Ejercicios Recursivos:

1. Diseñar un algoritmo recursivo para encontrar los números de Fibonacci. Posteriormente encontrar un algoritmo iterativo mediante la adaptación del algoritmo de la potencia entera. Los números de Fibonacci cumplen la propiedad $f_n = f_{n-1} + f_{n-2}, f_1 = 1, f_0 = 0$
2. Diseñar un algoritmo recursivo, con y sin memoria, y posteriormente un algoritmo iterativo que calcule los valores de la recurrencia $f_n = 2f_{n-1} + 3f_{n-2} - f_{n-3}, f_2 = 1, f_1 = 1, f_0 = 2$ mediante el cálculo de abajo arriba de los valores de la recurrencia.
3. Diseñar un algoritmo iterativo, con y sin memoria, y posteriormente encontrar un algoritmo iterativo que calcule los valores de la recurrencia $f_n = 4f_{n-1} + f_{n-2} + f_{n-3}, f_2 = 1, f_1 = 1, f_0 = 2$ mediante el cálculo de abajo arriba de los valores de la recurrencia.
4. Dar una definición recursiva y obtener un algoritmo recursivo para el cálculo de un número combinatorio basándose en las propiedades

$$\begin{cases} \binom{n}{0} = \binom{n}{n} = 1, & n \geq 0 \\ \binom{n}{1} = \binom{n}{n-1} = n, & n \geq 1 \\ \binom{n}{k} = \binom{n}{n-k}, & n \geq k \\ \binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}, & n > k \end{cases}$$

5. Dada la siguiente propiedad de la multiplicación entera

$$ab = \begin{cases} a + a(b-1), & b > 0 \\ 0, & b = 0 \end{cases}$$

   Encontrar una definición recursiva para la misma y posteriormente diseñar un algoritmo iterativo.
6. Dar una definición recursiva de las sucesivas potencias de 2.
7. Dar una definición recursiva de un número primo.

Árboles: Definiciones

1. **Árbol n-ario**: Es un tipo *Tree<E>* de la forma:

$$t = \begin{cases} create() \\ create(e) \\ create(e, t_0, t_1, , \dots, t_{n-1}) \end{cases}$$

   Con las operaciones adicionales:

$$e = t.getLabel()$$
$$nh = t.getNumChildren()$$
$$b_0 = t.isEmpty()$$
$$b_1 = t.isLeaf()$$
$$tr = t.getElement(i)$$

Dónde t, t_0, t_1, tr son árboles, *e* un elementos de tipo E y b_0, b_1 de tipo *Boolean*. Al primero lo llamamos árbol vacío, el segundo árbol hoja no vacío y al tercero árbol de n hijos. A $t_0, t_1, , \dots, t_{n-1}$ los llamamos hijos de t y a éste padre de aquellos. A e lo llamamos etiqueta o label. El árbol que no tiene padre lo llamamos raíz.

El tamaño de un árbol es el número de etiquetas que tiene.

Se define camino en un árbol a cualquier secuencia de árboles, $t_0, t_1, \dots, t_{r-1}$, tal que cada uno es padre del siguiente. La longitud del camino se define como el número de elementos de la secuencia menos uno $(r-1)$.

La altura de un árbol se define como la longitud del camino más largo que comienza en la raíz y termina en una hoja. Se puede calcular sumando uno a la altura del hijo que la tiene mayor. La altura de una hoja será de cero. La altura de un árbol se define como la altura de su raíz. La profundidad de un árbol se define como la longitud del camino que comienza en la raíz y termina en él. Se puede calcular como la profundidad de su padre más uno. La profundidad de la raíz es cero. A la profundidad de un árbol también se la denomina nivel del árbol en el árbol que lo contiene.

