



Pregunta 1

Incorrecta

Se puntuá 0,00 sobre 1,00

La serie denominada *tribonacci* se define de la siguiente manera: $T(0) = T(1) = 1$, $T(2) = 2$, i
 $T(n) = T(n - 3) + T(n - 2) + T(n - 1)$ para $n > 3$. Solo una de las afirmaciones siguientes es cierta. ¿Cuál es?

Seleccione una:

- a. Un algoritmo de programación dinámica iterativa para calcular $T(n)$ tendría un coste espacial $\Theta(n)$ y este coste no se podría reducir a $\Theta(1)$.
- b. No contesto (equivalente a no marcar nada). ×
- c. Un algoritmo de programación dinámica iterativa permite calcular el valor de $T(n)$ en tiempo $\Theta(n)$.
- d. Un algoritmo recursivo con memoización para calcular $T(n)$ para un n grande tendría una complejidad prohibitiva.

Pregunta 2

Incorrecta

Se puntuá -0,50 sobre 1,00

Una de las respuestas siguientes es **falsa**. ¿Cuál es? El problema del viajante de comercio ...

Seleccione una:

- a. ... se puede resolver exactamente usando un algoritmo de programación dinámica. ×
- b. No contesto (equivalente a no marcar nada).
- c. ... se puede resolver exactamente usando un algoritmo voraz derivado del de Kruskal.
- d. ... se puede resolver exactamente usando un algoritmo de búsqueda y enumeración como es el de vuelta atrás o el de ramificación y poda.

Pregunta 3

Correcta

Se puntuá 1,00 sobre 1,00

El problema del alfarero (solución continua con tiempos continuos): Se dispone de n clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, $m_i \in N$; El valor de cada pieza terminada, $v_i \in N$ y el tiempo necesario para su fabricación $t_i \in R$, $i \in [0..n - 1]$. ¿Cuántos objetos de cada clase hay que fabricar para maximizar la ganancia teniendo en cuenta que el tiempo total está limitado por $T \in R$?

Si el alfarero pudiera vender objetos sin terminar a un precio proporcional al estado de terminación. ¿Cuál de las siguientes estrategias sería más apropiada para resolverla?

Seleccione una:

- a. Un algoritmo voraz.
- b. Programación dinámica.
- c. Vuelta atrás.
- d. No contesto (equivalente a no marcar nada).



Pregunta 4

Correcta

Se puntuá 1,00 sobre 1,00

El problema del alfarero (solución discreta con tiempos discretos): Se dispone de n clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, $m_i \in N$; El valor de cada pieza terminada, $v_i \in N$ y el tiempo necesario para su fabricación $t_i \in N$, $i \in [0..n - 1]$. El tiempo total disponible viene dado por $T \in N$

Se pretende listar todas las posibilidades de fabricación de objetos. ¿Qué estrategia es la más adecuada?

Seleccione una:

- a. Un algoritmo voraz.
- b. No contesto (equivalente a no marcar nada).
- c. Ramificación y poda.
- d. Vuelta atrás.



Pregunta 5

Incorrecta

Se puntuá 0,00 sobre 1,00

Si $\lim_{n \rightarrow \infty} (g(n)/f(n)) = 0$, ¿Cuál de las siguientes expresiones NO puede darse?

Seleccione una:

- a. $g(n) \in \Omega(f(n))$
- b. No contesto (equivalente a no marcar nada).
- c. $g(n) \notin \Theta(f(n))$
- d. $f(n) \notin \Theta(g(n))$



Pregunta 6

Incorrecta

Se puntuá -0,50 sobre 1,00

La función `test()` procesa una lista de n elementos y devuelve un real. La definición de la función es recursiva. Primero descompone la lista en dos sublistas de la misma longitud usando un segmento de código que tiene una complejidad lineal con la longitud de la lista, envía cada una de las sublistas a `test()` para que la procese, hace una serie de operaciones, con el resultado y el valor de retorno, de coste temporal constante. ¿Cuál es el coste temporal asintótico de la función `test()` en función de n ?

