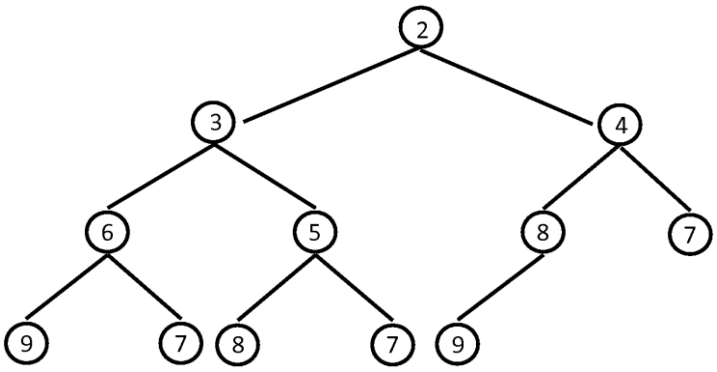
SEPTIEMBRE 2016 – ORIGINAL (MODELO A)

1. Dado el siguiente montículo:



Indique cuál de las siguientes afirmaciones es falsa:

(a) El montículo propuesto es un montículo de mínimos.

(b) El vector que lo representa de forma correcta es [2, 3, 4, 6, 5, 8, 7, 9, 7, 8, 7, 9].

(c) La operación "mínimo" en un montículo binomial tiene un coste O(1).

(d) El orden de complejidad de la operación de borrado de un elemento en el montículo es O(log n).

	Lista Enlazada	Montículo	Montículo Binomial	Mont. Fibonacci
insertar	O(1)	O(log n)	O(log n)	O(1)
mínimo/máximo	O(n)	O(1)	O(log n)	O(1)
borrar	O(n)	O(log n)	O(log n)	O(log n)

Respuesta C)

2. Indica cuál de las siguientes afirmaciones relativas a las tablas de dispersión es falsa:

(a) Una función hash debe repartir los valores h(x) en la tabla de manera equiprobable.

(b) Una función hash debe ser eficiente e indeterminista.

> **Falso**. Una función hash debe ser estable, rápida, determinista, con valor medio de complejidad constante (a lo sumo orden logarítmico).

(c) En la resolución de colisiones, el recorrido basado en una expresión cuadrática en lugar de lineal permite una mayor dispersión de las colisiones.

(d) Cambios pequeños en la clave deben resultar en cambios significativos en la función hash h(x).

Respuesta B)

3. El algoritmo Quicksort tiene:

(a) Caso medio de orden $O(n^2)$.

Falso. El coste es de orden completo, $O(n^2)$, o de O(n log n) mejorado tanto en el caso mejor como en el caso peor.

(b) Caso peor de orden $O(n^2 \log n)$.

Falso. No es cierto.

(c) Caso mejor de orden O(n).

Falso. O(n) es el coste de la función Pivotar.

(d) Caso mejor de orden O(n log n).

Cierto. Es su coste incluso en el caso peor.

Respuesta D)

4. Para resolver determinado problema hemos diseñado un algoritmo voraz. Indique cuál de las siguientes afirmaciones es falsa:

(a) Si encontramos un contraejemplo en el que el algoritmo no alcanza la solución óptima debemos probar con el siguiente valor distinto que nos proporcione la función de selección.

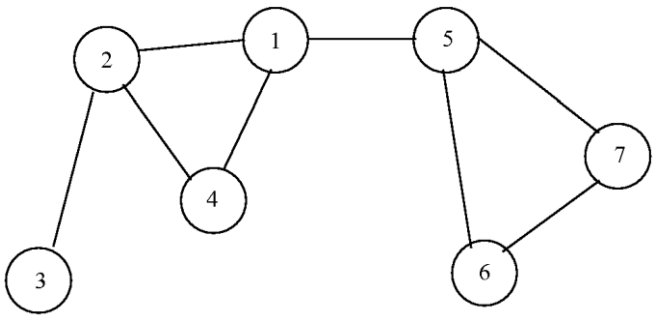
(b) Si encontramos un contraejemplo en el que el algoritmo no alcanza la solución óptima podemos afirmar que no es correcto.

(c) Necesitamos una demostración de optimalidad para poder asegurar que el algoritmo alcanza la solución óptima.

(d) La función de selección escoge al mejor de los candidatos restantes.

