# *INSTITUTO TECNOLOGICO SUPERIOR DE*

# *CHICONTEPEC*

***INGENIERIA EN:*** *SISTEMAS COMPUTACIONALES.*

***MATERIA:*** *PROGRAMACION LOGICA Y FUNCIONAL.*

***TEMA:*** *ARBOLES.*

***DOCENTE:*** *ING. EFREN FLORES CRUZ.*

***ALUMNO:*** *JOSE RICARDO MIGUELES GUERRA.*

***GRUPO:*** *“ISC-8”*

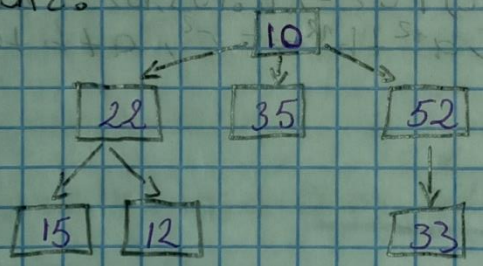
***FECHA DE ENTREGA:*** 02 DE ABRIL 2020.

Tema: Arboles generales

Un arbol es una estructura no lineal acíclica utilizada para organizar información de forma eficiente.

La definición es recursiva:

- Un árbol es una colección de valores $\{V_1, V_2 \ldots V_n\}$ tales que
* Si n = 0 el árbol se dice vacío.
* En otro caso, existe un valor destacado que se denomina raíz (p.e. $V_1$), y los demás elementos forman parte de colecciones disjuntas que a su vez son árboles. Estos árboles se llaman subárboles del raíz.




Las estructuras tipo árbol se usan principalmente para representar datos con una relación jerárquica entre sus elementos, como árboles genealógicos, tablas etc.

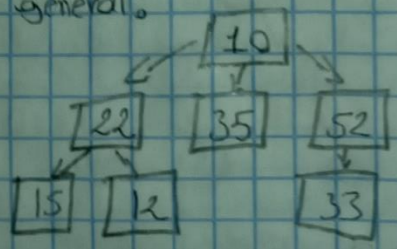
La terminología de los árboles se realiza con las típicas notaciones de las relaciones familiares en los árboles genealógicos: Padre, hijo, hermano, ascendente, descente etc.

Tema: Algunas definiciones

- Nodo, son los elementos del árbol.
- Raíz del árbol: todos los árboles que no están vacíos tienen un único nodo raíz. Todos los demás elementos o nodos se derivan o descienden de él.
- Nodo hoja es aquel nodo que no contiene ningún subárbol.
- Tamaño de un árbol es su número de nodos.
- A cada nodo que no es hoja se le asocia uno o varios sub arboles llamados descendientes o hijos.
- De igual forma, cada nodo tiene asociado un antecesor o ascendiente llamado Padre.
- Todos los nodos tienen un solo Padre excepto el raíz que no tiene Padre.
- Cada nodo tiene asociado un número de nivel que se determina por la longitud del camino desde el raíz al nodo específico.
- La altura o Profundidad de un árbol es el nivel más profundo más uno.

Ejemplo

Para ilustrar las definiciones se consideran el siguiente árbol general:

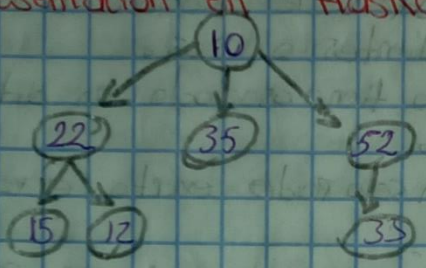



Por ejemplo en la figura:

