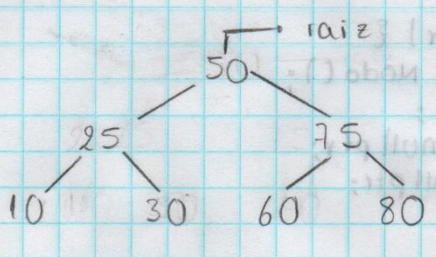
Universidad de las Fuerzas Armadas "ESPE"

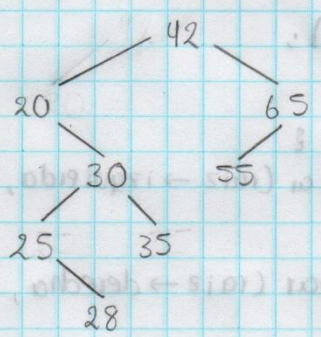
Mayely Simbaña
10/12/2025
MRC.

ARBOL BINARIO

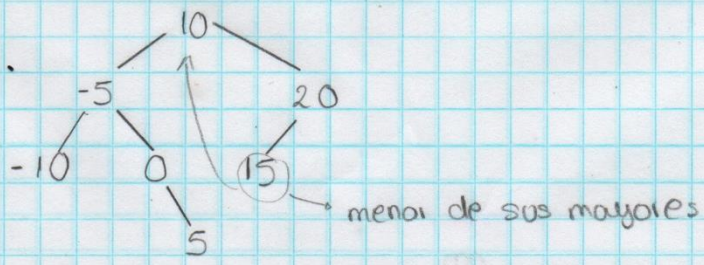
① Lista: 50, 25, 75, 10, 30, 60, 80



② Lista: 42, 20, 65, 30, 55, 25, 35, 28

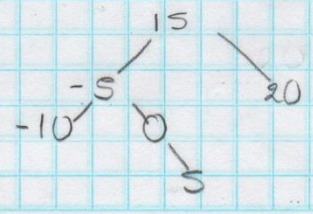


③ Lista: 10, -5, 20, 0, -10, 15, 5

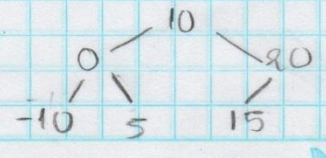


Si se elimina la raiz se busca al número más parecido, al menor de sus mayores

si se elimina 10 =>



Si se elimina -5



Código:

```
#Include <iostream>
usignname space std;

struct Nodo {

    int dato;
    Nodo* izquierda;
    Nodo* derecha;
};

Nodo* crearNodo (int valor) {
    Nodo* nuevo = new Nodo();
    nuevo → dato = valor;
    nuevo → izquierda = nullptr;
    nuevo → derecha = nullptr;
    return nuevo;
}

Nodo* insertar (Nodo* raiz, int valor) {

    if (raiz == nullptr) {
        return crearNodo (valor);
    }

    if (valor < raiz → dato) {
        raiz → izquierda = insertar (raiz → izquierda, valor);
    }
    else {
        raiz → derecha = insertar (raiz → derecha, valor);
    }

    return raiz;
}
```

ARBOL AVL

AVL → Por (Adelson - Velsky y Landis)

Detector de Inclinación (Factor de Equilibrio)

Izquierda — Derecha

Puede ser : -1, 0, 1

0: Equilibrado
1: Pesa más la izquierda
-1: Pesa más la derecha

Si el resultado es 2 o -2 ocurre La Rotación.

