# Respuestas para la modalidad **8**

1. ¿Cuál de estas afirmaciones es *falsa*?
   - (a) La memoización evita que un algoritmo recursivo ingenuo resuelva repetidamente el mismo problema.
   - [(b)] Los algoritmos iterativos de programación dinámica utilizan memoización para evitar resolver de nuevo los mismos subproblemas que se vuelven a presentar.
   - (c) La solución de programación dinámica iterativa al problema de la mochila discreta realiza cálculos innecesarios.
2. En el esquema de *vuelta atrás*, los mecanismos de poda basados en la mejor solución hasta el momento...
   - [(a)] ... pueden eliminar soluciones parciales que son factibles.
   - (b) Las otras dos opciones son ambas ciertas.
   - (c) ... garantizan que no se va a explorar nunca todo el espacio de soluciones posibles.
3. Si $\lim_{n\to\infty}(f(n)/n^2) = k$, y $k \neq 0$, ¿cuál de estas tres afirmaciones es *falsa*?
   - (a) $f(n) \in \Theta(n^2)$
   - [(b)] $f(n) \in \Theta(n^3)$
   - (c) $f(n) \in O(n^3)$
4. Los algoritmos voraces...
   - [(a)] ...se basan en la idea de que la solución óptima se puede construir añadiendo repetidamente el mejor elemento disponible.
   - (b) ...se basan en el hecho de que se puede ahorra esfuerzo guardando los resultados de cálculos anteriores en una tabla.
   - (c) ...se basan en el hecho de que una organización apropiada de los datos permite descartar la mitad de ellos en cada paso.
5. ¿Qué se entiende por *tamaño del problema*?
   - [(a)] La cantidad de espacio en memoria que se necesita para codificar una instancia de ese problema.
   - (b) El número de parámetros que componen el problema.
   - (c) El valor máximo que puede tomar una instancia cualquiera de ese problema.



6. Garantiza el uso de una estrategia "divide y vencerás" la existencia de una solución de complejidad temporal polinómica a cualquier problema?
   - [(a)] No.
   - (b) Sí, pero siempre que la complejidad temporal conjunta de las operaciones de descomposición del problema y la combinación de las soluciones sea polinómica.
   - (c) Sí, en cualquier caso.
7. De las siguientes expresiones, o bien dos son verdaderas y una es falsa, o bien dos son falsas y una es verdadera. Marca la que (en este sentido) es distinta a las otras dos.
   - [(a)] $\Theta(\log^2(n)) = \Theta(\log^3(n))$
   - (b) $\Theta(\log_2(n)) = \Theta(\log_3(n))$
   - (c) $\Theta(\log(n^2)) = \Theta(\log(n^3))$
8. Sea $A$ una matriz cuadrada $n \times n$. Se trata de buscar una permutación de las columnas tal que la suma de los elementos de la diagonal de la matriz resultante sea mínima. Indicad cuál de las siguientes afirmaciones es falsa.
   - [(a)] La complejidad temporal de la mejor solución posible al problema es $O(n \log n)$.
   - (b) Si se construye una solución al problema basada en el esquema de ramificación y poda, una buena elección de cotas optimistas y pesimistas podría evitar la exploración de todas las permutaciones posibles.
   - (c) La complejidad temporal de la mejor solución posible al problema está en $\Omega(n^2)$.
9. De las siguientes expresiones, o bien dos son verdaderas y una es falsa, o bien dos son falsas y una es verdadera. Marca la que (en este sentido) es distinta a las otras dos.
   - (a) $O(2^{\log_2(n)}) \subseteq O(n^2) \subset O(n!)$
   - (b) $(4^{\log_2(n)}) \subseteq O(n^2) \subset O(2^n)$
   - [(c)] $O(n^2) \subset O(2^{\log_2(n)}) \subset O(2^n)$
10. Cuando la descomposición recursiva de un problema da lugar a subproblemas de tamaño similar al original, ¿qué esquema promete ser más apropiado?
    - (a) Divide y vencerás, siempre que se garantice que los subproblemas no son del mismo tamaño.
    - (b) El método voraz.
    - [(c)] Programación dinámica.

11. Dada la relación de recurrencia:

$$T(n) = \begin{cases} 1 & \text{si } n \leq 1 \\ pT(\frac{n}{a}) + g(n) & \text{en otro caso} \end{cases}$$

(donde $p$ y $a$ son enteros mayores que 1 y $g(n) = n^k$), ¿qué tiene que ocurrir para que se cumpla $T(n) \in \Theta(n^k \log_a(n))$

(a) $p < a^k$
[(b)] $p = a^k$
(c) $p > a^k$

12. El uso de funciones de cota en ramificación y poda ...

(a) ...transforma en polinómicas complejidades que antes eran exponenciales.
(b) ...garantiza que el algoritmo va a ser más eficiente ante cualquier instancia del problema.
[(c)] ...puede reducir el número de instancias del problema que pertenecen al caso peor.