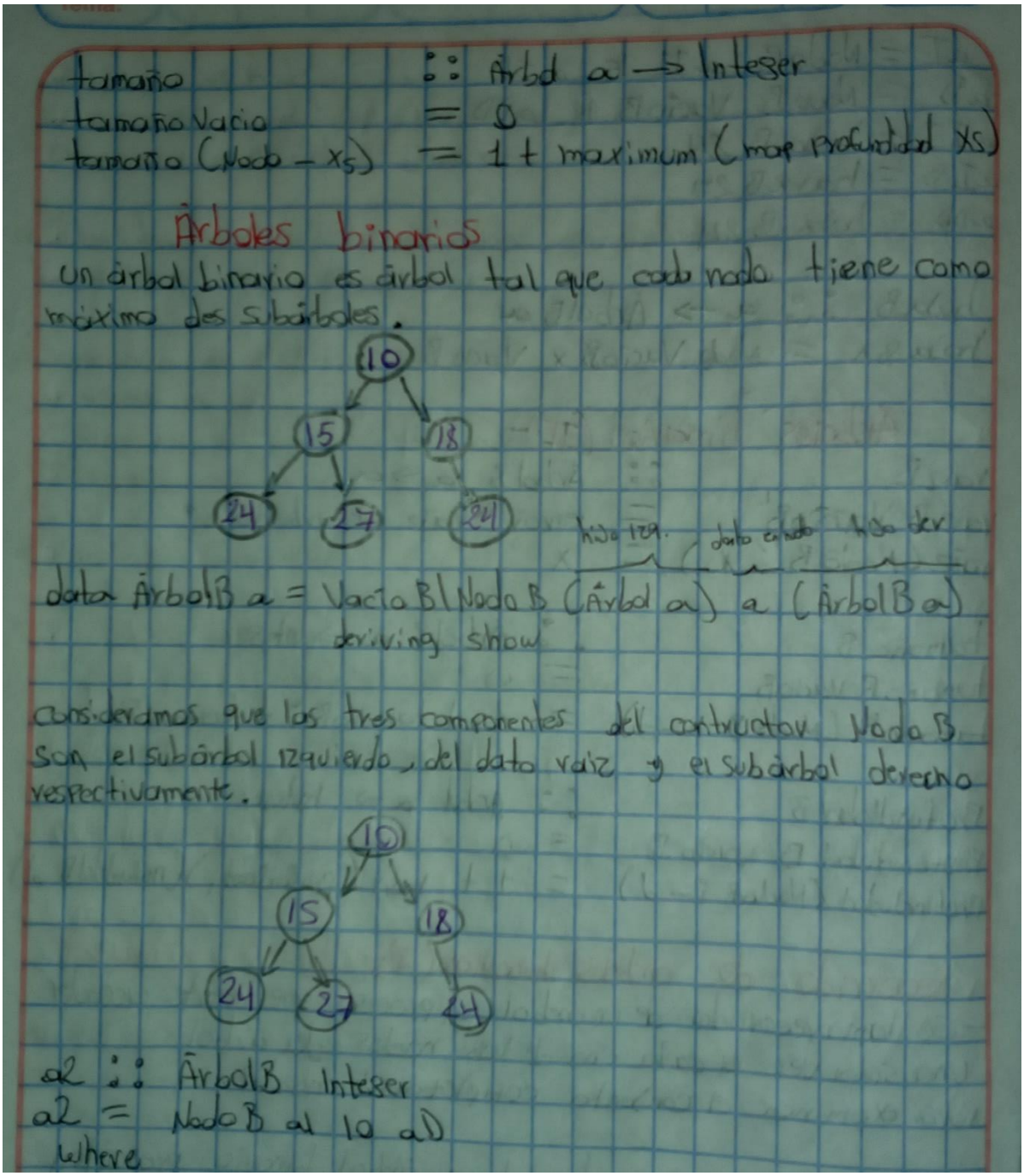
* Raíz: 10
* Nodos: 10, 22, 35, 52, 15, 12, 33
* Tamaño: 7

