Universidad de Valladolid
Departamento de Informática

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

I.T.I. Sistemas – Estructuras de Datos – Curso 04/05

Problemas de Análisis y Diseño de Algoritmos.

1. Un programador ha implementado un algoritmo que tarda, en su antiguo ordenador de 100 Mhz, una milésima de segundo en resolver un problema de tamaño $n = 20$. ¿Cuánto tardaría en resolver un problema de tamaño $n = 30$ en los siguientes casos?
   a) El algoritmo es de orden $O(n^3)$
   b) El algoritmo es de orden $O(2^n)$
   c) El algoritmo es de orden $O(n!)$

   ¿Por que factor debería multiplicar la velocidad de su ordenador para que volviera a tardar una milésima de segundo? **Nota**: 1 año = 31.557.600 segundos.

2. ¿Es posible que un algoritmo de orden $O(n^2)$ se ejecute más rápidamente que un algoritmo de orden $O(n)$ en los siguientes casos?
   a) Para un tamaño n = 100.
   b) Para un determinado valor de la entrada.
   c) Para el mejor caso del primer algoritmo.
   d) Para todos los valores de la entrada.
   e) Si se ejecutan en máquinas distintas y podemos elegir aquella en que se ejecuta el primer algoritmo.

3. Al medir experimentalmente un algoritmo se encuentra que nunca emplea más de $k_1 \cdot n^3$ operaciones ni menos de $k_2 \cdot n^2$, siendo n el tamaño de los datos de entrada y k_1 y k_2 constantes. ¿Cuál sería la hipótesis más adecuada al interpretar estos datos?
   - El algoritmo es $O(n^2)$ y $\Omega(n^3)$
   - El algoritmo es $O(n^3)$ y $\Omega(n^2)$
   - El algoritmo es $\Theta(n^{2.5})$
   - El algoritmo es $\Theta(n^2 + n^3)$

4. Analizar cual sería el orden de complejidad del algoritmo que usamos habitualmente para multiplicar dos números enteros. Se supondrá que los dos números tienen el mismo número de dígitos, n, y que este valor define el tamaño de la entrada. Contar únicamente las operaciones de producto y suma de dígitos individuales.

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

5. Analizar el número de operaciones en las que interviene un elemento del vector en el siguiente algoritmo (ordenación por selección):

```
type TVector = array[1..N] of real;

procedure ordenar(var V: TVector);
var
  I,J,Min : integer;
  Temp : real;
begin
  for I := 1 to N-1 do
  begin
    Min := I;
    for J := I+1 to N do
      if V[J] < V[Min] then Min := J;
    Temp := V[I];
    V[I] := V[Min];
    V[Min] := Temp
  end
end
```

Encontrar las fórmulas exactas para el peor y el mejor caso, identificar cuales son los tipos de entradas que producen esos casos y repetir el análisis utilizando notación asintótica.

6. Calcular el número de operaciones de incremento de *x* que realizan los siguientes fragmentos de programas, teniendo en cuenta que sólo deseamos conocer exactamente el término de mayor crecimiento, y el resto de términos se pueden expresar usando notación asintótica:

```
for i := 1 to n do
  for j := 1 to n do
    if j = i then
      x := x+1
```

```
for i := 1 to n do
  for j := n downto i do
    for k := i to j do
      x := x+1
```

```
for i := 1 to n do
  for j := 1 to i*i do
    for k := 1 to j*j do
      x := x+1
```

7. Calcular el orden de complejidad de las siguientes funciones recursivas (contar sólo las operaciones producto):

```
function f1(n: integer): integer;
begin
  if n < 1 then f1 := 1 else
    f1 := f1(n-3)*f1(n-3)
end;
```

```
function f2(n: integer): integer;
begin
  if n < 1 then f2 := 1 else
    f2 := f2(n div 2)+f2(n div 2)*
          f2(n div 2)
end;
```

```
function f3(n: integer): integer;
var i,x: integer;
begin
  if n < 1 then f3 := 1 else
  begin
    x := 1;
    for i := 1 to n do x := x*n;
    f3 := x+f3(n-1)
  end
end
```

```
function f4(n: integer): integer;
var i,x: integer;
begin
  if n < 1 then f4 := 1 else
  begin
    x := 1;
    for i := 1 to n*n do x := x*2;
    f4 := x*f4(n div 2)
  end
end
```

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

8. Demostrar si los siguientes enunciados son ciertos o falsos:

   a) ¿$\log 3^n \in \Theta(n)$?

   b) ¿$2^{n+1} \in O(2^n)$?

   c) ¿$3^n \in \Omega(2^n)$?

   d) ¿$2n^2 - n + 1 \in \Omega(n)$?

   e) ¿$\lg(n!) \in \Theta(n \lg n)$?

