE LA OPERACIÓN INSERTAR EN UNA LISTA ES INDEPENDIENTE DE SU IMPLEMENTACIÓN.	F	Un bosque se puede representar con un único árbol binario.	V
Un vector es un conjunto ordenado de pares <índice, valor>.	V	La complejidad temporal del recorrido por niveles es la misma que las de los recorridos in-pre-post orden	V
La semántica de la operación obtener en una lista con acceso por posición es la siguiente (IC= InsertarCabeza(Lista, ítem), p: posición, l1: lista, x: ítem):: obtener(crear(),p)=error_item(). si p = = primera(IC(l1,x)) entonces obtener(IC(l1,x),p)=x. sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)	F	Un camino en un árbol es una secuencia a1, ..., as de áboles tal que para todo . i PERTENECE {1, ..., s-1}, ai+1 es subárbol de ai.	V
La semántica de la operación base que actúa sobre una pila y devuelve el primer elemento apilado es la siguiente (p: pila, x: ítem):: base(crear())=error_item(). base(apilar(crear(),x))=x. base(apilar(p,x))=base(p)	V	El máximo número de nodos en un nivel i-1 de un árbol binario es $2^{i-2}$ , $i \geq 2$	V
En una cola circular enlazada, el elemento apuntado por fondo es el primero a desencolar.	F	El mínimo número de nodos que ha de tener un árbol binario de altura 4 para ser equilibrado respecto a la altura es 7	V
La semántica de la operación base que actúa sobre una pila y devuelve el primer elemento apilado es la siguiente (p: pila, x: ítem):: base(crear())=error_item(). base(apilar(crear(),x))=x. base(apilar(p,x))=base(x)	F	Existe un único árbol binario completo que se puede construir a partir del recorrido en postorden	V
LAS PILAS TAMBIÉN SE CONOCEN COMO LISTAS LIFO.	V	En un árbol binario enhebrado, el primer nodo del recorrido en inorden no tiene hebra izquierda	V

#### 4. Árboles: Conceptos generales

El grado de un árbol es el grado mínimo de todos los nodos de ese árbol	F	Un árbol de Fibonacci siempre está equilibrado respecto a la altura	V
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	F	El grado de un árbol es el máximo nivel de los nodos de un árbol	F
Dado un único recorrido de un árbol lleno, es posible reconstruir dicho árbol	V	En un árbol binario enhebrado todos los nodos tienen ó 2 hebras ó 2 hijos ó 1 hijo y 1 hebra	F
Dado un único recorrido de un árbol, es posible reconstruir dicho árbol	F	En un árbol binario enhebrado todas las hojas tienen 2 hebras.	F
Dado un único recorrido de cualquier árbol, es posible reconstruir dicho árbol	F	Si el nodo a borrar en un árbol binario de búsqueda tiene 2 hijos, éste se puede sustituir por el hijo mayor del subárbol derecho.	F
Un árbol binario completo con n nodos y altura k es un árbol binario lleno para esa misma altura	F	La profundidad de un subárbol es la longitud del único camino desde la raíz a dicho subárbol.	V
El grado de un nodo es el número de items asociados a dicho nodo	F	El grado de un árbol es el número mínimo de hijos que pueden tener sus subárboles.	F
Existe al menos un árbol, que representa los siguientes recorridos: inorden = YXZT, niveles = XTYZ	F	Un bosque de grado 3 se puede representar como un árbol binario.	V
		El máximo número de nodos en un árbol binario de altura k-1 es $2^{k-1}$ , $k \geq 1$ .	F
		En un árbol cada elemento puede tener varios predecesores, pero como máximo un sucesor.	F

## 5. Árboles de búsqueda

En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el menor del subárbol de la izquierda o por el mayor del subárbol de la derecha	F
En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el mayor del subárbol de la izquierda o por el menor del subárbol de la derecha	V
A los árboles generales también se les llama árboles multicamino de búsqueda	F
Cuando realizamos un recorrido en preorden en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor	F
Cuando realizamos un recorrido por niveles en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor	F
El menor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja	F
El mayor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja	F
El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda siempre se encuentra en la raíz.	F
El coste temporal de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	V
El coste temporal de insertar una etiqueta en un árbol binario de búsqueda es logarítmica respecto a la altura del árbol	F
A partir del recorrido por niveles de un árbol binario completo se puede obtener el árbol al que representa.	V
Suponiendo que tenemos un árbol binario de búsqueda lleno con n elementos, la búsqueda del elemento número $n/2$ según la relación de orden se realiza en tiempo logarítmico.	F
Sólo existen tres formas de recorrido en profundidad en un árbol binario: preorden, inorden y postorden	F
El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	V
El número de nodos de un árbol de Fibonacci de altura h es el máximo número de nodos que puede tener un árbol binario de altura h para que sea AVL.	F

## 6. Árboles AVL

Un árbol de Fibonacci siempre está balanceado respecto a la altura	V
Un árbol de Fibonacci es un árbol balanceado con respecto a la altura	V
Un árbol de Fibonacci es un árbol completamente equilibrado	F
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho solo se va a efectuar una rotación	V
Los árboles AVL son aquellos en los que el número de elementos en los subárboles izquierdo y derecho difieren como mucho en 1	F
Cuando se realiza un borrado en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación	F
Las rotaciones que hay que realizar en los árboles AVL para mantenerlos balanceados tienen un coste temporal lineal con respecto al número de items del árbol	F
La complejidad temporal en el peor caso y en el mejor caso de las operaciones inserción y borrado en un AVL son lineal y logarítmica respecto al número de nodos en el árbol	F
El factor de equilibrio en los nodos de un árbol AVL tiene que ser cero para que no haya que reequilibrar el árbol en una operación de inserción o borrado.	F
Un árbol completo siempre está balanceado respecto a la altura	V
La complejidad de las operaciones de equilibrado sugiere que estos árboles deben utilizarse sólo si las inserciones son considerablemente más frecuentes que las búsquedas.	F
Un árbol AVL es un árbol binario de búsqueda en el que la diferencia de nodos entre el subárbol izquierdo y derecho es como máximo uno.	F
En un árbol AVL siempre que se inserte una etiqueta hay que realizar una rotación	F
Un árbol completo es un árbol completamente equilibrado	F
Para realizar la inserción de un elemento en un árbol AVL que sea el máximo o el mínimo, se realizará como máximo una rotación doble	F
El borrado en un árbol AVL puede requerir una rotación en todos los nodos del camino de búsqueda.	V
En el borrado del AVL, la altura del árbol decrece siempre tras realizar una rotación simple.	F
El número mínimo de nodos que tiene un árbol AVL de altura 5 es 12.	V

Dado un recorrido en postorden se puede reconstruir más de un árbol AVL.	F
En un árbol AVL, al realizar una inserción de una sola clave se puede producir como máximo una rotación.	V
Un árbol AVL es un árbol balanceado respecto al número de nodos de los subárboles.	F

## 7. Árboles 2-3

Un árbol 2-3 es un árbol 2-camino de búsqueda	F
El número mínimo de elementos que se pueden almacenar en un árbol 2-3 de altura h es $3h-1$	F
El número mínimo de elementos que se pueden almacenar en un árbol 2-3 de altura h es $2h-1$	V
El mínimo número de elementos que se puede almacenar en un árbol 2-3 de altura h coincide con el número de elementos que hay en un árbol binario lleno de altura h	V
El coste temporal en el peor caso de la operación de inserción en un árbol 2-3-4 es $\log_2(n+1) \Leftrightarrow \log_2(n)$ siendo "n" el número total de ítems	V
Dado un árbol 2-3 de altura h con n ítems: $2h-1 \leq n \leq 3h-1$	V
Dado un árbol 2-3 con n ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 n)$	V
El grado del árbol 2-3 es 2	F
Los nodos de grado 0 de un árbol 2-3 pueden estar en distintos niveles del árbol	F
Sea un árbol 2-3 de altura h, el número total de nodos del árbol está entre $2h-1$ y $3h-1$	F
El número mínimo de elementos que se puede almacenar en un árbol 2-3 de altura h coincide con el número de elementos que hay en un árbol binario completo de altura h	F
Existe un único árbol 2-3 de altura 3 que representa a las etiquetas del 1 al 9.	F
En un árbol 2-3 la altura del árbol sólo aumenta cuando todas las hojas del árbol son de grado tres.	F
Dado un árbol 2-3 con n ítems con todos sus nodos del tipo 3-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_3(n+1))$	V
El máximo grado de cualquier nodo de un árbol 2-3 es 3	V

Dado un árbol 2-3 de altura h con n ítems se cumple que: $3h-1 < n < 2h-1$	F
La operación de Rotación en un árbol 2-3 se da cuando el hermano del nodo donde se efectúa es del tipo 3-Nodo.	V
Dado un árbol 2-3 de altura h con n ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 h)$	F
Dado un árbol 2-3 de altura h con n ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_3 n)$	F

## 8. Árboles 2-3-4

El árbol 2-3-4 no vacío tiene como mínimo dos claves en cada nodo	F
El árbol 2-3-4 no vacío tiene como mínimo una clave en cada nodo	V
En un árbol 2-3-4 de altura h, el máximo número de etiquetas se da cuando todos los nodos son de tipo 2-nodo.	V
a complejidad temporal en el peor caso de la operación inserción en un árbol 2-3-4 es $\log_2(n+1)$	V
La operación de inserción en un árbol 2-3-4 requiere una restructuración del árbol en el camino de vuelta de las hojas a la raíz	F
Para que decrezca la altura de un árbol 2-3-4 en una operación de borrado, el nodo raíz y sus hijos tienen que ser 2-nodo	V
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3-4	V
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3	F
Un árbol 2-3-4 es un árbol binario completo	F
En un árbol 2-3-4 todas las hojas están al mismo nivel	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	V
En un árbol 2-3-4 las inserciones siempre se realizan en las hojas.	V
La operación de borrado en un árbol 2-3-4 no requiere una restructuración del árbol en el camino de vuelta de las hojas a la raíz.	V
En un árbol 2-3-4 con n elementos, la altura de dicho árbol se encuentra entre $\log_4(n+1)$ y $\log_2(n+1)$ .	V

Un árbol 2-3-4 es un árbol 4-camino de búsqueda	V
El número de elementos que hay en un árbol 2-3-4 de altura h está comprendido entre $2h - 1$ y $4h - 1$	V
En el borrado del árbol 2-3-4 sólo se reduce la altura del árbol cuando p, q y r son 2-nodo.	V
En la inserción de un árbol 2-3-4 sólo crece la altura del árbol cuando se realiza una operación de DIVIDERAZ.	V
En la operación de inserción en un árbol 2-3-4 sólo se divide la raíz si ésta es un 3-nodo.	F
En la operación de borrado en un árbol 2-3-4 siempre que q sea 2-nodo hay que hacer una COMBINACIÓN o una ROTACIÓN.	V
En un árbol 2-3-4 los nodos pueden tener 1, 2, 3 ó 4 hijos.	F

#### 9. Árboles Rojos-Negros

El recorrido en inorden en un árbol rojo-negro es el mismo que el realizado sobre un árbol binario	V
El recorrido en preorden en un árbol rojo-negro es el mismo que el realizado sobre un árbol binario	V
El número de hijos negros en cualquier camino desde la raíz a las hojas es siempre el mismo en un árbol rojo-negro	V
Las operaciones de transformación en un árbol 2-3-4 se reducen a cambios de colores o rotaciones en un árbol rojo-negro	V
En un árbol rojo-negro la búsqueda de una etiqueta dependerá de los colores de los hijos de cada nodo	F
La inserción de una etiqueta en un árbol rojo-negro se efectuará creando un hijo de color negro	F
Un árbol rojo-negro es un árbol m-camino de búsqueda con $m=2$	V
Un árbol rojo-negro es un árbol binario balanceado respecto a la altura	F
Un árbol rojo-negro es un árbol balanceado respecto al número de hijos negros que hay desde la raíz hasta cada una de las hojas.	V
Se puede aplicar exactamente el mismo algoritmo de búsqueda de una etiqueta en un árbol binario de búsqueda que en un árbol rojo-negro	V

Las hojas en un árbol rojo-negro están enlazadas a sus padres siempre en color rojo.	F
En un árbol rojo-negro ningún camino desde la raíz a las hojas tiene dos o más hijos negros consecutivos.	F
Un árbol rojo-negro es un árbol binario de búsqueda que representa a un árbol 2-3-4.	V
Todas las operaciones (inserción y búsqueda) de un árbol rojo-negro se realizan en un tiempo $O(\log_2 n)$ , siendo $n$ el número de ítems.	V
En un árbol rojo-negro el algoritmo de búsqueda de una etiqueta es el mismo que el empleado en un árbol binario de búsqueda.	V
Para cualquier nodo de un árbol rojo-negro se cumple que el número de nodos de su subárbol izquierdo es el mismo que el de su subárbol derecho	F

#### 10. Árboles B

El árbol 2-3 es un árbol B m-camino de búsqueda con $m=3$	V
El árbol 2-3 es un árbol B m-camino de búsqueda con $m=2$	F
La raíz del árbol B m-camino de búsqueda siempre tiene al menos $m/2$ claves o etiquetas	F
La raíz del árbol B m-camino de búsqueda no vacío siempre tiene al menos una etiqueta	V
El árbol B m-camino de búsqueda si tiene un nodo que no sea hoja con "x" claves, entonces ese nodo tendrá " $x+1$ " hijos no vacíos.	V
El árbol B m-camino de búsqueda lleno con altura "k" tiene $2k-1$ claves.	F
El nodo de un árbol B m-camino de búsqueda puede tener como máximo "m" ítems.	F
El nodo de un árbol B m-camino de búsqueda puede tener como máximo "m" hijos no vacíos.	V
En el árbol B m-camino de búsqueda, si un nodo tiene "x" claves, entonces "m" es menor que "x"	F
El árbol B m-camino de búsqueda puede tener un nodo hoja en el nivel 3 y otro nodo hoja en el nivel 4.	F
En un árbol B m-camino de búsqueda con $m=18$ : en cualquier nodo excepto la raíz hay 8 ítems como mínimo	V

Un árbol B siempre se puede transformar en un árbol binario de búsqueda completo	V	
Un árbol B siempre se puede transformar en un árbol binario de búsqueda lleno	F	
El nodo de un árbol B m-camino de búsqueda con m=4 puede tener como máximo 4 hijos no vacíos.	V	
Las operaciones de inserción y borrado de un árbol 2-3 son las mismas que las de un árbol B de grado 3	V	
El nodo de un árbol B m-camino de búsqueda con m=100 puede tener como máximo 99 claves.	V	
El árbol B m-camino de búsqueda con altura "k" tiene como máximo mk-1 claves.	F	
El árbol B m-camino de búsqueda con altura "k" tiene como máximo mk -1claves	V	
La altura del árbol B m-camino de búsqueda con m>5 es menor o igual que la del árbol 2-3, o 2-3-4 para el mismo número de claves o etiquetas.	V	
El grado de un árbol B m-camino de búsqueda es "m"	V	
El árbol B m-camino de búsqueda tiene como máximo 2m -1claves.	F	
En un árbol B m-camino de búsqueda con m=16, en cualquier nodo excepto la raíz hay 7 ítems como mínimo.	V	
En el árbol B, la inserción de una clave sólo se puede realizar en las hojas del árbol.	V	
El grado de un árbol B m-camino de búsqueda es "m-1".	F	
En los conjuntos representados como listas enlazadas ordenadas, la complejidad temporal de la operación “pertenencia de un elemento al conjunto” es O(n), siendo n el número de elementos.		
La mejor representación de los conjuntos siempre es el vector de bits porque es la más eficiente espacialmente.		
En la representación de conjuntos mediante listas el espacio es proporcional al tamaño del conjunto representado.		
En la representación de conjuntos mediante las listas el espacio es proporcional al tamaño del conjunto universal.		
La complejidad de la unión de dos conjuntos implementados como listas no ordenadas de tamaño "n" y "m" respectivamente es O(n*m).		
En la representación de conjuntos mediante los vectores de bits, la complejidad espacial es proporcional al tamaño del conjunto representado.		
La complejidad de la intersección de dos conjuntos representados como listas ordenadas de tamaño "n" es O(n).		
El TAD Diccionario es un subtipo del TAD Conjunto		
La representación de conjuntos mediante vectores de bits tiene una complejidad espacial proporcional al tamaño del conjunto universal.		
La complejidad temporal de la búsqueda de un elemento en un conjunto de cardinalidad "n", representado como una lista, es O(n).		
La representación de conjuntos mediante vectores de bits tiene una complejidad espacial proporcional al tamaño del conjunto universal.		
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: ítem): Eliminar(Crear, x) <=> Crear. Eliminar(Insertar(C, x), y) <=> . si (x == y) entonces C sino Insertar(Eliminar(C, y), x)		
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: ítem):. Insertar(Insertar(C, x), y) <=> . si (x == y) entonces Insertar(C,x) sino Insertar(Insertar(C,y), x)		
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: ítem): Eliminar(Crear, x) <=> Crear. Eliminar(Insertar(C, x), y) <=> . si (x == y) entonces Eliminar(C, y) sino Insertar(Eliminar(C, y), x)		
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto (C: Conjunto; x: ítem): Operación(Crear) <=> 0. Operación (Insertar(C, x)) <=> 2 + Operación(C)		

## 11. Conjuntos I (Complejidad espacial/temporal – Algoritmos)

En los conjuntos representados como listas no ordenadas, la complejidad temporal de la operación “diferencia de conjuntos” es O(n).	F	
En los conjuntos representados como listas ordenadas, la complejidad temporal de la operación “diferencia de conjuntos” de cardinalidad “n” es O(n).	V	
La complejidad temporal de la diferencia de dos conjuntos de cardinalidad “n”, representados como listas ordenadas, es O(n).	V	
La complejidad temporal de la diferencia de dos conjuntos de cardinalidad “n”, representados como listas ordenadas, es O(n2).	F	
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: ítem):. Insertar(Insertar(C, x), y) <=> . si (x == y) entonces Insertar(C,x) sino Insertar(Insertar(C,y), x)		
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: ítem): Eliminar(Crear, x) <=> Crear. Eliminar(Insertar(C, x), y) <=> . si (x == y) entonces Eliminar(C, y) sino Insertar(Eliminar(C, y), x)		
La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto (C: Conjunto; x: ítem): Operación(Crear) <=> 0. Operación (Insertar(C, x)) <=> 2 + Operación(C)		

## 12. Conjuntos II (TAD Diccionario – Dispersion cerrada/abierta)

En el TAD Diccionario con dispersión cerrada, los elementos se almacenan en una tabla de tamaño fijo.	V
En el TAD Diccionario con dispersión cerrada, cualquier estrategia de redispersión cuyo siguiente intento esté sólo en función del anterior, producirá amontonamiento.	V
En el TAD Diccionario con dispersión cerrada, con función de redispersión "hi(x)=(H(x) + C*i) MOD B", "B" y "C" han de tener factores primos comunes mayores que uno, para que se busque una casilla libre por toda la tabla.	F
En el TAD Diccionario con dispersión abierta, para evitar el problema del amontonamiento es aconsejable que el tamaño de la tabla sea un número primo o que no tenga factores primos menores que 20.	F
El proceso de dispersión es opuesto al de ordenación	V
Sea una tabla de dispersión cerrada con estrategia de redispersión $hi(x)=(H(x) + C*i) MOD B$ , con $B=1000$ y $C=74$ . Para cualquier clave "x" se recorrerán todas las posiciones de la tabla buscando una posición libre.	F
En el TAD Diccionario con dispersión abierta, la operación de búsqueda de una clave tiene una complejidad $O(L)$ , con $L=\text{longitud de la lista de claves sinónimas colisionadas}$ .	V
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $hi(14)=(28 + 7*i) MOD 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.	V
En la dispersión abierta, el número de intentos a realizar en la búsqueda sin éxito siempre es mayor o igual que en el borrado.	V
El factor de carga de la dispersión cerrada siempre está entre 0 y 1	V
El factor de carga de la dispersión abierta siempre está entre 0 y 1	F
Sea una tabla de dispersión cerrada con función de dispersión $H(x)=x MOD B$ , con $B=100$ y "x" un número natural entre 1 y 2000. Sólo hay un valor de "x" que haga $H(x)=4$ .	F
Cuando implementamos un TAD Tabla de dispersión cerrada se usa una función de dispersión H tal que $H(x)$ devolverá un valor comprendido desde 0 hasta B, siendo B el número finito de clases en las que dividimos el conjunto.	F
En la dispersión cerrada sólo se producen colisiones entre claves sinónimas	F
Cuando utilizamos una tabla de dispersión cerrada de tamaño B, el número de elementos del conjunto está limitado a $B-1$ elementos	F
La dispersión cerrada con estrategia de redispersión aleatoria tiene la siguiente función de redispersión: $hi(x)=( hi-1(x) + C ) MOD B$ .	V

En la dispersión abierta sólo se producen colisiones entre claves sinónimas

V

Sea la dispersión cerrada con la siguiente función de redispersión:  $hi(x)=( hi-1(x) + C ) MOD B$ . Dos claves sinónimas (x, y) tendrán la misma secuencia de intentos, es decir,  $hi(x) = hi(y)$

V

En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14:  $hi(14)=(28 + 2*i) MOD 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.

F

En la búsqueda de elementos en una tabla de dispersion cerrada, hay que distinguir durante la búsqueda las casillas vacías de las suprimidas.

V

En el TAD Diccionario con dispersión cerrada, con función de redispersión " $hi(x)=(H(x) + k(x)*i) MOD B$ ", "B" y " $k(x)$ " no han de tener factores primos comunes mayores que uno, para que se busque una casilla libre por toda la tabla.

V

En el TAD Diccionario con dispersión cerrada, con función de redispersión " $hi(x)=(H(x) + k(x)*i) MOD B$ ", " $B=10$ " y " $k(x)=(x MOD B-1)+1$ ", para la clave  $x=2$  se recorrerán todas las posiciones de la tabla buscando una posición libre.

F

## 13. Conjuntos IV (TAD Cola de prioridades)

El TAD Cola de Prioridad representado por una lista ordenada, tendrá las siguientes complejidades:  $O(1)$  para el borrado, y  $O(n)$  para la inserción, siendo n el número de elementos.

V

El TAD Cola de Prioridad representado por un montículo, tendrá las siguientes complejidades:  $O(1)$  para el borrado, y  $O(\log n)$  para la inserción, siendo n el número de elementos

F

El montículo o HEAP mínimo es un árbol binario lleno que además es árbol mínimo.

F

El montículo o HEAP mínimo es un árbol binario completo que además es árbol mínimo.

V

El TAD Cola de Prioridad representado por una lista desordenada, tendrá coste  $O(1)$  para el borrado.

F

El montículo o HEAP mínimo es una estructura tipo árbol, en la que se tiene al elemento mínimo del conjunto en el nodo hoja más a la izquierda.

F

En un montículo el número de claves en el hijo izquierda de la raíz es mayor o igual que en su hijo derecha

V

El montículo mínimo es un árbol binario completo, en el que se ha establecido una relación de orden parcial por la que el elemento mínimo del conjunto aparece en el nodo raíz.

