Jorge Del Barco Garza
A01284234

13 Oct 2021

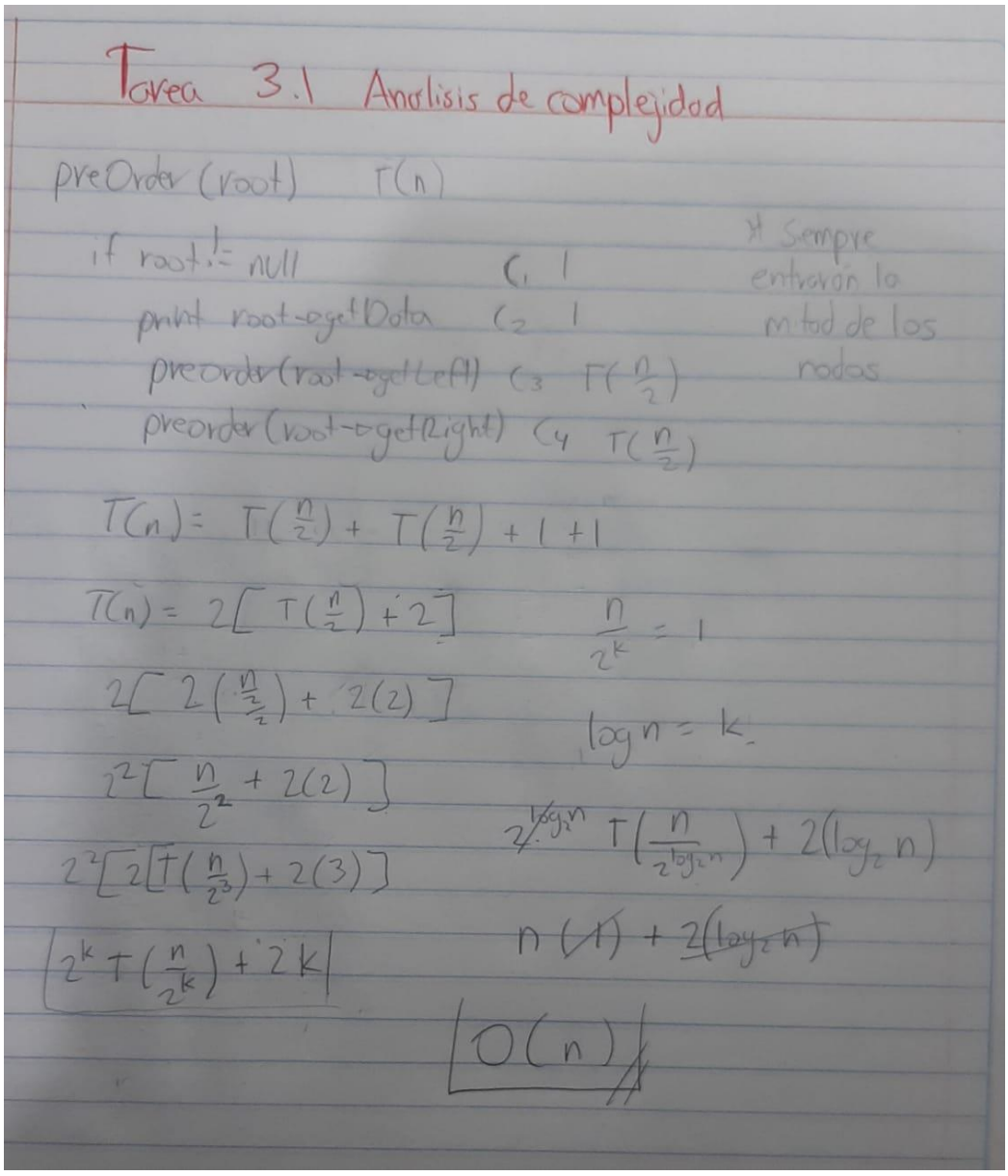
Actividad 3.1 – Operaciones avanzadas en un BST

Visit O(1)

```
void BST::visit(int order){

    switch (order)
    {
    case 1:

        preOrder(root);
        break;

    case 2:

        inOrder(root);
        break;

    case 3:

        postOrder(root);
        break;

    case 4:

        byLevel(root);
        break;

    default:
        break;
    }
```