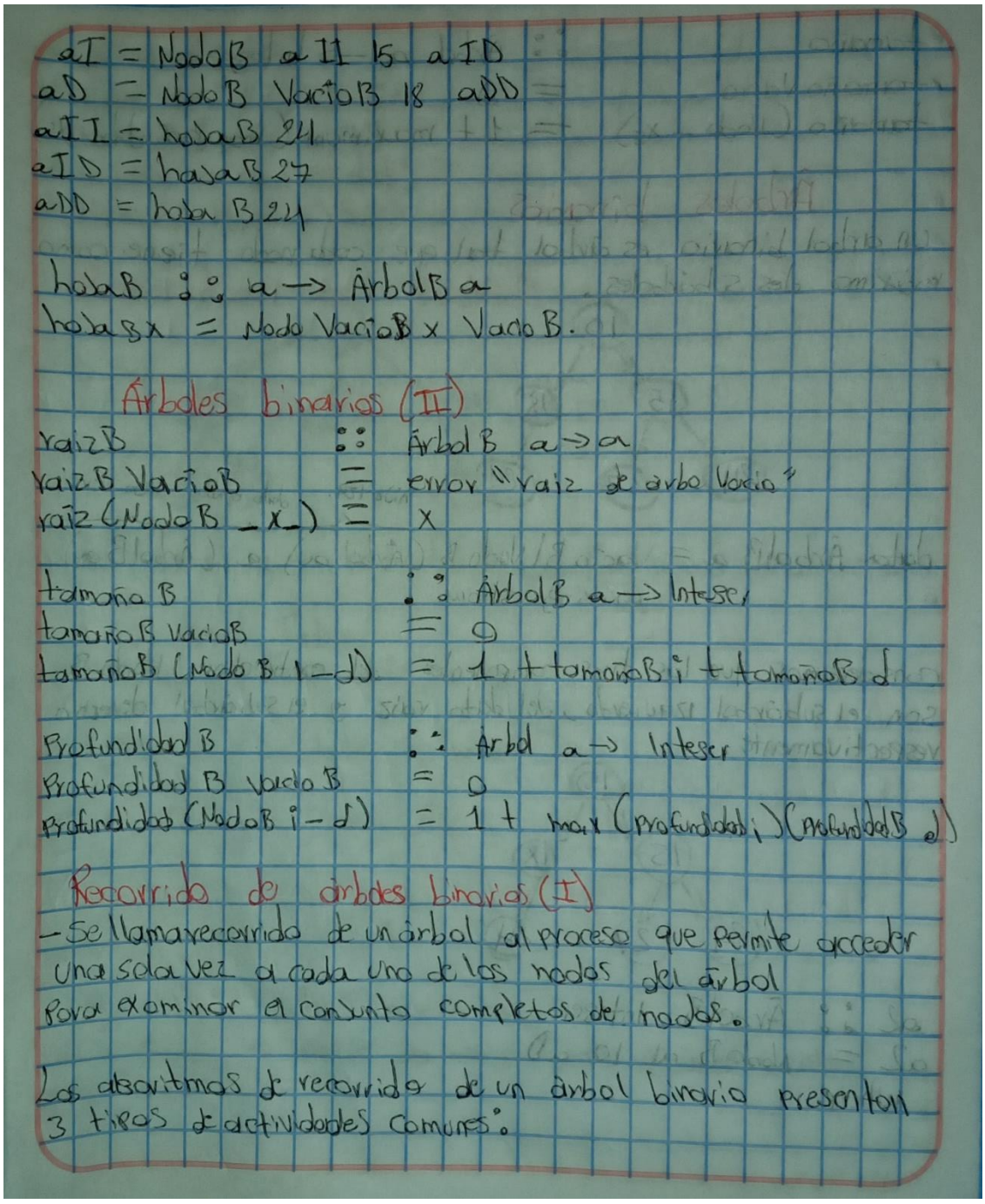
* Nivel 0 : 10
* Nivel 1 : 22, 35, 52
* Nivel 2 : 15, 12, 33
* Altura o profundidad : 3
* Hojas : 15, 12, 35, 33

## Representación en Haskell




```
data Árbol a = Vacio | Nodo a [Arbol a] deriving show
                       raiz   hijos

a1 :: Árbol Integer
a1 = Nodo 10 [a11, a12, a13]
  where
a11 = Nodo 22 [hoja 15, hoja 12]
a12 = hoja 35
a13 = Nodo 52 [hoja 33]

hoja :: a -> Árbol a
hoja x = Nodo x []

raiz :: Árbol a -> a
raiz Vacio = error a -> a "raiz de árbol Vacio"
raiz (Nodo x _) = x
```


tamaño :: Árbol a → Integer
tamañoVacio = 0
tamaño (Nodo _ xs) = 1 + maximum (map profundidad xs)
Árboles binarios
un árbol binario es árbol tal que cada nodo tiene como máximo dos subárboles.
10
15
18
24
27
24
hijo izq.
dato dado
hijo der
data ÁrbolB a = VaciaB | NodoB (Árbol a) a (ÁrbolB a)
deriving show
consideramos que los tres componentes del contructor NodoB son el subárbol izquierdo, del dato raíz y el subárbol derecho respectivamente.
10
15
18
24
27
24
a2 :: ÁrbolB Integer
a2 = NodoB a1 10 a0
where



```
aI = NodoB a II 15 a ID
aD = NodoB VacioB 18 aDD
aII = hojaB 24
aID = hojaB 27
aDD = hojaB 24

hojaB :: a -> ÁrbolB a
hojaB x = NodoVacioB x VacioB.
```

