

$$T_g(x) =$$

31

$$T(m) = \begin{cases} a \cdot T(m-b) + c \cdot m^k & \text{si } m \geq b \\ c \cdot m^k & \text{si } m < b \end{cases}$$

$$\Rightarrow T(m) = \begin{cases} \Theta(1) & \\ \Theta(m^k) & \\ \Theta(m^k) & \end{cases}$$

Hacer:

Se tiene un array ~~no ordenado~~ de  $m$  elem. enteros, y  $m \leq 5$ .  
 Diseñar un algoritmo de coste  $\Theta(m \log n)$  (peor caso) que determine si hay 2 elementos en el array cuya suma es 5.

Crear el array y aplicar búsqueda binaria en los  $m$  elementos.  $\Theta(\log m)$

Teórica

El ~~conjunto~~ TDA Conj.

12/10/2016

Contenedor, lo cont. de elem. Almacena valor (no que esto permitido hacer repeticiones).

Aprender teórica:

- Crear conj.
- Destruir " "  $\rightsquigarrow$  borrar
- Hacer alta  $\rightsquigarrow$  conj F y elem. No está.
- Hacer bajo  $\rightsquigarrow$  conj F y elem F
- Esta? O pertenece?  $\rightsquigarrow$  ~~conj F y~~  
post condición: no modif. el conj.
- Recorrido (opcional).
- Opcional = Operaciones con los conjuntos

## class Conj

public:

```
Conj();  
~Conj();  
void Alta (int);  
void Baja (int);  
bool Esta (int);
```

private:

NodoAB \*p; < Ejemplo

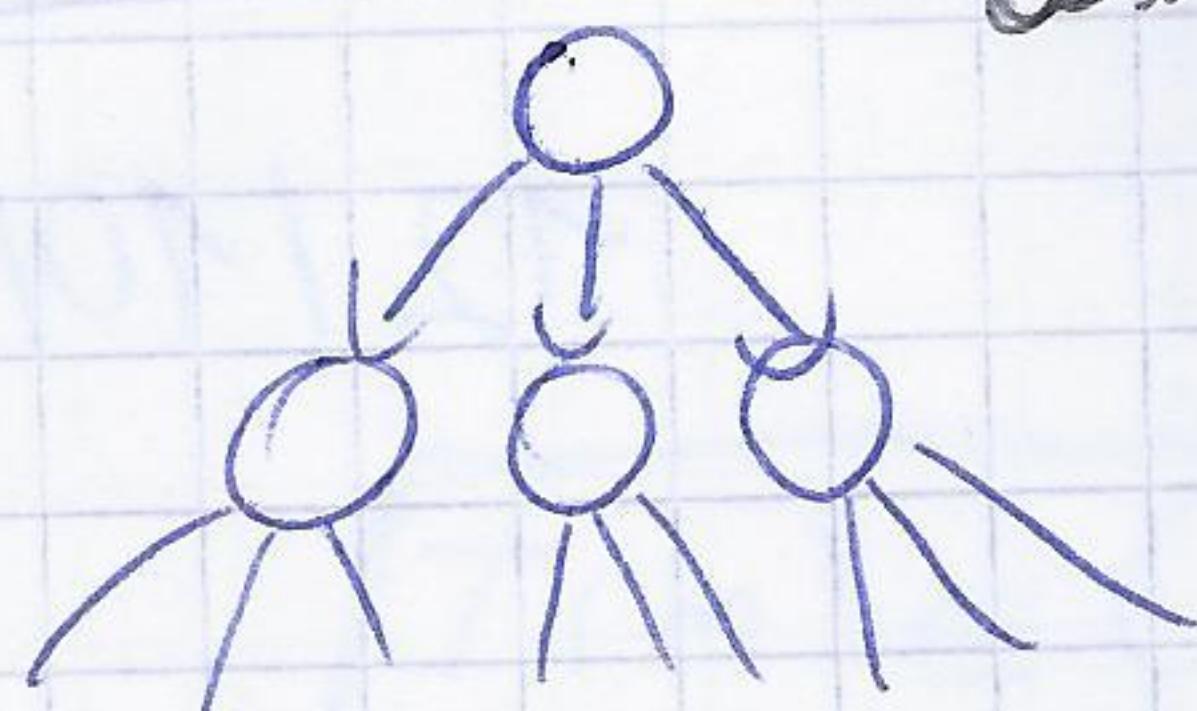
3

{ No interesa el orden de elem. (distinto a lista o vector)

{ Se puede implementar con un motor de estructuras (array, lista, etc) pero tener que no lo ordeno me cuesta en la funció "Este" y si los ordeno que cuesta en las búsquedas.