V

En un montículo de altura k el número total de nodos es  $2k-1$

F

En un montículo doble todas las claves del montículo máximo son mayores que las del montículo mínimo	F
Para todos los nodos de un montículo, se cumple que el número de nodos de su subárbol izquierdo es menor que el número de nodos de su subárbol derecho	F
En el proceso de inserción en un montículo máximo insertaremos en la siguiente posición libre para que siga siendo un árbol binario lleno.	F
Un montículo mínimo es un árbol binario lleno que además es árbol mínimo	F
En un HEAP MAXIMO los elementos de las hojas son los que tienen mayores valores en sus claves	F
Al realizar un recorrido en inorder de un montículo obtenemos una sucesión de claves ordenadas	F
En un montículo doble de altura h se pueden almacenar $2h-1$ claves.	F
El TAD Cola de Prioridad tendrá el mismo coste para la operación de borrado de un ítem tanto si se representa por una lista ordenada como desordenada.	F

#### 14. Conjuntos V (Árboles leftist)

Para todo nodo de un árbol Leftist, se cumple que el número de nodos de su hijo izquierdo es menor que el de su hijo derecho.	F
Para todo nodo de un árbol Leftist, se cumple que el número de nodos de su hijo izquierdo es mayor o igual que el de su hijo derecho.	F
En un árbol binario, el camino mínimo de la raíz es igual a la altura del árbol.	F
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es mayor o igual que la de su hijo derecho.	F
En un árbol binario lleno, el camino mínimo de la raíz (longitud del camino más corto hasta un árbol vacío) es igual a la altura del árbol.	V
En un árbol binario, el camino mínimo de la raíz (longitud del camino más corto hasta un árbol vacío) es igual a la altura del árbol.	F
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.	F
Un árbol 2-3-4 cumple las condiciones para ser también un árbol Leftist.	F
Un árbol binario completo cumple las condiciones para ser también un árbol Leftist.	V

Un árbol binario lleno cumple las condiciones para ser también un árbol Leftist.	V
Todo árbol binario de búsqueda lleno es un árbol Leftist mínimo.	F
En un árbol Leftist que a su vez es un árbol binario lleno, el camino mínimo de la raíz es igual a la altura del árbol.	V
En un árbol binario completo, el camino mínimo de la raíz es igual a la altura del árbol.	F
En un árbol Leftist, el camino mínimo de los nodos hoja siempre es 1.	V

#### 15. Conjuntos VI (Los tries y los árboles digitales)

Las operaciones de búsqueda e inserción de un elemento en un trie tienen complejidad temporal lineal con el número de elementos que pertenecen al conjunto.	F
La representación del nodo de un trie utilizando un vector de punteros siempre es más eficiente espacialmente que utilizar una lista de punteros.	F
La representación del nodo de un trie utilizando un vector de punteros siempre es más eficiente temporalmente que utilizar una lista de punteros.	V
La altura de un trie en su peor caso vendrá determinada por la cadena de mayor longitud almacenada en el árbol.	V
La altura de un trie en su peor caso vendrá determinada por la cadena de menor longitud almacenada en el árbol.	F
En un trie, la elección de la siguiente rama a explorar en un proceso de búsqueda, viene determinada por la siguiente letra de la palabra a buscar.	V
El proceso de búsqueda en un árbol digital, la elección de la siguiente rama a explorar viene determinada por la longitud de la clave buscada	F
La altura máxima de un árbol de búsqueda digital es " $n+1$ ", siendo n el número de bits de la clave.	V
La complejidad temporal de las operaciones de inserción y búsqueda en un árbol de búsqueda digital están en función del numero de bits de la clave.	V
El proceso de búsqueda en un árbol de búsqueda digital es igual que el del árbol binario de búsqueda.	F
La complejidad temporal de las operaciones de inserción y búsqueda en un árbol de búsqueda digital es $O(n)$ siendo n el numero de elementos del conjunto.	F
La complejidad temporal en su caso mejor de la operación de búsqueda en un trie con nodos terminales es de CASOMEJOR(1)	V

La altura de un trie con nodos terminales será como mínimo la longitud de la cadena más larga almacenada.	F
---	---

## 16. Grafos

En un multigrafo pueden existir infinitas aristas para un numero "n" de vértices.	V
En un grafo dirigido pueden existir infinitas aristas para un numero "n" de vértices.	F
En un grafo dirigido con K aristas y N vértices, una complejidad de O(K) es equivalente a la complejidad de O(N <sup>2</sup> ).	V
Un grafo no dirigido de n vértices es un árbol libre si está libre de ciclos y tiene "n-1" aristas	V
Al representar un grafo no ponderado de N vértices y K aristas con una matriz de adyacencia, la operación de búsqueda de una arista tiene una complejidad de O(N).	F
Ciclo es cualquier camino en el que el vértice primero y último coinciden.	F
En un grafo dirigido pueden existir infinitas aristas para un numero "n" de vértices.	F
Dado un grafo dirigido, siempre se cumple que Adyacencia_de_Salida(x) INTERSECCION Adyacencia_de_Entrada(x) = CONJUNTO_VACIO, donde x es un vértice del grafo.	F
En un grafo no dirigido de "n" vértices pueden existir infinitas aristas.	F
Sea G=(V,A) un grafo dirigido. Diremos que G"=(V",A") es un árbol extendido de G <=> V'=V, A" INCLUIDO A, PARATODO v PERTENECE V" ==> gradoE(v) <= 1	V
Al representar un grafo de N vértices y K aristas con una matriz de adyacencia, la operación de calcular la adyacencia de salida de un vértice, tiene una complejidad de O(N).	V
Al representar un grafo de N vértices y K aristas con una lista de adyacencia, la operación que halla la adyacencia de salida de un vértice, tiene una complejidad de O(N).	V
Al representar un grafo dirigido de N vértices y K aristas con una matriz de adyacencia, la matriz será simétrica respecto la diagonal principal.	F
Los arcos de retroceso de un recorrido en profundidad de un grafo dirigido, nos indican la presencia de un ciclo.	V
Al representar un grafo de N vértices y K aristas con una lista de adyacencia, la operación de hallar la adyacencia de entrada de un vértice, tiene una complejidad de O(K).	V

La siguiente secuencia de nodos de un grafo es un ciclo: 1,2,3,2,1	F
Un bosque extendido en profundidad de un grafo dirigido también es un grafo acíclico dirigido.	V
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de avance y de cruce es un grafo acíclico dirigido.	V
Los árboles extendidos de un grafo tienen que ser necesariamente un árbol binario.	F
Sea G un grafo no dirigido de n vértices. Si G tiene "n-1" aristas, entonces nunca podría tener un ciclo.	F
Al clasificar las aristas de un grafo no dirigido en un recorrido en profundidad, sólo existen aristas de árbol y de retroceso.	V
La representación de un grafo mediante una lista de adyacencia, siempre va a ser mejor tanto espacial como temporalmente que la representación mediante una matriz de adyacencia.	F
Los arcos de cruce de un recorrido en profundidad de un grafo dirigido, son los que van de un vértice a un descendiente propio del bosque extendido y no son "arcos del árbol".	F
Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de retroceso es un grafo acíclico dirigido.	F

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED julio 2017

### Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 4 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F*
<input type="checkbox"/>	<input type="checkbox"/>	2	V*
<input type="checkbox"/>	<input type="checkbox"/>	3	F*
<input type="checkbox"/>	<input type="checkbox"/>	4	V*
<input type="checkbox"/>	<input type="checkbox"/>	5	V*
<input type="checkbox"/>	<input type="checkbox"/>	6	F*
<input type="checkbox"/>	<input type="checkbox"/>	7	F*
<input type="checkbox"/>	<input type="checkbox"/>	8	F*
<input type="checkbox"/>	<input type="checkbox"/>	9	F*
<input type="checkbox"/>	<input type="checkbox"/>	10	F*
<input type="checkbox"/>	<input type="checkbox"/>	11	V*
<input type="checkbox"/>	<input type="checkbox"/>	12	V*
<input type="checkbox"/>	<input type="checkbox"/>	13	F*
<input type="checkbox"/>	<input type="checkbox"/>	14	V*
<input type="checkbox"/>	<input type="checkbox"/>	15	F*
<input type="checkbox"/>	<input type="checkbox"/>	16	V*

En la especificación algebraica de un TAD sólo las operaciones constructoras generadoras y las operaciones consultoras necesitan definir su semántica.

Una aplicación de los Grafos Acíclicos Dirigidos es la representación de órdenes parciales.

La complejidad temporal en su peor caso de las operaciones *apilar* y *desapilar* de una pila utilizando una representación enlazada (con punteros a nodo) es lineal respecto al tamaño de la pila.

La semántica de la operación *anterior* vista en clase es la siguiente:  
 VAR L1: lista; x: ítem; p: posición;  
*anterior(L1, primera(L1)) = error\_posicion();*  
*si p != ultima(L1) entonces anterior(L1, siguiente(L1, p)) = p*  
*anterior(inscabeza(L1, x), primera(L1)) = primera(inscabeza(L1, x))*

El grado de un árbol 2-3-4 es el máximo número de etiquetas que puede tener un nodo de dicho árbol más uno.

La semántica de la operación *nodos* del tipo arbin vista en clase es la siguiente:  
 VAR i, d: arbin; x: ítem;  
*nodos(crea\_arbin()) = 0*  
*nodos(enraizar(i, x, d)) = nodos(i) + nodos(d)*

El nivel de la raíz en un árbol binario es 0.

Dado un recorrido en postorden se puede reconstruir más de un árbol AVL.

Dado un árbol 2-3 de altura h con n ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem en su peor caso es  $O(\log_2 h)$

En un árbol 2-3-4 de altura h, el máximo número de nodos se da cuando todos los nodos son de tipo 2-nodo.

La especificación algebraica de la siguiente operación (*Operación: Conjunto  $\rightarrow$  natural*) indica que se devolverá el número de elementos del conjunto multiplicado por 3:  
 VAR C: Conjunto; x: Ítem;  
*Operación(Crear)  $\Leftrightarrow$  0*  
*Operación(Insertar(C, x))  $\Leftrightarrow$  3 + Operación(C)*

El factor de carga ( $\alpha$ ) de una tabla hash con dispersión cerrada cumple que  $0 \leq \alpha \leq 1$ .

En la tabla hash con dispersión cerrada, con función de redispersión “ $h_i(x) = (H(x) + k(x)*i) \bmod B$ ”, “ $B=10$ ” y “ $k(x)=(x \bmod B-1)+1$ ”, para la clave  $x=2$  se recorrerán todas las posiciones de la tabla buscando una posición libre.

El TAD Cola de Prioridad en el que no se permiten elementos repetidos, representado por una lista ordenada, tendrá coste  $O(n)$  para la inserción, con  $n$  el número de elementos del TAD.

En un montículo doble de altura h se pueden almacenar como máximo  $2^h - 1$  claves.

Al representar un grafo no dirigido con una matriz de adyacencia, su diagonal principal siempre tendrá valores FALSO.

## Examen PED julio 2017

Normas: • Tiempo para efectuar el examen: **2 horas**

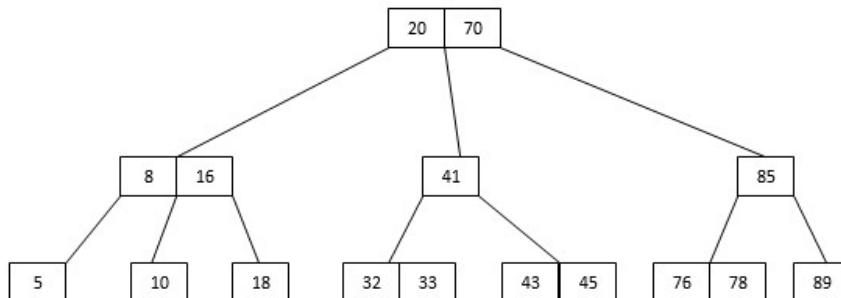
- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
- Cada pregunta se escribirá en hojas diferentes.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Cada pregunta vale 1,5 puntos (sobre 10).**

1. Utilizando la operación *borraultimo* que borra el último elemento de una lista (*borraultimo*: *lista* → *lista*) y las operaciones vistas en clase para el tipo lista de acceso por posición, definir la semántica de la operación **inversas** (*inversas*: *lista, lista* → *bool*) que indica si una lista es la inversa de otra lista dada.

Nota: los elementos de la lista son números naturales. No hay que definir ninguna operación auxiliar para el diseño de la recursividad. En la recursividad, las listas originales pueden ser modificadas.

2. Dado el recorrido por Niveles de un árbol 2-3: { 50, 60, 10, 45, 54, 56, 80, 5, 14, 15, 47, 52, 53, 55, 57, 66, 70, 82 }

- Construir el árbol 2-3 correspondiente.
- Sobre el árbol 2-3 obtenido en el apartado a) realizar las siguientes operaciones consecutivas: Insertar 13, Borrar 50, Borrar 56 y Borrar 14. Mostrar el tipo de operaciones realizadas en cada paso y la estructura del árbol. Aclaraciones: (1) si el nodo tiene dos hijos hay que sustituir por el mayor de la izquierda, (2) si el 2-nodo tiene dos hermanos, consultar el hermano de la derecha (3) **Si no se pone el detalle de las operaciones de reestructuración del árbol, entonces no se valorará la respuesta.**
- Sobre el árbol de tipo 2-3-4 mostrado a continuación, realiza las siguientes operaciones consecutivas: Insertar 31, Insertar 30 y Borrar 89. Mostrar el tipo de operaciones realizadas en cada paso y la estructura del árbol. Aclaraciones: (1) si el nodo tiene dos hijos hay que sustituir por el mayor de la izquierda, (2) Si hay dos nodos adyacentes a q, entonces r será el hermano de la izquierda (3) **Si no se pone el detalle de las operaciones de reestructuración del árbol, entonces no se valorará la respuesta.**



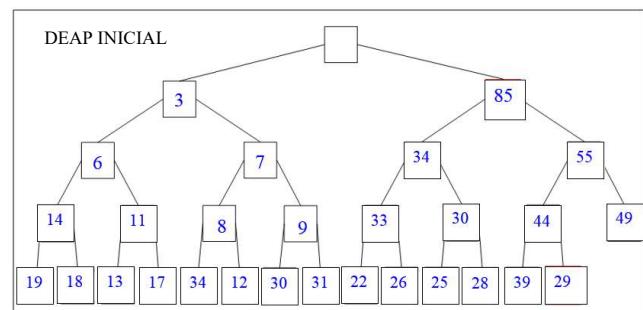
3.

a) Algoritmo HEAPSORT: indicar detallada y RAZONADAMENTE su complejidad en el caso peor.

b) Considera el siguiente DEAP inicial:

- Efectúa el BORRADO del MÁXIMO, expresando claramente el proceso hasta el estado final.
- Explica el último movimiento. ¿Tiene algo especial? Razónalo detalladamente.

c) Considera el DEAP inicial del apartado b). Efectúa el BORRADO del MÍNIMO, expresando claramente el proceso hasta el estado final.



4. Dado el siguiente algoritmo de ordenación de un vector de talla *n* mediante *inserción directa*, calcular la complejidad temporal y espacial en su caso **peor** y **mejor**. Dicha complejidad habrá de explicarse razonadamente, detallando los sumatorios (y su cálculo) que generan la función de **pasos de programa**. **Si no se pone dicho detalle, entonces no se valorará la respuesta.**

```

funcion INSERACION_DIRECTA (var a:vector[natural]; n: natural)
var i,j: entero; x:natural fvar
comienzo
  para i:=2 hasta n hacer
    x:=a[i]; j:=i-1
    mientras (j>0) ∧ (a[j]>x) hacer
      a[j+1]:=a[j]
      j:=j-1
    fmientras
    a[j+1]:=x
  fpara
fin
  
```

## Examen PED julio 2017. Soluciones

1.

inversas: lista, lista → bool

Var x: natural; L1, L2: lista;

inversas (crear\_lista(), crear\_lista()) = T

inversas (crear\_lista(), inscabeza(L1, x)) = F

inversas (inscabeza(L1, x), crear\_lista()) = F

**(0,5 puntos)**

inversas (inscabeza(L1, x), L2) =

    si (x == obtener(L2, ultima(L2))) entonces inversas (L1, borraultimo(L2))

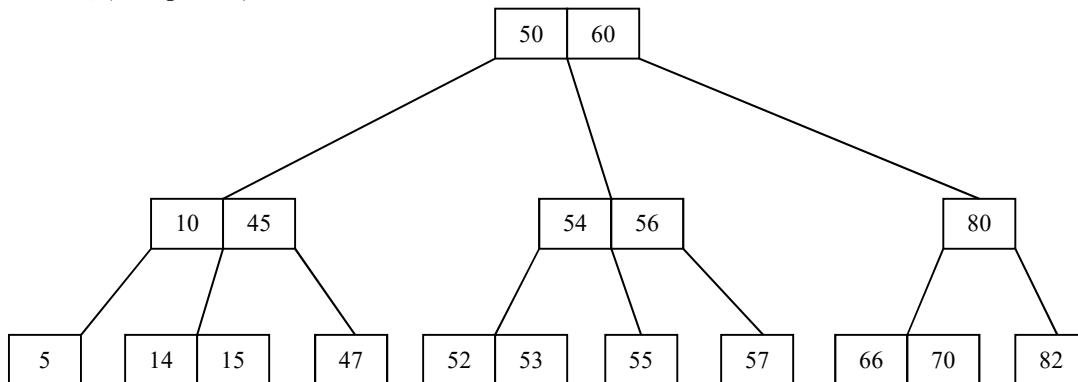
**(0,6 puntos)**

    si no F

**(0,4 puntos)**

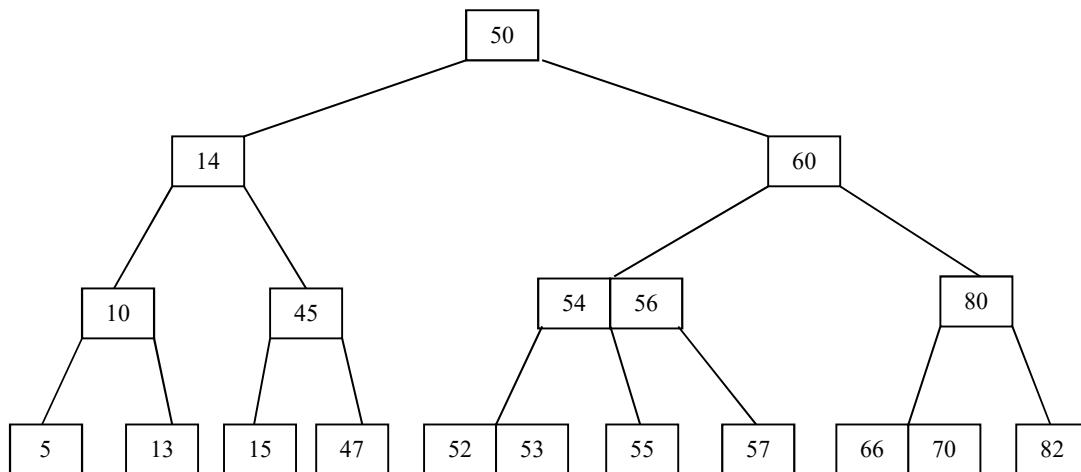
2.

a) **(0,25 puntos)**

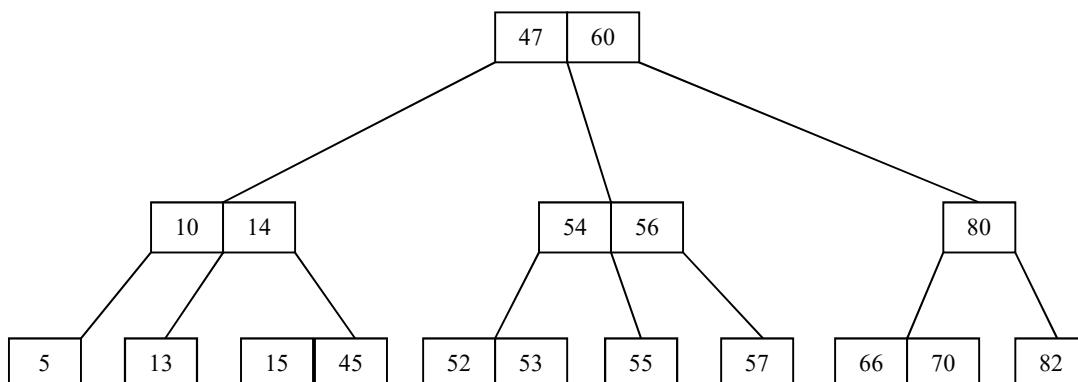


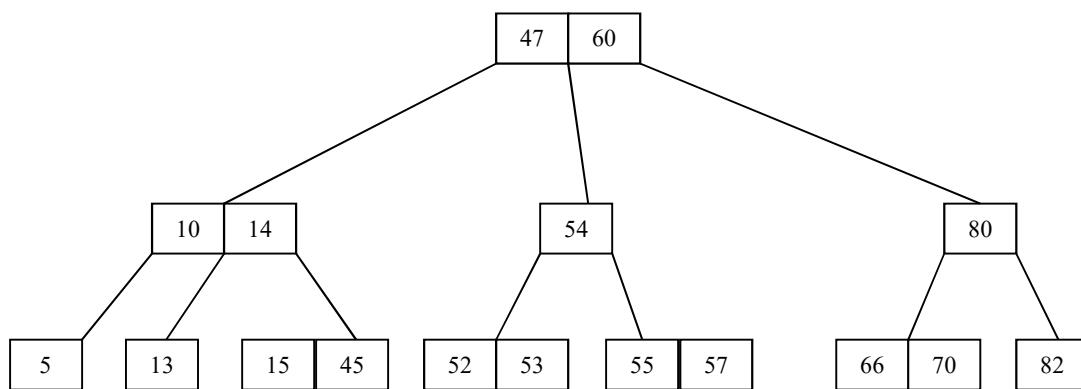
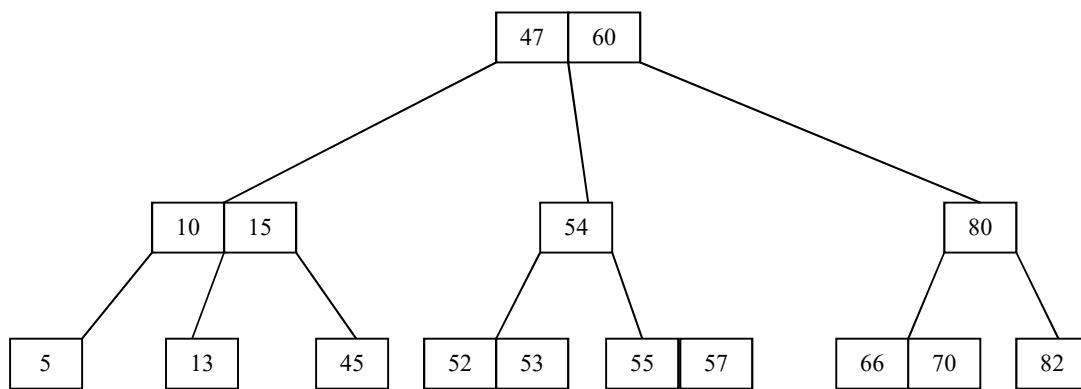
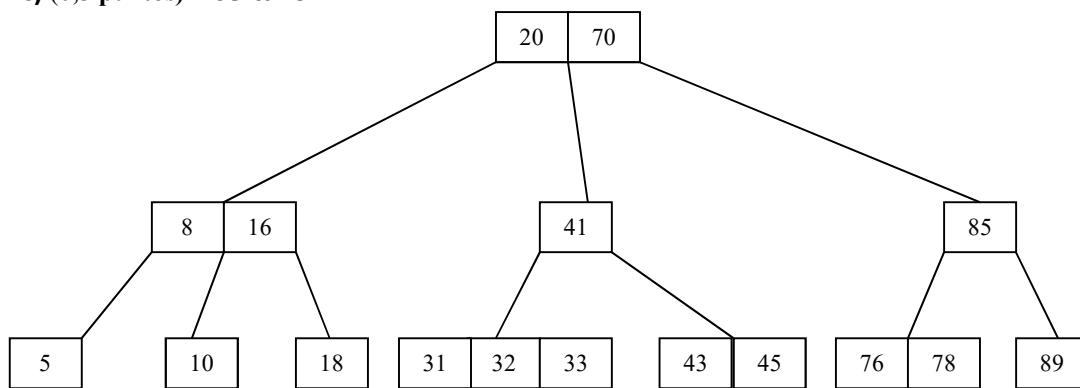
b) **(0,75 puntos)**

Insertar 13:

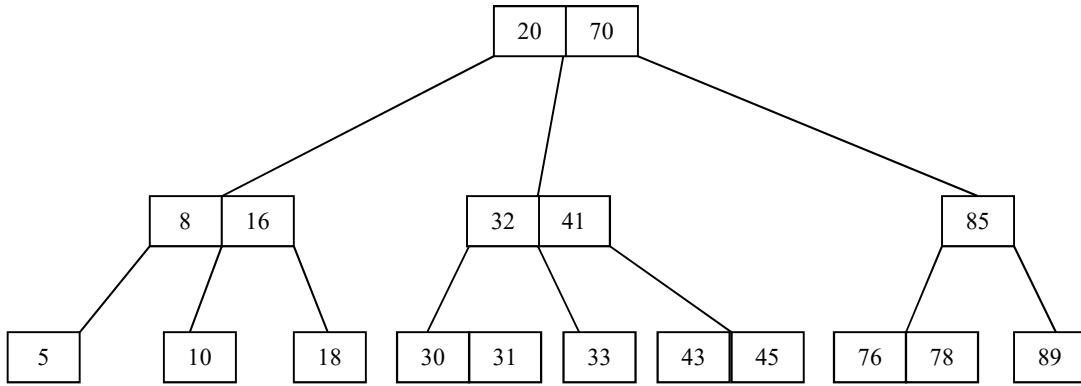


Borrar 50: Comb, comb, comb

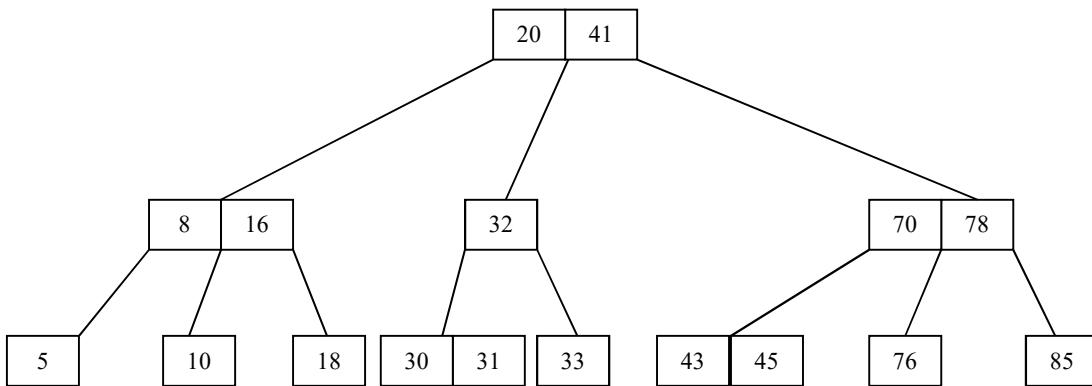


**Borrar 56: Comb****Borrar 14: Rot****c) (0,5 puntos) Insertar 31**

### Insertar 30: Divide Hijode2



### Borrar 89: Rotación, Rotación



### 3. a) Algoritmo HEAPSORT: indicar detallada y RAZONADAMENTE su complejidad en el caso peor. (0'4 pts.)

Tendría una complejidad  $O(n \log n)$ , con  $n$  el número de elementos del vector.

El algoritmo HeapSort utiliza un montículo simple, habría que realizar  $n$  operaciones de inserción y borrado, cada una de ellas con una complejidad  $O(\log n)$ .

- IMPLEMENTACIÓN (UN SÓLO VECTOR):
  - 1) Dejar parte izquierda del vector para el HEAP, y parte derecha para los elementos todavía no insertados.
  - 2) Borrar la raíz del HEAP llevándola a la parte derecha del vector.

