2. (a) Escribí el procedimiento "intercalar_cada" que recibe un arreglo a : **array**$[1..2^n]$ **of int** y un número natural i : **nat**; e intercala el segmento $a[1, 2^i]$ con $a[2^i + 1, 2 * 2^i]$, el segmento $a[2 * 2^i + 1, 3 * 2^i]$ con $a[3 * 2^i + 1, 4 * 2^i]$, etc. Cada uno de dichos segmentos se asumen ordenados. Por ejemplo, si el arreglo contiene los valores 3, 7, 1, 6, 1, 5, 3, 4 y se lo invoca con con $i = 1$ el algoritmo deberá devolver el arreglo 1, 3, 6, 7, 1, 3, 4, 5. Si se lo vuelve a invocar con este nuevo arreglo y con $i = 2$, devolverá 1, 1, 3, 3, 4, 5, 6, 7 que ya está completamente ordenado. El algoritmo asume que cada uno de estos segmentos está ordenado, y puede utilizar el procedimiento de intercalación dado en clase.

```
proc intercalar_cada(in/out a: array[1..2^n] of int, in i: nat)
    var int_start, int_mid, int_finish: nat

    int_start := 1
    int_finish := 2^(i+1)
    int_mid := (int_start + int_finish) div 2

    while int_finish ≤ 2^n do
        merge(a,int_start,int_mid,int_finish)
        int_start := int_finish + 1
        int_mid := int_mid + int_finish
        int_finish := int_finish + int_finish
    od
end proc
```

(b) Utilizar el algoritmo "intercalar_cada" para escribir una versión iterativa del algoritmo de ordenación por intercalación. La idea es que en vez de utilizar recursión, invoca al algoritmo del inciso anterior sucesivamente con $i = 0, 1, 2, 3$, etc.

```
proc ord_por_intercalacion_iterativo(in/out a: array[1..2^n] of T)
    for i := 0 to n do
        intercalar_cada(a,i)
    od
end proc
```