RECURSIVIDAD – ITERACIÓN

Hemos visto que la **recursividad** es una técnica **alternativa a la** iteración para la resolución de problemas. Por tanto, a la hora de diseñar un algoritmo habrá que decidir entre una alternativa u otra. Para tomar esta decisión se pueden tener en cuenta distintos criterios:

- En una primera etapa de diseño, generalmente se suele desear obtener **algoritmos claros y correctos**. Ya hemos visto casos en los que la **solución recursiva** es muy **clara** (por ejemplo cuando el problema viene definido recursivamente como es el caso de la función factorial o de la sucesión de Fibonacci). En cuanto a la **corrección** no vamos a entrar a estudiarla pero aquí aseguramos que la recursividad facilita la demostración de la corrección o no de un algoritmo.
- También se desea obtener **algoritmos eficientes** en tiempo de ejecución y, estudiando los ejemplos que hemos visto en este tema, podemos intuir que **las versiones iterativas de algoritmos son generalmente más eficientes que las recursivas**.

Por lo tanto, **¿cuándo elegir un algoritmo recursivo frente a otro iterativo?**. La respuesta es: cuando sea lo natural. Por ejemplo, hay veces en las que la dificultad conceptual de las soluciones iterativas no justifica la ganancia en tiempo de ejecución y, otras veces, como en el caso del quicksort, la solución recursiva da lugar a algoritmos muy eficientes. Lo ideal es utilizar recursividad cuando sea más claro y después en un segundo paso, transformar el algoritmo recursivo en uno iterativo de comportamiento equivalente (es decir, para los mismos datos de entrada se obtienen los mismos resultados). En este caso, la corrección de la versión recursiva asegura la corrección de la versión iterativa equivalente.

La transformación de un algoritmo recursivo en iterativo siempre puede hacerse. De hecho, existen **esquemas de transformación** de algoritmos recursivos en iterativos. Vamos a ver un caso particular de uno de estos esquemas de transformación.

Recordamos que un algoritmo tiene **recursividad lineal** si cada llamada genera a lo sumo una llamada recursiva (ej: factorial, NO Fibonacci). Además, un algoritmo recursivo lineal diremos que presenta **recursividad final** si la llamada recursiva es la última operación que se efectúa antes de terminar la ejecución del algoritmo (ej: búsqueda dicotómica recursiva, NO factorial)

Una forma de construir un algoritmo recursivo final a partir de uno recursivo lineal es aplicar **inmersión** en un problema más general. Este paso no es siempre sencillo.

Pasos para transformar un algoritmo recursivo final en otro iterativo

Esquema del algoritmo recursivo final:

```
subalgoritmo recursivoFinal ( x )  // x parámetros
principio
     si C(x) entonces  //condición que depende de los parámetros x
          A1(x)        //instrucciones caso trivial
     si_no
          A2(x)
          recursivoFinal(T(x)) // T(x) transformación de los
                               // parámetros de una llamada a otra
     fsi
fin
```

Pasos:

1) Reemplazar el condicional dentro del que se encuentra la llamada recursiva por un bucle.
2) La condición del caso trivial o base se considera condición de parada del bucle.
3) El cuerpo del bucle contendrá:
   - Sentencias que preceden a la llamada recursiva
   - La transformación de los parámetros del algoritmo (acercamiento a los casos triviales)
4) La solución al caso base se escribe al final del algoritmo iterativo, fuera del bucle.

Esquema del algoritmo iterativo:

```
subalgoritmo iterativo (x)
principio
          mientras que not C(x) hacer
               A2(x)
               x←T(x)
          fmq
          A1(x)
fin
```

Ejercicios :

1) A partir de la solución recursiva de la función factorial, obtener la solución iterativa.
2) A partir de la solución recursiva de la función potencia obtener la solución iterativa.