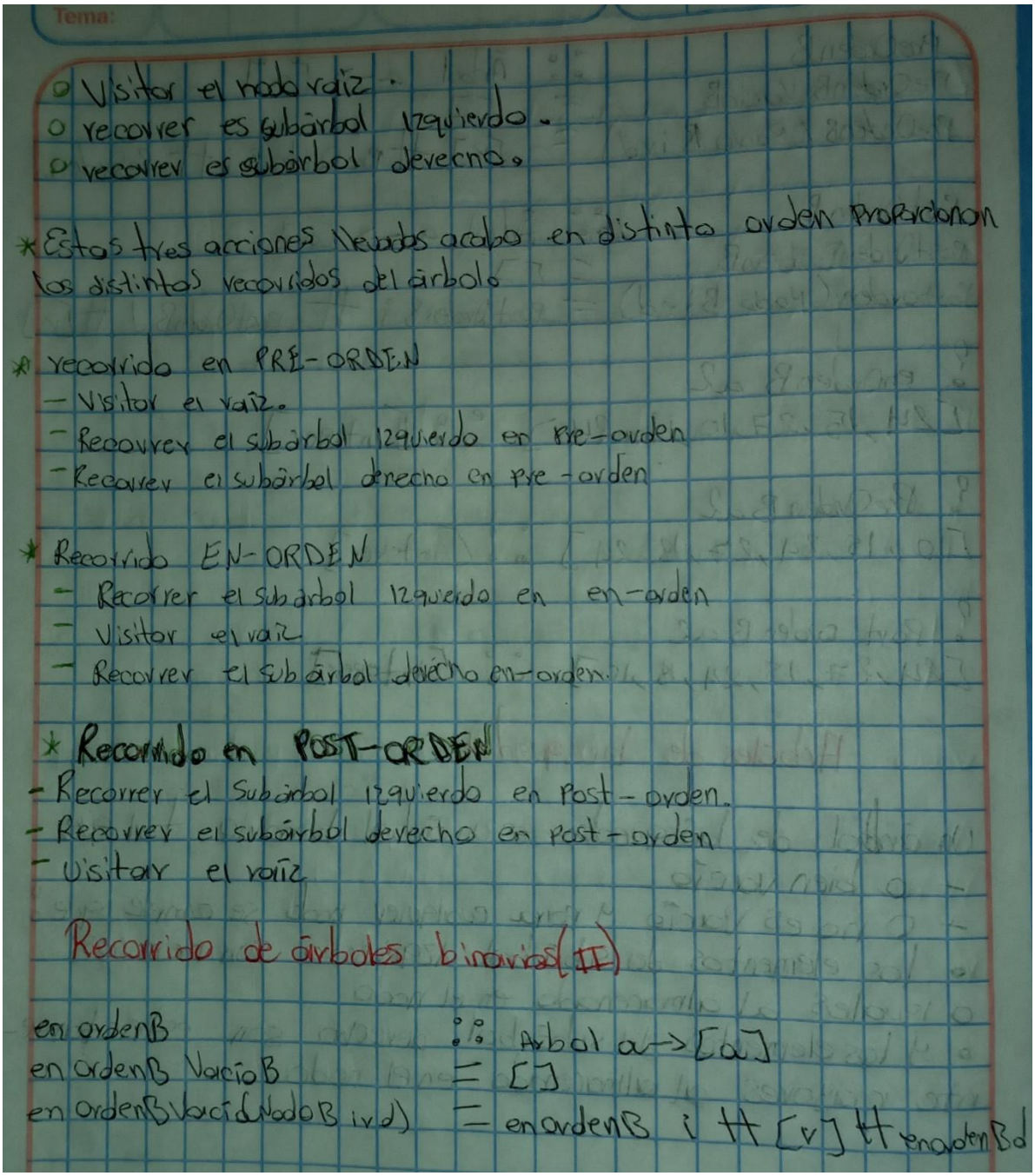
## Árboles binarios (II)

```
raizB            :: ÁrbolB a -> a
raizB VacioB     = error "raiz de arbol Vacio"
raiz (NodoB _ x _) = x

tamañoB          :: ÁrbolB a -> Integer
tamañoB VacioB   = 0
tamañoB (NodoB i _ d) = 1 + tamañoB i + tamañoB d

Profundidad B    :: Árbol a -> Integer
Profundidad B VacioB = 0
Profundidad (NodoB i _ d) = 1 + max (Profundidad i) (Profundidad d)
```