   Nota: Para el último caso puede ser conveniente utilizar el método de inducción: Suponer que es cierto para un determinado valor de *n* y demostrar que eso implica necesariamente que la expresión es cierta para *n*+1.

9. Calcular el orden de complejidad respecto al tiempo (suponiendo que deseamos contar las operaciones producto) y al espacio de la siguiente función:

```
function f(n: integer) : integer;
var i,x: integer;
begin
  if n < 1 then f := 1 else
  begin
    x := n;
    for i := 1 to n do x := x+1;
    f := f(n div 2)*f(n div 2)+f(n div 3)+x
  end
end;
```

10. Al analizar la eficiencia de una operación de inserción sobre una estructura de datos se encuentra que a veces la inserción tarda un tiempo $\Theta(n^3)$ pero eso implica que las siguientes *n* inserciones tardarán un tiempo de $O(1)$. Analizar el comportamiento del algoritmo en el peor y mejor caso, el caso promedio y el tiempo amortizado.

11. El algoritmo de búsqueda ternaria es similar a la búsqueda binaria pero se diferencia en que en cada etapa el subvector se divide en **tres** partes iguales (en lugar de dos) y se examinan los dos elementos que separan estas tres partes para decidir si se ha encontrado el valor o, en caso contrario, elegir una de estas tres partes para continuar la búsqueda en ella.

    Analizar cuál sería la complejidad de este algoritmo para las operaciones en que interviene un elemento del vector (se puede suponer que el tamaño del vector es una potencia de tres) y comprobar si mejora o no el orden de la búsqueda binaria.

12. Resolver el problema de las torres de Hanoi con cuatro postes. Evaluar el orden del algoritmo y comprobar si supone una mejora respecto al problema con tres postes. Verificar si es posible aplicar una estrategia divide y vencerás al problema.

13. Resolver el problema de las torres de Hanoi con tres postes cuando se parte de cualquier posición válida. **Sugerencia**: Encontrar primero un esbozo de solución y diseñar después el tipo de datos que represente la posición de los discos que sea más adecuado para esa forma de resolver el problema. Evaluar el orden del algoritmo obtenido.

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

14. El problema de evaluar la expresión a^b donde a es un valor real y b un entero positivo se puede resolver mediante un bucle donde se evalúe b veces el producto de a, o bien es posible aplicar la siguiente fórmula en un esquema divide y vencerás:

$$a^b = \begin{cases} \left(a^{b \operatorname{div} 2}\right)^2 & \text{si } b \text{ es par} \\ a \cdot \left(a^{b \operatorname{div} 2}\right)^2 & \text{si } b \text{ es impar} \end{cases}$$

    Desarrollar algoritmos para ambas soluciones y evaluar el orden de complejidad de cada uno tomando como tamaño de la entrada el número de bits de b.

15. Se dispone de una matriz de $n \times n$ elementos que pueden tomar uno de estos tres posibles valores: *pared*, *libre* o *marca*. La matriz representa un laberinto donde desde una casilla es posible acceder a cualquier casilla adyacente (superior, inferior, izquierda y derecha) que tenga el valor *libre* o *marca*. Inicialmente se proporciona la matriz que define el laberinto donde las casillas tienen los valores *libre* o *pared*, y la fila y columna de partida.

    Desarrollar un algoritmo que indique si existe o no un camino para salir del laberinto partiendo de la posición inicial y pasando únicamente por casillas marcadas inicialmente como *libres*. Salir del laberinto significa alcanzar una casilla libre en los límites de la matriz. Si la posición inicial está fuera del laberinto o corresponde a una casilla con valor *pared*, la respuesta será falso.