Seleccione una:

- a. No contesto (equivalente a no marcar nada).
- b. $\Theta(n \log n)$
- c. $\Theta(\log n)$
- d. $\Theta(n)$



Pregunta 7

Incorrecta

Se puntuá -0,50 sobre 1,00

¿Cuál de las siguientes formulaciones expresa mejor la complejidad temporal, en función del parámetro n , de la siguiente función? (asumimos que n es potencia exacta de 2)

```
int f(int n){  
    int k=0;  
    for (int i = 2; i <= n; i*=2)  
        for (int j=i; j > 0; j-=2)  
            k++;  
    return k;  
}
```

Seleccione una:

- a. No contesto (equivalente a no marcar nada).
- b. $\sum_{p=2}^{n/2} \frac{p-1}{2^p}$
- c. $\sum_{p=1}^{\lfloor \log_2 n \rfloor} 2^{p-1}$
- d. $\sum_{p=1}^{\lfloor \log_2 n \rfloor} 2^{p-1}$



Pregunta 8

Correcta

Se puntuá 1,00 sobre 1,00

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

Seleccione una:

- a. El algoritmo de Kruskal se puede acelerar notablemente si los vértices se organizan en una estructura union-find.
- b. El algoritmo de Prim va construyendo un bosque de árboles que va uniendo hasta que acaba con un árbol de recubrimiento de coste mínimo. ✓
- c. No contesto (equivalente a no marcar nada).
- d. El algoritmo de Prim se puede acelerar notablemente si se guarda, para cada vértice no visitado, los datos de la arista de mínimo peso que lo une a un vértice visitado.

Pregunta 9

Incorrecta

Se puntuá -0,50 sobre 1,00

La función $\text{textrm{test}()}$ procesa una lista de n elementos y devuelve un real. La definición de la función es recursiva. Primero descompone la lista en dos sublistas de la misma longitud usando un segmento de código que tiene una complejidad lineal con la longitud de la lista, y después envía una de las dos sublistas a $\text{textrm{test}()}$ para que la procese, hace una serie de operaciones, con el resultado y el retorno, de coste temporal constante. ¿Cuál es el coste temporal asintótico de la función $\text{textrm{test}()}$ en función de n ?

Seleccione una:

- a. No contesto (equivalente a no marcar nada).
- b. $\Theta(n)$
- c. $\Theta(\log n)$ ✗
- d. $\Theta(n \log n)$

Pregunta 10

Incorrecta

Se puntuá -0,50 sobre 1,00

Cuál de las siguientes es la complejidad temporal más ajustada para un algoritmo que calcula la potencia n -ésima de una matriz cuadrada, expresada en función de n ?

Seleccione una:

- a. No contesto (equivalente a no marcar nada).
- b. $O(n)$
- c. $O(n \log n)$ ✗
- d. $O(\log n)$

Pregunta 11

Correcta

Se puntuá 1,00 sobre 1,00

El problema del cambio: Se dispone de un conjunto finito de números naturales y se pretende obtener el subconjunto de menor tamaño cuyos elementos suman una cierta cantidad C. ¿Qué estrategia es la más apropiada para resolverlo?

Seleccione una:

- a. Un algoritmo voraz.
- b. No contesto (equivalente a no marcar nada).
- c. Programación dinámica.
- d. Ramificación y poda.



Pregunta 12

Correcta

Se puntuá 1,00 sobre 1,00

El funcionamiento del algoritmo de ordenación Heapsort es similar al algoritmo de ordenación por selección, ya que localiza el valor más grande y lo sitúa en la posición final del vector; a continuación, localiza el siguiente valor más grande y lo sitúa en la posición anterior a la última, etc.

¿Cuál de las afirmaciones siguientes es **cierta**?

Seleccione una:

- a. El algoritmo Heapsort tiene una complejidad $\mathcal{O}(n)$ en el caso peor, mejor que la complejidad $\mathcal{O}(n^2)$ del algoritmo de selección, porque Heapsort utiliza una algoritmo mucho más eficiente para localizar los valores del vector que valen más.
- b. No contesto (equivalente a no marcar nada).
- c. El algoritmo Heapsort tiene una complejidad $\mathcal{O}(n \log n)$ en el caso peor, mejor que la complejidad $\mathcal{O}(n^2)$ del algoritmo de selección, porque Heapsort utiliza una algoritmo mucho más eficiente para localizar los valores del vector que valen más.
- d. Por ello, los dos algoritmos tienen la misma complejidad en el caso peor, $\mathcal{O}(n^2)$, aunque la complejidad en el caso mejor de Heapsort es $\mathcal{O}(n \log n)$.