## Recorrido de árboles binarios (I)

- Se llama recorrido de un árbol al proceso que permite acceder una sola vez a cada uno de los nodos del árbol para examinar el conjunto completos de nodos.

Los algoritmos de recorrido de un árbol binario presentan 3 tipos de actividades comunes:



Tema:

- Visitar el nodo raiz.
- recorrer es subarbol izquierdo.
- recorrer es subarbol derecho.

* Estos tres acciones llevados acabo en distinta orden proporcionan los distintos recorridos del arbol.

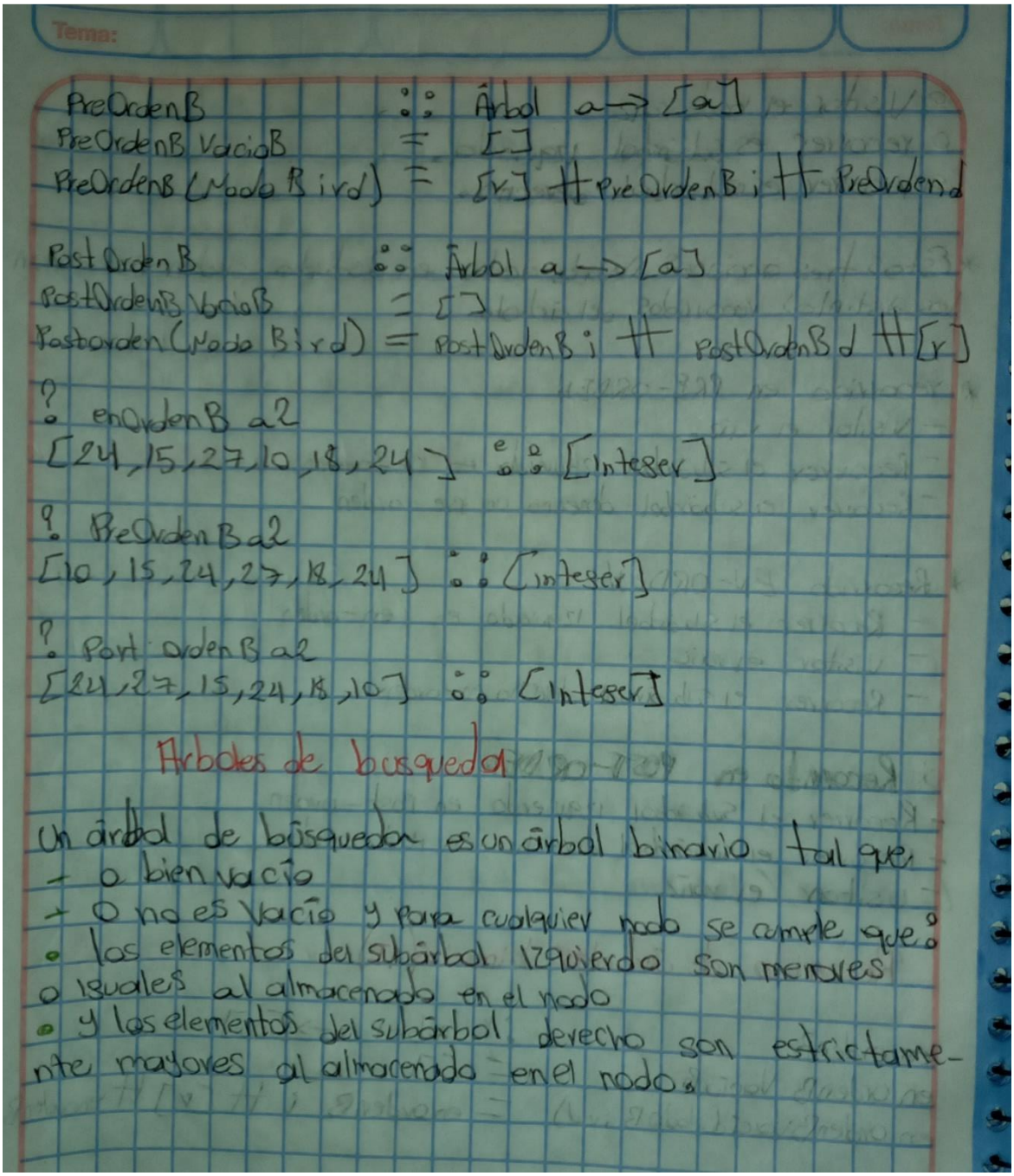
* recorrido en PRE-ORDEN
  - Visitar el raiz.
  - Recorrer el subarbol izquierdo en pre-orden
  - Recorrer el subarbol derecho en pre-orden