13. Si un problema de optimización lo es para una función que toma valores continuos ...

[(a)] ...la programación dinámica recursiva puede resultar mucho más eficiente que la programación dinámica iterativa en cuanto al uso de memoria.
(b) ...el uso de memoria de la programación dinámica iterativa y de la programación dinámica recursiva es el mismo independientemente de si el dominio es discreto o continuo.
(c) ...la programación dinámica iterativa siempre es mucho más eficiente que la programación dinámica recursiva en cuanto al uso de memoria.

14. En el esquema de *vuelta atrás* el orden en el que se van asignando los distintos valores a las componentes del vector que contendrá la solución...

(a) ... es irrelevante si no se utilizan mecanismos de poda basados en la mejor solución hasta el momento.
[(b)] Las otras dos opciones son ambas ciertas.
(c) ... puede ser relevante si se utilizan mecanismos de poda basados en estimaciones optimistas.



15. Dadas las siguientes funciones:

```
// Precondición: { 0 <= i < v.size(); i < j <= v.size() }
unsigned f( const vector<unsigned>&v, unsigned i, unsigned j ) {
  if( i == j+1 )
    return v[i];
  unsigned sum = 0;
  for( unsigned k = 0; k < j - i; k++ )
    sum += f( v, i, i+k+1) + f( v, i+k+1, j );
  return sum;
}

unsigned g( const vector<unsigned>&v ) {
  return f( v, v.begin(), v.end() );
}
```

Se quiere reducir la complejidad temporal de la función `g` usando programación dinámica iterativa. ¿cuál sería la complejidad espacial?

[(a)] Cuadrática.
(b) Se puede reducir hasta lineal.
(c) Cúbica.

16. Si $f \in \Theta(g_1)$ y $f \in \Theta(g_2)$ entonces

(a) $f \in \Theta(\text{máx}(g_1, g_2))$
[(b)] Las otras dos opciones son ambas ciertas.
(c) $f^2 \in \Theta(g_1 \cdot g_2)$

17. Los algoritmos de *vuelta atrás* que hacen uso de cotas optimistas generan las soluciones posibles al problema mediante ...

(a) ...un recorrido guiado por estimaciones de las que pueden ser las mejores ramas del árbol que representa el espacio de soluciones.
(b) ...un recorrido guiado por una cola de prioridad de donde se extraen primero los nodos que representan los subárboles más prometedores del espacio de soluciones.
[(c)] ...un recorrido en profundidad del árbol que representa el espacio de soluciones.

18. Sea la siguiente relación de recurrencia

$$T(n) = \begin{cases} 1 & \text{si } n \leq 1 \\ 2T(\frac{n}{2}) + g(n) & \text{en otro caso} \end{cases}$$

Si $T(n) \in O(n^2)$, ¿en cuál de estos tres casos nos podemos encontrar?

(a) $g(n) = n$
(b) $g(n) = n \log n$
[(c)] $g(n) = n^2$

19. En los algoritmos de *backtracking*, ¿Puede el valor de una cota pesimista ser mayor que el valor de una cota optimista? (se entiende que ambas cotas se aplican sobre el mismo nodo)

    (a) No, el valor de la cota pesimista de un nodo nunca puede ser superior al de la cota optimista de ese mismo nodo.

    [(b)] En general sí, si se trata de un problema de minimización, aunque en ocasiones ambos valores pueden coincidir.

    (c) En general sí, si se trata de un problema de maximización, aunque en ocasiones ambos valores pueden coincidir.

20. Tratándose de un esquema general para resolver problemas de maximización, ¿qué falta en el hueco?:

```
Solution BB( Problem p ) {
Node best, init = initialNode(p);
Value pb = init.pessimistic_b();
priority_queue<Node>q;
 q.push(init);
 while( ! q.empty() ) {
    Node n = q.top(); q.pop();
    q.pop();
    if( ????????? ){
      pb = max( pb, n.pessimistic_b());
      if( n.isTerminal() )
         best = n.sol();
      else
         for( Node n : n.expand() )
            if( n.isFeasible())
               q.push(n);
 }
return best;
}
```

    (a) `n.optimistic_b() <= pb`

    (b) `n.pessimistic_b() <= pb`

    [(c)] `n.optimistic_b() >= pb`

21. Cuál de los siguientes criterios proporcionaría una cota optimista para el problema de encontrar el camino mas corto entre dos ciudades (se supone que el grafo es conexo).

    (a) Calcular la distancia recorrida moviéndose al azar por el grafo hasta llegar (por azar) a la ciudad destino.