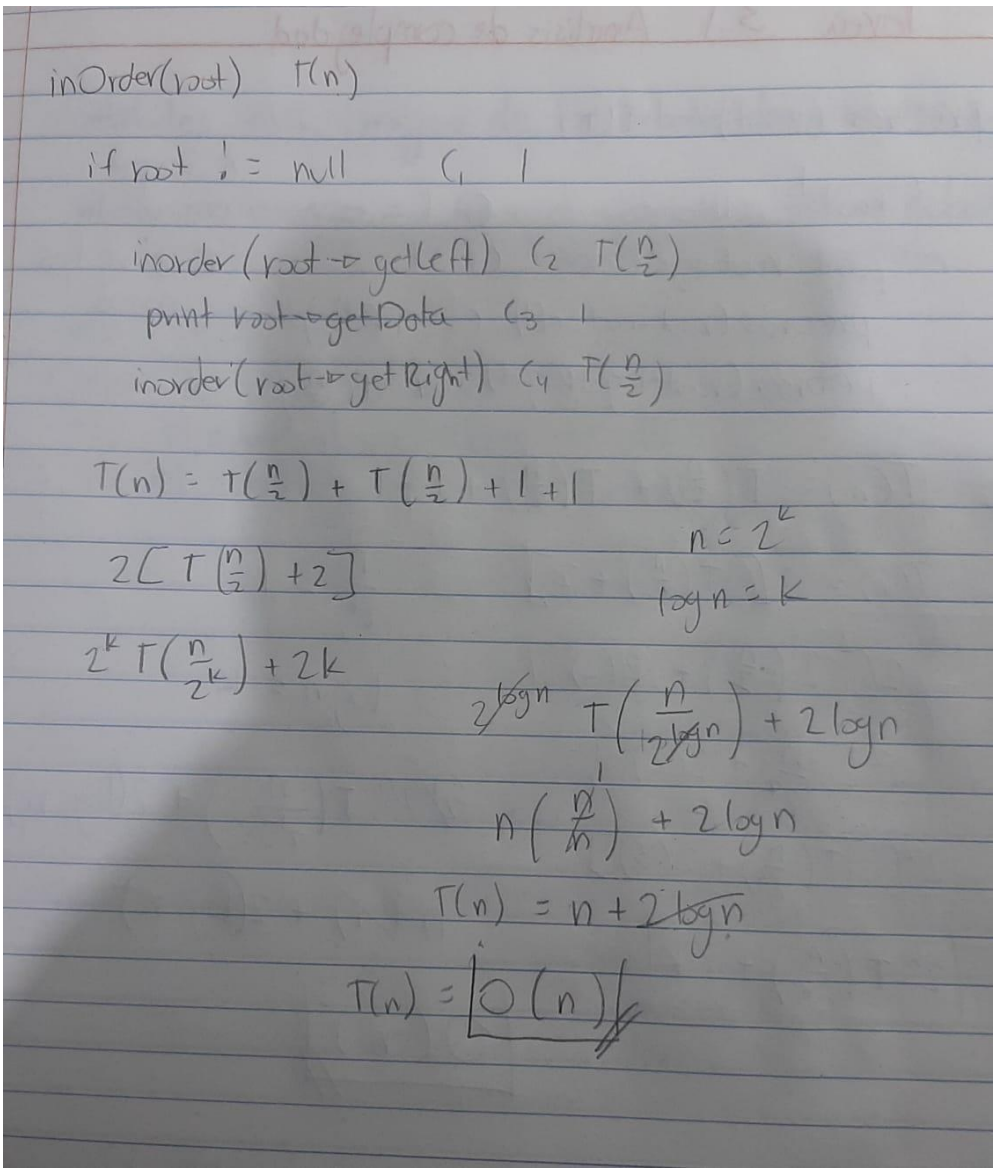
Preorder

```
void preOrder(Node* root){

    if (root != nullptr){

        std::cout << root->getData() << " ";
        preOrder(root->getLeft());
        preOrder(root->getRight());
    }
}
```