* Recorrido EN-ORDEN
  - Recorrer el subarbol izquierdo en en-orden
  - Visitar el raiz
  - Recorrer el subarbol derecho en-orden.

* Recorrido en POST-ORDEN
- Recorrer el Subarbol izquierdo en Post-orden.
- Recorrer el subarbol derecho en Post-orden
- Visitar el raiz

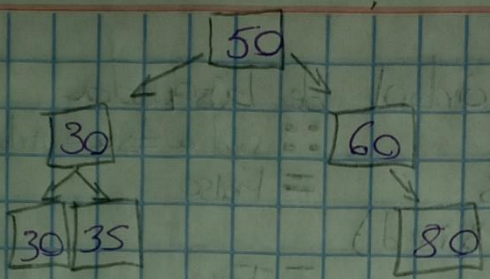
Recorrido de arboles binarios(II)

```
enordenB                    :: Arbol a -> [a]
enordenB VacioB             = []
enordenB (NodoB i v d)      = enordenB i ++ [v] ++ enordenB d
```


Tema:
PreOrdenB :: Árbol a -> [a]
PreOrdenB VacioB = []
PreOrdenB (NodoB i r d) = [r] ++ PreOrdenB i ++ PreOrden d
PostOrdenB :: Árbol a -> [a]
PostOrdenB VacioB = []
PostOrden (NodoB i r d) = PostOrdenB i ++ postOrdenB d ++ [r]
? enOrdenB a2
[24,15,27,10,18,24] :: [integer]
? PreOrdenB a2
[10,15,24,27,18,24] :: [integer]
? PostOrdenB a2
[24,27,15,24,18,10] :: [integer]
Arboles de busqueda
Un árbol de búsqueda es un árbol binario tal que:
- o bien vacío
- o no es vacío y para cualquier nodo se cumple que:
• los elementos del subárbol izquierdo son menores o iguales al almacenado en el nodo
• y los elementos del subárbol derecho son estrictamente mayores al almacenado en el nodo.

