se deduce que la altura o profundidad de un árbol binario completo de n nodos es:

$h = |Log_2 n| + 1$ (parte entera de $Log_2 n + 1$)

Por ejemplo, un árbol lleno de profundidad 4 (niveles 0 a 3) tiene $2^4 - 1 = 15$ nodos.

13.2.3. *TAD* Árbol binario

La estructura de árbol binario constituye un *tipo abstracto de datos;* las operaciones básicas que definen el *TAD árbol binario* son las siguientes:

Tipo de dato	Dato que se almacena en los nodos del árbol.
Operaciones	
CrearArbol	Inicia el árbol como vacío.
Construir	Crea un árbol con un elemento raíz y dos ramas, izquierda y derecha que son a su vez árboles.

EsVacio	Comprueba si el árbol no tiene nodos.
Raiz	Devuelve el nodo raíz.
Izquierdo	Obtiene la rama o subárbol izquierdo de un árbol dado.
Derecho	Obtiene la rama o subárbol derecho de un árbol dado.
Borrar	Elimina del árbol el nodo con un elemento determinado.
Pertenece	Determina si un elemento se encuentra en el árbol.

El **recorrido de un árbol binario** requiere que cada nodo del árbol sea procesado (visitado) una vez, y sólo una, en una secuencia predeterminada. Existen dos enfoques generales para la secuencia de recorrido, profundidad y anchura.

En el **recorrido en profundidad,** el proceso exige un camino desde la raíz a través de un hijo, al descendiente más lejano del primer hijo antes de proseguir a un segundo hijo. En otras palabras, en el recorrido en profundidad, todos los descendientes de un hijo se procesan antes del siguiente hijo.

En el **recorrido en anchura,** el proceso se realiza horizontalmente desde el raíz a todos sus hijos; a continuación, a los hijos de sus hijos y así sucesivamente hasta que todos los nodos han sido procesados. En el recorrido en anchura, cada nivel se procesa totalmente antes de que comience el siguiente nivel.

Nota de programación

La visita al nodo se representa mediante la llamada al método de `Nodo, visitar()`. ¿Qué hacer en el método? Depende de la aplicación que se esté realizando. Si simplemente se quieren listar los nodos, puede emplearse la siguiente sentencia:

```
void visitar()
{
   System.out.print(dato + " ");
}
```

Ejemplo 13.7

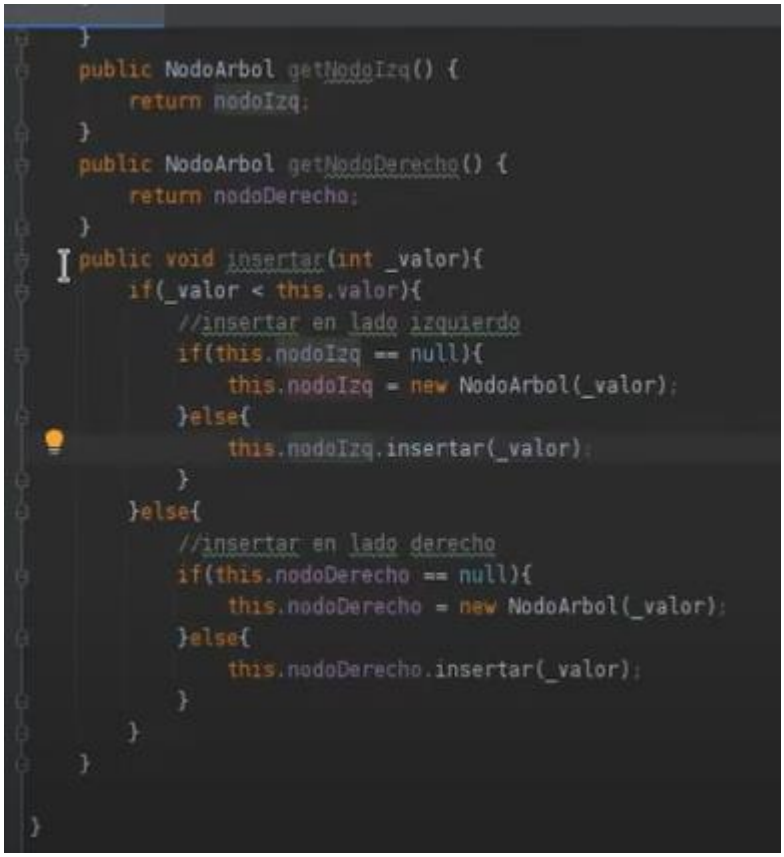
Dado un árbol binario, determinar el número de nodos de que consta.

El número de nodos se calcula recorriendo el árbol, y cada vez que se pasa por un nodo se incrementa la cuenta. El planteamiento es recursivo: el numero de nodos es `1` (nodo raíz) más el número de nodos de los subárboles izquierdo y derecho. El número de nodos de un árbol vacío es `0`, que será *el caso base* de la recursividad.

```
public static int numNodos(Nodo raiz)
{
   if (raiz == null)
      return 0;
   else
      return 1 + numNodos(raiz.subarbolIzdo()) +
                 numNodos(raiz.subarbolDcho());
}
```

A tener en cuenta

La altura de un árbol binario perfectamente equilibrado de `n` nodos es `log n`. Las operaciones que se aplican a los árboles AVL no requieren más del 44% de tiempo (en el caso más desfavorable) que si se aplican a un árbol perfectamente equilibrado.



```
    }
    public NodoArbol getNodoIzq() {
        return nodoIzq;
    }
    public NodoArbol getNodoDerecho() {
        return nodoDerecho;
    }
    public void insertar(int _valor){
        if(_valor < this.valor){
            //insertar en lado izquierdo
            if(this.nodoIzq == null){
                this.nodoIzq = new NodoArbol(_valor);
            }else{
                this.nodoIzq.insertar(_valor);
            }
        }else{
            //insertar en lado derecho
            if(this.nodoDerecho == null){
                this.nodoDerecho = new NodoArbol(_valor);
            }else{
                this.nodoDerecho.insertar(_valor);
            }
        }
    }

}
```