Centro Universitario de Mérida

TEMA 2 – "Divide y vencerás"

1. (Clase) Un programa que utiliza la técnica divide y vencerás, divide un problema de tamaño **n** en **a** subproblemas de tamaño **n/b**. El tiempo g(n) de la resolución directa (caso base) se considerará constante. El tiempo f(n) de combinar los resultados es $O(n^p)$, para simplificar consideraremos que $f(n) = d \cdot n^p$. Obtener el orden total de ejecución del algoritmo t(n), en función de los valores **a** y **b**.

2. Comprobar que los resultados anteriores son aplicables para cualquier valor de g(n) y para valores de f(n) de la forma $f(n) = a_p n^p + a_{p-1} n^{p-1} + ... + a_1 n + a_0$.

3. Usando la técnica divide y vencerás, resolver el problema de calcular la media de un array de **n** enteros, siendo n potencia de 2. Modificar el algoritmo para permitir valores de n que no sean potencia de 2. ¿Qué mejora se obtiene respecto a una simple versión iterativa? ¿Existe alguna situación en la que sea mejor la solución con divide y vencerás?

4. Supongamos que un problema puede resolverse de forma directa en un tiempo g(n) o bien, usando divide y vencerás, puede ser dividido en varios subproblemas de igual tamaño, necesitando en la división y en la combinación de resultados un tiempo f(n).
   a) Demostrar que si $O(g(n)) \subseteq O(f(n))$ entonces la solución usando divide y vencerás es siempre peor o igual (en cuanto a orden de complejidad) que resolver el problema usando el método directo. ¿Qué significa lo anterior y qué consecuencias tiene en la construcción de algoritmos con divide y vencerás?
   b) Demostrar que si $O(f(n)) \subseteq O(g(n))$, no siempre tiene que ser mejor la solución usando divide y vencerás. Sugerencia: la demostración se puede hacer dando un contraejemplo con valores de f(n) y g(n) adecuados.

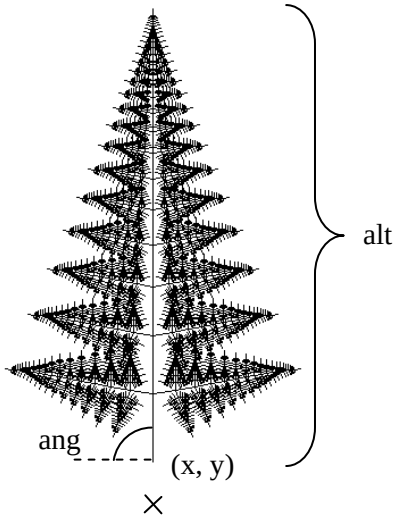
5. Un algoritmo divide y vencerás divide un problema de tamaño n en 2 subproblemas de tamaño n-1. El algoritmo aplica una solución directa cuando n = 2, que tarda un tiempo de 3 ms. Si el tiempo de realizar la combinación es f(n) = 1.2n ms, calcular el tiempo de ejecución exacto del algoritmo t(n).

6. Supongamos el problema de calcular todos los caminos mínimos en un grafo no dirigido. En general, ¿es posible descomponer fácilmente el problema en subproblemas, de forma que sea posible aplicar divide y vencerás? ¿En qué casos sí sería posible aplicar esta descomposición de forma sencilla? ¿Qué mejora se obtendría en tal caso?

7. Un programa para ordenar cadenas utiliza el método de ordenación por mezcla. La resolución para problemas de tamaño pequeño tarda un tiempo de $g(n) = 2n^2$, mientras que la mezcla requiere f(n) = 10n. Calcular cuál debería ser el tamaño del caso base para optimizar el tiempo de ejecución total t(n).

8. Diseñar un algoritmo para calcular el mayor y el segundo mayor elemento de un array de enteros, utilizando la técnica divide y vencerás. Calcular el número de comparaciones realizadas en el mejor y en el peor caso, suponiendo n potencia de 2. Comprueba si es posible aplicar los resultados obtenidos (en cuanto a órdenes de ejecución) a valores de n que no sean potencia de 2.

9. La figura de abajo representa el "helecho", una curva fractal que puede ser generada mediante una variedad de procedimientos. En su versión más simple, es posible utilizar un procedimiento de divide y vencerás para dibujarlo. Suponer que la cabecera del procedimiento es de la forma `Pinta_Helecho(x, y: ℕ; ang, alt: ℤ);` que pinta un helecho de altura **alt**, cuyo tallo principal tiene la base en **(x, y)** y tiene ángulo **ang** respecto a la horizontal. Escribir un procedimiento para dibujar el helecho, a partir de los parámetros anteriores, utilizando la técnica divide y vencerás.