⇒ Surge el tipo de dato: Orbol.

Orbol → En la vida real

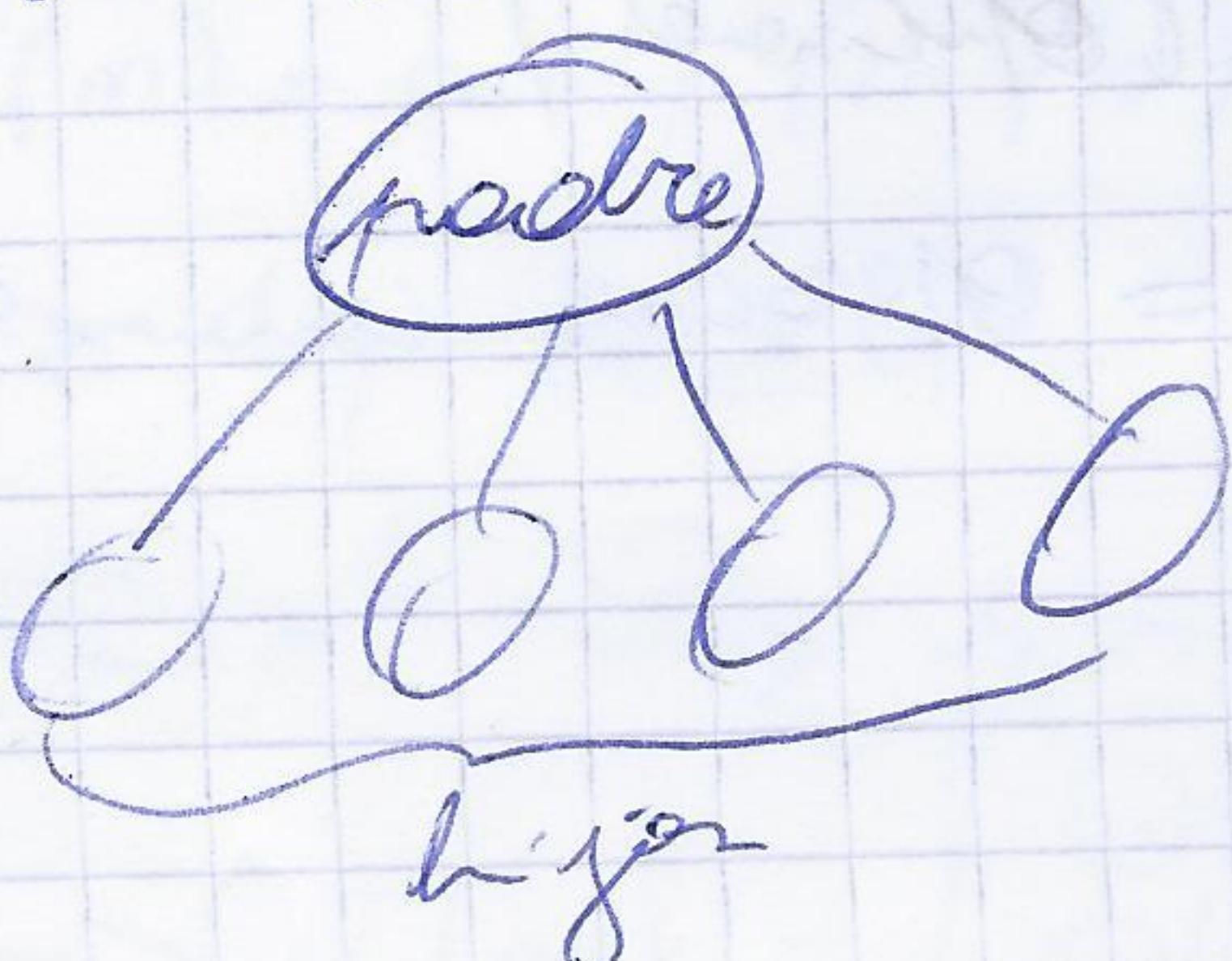


TDA: Tipo de dato abstr., que se puede implementar por nodos, en lista, etc. mientras que existan los operadores del mismo.  
Los caracteristicas del TDA se ve en la interfaz.

