

- Dado el problema del laberinto con tres movimientos se desea saber el número de caminos distintos desde la casilla inicial  $(1,1)$  hasta la casilla  $(n,m)$ , y para ello se aplica el esquema de programación dinámica para obtener un algoritmo lo más eficiente posible en cuanto a complejidad temporal y espacial. ¿Cuáles serían ambas complejidades?

Temporal  $\Theta(n \times m)$  y espacial  $\Theta(\min\{n, m\})$

Explicación: Usando programación dinámica empleamos almacenes para guardar las soluciones parciales, reduciendo la complejidad temporal a costa de la espacial. En este caso nos dicen que el algoritmo es el más eficiente posible por tanto la complejidad temporal será siempre como mínimo  $(n \times m)$  el tamaño del problema, ya que menor no puede ser y la complejidad espacial mejor será la de un vector de la dimensión más pequeña  $n \times m$  del problema que sera donde guardemos los resultados parciales y que se irá sobre escribiendo en cada iteración.

- Dado el problema del laberinto con tres movimientos, se desea saber el número de caminos distintos desde la casilla inicial  $(1,1)$  hasta la casilla final  $(n,m)$  y para ello se aplica un esquema de programación dinámica. En cuanto a la complejidad temporal ¿Cuál es la mejora de la versión recursiva con memoización frente a la recursiva ingenua que se obtiene apartir del esquema divide y vencerás?

De una complejidad exponencial que se obtendría con la ingenua se reduciría a polinómica con la memoización.

Explicación: Con la solución recursiva ingenua para cada casilla calculamos la versión recursiva de sus adyacentes es decir para la casilla  $(1,1)$  se llamaría la llamada recursiva para  $(1,2)$ ,  $(2,1)$  y  $(2,2)$  ó  $(n-1,m)$ ,  $(n,m-1)$ ,  $(n-1,m-1)$  y cada una de estas haría a su vez lo mismo hasta llegar al final o a un callejón sin salida donde empezaría a devolver las soluciones de las llamadas desde el final.

Como cada llamada lanza a su vez varias subllamadas la complejidad crece exponencialmente, pero usando memoización podemos almacenar las llamadas ya resueltas desde el final. Por tanto si una llamada recursiva ya ha resuelto ese cálculo solo tenemos que consultar el resultado, ahorrando un gran número de llamadas, ya que aunque se lancen varias llamadas para calcular cada casilla, el cálculo solo se va a realizar una vez, por tanto es la complejidad del cálculo más la de las consultas, ya no se realizan varias llamadas a si misma y la complejidad pasa de ser exponencial a polinómica que es mejor.

Las otras dos no pueden ser ya que 1º la complejidad de la ingenua nunca sera cuadrática ya que como hemos dicho cada llamada lanza múltiples llamadas recursivas y la 2º la versión con memoización nunca será peor que la recursiva ingenua, como mucho sería igual para casos excepcionales con llamadas y tamaños muy limitados (laborinto de 2x2)

- Dado el problema del laberinto con tres movimientos, se pretende conocer la longitud del camino de salida mas corto. Para ello se aplica el esquema voraz con un criterio de selección que consiste en elegir primero el movimiento este siempre que sea accesible. Si no lo es se descarta y se prueba con surEste, y por último, si este tampoco es posible, se escoge Sur. ¿Qué se puede decir del algoritmo obtenido?

Que en realidad no es un algoritmo voraz puesto que el criterio de selección no lo es.

Explicación: En un algoritmo Voraz nos basamos en un criterio de selección para elegir la opción más prometedora (elección que no se reconsidera) de modo que en este caso no es un algoritmo voraz ya que no se evalúa cual de las posibles soluciones es más prometedora, simplemente se elige la primera disponible según un orden establecido.

- Dado un problema del laberinto con tres movimientos ¿Cuál de las estrategias siguientes proveería de una cota optimista para ramificación y poda?

Las otras dos estrategias son ambas válidas.

Explicación: Suponer que en adelante todas las casillas son accesibles significa que el resultado sera el camino más corto a la salida de todos los posibles. Mientras que suponer que no se van a realizar más movimientos es igual a decir que ya has llegado al final. Como una cota optimista tiene que ser insuperable (no puede ser mejor) pero no tiene porque ser factible ambas son aún válidas.