- COMPLEJIDAD:  
 $O(n \log n)$

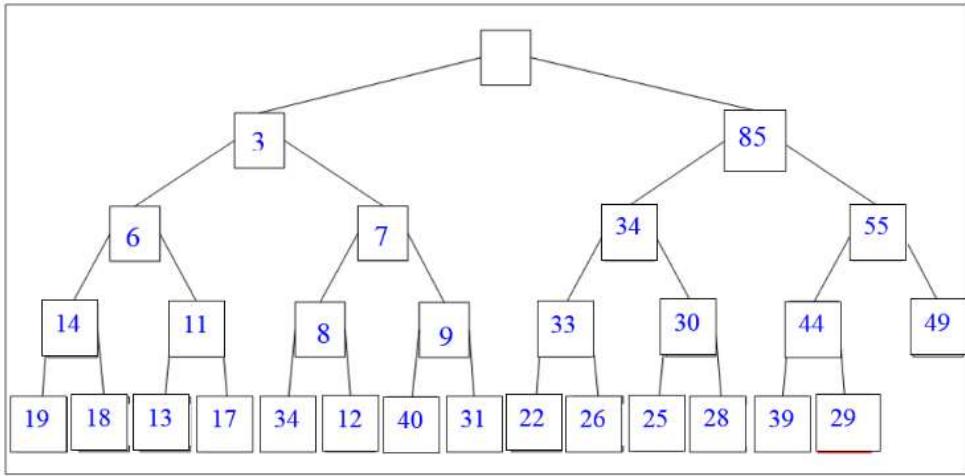
Complejidad de : 1)INSERTAR + 2)BORRAR  
 $N = \text{n\'um. de nodos a insertar al \'arbol HEAP}$   
 $\rightarrow O(N * \log_2 N) + O(N * \log_2 N)$   
 $\rightarrow O(N * \log_2 N)$

HEAP o MONTICULO:  
 INSERCIÓN:  $O(\log n)$   
 BORRADO:  $O(\log n)$

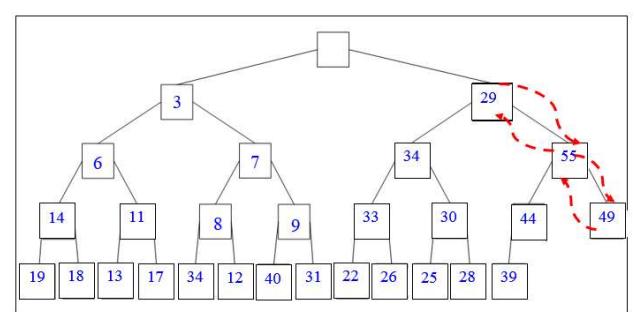
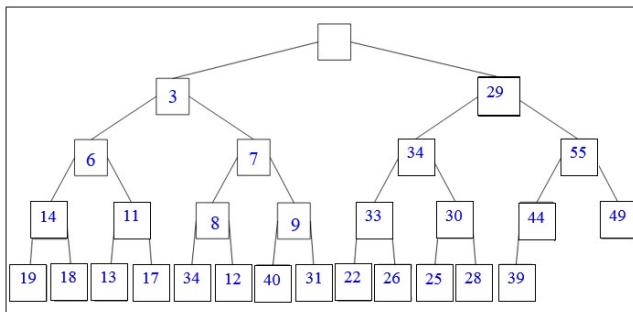
### b) Considera el siguiente DEAP inicial:

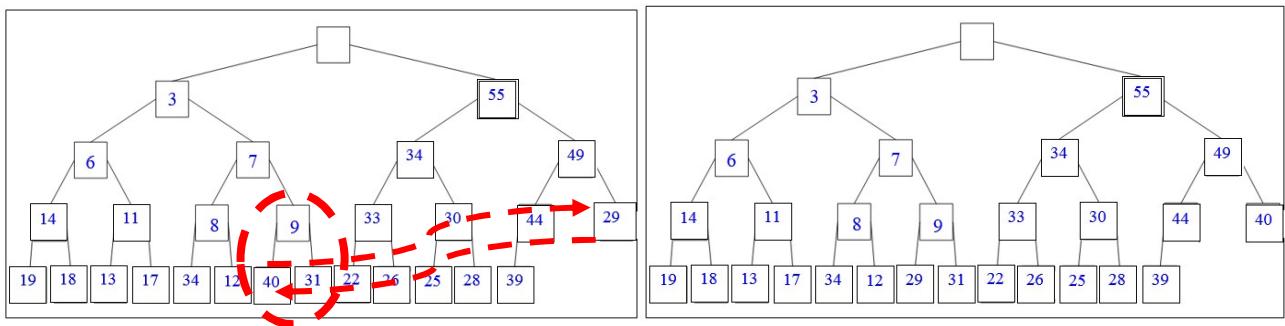
- b.1) Efectúa el BORRADO del MÁXIMO, expresando claramente el proceso hasta el estado final. (0'5 pts.)
- b.2) Explica el último movimiento. ¿Tiene algo especial? Razónalo detalladamente. (0'25 pts.)

# Inicial



b.1) Proceso y estado FINAL del DEAP:





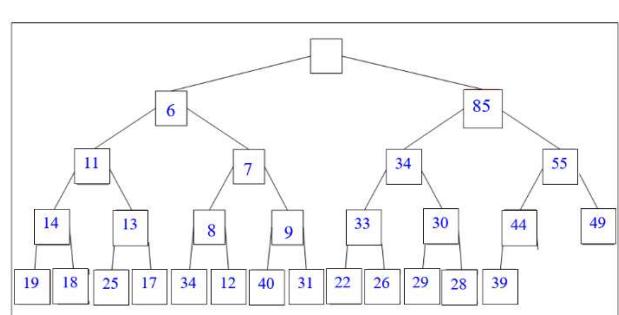
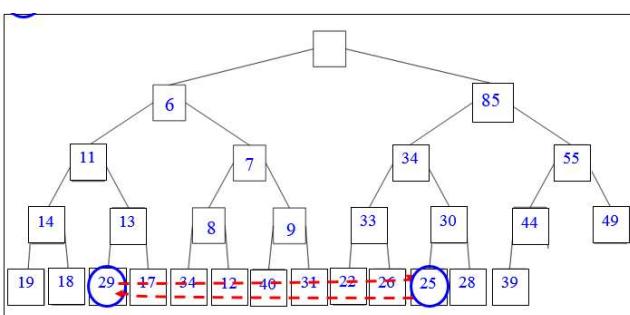
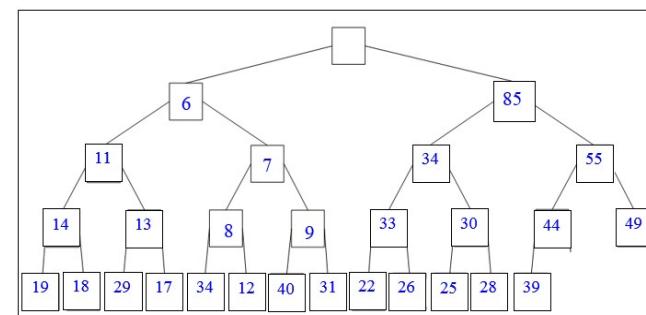
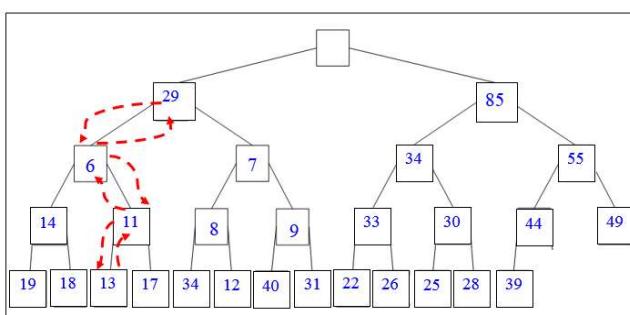
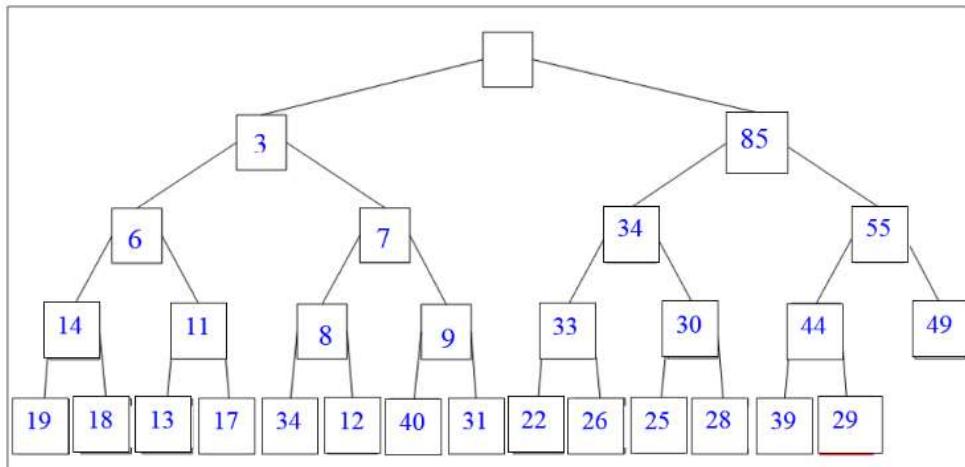
b.2) ÚLTIMO MOVIMIENTO en el DEAP : los elementos 29 Y 40 se INTERCAMBIAN (al ser el 40 el MAYOR de 1)

en este caso, hay que escoger del conjunto de los simétricos el valor máximo para que cada subárbol siga siendo Hep minimo y máximo.  
hay que escoger el máximo valor de sus simétricos. En este caso el 30. SIMÉTRICOS DE 29).

40

c) Considera el DEAP inicial del apartado b). Efectúa el BORRADO del MÍNIMO, expresando claramente el proceso hasta el estado final. (0'35 pts.)

Inicial



#### 4.

Complejidad temporal:

- **(0,5 puntos)** Caso mejor: en el *mientras* no entra nunca. El último elemento de destino es siempre menor que el primero de origen y como destino ya está ordenado  $\rightarrow$  vector ordenado ascendentemente  
$$\sum_{i=2..n} 1 = n - 2 + 1 = n-1 \in \Omega(n)$$
- **(0,5 puntos)** Caso peor: todos son mayores en destino. Hay que desplazarlos todos  $\rightarrow$  vector ordenado inverso  
$$\sum_{i=2..n} (1 + \sum_{j=1..i-1} 1) = \sum_{i=2..n} (1 + ((i-1) - 1 + 1)) = \sum_{i=2..n} (1+i-1) = \sum_{i=2..n} i = n*(n+1)/2 - 1 \in O(n^2)$$

Complejidad espacial: **(0,5 puntos)**

Mejor y peor caso coinciden puesto que solo se utiliza para la ordenación el vector de talla n:  $\Omega(n) = O(n)$

**Apellidos, Nombre:**

**DNI:**

# **Examen PED julio 2018**

## **Modalidad 0**

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual. Este test vale 4 puntos (sobre un total de 10 de la nota de Teoría).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación crear_pila() es una operación constructora modificadora.	<input type="checkbox"/>	<input type="checkbox"/>	<b>1</b> F
La semántica de la operación <i>recu</i> sobre el TIPO vector vista en clase es la siguiente: <b>VAR</b> v: vector; i, j: int; x: item; recu( crear_vector( ), i ) = error_item( ) recu( asig(v, i, x ), j ) <b>si</b> ( i == j ) <b>entonces</b> CIERTO <b>sino</b> FALSO <b>fsi</b>	<input type="checkbox"/>	<input type="checkbox"/>	<b>2</b> F
La función POT_2 descrita a continuación presenta una complejidad temporal O( $2^n$ ) función POT_2 (n: natural): natural opción n = 1: devuelve 2 n > 1: devuelve 2 * POT_2(n-1) fopción ffunción	<input type="checkbox"/>	<input type="checkbox"/>	<b>3</b> F
Las pilas también se conocen como listas LIFO.	<input type="checkbox"/>	<input type="checkbox"/>	<b>4</b> V
La sintaxis y semántica de la operación <i>nodosHoja</i> , que devuelve el número de nodos hoja de un árbol binario, es la siguiente: nodosHoja(arbin) → natural <b>VAR</b> i, d: arbin; x: item; nodosHoja(crea_arbin( )) = 0 nodosHoja(enraizar(i, x, d)) = <b>si</b> esvacío(i) & esvacío(d) <b>entonces</b> 1 <b>sino</b> nodosHoja(i) + nodosHoja(d)	<input type="checkbox"/>	<input type="checkbox"/>	<b>5</b> V
Es posible reconstruir un único árbol binario de búsqueda completo a partir de su recorrido en inorden.	<input type="checkbox"/>	<input type="checkbox"/>	<b>6</b> V
En el borrado de un elemento en un árbol AVL, la altura del árbol decrece siempre tras realizar una rotación simple.	<input type="checkbox"/>	<input type="checkbox"/>	<b>7</b> F
El grado de un árbol 2-3 es 3.	<input type="checkbox"/>	<input type="checkbox"/>	<b>8</b> V
Dado un árbol 2-3 de altura h con n items con todos sus nodos del tipo 3-Nodo: la complejidad de la operación de búsqueda de un ítem es O( $\log_2 n$ ).	<input type="checkbox"/>	<input type="checkbox"/>	<b>9</b> F
En la operación de inserción de un elemento en un árbol 2-3-4 sólo se divide la raíz si ésta es un 3-nodo.	<input type="checkbox"/>	<input type="checkbox"/>	<b>10</b> F
En la representación de conjuntos mediante los vectores de bits, la complejidad espacial es proporcional al tamaño del conjunto representado.	<input type="checkbox"/>	<input type="checkbox"/>	<b>11</b> F
En la dispersión cerrada se pueden producir colisiones entre claves sinónimas y no sinónimas.	<input type="checkbox"/>	<input type="checkbox"/>	<b>12</b> V
En un montículo doble, un elemento “i” del montículo mínimo tiene como máximo un elemento simétrico “j” del montículo máximo.	<input type="checkbox"/>	<input type="checkbox"/>	<b>13</b> V
Los árboles extendidos de un grafo dirigido tienen que ser necesariamente árboles binarios.	<input type="checkbox"/>	<input type="checkbox"/>	<b>14</b> F
Un bosque extendido en profundidad de un grafo dirigido también es un grafo acíclico dirigido.	<input type="checkbox"/>	<input type="checkbox"/>	<b>15</b> V

## Examen PED julio 2018

Normas: • Tiempo para efectuar el examen: **2 horas**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
- Cada pregunta se escribirá en hojas diferentes.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible
- **Cada pregunta vale 1,5 puntos (sobre un total de 10 de la nota de Teoría).**
- Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.

**1.** Utilizando exclusivamente operaciones constructoras generadoras, definir la sintaxis y la semántica de la operación *comprueba* que actúa sobre un **árbol binario completo** de números naturales e indica si es un HEAP máximo.

**2.** Sea el grafo NO dirigido representado por esta MATRIZ DE ADYACENCIA (se obvian las aristas de la DIAGONAL INFERIOR, ya que son repetitivas):

1    2    3    4    5    6    7    8

			X			X	
					X	X	
			X				
			X	X	X	X	
						X	
						X	

1  
2  
3  
4  
5  
6  
7  
8

a) Realizar el RECORRIDO EN PROFUNDIDAD empezando por el Vértice 1.

b) Realizar el RECORRIDO EN ANCHURA empezando por el Vértice 1.

c) Realizar el BOSQUE EXTENDIDO EN PROFUNDIDAD con su correspondiente CLASIFICACIÓN DE ARCOS.

d) Realizar el BOSQUE EXTENDIDO EN ANCHURA con su correspondiente CLASIFICACIÓN DE ARCOS.

### NOTAS:

- Criterio para el recorrido: adyacencia de cada vértice ordenada de MENOR A MAYOR valor.
- Formato para representar la CLASIFICACIÓN DE ARCOS para los apartados c) y d). Se debe representar en la matriz de adyacencia.

**3.** A partir del árbol AVL representado por el siguiente vector:

20	10	50	5	15	30	60	3		12	18			55								16
----	----	----	---	----	----	----	---	--	----	----	--	--	----	--	--	--	--	--	--	--	----

1    2    3    4    5    6    7    8    9    10    11    12    13    14    15    16    17    18    19    20    21    22

a) Realiza los siguientes borrados de forma consecutiva: Borra(50), Borra(12), Borra(15), Borra(10). Muestra el árbol después de cada borrado e indica las operaciones realizadas. Nota: Si el nodo tiene dos hijos sustituir por el mayor de la izquierda.

b) Dibuja un árbol equilibrado óptimo de altura 4. Justifica tu respuesta.

**4. a)** Realizad el borrado del mínimo en el montículo mínimo cuyo recorrido en inorden es el siguiente: 32, 15, 10, 27, 8, 30, 13, 14, 5, 23, 12, 20, 7, 37, 18, 35

**b)** Escribid en C++ el código de la función (que no pertenece a la clase) “void BorradoMaxEnMonticuloMinimo (MonticuloMinimo& a)” que realiza el borrado del elemento máximo en el montículo mínimo “a” (actualizándolo tras el borrado), utilizando ÚNICAMENTE las operaciones descritas en la parte pública de la clase “MonticuloMinimo”:

```
class MonticuloMinimo {
public:
    MonticuloMinimo& operator=( MonticuloMinimo&); // Operador asignación
    int Tamanyo(void); // Devuelve el número de elementos del montículo
    void Insertar(int elementoAinsertar); // Inserta elementoAinsertar
    int Minimo(void); // Devuelve el mínimo del montículo
    void BorrarMinimo(void); // Borra el mínimo del montículo
};
```

**c)** Aplicad el algoritmo del apartado b) sobre el montículo inicial del apartado a) (antes de realizar el borrado del mínimo). Será suficiente mostrar el resultado final con la explicación de cómo se ha llegado al mismo.

## Examen PED julio 2018. Soluciones

1. Sintaxis:

comprueba(arbin) → bool

Semántica:

Var i1, i2, d1, d2: arbin; u, v1, v2: item;

[enraizar(i,x,d) → E(i,x,d)]

comprueba( E(crear\_arbin(), u, crear\_arbin()) ) = TRUE

comprueba( E(E(crear\_arbin(), v1, crear\_arbin()), u, crear\_arbin()) ) =  
si (u>v1) entonces TRUE  
sino FALSE

comprueba( E(crear\_arbin(), u, E(crear\_arbin(), v1, crear\_arbin())) ) = FALSE //NO HACE FALTA. EL ARBOL DE ENTRADA ES COMPLETO

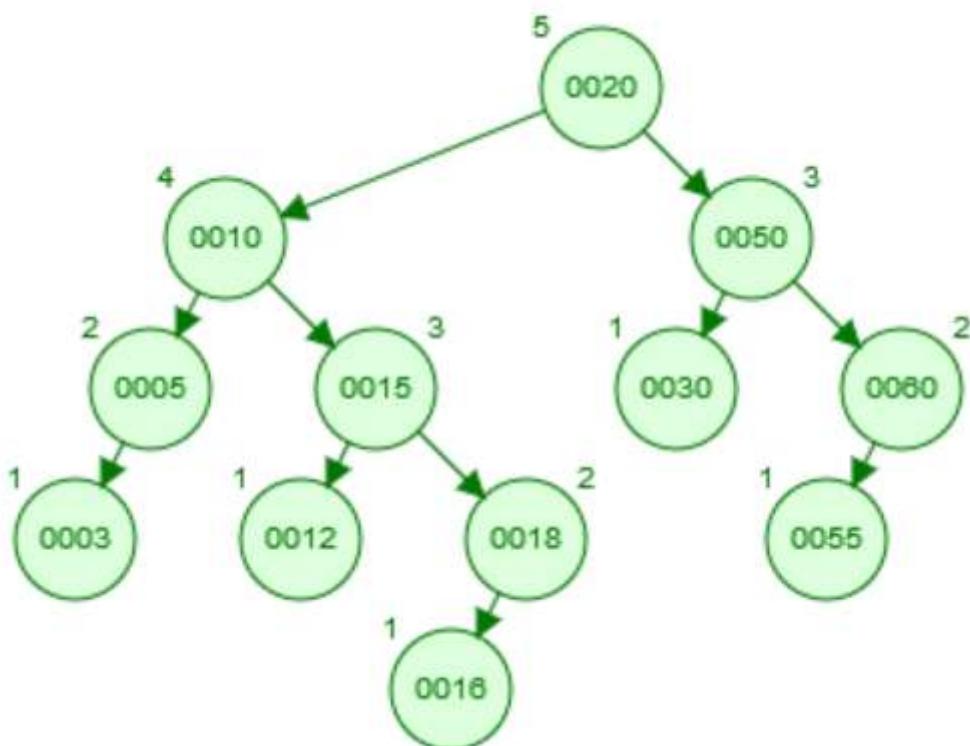
comprueba( E(E(i1, v1, d1), u, E(i2, v2, d2)) ) =

si (u>v1) Y (u>v2) entonces comprueba(E(i1, v1, d1)) Y comprueba(E(i2, v2, d2))

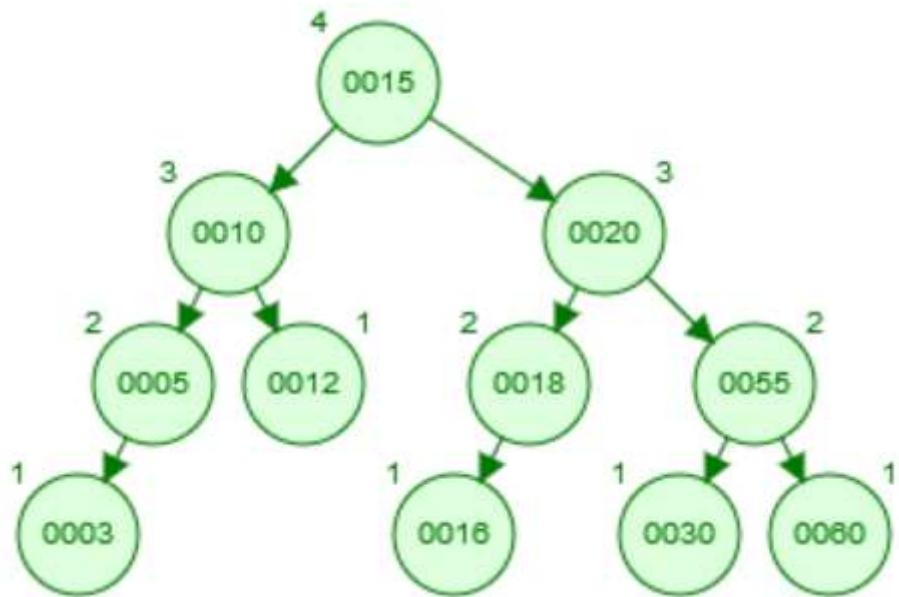
2.

3.

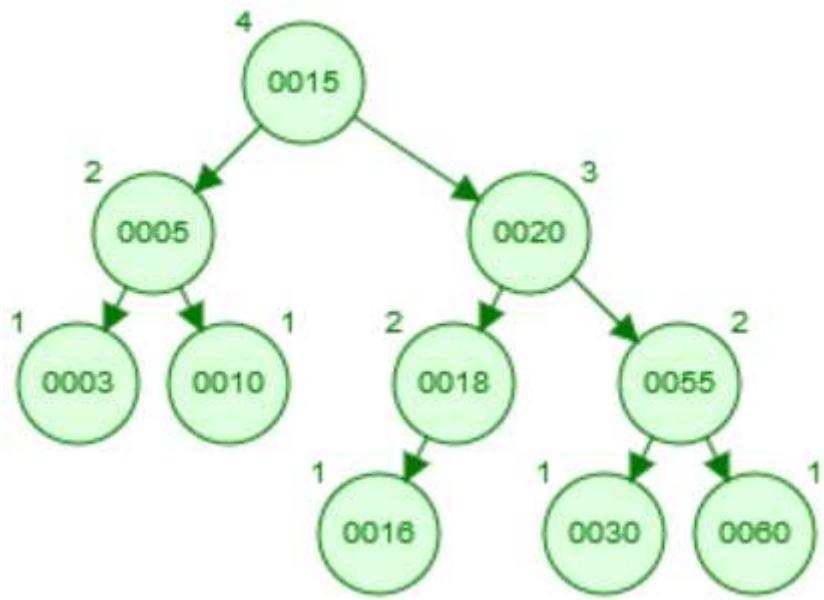
a) AVL INICIAL



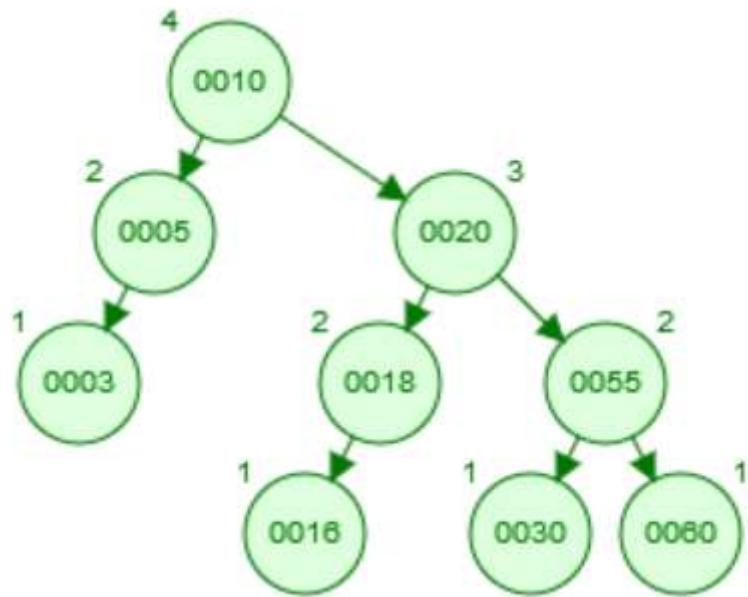
BORRA(50) → ROT DI, ROT ID



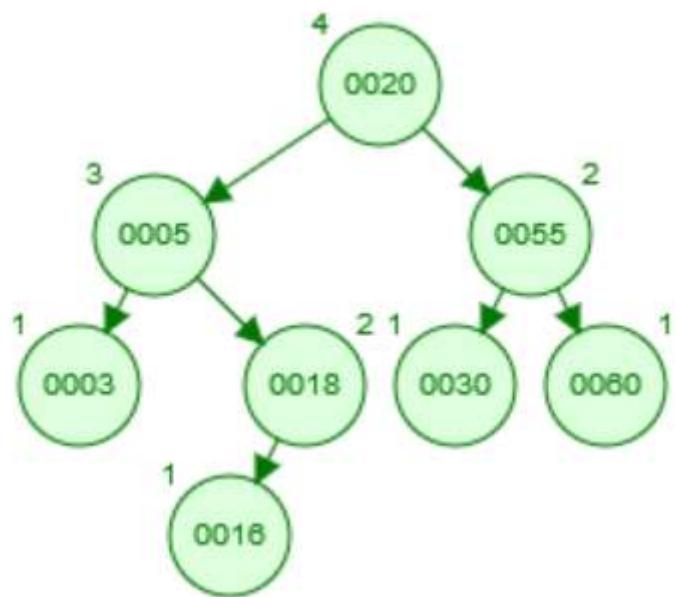
BORRA(12) ROT II



BORRA(15)



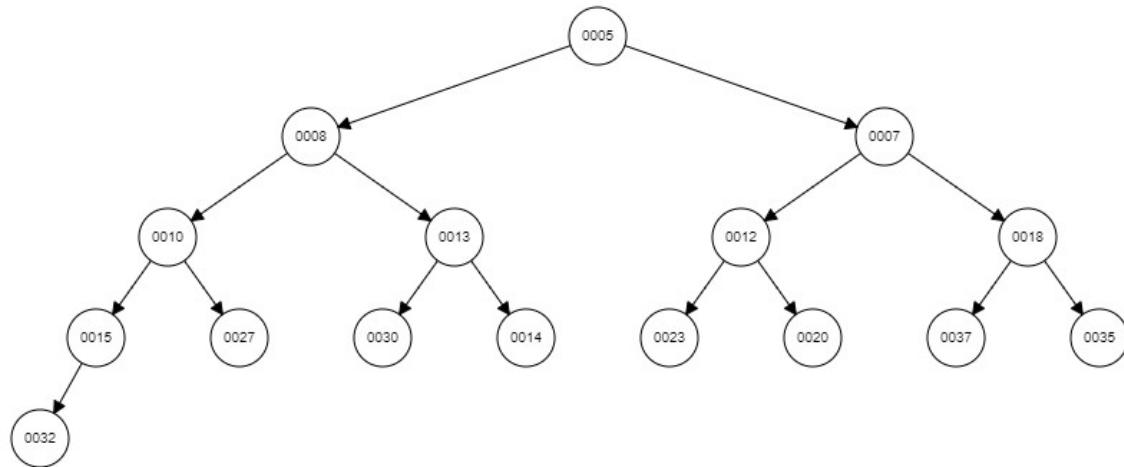
BORRA(10) ROT DD



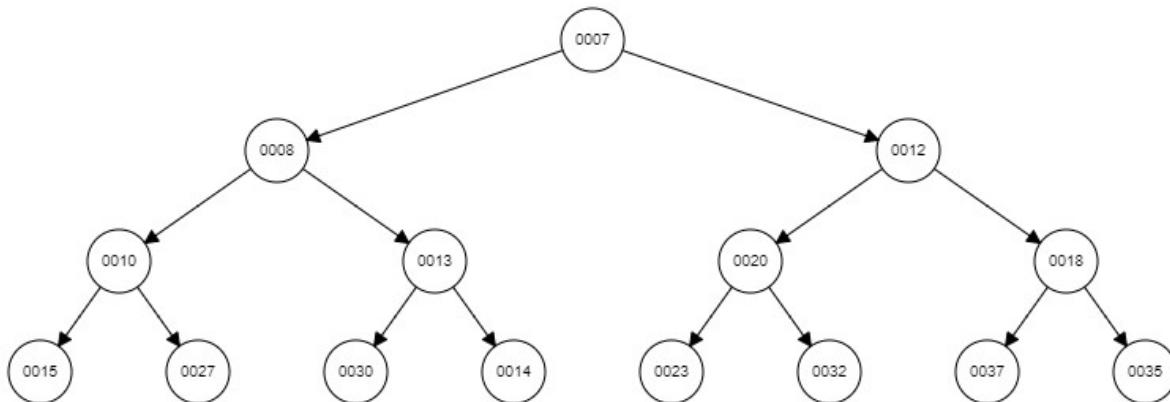
- b) Un árbol equilibrado óptimo es aquel cuyo número de nodos es igual a  $2^h - 1$ . Un árbol equilibrado óptimo de altura 4 se correspondería con un árbol binario de búsqueda lleno con 15 nodos.