    (b) Utilizar la solución (subóptima) que se obtiene al resolver el problema mediante un algoritmo voraz.

    [(c)] Calcular la distancia geométrica (en línea recta) entre la ciudad origen y destino.



22. Cuando se resuelve el problema de la mochila discreta usando la estrategia de vuelta atrás, ¿puede ocurrir que se tarde menos en encontrar la solución óptima si se prueba primero a meter cada objeto antes de no meterlo?

    [(a)] Sí, pero sólo si se usan cotas optimistas para podar el árbol de búsqueda.

    (b) No, ya que en cualquier caso se deben explorar todas las soluciones factibles.

    (c) Sí, tanto si se usan cotas optimistas para podar el árbol de búsqueda como si no.

23. El siguiente programa resuelve el problema de cortar un tubo de longitud $n$ en segmentos de longitud entera entre 1 y $n$ de manera que se maximice el precio de acuerdo con una tabla que da el precio para cada longitud, pero falta un trozo. ¿Qué debería ir en lugar de **XXXXXXX**?

```
void fill(price m[]) {
for (index i=0;i<=n;i++) m[i]=-1;
}

price cutrod(length n, price m[], price p[]) {
price q;
if (m[n]>=0) return m[n];
if (n==0) q=0;
else {
       q=-1;
       for (index i=1;i<=n;i++)
          q=max(q,p[i]+cutrod(XXXXXXX));
}
m[n]=q;
return q;
}
```

    [(a)] `n-i,m,p`

    (b) `n-m[n],m,p`

    (c) `n,m[n]-1,p`

24. La siguiente relación de recurrencia expresa la complejidad de un algoritmo recursivo, donde $g(n)$ es una función polinómica:

$$T(n) = \begin{cases} 1 & \text{si } n \leq 1 \\ 2T(\frac{n}{2}) + g(n) & \text{en otro caso} \end{cases}$$

Di cuál de las siguientes afirmaciones es *falsa*:

(a) Si $g(n) \in \Theta(n)$ la relación de recurrencia representa la complejidad temporal del algoritmo de ordenación *mergesort*.

(b) Si $g(n) \in \Theta(n)$ la relación de recurrencia representa la complejidad temporal en el caso mejor del algoritmo de de ordenación *quicksort*.

[(c)] Si $g(n) \in \Theta(1)$ la relación de recurrencia representa la complejidad temporal del algoritmo de búsqueda dicotómica.

25. Se quiere ordenar $d$ números distintos comprendidos entre $1$ y $n$. Para ello se usa un array de $n$ booleanos que se inicializan primero a *false*. A continuación se recorren los $d$ números cambiando los valores del elemento del vector de booleanos correspondiente a su número a *true*. Por último se recorre el vector de booleanos escribiendo los índices de los elementos del vector de booleanos que son *true*. ¿Es este algoritmo más rápido (asintóticamente) que el *mergesort*?

[(a)] Sólo si $d \ \log \ d > k \, n$ (donde $k$ es una constante que depende de la implementación)

(b) No, ya que este algoritmo ha de recorrer varias veces el vector de booleanos.

(c) Sí, ya que el *mergesort* es $O(n \ \log \ n)$ y este es $O(n)$

26. Ante un problema de optimización resuelto mediante *backtracking*, ¿Puede ocurrir que el uso de las cotas pesimistas y optimistas sea inútil, incluso perjudicial?

[(a)] Sí, puesto que es posible que a pesar de utilizar dichas cotas no se descarte ningún nodo.

(b) No, las cotas tanto optimistas como pesimistas garantizan la reducción del espacio de soluciones y por tanto la eficiencia del algoritmo.

(c) Según el tipo de cota, las pesimistas puede que no descarten ningún nodo pero el uso de cotas optimistas garantiza la reducción el espacio de búsqueda.

27. Supongamos el algoritmo de ordenación *Mergesort* modificado de manera que, en lugar de dividir el vector en dos partes, se divide en tres. Posteriormente se combinan las soluciones parciales. ¿Cuál sería la complejidad temporal del nuevo algoritmo?

(a) $n^2 \log(n)$

(b) $n \log^2(n)$

[(c)] $n \log(n)$



28. La solución recursiva ingenua (pero correcta) a un problema de optimización llama más de una vez a la función con los mismos parámetros. Una de las siguientes tres afirmaciones es falsa.

[(a)] Se puede mejorar la eficiencia del algoritmo convirtiendo el algoritmo recursivo directamente en iterativo sin cambiar su funcionamiento básico.

(b) Se puede mejorar la eficiencia del algoritmo definiendo de antemano el orden en el que se deben calcular las soluciones a los subproblemas y llenando una tabla en ese orden.