- Dado el problema del laberinto con tres movimientos ¿Se puede aplicar un esquema de programación dinámica para obtener un camino de salida?

Sí, en caso de existir con este esquema siempre se puede encontrar un camino de salida.

Explicación: Va a probar todas las posibilidades, solo ahorraremos repetir calabozos.

Si existe un camino va a encontrarlo y basta con almacenar o devolver las coordenadas una vez que hemos llegado al final con el recorrido por ejemplo.

- Dado el problema del laberinto con tres movimientos, se desea saber el número de caminos distintos desde la casilla inicial  $(l, l)$  hasta la casilla  $(h, m)$  y para ello se aplica un esquema divide y vencerás ¿Cuál sería la recursividad apropiada para el caso general?

$$nc(h, m) = nc(h-1, m) + nc(h, m-1) + nc(h-1, m-1)$$

Explicación: el número de caminos es el total del número de caminos que hay siguiendo el camino de abajo, más el número de caminos de la diagonal más el número de caminos del camino de la izquierda. Donde cada uno es un subproblema que se resuelve con una llamada recursiva.

- Se desea obtener todas las permutaciones para una lista de n elementos  
¿Qué enfoque es el más adecuado?

Vuelta atrás, es el más eficiente para este problema.

Explicación: Vuelta atrás funciona recorriendo en orden todo el árbol de decisiones, recorre todas las hojas de una rama volviendo atrás una vez que hemos recorrido una rama para pasar a la siguiente.

No solo nos permite sacar todas las permutaciones posibles de la lista sino que además podemos devolverlas ordenadas por ello es el más adecuado.

Divide y vencerás sacaría todas pero sin orden ni coherencia, conforme fuera solucionando y ramificación y cada no tiene sentido ya que quitaría soluciones y nos interesa obtenerlas todas.

- ¿De dónde se deduce de  $f(n)$  y  $g(n)$  si se cumple  $\lim_{n \rightarrow \infty} (f(n)/g(n)) = K$ , con  $K \neq 0$ ?

$$f(n) \in O(g(n)) \text{ y } g(n) \in O(f(n))$$

Explicación: Si  $f(n)$  está acotada superiormente (o pertenece al orden de complejidad de la cota superior) de  $g(n)$  y a su vez  $g(n)$  está acotada superiormente por  $f(n)$  significa que pertenecen al mismo orden de complejidad.

Detaileda: Diapositiva 34 Tema 2: Según la notación asintótica

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = K, K \neq 0, K \neq \infty \Rightarrow \Theta(f) = \Theta(g)$$

Si cuando el tamaño del problema crece para  $\frac{f(n)}{g(n)}$  da un número K entre 0 e infinito (un real) significa que el coste exacto de f es igual que el de g. Si el coste exacto es  $\Theta(f) = O(f) \wedge \Omega(f)$  ó por simplificar esta entre la cota superior e inferior y sabiendo las equivalencias:

$$O(f) = O(g) \Rightarrow f \in O(g) \wedge g \in O(f)$$

$$\Omega(f) = \Omega(g) \Rightarrow f \in \Omega(g) \wedge g \in \Omega(f)$$

Es decir si f y g tienen la misma cota entonces f está dentro de la cota de g mientras que g está dentro de la de f.

Entonces si el coste exacto está entre ambas cotas:  $\Theta(f) = \Theta(g) \Rightarrow$

$$f \in O(g) \wedge g \in O(f) \wedge f \in \Omega(g) \wedge g \in \Omega(f)$$

Es decir si  $\frac{f(n)}{g(n)}$  sigue dando un número real conforme va creciendo el problema significa que  $f$  y  $g$  tienen el mismo corte exacto y están entre las mismas cotas, es decir se cumple:

$$\Theta(f) = \Theta(g) \rightarrow y \text{ por tanto estas dos tambien:}$$

$$f(n) \in O(g(n)) \wedge g(n) \in O(f(n))$$

$$f(n) \in \Omega(g(n)) \wedge g(n) \in \Omega(f(n))$$

- En el esquema de retro atras, los mecanismos de poda basados en la mejor solución hasta el momento...