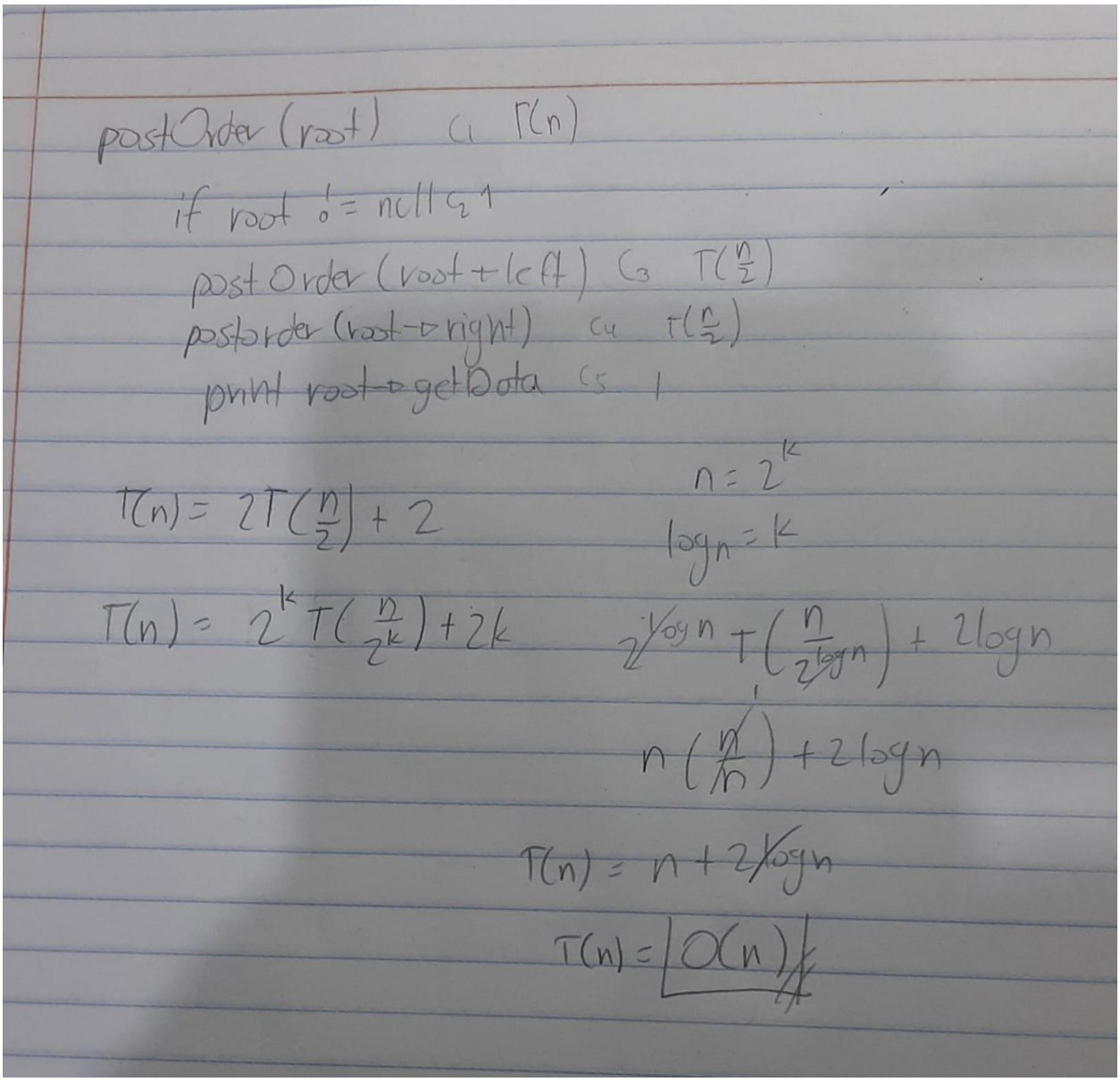
Inorder

```
void inOrder(Node* root){

    if( root != nullptr){

        inOrder(root->getLeft());
        std::cout << root->getData() << " ";
        inOrder(root->getRight());
    }

}
```