    Nota: Se permite modificar el contenido de la matriz haciendo uso del valor marca con el propósito de no volver a pasar por una casilla ya recorrida.

    Modificar el algoritmo obtenido para que, en el caso de que exista una solución, imprima las filas y columnas de las casillas que se deben recorrer para salir del laberinto.

16. Dado un número entero n se desea obtener todas las posibles formas de expresarlo como sumas de los enteros 1 .. n, permitiendo que el mismo sumando aparezca varias veces. Se considerarán iguales (y por lo tanto sólo se debe escribir una de ellas) aquellas soluciones que contengan los mismos sumandos pero ordenados de otra manera. Por ejemplo, si $n = 4$, se deberían escribir las soluciones 4=1+1+1+1, 4=2+1+1, 4=2+2, 4=3+1, 4=4.

    Crear un algoritmo que resuelva el problema. No importa el orden en que aparecen las soluciones ni el orden de los sumandos dentro de cada solución.

17. Crear un programa que lea una cadena de caracteres, compruebe no existe en ella ningún carácter repetido, y en ese caso escriba todas las cadenas que se pueden formar por permutaciones de los caracteres de la cadena original. Por ejemplo, si la cadena leída fuera "DATO", el programa escribiría:

DATO	DAOT	DTAO	DTOA	DOAT	DOTA
ADTO	ADOT	ATDO	ATOD	AODT	AOTD
TDAO	TDOA	TADO	TAOD	TODA	TOAD
ODAT	ODTA	OADT	OATD	OTDA	OTAD

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

18. El problema las ***n* reinas** consiste en colocar *n* reinas sobre un tablero de ajedrez de *n* x *n* casillas de tal forma que ninguna reina pueda *amenazar* a otra. Aplicando la técnica de backtracking crear un algoritmo que, dado un valor de *n* entre 1 y 12:

    a) Presente una solución o bien indique que no existe ninguna.

    b) Presente todas las soluciones existentes.

    Diseñar los tipos de datos más adecuados para facilitar la operación de comprobar si una casilla es accesible a alguna de las reinas dispuestas en el tablero.

19. Se dispone de un tablero de ajedrez de *n* x *n* casillas. El problema del **recorrido del caballo** consiste en encontrar una forma de recorrer, comenzando en la casilla (1,1), todas las casillas del tablero pasando de una a otra mediante saltos de caballo y sin pasar más de una vez por cada casilla. Crear un algoritmo que resuelva el problema.

20. Analizar cómo se podría aplicar la estrategia divide y vencerás al problema de multiplicar dos números positivos de *n* bytes de longitud. Se puede suponer que los números están almacenados en vectores cuyos elementos son de tipo byte y que *n* (el tamaño de los vectores) es potencia de dos.

21. Analizar el orden asintótico del algoritmo anterior y compararlo con el algoritmo clásico de multiplicación. Sólo es necesario contar las operaciones de suma, resta y producto entre valores de tipo byte.

22. Crear un algoritmo recursivo que escriba por pantalla un valor *n* en base *b* ($2 \le b \le 10$). Estimar el orden de complejidad del algoritmo en función del número de dígitos de *n*, contando las operaciones de resta y división. Aplicar la estrategia divide y vencerás al algoritmo anterior y comprobar si mejora el orden asintótico obtenido.

23. Usar la técnica de la tabla de resultados parciales para desarrollar un algoritmo que calcule el coeficiente binomial $C(n,m)$. ¿Cuál es el orden de complejidad temporal y espacial del algoritmo? Compararlo con el orden de la solución recursiva.

$$C(n,m) = \begin{cases} 1 & \text{caso contrario} \\ C(n-1,m) + C(n-1,m-1) & 0 < m < n \end{cases}$$

24. ¿Es posible utilizar la técnica de la tabla de resultados parciales para calcular la función de Ackerman?

$$A(n,m) = \begin{cases} m+1 & \text{si } n = 0 \\ A(n-1,1) & \text{si } m = 0 \\ A(n-1, A(n,m-1)) & \text{caso contrario} \end{cases}$$

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

25. **El problema de la mochila 0/1**. En este problema disponemos de dos vectores, $v_1...v_n$, y $p_1...p_n$ que indican, respectivamente, el valor y el peso de cada uno de n objetos. El objetivo es decidir, para cada objeto, si lo incluimos o no en una mochila cuya capacidad máxima es M kilos. De entre todas las maneras distintas de llenar la mochila sin superar ese límite, debemos escoger aquella que proporcione la mochila más valiosa.