Pregunta 13

Correcta

Se puntuá 1,00 sobre 1,00

Una de las afirmaciones siguientes es **cierta** y las otras dos **falsas**. Indicad cuál es la falsa.

Seleccione una:

- a. $\mathcal{O}(n^n) \in O(n!)$
- b. $\mathcal{O}(3^n) \in O(2^n)$
- c. La complejidad temporal de Quicksort es $\mathcal{O}(n^2)$ y $\Omega(n \log n)$
- d. No contesto (equivalente a no marcar nada).



Pregunta 14

Incorrecta

Se puntuá -0,50 sobre 1,00

El problema del alfarero (solución discreta con tiempos continuos): Se dispone de $\{n\}$ clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, $\{m_i \in N\}$; El valor de cada pieza terminada, $\{v_i \in N\}$ y el tiempo necesario para su fabricación $\{t_i \in R\}, \{i \in [0..n-1]\}$. El tiempo disponible para la fabricación de objetos está limitado por $\{T \in R\}$

Se pretende resolver mediante ramificación y poda y para ello se hace uso de una cota que consiste en coger, de entre las clases aún no consideradas, un número al azar de objetos a fabricar siempre que se cumpla las restricciones del problema ¿Que podemos decir de esta cota?

Seleccione una:

- a. Que es una cota optimista. ✗
- b. Que no es cota, ni optimista ni pesimista
- c. No contesto (equivalente a no marcar nada).
- d. Que es una cota pesimista.

Pregunta 15

Correcta

Se puntuá 1,00 sobre 1,00

En el método voraz ...

Seleccione una:

- a. ... para garantizar la solución óptima, las decisiones solo pueden pertenecer a dominios discretos o discretizables.
- b. ... es habitual preparar los datos para disminuir el coste temporal de la función que determina cuál es la siguiente decisión a tomar. ✓
- c. No contesto (equivalente a no marcar nada).
- d. ... para garantizar la solución óptima, las decisiones solo pueden pertenecer a dominios continuos.

Pregunta 16

Correcta

Se puntuá 1,00 sobre 1,00

El problema del alfarero (solución discreta con valores y tiempos continuos): Se dispone de n clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, m_i ; El valor de cada pieza terminada, v_i y el tiempo necesario para su fabricación t_i , $i \in [0..n-1]$. ¿Cuántos objetos de cada clase hay que fabricar para maximizar la ganancia teniendo en cuenta que el tiempo total está limitado por T ?

Se pretende resolverlo mediante ramificación y poda. Las siguientes funciones tratan de estimar una ganancia aproximada para la parte del nodo aún sin completar. ¿cuál es la mejor para usarla como parte de la cota optimista?

a. No contesto (equivalente a no marcar nada).

b.

```
double optimistic( const vector<int> &m, const vector<double> &v, const vector<double> &t, double T, size_t from) {

    double gain = 0.0;
    for( size_t i = from; i < m.size() && T > 0; i++ ){
        double num_objs = min( T/t[i], double(m[i]));
        gain += num_objs * v[i];
        T -= num_objs * t[i];
    }
    return gain;
}
```

c.

```
double optimistic( const vector<int> &m, const vector<double> &v, const vector<double> &t, double T, size_t from) {

    double gain = 0.0;
    for( size_t i = from; i < m.size() && T > 0; i++ ){
        for( int j = 1; j <= m[i]; j++ ) {
            gain += v[i];
            T -= t[i];
        }
    }
    return gain;
}
```

d.

```
double optimistic( const vector<int> &m, const vector<double> &v, const vector<double> &t, double T, size_t from) {

    double gain = 0.0;
    for( size_t i = from; i < m.size() && T > 0; i++ ){
        for( int j = 1; j <= m[i]; j++ ) {
            if( t[i] < T ) {
                gain += v[i];
                T -= t[i];
            }
        }
    }
    return gain;
}
```

Pregunta 17

Correcta

Se puntuá 1,00 sobre 1,00

Las soluciones factibles a un problema de optimización deben cumplir dos restricciones y queremos resolver el problema mediante vuelta atrás o ramificación y poda. ¿Cuál de las siguientes afirmaciones es **cierta**?