Diseñar algoritmos recursivos para calcular los siguientes objetivos:

1. Calcular el número de etiquetas del árbol (tamaño del árbol)
2. Suma de las etiquetas del árbol que son pares (asumiendo que son de tipo entero)
3. Calcular la altura de un árbol
4. Si contiene la etiqueta *a*
5. Profundidad de un vértice en un árbol
6. Comprobar si dos árboles son iguales
7. Obtener la copia de un árbol
8. Obtener la copia simétrica de un árbol
9. Convertir un árbol en una cadena de texto.
10. Convertir un árbol binario en una lista que contiene sus etiquetas en recorrido en preorden. En este recorrido se añade primero la etiqueta del árbol actual, luego las de su hijo izquierdo y por último las del derecho

11. Convertir un árbol binario en una lista que contiene sus etiquetas en recorrido en inorden. En este recorrido se añade primero las etiquetas de su hijo izquierdo, luego la etiqueta del árbol actual y por último las del derecho
12. Convertir un árbol binario en una lista que contiene sus etiquetas en recorrido en posorden. En este recorrido se añade primero las etiquetas de su hijo izquierdo, luego las del derecho y por último la etiqueta del árbol actual.
13. Convertir un árbol binario en una lista que contiene sus etiquetas en recorrido en anchura o por niveles. En este recorrido se añade primero la etiqueta del árbol actual, luego las de sus hijos, posteriormente los hijos de estos y así sucesivamente. Es decir, se añaden por niveles: 0, 1, …
14. Obtener una lista con todas las etiquetas del árbol. Discuta las posibles formas de hacerlo.

Más problemas recursivos

1. Buscar si el elemento e está contenido en lista ls.
    a. Asumir que la lista no está ordenada
    b. Asumir que la lista está ordenada
2. Encontrar el índice del elemento e en lista ls y si no lo contiene entonces devolver -1.
    a. Asumir que la lista no está ordenada
    b. Asumir que la lista está ordenada
3. El problema consiste en dada una lista no ordenada encontrar el elemento que ocuparía la posición *k-ésima* con respecto a un orden dado.
4. Se dispone de una lista no ordenada de canciones. Se quiere buscar la canción (o lista de canciones en caso de repetición) que ocuparía la posición k si estuvieran ordenadas según su duración (primero las canciones más cortas).
5. El problema consiste en dada una lista ordenarla con respecto a un orden dado. Para ello se parte la lista en dos mitades iguales, se ordena cada sublista y mezclan ordenadamente los resultados mediante al algoritmo de fusión de listas (algoritmo de *Mergesort*).
1. Un segundo algoritmo para ordenar listas es el *Quicksort*. Este algoritmo elige en primer lugar un elemento de la lista que llamaremos p (pivote). En segundo lugar, mediante el algoritmo de la bandera holandesa, reordena la lista en los menores al pivote, los iguales y los mayores. Por ultimo repite el procedimiento para los mayores y los menores al pivote.
6. Dada una matriz cuadrada de $n\ x\ n$ números enteros, siendo n una potencia de 2, decidir si la matriz cumple determinada propiedad. Una matriz cumple la propiedad si en valor en la casilla superior izquierda es menor al de la casilla inferior derecha y cada una de las cuatro submatrices cumplen la propiedad.

7. Dado un vector ordenado y rotado k veces diseñar un algoritmo $\Theta(\log n)$ que encuentre el elemento mayor del vector, suponiendo que no conocemos el valor de k.
8. Sea a un array de n elementos de tipo entero ordenados en orden estrictamente creciente. Encontrar un algoritmo de complejidad $\Theta(\log n)$ en el caso peor, para determinar la posición i tal que $a[i] = i$. (Se supone que dicha posición existe)
9. Dada una lista de palabras, ordenada alfabéticamente, se desea encontrar la posición de palabra dada P y si la palabra no está devolverá -1. Se quiere diseñar un algoritmo que divida el vector en tres partes y seguir buscando en el tercio correspondiente. Si la función de comparación de palabras tiene complejidad $\Theta(m)$, siendo m el tamaño de las palabras a comparar, hallar de complejidad del algoritmo en el caso peor.
10. Supongamos ahora que tenemos una secuencia de enteros, positivos y negativos. El problema que se trata de resolver es encontrar la sub-secuencia cuya suma sea máxima. Por ejemplo, si tenemos la secuencia 1, -2, **11, -4, 13**, -5, 2, 3, entonces la sub-secuencia de suma máxima es la sombreada, que suma 20.
11. En una lista de movimientos de una cuenta corriente se recogen las fechas y la cantidad en euros de los ingresos (positivos) y reintegros (negativos). Llamamos saldo parcial de un periodo de tiempo a la suma de ingresos y reintegros durante ese periodo. Queremos encontrar el saldo parcial máximo y su período de tiempo correspondiente