    Es decir, si expresamos la solución como el vector $x_1...x_n$ donde $x_i = 1$ indica que el objeto i se incluye en la mochila y $x_i = 0$ indica que el objeto i no se incluye, se debe cumplir que:

    $$\sum_{i=1}^{n} x_i \cdot p_i \le M \text{ (restricción), y que } \sum_{i=1}^{n} x_i \cdot v_i \text{ (valor de la mochila) es máximo.}$$

    Resolver este problema utilizando backtracking y estimar el orden de complejidad en el peor caso, contando las operaciones en que se acceda a vectores.

26. Dado un conjunto de n enteros positivos, $v_1...v_n$, determinar si existe algún subconjunto cuya suma sea igual al valor s (también positivo).

27. El **problema de la herencia** consiste en averiguar si existe un reparto de n objetos en dos partes cuyo valor sea el mismo. El valor de cada objeto viene dado por el vector $v_1...v_n$.
    - Intentar encontrar una solución mediante programación dinámica. Verificar si se cumple el principio de suboptimalidad.
    - Reformular el problema de manera que pueda resolverse en términos del problema anterior.

28. Un cierto número de usuarios, n, envían simultáneamente un documento a la impresora común, la cual debe determinar su orden de impresión. Las longitudes de los documentos enviados son $l_1...l_n$, siendo l_i la longitud del documento enviado por el usuario i (la numeración de los usuarios es arbitraria). Suponiendo que el tiempo que se tarda en imprimir un documento es proporcional a su longitud, diseñar un algoritmo que indique el orden óptimo en que se deben imprimir de manera que se minimice el tiempo medio de espera de cada usuario.

    El tiempo de espera del usuario i-ésimo vendrá dado por el orden que haya establecido la impresora para su documento. Si su documento es el j-ésimo en imprimirse, su tiempo de espera será la suma de los tiempos de impresión de los j primeros documentos según ese orden (se incluye el suyo en la suma).

29. Se dispone de una secuencia de n enteros, $v_1...v_n$, y se desea hallar la subsecuencia cuya suma sea máxima. Por ejemplo, dada la secuencia -2, 10, 5, -3, 4, -6, -2, 7 la subsecuencia de suma máxima es la comprendida entre los índices 2 y 5 (10, 5, -3, 4).

    a) Obtener un algoritmo $O(n^3)$ utilizando la estrategia de fuerza bruta (comprobar las $(n^2 - n)/2$ subsecuencias posibles).

    b) Refinar el algoritmo anterior para obtener un orden $O(n^2)$.

    c) Aplicar la estrategia divide y vencerás para conseguir un algoritmo $O(n \lg n)$.

    d) Intentar encontrar un algoritmo heurístico con un tiempo $O(n)$.

Edificio de Tecnologías de la Información | Campus Miguel Delibes Ctra. del Cementerio s/n | Tel.: (983) 42 36 70 Fax: (983) 42 36 71

30. Dada una secuencia descendente de enteros, $v_1...v_n$, crear un algoritmo que encuentre la posición del menor valor positivo de la secuencia en un tiempo mejor que $O(n)$.

31. Dado un vector ordenado de enteros, $v_1...v_n$, donde no existen elementos repetidos, crear un algoritmo que determine si existe algún elemento igual a su índice (es decir, si $v_i = i$) en un tiempo mejor que $O(n)$.

32. Dada una secuencia de valores, $X \equiv x_1 \ldots x_n$, se denomina **subsecuencia no contigua** de X a toda secuencia cuyos componentes pertenecen a un subconjunto de valores de X y están dados con el mismo orden de aparición que en la secuencia X. Es decir, que la secuencia $Y \equiv y_1 \ldots y_m$ es una subsecuencia no contigua de X si cumple:

$$\forall\, i : 1 \le i < m : \exists\, j, k : j < k : (y_i = x_j) \wedge (y_{i+1} = x_k)$$

Por ejemplo, una posible subsecuencia no contigua de (1,2,3,4,5,6,7,8,9) podría ser la secuencia (2,3,5,7,8,9). Resolver el problema anterior cuando no se exige que la subsecuencia sea contigua.