Seleccione una:

- a. La cota optimista usada para podar nunca se puede basar en la relajación de ninguna de las restricciones que deben cumplir las soluciones factibles.
- b. La cota optimista usada para podar se debe basar en relajar ambas restricciones simultáneamente.
- c. No contesto (equivalente a no marcar nada).
- d. La cota optimista usada para podar se puede basar en relajar una cualquiera de las dos restricciones.



Pregunta 18

Correcta

Se puntuá 1,00 sobre 1,00

Indica cuál es la complejidad, en función de $\backslash(n\backslash)$, del fragmento siguiente:

```
for( int i = 0; i < n; i++ ) {  
    A[i] = 0;  
    for( int j = 0; j < 20; j++ )  
        A[i] += B[j];  
}
```

- a. $\backslash(\backslash\Theta(n \log n)\backslash)$
- b. $\backslash(\backslash\Theta(n^2)\backslash)$
- c. $\backslash(\backslash\Theta(n)\backslash)$
- d. No contesto (equivalente a no marcar nada)



Pregunta **19**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué hace la siguiente función?

```
void f( vector<int> &A ) {  
    priority_queue<int> pq;  
  
    for( auto a: A )  
        pq.push(a);  
  
    A.clear();  
    while( !pq.empty() ) {  
        A.push_back(pq.top());  
        pq.pop();  
    }  
}
```

- a. No contesto (equivalente a no marcar nada)
- b. Invierte el vector A (el último elemento quedará el primero)
- c. Nada, deja el vector como estaba
- d. Ordena el vector A



Pregunta 20

Incorrecta

Se puntuó 0,00 sobre 1,00

Una empresa tiene (M) referencias en su stock. Cada referencia $(j \in [1, M])$ tiene un peso (p_j) y un valor (v_j) y dispone de (n_j) unidades en su stock. Dispone de un solo camión en el que puede cargar como máximo un peso (P) . Indicad cuál de las tres funciones siguientes representa una posible solución voraz aproximada al problema de cargar el camión de manera que se transporte un valor máximo.

Seleccione una:

 a. int f(

```
const vector<int> &p,
const vector<int> &v,
const vector<int> &n,
int P,
int k
) {
    if( k == 0 || P == 0 )
        return 0;
    int gain = 0
    for( int num_objs = 0; num_objs <= n[k-1]; num_objs++ )
        gain = max( gain, f(p, v, n, P - num_objs * p[k-1], k-1));
    return gain;
}
```

 b. int f(

```
const vector<int> &p,
const vector<int> &v,
const vector<int> &n,
int P,
int k
) {
    if( k == 0 || P == 0 )
        return 0;
    int gain = 0
    for( int num_objs = 0; num_objs <= 1; num_objs++ )
        gain = max( gain, f(p, v, n, P - num_objs * p[k-1], k-1));
    return gain;
}
```

 c. int f(

```
const vector<int> &p,
const vector<int> &v,
const vector<int> &n,
int P,
int k
) {
    if( k == 0 || P == 0 )
        return 0;
    int num_objs = min( P/p[k-1], n[k-1]);
    return num_objs * v[k-1] + f( p, v, n, P - num_objs * p[k-1], k-1);
}
```

 d. No contesto (equivalente a no marcar nada). ×

Pregunta 21

Incorrecta

Se puntuá -0,50 sobre 1,00

El problema del alfarero (solución discreta con tiempos continuos): Se dispone de $\{n\}$ clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, $\{m_i \in N\}$; El valor de cada pieza terminada, $\{v_i \in N\}$ y el tiempo necesario para su fabricación $\{t_i \in R\}, \{i \in [0..n-1]\}$. ¿Cuántos objetos de cada clase hay que fabricar para maximizar la ganancia teniendo en cuenta que el tiempo total está limitado por $\{T \in R\}$?

¿Cuál de los siguientes esquemas algorítmicos resultaría más eficiente para resolverlo?

Seleccione una:

- a. Programación dinámica.
- b. Un algoritmo voraz. ✗
- c. No contesto (equivalente a no marcar nada).
- d. Vuelta atrás.

Pregunta 22

Correcta

Se puntuá 1,00 sobre 1,00

El problema del alfarero (solución discreta con valores y tiempos discretos): Se dispone de $\{n\}$ clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, $\{m_i \in N\}$; El valor de cada pieza terminada, $\{v_i \in N\}$ y el tiempo necesario para su fabricación $\{t_i \in N\}, \{i \in [0..n-1]\}$. ¿Cuál es el valor máximo de los objetos que puede fabricar teniendo en cuenta que el tiempo total está limitado por $\{T \in N\}$?