...pueden eliminar vectores que representan posibles soluciones factibles.

Explicación: No solo predicho que deben, ya que esto ocurre cuando estima que las posibles soluciones no van a mejorar la mejor que ya tenemos hasta el momento que es el funcionamiento básico de la poda para ahorrar calcular soluciones no prometedoras.

- Si el corte temporal de un algoritmo es  $T(n)$ , ¿Cuál de las siguientes situaciones es imposible?

$$T(n) \in \Theta(n) \text{ y } T(n) \in \Omega(n^2)$$

Explicación: No puede pasar que tenga un corte exacto de  $n$  y una cota inferior de  $n^2$ , lo que estan diciendo es que el problema tiene un corte exacto de  $n$ , es decir que se ejecuta en un tiempo  $n$  pero como mínimo necesita un tiempo  $n^2$  para ejecutarse (cota inferior), eso es imposible.

$T(n) \in O(n)$  y  $T(n) \in \Theta(n)$  si puede ser porque si tarda  $n$  si se cumple que como mucho tardara  $n$  en ejecutarse (cota superior) y  $T(n) \in \Omega(n)$  y  $T(n) \in \Theta(n^2)$  tambien puede porque si el corte promedio es de  $n^2$  si esta dentro de las posibilidades que como mínimo tarde  $n$ .

- Sea la siguiente relación de recurrencia

$$T(n) \begin{cases} L & \text{Si } n \leq L \\ 2T(n/2) + g(n) & \text{en otro caso} \end{cases}$$

Si  $T(n) \in O(n)$  ¿en cual de los tres casos nos podemos encontrar?

Las otras dos opciones ambas son ciertas ( $g(n) = \log n$  y  $g(n) = \sqrt{n}$ )

Explicación: ya que se cumple la condición ( $b = h = a$ ) podemos aplicar el teorema de reducción

$$T(n) = aT\left(\frac{n}{a}\right) + g(n) \quad g(n) \in \Theta(n^K)$$

$$\Theta(n^K) \quad \text{Si } K > 1$$

$$\Theta(n \log n) \quad \text{Si } K = 1$$

$$\Theta(n) \quad \text{Si } K < 1 \quad \text{el enunciado nos dice ya a qué orden pertenece.}$$

Por tanto  $g(n) \in \Theta(n^K)$  para esta recurrencia cuando  $K < 1$

Entonces se cumple que sea  $g(n) = \sqrt{n}$  ya que  $\sqrt{x} = x^{\frac{1}{2}}$  y por tanto sera menor que  $n$  y estara dentro del  $\Theta(n)$  y lo mismo pasa con  $g(n) = \log n$ .

- ¿Qué estrategia de búsqueda es a priori más apropiada en un esquema de vuelta atrás?

En el esquema de vuelta atrás no se pueden definir estrategias de búsqueda.

Explicación: No tiene sentido definir una estrategia de búsqueda ya que vuelta atrás recorre todo el espacio del árbol de soluciones, no es un algoritmo que discrimine soluciones buscando la correcta.

- En ausencia de costas optimistas y pesimistas, la estrategia de vuelta atrás...

... No recorre todo el árbol si hay manera de descartar subárboles que representan conjuntos de soluciones no factibles.

Explicación: Si estimamos ó tenemos un método para comprobar que una solución parcial dada es una solución al problema no tenemos porque expandir e introducir sus hijos en la pila de nodos vivos ahorrandonos así explorar sus subárboles.

- Un algoritmo recursivo basado en el esquema divide y vencerás...

... alcanza su máxima eficiencia cuando el problema de tamaño  $n$  se divide en a problemas de tamaño  $n/a$ .

Explicación: Literalmente cuando se cumple el teorema de reducción que dice: los mejores resultados en cuanto corte se producen cuando en las llamadas recursivas el tamaño de los subproblemas son del mismo tamaño.