4.

- a) Montículo mínimo inicial:



Tras el borrado del mínimo:

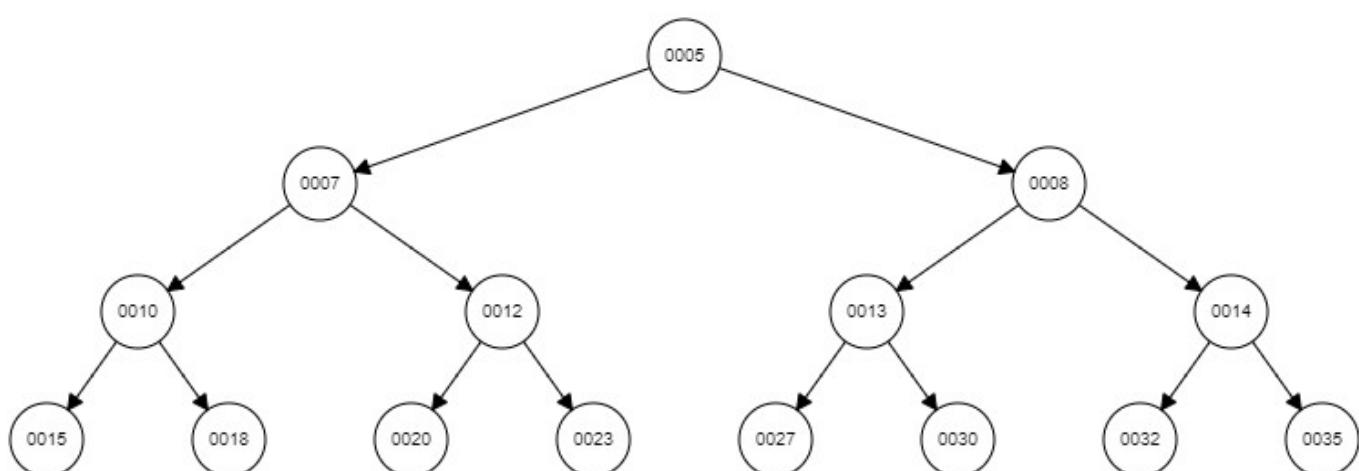


b) Esto se podría realizar de maneras muy diversas, siempre consistiendo en la búsqueda del máximo elemento (el último elemento tras borrados sucesivos) y sus inserciones en el montículo resultante. Pseudocódigo:

```

void BorradoMaxEnMonticuloMinimo (MonticuloMinimo& a) {
    MonticuloMinimo b;
    while(a.Tamanyo() > 1) {
        b.Insertar(a.Minimo());
        a.BorrarMinimo();
    }
    a = b;
}
  
```

c) Resultado de las inserciones: 5, 7, 8, 10, 12, 13, 14, 15, 18, 20, 23, 27, 30, 32, 35



Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED junio 2017

### Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 2 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	V
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V
<input type="checkbox"/>	<input type="checkbox"/>	15	V

En la definición de Tipo Abstracto de Datos:  
"La manipulación de los datos sólo depende del comportamiento descrito en su especificación (qué hace) y es independiente de su implementación (cómo se hace)".

En una cola representada a partir de una lista enlazada simple con un único puntero al principio de la lista (cabeza de la cola), todas las operaciones de la cola (Cabeza, Encolar, Desencolar y EsVacía) tienen una complejidad de O(1).

El máximo número de nodos en un árbol binario de altura k-1 es  $2^k - 1$ ,  $k \geq 1$ .

Dado el recorrido por niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.

Un árbol está equilibrado respecto a la altura si y solo si para cada uno de sus nodos ocurre que las alturas de los dos subárboles difieren como mucho en 1.

Dado un árbol 2-3 de altura h con n items se cumple que:  $3^{h-1} < n < 2^h - 1$

En el algoritmo de borrado en un árbol 2-3-4 siempre que q sea 2-nodo hay que hacer una COMBINACIÓN o una ROTACIÓN.

Utilizando la representación de conjuntos mediante vectores de bits, la operación de búsqueda de un elemento tiene una complejidad espacial equivalente al tamaño del conjunto universal.

La especificación algebraica de la siguiente operación eliminaría todas las ocurrencias de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):

Eliminar(Crear, x)  $\Leftrightarrow$  Crear

Eliminar(Insertar(C, x), y)  $\Leftrightarrow$

si ( $x == y$ ) entonces Eliminar(C, y) sino Insertar(Eliminar(C, y), x)

Sea una tabla de dispersión cerrada con función de dispersión  $H(x)=x \bmod B$ , con  $B=100$  y "x" un número natural entre 1 y 2000. Sólo hay un valor de "x" que haga  $H(x)=4$ .

En la dispersión abierta sólo se producen colisiones entre claves sinónimas.

El TAD Cola de Prioridad representado por una lista desordenada, tendrá coste O(1) para la operación de borrado de un elemento.

En un montículo de altura k el número total de elementos es  $2^k-1$ .

Al representar un grafo de N vértices y K aristas con una lista de adyacencia, la operación de hallar la adyacencia de entrada de un vértice tiene una complejidad de O(K).

Un bosque extendido en profundidad de un grafo no dirigido es un grafo acíclico.

## Examen PED junio 2017

**Normas:** • Tiempo para efectuar el examen: **2 horas y 15 minutos**

- En la cabecera de cada hoja **Y EN ESTE ORDEN** hay que poner: **APELLIDOS, NOMBRE**.
- Cada pregunta se escribirá en hojas diferentes.
- Las soluciones al examen se dejarán en el campus virtual.
- Se puede escribir el examen con lápiz, siempre que sea legible.
- **Cada pregunta vale 1,5 puntos (sobre 10).**
- Las fechas de “Publicación de notas” y “Revisión del examen teórico” se publicarán en el Campus Virtual.

**1.** Utilizando exclusivamente las operaciones constructoras generadoras del tipo natural y la operación MOD entre números naturales, definir la semántica de la operación *primo* (*primo*: *natural* → *bool*) que indica si un número natural es primo.

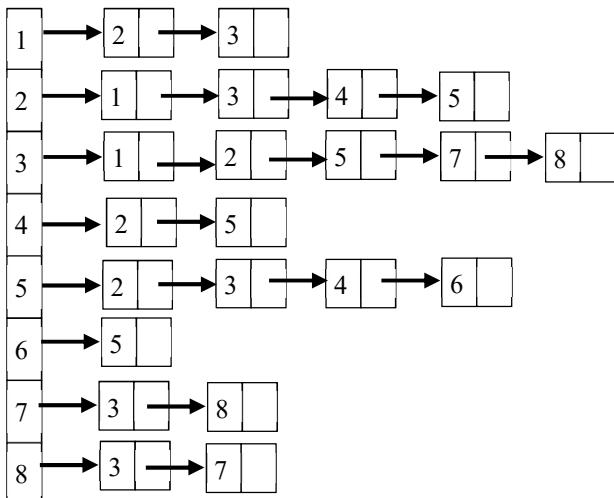
Nota: será necesario definir una operación auxiliar *primoAux* para hacer los cálculos parciales de todos los posibles divisores de un número dado

**2. a)** Insertar en una tabla de dispersión cerrada (Hash) de tamaño  $B=11$ , los elementos: 36, 14, 15, 3, 47, 4, 5, 7, 23, 10, 8. Para ello, debes usar la estrategia de REDISPERSIÓN CERRADA CON 2<sup>a</sup> FUNCIÓN HASH. (Hay que insertar los 11 elementos en la tabla sin reestructurar la misma; al acabar las inserciones la tabla se quedará llena).

**b)** Define la fórmula para reestructurar la tabla Hash de dispersión cerrada dependiendo del número de elementos insertados.

**c)** En este ejercicio, ¿a partir de la inserción de qué elemento habría que reestructurar la tabla? ¿Qué tamaño tendría la tabla, tras la reestructuración? Justifica tus respuestas.

**3.** Dada la siguiente lista de adyacencia (vector de listas) de un grafo **NO** dirigido:



### Aclaraciones:

- Cada lista de adyacencia está ordenada de menor a mayor.
- Los números situados a la izquierda indican el vértice inicial del que se obtiene la lista de adyacencia.
- La lista contiene aristas repetidas.

a) ¿El vértice 6 está conectado únicamente con el vértice 5? Justifica tu respuesta

b) ¿Es un grafo conexo? ¿Por qué?

c) Dados los siguientes recorridos obtenidos a partir de las listas de adyacencia ordenadas de menor a mayor. Indica si son del tipo BFS, DFS o no son recorridos válidos para el grafo no dirigido proporcionado.

R1: 4,2,1,3,5,6,7,8

R2: 4,2,5,3,6,7,8,1

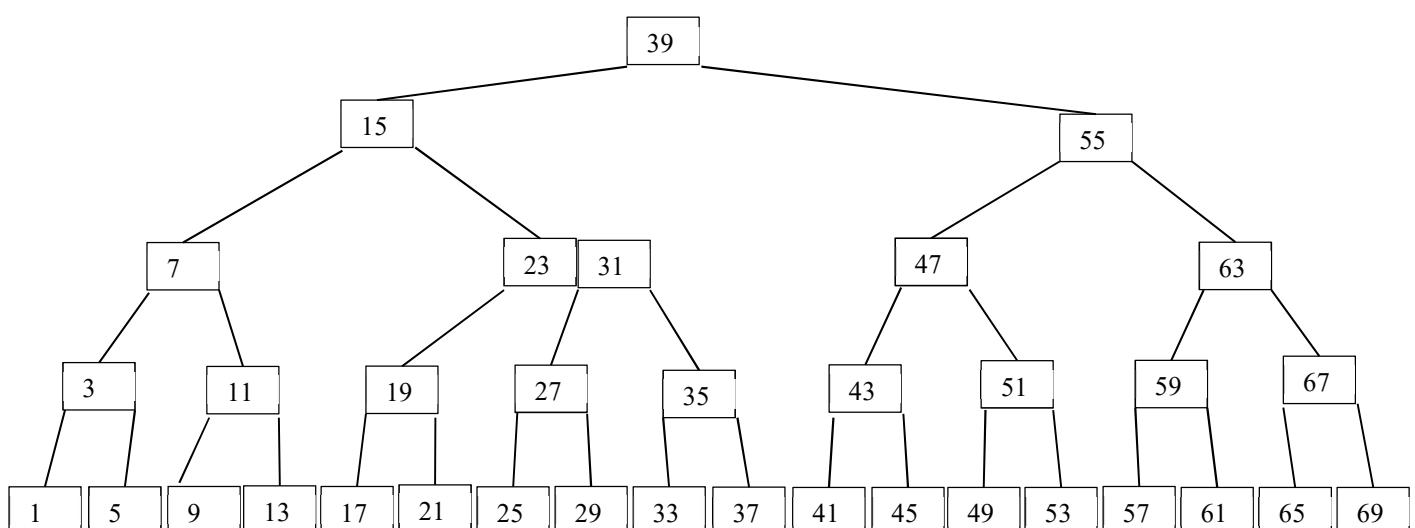
R3: 5,2,3,1,4,7,8,6

R4: 1,2,3,4,5,7,8,6

R5: 7,3,8,1,2,5,4,6

**4.** Sobre el siguiente árbol, realizar el borrado de la clave 15, indicando las operaciones realizadas, con los criterios de sustituir por el mayor del hijo izquierda (cuando se borra un nodo con 2 hijos), y consultar el hermano de la izquierda:

- a) Considerando dicho árbol del tipo 2-3  
 b) Considerando dicho árbol del tipo 2-3-4



## Examen PED junio 2017. Soluciones

1.

primo: natural  $\rightarrow$  bool

primoAux: natural, natural  $\rightarrow$  bool

Var x,y: natural;

primo (cero) = F

primo (suc(cero)) = F

primo (suc(suc(x))) = primoAux ( suc(suc(x)), suc(x) )

Hasta aquí 0,3 puntos

primoAux ( x, suc(cero) ) = T

+0,2 puntos

primoAux ( suc(suc(x)), suc(y) ) = si ( suc(suc(x)) MOD suc(y) ) == 0

entonces F

sino primoAux ( suc(suc(x)), y )

+1 puntos

2.

**36, 14, 15, 3, 47, 4, 5, 7, 23, 10, 8. B=11**

a) (1,1 puntos)

0	47
---	----

1	7
2	23
3	36
4	15
5	5
6	8
7	3
8	14
9	4
10	10

$$H(36) = 3 \quad (1^a)$$

$$H(14) = 3 ; h_1(14) = (3+k(14)) \text{ MOD } 11 = 8 \quad (2^a)$$

$$H(15) = 4 \quad (1^a)$$

$$H(3) = 3 ; h_1(3) = (3+k(3)) \text{ MOD } 11 = 7 \quad (2^a)$$

$$H(47) = 3 ; h_1(47) = (3+k(47)) \text{ MOD } 11 = 0 \quad (2^a)$$

$$H(4) = 4 ; h_1(4) = (4+k(4)) \text{ MOD } 11 = 9 \quad (2^a)$$

$$H(5) = 5 \quad (1^a)$$

$$H(7) = 7 ; h_1(7) = (7+k(7)) \text{ MOD } 11 = 4 ; h_2(7) = (4+k(7)) \text{ MOD } 11 = 1 \quad (3^a)$$

$$H(23) = 1 ; h_1(23) = (1+k(23)) \text{ MOD } 11 = 5 ;$$

$$h_2(23) = (5+k(23)) \text{ MOD } 11 = 9$$

$$h_3(23) = (9+k(23)) \text{ MOD } 11 = 2 \quad (4^a)$$

$$H(10) = 10 \quad (1^a)$$

$$H(8) = 8 ; h_1(8) = (8+k(8)) \text{ MOD } 11 = 6 \quad (2^a)$$

b) (0,1 puntos)

$$\alpha = \frac{n}{|B|}$$

$$n \geq 0,9 * B$$

(H. Cerrado  $\rightarrow$  reestructura cuando  $\alpha \geq 0,9$ )

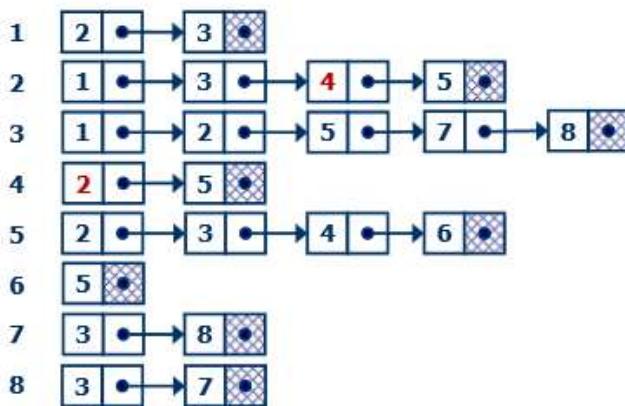
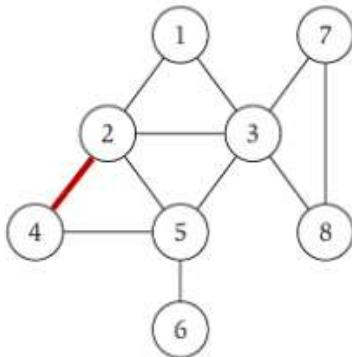
c) (0,3 puntos)

Como  $(0,9 * B) = (0.9 * 11) = 9.9 \rightarrow$  A partir del **elemento 10º**, hay que reestructurar.

**El elemento 10º es el 10.** Antes de insertar el **10**, hay que reestructurar.

Reestructurar a tamaño TEORICO = 22  $\rightarrow$  Como no es primo, hay que reestructurar al primo más cercano = **23**

3.



a) (0,25 puntos) Sí. Porque según la lista de adyacencia existe una única arista que une el vértice 6 y el vértice 5.

b) (0,25 puntos) Sí. Porque para todo  $v_i, v_j$  que pertenece a  $V(G)$  existe un camino de  $v_i$  a  $v_j$  en  $G$

c) (1 puntos) R1: 4,2,1,3,5,6,7,8 DFS

R2: 4,2,5,3,6,7,8,1 No

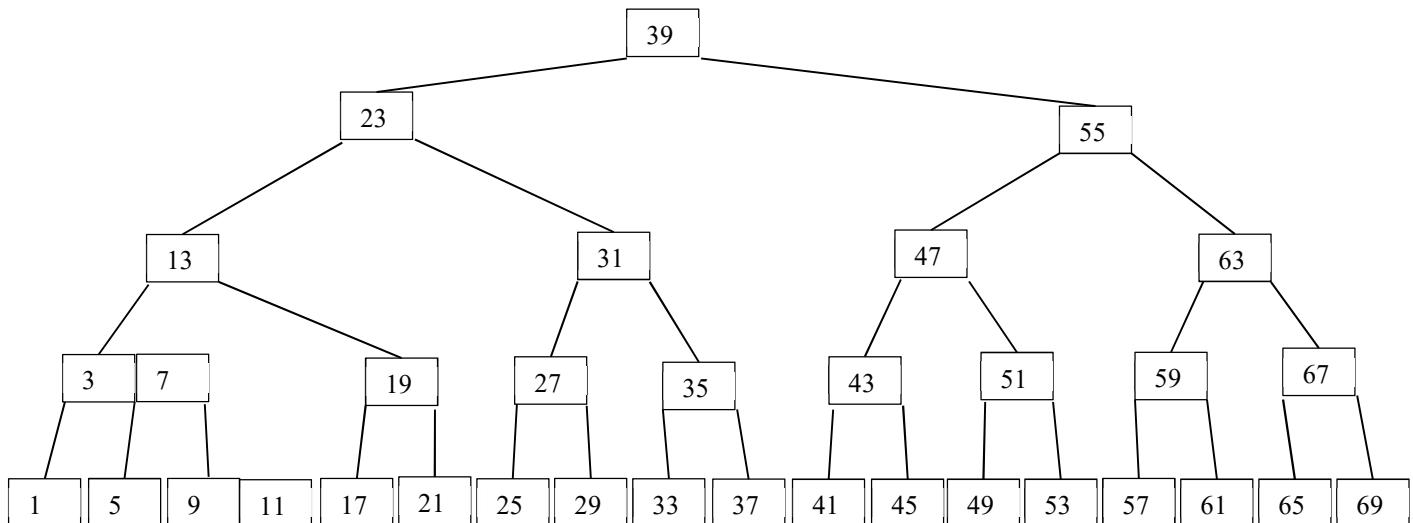
R3: 5,2,3,1,4,7,8,6 No

R4: 1,2,3,4,5,7,8,6 BFS

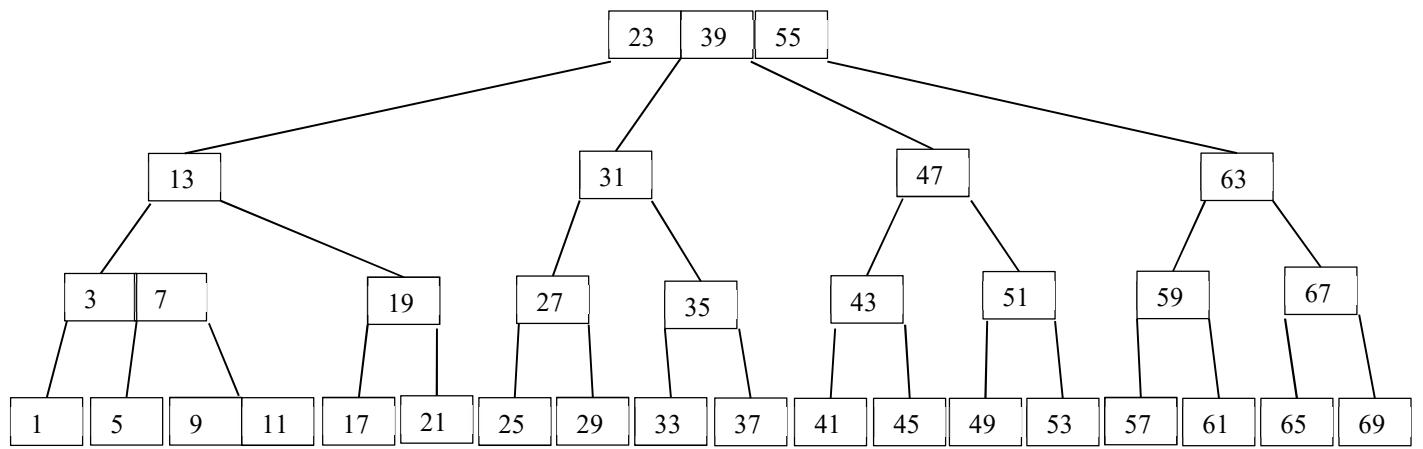
R5: 7,3,8,1,2,5,4,6 BFS

4.

a) (0,75 puntos) 2 combinaciones y 1 rotación



b) (0,75 puntos) 3 combinaciones y 1 rotación



Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **25 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1    V
<input type="checkbox"/>	<input type="checkbox"/>	2    F
<input type="checkbox"/>	<input type="checkbox"/>	3    V
<input type="checkbox"/>	<input type="checkbox"/>	4    F
<input type="checkbox"/>	<input type="checkbox"/>	5    F
<input type="checkbox"/>	<input type="checkbox"/>	6    V
<input type="checkbox"/>	<input type="checkbox"/>	7    V
<input type="checkbox"/>	<input type="checkbox"/>	8    F
<input type="checkbox"/>	<input type="checkbox"/>	9    F
<input type="checkbox"/>	<input type="checkbox"/>	10    F
<input type="checkbox"/>	<input type="checkbox"/>	11    V
<input type="checkbox"/>	<input type="checkbox"/>	12    V
<input type="checkbox"/>	<input type="checkbox"/>	13    F
<input type="checkbox"/>	<input type="checkbox"/>	14    V

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 1

**Normas:**

- \* Tiempo para efectuar el test: **25 minutos**.
- \* Una pregunta mal contestada elimina una correcta.
- \* Las soluciones al examen se dejarán en el campus virtual.
- \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- \* En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Sea un vector de números naturales. La operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con “0”, vista en clase, se define así: eliminar: vector -> vector Var v:vector; i: entero; x:natural; eliminar(crear()) = crear() si (i MOD 2) == 0 entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x) si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)	<input type="checkbox"/>	<input type="checkbox"/>	2 F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras. Diremos que se trata de una operación derivada.	<input type="checkbox"/>	<input type="checkbox"/>	4 V
En la escala de complejidades, la mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo “n” la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	5 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La operación <i>BorrarItem</i> tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) =      Crear BorrarItem( IC(L1,j), i) =    si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista.	<input type="checkbox"/>	<input type="checkbox"/>	8 F
La operación <i>base</i> , vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente: base(pila) -> item Var p: pila; x: item; base(crear()) = error() base(apilar(crear(),x)) = x base(apilar(p,x)) = base(desapilar(p))	<input type="checkbox"/>	<input type="checkbox"/>	9 V
El número mínimo de nodos que tiene un árbol AVL de altura 4 es 7.	<input type="checkbox"/>	<input type="checkbox"/>	10 F
El grado de un nodo es el número máximo de items asociados a dicho nodo.	<input type="checkbox"/>	<input type="checkbox"/>	11 V
El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5      ] el elemento 5 es el hijo izquierda del elemento 8.	<input type="checkbox"/>	<input type="checkbox"/>	13 F
Cuando se realiza un borrado en un árbol AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	14 F
Dado un único recorrido de un árbol binario, es posible reconstruir dicho árbol.			