{ Orden jerárquico

{ No lo volvemos en la metá

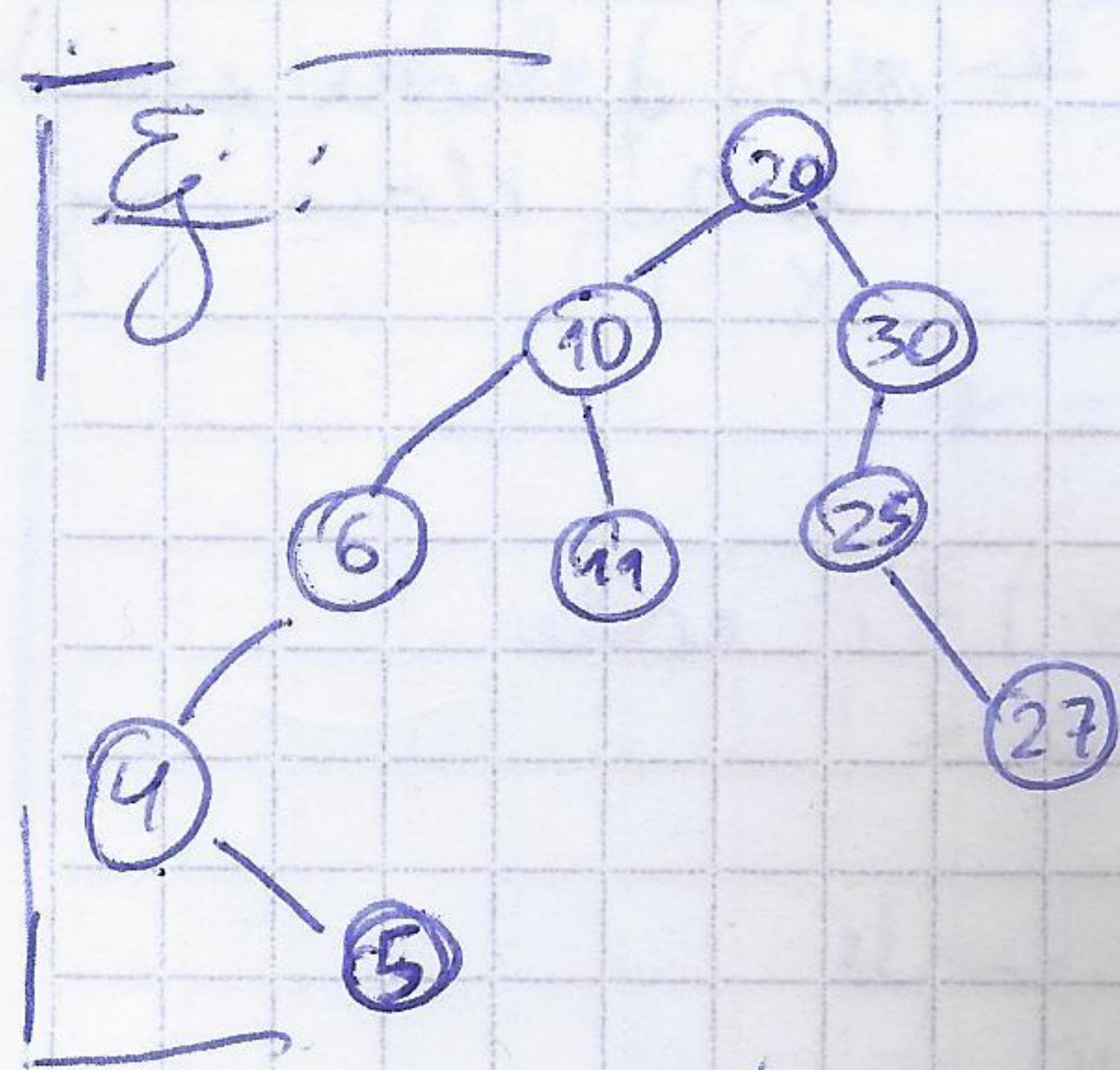
Estructura de datos Orbol: Serie de estructuras (nodos). Uno de esos nodos se distingue porque tiene padre.



- Se puede restringir la cont. de hijos.