33. Dada la secuencia de enteros $v_1...v_n$ diseñar un algoritmo para encontrar la subsecuencia ascendente no contigua de mayor longitud (número de elementos). **Nota**: Ascendente significa que cada elemento sea estrictamente mayor que el anterior.

34. Se dispone de dos secuencias de valores, A $\equiv a_1 \ldots a_n$ y B $\equiv b_1 \ldots b_m$. Se desea encontrar la mayor (en el sentido de tener más elementos) secuencia C $\equiv c_1 \ldots c_p$ que cumpla ser una subsecuencia no contigua de A y ser una subsecuencia no contigua de B.

Por ejemplo, dadas las secuencias de caracteres CGATAATTGAGA y GTTCCTAATA, una solución al problema podría ser la secuencia GTAATA, de seis caracteres de longitud (se muestran subrayados los caracteres que forman parte de la subsecuencia).

Nota: Este problema se suele denominar **subsecuencia común maximal**, y aparece en muchos contextos distintos, por ejemplo:

- Para obtener un índice de similitud entre dos documentos. El tamaño de la subsecuencia solución puede utilizarse para medir el parecido (las partes comunes) entre documentos.
- Para crear ficheros de actualización de programas (más conocidos como *patches*), en los que sólo se incluya las partes que han cambiado respecto al programa original.
- En ingeniería genética, para identificar genes similares entre especies distintas o mutaciones sobre una misma especie.

35. Se dispone de un vector que contiene n enteros positivos cuyo valor está acotado entre 1 y n^2. ¿Alguna de las estrategias de ordenación estudiadas puede ordenar sus elementos en un tiempo $O(n)$?

36. Se dispone de un vector que contiene n elementos, todos ellos iguales. ¿Cuál sería la eficiencia del algoritmo de ordenación rápida (en sus distintas variantes) aplicado a éste vector?

37. Encontrar el orden de los enteros 1,2,3,4,5 que provoca el peor caso para el algoritmo de ordenación rápida con el segundo método de partición y cuyo elemento pivote es el medio.

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

38. Se desea ordenar en un tiempo de $O(n)$ una estructura de datos que almacena n elementos representando uno de tres colores posibles (rojo, blanco o azul). Los colores pueden compararse entre sí, siendo rojo < blanco < azul.
    - Crear un algoritmo que resuelva el problema suponiendo que la estructura es un vector.
    - Crear un algoritmo que resuelva el problema suponiendo que la estructura es un TAD donde las únicas operaciones posibles son **color(i)**, que devuelve el color i-ésimo e **intercambia(i,j)**, que intercambia los colores i y j.

39. Diseñar un TAD que represente un conjunto de elementos en el que se pueden realizar las operaciones de añadir un elemento al conjunto, comprobar la pertenencia de un elemento al conjunto y comprobar si un conjunto es igual a otro. Los elementos pueden compararse entre sí mediante el operador de igualdad.

40. En una estructura de datos se encuentran almacenados n elementos con claves numéricas, y los valores de esas n claves son, en orden ascendente, $c_1 \ldots c_n$. Se sabe que a la estructura de datos no se le van a añadir ni eliminar elementos, y que la única operación que se va a realizar sobre ella es buscar el elemento con un valor concreto de la clave (se puede suponer que todas las búsquedas son exitosas). Se conoce, además, cual es la frecuencia con que se va a preguntar sobre cada valor de la clave, y se dispone del vector que almacena esa información, $f_1 \ldots f_n$. ($f_1+f_2+\ldots+f_n = 1$).
    - Suponiendo que la estructura de datos es un árbol binario de búsqueda, crear un algoritmo que indique cual es la organización óptima de ese árbol, en el sentido que minimice, a largo plazo, el tiempo que se tarda en realizar búsquedas que obedezcan a la distribución de frecuencia proporcionada.
    - Indicar otra posible estructura de datos para este problema.