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013 Modalidad 2

Normas:

- \* Tiempo para efectuar el test: **25 minutos**.
- \* Una pregunta mal contestada elimina una correcta.
- \* Las soluciones al examen se dejarán en el campus virtual.
- \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- \* En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación <i>base</i> , vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente: base(pila) -> item Var p: pila; x: item; base(crear()) = error() base(apilar(crear(),x)) = x base(apilar(p,x)) = base(desapilar(p))	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Sea un vector de números naturales. La operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con “0”, vista en clase, se define así: eliminar: vector -> vector Var v:vector; i: entero; x:natural; eliminar(crear()) = crear() si (i MOD 2) == 0 entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x) si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)	<input type="checkbox"/>	<input type="checkbox"/>	2 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4 F
En la escala de complejidades, la mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo “n” la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	5 F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La operación <i>BorrarItem</i> tiene la siguiente sintaxis y semántica: BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) =      Crear BorrarItem( IC(L1,j), i) =    si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista.	<input type="checkbox"/>	<input type="checkbox"/>	8 V
En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras. Diremos que se trata de una operación derivada.	<input type="checkbox"/>	<input type="checkbox"/>	9 F
Dado un único recorrido de un árbol binario, es posible reconstruir dicho árbol.	<input type="checkbox"/>	<input type="checkbox"/>	10 F
Cuando se realiza un borrado en un árbol AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	11 V
El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5      ] el elemento 5 es el hijo izquierda del elemento 8.	<input type="checkbox"/>	<input type="checkbox"/>	13 F
El grado de un nodo es el número máximo de items asociados a dicho nodo.	<input type="checkbox"/>	<input type="checkbox"/>	14 V

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen TAD/PED diciembre 2004

### Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - **El test vale un 40% de la nota de teoría.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	V 1.
<input type="checkbox"/>	<input type="checkbox"/>	V 2.
<input type="checkbox"/>	<input type="checkbox"/>	F 3.
<input type="checkbox"/>	<input type="checkbox"/>	F 4.
<input type="checkbox"/>	<input type="checkbox"/>	F 5.
<input type="checkbox"/>	<input type="checkbox"/>	V 6.
<input type="checkbox"/>	<input type="checkbox"/>	V 7.
<input type="checkbox"/>	<input type="checkbox"/>	V 8.
<input type="checkbox"/>	<input type="checkbox"/>	V 9.
<input type="checkbox"/>	<input type="checkbox"/>	V 10.
<input type="checkbox"/>	<input type="checkbox"/>	F 11.
<input type="checkbox"/>	<input type="checkbox"/>	V 12.
<input type="checkbox"/>	<input type="checkbox"/>	F 13.
<input type="checkbox"/>	<input type="checkbox"/>	V 14.

En un Tipo Abstracto de Datos la manipulación de los objetos o valores de un tipo, sólo depende del comportamiento descrito en su especificación y es independiente de su implementación.

La complejidad temporal (en su peor caso) de la operación de insertar un elemento en una cola circular enlazada que no admite elementos repetidos es  $O(n)$ .

La complejidad temporal de la operación *apilar* en una pila es independiente de su implementación.

Sea el TIPO *arbin* definido en clase. La semántica de la operación *nodos* es la siguiente:

```
var i,d:arbin; x:item;
nodos(crear_arbin())=0
nodos(enraizar(i,x,d))=nodos(i)+nodos(d)
```

En un árbol AVL, el factor de equilibrio de un nodo se define como el valor absoluto de la altura del subárbol de la derecha menos la altura del subárbol de la izquierda.

El mínimo número de elementos que se puede almacenar en un árbol 2-3 de altura  $h$  coincide con el número de elementos que hay en un árbol binario lleno de altura  $h$ .

Al insertar un único ítem en un árbol 2-3-4 puede ocurrir que haya que realizar dos operaciones: DIVIDERAI (p) y DIVIDEHIJODE2 (p, q).

La inserción de una etiqueta en un árbol rojo-negro se efectuará creando un hijo de color rojo.

Un árbol binario de búsqueda con altura 7 y 127 nodos es un árbol B con  $m=3$

En la operación de búsqueda de un elemento en un conjunto ordenado representado mediante un vector de elementos ordenado (representación no enlazada de una lista), se puede utilizar el algoritmo de búsqueda binaria.

El factor de carga de la dispersión abierta siempre está entre 0 y 1.

En un montículo doble, un elemento “i” del montículo mínimo tiene como máximo un elemento simétrico “j” del montículo máximo.

Un árbol binario de búsqueda cumple las propiedades de un árbol Leftist.

La siguiente secuencia de nodos de un grafo es un ciclo: 1,2,3,1.

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED diciembre 2005 Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - **El test vale un 40% de la nota de teoría.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1. V
<input type="checkbox"/>	<input type="checkbox"/>	2. F
<input type="checkbox"/>	<input type="checkbox"/>	3. F
<input type="checkbox"/>	<input type="checkbox"/>	4. F
<input type="checkbox"/>	<input type="checkbox"/>	5. F
<input type="checkbox"/>	<input type="checkbox"/>	6. V
<input type="checkbox"/>	<input type="checkbox"/>	7. F
<input type="checkbox"/>	<input type="checkbox"/>	8. F
<input type="checkbox"/>	<input type="checkbox"/>	9. F
<input type="checkbox"/>	<input type="checkbox"/>	10. V
<input type="checkbox"/>	<input type="checkbox"/>	11. V
<input type="checkbox"/>	<input type="checkbox"/>	12. V
<input type="checkbox"/>	<input type="checkbox"/>	13. F
<input type="checkbox"/>	<input type="checkbox"/>	14. F

Las colas también se conocen como listas FIFO.

Un árbol binario lleno es también un árbol 2-3-4.

Sea un árbol 2-3-4 inicialmente vacío. Tras utilizar las operaciones de inserción y borrado de un árbol 2-3-4 siempre se consigue un árbol binario lleno.

Un árbol binario completo es un árbol 2-3-4.

Un árbol binario completo de altura  $h$  y con  $2^h - 1$  nodos es un árbol 2-3-4.

Un árbol Rojo – Negro con todos sus enlaces negros tiene los mismos nodos que el árbol 2-3-4 equivalente.

En un árbol B todos los nodos han de tener al menos  $m/2$  hijos o  $(m-1)/2$  claves.

En una tabla Hash con dispersión cerrada, las casillas vacías hay que diferenciarlas de las suprimidas para realizar una inserción.

En un Hash cerrado, el factor de carga ( $\alpha = n/B$ ) puede ser mayor que 1 cuando  $n$  sea mayor que  $B$ .

En un multigrafo pueden existir infinitas aristas para un numero “n” de vértices.

La operación concatena es un enriquecimiento de las operaciones definidas para el tipo cola.

Es posible reconstruir un único árbol AVL a partir de un recorrido por niveles.

El borrado en un árbol AVL nunca requiere más de una rotación en el camino de búsqueda.

En la escala de complejidades, la complejidad cuadrática es menor que la logarítmica.

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED enero 2006

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **35 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En la especificación de un TAD, las operaciones auxiliares son visibles para los usuarios.	<input type="checkbox"/>	<input type="checkbox"/>	1. F
Una operación del TAD X que tenga la sintaxis Crear() →X es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2. V
En C++, los miembros <i>protected</i> son privados para el exterior, pero permiten el acceso a las clases derivadas.	<input type="checkbox"/>	<input type="checkbox"/>	3. V
Dadas las clases <i>TDir</i> y <i>TVectorDir</i> :	<input type="checkbox"/>	<input type="checkbox"/>	4. F
<pre>class TDir {     public: ....     private:         int e1;         char c1; };</pre>	<input type="checkbox"/>	<input type="checkbox"/>	5. F
<pre>class TVectorDir {     public: ....     private:         TDir *vector;         int dim; };</pre>	<input type="checkbox"/>	<input type="checkbox"/>	6. V
<pre>TVectorDir::~TVectorDir () {     if (v!=NULL)         delete v;     dim=0;     v=NULL; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	7. F
¿Es correcta la implementación del destructor de <i>TVectorDir</i> ?	<input type="checkbox"/>	<input type="checkbox"/>	8. F
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input type="checkbox"/>	9. V
Dado un único recorrido de un árbol binario lleno es posible reconstruir dicho árbol	<input type="checkbox"/>	<input type="checkbox"/>	10. V
Sea el tipo <i>arbin</i> definido en clase. La semántica de la operación <i>nodos</i> es la siguiente:  Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	11. F
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	12. V
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	13. V
Un árbol completo siempre está balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	14. V
En un árbol AVL siempre que se inserte una etiqueta hay que realizar una rotación	<input type="checkbox"/>	<input type="checkbox"/>	15. F
El coste temporal en el peor caso de la operación de inserción en un árbol 2-3-4 es $\log_2(n+1) \approx \log_2(n)$ siendo “n” el número total de ítems	<input type="checkbox"/>	<input type="checkbox"/>	16. V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	17. F
La semántica de la operación <i>anterior</i> vista en clase es la siguiente:  VAR L1: lista; x: item; p: posicion; anterior( L1, primera( L1 ) ) = error_posicion(); si p != ultima( L1 ) entonces anterior( L1, siguiente( L1, p ) ) = p anterior( inscabeza( L1, x ), primera( L1 ) ) = primera( inscabeza( L1, x ) )	<input type="checkbox"/>	<input type="checkbox"/>	18. F
Sea el tipo <i>vector</i> definido en clase. La semántica de la operación <i>recu</i> es la siguiente:  Var v:vector; i,j:int; x:item; recu(crear_vector(),i)=error_item(); si i<>j entonces recu(asig(v,i,x),j)=recu(v,j) sino recu(asig(v,i,x),j)=TRUE	<input type="checkbox"/>	<input type="checkbox"/>	
<i>crear_pila()</i> , <i>apilar(pila,item)</i> y <i>desapilar(pila)</i> son operaciones constructoras del tipo <i>pila</i> .	<input type="checkbox"/>	<input type="checkbox"/>	
Todo árbol binario de búsqueda es un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	
La semántica de la operación <i>Base</i> que actúa sobre una <i>pila</i> y devuelve el primer elemento apilado es la siguiente:  Base(crear_pila ()) = crear_pila () Base(apilar(crear_pila (), x)) = x Base(apilar(p, x)) = Base(p)	<input type="checkbox"/>	<input type="checkbox"/>	

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED diciembre 2008

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
Los enriquecimientos forman parte de la definición de un TAD.	<input type="checkbox"/>	<input type="checkbox"/>	1	F
Sea el método <i>Primera</i> perteneciente a la clase <i>TLista</i> que devuelve la primera posición de la lista que lo invoca:	<input type="checkbox"/>	<input type="checkbox"/>	2	F
<pre>TPosicion TLista::Primera() {     TPosicion p;     p.pos = lis;     return p; }</pre>	<pre>class TLista { public: ... private: TNodo *lis; }</pre>			
En el método <i>Primera</i> , se invoca a la sobrecarga del operador asignación entre objetos del tipo <i>TPosicion</i> .	<input type="checkbox"/>	<input type="checkbox"/>	3	F
En C++, las funciones y clases AMIGAS es obligatorio declararlas siempre antes de la sección PUBLIC de la clase	<input type="checkbox"/>	<input type="checkbox"/>	4	F
En la escala de complejidades se cumple que $O(\log n) \subset O(\log \log n)$ .	<input type="checkbox"/>	<input type="checkbox"/>	5	F
Sea el tipo <i>cola</i> definido en clase. La semántica de la operación <i>cabeza</i> es la siguiente:				
$\begin{aligned} Var\ c:cola;\ x:item; \\ cabeza(crear\_cola())=&error\_item() \\ si\ esvacia(c)\ entonces\ cabeza(encolar(c,x))=&x \\ \text{sino}\ cabeza(encolar(c,x)))=&encolar(cabeza(c),x) \end{aligned}$				
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input type="checkbox"/>	6	F
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	7	F
Un árbol completo siempre está balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	8	V
El número mínimo de elementos que se pueden almacenar en un árbol 2-3 de altura $h$ es $2^{h-1}$	<input type="checkbox"/>	<input type="checkbox"/>	9	V
Para que decrezca la altura de un árbol 2-3-4 en una operación de borrado, el nodo raíz y sus hijos tienen que ser 2-nodo	<input type="checkbox"/>	<input type="checkbox"/>	10	V
Un árbol rojo-negro es un árbol binario balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	11	F
El árbol 2-3 es un árbol B m-camino de búsqueda con $m=2$	<input type="checkbox"/>	<input type="checkbox"/>	12	F
Sea una tabla de dispersión cerrada con estrategia de redispersión $h_i(x)=(H(x) + C*i) \bmod B$ , con $B=1000$ y $C=74$ . Para cualquier clave “ $x$ ” se recorrerán todas las posiciones de la tabla buscando una posición libre.	<input type="checkbox"/>	<input type="checkbox"/>	13	F
Para todo nodo de un árbol Leftist, se cumple que el número de nodos de su hijo izquierdo es menor que el de su hijo derecho.	<input type="checkbox"/>	<input type="checkbox"/>	14	F
En un multigrafo pueden existir infinitas aristas para un número “ $n$ ” de vértices.	<input type="checkbox"/>	<input type="checkbox"/>	15	V

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED diciembre 2009

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **22 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1    F
Para el siguiente fragmento de código C++ de un posible método perteneciente a la conocida clase TCoordenada, la línea "delete b;" liberaría correctamente la memoria dinámica de <i>b</i> .		
<pre>void Funcion(void) {     TCoordenada *a = new TCoordenada;     TCoordenada *b = new TCoordenada[5];     (. . . . .)         delete b; }</pre>		
El resultado del cálculo de la complejidad temporal en el mejor caso de un algoritmo X, da como resultado $n + n^*\log(n)$ . Por lo tanto, diremos que la complejidad del algoritmo X cuando $n \rightarrow \infty$ pertenece a $\Omega(n)$ .		
Las pilas también se conocen como listas LIFO.		
Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol.		
A los árboles generales también se les llama árboles multicamino de búsqueda.		
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho solo se va a efectuar una rotación.		
La altura de un árbol 2-3 únicamente crece cuando se inserta un elemento y todos los nodos del árbol son 3-nodo.		
Con las operaciones de inserción y borrado es posible conseguir un árbol 2-3-4 de altura 4 con todos sus nodos de tipo 2-nodo.		
Las operaciones de transformación cuando se inserta un elemento en un árbol 2-3-4, en el caso de un árbol rojo-negro, se reducen a cambios de colores o rotaciones.		
El árbol 2-3 es un árbol B m-camino de búsqueda con m=2.		
La dispersión abierta elimina el problema del clustering secundario.		
Sea una tabla de dispersión cerrada con estrategia de redispersión $h_i(x) = (H(x) + C*i) \bmod B$ , con $B=1000$ y $C=74$ . Para cualquier clave "x" se recorrerán todas las posiciones de la tabla buscando una posición libre cuando se inserta el elemento.		
El siguiente árbol es un montículo máximo:		
<pre> graph TD     10 --- 7     10 --- 9     7 --- 5     7 --- 2     9 --- 6     9 --- 4     9 --- 1     8 --- 8   </pre>		
Para todo nodo de un árbol Leftist, se cumple que el número de nodos de su hijo izquierdo es mayor o igual que el de su hijo derecho.		
Un grafo no dirigido de $n$ vértices es un árbol si está libre de ciclos y tiene $n-1$ aristas.		
<input type="checkbox"/>	<input type="checkbox"/>	13    F
<input type="checkbox"/>	<input type="checkbox"/>	14    F
<input type="checkbox"/>	<input type="checkbox"/>	15    V

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED diciembre 2010

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1    F
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:		
<pre>TPosicion TLista::Primera() { TPosicion p;   p.pos = lis;   return p; }</pre>		
En la línea resaltada, se invoca a la sobrecarga del operador asignación entre objetos del tipo TPosicion.		
En la escala de complejidades se cumple que $O(n \log n) \subset O(n^2)$ .		
La operación BorrarItem tiene la siguiente sintaxis y semántica:		
BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) =      Crear BorrarItem( IC(L1,j), i) =    si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )		
Esta operación borra la primera ocurrencia del ítem que se encuentra en la lista		
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo		
A los árboles generales también se les llama árboles multicamino de búsqueda		
Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, sólo se va a efectuar una rotación como mucho		
Dado un árbol 2-3 con $n$ ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es $O(\log_2 n)$		
Un árbol Rojo-Negro es una representación sobre árbol binario de un árbol 2-3		
Un árbol rojo-negro es un árbol binario balanceado respecto a la altura		
La raíz del árbol B m-camino de búsqueda siempre tiene al menos $m/2$ claves o etiquetas		
La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):		
$\text{Eliminar}(\text{Crear}, x) \Leftrightarrow \text{Crear}$ $\text{Eliminar}(\text{Insertar}(C, x), y) \Leftrightarrow$ $\quad \quad \quad \text{si } (x == y) \text{ entonces } C \text{ sino Insertar}(\text{Eliminar}(C, y), x)$		
En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14: $h_i(14) = (28 + 7*i) \text{ MOD } 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.		
El TAD Cola de Prioridad representado por un montículo, tendrá las siguientes complejidades: $O(1)$ para el borrado, y $O(\log n)$ para la inserción, siendo $n$ el número de elementos.		
En un árbol binario lleno, el camino mínimo de la raíz (longitud del camino más corto hasta un árbol vacío) es igual a la altura del árbol.		
La altura máxima de un árbol de búsqueda digital es “n+1”, siendo n el número de bits de la clave.		
<input type="checkbox"/>	<input type="checkbox"/>	12    V
<input type="checkbox"/>	<input type="checkbox"/>	13    F
<input type="checkbox"/>	<input type="checkbox"/>	14    V
<input type="checkbox"/>	<input type="checkbox"/>	15    V

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED junio 2011

### Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V

EsVacia: PILA -> BOOLEAN  
Si P y Q son pilas: Q = EsVacia ( P ), es una expresión sintácticamente correcta

En C++, cuando se sobrecarga un operador que no modifica al operando izquierdo (por ejemplo : "+") se debe crear un objeto temporal, que luego el método devuelve por valor

La complejidad temporal (en su caso promedio) del siguiente fragmento de código es  $\Theta(n^2)$

```
int i, length, n, i1, i2, k;
for (i = 0, length = 1; i < n-1; i++) {
    for (i1 = i2 = k = i; k < n-1 && a[k] < a[k+1]; k++, i2++);
    if (length < i2 - i1 + 1) length = i2 - i1 + 1;
}
```

En cualquier tipo de datos lineal cada elemento tiene como máximo un único sucesor y un único predecesor

El máximo número de nodos en un árbol binario de altura  $k-1$  es  $2^k - 1$ ,  $k \geq 1$ .

En la inserción, en el peor de los casos, las rotaciones realizadas en los árboles AVL para mantenerlos balanceados tienen un coste temporal lineal respecto al número de ítems del árbol

El borrado de un elemento en un árbol 2-3 se realiza en las hojas. Se pueden producir reestructuraciones del árbol en el camino de vuelta desde las hojas a la raíz del árbol.

En un árbol 2-3-4 de altura=2 y número de elementos=15, si se insertara un nuevo elemento se tendría que hacer un DIVIDERAZ y un DIVIDEHIJODE2

Un árbol rojo-negro es un árbol B con  $m=2$

Todo árbol binario de búsqueda es un árbol B con  $m=4$ .

La complejidad temporal en su peor caso de la operación de Unión entre 2 conjuntos con  $m$  elementos cada uno y representados con una lista desordenada es  $O(m^2)$ .

En dos tablas de dispersión cerrada y abierta con tamaños  $B=7$  y  $B=6$  respectivamente, siempre se cumple que el factor de carga en la abierta es mayor que en la cerrada.

Todo árbol binario que además es árbol mínimo es un Heap Mínimo

La complejidad temporal, en su peor caso, de la operación de PERTENECE de un elemento de tamaño  $N$  en un árbol de búsqueda digital es  $O(N+1)$ .

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED julio 2011

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	F

En C++, la expresión `return &c;` devuelve la dirección de memoria de la variable `c`.

En C++, una función no puede tener todos sus parámetros con valores por omisión o por defecto.

En la escala de complejidades se cumple que  $O(\log n) \subset O(\log \log n)$ .

La operación `BorrarItem`, que borra todas las ocurrencias del ítem `i` que se encuentren en la lista, tiene la siguiente sintaxis y semántica:

```

BorrarItem: LISTA, ITEM -> LISTA
BorrarItem( Crear, i) =      Crear
BorrarItem( IC(L1,j), i) =   si ( i == j ) entonces BorrarItem ( L1, i )
                           sino IC ( BorrarItem ( L1, i ), j )

```

Un árbol con un único nodo tiene un único camino cuya longitud es 1

En cualquier tipo de datos árbol, cada elemento puede tener varios predecesores, pero como máximo un sucesor.

El siguiente árbol está balanceado con respecto a la altura

Si se inserta un elemento en un árbol 2-3 y todos los nodos que están en el camino desde la raíz a la hoja donde se inserta el elemento son del tipo 3-nodo, la altura del árbol 2-3 resultante crece con respecto al árbol 2-3 original.

En un árbol 2-3-4 de altura 3 donde todos sus nodos son del tipo 3-nodo, el número de elementos total es 27.

En un árbol rojo-negro, el número de enlaces negros ha de ser mayor que el de enlaces rojos.

El nodo de un árbol B m-camino de búsqueda con  $m=100$  puede tener como máximo 99 claves.

La complejidad temporal, en su peor caso, de la operación de Unión entre 2 conjuntos con  $m$  elementos cada uno y representados con una lista ordenada es  $O(m)$ .

En el Hash cerrado la tabla de dispersión de tamaño  $B$  se tiene que reestructurar cuando se cumpla que el número de elementos  $n \geq 2B$ .

En un TRIE la complejidad temporal en su peor caso de la función `Pertenece` es  $O(n)$  siendo  $n$  el número de nodos del árbol

Apellidos:

Nombre:

Convocatoria:

DNI:

# Examen PED junio 2012. Grado en Informática

## Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	V
<input type="checkbox"/>	<input type="checkbox"/>	14	V

En C++, la instrucción "int &a = 1 ; " daría error de compilación

En C++, cuando se emplea layering, desde la clase A que contiene un objeto de la clase B siempre se puede acceder a la parte privada del objeto contenido de la clase B

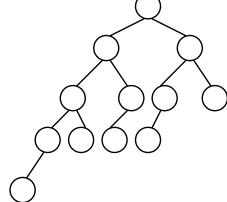
Paso de programa es una secuencia de operaciones con contenido semántico cuyo coste es dependiente de la talla del problema

La semántica de la operación cima del tipo pila vista en clase es la siguiente:  
 VAR p: pila, e: ítem;  
 cima( crear() ) = error()  
 cima( apilar( p, e ) ) = cima( p )

Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol

En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el mayor del subárbol de la izquierda o por el menor del subárbol de la derecha

El siguiente árbol está balanceado con respecto a la altura



El borrado de un elemento en un árbol 2-3 se realiza en las hojas. Se pueden producir reestructuraciones del árbol aplicando el algoritmo descendente que empieza en la raíz del árbol y finaliza en las hojas.

En un árbol 2-3-4 el máximo número elementos del nivel N es  $3*2^{N-1}$

La especificación algebraica de la siguiente operación eliminaría todas las claves repetidas de un determinado ítem (C: ConjuntoConClavesRepetidas; x, y: Ítem):

Eliminar(Crear, x)  $\Leftrightarrow$  Crear  
 Eliminar(Insertar(C, x), y)  $\Leftrightarrow$   
 si (x == y) entonces C sino Insertar(Eliminar(C, y), x)

En el TAD Diccionario con dispersión cerrada, los elementos se almacenan en una tabla de tamaño fijo.

Todo Heap Mínimo cumple las condiciones de ser un árbol binario y un árbol mínimo

En un multigrafo pueden existir infinitas aristas para un número “n” de vértices.

En un grafo dirigido con K arcos (el número máximo de arcos en el grafo) y N vértices, una complejidad de  $O(K)$  es equivalente a la complejidad de  $O(N^2)$ .

**Examen PED junio 2012.**  
**Grado en Ingeniería Informática**  
**Modalidad 0**

**Normas:** • La entrega del test no corre convocatoria.

• Tiempo para efectuar el test: **20 minutos**.

• Una pregunta mal contestada elimina una correcta.

• Las soluciones al examen se dejarán en el campus virtual.

• **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**

• En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	V
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	F
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	F
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V
<input type="checkbox"/>	<input type="checkbox"/>	15	F

Longitud: LISTA -> NATURAL  
Si L es una lista, a es un ítem de la lista: a = Longitud ( L ) es un uso sintácticamente incorrecto de la operación

En layering los métodos de la clase derivada pueden acceder a la parte pública de la clase base.

En C++, el valor de la variable q al finalizar este fragmento de código es 7:

```
int q = 0;
int i;
for(i = 1; i < 5; i = i + 1)
    if(i != q)
        q += i;
```

En la escala de complejidades se cumple que  $O(\log n) \subset O(\log \log n)$ .

En cualquier tipo de datos lineal cada elemento tiene un único sucesor y varios predecesores

Un árbol binario completo con n nodos y altura k es un árbol binario lleno para esa misma altura

El menor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja

Los árboles AVL son aquellos en los que el número de elementos en los subárboles izquierdo y derecho difieren como mucho en 1

Un árbol 2-3 es un árbol 2-ario de búsqueda

El árbol 2-3-4 no vacío tiene como mínimo una clave en cada nodo

En la representación de conjuntos mediante las listas el espacio es proporcional al tamaño del conjunto universal.

En el TAD Diccionario con dispersión abierta, la operación de búsqueda de una clave tiene una complejidad  $O(L)$ , con  $L$ =longitud de la lista de claves sinónimas colisionadas.

El montículo o HEAP mínimo es un árbol binario lleno que además es árbol mínimo.

Un grafo no dirigido de n vértices es un árbol si está libre de ciclos y tiene “n-1” aristas

Al representar un grafo dirigido de N vértices y K aristas con una matriz de adyacencia, la operación de búsqueda de una arista tiene una complejidad de  $O(N)$ .

