3.1) si se puede, pero no se podría manejar los nodos solo con los nombres, se tendría que poner alguna clase de número que el usuario también sepa cuál es, y de resto tendría las mismas funciones que el árbol binario de búsqueda, aunque si solo se quiere hacer con el nombre del familiar del árbol, es imposible que llegue a ser logarítmica en el peor de los casos, ya que tendría que pasar por todos los nodos para poder saber si la persona que está buscando está o no en el árbol, ya que como solo tienen los nombre y nada más, no existe nada que sea mayor que nada, y ningún nombre tendría correlación con otro, por lo tanto tampoco se puede con orden alfabético. Por lo que se tendría que pasar por todos los nodos examinándolos.

3.2) lo que hace es tomar los datos que el usuario ingresa en la consola y los guarda en una lista, después toma esa lista y crea un árbol, como el usuario ingresa los números en pre orden, el primer número es la raíz, y va ingresando los números en el orden de la lista, y al hacer esto, creamos el árbol que le corresponde, y después tomamos ese árbol y le sacamos el post orden como lo hacemos para cualquier árbol binario de búsqueda.

3.3)

```
class Nodo { // C0
  public Nodo(int n) {  //C1
            dato = n;// C2
  }
  public int dato;  // C3
  public Nodo izq; //C4
  public Nodo der; //C5
   }
```

O(1), ya que esta clase por sí sola no depende de una cantidad, sino de un solo número, y pude hace todo en tiempo constante .

```
class Arbol { //C0
        public Nodo raiz; //C1
        private void insertar(Nodo nodo, int n){   //C2
           if (nodo.dato == n) //C3
              return ; //C4
           if (n > nodo.dato) // +C5
              if (nodo.der == null) // C6
                 nodo.der = new Nodo(n); // C7
              else // C8
                 insertar(nodo.der, n); // T((n/2)-1) + C8
           else //
              if (nodo.izq == null) //C9
                 nodo.izq = new Nodo(n); //C10
              else // C11
                 insertar(nodo.izq, n);   // T((n/2)-1) + C12
        }
        public void insertar(int n){ //C13
            if (raiz == null) //C14
              raiz = new Nodo(n); / /C15
            else // C16
              insertar(raiz, n); //C17
        }
        public void postOrder(Nodo raiz) { //C18
           if(raiz !=  null) { //C19
              postOrder(raiz.izq); // T(n-1) +C20
              postOrder(raiz.der); // T(n-1) +C21
              if(raiz.dato != 0) //C22
              System.out.println(raiz.dato); //T(n-1) +C23
```

```
            }
        }
    }
```

$O(\log_2 n)$ para insertar, ya que solamente va a insertar eliminando la mitad, y después la mitad de eso, y después la mitad de eso.

O(n) para el postOrder, ya que tiene que pasar por todos los nodos n para poder construir el orden de impresión.

Siendo n el número de nodos del árbol.

```
import java.util.*; //C1

public class Postorden //C2

{

   List<Integer> numeros = new ArrayList<Integer>(); //C3


   public Postorden() //C4

   {

      Scanner reader = new Scanner(System.in); //C5

      int numero = 0; //C6


      System.out.println("Introduce números. El cero para salir"); //C7


      do {  // T(n)+C8

         try {  // T(n)+C9

            numero = reader.nextInt(); // T(n)+C10

            numeros.add(numero); // T(n)+C11

         } catch (InputMismatchException ime){ // T(n)+C12

            System.out.println("¡Cuidado! Solo puedes insertar números. "); // T(n)+C13

            reader.next(); // T(n)+C14

         }

      } while (numero!=0); // T(n)+C15

   }
```

```
    public void convertir() //C16
    {
        Arbol Punto2 = new Arbol(); //C19
        for(int i = 0; i<numeros.size();i++){ //T(n)+C20
            Punto2.insertar(numeros.get(i)); // T(n)+C21
        }
        Punto2.postOrder(Punto2.raiz); //C22
    }
}
```

O(n) Para introducir los números se demora un tiempo constante para cada número, pero se demora n ya que el usuario pone tantos números como él quiere.

$O(n\log_2 n)$ para convertir, ya que tiene que analizar cada valor para poderlo poner en su posición respectiva que le da la n, y $\log_2 n$ es porque para insertarlo en el árbol tiene que hacer la función insertar de la clase árbol.

Siendo n el número de datos que ingreso el usuario.

4)

4.1) a)altura(node.izq)
b) altura(node.izq)

4.2) c

4.3) a)false
b)a.data
c)a.der , suma - a.data
d) a.izq , suma - a.data

4.4)
4.4.1) a
4.4.2) a
4.4.3) d
4.4.4) a

4.5) a)p.data ==toInsert
b)p.data > toInsert

4.6)
4.6.1) d
4.6.2) a) 0
b) == 0

7.1) 1
7.2) 3