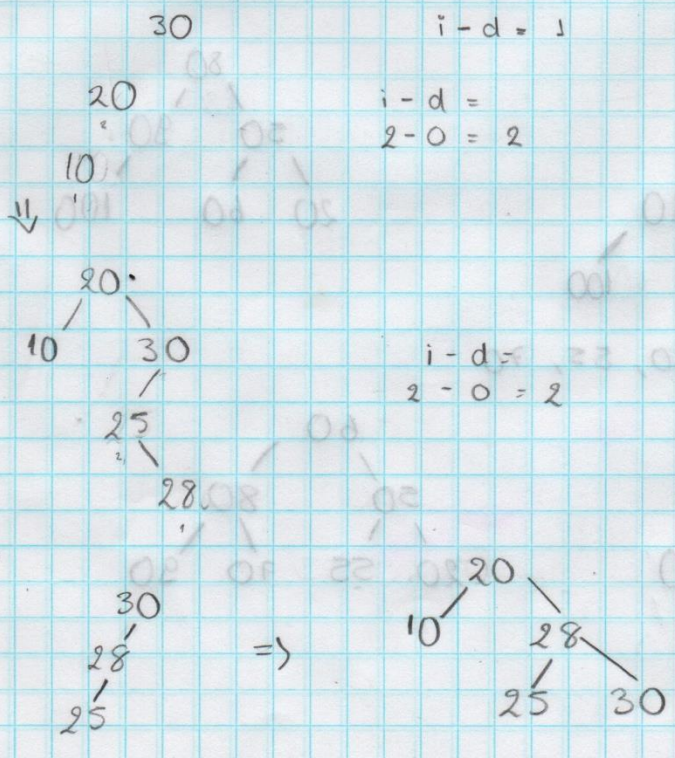
CASO A: Línea recta de 3 números
se toma el número del medio y lo subes

CASO B: Zig-zag

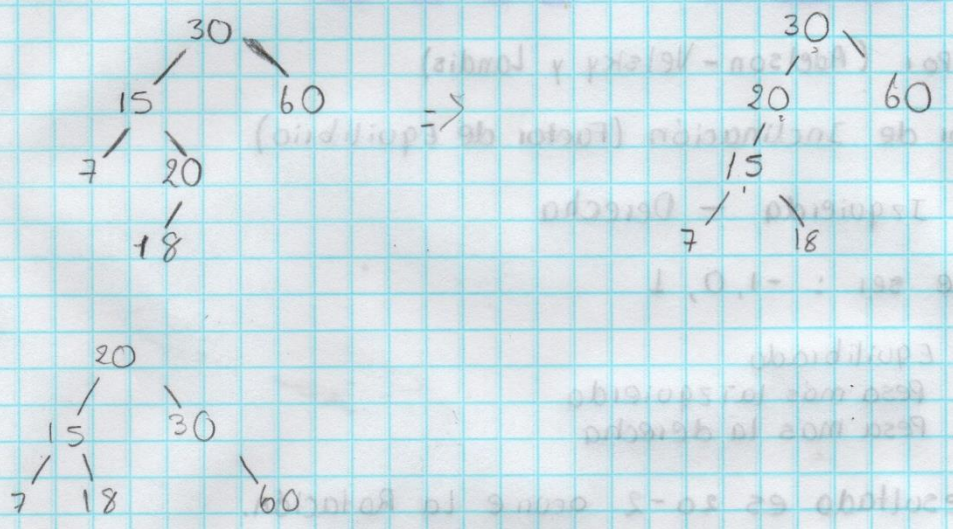
1: Primero se convierte en una línea recta
2: Luego se hace lo del caso A (subir el de en medio)