- Se desea ordenar una lista enlazada de  $n$  elementos haciendo uso del algoritmo mergesort. En este caso, al tratarse de una lista, la complejidad asintótica de realizar la división en subproblemas resulta ser lineal con el tamaño del problema esa lista. ¿Cuál sería el corte temporal de realizar esa ordenación?

$$\Theta(n \log n)$$

Explicación: Como en todos los casos la complejidad de la división en subproblemas es lineal con el tamaño, no se puede hacer la división sin recorrer el problema entero por tanto la complejidad es la misma que la del mergesort.  $\Theta(n \log n)$

- Dado un problema de minimización resuelto mediante un esquema de ramificación y poda, ¿Qué propiedad cumple la cota optimista?

Las otras dos opciones son ambas falsas:

- Siempre es mayor o igual que la mejor solución posible alcanzada
- Asegura un ahorro en la comprobación de todas las soluciones factibles

Explicación: La cota optimista es mayor o igual que la mejor solución posible alcanzada para los problemas de maximización, este es de minimización, debería ser menor o igual.

Tampoco asegura un ahorro en la comprobación depende de como este definida, si esta mal definida podría no solo no hacer que el algoritmo tarde menos sino aumentarlo incluso.

- 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:

$$\Theta(n) \subset \Theta(n^2)$$

Explicación: Esta es falsa ya que el coste exacto  $n$  no puede pertenecer al coste exacto  $n^2$ , el coste exacto es ero, exacto, no puedes decir que  $n$  vale  $n^2$ .

Las otras dos son ciertas ya que,  $n + n \log n \in \Omega(n)$ ,  $n + n \log n$  si que esta acotado inferiormente por  $n$ , nunca va a valer menos que  $n$ .

y  $O(2^{\log n}) \subset O(n^2)$  tambien es cierta ya que  $2^{\log_2 n} = n^{\log_2 2} = n$  por tanto  $O(n) \subset O(n^2)$

- ¿Qué complejidad se obtiene apartir de la relación de recurrencia  $T(h) = 8T(h/2) + h^3$  con  $T(1) = O(1)$ ?

$$O(n^3 \log n)$$

Explicación: Calculo general de recurrencia para  $T(h) = h(T)\left(\frac{h}{b}\right) + g(h) \Rightarrow$  Suponiendo un entero tal que  $g(h) \in \Theta(n^k) \Rightarrow g(h) \in \Theta(h^k)$   $\begin{cases} h=8 \\ b=2 \end{cases}$   $\Rightarrow k=3$

$$T(h) \in \begin{cases} \Theta(h^k) & \text{Si } h < b^k \\ \Theta(h^k \log_b h) & \text{Si } h = b^k \leftarrow 8 = 2^3 \text{ por tanto } O(n^3 \log n) \\ \Theta(n^{\log_b h}) & \text{Si } h > b^k \end{cases}$$

- Cuando la descomposición de un problema da lugar a subproblemas de tamaño similar al original, muchos de los cuales se repiten ¿que esquema es a priori más apropiado?

### Programación dinámica.

Explicación: Porque al haber subproblemas repetidos si los almacenamos nos ahorraremos tener que volver a calcularlos además cuanto mas similar sea el tamaño de los subproblemas mayor sera el ahorro de tiempo por tener la solución parcial almacenada.

- ¿En ramaificación y poda, tiene sentido utilizar la cota optimista de los nodos como criterio para ordenar la lista de nodos vivos?

Sí, aunque no es una garantía de que sea una buena estrategia de búsqueda.

Explicación: las cotas optimistas no tienen que ser factibles, ordenando el vector de nodos vivos por cota podría darse el caso de que muchos tuvieran una buena cota optimista pero no fueran factibles y aún así se explorarían primero. Pero si sigue son factibles serían los más prometedores los que se explorarían primero.

- En el problema del viajante de comercio queremos listar todas las soluciones ~~factibles~~ factibles.

El orden en el que se exploran las soluciones parciales no es relevante; por ello, la técnica de ramaificación y poda no aporta nada con respecto vuelta atrás.

Explicación: Vuelta atrás va a explorar todo el espacio de soluciones del árbol de decisiones, ramaificación y poda en este caso no nos aporta ninguna mejora ya que como tenemos que listar todas las soluciones no tiene sentido aplicar cotas y no nos importa que se exploren primero las soluciones parciales más prometedoras puesto que se requiere que se exploren todas.