Respuesta A)

5. Dado el grafo de la figura:



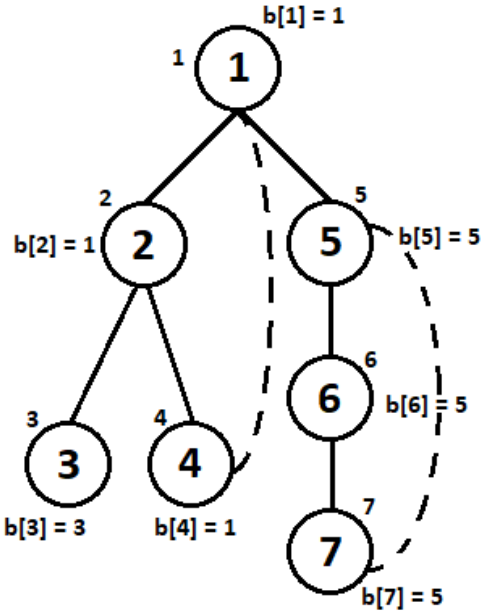
Los valores, al finalizar el algoritmo de cálculo de los puntos de articulación son:

(a) numOrden[]=[1, 2, 3, 4, 5, 6, 7] y bajo[]=[1, 1, 3, 1, 5, 5, 5]

(b) numOrden[]=[1, 2, 3, 4, 5, 6, 7] y bajo[]=[1, 2, 3, 4, 5, 6, 5]

(c) numOrden[]=[1, 3, 7, 3, 2, 5, 6] y bajo[]=[1, 2, 3, 4, 5, 6, 5]

(d) Ninguno de los anteriores



Nº orden = 1, 2, 3, 4, 5, 6, 7

Nodo	bajo[v]
1	1
2	1
3	3
4	1
5	5
6	5
7	5

Puntos de Articulación (PDA) = 1, 2 y 5

Respuesta A)

6. Dado el siguiente algoritmo:

```
// Precondición: i pertenece a {1,2,3}
hanoi(n,i,j) {
     si n=1 entonces escribe "Mover de "  i   "hasta"   j
     sino {
          hanoi(n-1, i    , 6-i-j)
          hanoi(1  , i    , j)
          hanoi(n-1, 6-i-j, j)
     }
}
```

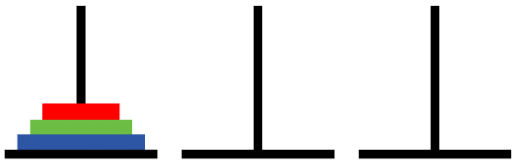
El coste asintótico temporal pertenece al orden:

(a) O(2n)

(b) $O(2^n)$

(c) $O(\log_2 n)$

(d) $O(n^2)$



Se realizan 3 llamadas recursivas, siendo éste un problema de reducción por substracción.

- n: Nº de discos
- a: Nº de llamadas recursivas
- k: Coste de las llamadas no recursivas

a = 2

k = Todas ellas tienen un coste 0, es decir, constante

b = 1

La instrucción “si n=1…” tiene un coste constante O(1).

$$T(n) = a \cdot T(n - b) + cn^k$$

$$T(n) \in O(2^n)$$

$$T(n) \in O(a^{n/b}) \text{ ya que } a > 1$$

Respuesta B)

PROBLEMA (4 Puntos)

Una caja con n bombones se considera "aburrida" si se repite un mismo tipo de bombón (por ejemplo, el bombón de "praliné") más de n/2 veces. Programar un algoritmo que decida si una caja es "aburrida" y devuelva (en su caso) el tipo de bombón que le confirme dicha propiedad.

La resolución del problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).

El esquema más apropiado es Divide y Vencerás. Se trata de resolver el problema del cálculo del elemento mayoritario en un vector.

El algoritmo parte de la observación de que, si un vector tiene un elemento mayoritario, entonces también ha de ser mayoritario en alguna de las mitades del vector, por lo que es posible dividir el vector en dos mitades y buscar el elemento mayoritario en cada una de ellas recursivamente, para, posteriormente, combinar las soluciones de los subproblemas.

El esquema general de Divide y Vencerás es el siguiente:

```
fun DyV(problema)
    si trivial(problema) entonces
        dev solución-trivial
    sino hacer
        {p_1, p_2, ...p_k} ← descomponer(problema)
        para i ∈ (1..k) hacer
            s_i ← DyV(p_i)
        fpara
    fsi
    dev combinar(s_1, s_2, ...s_k)
ffun
```

2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).

La estructura de datos es un array de n elementos en el que se debe almacenar, en cada una de sus posiciones, el tipo de bombón correspondiente.

3. Algoritmo <u>completo</u> a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz, debe realizarse la demostración de optimalidad. Si se trata del esquema de programación dinámica, deben proporcionarse las ecuaciones de recurrencia.

La solución completa se encuentra en el Libro de Texto base de la asignatura, en la página 115.

4. Estudio del coste <u>del algoritmo desarrollado</u> (0,5 puntos solo si el punto 1 es correcto).

El coste del algoritmo es del orden O(n log n).