INSTITUTO TECNOLÓGICO SUPERIOR DE CHICONTEPEC

INGENIERÍA EN SISTEMAS COMPUTACIONALES

Producto: Resumen de Árboles.

Asignatura: Programación Lógica y Funcional.

Docente: Ing. Efrén Flores Cruz.

Estudiante: Manuel Zúñiga Hernández.

Semestre: Octavo

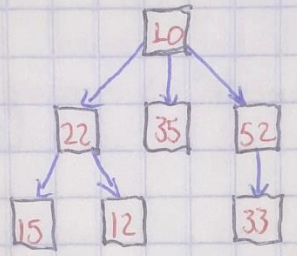
Chicontepec, Veracruz, a 29 de Marzo 2020

Arboles generales.

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

Un árbol es una colección de valores $\{v_1, v_2, \ldots v_n\}$ tales que:

* Si $n = 0$ el árbol se dice vacío.
* Otro caso, existe un valor destacado que se denomina raiz (p.e. v_1), y los demas elementos forman parte de colecciones disjuntas que a su vez son árboles. Estos se llaman subárboles del raíz.



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

La terminología de los árboles se realizan con las típicas notaciones de las relaciones familiares en los árboles geneológicos: padre, hijo, hermano, ascendente, descendente, etc.

Algunas definiciones.

* Nodo, son los elementos del árbol.
* Raiz del árbol: Todos los árboles que no están vacios tienen un único nodo raiz.

* Nodo hoja, es aquel nodo que no contiene ningun subárbol.
* Tamaño de un árbol es su número de nodos.
* A cada nodo que no es hoja se le asocia uno o varios subárboles 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 denomina por la longitud del camino desde el raíz nodo especifico.
* La altura o profundidad de un árbol es el nivel más profundo más uno.

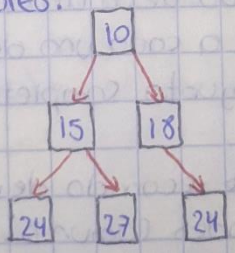
Representación en Haskell.

data Árbol a = Vacio | Nodo a [Árbol a] deriving show

a: raiz, [Árbol a]: hijos.

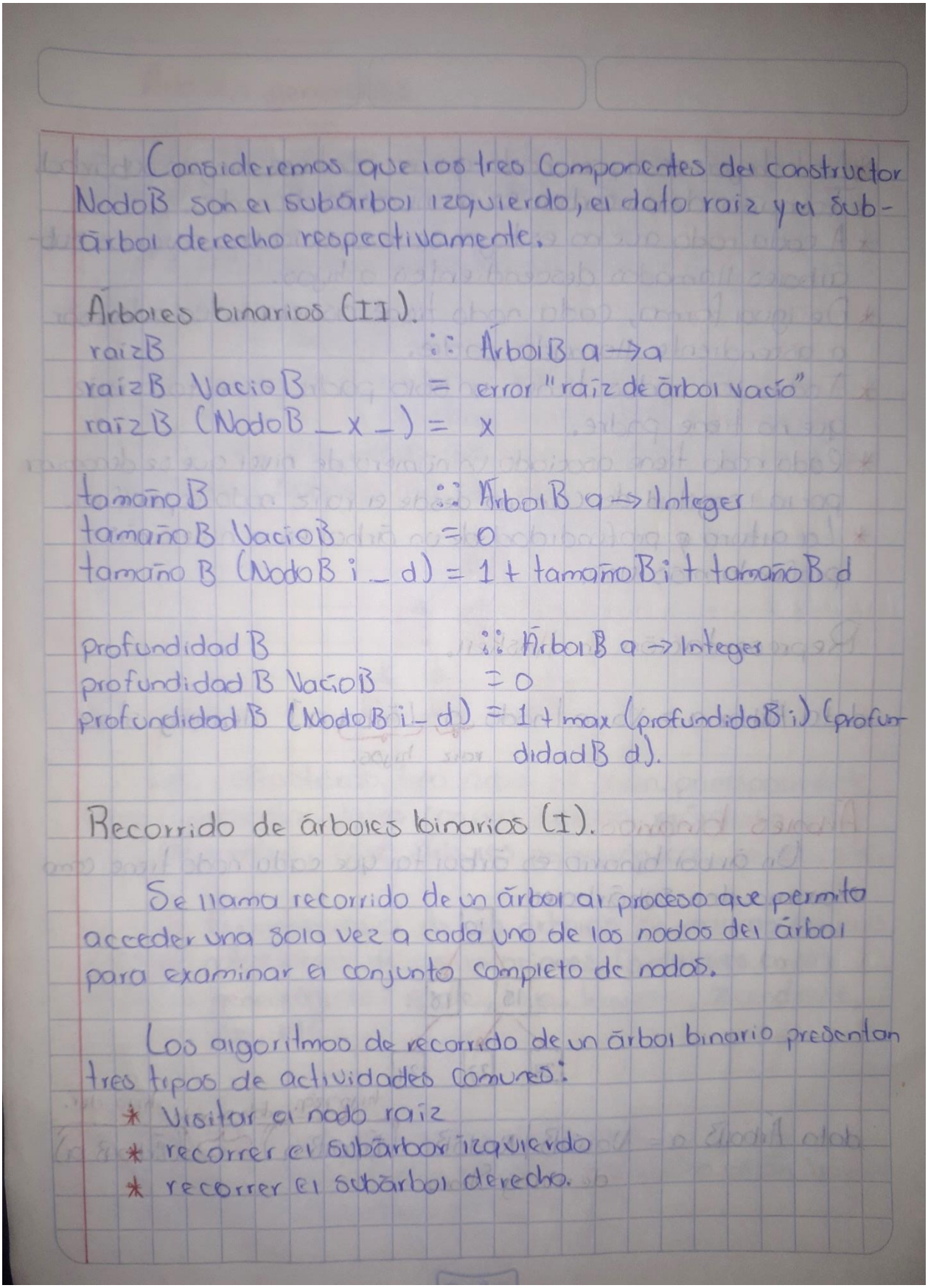
Árboles binarios.