- El esquema voraz...

Las otras dos opciones son falsas:

- Garantiza encontrar una solución a cualquier problema, aunque puede que no sea óptima.
- Puede que no encuentre solución pero si lo hace garantiza que sea la óptima.

Explicación: Los algoritmos voraces no garantizan que encuentren una solución, ni en caso de encontrarla sea la óptima. Lo que garantiza es que en caso de encontrarla es una solución válida (factible).

Como el algoritmo voraz avanza hasta la solución evaluando cada decisión y cogiendo la opción más prometedora sin reconsiderarla. Luego es posible que avance por un camino que sea prometedor pero que luego no tenga solución ó que la que antes parecía la más prometedora luego no lo sea.

- Decid cuál de estas tres es la cota optimista más ajustada al valor óptimo de la mochila discreta.

El valor de la mochila continua correspondeiente.

Explicación: Para la mochila discreta, es decir sin fraccionamiento, la mejor cota que podemos usar es la de la mochila con fraccionamiento (continua) porque es la que mejor se ajusta debido a que nos proporciona exactamente el mejor resultado posible para el problema. Es decir nos da el resultado de llenar la mochila hasta los topes con los objetos con mayor relación peso/valor. Este valor es insuperable (como mucho alcanzable) por la mochila discreta ya que podrían no caber todos los objetos. Por eso nos da una cota optimista perfecta para el problema, ya que es un como mucho puedo tener esto, es insuperable pero si alcanzable.

- En el esquema de ramificación y poda ¿que estructura es la más adecuada si queremos realizar una exploración por niveles?  
cola.

Explicación: Las estructuras más adecuadas según el tipo de recorrido que queramos obtener son:

Anchura (estrategia FIFO) => cola

Profundidad (estrategia LIFO) => pila

Direccional (estrategia mínimo corte LC) => montículo (HEAP)

El recorrido por niveles es un tipo de recorrido en anchura por eso la estructura más óptima es una cola, ya que es una FIFO (primero en entrar, primero en salir) e irá devolviendo la raíz del arbol y de los subárboles de este consecutivamente hasta llegar a las hojas recorriendo así todos los niveles.

- De las siguientes afirmaciones marca la que es verdadera

En un esquema de vuelta atrás las cotas pesimistas no tienen sentido si lo que se pretende es obtener todas las soluciones factibles.

Explicación: Porque sino no se explorarían las soluciones que fueran peor que la que tenemos hasta el momento (cota pesimista) y no las obtendríamos todas. Las otras dos afirmaciones "Las cotas pesimistas no son compatibles con un esquema de vuelta atrás", es falso ya que como ya se ha explicado las cotas pesimistas se utilizan para no explorar soluciones que son peor que la que ya tenemos hasta el momento. y "El esquema de vuelta atrás no es compatible con el uso conjunto de cotas pesimistas y optimistas" es falso, se usan ambas a ser posible para acotar lo más posible el espacio de búsqueda de soluciones para tener que explorar las menos posibles, se puede emplear solo una de ellas también pero no hay problema alguno por usar ambas al tiempo.

- Un tubo de  $n$ -centímetros de largo se puede cortar en segmentos de 1 cm, 2 cm, etc... Existe una lista de los precios a los que se venden los segmentos de cada longitud. Una de las maneras de cortar el tubo es la que más ingresos producirá. Se quiere resolver el problema mediante vuelta atrás. ¿Cuál sería la forma más adecuada de representar las posibles soluciones?

### Un Vector de Booleanos.

Explicación: Si planteamos el problema como un conjunto de decisiones para cada cm del tubo si va o no a realizarse corte, el problema es muy fácil resolvélo mediante vuelta atrás ya que es un árbol binario de decisiones y es mucho más fácil devolver la solución óptima tal cual. Además nos interesa solo la manera de cortar el tubo que maximiza los beneficios no nos interesa conocer el valor acumulado por cada corte.