Para ello se escribe la siguiente función siguiendo la técnica de divide y vencerás:

```
int potter( const vector<int> &v, const vector<int> &m, const vector<int> &t, int T, int k ) {  
    if( k == 0 )  
        return 0;  
    int max_earnings = -1;  
    for( int c = 0; c <= m[k-1]; c++ ) {  
        int earnings = 0;  
        if( T >= c * t[k-1] )  
// ==> Falta una línea <==  
            max_earnings = max( max_earnings, earnings );  
    }  
    return max_earnings;  
}
```

¿Cuál es la línea que falta?

- a. No contesto (equivalente a no marcar nada)
- b. ✓
- c.
- d.

Pregunta 23

Incorrecta

Se puntuá 0,00 sobre 1,00

Sea el vector $\langle v = \{1, 3, 2, 7, 4, 6, 8\} \rangle$ cuyos elementos están dispuestos formando un montículo de mínimos. Posteriormente añadimos en la última posición del vector un elemento nuevo con valor 5. ¿Qué operación hay que hacer para que el vector siga representando un montículo de mínimos?

Seleccione una:

- a. No contesto (equivalente a no marcar nada). ✗
- b. No hay que hacer nada pues el vector $\langle v = \{1, 3, 2, 7, 4, 6, 8, 5\} \rangle$ también es un montículo de mínimos.
- c. Intercambiar el 7 con el 5.
- d. Intercambiar el 8 con el 5.

Pregunta 24

Correcta

Se puntuá 1,00 sobre 1,00

Indica cuál es la complejidad, en función de $\langle n \rangle (\langle n \geq 0 \rangle)$, del fragmento siguiente:

```
int f( int n ) {
    if( n == 0)
        return n;
    return f(n/2)*f(n/2);
}
```

- a. No contesto (equivalente a no marcar nada)
- b. $\langle \Theta(n) \rangle$ ✓
- c. $\langle \Theta(n \log n) \rangle$
- d. $\langle \Theta(\log n) \rangle$

Pregunta 25

Correcta

Se puntuá 1,00 sobre 1,00

Dada la siguiente función recursiva:

```
unsigned f( unsigned a, unsigned b ) {  
    if( a < 3 )  
        return a + 2*b;  
    return f(a-1, (7*b)%10);  
}
```

donde suponemos que siempre se va a invocar la función con $b < 10$.

Queremos acelerarla aplicando la técnica de programación dinámica iterativa. ¿Cómo quedaría?

a.

```
unsigned f( unsigned a, unsigned b ) {  
    vector< vector<unsigned>> M(a, vector<unsigned>(10));  
    for( unsigned j = 0; j < 10; j++ )  
        for( unsigned i = 0; i <= a; i++ )  
            if( i < 3 )  
                M[i][j] = i + 2*j;  
            else  
                M[i][j] = M[i-1][(7*j)%10];  
    return M[a][b];  
}
```

b. No contesto (equivalente a no marcar nada)

c.

```
unsigned f( unsigned a, unsigned b ) {  
    vector< vector<unsigned>> M(a+1, vector<unsigned>(10));  
    for( unsigned i = 0; i <= a; i++ )  
        for( unsigned j = 0; j < 10; j++ )  
            if( i < 3 )  
                M[i][j] = i + 2*j;  
            else  
                M[i][j] = M[i-1][(7*j)%10];  
    return M[a][b];  
}
```

d.

```
unsigned f( unsigned a, unsigned b ) {  
    vector< vector<unsigned>> M(a+1, vector<unsigned>(10));  
    for( unsigned j = 0; j < 10; j++ )  
        for( unsigned i = 0; i <= a; i++ )  
            if( i < 3 )  
                M[i][j] = i + 2*j;  
            else  
                M[i][j] = M[i-1][(7*j)%10];  
    return M[a][b];  
}
```

Pregunta **26**

Correcta

Se puntuá 1,00 sobre 1,00

Dada la siguiente función construida mediante la técnica memoización:

```
int f( vector<unsigned> &x, unsigned i ) {  
    if( x[i] != SENTINEL )  
        return x[i];  
    if( i < 5 )  
        return i;  
    return x[i] = f(x, i-1) + f(x, i-3);  
}
```

```
int f( unsigned i ){  
    vector<unsigned> x(i, SENTINEL);  
    return f( x, i );  
}
```

¿Cuál es la declaración para SENTINEL más adecuada?