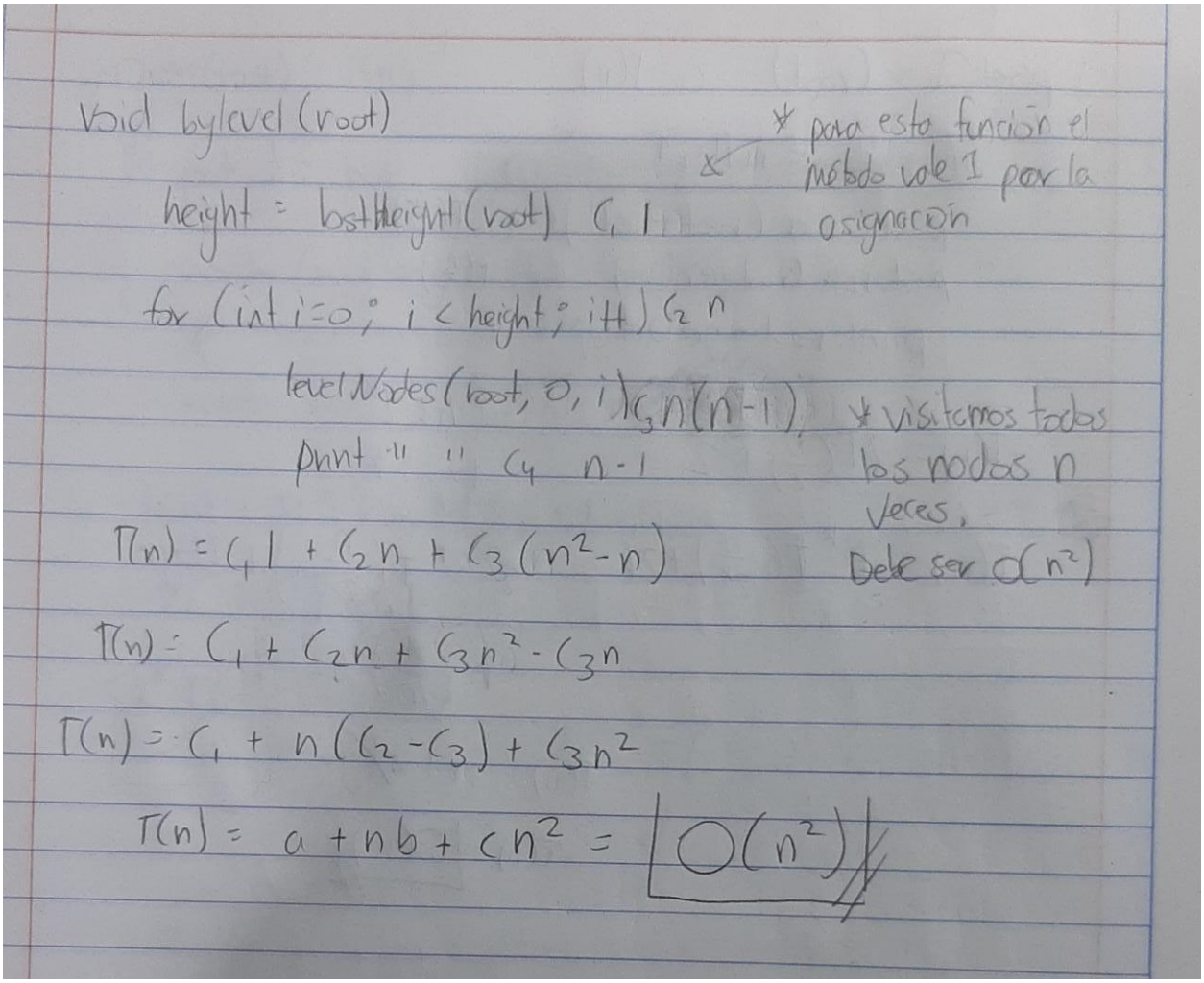
Postorder

```
void postOrder(Node* root){

    if (root != nullptr){

        postOrder(root->getLeft());
        postOrder(root->getRight());
        std::cout << root->getData() << " ";
    }
}
```