- Se desea resolver el problema de la potencia enésima ( $x^n$ ), asumiendo que  $n$  es par y que se utilizará la siguiente recurrencia:  $\text{pot}(x, n) = \text{pot}(x, n/2) * \text{pot}(x, n/2)$  ¿Qué esquema resulta más eficiente en cuanto al coste temporal?

### Programación Dinámica.

Explicación: Usando esta recurrencia tenemos que calcular dos veces los subproblemas con lo cual almacenando los resultados nos ahorramos repetir los cálculos.

- Si  $f \in \Omega(g_1)$  y  $f \in \Omega(g_2)$  entonces  
 $f \in \Omega(g_1 + g_2)$ .

Explicación: Si  $f$  está acotada por  $g_1$  y también por  $g_2$  entonces su cota inferior es la menor de la suma de los dos esto se puede expresar como la solución, la cota inferior de la suma de  $g_1$  y  $g_2$  ó también sería equivalente decir  $f \in \Omega(\min(g_1, g_2))$  pero en las otras respuestas está puesto como no pertenece ( $\notin$ ) para engañar

- ¿Qué nos proporciona la media entre el coste temporal asintótico (o complejidad temporal) en el caso peor y el coste temporal asintótico en el caso mejor?

Nada de interés.

Explicación: La media entre el caso peor y mejor no tiene nada que ver con el caso promedio, el caso promedio depende de la densidad de casos.

Es la media de el valor de todos los casos para esa instancia del problema, mientras que lo que dice el enunciado es el valor medio entre lo mejor y lo peor de una cierta instancia del problema.

- Dado el problema de las torres de Hanoi resuelto mediante divide y vencerás, ¿Cuál de las siguientes relaciones de recurrencia expresa mejor su complejidad temporal para el caso general, siendo  $n$  el número de discos?

$$T(n) = 2T(n-1) + 1$$

Explicación: El problema de las torres de Hanoi se plantea como dos subproblemas, llevar todos los discos menos el último a la torre auxiliar (nuevo problema de  $n-1$ ), y después de pasar el último disco a su posición final el problema de mover el resto ( $n-1$ ) de la torre auxiliar a la final. De modo que se producen siempre dos subproblemas de  $n-1$  más el coste de llevar el coste de pasar el último disco a la torre final, que vale 1 porque es constante.

- ¿Qué ocurre si la cota pesimista de un nodo se corresponde con una solución que no es factible?

Que el algoritmo es incorrecto pues podría descartarse un nodo que conduce a la solución óptima.

Explicación: Las cotas pesimistas deben ser factibles, son un por lo menos tengo esto. De no ser así al comparar un nodo con la mejor hasta el momento podría descartarse por ser peor aunque esta no fuera una solución valida y podríamos no poder llegar a la solución óptima.

## - El esquema de vuelta atrás...

Garantiza que se encuentra la solución óptima a cualquier problema de selección discreta.

Explicación: Vuelta atrás recorre todo el conjunto de decisiones explorando todas las soluciones, si el problema puede plantearse como un árbol de decisiones, un problema que consiste en seleccionar opciones consecutivamente entre unas ya definidas, el algoritmo siempre encontrara la más óptima entre ellas.

## - Dado un problema de maximización resuelto mediante un esquema de ramificación y poda ¿Qué ocurre si la cota optimista resulta ser un valor excesivamente elevado?

Que se podrían explorar más nodos de los necesarios.

Explicación: La cota optimista es un valor insuperable que indica como mucho lo que se puede aspirar a obtener. Si es demasiado elevada entonces se exploraran más nodos con una estimación muy alta aunque sean valores mayores de los que realmente puede llegar a dar y por tanto estaremos explorando nodos que son innecesarios para llegar a la solución.

## - El coste temporal asintótico de insertar un elemento en un vector ordenado de forma que continúe ordenado es...

...  $O(n)$

Explicación: Esto es así ya que como mucho recorremos el vector una vez. El proceso sería empezar en un extremo del vector e incomparando hasta encontrar la posición donde pertenece y lo insertamos reorganizando el resto, con rotaciones hasta llegar al final como el algoritmo de burbuja por ejemplo.

