Respuestas para la modalidad **4**

1. Se quieren 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) Sí, ya que el *mergesort* es $O(n \log n)$ y este es $O(n)$

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

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

2. En el esquema de vuelta atrás, los mecanismos de poda basados en la mejor solución hasta el momento...

   (a) ... garantizan que no se va a explorar nunca todo el espacio de soluciones posibles.

   (b) Las dos anteriores son verdaderas.

   [(c)] ... pueden eliminar soluciones parciales que son factibles.

3. Uno de estos tres problemas no tiene una solución trivial y eficiente que siga el esquema voraz.

   (a) El problema de la mochila continua.

   [(b)] El problema del cambio.

   (c) El problema de la mochila discreta sin limitación en la carga máxima de la mochila.

4. 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) ... puede ser relevante si se utilizan mecanismos de poda basados en estimaciones optimistas.

   [(b)] Las dos anteriores son ciertas.

   (c) ... es irrelevante si no se utilizan mecanismos de poda basados en la mejor solución hasta el momento.

5. 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) cúbica

   (b) exponencial

   [(c)] cuadrática

6. ¿Cuál es el coste temporal asintótico de la siguiente función?

```
void f(int n, int arr[]) {
int i = 0, j = 0;
   for(; i < n; ++i)
     while(j < n && arr[i] < arr[j])
       j++;
}
```

   (a) $O(n \log n)$

   (b) $O(n^2)$

   [(c)] $O(n)$

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

   [(a)] ... un recorrido en profundidad 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 guiado por estimaciones de las mejores ramas del árbol que representa el espacio de soluciones.

8. ¿Cuál de estos problemas tiene una solución eficiente utilizando *programación dinámica*?

   (a) La mochila discreta sin restricciones adicionales.

   [(b)] El problema del cambio.

   (c) El problema de la asignación de tareas.

9. Sea n el número de elementos que contienen los vectores w y v en la siguiente función f. ¿Cuál es su complejidad temporal asintótica en función de n asumiendo que en la llamada inicial el parámetro i toma valor n?

```
float f(vector <float>&w, vector<unsigned>&v,
unsigned P, int i){
float S1, S2;
 if (i>=0){
    if (w[i] <= P)
      S1= v[i] + f(w,v,P-w[i],i-1);
    else S1= 0;
      S2= f(w,v,P,i-1);
    return max(S1,S2);
 }
return 0;
}
```

[(a)] $\Omega(n)$ y $O(2^n)$
(b) $\Theta(2^n)$
(c) $\Omega(n)$ y $O(n^2)$

10. En un algoritmo de *ramificación y poda*, si la lista de nodos vivos no está ordenada de forma apropiada ...

(a) ...podría ocurrir que se pode el nodo que conduce a la solución óptima.
[(b)] ...podría ocurrir que se exploren nodos de forma innecesaria.
(c) ...podría ocurrir que se descarten nodos factibles.

11. En los algoritmos de *ramificación y poda* ...

(a) Una cota optimista es necesariamente un valor alcanzable, de no ser así no está garantizado que se encuentre 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 insuperable, de no ser así se podría podar el nodo que conduce a la solución óptima.

12. Se desea encontrar el camino mas corto entre dos ciudades. Para ello se dispone de una tabla con la distancia entre los pares de ciudades en los que hay carreteras o un valor centinela (por ejemplo, -1) si no hay, por lo que para ir de la ciudad inicial a la final es posible que haya que pasar por varias ciudades. Como también se conocen las coordenadas geográficas de cada ciudad se quiere usar la distancia geográfica (en línea recta) entre cada par de ciudades como cota para limitar la búsqueda en un algoritmo de vuelta atrás. ¿Qué tipo de cota sería?

    [(a)] Una cota optimista.

    (b) Una cota pesimista.

    (c) No se trataría de ninguna poda puesto que es posible que esa heurística no encuentre una solución factible.

13. 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 el algoritmo no encuentre ninguna solución.

    (b) Que la solución no sea la óptima.

    [(c)] 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.

14. ¿Cuál de estas estrategias para calcular el n-ésimo elemento de la serie de Fibonacci ($f(n) = f(n-1)+f(n-2), f(1) = f(2) = 1$) es más eficiente?

    (a) Para este problema, las dos estrategias citadas serían similares en cuanto a eficiencia

    [(b)] Programación dinámica

    (c) La estrategia voraz

15. 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 $g(n) \in O(1)$ la relación de recurrencia representa la complejidad temporal del algoritmo de búsqueda dicotómica.

    [(b)] Si $g(n) \in O(n^2)$ la relación de recurrencia representa la complejidad temporal del algoritmo de búsqueda por inserción.

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

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

[(a)] $g(n) = 1$

(b) $g(n) = n^2$

(c) $g(n) = n$

17. ¿Cuál es la definición correcta de $O(f)$?

(a) $O(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | \exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, f(n) \leq cg(n)\}$

(b) $O(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | \forall c \in \mathbb{R}, \forall n_0 \in \mathbb{N}, \forall n \geq n_0, f(n) \leq cg(n)\}$

[(c)] $O(f) = \{g : \mathbb{N} \to \mathbb{R}^+ | \exists c \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, g(n) \leq cf(n)\}$

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

(a) La programación dinámica iterativa siempre es mucho más eficiente que la programación dinámica recursiva 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 recursiva puede resultar mucho más eficiente que la programación dinámica iterativa en cuanto al uso de memoria.