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **25 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1    V
<input type="checkbox"/>	<input type="checkbox"/>	2    F
<input type="checkbox"/>	<input type="checkbox"/>	3    V
<input type="checkbox"/>	<input type="checkbox"/>	4    F
<input type="checkbox"/>	<input type="checkbox"/>	5    F
<input type="checkbox"/>	<input type="checkbox"/>	6    V
<input type="checkbox"/>	<input type="checkbox"/>	7    V
<input type="checkbox"/>	<input type="checkbox"/>	8    F
<input type="checkbox"/>	<input type="checkbox"/>	9    F
<input type="checkbox"/>	<input type="checkbox"/>	10    F
<input type="checkbox"/>	<input type="checkbox"/>	11    V
<input type="checkbox"/>	<input type="checkbox"/>	12    V
<input type="checkbox"/>	<input type="checkbox"/>	13    F
<input type="checkbox"/>	<input type="checkbox"/>	14    V

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 1

**Normas:**

- \* Tiempo para efectuar el test: **25 minutos**.
- \* Una pregunta mal contestada elimina una correcta.
- \* Las soluciones al examen se dejarán en el campus virtual.
- \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- \* En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Sea un vector de números naturales. La operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con “0”, vista en clase, se define así:  eliminar: vector -> vector  Var v:vector; i: entero; x:natural; eliminar(crear()) = crear() si (i MOD 2) == 0 entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x) si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)	<input type="checkbox"/>	<input type="checkbox"/>	2 F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras. Diremos que se trata de una operación derivada.	<input type="checkbox"/>	<input type="checkbox"/>	4 V
En la escala de complejidades, la mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo “n” la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	5 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La operación <i>BorrarItem</i> tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) = Crear BorrarItem( IC(L1,j), i) = si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista.	<input type="checkbox"/>	<input type="checkbox"/>	8 F
La operación <i>base</i> , vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente:  base(pila) -> item  Var p: pila; x: item; base(crear()) = error() base(apilar(crear(),x)) = x base(apilar(p,x)) = base(desapilar(p))	<input type="checkbox"/>	<input type="checkbox"/>	9 V
El número mínimo de nodos que tiene un árbol AVL de altura 4 es 7.	<input type="checkbox"/>	<input type="checkbox"/>	10 F
El grado de un nodo es el número máximo de items asociados a dicho nodo.	<input type="checkbox"/>	<input type="checkbox"/>	11 V
El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5  ] el elemento 5 es el hijo izquierda del elemento 8.	<input type="checkbox"/>	<input type="checkbox"/>	13 F
Cuando se realiza un borrado en un árbol AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	14 F
Dado un único recorrido de un árbol binario, es posible reconstruir dicho árbol.			

Apellidos, Nombre:  
DNI:

## Examen PED abril 2013

### Modalidad 2

Normas:

- \* Tiempo para efectuar el test: **25 minutos**.
- \* Una pregunta mal contestada elimina una correcta.
- \* Las soluciones al examen se dejarán en el campus virtual.
- \* **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- \* En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación <i>base</i> , vista en clase, que actúa sobre una pila y devuelve la base de la pila (el primer elemento que se ha apilado) es la siguiente:  base(pila) -> item  Var p: pila; x: item; base(crear()) = error() base(apilar(crear(),x)) = x base(apilar(p,x)) = base(desapilar(p))	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Sea un vector de números naturales. La operación <i>eliminar</i> que borra las posiciones pares del vector marcándolas con “0”, vista en clase, se define así:  eliminar: vector -> vector  Var v:vector; i: entero; x:natural; eliminar(crear()) = crear() si (i MOD 2) == 0 entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x) si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)	<input type="checkbox"/>	<input type="checkbox"/>	2 F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En C++, después de invocar el destructor (~NombreClase) de un objeto, no se puede acceder a los miembros (propiedades y métodos) de dicho objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4 F
En la escala de complejidades, la mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo “n” la talla del problema.	<input type="checkbox"/>	<input type="checkbox"/>	5 F
En C++, el puntero this sólo se puede usar dentro de los métodos de la clase.	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La operación <i>BorrarItem</i> tiene la siguiente sintaxis y semántica:  BorrarItem: LISTA, ITEM -> LISTA BorrarItem( Crear, i) = Crear BorrarItem( IC(L1,j), i) =     si ( i == j ) entonces L1 sino IC ( BorrarItem ( L1, i ), j )	<input type="checkbox"/>	<input type="checkbox"/>	7 V
Esta operación borra la primera ocurrencia del item que se encuentra en la lista.	<input type="checkbox"/>	<input type="checkbox"/>	8 V
En general, las operaciones modificadoras y consultoras se especifican en términos de las generadoras. En ocasiones, una operación modificadora puede especificarse en términos de otras modificadoras o consultoras. Diremos que se trata de una operación derivada.	<input type="checkbox"/>	<input type="checkbox"/>	9 F
Dado un único recorrido de un árbol binario, es posible reconstruir dicho árbol.	<input type="checkbox"/>	<input type="checkbox"/>	10 F
Cuando se realiza un borrado en un árbol AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho sólo se va a efectuar una rotación.	<input type="checkbox"/>	<input type="checkbox"/>	11 V
El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	12 V
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5  ] [ ] el elemento 5 es el hijo izquierda del elemento 8.	<input type="checkbox"/>	<input type="checkbox"/>	13 F
El grado de un nodo es el número máximo de items asociados a dicho nodo.	<input type="checkbox"/>	<input type="checkbox"/>	14 V

Apellidos, Nombre:

DNI:

## Examen PED abril 2014

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
La operación <i>palindromo</i> (sobre un vector) vista en clase es la siguiente:  Var v: vector; i,x: natural; <i>palindromo(crear_vector())</i> = VERDADERO <i>palindromo(asig(v,i,x))</i> = si $i \leq 50$ entonces si <i>recu(v,100-i+1)</i> == x entonces <i>palindromo(asig(v,i,x))</i> sino FALSO sino VERDADERO	<input type="checkbox"/>	<input type="checkbox"/>	1    F
Una operación del TAD X que tenga la sintaxis <i>Crear() →X</i> es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2    V
En C++, al hacer layering los métodos de la clase derivada pueden acceder a la parte pública y privada de la clase base.	<input type="checkbox"/>	<input type="checkbox"/>	3    F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4    V
El algoritmo de intercambio directo o burbuja estudiado en clase (ordenación de los elementos de un vector) tiene una complejidad promedio de $\Theta(n^2)$ , siendo n el número de elementos del vector.	<input type="checkbox"/>	<input type="checkbox"/>	5    V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.	<input type="checkbox"/>	<input type="checkbox"/>	6    V
La complejidad temporal en el peor caso para la inserción de un elemento en una lista ordenada y en otra no ordenada, que no permiten elementos repetidos, siempre es lineal con el número de elementos en ambos casos.	<input type="checkbox"/>	<input type="checkbox"/>	7    V
El tipo de datos vector (visto en clase) se define como un conjunto en el que sus componentes ocupan posiciones consecutivas de memoria.	<input type="checkbox"/>	<input type="checkbox"/>	8    F
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:  Var i,d:arbin; x:item; <i>nodos(crear_arbin())=0</i> <i>nodos(eraizar(i,x,d))=nodos(i)+nodos(d)</i>	<input type="checkbox"/>	<input type="checkbox"/>	9    F
Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.	<input type="checkbox"/>	<input type="checkbox"/>	10    V
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	11    V
A los árboles generales también se les llama árboles multicamino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	12    F
Las rotaciones que hay que realizar en los árboles AVL para mantenerlos balanceados tienen un coste temporal (en su peor caso) lineal con respecto al número de items del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	13    F
El grado de los árboles AVL puede ser +1, 0 ó -1.	<input type="checkbox"/>	<input type="checkbox"/>	14    F

Apellidos, Nombre:

DNI:

## Examen PED marzo 2015

### Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	V
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	V
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	V
<input type="checkbox"/>	<input type="checkbox"/>	14	F

En la especificación algebraica, las operaciones constructoras se clasifican en generadoras y modificadoras.			
Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes:   $\text{VAR } x, y: \text{natural};$    $\text{suma}(x, \text{cero}) = x$    $\text{suma}(\text{cero}, x) = x$    $\text{suma}(x, \text{suc}(y)) = \text{suma}(x, y)$			
Dentro de la especificación algebraica de los números naturales, definimos la sintaxis de la función F como: F: natural  $\rightarrow$  BOOL, y su semántica como: F(cero)=TRUE, F(suc(cero))=FALSE, F(suc(suc(x)))=F(x). Para el número natural x=35, la función F devolvería FALSE. Nota: se asume que x=35 es la forma simplificada de indicar x=suc(suc(...suc(cero)...))).			
Todo árbol binario de altura 9 y 511 nodos es un árbol binario lleno y además es árbol binario de búsqueda.			
Sea el método Primera perteneciente a la clase TLista que devuelve la primera posición de la lista que lo invoca:			
``` TPosicion TLista::Primera() {     TPosicion p;     p.pos = primero;     return p; } ```  ``` class TLista { public: ... private: TNodo *primero; } ```			
En el método Primera, se invoca de forma implícita a los constructores de TPosicion y TLista.			
En C++, si la variable p es un puntero a un objeto, entonces la expresión p.f() es sintácticamente correcta.			
La complejidad temporal del siguiente fragmento de código es O(n)			
``` int i, j, n, sum; for (i = 4; i < n; i++) {     for (j = i-3, sum = a[i-4]; j <= i; j++) sum += a[j];     cout << "La suma del subarray " << i-4 << " es " << sum << endl; } ```			
La mejor complejidad temporal que se puede conseguir en un algoritmo es O(n), siendo "n" la talla del problema.			
Es posible reconstruir un único árbol binario de búsqueda a partir de su recorrido en postorden			
El máximo número de nodos en un nivel i-1 de un árbol binario es  $2^{i-2}$ ,  $i \geq 2$			
Un camino en un árbol es una secuencia  $a_1, \dots, a_s$  de árboles tal que para todo  $i \in \{1, \dots, s-1\}$ ,  $a_{i+1}$  es subárbol de  $a_i$ .			
El grado de un árbol es el máximo nivel que pueden tener sus subárboles.			
La operación desencolar vista en clase es la siguiente:			
**VAR** c: cola, x: ítem;   $\text{desencolar( crear( ) ) = crear( )}$   **si** esvacia( c ) **entonces** desencolar( encolar( c, x ) ) = crear( )  **si no** desencolar( encolar( c, x ) ) = encolar( desencolar( c ), x )			
El ítem medio (según la relación de orden en la búsqueda) almacenado en un árbol binario de búsqueda siempre se encuentra en la raíz.			

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED enero 2006

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **35 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En la especificación de un TAD, las operaciones auxiliares son visibles para los usuarios.	<input type="checkbox"/>	<input type="checkbox"/>	1. F
Una operación del TAD X que tenga la sintaxis Crear() →X es una operación constructora generadora.	<input type="checkbox"/>	<input type="checkbox"/>	2. V
En C++, los miembros <i>protected</i> son privados para el exterior, pero permiten el acceso a las clases derivadas.	<input type="checkbox"/>	<input type="checkbox"/>	3. V
Dadas las clases <i>TDir</i> y <i>TVectorDir</i> :	<input type="checkbox"/>	<input type="checkbox"/>	4. F
<pre>class TDir {     public: ....     private:         int e1;         char c1; };</pre>	<input type="checkbox"/>	<input type="checkbox"/>	5. F
<pre>class TVectorDir {     public: ....     private:         TDir *vector;         int dim; };</pre>	<input type="checkbox"/>	<input type="checkbox"/>	6. V
<pre>TVectorDir::~TVectorDir () {     if (v!=NULL)         delete v;     dim=0;     v=NULL; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	7. F
¿Es correcta la implementación del destructor de <i>TVectorDir</i> ?	<input type="checkbox"/>	<input type="checkbox"/>	8. F
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo	<input type="checkbox"/>	<input type="checkbox"/>	9. V
Dado un único recorrido de un árbol binario lleno es posible reconstruir dicho árbol	<input type="checkbox"/>	<input type="checkbox"/>	10. V
Sea el tipo <i>arbin</i> definido en clase. La semántica de la operación <i>nodos</i> es la siguiente: Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)	<input type="checkbox"/>	<input type="checkbox"/>	11. F
A los árboles generales también se les llama árboles multicamino de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	12. V
El ítem medio (según la relación de orden) almacenado en un árbol binario de búsqueda lleno siempre se encuentra en la raíz.	<input type="checkbox"/>	<input type="checkbox"/>	13. V
Un árbol completo siempre está balanceado respecto a la altura	<input type="checkbox"/>	<input type="checkbox"/>	14. V
En un árbol AVL siempre que se inserte una etiqueta hay que realizar una rotación	<input type="checkbox"/>	<input type="checkbox"/>	15. F
El coste temporal en el peor caso de la operación de inserción en un árbol 2-3-4 es $\log_2(n+1) \approx \log_2(n)$ siendo “n” el número total de ítems	<input type="checkbox"/>	<input type="checkbox"/>	16. V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	17. F
La semántica de la operación <i>anterior</i> vista en clase es la siguiente: VAR L1: lista; x: ítem; p: posición; anterior( L1, primera( L1 ) ) = error_posicion(); si p != ultima( L1 ) entonces anterior( L1, siguiente( L1, p ) ) = p anterior( inscabeza( L1, x ), primera( L1 ) ) = primera( inscabeza( L1, x ) )	<input type="checkbox"/>	<input type="checkbox"/>	18. F
Sea el tipo <i>vector</i> definido en clase. La semántica de la operación <i>recu</i> es la siguiente: Var v:vector; i,j:int; x:ítem; recu(crear_vector(),i)=error_item(); si i<>j entonces recu(asig(v,i,x),j)=recu(v,j) sino recu(asig(v,i,x),j)=TRUE	<input type="checkbox"/>	<input type="checkbox"/>	
<i>crear_pila()</i> , <i>apilar(pila,item)</i> y <i>desapilar(pila)</i> son operaciones constructoras del tipo <i>pila</i> .	<input type="checkbox"/>	<input type="checkbox"/>	
Todo árbol binario de búsqueda es un árbol 2-3-4	<input type="checkbox"/>	<input type="checkbox"/>	
La semántica de la operación <i>Base</i> que actúa sobre una <i>pila</i> y devuelve el primer elemento apilado es la siguiente: Base(crear_pila ()) = crear_pila () Base(apilar(crear_pila (), x)) = x Base(apilar(p, x)) = Base(p)	<input type="checkbox"/>	<input type="checkbox"/>	

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED febrero 2007

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **35 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V

En C++ y cuando se emplea composición (*layering*), los métodos de la clase derivada pueden acceder a la parte pública de la clase base.

En C++, el puntero *this* no se puede emplear para modificar el objeto al que apunta.

En C++, los constructores se pueden invocar explícitamente cuando el programador lo desee (por ejemplo: *TLista a; a.TLista();*).

En C++, la siguiente declaración es incorrecta: *int& a = I;*

En la escala de complejidades se cumple que  $O(\log n) \subset O(\log \log n)$ .

El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de  $\Omega(1)$ .

Para un vector de naturales, se define la operación *eliminar* que borra las posiciones pares del vector marcándolas con "0" (para calcular el resto de una división, se puede utilizar la operación MOD). La sintaxis y la semántica de la operación *eliminar* es la siguiente:

```
eliminar: vector → vector
Var v:vector; i: entero; x:natural;
eliminar(crear_vector()) = crear_vector()
si (i MOD 2) == 0
entonces eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,0)
si no eliminar(asignar(v,i,x)) = asignar(eliminar(v),i,x)
```

La semántica de la operación *ultima* vista en clase es la siguiente:

```
VAR L1: lista; x: item;
si esvacia( L1 ) entonces
ultima( inscabeza( L1, x ) = primera( L1 )
si no ultima( inscabeza( L1, x ) = ultima( L1 )
```

Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en preorden con 62 etiquetas.

La sintaxis y semántica de la operación *quita\_hojas*, que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas, es la siguiente:

```
quita_hojas(arbin) → arbin
VAR i, d: arbin; x: item;
quita_hojas(crea_arbin()) = crea_arbin()
quita_hojas(enraizar(crea_arbin(), x, crea_arbin())) = crea_arbin()
quita_hojas(enraizar(i, x, d)) =
enraizar(quita_hojas(i), x, quita_hojas(d))
```

Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 2 y 1 hoja es posible reconstruir 2 árboles binarios.

Todo árbol AVL es un árbol 2-3-4

La operación (*DIVIDEHIJODE2 (p, q)*) en la inserción de un elemento en un árbol 2-3-4 puede aumentar la altura del árbol original.

En el algoritmo del borrado de un elemento en un árbol 2-3-4 si *q* es 2-nodo y *r* es 3-nodo hay que hacer una ROTACIÓN.

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED febrero 2008

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **30 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F		
La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.	<input type="checkbox"/>	<input type="checkbox"/>	1	V
Dada la sintaxis de la función $IC(lista,item) \rightarrow lista$ , que inserta un elemento a la cabeza de la lista pasada como parámetro y $crear() \rightarrow lista$ , que crea una lista vacía. La siguiente secuencia: $IC(IC(crear(),a),b),c$ , daría como resultado una lista con los elementos en este orden: $a \rightarrow b \rightarrow c$ , donde $a$ es el primer elemento de la lista	<input type="checkbox"/>	<input type="checkbox"/>	2	F
Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como: $F: natural \rightarrow BOOL$ , y su semántica como: $F(cero)=TRUE$ , $F(suc(cero))=FALSE$ , $F(suc(suc(x)))=F(x)$ . Para el número natural $x=35$ , la función F devolvería TRUE.	<input type="checkbox"/>	<input type="checkbox"/>	3	F
En C++, si un objeto se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4	V
En C++, el constructor de copia recibe como argumento un objeto del mismo tipo pasado por referencia o por valor.	<input type="checkbox"/>	<input type="checkbox"/>	5	F
El algoritmo de búsqueda binaria estudiado en clase (búsqueda de un elemento en un vector ordenado) tiene una complejidad de $\Omega(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	6	V
La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución	<input type="checkbox"/>	<input type="checkbox"/>	7	V
La operación BorrarItem, que borra todas las ocurrencias del ítem i que se encuentren en la lista, tiene la siguiente sintaxis y semántica:	<input type="checkbox"/>	<input type="checkbox"/>	8	V
BorrarItem: LISTA, ITEM $\rightarrow$ LISTA BorrarItem( Crear, i) = Crear BorrarItem( IC(L1,j), i) = si ( i == j ) entonces BorrarItem (L1, i ) sino IC ( BorrarItem (L1, i ), j )				
La semántica de la operación obtener en una lista con acceso por posición es la siguiente (IC= InsertarCabeza(Lista, Ítem), p: posición, l1: lista, x: ítem):	<input type="checkbox"/>	<input type="checkbox"/>	9	F
obtener(crear(),p)=error_item() si p == primera(IC(l1,x)) entonces obtener(IC(l1,x),p)=x sino obtener(IC(l1,x),p)=IC(obtener(l1,p),x)				
El máximo número de nodos en un nivel $i-1$ de un árbol binario es $2^{i-2}$ , $i \geq 2$	<input type="checkbox"/>	<input type="checkbox"/>	10	V
Sea el TIPO arbin definido en clase. La semántica de la operación nodos es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/>	11	F
Var i,d:arbin; x:item; nodos(crear_arbin())=0 nodos(enraizar(i,x,d))=nodos(i)+nodos(d)				
El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es lineal con la altura del árbol	<input type="checkbox"/>	<input type="checkbox"/>	12	V
Se puede obtener un único árbol 2-3-4 a partir de su recorrido por niveles	<input type="checkbox"/>	<input type="checkbox"/>	13	V

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen PED febrero 2009

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **25 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	F
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F

En C++, al declarar una clase "A" como AMIGA de otra clase "B" , todas las funciones miembro de "B" automáticamente pasan a ser funciones AMIGAS de "A"

En C++, si una clase "B" se construye por composición (layering) , a partir de otra clase "A" , definiendo un objeto miembro de la clase "A" en su parte privada, al invocar al constructor de "B" se invoca antes al constructor de "A" y luego al de "B"

Las funciones y clases amigas se tienen que declarar en la parte pública de la clase.

Para el siguiente algoritmo, la complejidad temporal en su peor caso sería O(1):

```
for (i=0; i<100; i++)
    for (j=0; j<100; j++)
        if (v[i]<v[j]) v[i]=v[j];
        else v[j]=v[i];
```

La complejidad temporal en su peor caso del siguiente fragmento de código es O(n)

```
int i, j, n, sum;
for (i = 4; i < n; i++) {
    for (j = i-3, sum = a[i-4]; j <= i; j++) sum += a[j];
    cout << "La suma del subarray " << i-4 << " es " << sum << endl; }
```

La semántica de la operación esvacíapos del tipo vector vista en clase es la siguiente:

```
VAR v: vector; i, j: int; x: item;
esvacíapos( crear(), i ) = CIERTO
esvacíapos( asig( v, i, x ), j )
    si ( i == j ) entonces FALSO
    si no esvacíapos( v, j ) fsi
```

La semántica de la operación anterior vista en clase es la siguiente:

```
VAR L1: lista; x: item; p: posicion;
anterior( L1, primera( L1 ) ) = error_posicion();
si p != ultima( L1 ) entonces anterior( L1, siguiente( L1, p ) ) = p
anterior( inscabeza( L1, x ), primera( L1 ) ) = L1
```

La sintaxis y semántica de la operación simétricos, que comprueba que 2 árboles binarios son simétricos, es la siguiente:

```
simétricos(arbin, arbin) → bool
VAR i1, d1, i2, d2: arbin; x, y: item;
simétricos(enraizar(i1, x, d1), crea_arbin() ) = FALSO
simétricos(crea_arbin(), enraizar(i1, x, d1)) = FALSO
simétricos(enraizar(i1, x, d1), enraizar(i2, y, d2)) =
    si (x == y) entonces (simétricos(i1, d2) & simétricos (d1, i2))
    sino FALSO
```

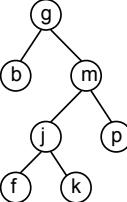
Un árbol binario completo es un AVL

Apellidos, Nombre:  
DNI:

## Examen PED enero 2010

### Modalidad 0

- Normas:**
- La entrega del test **no** corre convocatoria.
  - Tiempo para efectuar el test: **15 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En C++, el valor de la variable q al finalizar este fragmento de código es 11:  <pre>int q = 0; int k = 5; do {     q += k;     k++; } while(q &lt; 7);</pre>	<input type="checkbox"/>	<input type="checkbox"/>	1 V
La complejidad temporal (en su caso mejor) del siguiente fragmento de código es $\Omega(n)$  <pre>int i, length, n, i1, i2, k; for (i = 0, length = 1; i &lt; n-1; i++) {     for (i1 = i2 = k = i; k &lt; n-1 &amp;&amp; a[k] &lt; a[k+1]; k++, i2++);     if (length &lt; i2 - i1 + 1) length = i2 - i1 + 1; }</pre>	<input type="checkbox"/>	<input type="checkbox"/>	2 V
La semántica de la operación insertar del tipo lista vista en clase es la siguiente:  <i>VAR L1: lista; x,y: item; p: posicion;</i>  <i>insertar( crear(), p, x ) = crear()</i> <i>si p == primera( inscabeza( L1, x ) ) entonces</i> <i>    insertar( inscabeza( L1, x ), p, y ) = inscabeza( inscabeza( L1, x ), y )</i> <i>si no insertar( inscabeza( L1, x ), p, y ) = inscabeza( insertar( L1, p, y ), x )</i>	<input type="checkbox"/>	<input type="checkbox"/>	3 F
El grado de un árbol es el grado mínimo de todos los nodos de ese árbol	<input type="checkbox"/>	<input type="checkbox"/>	4 F
El siguiente árbol es binario de búsqueda	<input type="checkbox"/>	<input type="checkbox"/>	5 F
 <pre> graph TD     g((g)) --&gt; b((b))     g((g)) --&gt; m((m))     b((b)) --&gt; f((f))     b((b)) --&gt; j((j))     j((j)) --&gt; k((k))     j((j)) --&gt; p((p))   </pre>			
Dada la siguiente representación secuencial del árbol binario A, [14 8 19 5  ] el elemento 5 es el hijo izquierda del elemento 8	<input type="checkbox"/>	<input type="checkbox"/>	6 V
En el algoritmo de borrado de un elemento de un árbol AVL, tenemos que actualizar los factores de equilibrio de todos los nodos que han intervenido en la búsqueda del elemento a borrar	<input type="checkbox"/>	<input type="checkbox"/>	7 V
En el algoritmo del borrado de un elemento en un árbol 2-3-4 siempre que q sea 2-nodo hay que hacer una reestructuración.	<input type="checkbox"/>	<input type="checkbox"/>	8 V
El árbol 2-3-4 no vacío tiene como mínimo dos claves en cada nodo	<input type="checkbox"/>	<input type="checkbox"/>	9 F
La operación <i>BorrarItem</i> , que borra todas las ocurrencias del item <i>i</i> que se encuentren en la lista, tiene la siguiente sintaxis y semántica:  <i>BorrarItem: LISTA, ITEM -&gt; LISTA</i> <i>BorrarItem( Crear, i ) = Crear</i> <i>BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces BorrarItem ( L1, i )</i> <i>                              sino IC ( BorrarItem ( L1, i ), j )</i>	<input type="checkbox"/>	<input type="checkbox"/>	10 V

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen TAD/PED julio 2010

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - El test vale un 40% de la nota de teoría: **4 puntos**.
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
<input type="checkbox"/>	<input type="checkbox"/>	2	F
<input type="checkbox"/>	<input type="checkbox"/>	3	F
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	F
<input type="checkbox"/>	<input type="checkbox"/>	8	V
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	F
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	V
<input type="checkbox"/>	<input type="checkbox"/>	14	V
<input type="checkbox"/>	<input type="checkbox"/>	15	F
<input type="checkbox"/>	<input type="checkbox"/>	16	F

Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como:  $F: \text{natural} \rightarrow \text{BOOL}$ , y su semántica como:  $F(\text{cero})=\text{TRUE}$ ,  $F(\text{suc}(\text{cero}))=\text{FALSE}$ ,  $F(\text{suc}(\text{suc}(x)))=F(x)$ . Para el número natural  $x=35$ , la función F devolvería FALSE.

En C++, la memoria que se reserva con new se libera automáticamente por el destructor.

La mejor complejidad temporal que se puede conseguir en un algoritmo es  $O(n)$ , con "n" como la talla del problema.

La semántica de la operación *sublista* del tipo lista vista en clase es la siguiente:

```

VAR L: lista; x, y: item; n: natural; p: posicion;
sublista( L, p, 0 ) = crear()
sublista( crear(), p, n ) = crear()
si p == primera( inscabeza( L, x ) ) entonces
    sublista( inscabeza( L, x ), p, n ) = inscabeza( sublista( L, primera( L ), n ), x )
    si no sublista( inscabeza( L, x ), p, n ) = sublista( L, p, n )

```

Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol

Cuando realizamos un recorrido en inorden en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor

Todo árbol binario de búsqueda es un árbol mínimo

El siguiente árbol está balanceado con respecto a la altura

Todo árbol binario de búsqueda es un árbol 2-3.

En un árbol 2-3-4 sólo los nodos hoja y la raíz pueden ser de tipo 2-nodo

En un árbol rojo-negro la búsqueda de una etiqueta dependerá de los colores de los hijos de cada nodo

El árbol 2-3 es un árbol B m-camino de búsqueda con  $m=2$

El TAD Diccionario es un subtipo del TAD Conjunto

La dispersión abierta elimina el problema del clustering secundario.