## Árboles binarios: (ABB o BST "binary search tree")

Es, o bien nulo, o bien consta de un nodo raíz y de dos árboles hijos, izq. y der. que son A.B.



debris que da  
class NodoAB{  
private  
int info;  
NodoAB \*izq, \*der;

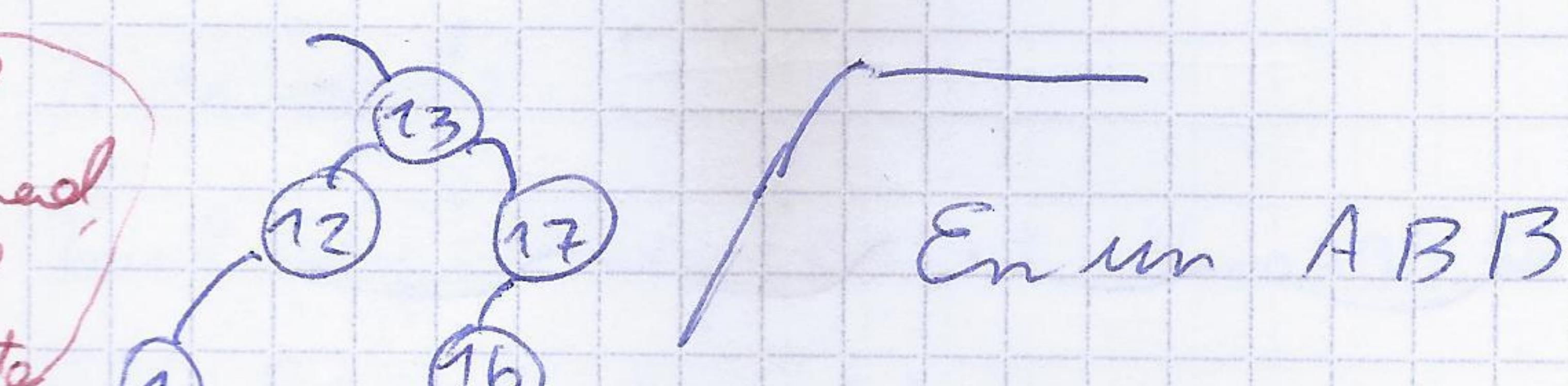
(dato ordenado {  
    izq, todos los datos menores  
    que 20 y a la derecha)  
    dato mayor que 20}

todo dato  
del subárbol < dato de  
izq. < dato de  
modo < todo dato  
del subárbol  
derecho.

Ej.: Procesamiento. Algoritmo para incluir un dato.

Datos a incluir: 13, 17, 12, 1, 4, 12, 16:

el árbol  
se queda lineal  
de lo que  
degenera a lista



En un ABB

$O(\log N)$   $\leq$  Coste de  
la búsqueda de  
dato en ABB  $\leq O(N)$   
Si ABB degenera  
en lista

Si ABB está  
dado como totalmente  
(pero en  
triángulo)

$\hookrightarrow 16$

## Implementación Alta Log:

```

void Cong :: Alta(int *x) {
    if (!p) {
        p = new NodoABB(x);
        // si nacimiento es Null
    } else {
        bool Listo = false;
        NodoABB aux = p;
        while (!Listo) {
            if (x == aux->info)
                Listo = true;
            else {
                if (aux->derech)
                    aux = aux->derech;
                else {
                    aux->derech = new NodoABB(x);
                    Listo = true;
                }
            }
        }
        else {
            if (aux->izq)
                aux = aux->izq;
            else {
                aux->izq = new NodoABB(x);
                Listo = true;
            }
        }
    }
}

```

class NodoABB {  
private:  
 int info;  
 NodoABB \*izq, \*derech;

public:  
 NodoABB(int x) { info = x;  
 izq = derech = 0; }

friend class Log;  $\Rightarrow$  opera que se pide en  
 $\Rightarrow$  la clase Log

$\} \rightarrow$  descendencia

$\} \rightarrow$  no lo encuentra  
 $\} \rightarrow$  lo crea

$\} \rightarrow$  descendencia.

$\} \rightarrow$  no lo encuentra  
 $\} \rightarrow$  lo crea.

## Alta recursiva:

1) m't público:

```

void Cong :: Alta(int x) {
    Alta(p, x); }

```

2) m't privado:

```

void Cong :: Alta(NodoABB *&q, int x) {

```

```

    if (!q) { q = new NodoABB(x); }
    else if (x > q->info) Alta(q->derech, x); }
    else if (x < q->info) Alta(q->izq, x); }

```

$\} \rightarrow$  descendencia

## // Busqueda iterativa:

```

bool Cong::Este (int x) {
    bool r = false;
    NodoABB *aux = p;
    while (!r && (aux)) {
        if (x == aux->info) {
            r = true;
        } else if (x > aux->info) {
            aux = aux->der;
        } else {
            aux = aux->izq;
        }
    }
    return r;
}

```

## // Busqueda recursivo

### // publico

```

bool Cong::Este (int x) {
    return Este(p, x);
}

```

### // privado

```

bool Cong::Este (const NodoABB *& q, int x) {
    if (!q) return false;
    else if (x == q->info) return true;
    else if (x > q->info) return Este(q->der, x);
    else return Este(q->izq, x);
}

```

Arbol Árbol es un caso particular de grafo.