- a. `const unsigned SENTINEL = -1;`
- b. `const unsigned SENTINEL = numeric_limits<unsigned>::max();`
- c. `const unsigned SENTINEL = 0;`
- d. No contesto (equivalente a no marcar nada)

Pregunta **27**

Correcta

Se puntuá 1,00 sobre 1,00

¿Qué complejidad tiene la siguiente función?

```
void f( vector<int> &A ) {  
    priority_queue<int> pq( begin(A), end(A));  
  
    A.clear();  
    while( !pq.empty() ) {  
        A.push_back(pq.top());  
        pq.pop();  
    }  
}
```

Suponed que la cola de prioridad está implementada como un heap y que $\lceil n \rceil = A.size()$.
`priority_queue<int> pq(begin(A), end(A))` construye un heap a partir de los datos que hay en el vector A.

- a. $\Theta(n^2)$
- b. $\Theta(n)$
- c. $\Theta(n \log n)$
- d. No contesto (equivalente a no marcar nada)

Pregunta 28

Incorrecta

Se puntuá 0,00 sobre 1,00

Queremos aplicar la técnica de memoización a la siguiente función recursiva:

```
double f( double x ) {  
    if( x <= 2 )  
        return x;  
    return f(sqrt(x-1)) + f(sqrt(x-2));  
}
```

¿Cuál sería un buen candidato para el almacén? [la función sqrt() calcula la raíz cuadrada]

- a. vector<double> M(xMax+1); (donde xMax es el valor de x en la primera llamada)
- b. No se puede aplicar la técnica de memoización
- c. vector<double, double> M(xMax+1,xMax+1); (donde xMax es el valor de x en la primera llamada)
- d. No contesto (equivalente a no marcar nada) ✗

Pregunta 29

Incorrecta

Se puntuá -0,50 sobre 1,00

¿Cuál de las siguientes formulaciones expresa mejor el número de llamadas recursivas que hace Quicksort en el mejor de los casos?

Seleccione una:

- a. No contesto (equivalente a no marcar nada).
- b. $\sum_{i=0}^n \log n$
- c. $\sum_{i=0}^{\log n} 2^i$
- d. $\sum_{i=0}^{\log n} 1$ ✗

Pregunta 30

Incorrecta

Se puntuá -0,50 sobre 1,00

Se dispone de n clases de objetos. De cada una de ellas se conoce el número máximo de piezas que se puede fabricar, $m_i \in N$ y el tiempo necesario para su fabricación $t_i \in R$, $i \in [0..n-1]$. Queremos listar todas las posibilidades de fabricación de objetos teniendo en cuenta que el tiempo total está limitado por $T \in R$.

Para ello hemos hecho el siguiente programa donde faltan unas líneas:

```
void combinations( const vector<int> &m, const vector<double> &t, double T, size_t k, vector<int> &x) {  
    if( k == m.size() ) {  
        print_comb(x);  
        return;  
    }  
    // ==> Aquí falta código <==  
}  
  
void combinations( const vector<int> &m, const vector<double> &t, double T ) {  
    vector<int> x(m.size());  
    combinations(m, t, T, 0, x);  
}
```

¿Cuales son las líneas que faltan? [suponed que print_comb() imprime correctamente la combinación que hay codificada en x]

- a.

```
for( int j = 0; j < m[k]; j++ ) {  
    x[j] = k;  
    if( T >= j * t[k] )  
        combinations( m, t, T - j * t[k], k+1, x );  
}
```
- b. No contesto (equivalente a no marcar nada)
- c.

```
for( int j = 0; j < m[k]; j++ ) {  
    x[k] = j;  
    if( T >= j * t[k] )  
        combinations( m, t, T - j * t[k], k+1, x );  
}
```

✗
- d.

```
for( int j = 0; j <= m[k]; j++ ) {  
    x[k] = j;  
    if( T >= j * t[k] )  
        combinations( m, t, T - j * t[k], k+1, x );  
}
```