10. En un algoritmo divide y vencerás, en lugar de aplicar el caso base cuando **n** es menor que cierto valor, se aplica la recurrencia siempre un número **r** de veces. Calcular cuál será el tiempo de ejecución, suponiendo que un problema de tamaño **n** es dividido en **a** problemas de tamaño **n/b**, siendo el tiempo del caso base $g(n) = n^q$, y el tiempo de la mezcla $f(n) = n^p$.

11. Sea T[1..n] un array ordenado formado por enteros diferentes, algunos de los cuales pueden ser negativos. Dar un algoritmo que pueda hallar un índice **i** tal que $1 \leq i \leq n$ y $T[i]=i$, siempre que este índice exista. El algoritmo debe estar en un tiempo de ejecución O(log n) en el peor caso. ¿En qué tipo de algoritmos lo clasificarías?

12. Dado un array de enteros E[1..N], se dice que un entero es un elemento mayoritario de E si aparece más de n/2 veces en E. Dar un algoritmo para calcular si un array E[1..N] contiene un elemento mayoritario. El algoritmo tiene que funcionar en tiempo lineal en el peor caso.

13. Considera la aplicación de la técnica divide y vencerás sobre el siguiente problema. Dado un array de **N** números enteros, buscamos la cadena de **n** celdas consecutivas

en este array cuya suma sea máxima (obviamente n<N). ¿Es conveniente aplicar divide y vencerás en este caso? ¿Cómo sería una resolución directa del problema?

14. Calcular el orden exacto Θ del número promedio de comparaciones que se hacen en la ordenación por mezcla dentro de la función de mezcla.

15. Construir un algoritmo para calcular el producto de dos matrices cuadradas triangulares superiores, siendo el número de filas y columnas potencia de 2. Se supondrá que disponemos de un procedimiento `Multip1(A, B, C)`, para multiplicar matrices (triangulares o no) en un tiempo $O(n^3)$. Obtener el tiempo de ejecución del algoritmo creado.
    Recordatorio: una matriz se dice que es triangular superior si todos los elementos que están por debajo de la diagonal principal valen 0. Se dice que es triangular inferior si todos los elementos encima de la diagonal principal valen 0. Ejemplo de matriz triangular superior:

1	3	5
0	2	4
0	0	2

16. Comentar cómo se implementaría con un método divide y vencerás similar al de Strassen la multiplicación de una matriz cuadrada triangular por una cuadrada no triangular. Habrá que indicar qué funciones sería necesario implementar y qué habría que hacer para ahorrar memoria en las llamadas recursivas. Estudiar también el tiempo de ejecución.

17. Se trata de resolver el problema de encontrar el cuadrado de unos más grande en una tabla cuadrada de bits (un array n x n).
    a) Programar un método directo para resolver el problema y dar una cota superior (no demasiado mala) de su tiempo de ejecución.
    b) Programar un método para resolver el problema por divide y vencerás y dar una cota superior (no demasiado mala) de su tiempo de ejecución.

18. Realizar un algoritmo que, utilizando la técnica de divide y vencerás, encuentre el mayor y menor elementos de un array. Estudiar el número de comparaciones y asignaciones de elementos del tipo que hay en el array.

19. Programar un método de ordenación de un array similar a la ordenación por mezcla, pero dividiendo el problema en 3 subproblemas (de tamaños parecidos) en lugar de en 2.

20. Suponer que trabajamos con enteros largos de tamaño **n**. Como vimos en clase, el algoritmo clásico de multiplicación es un $O(\mathbf{n}^2)$ y el método de Karatsuba y Ofman consigue una reducción del orden al aplicar divide y vencerás con 3 subproblemas de tamaño **n**/2, y tiempo de dividir y combinar en O(**n**).

Queremos diseñar otro algoritmo de divide y vencerás con un orden mejor que el de Karatsuba y Ofman, para lo cual se debería desarrollar otra descomposición recursiva. ¿Cómo tendría que ser la descomposición para conseguir la mejora? Indicar justificadamente por lo menos dos tipos de descomposiciones (es decir, el tamaño de los subproblemas y el número de estos) y el orden de complejidad que se obtendría con las mismas. Considerar que la división del problema y combinación son siempre O(**n**), y que no puede existir ninguna descomposición en **a** o menos subproblemas si el tamaño de estos es de **n/a**.

21. Un algoritmo de divide y vencerás descompone un problema de tamaño **n** en tres subproblemas de tamaño **n/2** y cuatro subproblemas de tamaño **n/4**. La división y combinación requieren $\mathbf{3n^2}$, y el caso base **n!** Otro algoritmo resuelve el mismo problema, también con divide y vencerás, pero en este caso usando un subproblema de tamaño **n-3** y dos de tamaño **n-2**. El tiempo de la división y combinación es despreciable, y el caso base es de orden constante. En ambos algoritmos el caso base se aplica con **n** menor que 5.
    a) Calcula el orden de complejidad de cada uno de los dos anteriores algoritmos. ¿Cuál es preferible en cuanto a orden de complejidad?
    b) Suponer que los dos algoritmos anteriores usan **k** bytes de memoria en cada llamada recursiva (para almacenar parámetros, variables locales, dirección de retorno, etc., en la pila), siendo **k** una constante. Calcular la utilización de memoria de los algoritmos. ¿Cuál es preferible respecto al uso de memoria?