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

[(a)] ...puede reducir el número de instancias del problema que pertenecen al caso peor.

(b) ...garantiza que el algoritmo va a ser más eficiente ante cualquier instancia del problema.

(c) ...transforma en polinómicas complejidades que antes eran exponenciales.

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

```
Solution BB( Problem p ) {
Node best, init = initialNode(p);
Value pb = init.pessimistic_b();
priority_queue<Node>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.pessimistic_b() <= pb`

(b) `n.optimistic_b() >= pb`

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

21. Estudiad la relación de recurrencia:

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

(donde p y q son enteros mayores que 1). Di cuál de los siguientes esquemas algorítmicos produce de manera natural relaciones de recurrencia así.

(a) Programación dinámica

[(b)] Divide y vencerás

(c) Ramificación y poda

22. En los algoritmos de *ramificación y poda*, ¿el valor de una cota pesimista es menor que el valor de una cota optimista? (se entiende que ambas cotas se aplican sobre el mismo nodo)

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

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

(c) Si, siempre es así.

23. 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, tanto si se usan cotas optimistas para podar el árbol de búsqueda como si no.

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

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

24. Garantiza el uso de una estrategia "divide y vencerás" la existencia de una solución de complejidad temporal polinómica a cualquier problema?

    (a) Sí, en cualquier caso.

    (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)] No

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

    [(a)] $O(n)$

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

    (c) $\Omega(n)$ y $O(n^2)$

26. ¿Se puede reducir el coste temporal de un algoritmo recursivo almacenando los resultados devueltos por las llamadas recursivas?

    (a) No, sólo se puede reducir el coste convirtiendo el algoritmo recursivo en iterativo

    [(b)] Sí, si se repiten llamadas a la función con los mismos argumentos

    (c) No, ello no reduce el coste temporal ya que las llamadas recursivas se deben realizar de cualquier manera

27. ¿Para cuál de estos problemas de optimización se conoce una solución voraz?

    (a) El problema de la mochila discreta.

    (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 árbol de recubrimiento mínimo para un grafo no dirigido con pesos.

28. Cuál de los siguientes criterios proveería 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 geométrica (en línea recta) entre la ciudad origen y destino.

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

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

29. Decid cuál de estas tres es la cota pesimista más ajustada al valor óptimo de la mochila discreta:

    [(a)] El valor de la mochila discreta que se obtiene usando un algoritmo voraz basado en el valor específico de los objetos.

    (b) El valor de una mochila que contiene todos los objetos restantes aunque se pase del peso máximo permitido.

    (c) El valor de la mochila continua correspondiente.

30. Cuando la descomposición recursiva de un problema da lugar a subproblemas de tamaño similar, ¿qué esquema promete ser más apropiado?

    [(a)] Programación dinámica.

    (b) El método voraz

    (c) Divide y vencerás, siempre que se garantice que los subproblemas no son del mismo tamaño.

31. 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 guardando en una tabla el valor devuelto para cada conjunto de parámetros de cada llamada cuando ésta se produce por primera vez.

    (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 convirtiendo el algoritmo recursivo directamente en iterativo sin cambiar su funcionamiento básico.

32. ¿Cuál es la diferencia principal entre una solución de vuelta atrás y una solución de ramificación y poda para el problema de la mochila?

    [(a)] El orden de exploración de las soluciones.

    (b) El hecho que la solución de ramificación y poda puede empezar con una solución subóptima voraz y la de vuelta atrás no.

    (c) El coste asintótico en el caso peor.

33. ¿Qué tienen en común el algoritmo que obtiene el k-ésimo elemento más pequeño de un vector (estudiado en clase) y el algoritmo de ordenación *Quicksort*?

    [(a)] La división del problema en subproblemas.
    (b) El número de llamadas recursivas que se hacen.
    (c) La combinación de las soluciones a los subproblemas.

34. Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ entonces ...

    (a) $\ldots g(n) \in O(f(n))$
    (b) $\ldots f(n) \in \Theta(g(n))$
    [(c)] $\ldots f(n) \in O(g(n))$

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

    (a) ...determina la complejidad temporal en el peor de los casos del algoritmo.
    [(b)] ...puede influir en el número de nodos que se descartan sin llegar a expandirlos.
    (c) ...nunca afecta al tiempo necesario para encontrar la solución óptima.

36. Si $f(n) \in O(n^2)$, ¿podemos decir siempre que $f(n) \in O(n^3)$?

    (a) Sólo para valores bajos de n
    (b) No, ya que $n^2 \notin O(n^3)$
    [(c)] Sí ya que $n^2 \in O(n^3)$

37. Sea $g(n) = \sum_{i=0}^{K} a_i n^i$. Di cuál de las siguientes afirmaciones es falsa:

    (a) $g(n) \in \Omega(n^K)$
    [(b)] Las otras dos afirmaciones son ambas falsas.
    (c) $g(n) \in \Theta(n^K)$

38. 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,m[n]-1,p`

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

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

39. 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 está en $\Omega(n^2)$.

(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 es $O(n \log n)$.

40. La versión de *Quicksort* que utiliza como pivote el elemento del vector que ocupa la posición central ...

(a) ... se comporta peor cuando el vector ya está ordenado.

[(b)] ... se comporta mejor cuando el vector ya está ordenado.

(c) ...no presenta caso mejor y peor para instancias del mismo tamaño.