- Dada la siguiente función:

```
int exa (string & cad, int pri, int ult) {  
    if (pri >= ult)  
        return L;  
    else  
        if (cad[pri] == cad[ult])  
            return exa (cad, pri+1, ult-1);  
        else  
            return Ø;  
}
```

¿Cuál es su complejidad asintótica?

$O(n)$

Explicación: Esta función comprueba si la cadena que se le pasa es un palíndromo.  
La complejidad tiene una cota superior de  $n$  debido a que solo se recorre  
como mucho una vez la cadena. Lo que hace es comparar que el  
primer y último carácter sean iguales y si lo son hace la llamada  
recursiva a las posiciones del segundo y penúltimo carácter y así  
 $(n-2)$     ↓  
              sucesivamente, esto quiere decir que en el peor caso que sera  
cuando haya comparado todo que habrá recorrido la cadena completa

- Dada la siguiente función: `int era(Vector <int> & v){`  
`int j, i=1, n=v.size();`  
`if(n>1) do {`  
    `int x = v[i];`  
    `for(j=i; j>0 and v[j-1] > x; j--)`  
        `v[j] = v[j-1];`  
    `v[j] = x;`  
    `i++;`  
} while (i < n);  
return 0;  
}

Marcad la opción correcta

La complejidad temporal en el mejor de los casos es  $\Omega(n)$

Explicación: lo que hace este código es organizar un vector de enteros de forma ascendente, coje la posición 1 del vector (segundo elemento) lo pone en una variable y despues hace un bucle desde esa posición hasta el principio de modo que si el anterior es mayor lo pasa a la posición siguiente, lo va llevando al final del vector y despues de acabar el bucle sobre escribe la ultima posición consultada por la variable. Y esto se va repitiendo para todos los valores del vector.

El caso mejor es que el vector está ordenado de modo que no entraña bucle y simplemente recorreria el vector una vez cogiendo el valor de cada posición para comprobarlo.

- Dada la siguiente función

```
int exa (vector<int> &v) {
    int i, sum=0, n=v.size();
    if (n>0) {
        int j=n;
        while (sum<100) {
            j=j/2;
            sum=0;
            for (i=j; i<n; i++) {
                sum+=v[i];
            }
            if (j==0) sum=100;
        }
        return j;
    } else
        return -1;
}
```

Marcad la opción correcta

La complejidad temporal en el mejor de los casos es  $\Omega(n)$

Explicación: El algoritmo ve a  $\log n$  desde la mitad del vector ( $j$ ) hasta el final sumando los valores y si no llega a 100 poner el sumador a 0 y volver a empezar con  $j$  valiendo la mitad, es decir desde un cuarto hasta el final del vector y así hasta que sume 100 ó  $j$  salga 0. Es un bucle interno que se ejecuta  $\log n$  veces y que va a ir desde  $\log n$  hasta n veces aumentando cada vez (esto equivale a  $n$ )

- Dada la siguiente función (donde  $\max(a, b) \in \Theta(1)$ ):

```

float exa(vector<float> &v, vector<int> &p, int P, int i){
    float a, b;
    if (i == 0) {
        if (p[i] <= P)
            a = v[i] + exa(v, p, P - p[i], i - 1);
        else
            a = 0;
        b = exa(v, p, P, i - 1);
    }
    return max(a, b);
}
return 0;
}

```

Markad la opción correcta

La complejidad temporal en el peor de los casos es  $O(2^n)$

Explicación: El peor de los casos cada ejecución de la función causará dos llamadas recursivas, cuando se produce más de una llamada recursiva por cada llamada la complejidad crece exponencialmente. Este algoritmo es un tipo de codificación de la mochila discreta.

-¿Cuál sería la complejidad temporal de la siguiente función tras aplicar programación dinámica?

```
doble f (int n, int m) {  
    if (n==0) return 1;  
    return m * f(n-1, m) + f(n-2, m);  
}
```

$\Theta(n)$

Explicación: En esta función pasa como con el cálculo de fibonacci al almacenar los resultados de las llamadas recursivas previas nos ahorramos las llamadas recursivas repetidas, en este caso al almacenarlo solo haremos el cálculo una vez para cada valor de  $n$  por tanto la complejidad temporal es de  $n$ .