(c) Se puede mejorar la eficiencia del algoritmo guardando en una tabla el valor devuelto para cada conjunto de parámetros de cada llamada cuando ésta se produce por primera vez.

29. ¿Para qué puede servir la cota pesimista de un nodo de *ramificación y poda*?

[(a)] Para actualizar el valor de la mejor solución hasta el momento.

(b) Para obtener una cota optimista más precisa.

(c) Para descartar el nodo si no es prometedor.

30. ¿Cuál de estas afirmaciones es *falsa*?

(a) Hay problemas de optimización para los cuales se puede obtener siempre la solución óptima utilizando una estrategia voraz.

[(b)] Todos los problemas de optimización tienen una solución voraz que es óptima sea cual sea la instancia a resolver.

(c) Hay problemas de optimización en los cuales el método voraz sólo obtiene la solución óptima para algunas instancias y un subóptimo para muchas otras instancias.

31. El algoritmo de ordenación *Mergesort* divide el problema en dos subproblemas. ¿Cuál es la complejidad temporal asintótica de realizar esa división?

[(a)] $O(1)$

(b) $O(n \ \log \ n)$

(c) $O(n)$

32. En un algoritmo de *ramificación y poda*, el orden escogido para priorizar los nodos en la lista de nodos vivos ...

(a) ...nunca afecta al tiempo necesario para encontrar la solución óptima.

[(b)] ...puede influir en el número de nodos que se descartan sin llegar a expandirlos.

(c) ...determina la complejidad temporal en el peor de los casos del algoritmo.

33. La complejidad temporal del la solución de *vuelta atrás* al problema de la mochila discreta es ...
    - [a] ... exponencial en el caso peor..
    - (b) ... exponencial en cualquier caso.
    - (c) ... cuadrática en el caso peor.

34. ¿Cuál de estos problemas tiene una solución eficiente utilizando *programación dinámica*?
    - (a) El problema de la asignación de tareas.
    - [b] El problema del cambio (devolver una cantidad de dinero con el mínimo número de monedas).
    - (c) La mochila discreta sin restricciones adicionales.

35. La función $\gamma$ de un número semientero positivo (un número es semientero si al restarle 0.5 es entero) se define como:
```
double gamma( double n ) { // Se asume n>=0.5 y n-0.5 entero
  if( n == 0.5 )
    return sqrt(PI);
  return n * gamma( n - 1);
}
```
    ¿Se puede calcular usando programación dinámica iterativa?
    - [a] Sí, pero la complejidad temporal no mejora.
    - (b) No, ya que no podríamos almacenar los resultados intermedios en el almacén.
    - (c) No, ya que el índice del almacén sería un número real y no entero.

36. Para uno de estos tres problemas no se conoce una solución trivial y eficiente que siga el esquema voraz.
    - (a) El problema de la mochila discreta sin limitación en la carga máxima de la mochila.
    - [b] El problema del cambio (devolver una cantidad de dinero con el mínimo número de monedas).
    - (c) El problema de la mochila continua.

37. En los algoritmos de *ramificación y poda* ...
    - [a] Una cota optimista es necesariamente un valor insuperable, de no ser así se podría podar el nodo que conduce a la solución óptima.
    - (b) Una cota pesimista es el valor que a lo sumo alcanza cualquier nodo factible que no es el óptimo.
    - (c) Una cota optimista es necesariamente un valor alcanzable, de no ser así no está garantizado que se encuentre la solución óptima.



38. ¿Cuál es la definición correcta de $\Omega(f)$?
    - [a] $\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | \exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, g(n) \geq cf(n)\}$
    - (b) $\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | \forall c \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, f(n) \geq cg(n)\}$
    - (c) $\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | \exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, f(n) \geq cg(n)\}$

39. Cuando se usa un algoritmo voraz para abordar la resolución de un problema de optimización por selección discreta (es decir, un problema para el cual la solución consiste en encontrar un subconjunto del conjunto de elementos que optimiza una determinada función), ¿cuál de estas tres cosas es imposible que ocurra?
    - [a] Que se reconsidere la decisión ya tomada anteriormente respecto a la selección de un elemento a la vista de la decisión que se debe tomar en un instante.
    - (b) Que la solución no sea la óptima.
    - (c) Que el algoritmo no encuentre ninguna solución.

40. ¿Para cuál de estos problemas de optimización se conoce al menos una solución voraz óptima?
    - [a] El árbol de recubrimiento mínimo para un grafo no dirigido con pesos.
    - (b) El problema de la asignación de coste mínimo de $n$ tareas a $n$ trabajadores cuando el coste de asignar la tarea $i$ al trabajador $j$, $c_{ij}$ está tabulado en una matriz.
    - (c) El problema de la mochila discreta.