2 Tipos de recorridos:

- Rec. en profundidad: Siempre se usa el modo de búsqueda primero.

variantes:  
} {  
    ↳ preorden  
    ↳ posorden  
    ↳ inorden

- Rec. en Anchura (primero.)

### Rec. prof.

- Pre orden: 1º la raiz, desp. el lado izq., desp. el lado derech.

Preorden (p) {

if (p){  
    procesar info de nodo apunt. por p.  
    Preorden (izq de p)  
    Preorden (derech de p)

} {

- Pos orden (p) {

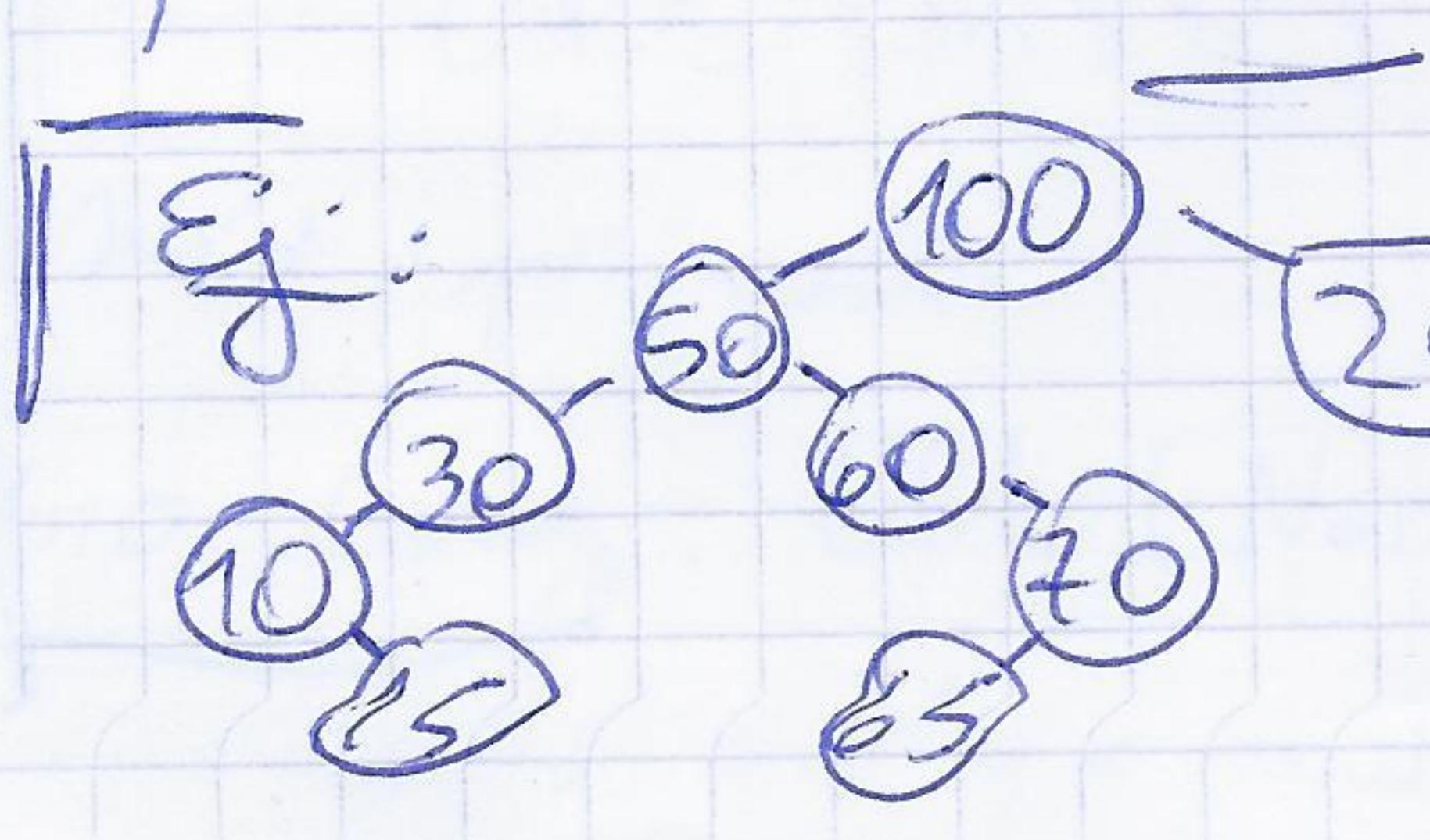
if (p){  
    Pos orden (izq de p)  
    Pos orden (derech de p)  
    Procesar info de nodo apunt. por p.

