1. Traversal tree

Problema

Dados dos arreglos llamados *In* y *Pre* correspondientes al *Inorder* y *Preorder* de un arbol binario cualquiera, obtener el arreglo correspondiente al *Postorder*. Como precondición, el árbol no tiene elementos repetidos.

Idea del algoritmo

La manera de recorrer un árbol binario utilizando estos métodos es la siguiente:

- Inorder: Izquierdo - Medio - Derecho.
- Preorder: Medio - Izquierdo - Derecho.
- Postorder: Izquierdo - Derecho - Medio.

La primer posición del arreglo *Preorder* nos indica la raiz del árbol. Sea z la posición de la raiz en el arreglo *Inorder*, i, y j las posiciones que indican el comienzo y el final del árbol en el arreglo respectivamente, sabemos que:

- En el intervalo $[i, ..., z-1]$ del arreglo *Inorder* se encuentra el subárbol izquierdo, por la manera en la cual se recorre el árbol con esta técnica. En caso de que el intevarlo sea vacío, no existe el subárbol correspondiente.
- En el intervalo $[z+1, ..., j]$ del arreglo *Inorder* se encuentra el subárbol derecho, por la manera en la cual se recorre el árbol con esta técnica. En caso de que el intevarlo sea vacío, no existe el subárbol correspondiente.

Luego, el algoritmo consiste en llamar recursivamente a ambos subárboles (en caso de que existan) hasta llegar a una hoja. En ese caso (el caso base) se introduce la hoja al arreglo *Postorder*.

Noten como, al llamar primero recursivamente sobre el subárbol izquierdo, se inserta primero el subárbol izquierdo, luego el derecho, y por último la raiz, cumpliendo con el esquema que utiliza *Postorder* para recorrer el árbol.

Pseudocodigo 1 crearPostorder(in **arreglo(nat)** In, in **arreglo(nat)** **Pre** → res: **arreglo(nat)**) $\mathcal{O}(n)$

```
n ← Tam(In)-1
res ← arreglo(n)
crearPostorder2(In, Pre, res, 0, n, 0, 0)
```

Pseudocodigo 2

crearPostorder2(in **arreglo(nat)** In, in **arreglo(nat)** Pre, out **arreglo(nat)** Post, in/out **nat** i, in/out **nat** j, in/out **nat** pr, in/out **nat** po) → res: **arreglo(nat)**	$\mathcal{O}(n^2)$
if $i == j$ **then**	$\mathcal{O}(1)$
Post[po] ← In[i]	$\mathcal{O}(1)$
pr ← pr + 1	$\mathcal{O}(1)$
po ← po + 1	$\mathcal{O}(1)$
else	
int raiz ← Pre[pr]	$\mathcal{O}(1)$
pr ← pr + 1	$\mathcal{O}(1)$
z ← i	$\mathcal{O}(1)$
while In[z] ≠ raiz **do**	$\mathcal{O}(n)$
z ← z + 1	$\mathcal{O}(1)$
end while	
crearPostorder2(In, Pre, Post, i, z-1, pr)	$\mathcal{O}(1)$
crearPostorder2(In, Pre, Post, z+1, j, pr)	$\mathcal{O}(1)$
Post[po] ← raiz	$\mathcal{O}(1)$
po ← po + 1	$\mathcal{O}(1)$
end if	

En este caso no podemos utilizar el teorema maestro ya que no podemos garantizar que cuando llamamos recursivamente sobre el subárbol izquierdo y el subárbol derecho, estos tengan el mismo tamaño. Todas las operaciones son $\mathcal{O}(1)$ (incluido el llamado recursivo) En cada caso recursivo buscamos la raiz en el arreglo *In*, con complejidad lineal. Entonces, la complejidad está dada por la cantidad de veces que se ejecuta el algoritmo para un árbol de n nodos la cual es n, multiplicada por el costo de cada caso recursivo, que también es n. Esto sucede porque tanto el caso base como el recursivo recorren un único nodo (y nunca se repite uno ya visitado). Por ende la complejidad temporal del algoritmo es $\mathcal{O}(n^2)$. El algoritmo tiene complejidad $\mathcal{O}(nlogn)$ cuando el árbol es balanceado y $\mathcal{O}(n)$ cuando es degenerado a derecha.