Un árbol binario es árbol tal que cada nodo tiene como maximo dos subárboles.



data ÁrbolB a = VacioB | NodoB (ÁrbolB a) a (ÁrbolB a)
deriving Show

hijo izq. dato en nodo hijo der.



Consideremos que los tres Componentes del constructor NodoB son el subárbol izquierdo, el dato raiz y el subárbol derecho respectivamente.

Árboles binarios (II).

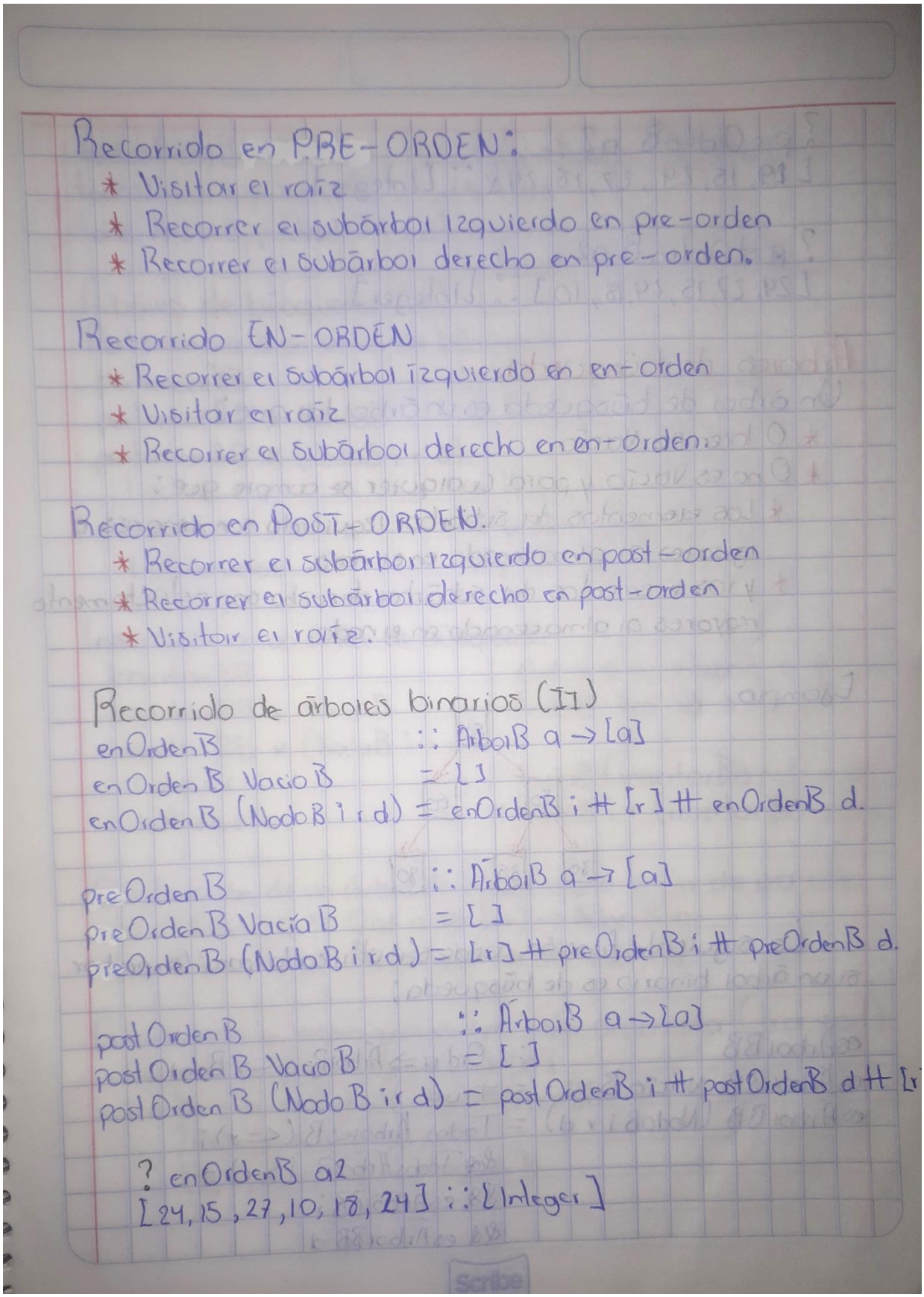
```
raizB                  :: ArbolB a -> a
raizB VacioB           = error "raiz de árbol vacío"
raizB (NodoB _ x _)    = x

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

profundidadB                  :: ArbolB a -> Integer
profundidadB VacioB           = 0
profundidadB (NodoB i _ d)    = 1 + max (profundidadB i) (profun
                                didadB 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 completo de nodos.

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

* Visitar el nodo raiz
* recorrer el subárbol izquierdo
* recorrer el subárbol derecho.



Recorrido en PRE-ORDEN:
* Visitar el raíz
* Recorrer el subárbol izquierdo en pre-orden
* Recorrer el subárbol derecho en pre-orden.

Recorrido EN-ORDEN
* Recorrer el subárbol izquierdo en en-orden
* Visitar el raíz
* Recorrer el subárbol derecho en en-orden.

Recorrido en POST-ORDEN.
* Recorrer el subárbol izquierdo en post-orden
* Recorrer el subárbol derecho en post-orden
* Visitar el raíz.

Recorrido de árboles binarios (II)