22. Considerar el siguiente algoritmo recursivo basado en divide y vencerás.

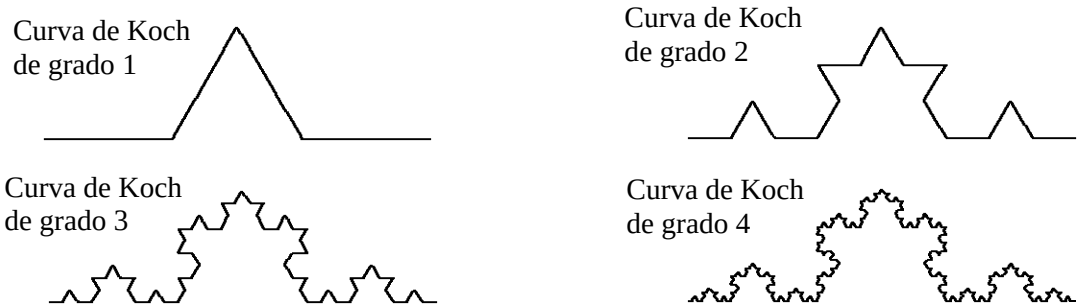
```
fun DV (n: ℤ) devuelve ℝ
variables m, k: ℤ;  d: ℝ
inicio
   d ← 1.0 + n
   si n>1 entonces
      m ← n/2
      d ← d + DV(m)
      si m>1 entonces
         k ← m/2
         d ← d + 4*DV(k)
         d ← d + 8*DV(k)
      finsi
   finsi
     devolver d
```

Se pide:

a) Estimar el orden de complejidad del algoritmo, usando la notación asintótica que creas más conveniente.
b) Calcular el valor devuelto por el algoritmo mediante una fórmula explícita, es decir, no recursiva. Si aparecen más de 3 constantes, se puede dejar indicado el sistema de ecuaciones que habría que resolver.
c) Estimar el orden exacto, Θ, del consumo de memoria del algoritmo.

23. Un problema se puede resolver usando divide y vencerás de dos formas distintas. En la primera forma se divide el problema de tamaño **n** en dos subproblemas de tamaño **n**/2, y la combinación tiene un coste **n**. Por la segunda forma se divide el problema en tres subproblemas de tamaño **n**/3, y la combinación tiene un coste 2**n**. En ambos métodos el tamaño del caso base es 1, y el coste de resolver el caso base y de comprobar si el problema es suficientemente pequeño y de dividir el problema también tiene coste 1.
    a) Calcular el tiempo de ejecución de los dos algoritmos, y determinar cuál de los dos es preferible en cuanto al tiempo de ejecución. ¿Y en cuanto al orden de complejidad?
    b) Estudiar la ocupación de memoria de los dos algoritmos, e indicar cuál es preferible en cuanto a ocupación de memoria.

24. La curva de Koch es una curva fractal que se construye de la siguiente forma. La curva de Koch de grado 0 es un segmento recto de longitud **m**. La curva de Koch de grado 1 se forma sustituyendo ese segmento por 4 segmentos contiguos, de longitud **m**/3, donde los dos centrales forman un ángulo de 60º (como se muestra en la figura de abajo). En general, la curva de Koch de grado **k** se forma sustituyendo cada segmento recto de la curva de Koch de grado **k**-1 por 4 segmentos de 1/3 de longitud del original, donde los dos centrales forman un ángulo de 60º.



a) Escribe un procedimiento para pintar la curva de Koch de grado **k** y longitud **m**, empezando en un punto (**x**, **y**). Indica la cabecera del procedimiento. Para pintar se deben usar las operaciones: `Colocar(x,y)`: coloca el puntero en la posición (x,y); `MoverA(x,y)`: pinta una línea desde la posición actual del puntero hasta (x,y), que pasa a ser la nueva posición actual; `Línea(long, ang)`: pinta una línea de longitud **long** y con ángulo **ang**, empezando en la posición actual, y el otro extremo de la línea pasa a ser la nueva posición actual del puntero.
b) Calcula el tiempo de ejecución y el orden de complejidad del procedimiento del anterior apartado. Se supone que el tiempo de dibujar una línea de longitud **n** es un O(**n**). ¿Cuánto mide la curva de Koch de grado **k**? Suponiendo que el algoritmo tarda 3.25 segundos para cierto **m** y **k**=12, estima de la forma más precisa posible cuánto tardará para **k**=18 y el mismo **m**.