Level by level

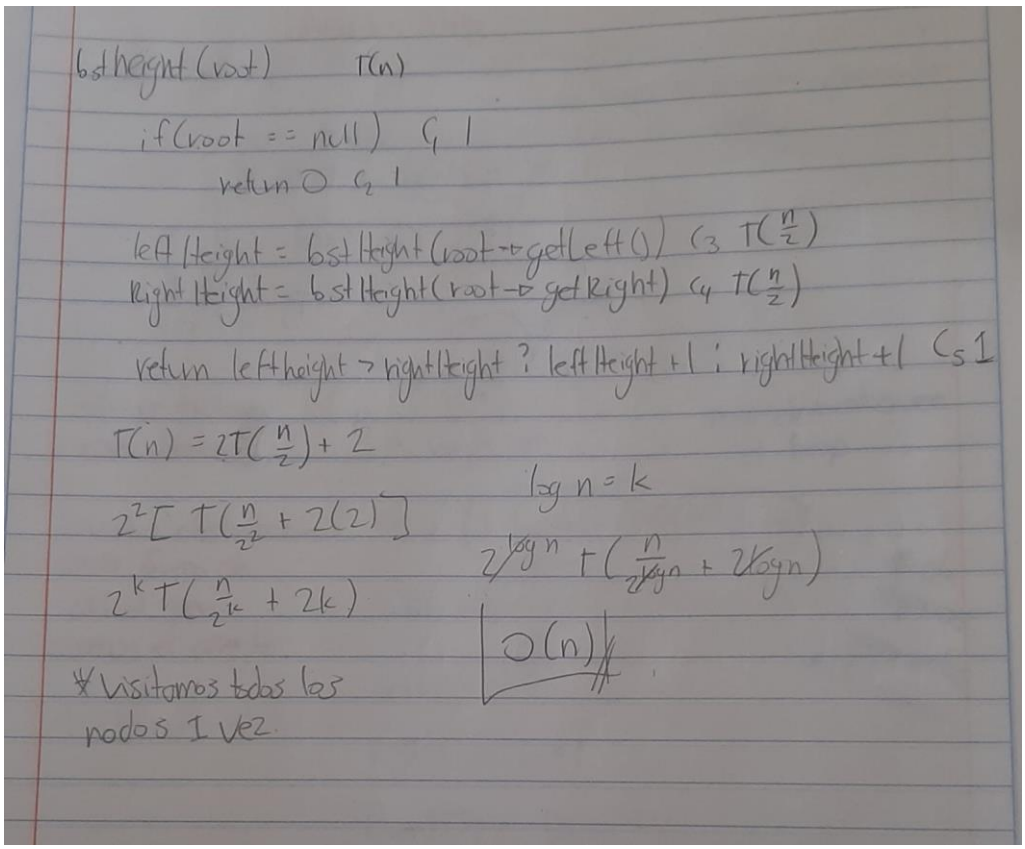
```
void byLevel(Node* root){

    // Guardamos la altura del árbol
    int height = bstHeight(root);

    for (int i = 0; i < height; i++){
        levelNodes(root, 0, i);
        std::cout << std::endl;
    }
}
```