```
enOrdenB                    :: ArbolB a -> [a]
enOrdenB VacioB             = []
enOrdenB (NodoB i r d) = enOrdenB i ++ [r] ++ enOrdenB d.

preOrdenB                   :: ArbolB a -> [a]
preOrdenB Vacia B           = []
preOrdenB (NodoB i r d) = [r] ++ preOrdenB i ++ preOrdenB d.

postOrdenB                  :: ArbolB a -> [a]
postOrdenB VacioB           = []
postOrdenB (NodoB i r d) = postOrdenB i ++ postOrdenB d ++ [r

? enOrdenB a2
[24,15,27,10,18,24] :: [Integer]
```

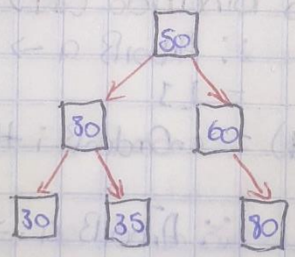
```
? preOrdenB a2
[10, 15, 24, 27, 18, 24] :: [Integer]

? postOrdenB a2
[24, 27, 18, 24, 18, 10] :: [Integer]
```

Árboles de búsqueda.

Un árbol de búsqueda es un árbol tal que:

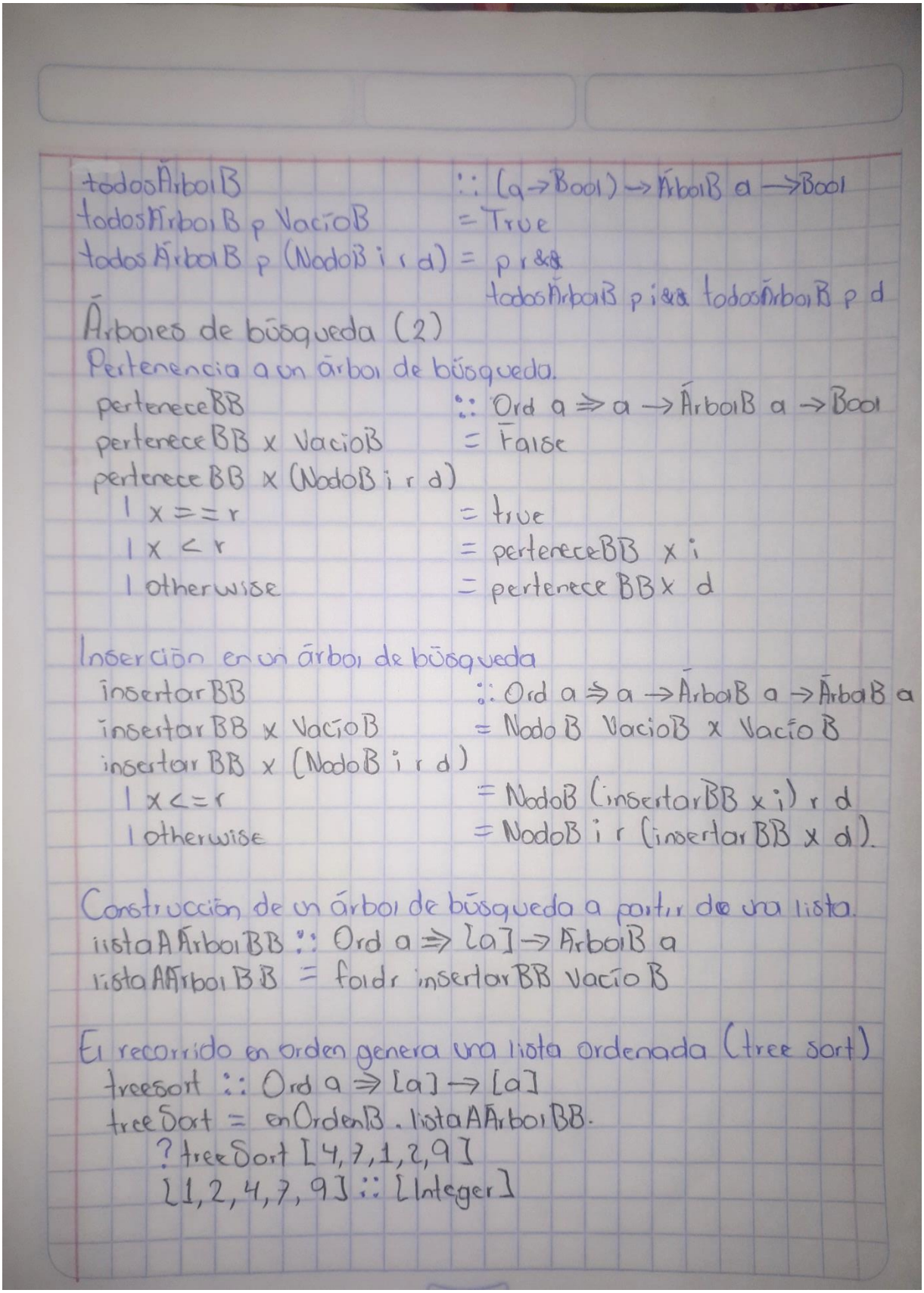
* O bien es vacío
* O no es vacío y para cualquier se cumple que:
  * Los elementos del subárbol izquierdo son menores o iguales al almacenado en el nodo.
  * y los elementos del subárbol derecho son estrictamente mayores al almacenado en el nodo.

Ejemplo



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

```
esArbolBB                    :: Ord a => ArbolB a -> Bool
