Examen PED marzo 2019
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 elección de la estructura de datos que soportará el TAD debe tomarse en la fase de especificación, no en la fase de implementación.	❑	❑	1	F
La sintaxis y la semántica de la operación simétricos que actúa sobre 2 árboles binarios y comprueba si son simétricos, se define del siguiente modo: simétricos(arbin, arbin) --> bool VAR i1, d1, i2, d2: arbin; x, y: item; simétricos(crea_arbin( ), crea_arbin( )) = VERDAD 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)) = (simétricos(i1, d2) & simétricos (d1, i2))	❑	❑	2	F

La complejidad temporal del siguiente fragmento de código es O(n*p)
int i, j, sum;
for (i = 1; i < n; i++);
for (j = 1, sum = a[i-4]; j <= p; j++) sum += a[j];
cout << “La suma del subarray “ << i-4 << “ es “ << sum << endl;
	❑	❑	3	F

En C++, el puntero *this* se tiene que declarar en todos los constructores de la clase	❑	❑	4	F
Dada la operación Examen definida como sigue: Examen(Lista, Lista) --> Lista VAR L1,L2:Lista; x,y:item Examen(crear_lista(), crear_lista()) = crear_lista() Examen(IC(L1, x), crear_lista()) = IC(L1, x) Examen(crear_lista(), IC(L1, x)) = crear_lista() Examen(IC(L1, x), IC(L2, y)) = IC(IC(Examen(L1, L2), y), x) Si se aplica Examen(L1, L2) con L1=(a, b, c, d) y L2=(m, n, o), en la que "a" es el primer elemento de L1, y "m" es el primer elemento de L2, se obtiene como resultado la Lista (a, m, b, n, c, o).	❑	❑	5	F
Los árboles multicamino de búsqueda son especialmente útiles cuando la memoria principal es insuficiente para almacenar todas las etiquetas.	❑	❑	6	V
La sintaxis y la semántica de la operación quita_hojas que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas se define del siguiente modo: quita_hojas(arbin) --> arbin VAR i, d: arbin; x: item; quita_hojas(crea_arbin( )) = crea_arbin( ) quita_hojas(enraizar(i, x, d)) = enraizar(quita_hojas(i), x, quita_hojas(d))	❑	❑	7	F
En un árbol cada elemento puede tener varios predecesores, pero como máximo un sucesor.	❑	❑	8	F
El recorrido en postorden de un árbol binario es el inverso especular del recorrido en preorden del mismo árbol.	❑	❑	9	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 EsVacía) tienen una complejidad temporal (en su peor caso) constante.	❑	❑	10	F
En las pilas, las inserciones y borrados se realizan por el mismo extremo.	❑	❑	11	V

El siguiente fragmento de código se cuentan como 2000 pasos en el cálculo de la complejidad temporal:
int ejemplo (int n) {
int i;
for (i=0; i < 2000; i++)
n+ = n;
return n; }
	❑	❑	12	F

La función de búsqueda BINARIA de un elemento en una lista ordenada, utilizando una representación enlazada, tiene una complejidad temporal (peor caso) de $O(\log_2 n)$.	❑	❑	13	F