Ejemplo

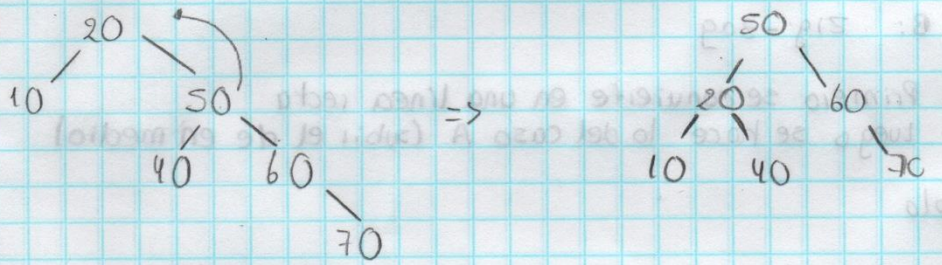
Lista: 30, 20, 10, 25, 28



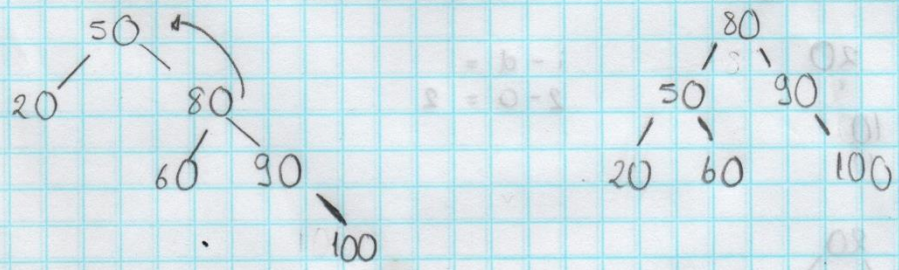
Lista: 30, 15, 60, 7, 20, 18



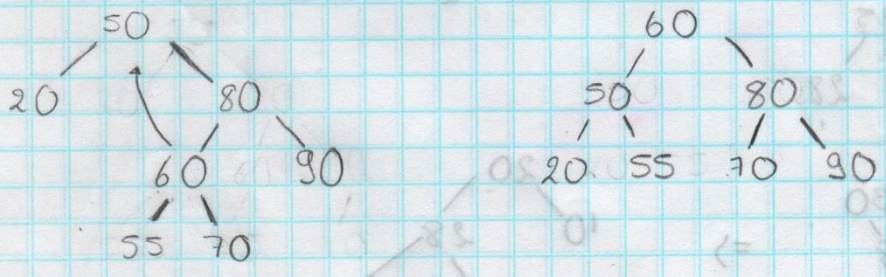
Lista: 20, 10, 50, 40, 60, 70



Lista: 50, 20, 80, 60, 90, 100



Lista: 50, 20, 80, 60, 90, 55, 70



Código:

```
struct Nodo {
    int dato;
    Nodo* izquierda;
    Nodo* derecha;
    int altura;
};

int obtenerAltura (Nodo* n) {
    if (n == nullptr) return 0;
    return n->altura;
}

Nodo* crearNodo (int dato) {
    Nodo* nodo = new Nodo ();
    nodo->dato = dato;
    nodo->izquierda = null;
    nodo->derecha = null;
    nodo->altura = 1;
    return nodo;
}

int obtenerBalance (Nodo* n) {
    if (n == null) return 0;
    return obtenerAltura (n->izquierda) - ObtenerAltura (n->derecha);
}

Nodo* rotarDerecha (Nodo* Y) {
    Nodo* X = Y->izquierda
    Nodo* T2 = X->derecha

    X->derecha = Y;
    Y->izquierda = T2;

    Y->altura = max (obtenerAltura (Y->izquierda), obtenerAltura (Y->derecha))+1;
    X->altura = max (obtenerAltura (X->izquierda), obtenerAltura (X->derecha))+1;

    return X;
}

Nodo* rotarIzquierda (Nodo* X) {
    Nodo* Y = X->derecha;
    Nodo* T2 = Y->izquierda;

    Y->izquierda = X;
    X->derecha = T2;

    X->altura = max (obtenerAltura (X->izquierda), obtenerAltura (X->derecha)) +1;
    Y->altura = max (obtenerAltura (Y->izquierda), obtenerAltura (Y->derecha)) +1;

    return Y;
}
```

```
Nodo* insertar (Nodo* nodo, int dato) {
    if (nodo == null) return crearNodo(dato);

    if (dato < nodo->dato)
        nodo->izquierda = insertar(nodo->izquierda, dato);

    else if (dato > nodo->dato)
        nodo->derecha = insertar(nodo->derecha, dato);

    else
        return nodo;

nodo->altura = 1 + max(obtenerAltura(nodo->izquierda), obtenerAltura(nodo->derecha));

int balance = obtenerBalance(nodo);

if (balance < -1 && dato > nodo->derecha->dato)
    return rotarIzquierda(nodo);

if (balance > 1 && dato > nodo->izquierda->dato)
    nodo->izquierda = rotarIzquierda(nodo->izquierda);
    return rotarDerecha(nodo);
}

if (balance < -1 && dato < nodo->derecha->dato) {
    nodo->derecha = rotarDerecha(nodo->derecha);
    return rotarIzquierda(nodo);
}
    return nodo;
}
```