En un montículo doble de altura  $h$  se pueden almacenar un máximo de  $2^{h-2}$  claves.

Un árbol leftist mínimo es un árbol binario mínimo tal que si no es vacío:  $\text{CMÍN}(\text{HijoIzq}(x)) > \text{CMÍN}(\text{HijoDer}(x))$  para todo  $x$  no vacío

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen TAD/PED julio 2004

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - **El test vale un 40% de la nota de teoría.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	<b>V</b>	<b>F</b>						
La operación crear_pila() es constructora modificadora.	<input type="checkbox"/>	<input type="checkbox"/>	F 1.					
En C++, una forma correcta de copiar una cadena es la siguiente: <pre>char a[50] = "Tipos Abstractos de Datos"; char *b; b = new char[strlen(a)]; strcpy(b, a);</pre>	<input type="checkbox"/>	<input type="checkbox"/>	F 2.					
El caso peor de la búsqueda es más eficiente en una lista ordenada que en una lista cuyos elementos no están ordenados.	<input type="checkbox"/>	<input type="checkbox"/>	F 3.					
En un árbol binario cada elemento puede tener como máximo dos predecesores.	<input type="checkbox"/>	<input type="checkbox"/>	F 4.					
Si se implementa el algoritmo de ordenación de un vector “heapsort” utilizando un heap máximo los elementos quedan ordenados en el vector de forma descendente.	<input type="checkbox"/>	<input type="checkbox"/>	F 5.					
Dado un recorrido en preorder (RID) de un árbol AVL es posible reconstruir un único árbol AVL.	<input type="checkbox"/>	<input type="checkbox"/>	V 6.					
El número máximo de elementos que se puede almacenar en un árbol 2-3 de altura h es $3^{h-1}$	<input type="checkbox"/>	<input type="checkbox"/>	V 7.					
Al borrar un elemento en un árbol 2-3-4 se puede realizar una operación de DIVIDERAZ.	<input type="checkbox"/>	<input type="checkbox"/>	F 8.					
En un árbol B m-camino de búsqueda con m=16: en cualquier nodo excepto la raíz hay 8 ítems como mínimo.	<input type="checkbox"/>	<input type="checkbox"/>	F 9.					
La especificación algebraica de la siguiente operación permite la inserción de claves repetidas (C: ConjuntoConClavesRepetidas; x, y: Ítem): $\text{Insertar}(\text{Insertar}(C, x), y) \Leftrightarrow \text{Insertar}(\text{Insertar}(C, y), x)$	<input type="checkbox"/>	<input type="checkbox"/>	V 10.					
En el TAD Diccionario con dispersión cerrada, con función de redispersión “ $h_i(x) = (H(x) + k(x)*i) \text{ MOD } B$ ”, con B=6 se puede dar la situación de que en una búsqueda no se acceda a todas las posiciones de la tabla.	<input type="checkbox"/>	<input type="checkbox"/>	V 11.					
El siguiente vector representa un montículo máximo:	<input type="checkbox"/>	<input type="checkbox"/>	V 12.					
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">10</td><td style="padding: 5px;">5</td><td style="padding: 5px;">3</td><td style="padding: 5px;">1</td><td style="padding: 5px;">2</td></tr> </table>	10	5	3	1	2			
10	5	3	1	2				
Para todo nodo de un árbol Leftist, se cumple que la altura de su hijo izquierdo es menor que la de su hijo derecho.	<input type="checkbox"/>	<input type="checkbox"/>	F 13.					
La complejidad temporal de la búsqueda en un trie es lineal respecto al número de palabras almacenadas	<input type="checkbox"/>	<input type="checkbox"/>	F 14.					
Un bosque extendido en profundidad de un grafo no dirigido es un grafo acíclico.	<input type="checkbox"/>	<input type="checkbox"/>	V 15.					

Apellidos:  
Nombre:  
Convocatoria:  
DNI:

## Examen TAD/PED Junio 2005

### Modalidad 0

Normas:

- La entrega del test no corre convocatoria.
- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
- **El test vale un 40% de la nota de teoría.**
- En la HOJA DE CONTESTACIONES el verdadero se corresponderá con la A y el falso con la B.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1 F
<input type="checkbox"/>	<input type="checkbox"/>	2 V
<input type="checkbox"/>	<input type="checkbox"/>	3 F
<input type="checkbox"/>	<input type="checkbox"/>	4 V
<input type="checkbox"/>	<input type="checkbox"/>	5 V
<input type="checkbox"/>	<input type="checkbox"/>	6 F
<input type="checkbox"/>	<input type="checkbox"/>	7 V
<input type="checkbox"/>	<input type="checkbox"/>	8 V
<input type="checkbox"/>	<input type="checkbox"/>	9 F
<input type="checkbox"/>	<input type="checkbox"/>	10 V
<input type="checkbox"/>	<input type="checkbox"/>	11 F
<input type="checkbox"/>	<input type="checkbox"/>	12 F
<input type="checkbox"/>	<input type="checkbox"/>	13 F
<input type="checkbox"/>	<input type="checkbox"/>	14 F

Siempre que se realiza una rotación DD en el borrado de un elemento en un árbol AVL decrece la altura del subárbol sobre el que se realiza la rotación.

Un árbol 2-3 es un árbol  $m$ -camino de búsqueda con  $m=3$ .

La complejidad temporal (en su peor caso) de buscar un elemento en un vector ordenado utilizando un algoritmo de búsqueda binaria es  $O(\log_2 n)$  [siendo  $n$  el número de elementos].

Un árbol rojo-negro, en el que no hay ningún enlace rojo, es un árbol binario lleno

En una tabla de dispersión cerrada con la siguiente función de redispersión para la clave 14:  $h_i(14) = (28 + 7*i) \bmod 2000$ , se recorrerán todas las posiciones de la tabla buscando una posición libre.

En el algoritmo HeapSort, después del primer paso de insertar todos los elementos en un Heap representado como un vector, los elementos del mismo quedan ordenados.

Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de avance y de cruce es un grafo acíclico dirigido.

Sea  $G=(V,A)$  un grafo dirigido. Diremos que  $G''=(V'',A'')$  es un árbol extendido de  $G \Leftrightarrow V''=V$ ,  $A'' \subset A$ ,  $\forall v \in V'' \Rightarrow \text{grado}_E(v) \leq 1$ .

La operación *BorrarItem*, que borra la primera ocurrencia del ítem  $i$  que se encuentra en la lista, tiene la siguiente sintaxis y semántica:

*BorrarItem: LISTA, ITEM -> LISTA*  
*BorrarItem( Crear, i ) = Crear*  
*BorrarItem( IC(L1,j), i ) = si ( i == j ) entonces BorrarItem ( L1, i )*  
*sino IC ( BorrarItem ( L1, i ), j )*

Se puede reconstruir un único árbol binario de búsqueda teniendo sus recorridos en preorden y postorden.

La estructura de datos árbol aparece porque los elementos que lo constituyen mantienen una estructura jerárquica, obtenida a partir de estructuras lineales, al eliminar el requisito de que cada elemento tiene como máximo un predecesor.

Al realizar un recorrido en inorder de un montículo obtenemos una sucesión de claves ordenadas.

La función de redispersión en una tabla hash abierta tiene que cumplir como condición para que se recorran todas las posiciones del vector que el valor de  $B$  sea primo

Un grafo que tiene componentes fuertemente conexas es un grafo necesariamente libre de ciclos

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED junio 2008

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **15 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	V
Las ecuaciones (vistas en clase) que permiten realizar la multiplicación de números naturales son las siguientes:			
$VAR\ x,\ y: natural;$ $mult(cero,\ x) = cero$ $mult(x,\ cero) = cero$ $mult(suc(y),\ x) = suma(mult(y,\ x),\ x)$			
En C++ y cuando se emplea composición (layering), los métodos de la clase derivada pueden acceder a la parte protegida de la clase base.			
Para el siguiente algoritmo, la complejidad sería $O(n^2)$ :			
<pre>for (i=0; i&lt;100; i++)     for (j=0; j&lt;100; j++)         if (v[i]&lt;v[j]) v[i]=v[j];         else v[j]=v[i];</pre>			
Dados los recorridos de preorden, postorden y niveles de un árbol binario sólo se puede reconstruir un único árbol binario.			
Sabiendo que A es un árbol binario de búsqueda completo y dado su recorrido inorden 1,4,6,7,9,12,14,20,23. La secuencia 12,7,20,4,9,14,23,1,6, se corresponde con su recorrido por niveles.			
Cuando se borra un elemento en un AVL habrá casos en los que no sea necesario reestructurar el árbol. Para aquellos en los que sí se necesita reestructuración, se realizará una única rotación.			
Dado un árbol 2-3, si la clave a borrar $x$ está en un nodo no hoja, $x$ se podría sustituir por la clave anterior a $x$ en el recorrido en inorden del árbol, y continuar con el algoritmo de borrado.			
La altura $h$ de un árbol 2-3-4 con $n$ elementos se encuentra entre los límites $\log_4 (n+1) \geq h \geq \log_2 (n+1)$ .			
Un árbol binario completo es un árbol 2-3-4			
Un árbol Rojo – Negro de altura 2 puede tener todos sus enlaces de color rojo			
En la inserción en un árbol B se realiza un recorrido descendente desde la raíz en el que se van dividiendo los nodos llenos que nos encontraremos por el camino hasta las hojas.			
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	V
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	V
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	F

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED junio 2010

### Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	F
En C++, la expresión return *c; devuelve la dirección de memoria de la variable c.			
En un multigrafo pueden existir infinitas aristas para un número “n” de vértices.			
La semántica de la operación obtener del tipo lista vista en clase es la siguiente:			
VAR L1: lista; x: item; p: posicion;			
<pre>obtener( crear( ), p ) = error_item() si p == primera( inscabeza( L1, x ) ) entonces     obtener( inscabeza( L1, x ), p ) = x     si no obtener( inscabeza( L1, x ), p ) = inscabeza( obtener( L1, p ), x )</pre>			
<input type="checkbox"/>	<input type="checkbox"/>	4	F
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	V
<input type="checkbox"/>	<input type="checkbox"/>	7	F
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	F
<input type="checkbox"/>	<input type="checkbox"/>	10	F
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	F
<input type="checkbox"/>	<input type="checkbox"/>	13	V
El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo			
A los árboles generales también se les llama árboles multicamino de búsqueda			
Un árbol binario de búsqueda equilibrado respecto a la altura tiene una complejidad temporal en su peor caso en la búsqueda de $O(\log_2(n))$ , con $n$ el número de elementos del árbol.			
En un árbol 2-3 la altura siempre disminuye si al borrar un elemento se produce una combinación con los elementos de la raíz del árbol			
En la operación de borrado de un elemento en un árbol 2-3-4, si hay que realizar reestructuraciones, éstas se realizan desde las hojas hacia la raíz.			
Las rotaciones en un árbol Rojo – Negro requieren un cambio de color en los nodos implicados.			
Todo árbol binario de búsqueda es un árbol B con $m=3$ .			
En la dispersión cerrada puede haber colisiones entre claves sinónimas y no sinónimas.			
Un Heap Mínimo es un árbol binario que además es árbol mínimo			
En un árbol leftist se cumple que:			
$CMIN(x) = 1 + CMIN(HijoDer(x))$ para todo $x$ no vacío y $x$ con dos hijos			

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen TAD/PED septiembre 2004

### Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - **El test vale un 40% de la nota de teoría.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	F 1.
<input type="checkbox"/>	<input type="checkbox"/>	F 2.
<input type="checkbox"/>	<input type="checkbox"/>	F 3.
<input type="checkbox"/>	<input type="checkbox"/>	V 4.
<input type="checkbox"/>	<input type="checkbox"/>	F 5.
<input type="checkbox"/>	<input type="checkbox"/>	V 6.
<input type="checkbox"/>	<input type="checkbox"/>	V 7.
<input type="checkbox"/>	<input type="checkbox"/>	F 8.
<input type="checkbox"/>	<input type="checkbox"/>	F 9.
<input type="checkbox"/>	<input type="checkbox"/>	V 10.
<input type="checkbox"/>	<input type="checkbox"/>	F 11.
<input type="checkbox"/>	<input type="checkbox"/>	F 12.
<input type="checkbox"/>	<input type="checkbox"/>	F 13.
<input type="checkbox"/>	<input type="checkbox"/>	V 14.

Apellidos:

Nombre:

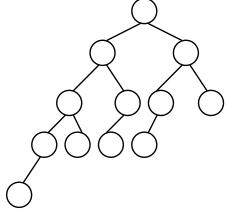
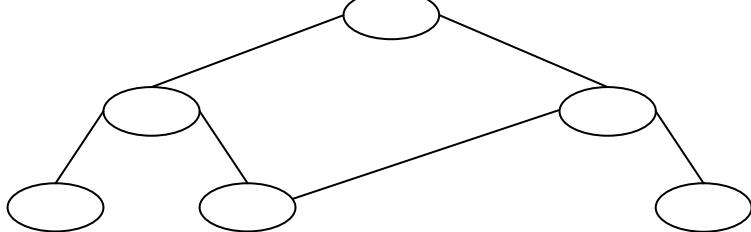
Convocatoria:

DNI:

## Examen TAD/PED septiembre 2005

### Modalidad 0

- Normas:
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **20 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - **El test vale un 40% de la nota de teoría.**
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

<p>El siguiente árbol está balanceado con respecto a la altura</p> 	<input type="checkbox"/> <input checked="" type="checkbox"/> 1. V
<p>La siguiente función de C++, <code>int&amp; Incremento(int valor){valor=valor+5;return valor;}</code>, devuelve el resultado por referencia.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 2. V
<p>En las colas, las inserciones y borrados se realizan por el mismo extremo.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 3. F
<p>La siguiente estructura es un árbol binario:</p> 	<input type="checkbox"/> <input checked="" type="checkbox"/> 4. F
<p>Un árbol completo es un árbol completamente equilibrado</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 5. F
<p>Los árboles AVL son árboles balanceados con respecto a la altura de los subárboles.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 6. V
<p>En un árbol 2-3, la diferencia en número de nodos entre los subárboles de la raíz es como mucho 1.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 7. F
<p>Un árbol rojo-negro, en el que no hay ningún enlace rojo, es un árbol binario completo.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 8. F
<p>Un árbol binario de búsqueda con altura 7 y 127 nodos es un árbol B con m=2</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 9. V
<p>En un árbol m-camino de búsqueda, todos los nodos excepto la raíz tienen al menos m/2 hijos.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 10. F
<p>En la dispersión cerrada se pueden producir colisiones entre claves no sinónimas</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 11. V
<p>En un montículo doble, un elemento "j" del montículo máximo es el simétrico de un único elemento "i" del montículo mínimo.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 12. F
<p>Un árbol Rojo-Negro cumple las propiedades de un árbol Leftist.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 13. F
<p>Al representar un grafo no dirigido con una matriz de adyacencia, su diagonal principal (casillas i,i) siempre tendrá valores Falso.</p>	<input type="checkbox"/> <input checked="" type="checkbox"/> 14. V

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen TAD/PED septiembre 2006

### Modalidad 0

- Normas:**
- La entrega del test no corre convocatoria.
  - Tiempo para efectuar el test: **18 minutos**.
  - Una pregunta mal contestada elimina una correcta.
  - Las soluciones al examen se dejarán en el campus virtual.
  - **Una vez empezado el examen no se puede salir del aula hasta finalizarlo. A continuación comenzará el siguiente ejercicio.**
  - El test vale un 40% de la nota de teoría.
  - En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F	
<input type="checkbox"/>	<input type="checkbox"/>	1. V
<input type="checkbox"/>	<input type="checkbox"/>	2. F
<input type="checkbox"/>	<input type="checkbox"/>	3. V
<input type="checkbox"/>	<input type="checkbox"/>	4. F
<input type="checkbox"/>	<input type="checkbox"/>	5. F
<input type="checkbox"/>	<input type="checkbox"/>	6. F
<input type="checkbox"/>	<input type="checkbox"/>	7.
<input type="checkbox"/>	<input type="checkbox"/>	8. V
<input type="checkbox"/>	<input type="checkbox"/>	9. V
<input type="checkbox"/>	<input type="checkbox"/>	10. F
<input type="checkbox"/>	<input type="checkbox"/>	11. V
<input type="checkbox"/>	<input type="checkbox"/>	12. V

Las operaciones constructoras generadoras de un tipo permiten obtener cualquier valor de dicho tipo.

En C++, si no se ha implementado la sobrecarga del operador asignación, se invoca automáticamente al constructor de copia.

Es posible reconstruir un único árbol binario de altura 6 a partir de un recorrido en postorden con 63 etiquetas.

La semántica de la operación nodos del tipo *arbin* vista en clase es la siguiente:

```
VAR i, d: arbin; x: item;
nodos( crea_arbin() ) = 0
nodos( enraizar( i, x, d ) ) = nodos( i ) + nodos( d )
```

Se puede reconstruir un único árbol binario cualquiera teniendo sus recorridos en preorden y postorden.

La semántica de la operación *recu* vista en clase es la siguiente:

```
VAR v: vector; i, j: int; x,: item;
recu( crear_vector( ), i ) = error_item( )
recu( asig(v, i, x ), j )
si ( i == j ) entonces x
sino FALSO
fsi
```

En un árbol AVL cuyo factor de equilibrio es 2, al insertar un elemento en la rama derecha, el árbol vuelve al estado de equilibrio.

Dado un árbol 2-3 de altura  $h$  con  $n$  ítems con todos sus nodos del tipo 2-Nodo: la complejidad de la operación de búsqueda de un ítem es  $O(\log_2 n)$

Un árbol binario de búsqueda lleno de altura 4 es un árbol 2-3-4, pero no se puede conseguir a partir de un árbol inicialmente vacío y utilizando las operaciones de inserción y borrado de un árbol 2-3-4.

Un grafo no dirigido puede tener aristas que empiecen y acaben en el mismo vértice.

El siguiente árbol es leftist mínimo:

```

graph TD
    1((1)) --- 7((7))
    1 --- 3((3))
    7 --- 4((4))
    3 --- 4
  
```

Un trie cumple las propiedades de un árbol general.

## Preguntas tipo test PED

F. Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes:

VAR x, y: natural

Suma(x,cero)=x

Suma(cero,x)=x

Suma(x,(suc(y))) = suma(suc(x),y)

F. En C++, la siguiente declaración es INCORRECTA: const int& a = 1;

F. Dada las clases TDir y TVectorDir con todos sus métodos implementados: constructor, constructor sobrecargado, sobrecarga del corchete, sobrecarga del operador salida (se muestra el contenido de cada posición del vector dejando un espacio), etc. Nota: El constructor por defecto crea un vector vacío. El constructor a partir de una dimensión pone todos los elementos a 0.

```
class TDir {  
public: ....  
private:  
int e1;  
int e2;  
};  
  
class TVectorDir {  
public: ....  
private:  
TDir *vector;  
int longitud;  
};  
  
main()  
{  
TDir a(1,1);  
TVectorDir v;  
cout<<"v_an-  
tes="<<v<<endl;  
v[1]=a;  
cout<<"v_des-  
pues="<<v<<endl;  
}
```

El resultado obtenido tras la ejecución del main() sería: v\_anteriores = 0 0 v\_despues = 1 1

V. La Complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución

F. La Complejidad temporal de un algoritmo depende de la complejidad espacial del mismo.

V. La semántica de la operación *desencolar* vista en clase es la siguiente:

VAR: c: cola, x: ítem;  
Si es vacía(c) entonces  
    Desencolar(encolar(c,x))=crear\_cola()  
Si no desencolar(encolar(c,x)) = encolar(desencolar(c),x)

V. Un árbol con único nodo es un árbol lleno

F. Un árbol con un único nodo tiene un único camino cuya longitud es 1.

V. Dados los recorridos de preorder, postorden y niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.

V. En la representación de conjuntos mediante listas, la complejidad espacial es proporcional al tamaño del conjunto representado.

F. En un grafo dirigido pueden existir infinitas aristas para un número "n" de vértices.

F. Para el siguiente fragmento de código C++ de un posible método perteneciente a la conocida clase TCoordenada, la línea "delete b"; liberaría correctamente la memoria dinámica en b.

```
void Funcion(void) {  
TCoordenada *a = new TCoordenada;  
TCoordenada *b = new TCoordenada[5];  
(. . . .)  
    delete b;  
}
```

F. El resultado del cálculo de la complejidad temporal en el mejor caso de un algoritmo X, da como resultado  $n + n^* \log(n)$ . Por lo tanto, diremos que la complejidad del algoritmo X cuando  $n \rightarrow \infty$  pertenece a  $\Omega(n)$ .

V. Las pilas también se conocen como listas LIFO.

V. Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol.

F. A los árboles generales también se les llama árboles multicamino de búsqueda.

V. Cuando se realiza una inserción en un AVL, en el camino de vuelta atrás para actualizar los factores de equilibrio, como mucho solo se va a efectuar una rotación.

F. La altura de un árbol 2-3 únicamente crece cuando se inserta un elemento y todos los nodos del árbol son 3-nodo.

F. Con las operaciones de inserción y borrado es posible conseguir un árbol 2-3-4 de altura 4 con todos sus nodos de tipo 2-nodo.

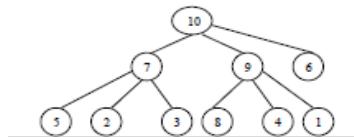
V. Las operaciones de transformación cuando se inserta un elemento en un árbol 2-3-4, en el caso de un árbol rojo-negro, se reducen a cambios de colores o rotaciones.

F. El árbol 2-3 es un árbol B m-camino de búsqueda con  $m=2$

V. La dispersión abierta elimina el problema del clustering secundario.

F. Sea una tabla de dispersión cerrada con estrategia de redispersión  $hi(x) = (H(x)+C*i) \bmod B$ , con  $B=1000$  y  $C=74$ . Para cualquier clave "x" se recorrerán todas las posiciones de la tabla buscando una posición libre cuando se inserta el elemento.

F. El siguiente árbol es un montículo máximo:



F. Para todo nodo de un árbol Leftist, se cumple que el número de nodos de su hijo izquierdo es mayor o igual que el de su hijo derecho.

V. Un grafo no dirigido de  $n$  vértices es un árbol si está libre de ciclos y tiene  $n-1$  aristas.

V. Dentro de la especificación algebraica de los números naturales definimos la sintaxis de la función F como  $F: \text{natural} \rightarrow \text{BOOL}$ , y su semántica como:  $F(\text{cero})=\text{TRUE}$ ,  $F(\text{suc}(\text{cero}))=\text{False}$ ,  $F(\text{suc}(\text{suc}(x))) = F(x)$ . Para el número natural  $x=35$ , la función F devolvería false.

F. En C++, la memoria que se reserva con new se libera automáticamente por el destructor.

F. La mejor complejidad temporal que se puede conseguir en un algoritmo es  $O(n)$ , con "n" como la talla del problema.

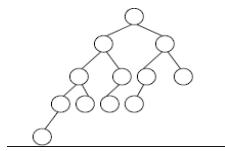
F. La semántica de la operación sublistा del tipo lista vista en clase es la siguiente:

```
VAR L: lista; x, y: item; n: natural; p: posicion;
sublista( L, p, 0 ) = crear()
sublista( crear(), p, n ) = crear()
si p == primera( inscabeza( L, x ) ) entonces
    sublista( inscabeza( L, x ), p, n ) = inscabeza( sublista( L, primera( L ), n ), x )
    si no sublista( inscabeza( L, x ), p, n ) = sublista( L, p, n )
```

V. Cuando realizamos un recorrido en inorder en un árbol binario de búsqueda las etiquetas aparecen ordenadas de menor a mayor.

F. Todo árbol binario de búsqueda es un árbol mínimo

V. El siguiente árbol está balanceado con respecto a la altura



F. Todo árbol binario de búsqueda es un árbol 2-3

F. En un árbol 2-3-4 sólo los nodos hoja y la raíz pueden ser de tipo 2-nodo.

F. En un árbol rojo-negro la búsqueda de una etiqueta dependerá de los colores de los hijos de cada nodo

V. El TAD Diccionario es un subtipo del TAD Conjunto

F. En un montículo doble de altura h se pueden almacenar un máximo de  $2^{h-2}$  claves

F. Un árbol leftist mínimo es un árbol binario mínimo tal que si no es vacío:  $\text{CMIN}(\text{HijoIzq}(x)) > \text{CMIN}(\text{HijoDer}(x))$  para todo x no vacío.

V. Para el tratamiento de errores en la especificación algebraica, se añaden funciones constantes que devuelven un valor del tipo que causa el error.

F. La complejidad temporal (en su caso peor) del siguiente fragmento de código es  $O(n^2)$

```
int i, j, n, sum;
for (i = 4; i < n; i++) {
    for (j = i-3, sum = a[i-4]; j <= i; j++) sum += a[j];
    cout << "La suma del subarray " << i-4 << " es " << sum << endl; }
```

V. Es posible obtener una representación enlazada de una cola utilizando un único puntero que apuntará al fondo de la cola

F. Dado un único recorrido de cualquier árbol, siempre es posible reconstruir dicho árbol

F. El coste temporal en su peor caso de insertar una etiqueta en un árbol binario de búsqueda es logarítmico respecto a la altura del árbol.

F. La complejidad temporal en el peor caso y en el mejor caso de la operación inserción en un AVL son lineales y logarítmica respecto al número de nodos en el árbol.

V. Dado un árbol 2-3 de altura h con n ítems:  $2^h - 1 \leq n \leq 3^h - 1$

V. Los nodos hoja de un árbol 2-3 han de estar en el mismo nivel del árbol

V. Para que decrezca la altura de un árbol 2-3-4 en una operación de borrado, el nodo raíz y sus hijos tienen que ser 2-nodos

V. Un árbol 2-3-4 es un árbol 4-camino de búsqueda

F. La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (Operación (Conjunto) -> Natural; Var: C: Conjunto; x: ítem):

*Operación(Crear)->1*  
*Operación(Insertar(C,x)) -> 3 + Operación(C)*

V. En un montículo el número de claves en el hijo izquierda de la raíz es mayor o igual que en su hijo derecho.

F. La siguiente secuencia de nodos de un grafo es un ciclo: 1,2,3,2,1

F. Un bosque extendido en profundidad de un grafo dirigido al que se le añaden los arcos de retroceso es un grafo acíclico dirigido.