41. Se desean implementar operaciones de pila sobre un conjunto de datos almacenados en una estructura de tipo cola. Suponiendo que la cola tiene una implementación óptima y que el espacio adicional disponible es $O(1)$, crear algoritmos que implementen las operaciones de añadir y quitar elementos de pila indicando su complejidad.

42. Se desean empaquetar n ficheros de tamaños $a_1 \ldots a_n$, para formar un único fichero que los englobe. El sistema operativo proporciona un comando para empaquetar ficheros, pero desafortunadamente sólo permite empaquetar dos ficheros a la vez. Se usará la notación **[A,B]** para representar el fichero resultante de empaquetar los ficheros **A** y **B** utilizando esta herramienta.

    Diseñar un algoritmo que implemente la solución más eficiente para empaquetar n ficheros, teniendo en cuenta que el tiempo que se tarda en empaquetar dos ficheros es igual a la suma de sus tamaños, y que el tamaño del fichero resultante también es igual a la suma de esos tamaños.

    Por ejemplo, si se tienen 3 ficheros, **A**, **B** y **C**, de tamaños 3, 1 y 5, el empaquetamiento **[A [B C]]** tardará 15 unidades de tiempo (6 de empaquetar **B** y **C** mas 9 de empaquetar el resultado con **A**), mientras que el empaquetamiento **[[A B] C]** tardaría 13 unidades de tiempo.

    Nota: El empaquetamiento debe respetar el orden original de los ficheros. Por ejemplo, **[C [A B]]** no sería un empaquetamiento válido.

Universidad de Valladolid
Departamento de Informática

Edificio de Tecnologías de la Información | Campus Miguel Delibes, Ctra. del Cementerio s/n | Tel.: (983) 42 36 70, Fax: (983) 42 36 71

43. El lenguaje de programación FORTH, que se hizo popular a finales de los años 70, disponía únicamente de dos tipos de datos: Enteros y estructuras tipo pila. Las operaciones que se podían realizar sobre las pilas eran la inserción de un elemento (que pasaba a ser la cabeza de la pila), la extracción de la cabeza de la pila y la consulta del número de valores almacenados. Todas estas operaciones se realizaban en un tiempo $O(1)$.

    Analizar cual sería el coste temporal y espacial de implementar el algoritmo de ordenación por fusión en este lenguaje. El algoritmo partiría de una pila con n enteros almacenados y devolvería la misma pila pero con los valores organizados de manera que al extraer todos los elementos de la pila formasen una sucesión creciente. Se pueden usar todas las variables auxiliares que se requieran, teniendo en cuenta, por supuesto, que sólo pueden ser de tipo entero o pila.

44. Dar un algoritmo para imprimir el contenido de una lista simplemente enlazada en orden inverso. El algoritmo deberá tener un tiempo de ejecución de $O(n)$, utilizando un espacio adicional de $O(1)$.

    Nota: A pesar de que la operación pedida es de lectura, se permite que el algoritmo vaya modificando la estructura de datos, siempre y cuando al finalizar la deje como estaba.

45. Describa cómo almacenar matrices triangulares (inferiores y superiores) y k-diagonales de manera que no se malgaste espacio y se pueda seguir accediendo a los elementos en $O(1)$.

46. Proporcionar un algoritmo que realice la búsqueda binaria sobre una lista enlazada ordenada. Analizar su tiempo de ejecución.

47. Dado el tipo abstracto de datos pila, dar un algoritmo para obtener el valor del menor elemento almacenado (la pila debe quedar como estaba).

48. ¿Son correctas las ecuaciones que describen las operaciones del siguiente TAD?

    ESPECIFICACIÓN TAD_EXAMEN
    USA ENTEROS, BOOLEANOS
    TAD bloque[elemento]
    OPERACIONES
    - $\varnothing : \rightarrow$ bloque
    - $\square$ + $\square$: bloque, elemento $\rightarrow$ bloque
    - $\square$ = $\square$: bloque, bloque $\rightarrow$ booleano

    VARIABLES
    x, y : elemento; b1, b2 : bloque
    ECUACIONES
    - $\varnothing = \varnothing$ == **T**
    - (x + b1) = (x + b2) == b1 = b2
    - $(x \neq y) \Rightarrow$ (x + b1) = (y + b2) == **F**

    FIN_ESPECIFICACIÓN