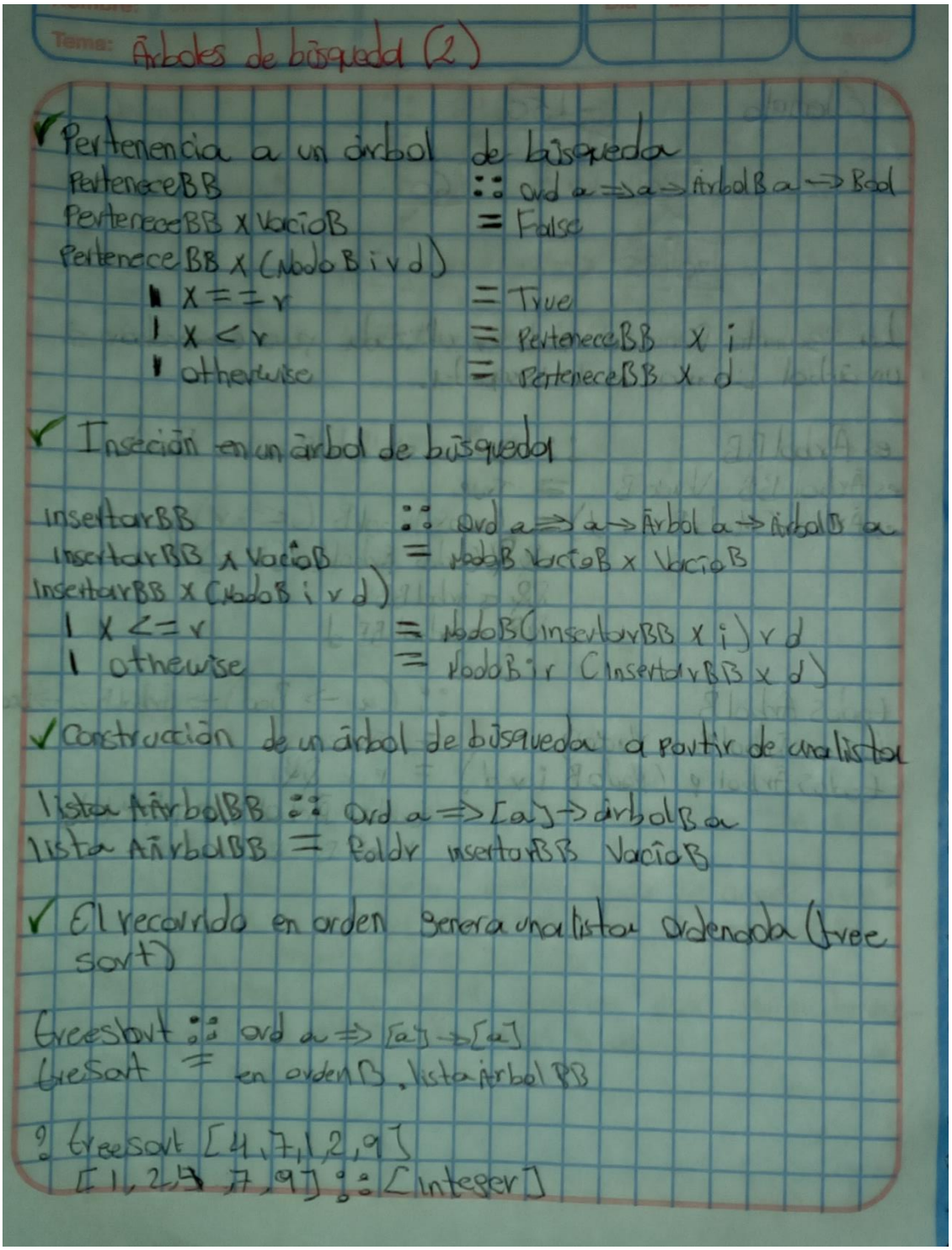
Tema:

Ejemplo




La siguiente función puede ser utilizada para comprobar si un árbol binario es de búsqueda.

```
esÁrbolBB :: Ord a => ÁrbolB a -> Bool
esÁrbolBB VacioB = True
esÁrbol (NodoB i r d) = todosÁrbolB (<= r) i
                        && todosÁrbolB (> r) d
                        && esÁrbolBB i
                        && esÁrbolBB d

todosÁrbolB :: (a -> Bool) -> ÁrbolB a -> Bool
todosÁrbolB p VacioB = True
todosÁrbol p (NodoB i r d) = p r &&
                             todosÁrbolB p i && todos
                             ÁrbolB p d
```


Tema: Árboles de búsqueda (2)
✓ Pertenencia a un árbol de búsqueda
perteneceBB :: Ord a => a -> ArbolB a -> Bool
perteneceBB x VacioB = False
perteneceBB x (NodoB i v d)
| x == v = True
| x < v = perteneceBB x i
| otherwise = perteneceBB x d
✓ Inserción en un árbol de búsqueda
insertarBB :: Ord a => a -> ArbolB a -> ArbolB a
insertarBB x VacioB = NodoB VacioB x VacioB
insertarBB x (NodoB i v d)
| x <= v = NodoB (insertarBB x i) v d
| otherwise = NodoB i v (insertarBB x d)
✓ Construcción de un árbol de búsqueda a partir de una lista
listaArbolBB :: Ord a => [a] -> ArbolB a
listaArbolBB = foldr insertarBB VacioB
✓ El recorrido en orden genera una lista ordenada (tree sort)
treesort :: Ord a => [a] -> [a]
treeSort = enOrdenB . listaArbolBB
? treeSort [4,7,1,2,9]
[1,2,4,7,9] :: [Integer]