- F.** Es posible reconstruir un único árbol binario de búsqueda de n elementos ( $n > 1$ ), a partir de su recorrido en inorden.
- F.** Sea un árbol binario lleno cuyo recorrido en inorden es: 10,15,17,20,21,28,35. La secuencia de árboles cuyas etiquetas son 35,28,20 es un camino en el mencionado árbol.
- V.** Un multigrafo es un grafo que no tiene ninguna restricción: pueden existir arcos flexivos y múltiples ocurrencias del mismo arco.
- V.** El número máximo de arcos que pueden existir en un grafo dirigido de n vértices son:  $n(n-1) + n$
- V.** En un grafo dirigido, un ciclo es un camino simple en el que el vértice primero y último coinciden.
- F.** En la especificación algebraica, para definir la semántica de una operación de un tipo de datos sólo se pueden utilizar las operaciones generadoras constructoras.
- V.** El TAD vector visto en clase se define como un conjunto ordenado de pares <índice, valor>. Para cada índice definido dentro de un rango finito existe asociado un valor.
- F.** En un árbol AVL cuyo nodo raíz tiene un factor de equilibrio +1 siempre que se inserte un nuevo elemento hay que realizar una rotación.
- F.** Todo árbol completo es un árbol completamente equilibrado.
- V.** En un árbol 2-3, la altura siempre disminuye si la raíz es de tipo 2-nodo y al efectuar el borrado de un elemento es necesario realizar una combinación con el nodo raíz.
- F.** La operación de borrar un elemento en un árbol 2-3-4 finaliza cuando el nodo p es el nodo que contiene al elemento que se desea borrar.
- V.** La complejidad en su caso peor, de la unión de dos conjuntos implementados como listas no ordenadas de tamaño "n" y "m" respectivamente es de  $O(n*m)$ .
- F.** Cuando implementamos un TAD Tabla de dispersión cerrada se usa una función de dispersión H tal que  $H(x)$  devolverá un valor comprendido entre 0 y B, siendo B el número finito de clases en las que dividimos el conjunto.
- V.** El montículo mínimo o HEAP mínimo es un árbol binario completo que además es árbol mínimo.
- F.** En la especificación algebraica, una operación es una función que toma como parámetros (entrada) uno o más valores de diversos tipos, y produce como resultado un solo valor de otro tipo.
- V.** Las ecuaciones (vistas en clase) que permiten realizar la multiplicación de números naturales son las siguientes:
- $$\begin{aligned} & \text{VAR } x, y: \text{natural} \\ & \text{Mult(cero, } x\text{)} = \text{cero} \\ & \text{Mult}(x, \text{cero}) = \text{cero} \\ & \text{Mult}(\text{suc}(y), x) = \text{suma}(\text{mult}(y, x), x) \end{aligned}$$
- V.** En la especificación algebraica, para el tratamiento de errores se añade una constante a la signatura que modeliza un valor de error, por ejemplo  $\text{error}_{\text{nat}} \rightarrow \text{natural}$
- V.** En C++, si se declara un objeto a (p. ej. TPoro a;) cuando la variable a se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.
- F.** Las ecuaciones (vistas en clase) para la operación recu de un vector son las siguientes:
- $$\begin{aligned} & \text{Recu}(\text{crear}(), i) = \text{error}() \\ & \text{Recu}(\text{asig}(v, i, x), j) \\ & \quad \text{Si } (i == j) \text{ entonces } j \\ & \quad \text{Si no } \text{recu}(v, j) \text{ fsi} \end{aligned}$$

V. La complejidad temporal de la operación *desapilar* (vista en clase) utilizando vectores (con un índice que indica la cima de la pila) o utilizando listas enlazadas es la misma.

F. La complejidad temporal del siguiente fragmento de código es  $O(n^2)$

```
Int i, j, n, sum;  
For ( i =4; i<n; i++)  
    For(j = i-3; sum = a[i-4]; j<=i; j++) sum += a[j];  
    Cout << "La suma del subarray" << i-4 << "es" << sum << endl;
```

V. En las colas circulares enlazadas vistas en clase, las operaciones *encolar* y *desencolar* tienen complejidad temporal  $O(1)$ .

V. Las ecuaciones (vistas en clase) para la operación *desencolar* son las siguientes:

```
Desencolar(crear()) = crear()  
Si es vacia(c) entonces  
    Desencolar(encolar(c,x)) = crear()  
Si no desencolar(encolar(c,x)) =  
    Encolar(desencolar(c),x)
```

V. Es posible reconstruir un único árbol binario de búsqueda a partir de un recorrido en preorden.

F. Un camino en un árbol es una secuencia  $a_1, \dots, a_s$  de árboles tal que para todo  $i \in \{1, \dots, s-1\}$ ,  $a_i$  es subárbol de  $a_{i+1}$ .

F. A los árboles generales también se les llama árboles multicamino de búsqueda.

F. La semántica de la operación *quita\_hojas* que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas es la siguiente:

```
VAR i, d: arbin; x: ítem;  
Quita_hojas(crear_arbin()) = crear_arbin()  
Quita_hojas(enraizar(crear_arbin(), x, crea_arbin())) =  
    Enraizar(crear_arbin(), x, crear_arbin())  
Quita_hojas(enraizar(i,x,d)) =  
    Enraizar(quita_hojas(i), x, quita_hojas(d))
```

V. Profundidad de un subárbol es la longitud del único camino desde la raíz a dicho subárbol.

F. En la inserción de un elemento en un árbol 2-3, la altura del árbol resultado siempre crece (con respecto al árbol original) cuando la raíz del árbol original es un 3-nodo.

V. En la inserción de un elemento en un árbol 2-3-4, la altura del árbol resultado siempre crece (con respecto al árbol original) cuando la raíz del árbol original es un 4-nodo.

V. En el algoritmo de borrado de un elemento en un árbol 2-3-4, siempre que el nodo “q” sea 2-nodo hay que hacer reestructuraciones.

V. La complejidad temporal de la operación *desapilar* (vista en clase) utilizando vectores (con un índice que indica la cima de la pila) o utilizando listas enlazadas es la misma.

F. La semántica de la operación *quita\_hojas* que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas es la siguiente:

```
VAR i, d: arbin; x: ítem;  
Quita_hojas(crear_arbin()) = crear_arbin()  
Quita_hojas(enraizar(crear_arbin(), x, crea_arbin())) = enraizar(crear_arbin(), x, crear_ar-  
bin())  
Quitar_hojas(enraizar(i,x,d)) = enraizar(quita_hojas(i), x, quita_hojas(d))
```

F. Todo árbol mínimo es un árbol binario de búsqueda.

F. El grado de los árboles AVL puede ser +1, 0 ó -1.

F. Todo árbol binario de búsqueda es un árbol 2-3

V. Es un árbol 2-3-4 el máximo número elementos del nivel N es  $3*2^{2N-2}$

V. La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C: conjunto; x: Item)

$$\begin{aligned}Operación(Crear) &\leftrightarrow 0 \\Operación(Insertar)(C,x) &\leftrightarrow 3 + Operación(C)\end{aligned}$$

V. En el TAD Diccionario con dispersión cerrada, con función de redispersión “ $hi(x) = (H(x) + k(x)*i) \text{ MOD } B$ ”, con  $B=6$  se puede dar la situación de que en una búsqueda no se acceda a todas las posiciones de la tabla.

V. En un Hash cerrado con factor de carga  $\alpha$ , se cumple que  $0 \leq \alpha \leq 1$ .

F. En un montículo doble, un elemento “j” del montículo máximo es el simétrico de un único elemento “i” del montículo mínimo.

V. Un multígrafo es un grafo que no tienen ninguna restricción: pueden existir arcos reflexivos y múltiples ocurrencias del mismo arco.

V. Sea  $G=(V,A)$  un grafo dirigido. Diremos que  $G''=(V'',A'')$  es un árbol extendido de  $G \rightarrow V''=V$ ,  $A'' \subset A$ , ( $\text{todos } v \in V'' \rightarrow \text{gradoE}(v) \leq 1$ )

V. La complejidad logarítmica aparece en algoritmos que descartan muchos valores (generalmente la mitad) en un único paso.

F. En C++, la parte privada de una clase sólo es accesible por los métodos de la propia clase.

V. El tipo posición en una lista con acceso por posición se puede instanciar a diferentes tipos de bojetos.

V. El recorrido en postorden es el inverso especular del recorrido en preorden para un árbol binario dado.

V. En un árbol AVL las inserciones siempre se realizan en las hojas.

F. Dado un árbol 2-3 de altura h con n ítems con todos su nodos del tipo 3-Nodo: la complejidad de la operación de búsqueda de un ítem es  $O(\log_3 h)$ .

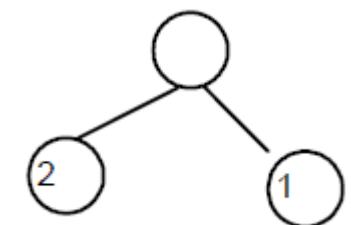
V. En un árbol 2-3-4 las reestructuraciones se realizan desde la raíz hacia las hojas.

F. En un árbol rojo-negro, el número de enlaces negros ha de ser mayor que el de enlaces rojos.

F. La altura del árbol B m-camino de búsqueda es “ $\log_m n$ ”, con “n=número total de claves”.

F. La función de redispersión en una tabla hash abierta, para que se recorran todas las posiciones del vector, tiene que cumplir que el valor de B sea primo.

F. El siguiente árbol es un montículo doble:



F. Todo árbol Leftist cumple las condiciones para ser un árbol binario de búsqueda.

V. Al representar un grafo dirigido de N vértices y K aristas con una lista de adyacencia, la operación de hallar la adyacencia de entrada de un vértice, tiene una complejidad de  $O(N^2)$ .

F. En C++, el constructor de copia sustituye al operador asignación.

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED junio 2016

### Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 2 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

V	F		
<input type="checkbox"/>	<input type="checkbox"/>	1	F
<input type="checkbox"/>	<input type="checkbox"/>	2	V
<input type="checkbox"/>	<input type="checkbox"/>	3	V
<input type="checkbox"/>	<input type="checkbox"/>	4	V
<input type="checkbox"/>	<input type="checkbox"/>	5	F
<input type="checkbox"/>	<input type="checkbox"/>	6	F
<input type="checkbox"/>	<input type="checkbox"/>	7	F
<input type="checkbox"/>	<input type="checkbox"/>	8	F
<input type="checkbox"/>	<input type="checkbox"/>	9	V
<input type="checkbox"/>	<input type="checkbox"/>	10	V
<input type="checkbox"/>	<input type="checkbox"/>	11	V
<input type="checkbox"/>	<input type="checkbox"/>	12	V
<input type="checkbox"/>	<input type="checkbox"/>	13	F
<input type="checkbox"/>	<input type="checkbox"/>	14	V
<input type="checkbox"/>	<input type="checkbox"/>	15	V

En la inserción de un elemento en un árbol 2-3, la altura del árbol resultado siempre crece (con respecto al árbol original) cuando la raíz del árbol original es un 3-nodo.

En la inserción de un elemento en un árbol 2-3-4, la altura del árbol resultado siempre crece (con respecto al árbol original) cuando la raíz del árbol original es un 4-nodo.

En el algoritmo de borrado de un elemento en un árbol 2-3-4, siempre que el nodo “q” sea 2-nodo hay que hacer reestructuraciones.

La complejidad temporal de la operación desapilar (vista en clase) utilizando vectores (con un índice que indica la cima de la pila) o utilizando listas enlazadas es la misma.

La semántica de la operación quita\_hojas que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas es la siguiente:

```
VAR i, d: arbin; x: item;
quita_hojas(crea_arbin()) = crea_arbin()
quita_hojas(enraizar(crea_arbin(), x, crea_arbin()) = enraizar(crea_arbin(), x, crea_arbin())
quita_hojas(enraizar(i, x, d)) = enraizar(quita_hojas(i), x, quita_hojas(d))
```

Todo árbol mínimo es un árbol binario de búsqueda

El grado de los árboles AVL puede ser +1, 0 ó -1.

Todo árbol binario de búsqueda es un árbol 2-3.

En un árbol 2-3-4 el máximo número elementos del nivel N es  $3*2^{2N-2}$

La especificación algebraica de la siguiente operación indica que se devolverá el número de elementos del conjunto multiplicado por 3 (C: Conjunto; x: Ítem):

$$\begin{aligned} \text{Operación(Crear)} &\Leftrightarrow 0 \\ \text{Operación(Insertar}(C, x)\text{)} &\Leftrightarrow 3 + \text{Operación}(C) \end{aligned}$$

En el TAD Diccionario con dispersión cerrada, con función de redispersión “ $hi(x)=(H(x) + k(x)*i) \text{ MOD } B$ ”, con  $B=6$  se puede dar la situación de que en una búsqueda no se acceda a todas las posiciones de la tabla.

En un Hash cerrado con factor de carga  $\alpha$ , se cumple que  $0 \leq \alpha \leq 1$

En un montículo doble, un elemento “j” del montículo máximo es el simétrico de un único elemento “i” del montículo mínimo.

Un multigrafo es un grafo que no tiene ninguna restricción: pueden existir arcos reflexivos y múltiples ocurrencias del mismo arco.

Sea  $G=(V,A)$  un grafo dirigido. Diremos que  $G''=(V'',A'')$  es un árbol extendido de  $G \Leftrightarrow V''=V, A'' \subset A, \forall v \in V'' \Rightarrow \text{gradoE}(v) \leq 1$

Apellidos:

Nombre:

Convocatoria:

DNI:

## Examen PED julio 2016

### Modalidad 0

Normas:

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- Este test vale 4 puntos (sobre 10).
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F							
La complejidad temporal (en su caso mejor) del siguiente fragmento de código es $\Omega(n)$ int i, length, n, i1, i2, k; for (i = 0, length = 1; i < n-1; i++) { for (i1 = i2 = k = i; k < n-1 && a[k] < a[k+1]; k++, i2++); if (length < i2 - i1 + 1) length = i2 - i1 + 1; }	<input type="checkbox"/>	<input type="checkbox"/>	1	V					
La complejidad temporal (en su peor caso) de la operación de insertar un elemento en una cola circular enlazada que no admite elementos repetidos es $O(n)$ , siendo $n$ el número de elementos de la cola.	<input type="checkbox"/>	<input type="checkbox"/>	2	V					
Un árbol con un único nodo es un árbol completo.	<input type="checkbox"/>	<input type="checkbox"/>	3	V					
El nivel de la raíz en un árbol binario es 0.	<input type="checkbox"/>	<input type="checkbox"/>	4	F					
Todo árbol binario mínimo es un árbol binario de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	5	F					
Un árbol binario de búsqueda completo es un AVL.	<input type="checkbox"/>	<input type="checkbox"/>	6	V					
El número de rotaciones que se nos pueden dar en el borrado de un elemento en un AVL son como máximo 3 menos que la altura del árbol.	<input type="checkbox"/>	<input type="checkbox"/>	7	F					
Dado un árbol 2-3 con $n$ ítems con todos sus nodos del tipo 2-Nodo. La complejidad de la operación de búsqueda de un ítem en el mencionado árbol es $O(\log_2 n)$ .	<input type="checkbox"/>	<input type="checkbox"/>	8	V					
En un árbol 2-3-4 los nodos pueden tener 1, 2, 3 ó 4 hijos.	<input type="checkbox"/>	<input type="checkbox"/>	9	F					
La mejor representación de los conjuntos siempre es el vector de bits porque es la más eficiente espacialmente.	<input type="checkbox"/>	<input type="checkbox"/>	10	F					
Sea una tabla de dispersión cerrada con estrategia de redispersión $h_i(x) = (H(x) + C*i) \bmod B$ , con $B=1000$ y $C=74$ . Para cualquier clave “x” que se desee insertar, se recorrerán todas las posiciones de la tabla buscando una posición libre.	<input type="checkbox"/>	<input type="checkbox"/>	11	F					
El siguiente vector representa un montículo máximo:	<input type="checkbox"/>	<input type="checkbox"/>	12	V					
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">10</td> <td style="padding: 5px;">5</td> <td style="padding: 5px;">3</td> <td style="padding: 5px;">1</td> <td style="padding: 5px;">2</td> </tr> </table>	10	5	3	1	2	<input type="checkbox"/>	<input type="checkbox"/>	13	V
10	5	3	1	2					
Sea $G=(V,A)$ un grafo dirigido. Diremos que $G''=(V'',A'')$ es un árbol extendido de $G \Leftrightarrow V''=V$ , $A'' \subset A$ , $\forall v \in V'' \Rightarrow \text{gradoE}(v) \leq 1$	<input type="checkbox"/>	<input type="checkbox"/>	14	F					
Un digrafo es un multigrafo que no contiene arcos reflexivos.	<input type="checkbox"/>	<input type="checkbox"/>	15	F					
La especificación algebraica de la operación <i>longitud</i> definida en clase para el tipo lista es la siguiente:	<input type="checkbox"/>	<input type="checkbox"/>	16	F					
VAR L1: lista; x: ítem; <i>longitud</i> (crear()) = 0 <i>longitud</i> (inscabeza(L1, x)) = 1 + inscabeza( <i>longitud</i> (L1), x)									
En la especificación algebraica de un tipo de datos las operaciones modificadoras devuelven un valor de un tipo diferente al que se está definiendo.									

Apellidos, Nombre:

DNI:

## Examen PED abril 2016

### Modalidad 0

**Normas:**

- Tiempo para efectuar el test: **20 minutos**.
- Una pregunta mal contestada elimina una correcta.
- Las soluciones al examen se dejarán en el campus virtual.
- **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**
- En la hoja de contestaciones el verdadero se corresponderá con la **A**, y el falso con la **B**.

	V	F	
En la especificación algebraica, una operación es una función que toma como parámetros (entrada) uno o más valores de diversos tipos, y produce como resultado un solo valor de otro tipo.	<input type="checkbox"/>	<input type="checkbox"/>	1 F
Las ecuaciones (vistas en clase) que permiten realizar la multiplicación de números naturales son las siguientes:  VAR x, y: natural; mult(cero, x) = cero mult(x, cero) = cero mult(suc(y), x) = suma(mult(y, x), x)	<input type="checkbox"/>	<input type="checkbox"/>	2 V
En la especificación algebraica, para el tratamiento de errores se añade una constante a la signatura que modeliza un valor de error, por ejemplo $\text{error}_{\text{nat}} \rightarrow \text{natural}$ .	<input type="checkbox"/>	<input type="checkbox"/>	3 V
En C++, si se declara un objeto <i>a</i> (p. ej. <i>TPoro a;</i> ) cuando la variable <i>a</i> se sale de ámbito entonces se invoca automáticamente al destructor de ese objeto.	<input type="checkbox"/>	<input type="checkbox"/>	4 V
Las ecuaciones (vistas en clase) para la operación <i>recu</i> de un vector son las siguientes:  <i>recu( crear( ), i ) = error( )</i> <i>recu( asig(v, i, x), j )</i> <i>    si ( i == j ) entonces j</i> <i>    si no recu( v, j ) fsi</i>	<input type="checkbox"/>	<input type="checkbox"/>	5 F
La complejidad temporal de la operación <i>desapilar</i> (vista en clase) utilizando vectores (con un índice que indica la cima de la pila) o utilizando listas enlazadas es la misma.	<input type="checkbox"/>	<input type="checkbox"/>	6 V
La complejidad temporal del siguiente fragmento de código es $O(n^2)$  int i, j, n, sum; for (i = 4; i < n; i++) { for (j = i-3, sum = a[i-4]; j <= i; j++) sum += a[j]; cout << "La suma del subarray " << i-4 << " es " << sum << endl; }	<input type="checkbox"/>	<input type="checkbox"/>	7 F
En las colas circulares enlazadas vistas en clase, las operaciones <i>encolar</i> y <i>desencolar</i> tienen complejidad temporal $\Theta(1)$ .	<input type="checkbox"/>	<input type="checkbox"/>	8 V
Las ecuaciones (vistas en clase) para la operación <i>desencolar</i> son las siguientes:  <i>desencolar( crear( ) ) = crear( )</i> <i>si esvacia( c ) entonces</i> <i>    desencolar( encolar( c, x ) ) = crear( )</i> <i>si no desencolar( encolar( c, x ) ) =</i> <i>    encolar( desencolar( c ), x )</i>	<input type="checkbox"/>	<input type="checkbox"/>	9 V
Es posible reconstruir un único árbol binario de búsqueda a partir de un recorrido en preorden.	<input type="checkbox"/>	<input type="checkbox"/>	10 V
Un camino en un árbol es una secuencia $a_1, \dots, a_s$ de árboles tal que para todo $i \in \{1, \dots, s-1\}$ , $a_i$ es subárbol de $a_{i+1}$ .	<input type="checkbox"/>	<input type="checkbox"/>	11 F
A los árboles generales también se les llama árboles multicamino de búsqueda.	<input type="checkbox"/>	<input type="checkbox"/>	12 F
La semántica de la operación <i>quita_hojas</i> que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas es la siguiente:  VAR i, d: arbin; x: item; <i>quita_hojas(crea_arbin( ) ) = crea_arbin( )</i> <i>quita_hojas(enraizar(crea_arbin(), x, crea_arbin()) =</i> <i>    enraizar(crea_arbin(), x, crea_arbin()</i> <i>    quita_hojas(enraizar(i, x, d)) =</i> <i>        enraizar(quita_hojas(i), x, quita_hojas(d))</i>	<input type="checkbox"/>	<input type="checkbox"/>	13 F
Profundidad de un subárbol es la longitud del único camino desde la raíz a dicho subárbol.	<input type="checkbox"/>	<input type="checkbox"/>	14 V

# **Examen PED diciembre 2007**

## **Modalidad 0**

**Normas:** • La entrega del test no corre convocatoria.

• Tiempo para efectuar el test: **15 minutos**.

• Una pregunta mal contestada elimina una correcta.

• Las soluciones al examen se dejarán en el campus virtual.

• **Una vez empezado el examen no se puede salir del aula hasta finalizarlo.**

• En la **hoja de contestaciones** el verdadero se corresponderá con la **A**, y el falso con la **B**.

<p>Las ecuaciones (vistas en clase) que permiten realizar la suma de números naturales son las siguientes:</p> <p>VAR x, y: natural;</p> <p>suma(x, cero) = x suma(cero, x) = x suma(x, suc(y)) = suma(suc(x), y)</p> <p>En C++, la siguiente declaración es INCORRECTA : const int&amp; a = 1;</p> <p>Dadas las clases TDir y TVectorDir con todos sus métodos implementados: constructor, constructor sobrecargado, sobrecarga del corchete, sobrecarga del operador salida (se muestra el contenido de cada posición del vector dejando un espacio), etc.</p> <p>Nota: El constructor por defecto crea un vector vacío.</p> <p>El constructor a partir de una dimensión, pone todos los elementos a 0.</p> <pre style="font-family: monospace;">class TDir           class TVectorDir        main() {     public: ....      public: ....          TDir a(1,1);     private: ....     private: ....         TVectorDir v;     int e1;            TDir *vector;        cout&lt;&lt;"v_anter= "&lt;&lt;v&lt;&lt;endl;     int e2;            int longitud;       v[1]=a; };  };  cout&lt;&lt;"v_despues= "&lt;&lt;v&lt;&lt;endl; }</pre> <p>El resultado obtenido tras la ejecución del <i>main()</i> sería:</p> <pre style="font-family: monospace;">v_anter= 0 0 v_despues= 1 1</pre> <p>La complejidad espacial es la cantidad de recursos espaciales que un algoritmo consume o necesita para su ejecución.</p> <p>La complejidad temporal de un algoritmo depende de la complejidad espacial del mismo.</p> <p>La semántica de la operación <i>desencolar</i> vista en clase es la siguiente:</p> <pre style="font-family: monospace;">VAR c: cola, x: item; si esvacia( c ) entonces     desencolar( encolar( c, x ) ) = crear_cola( )     si no desencolar( encolar( c, x ) ) = encolar( desencolar( c ), x )</pre> <p>Un árbol con un único nodo es un árbol lleno.</p> <p>Un árbol con un único nodo tiene un único camino cuya longitud es 1.</p> <p>Dados los recorridos de preorden, postorden y niveles de un árbol binario de altura 7 y 64 hojas es posible reconstruir un único árbol binario.</p> <p>En la representación de conjuntos mediante listas, la complejidad espacial es proporcional al tamaño del conjunto representado.</p> <p>En un grafo dirigido pueden existir infinitas aristas para un número "n" de vértices.</p>	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; width: 20px;">V</th> <th style="text-align: left; width: 20px;">F</th> <th style="text-align: left;"></th> </tr> </thead> <tbody> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">1    F</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">2    F</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">3    F</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">4    V</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">5    F</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">6    V</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">7    V</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">8    F</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">9    V</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">10    V</td> </tr> <tr> <td><input type="checkbox"/></td> <td><input type="checkbox"/></td> <td style="text-align: right;">11    F</td> </tr> </tbody> </table>	V	F		<input type="checkbox"/>	<input type="checkbox"/>	1    F	<input type="checkbox"/>	<input type="checkbox"/>	2    F	<input type="checkbox"/>	<input type="checkbox"/>	3    F	<input type="checkbox"/>	<input type="checkbox"/>	4    V	<input type="checkbox"/>	<input type="checkbox"/>	5    F	<input type="checkbox"/>	<input type="checkbox"/>	6    V	<input type="checkbox"/>	<input type="checkbox"/>	7    V	<input type="checkbox"/>	<input type="checkbox"/>	8    F	<input type="checkbox"/>	<input type="checkbox"/>	9    V	<input type="checkbox"/>	<input type="checkbox"/>	10    V	<input type="checkbox"/>	<input type="checkbox"/>	11    F
V	F																																				
<input type="checkbox"/>	<input type="checkbox"/>	1    F																																			
<input type="checkbox"/>	<input type="checkbox"/>	2    F																																			
<input type="checkbox"/>	<input type="checkbox"/>	3    F																																			
<input type="checkbox"/>	<input type="checkbox"/>	4    V																																			
<input type="checkbox"/>	<input type="checkbox"/>	5    F																																			
<input type="checkbox"/>	<input type="checkbox"/>	6    V																																			
<input type="checkbox"/>	<input type="checkbox"/>	7    V																																			
<input type="checkbox"/>	<input type="checkbox"/>	8    F																																			
<input type="checkbox"/>	<input type="checkbox"/>	9    V																																			
<input type="checkbox"/>	<input type="checkbox"/>	10    V																																			
<input type="checkbox"/>	<input type="checkbox"/>	11    F																																			