} {

- In orden (p) → El es interesante

if (p){  
    In orden (izq de p)  
    Procesar de info ...  
    In orden (derech de p)

} {



proc. de info

• Pre orden: 100 - 50 - 30 - 10 - 15 - 60 - 70 - 65 -  
200 - 400 - 300 - 600 - 450

• Pos orden: 15 - 10 - 30 - 65 - 70 - 60 - 50 -  
300 - 450 - 600 - 400 - 200 - 100

• In orden: 10 - 15 - 30 - 50 - 60 - 65 - 70 - 100 -  
200 - 300 - 400 - 450 - 550 - 600

Eg: Inorden: Recurso. (Nós iteradores)

II : N orden relativo  
II met público

```
Void Conigur :: InOrder(){
    inOrder(p);
```

# II Mít. privado

```
void Conjunto::Inorden(const NodoABB *&q){
```

if (q) {

inORDEN(q  $\Rightarrow$   $\neg q$ );

`set::cout <math>q \rightarrow^{\delta^+} \text{info}; \rightarrow \text{Prozess}  
in order ( $q \rightarrow \text{durch}$ );`

in order ( $q \Rightarrow$  dreh);

3

Rec. Andhra

> No se plantan recursos vivos ya que es más intuitivo.

↳ Mordhete se nun aler.

Borre de nivel o nivel la info de los nodos.

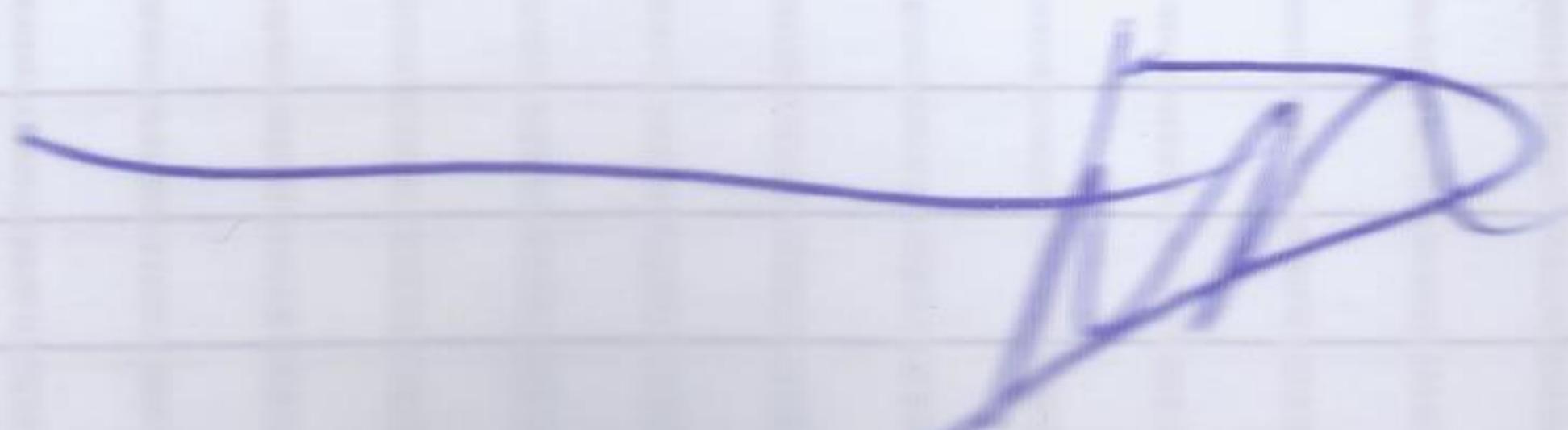
- Acer rain

Mientras Cole novocá{

dene color en x

Proleson x

Si x tiene ing. no nulo  $\Rightarrow$  orden  $\geq q$



Borrado de nodos en ABB:

Contemple el siguiente caso:

1º Nodo sin hijos

2º Nodo con 1 hijo

3º " " 2 hijos: Reemplaza el dato actual por "menor de mayores" o "mayor de menores" y elimina los hijos (uno t  
2 ya que no puede tener más de un hijo)