Height

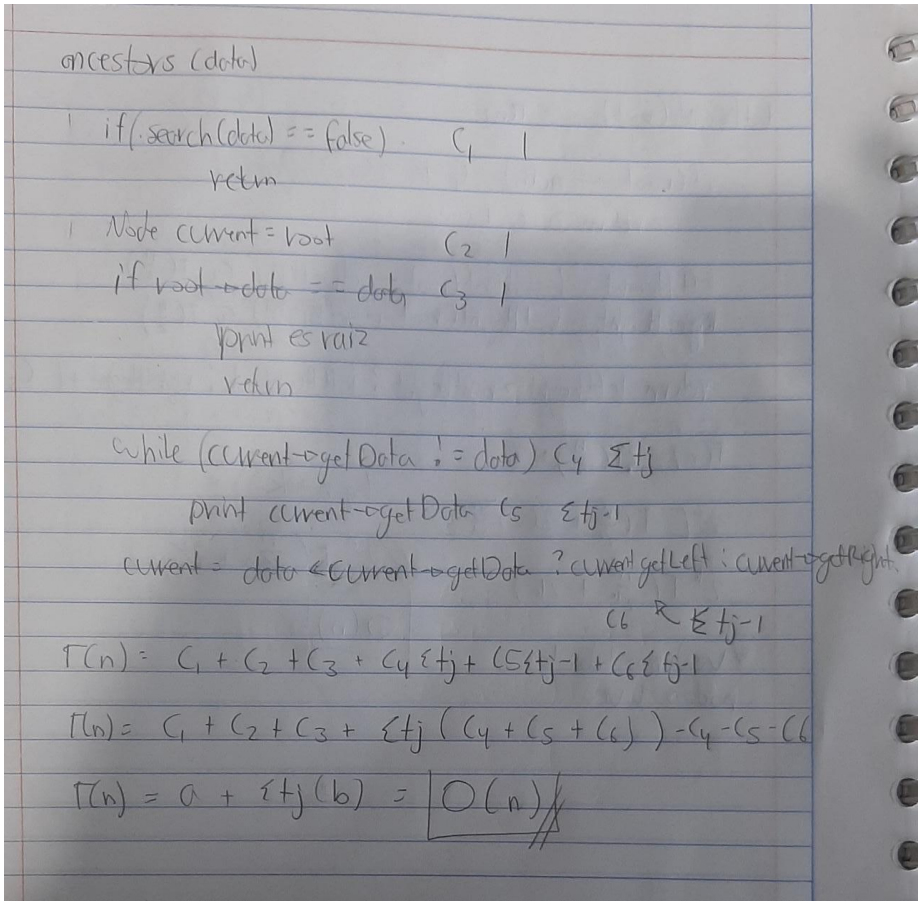
```cpp
/* Función que devuelve la altura de un BST*/
int BST::height(){

    // devolvemos el valor de altura que encontramos con la función.
    return bstHeight(root);

}
```

```cpp
/* Función escondida que encuentra la altura de un arbol recursivamente.*/
int bstHeight(Node* root){

    if (root == nullptr){
            return 0;
        }

    int leftHeight = bstHeight(root->getLeft());
    int rightHeight = bstHeight(root->getRight());

    // sumar 1 al lado que tenga más altura
    return  leftHeight > rightHeight ? leftHeight + 1 : rightHeight + 1;
}
```



Ancestors

```cpp
/* Función que visita los ancestros de un elemento del árbol.*/
void BST::ancestors(int data){

    // si no se encuentra ese elemento en el árbol no tiene ancestros.
    if (search(data) == false){
        std::cout << "Ese elemento no existe en el arbol. No hay ancestros.\n";
        return;
    }

    Node *current = root;

    // si el valor a revisar sus ancestros es la raíz, entonces imprimimos mensaje.
    if (root->getData() == data){
        std::cout << "Este nodo no tiene ancestros. Es la raiz." << std::endl;
    }

    while(current->getData() != data){

        std::cout << current->getData() << " ";

        // operador ternario-> (CONDICIÓN) ? ( TRUE que haces si es cierto) : ( FALSE que haces si es falso)
        // el pivote se mueve a la izquierda o a la derecha
        current = (data < current->getData()) ? current->getLeft() : current->getRight();

    }
}
```



WhatlevelAmI

```cpp
/* Función para encontrar en qué nivel se encuentra un elemento del árbol*/
int BST::whatlevelamI(int data){

    // El nodo raíz se ubica en el nivel 0.
    int nivel = 0;

    // Si el dato no se encuentra en el árbol
    if (search(data) == false){
        return -1;
    }

    Node *current = root;

    while(current->getData() != data){

        // operador ternario-> (CONDICIÓN) ? ( TRUE que haces si es cierto) : ( FALSE que haces si es falso)
        // el pivote se mueve a la izquierda o a la derecha
        current = (data < current->getData()) ? current->getLeft() : current->getRight();

        nivel = nivel + 1;

    }
    return nivel;
}
```