esArbolBB vacioB             = True
esArbolBB (NodoB i r d)      = TodosArbolesB (<= r) i
                               && TodosArbolB (> r) d
                               && esArbolBB i
                               && esArbolBB d
```



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

Árboles de búsqueda (2)

Pertenencia a un árbol de búsqueda.

```
perteneceBB :: Ord a => a -> ÁrbolB a -> Bool
perteneceBB x VacioB = False
perteneceBB x (NodoB i r d)
  | x == r = true
  | x < r  = perteneceBB x i
  | otherwise = perteneceBB x d
```

Inserción en un árbol de búsqueda

```
insertarBB :: Ord a => a -> ÁrbolB a -> ÁrbaB a
insertarBB x VacioB = NodoB VacioB x VacioB
insertarBB x (NodoB i r d)
  | x <= r = NodoB (insertarBB x i) r d
  | otherwise = NodoB i r (insertarBB x d).
```

Construcción de un árbol de búsqueda a partir de una lista.

```
listaAÁrbolBB :: Ord a => [a] -> ÁrbolB a
listaAÁrbolBB = foldr insertarBB VacíoB
```

El recorrido en orden genera una lista ordenada (tree sort)

```
treesort :: Ord a => [a] -> [a]
treeSort = enOrdenB . listaAÁrbolBB.
   ?treeSort [4,7,1,2,9]
   [1,2,4,7,9] :: [Integer]
```