Disclaimer: estas son las notas hechas en la clase TM, puede ser que alguna cosa no se entienda si no participaron de la clase

```
proc nuevoSistema() : Sistema                                                          O(1)
proc registrarMateria(inout s : Sistema, in m : materia)                               O(log m)
proc registrarNota(inout s : Sistema, in m : materia, in a : alumno, in n : nota)      O(log n + log m)
proc notaDeAlumno(in s : Sistema, in a : alumno, m : materia) : nota                   O(log n + log m)
proc cantAlumnosConNota(in s : Sistema, in m : materia, n : nota) : Z                  → O(log m)
proc cantAlumnosAprobados(in s : Sistema, in m : materia) : Z
```

?

m = cant materias , n = cant alumnos

① estructura

var → dm = DiccLog < materia (K), diccLog < alumno, nota > (V) > ¿cumple las comp pedidas?

nuevoSistema ⇝ dm = diccionarioVacio (tipos: mat, diccLog<alum, nota>)
O(1) O(1)

Registrar Materia: insertar m:materia en dm: diccionarioLog ⇝ O(log m) ✓

Registrar Nota:
buscar materia + { buscar (si a:alumno esta) en da: diaLog ; insertar (si a no está) }
⇝ O(log m) + O(log n) ✓

Nota De Alumno:
buscar m en dm + buscar a en da ⇝ log(m) + log(n) ✓

Cant Alumnos Con Nota
buscar m en dm + para cada a ver la nota (recorrer todo da)
O(log m) + O(n) = max{O(log m), O(n)} = O(n) ✗
asumiendo que hay más alumnos que materias (m ≤ n)

Vemos que con solo dm no alcanza, proponemos también:

var dn: diccLog < materias, notas: array[11] >

en cada pos tengo la cant de alumnos con esa nota

Cant AlumnosConNota: buscar m materia + ver una pos de un array
⇝ O(log m) + O(1) = O(log m) ✓

cantAlumnosAprobados buscar m materia + contar la pos 7, 8, 9 y 10 del array
⇝ O(log m) + O(1) = O(log m) ✓

Veamos si las anteriores siguen OK:

RegistrarMateria en O(log m) → insertar m materia en cm → O(log m) + O(log m) (anterior) ✓
RegistrarNota en O(log n + log m) → buscar m en cm + modificar nota
NotaDeAlumno en O(log n + log m) O(log m) + O(1) + O(log n + log m) ✓

Modulo sistema implementa (TAD) Sistema

var : dm = diccLog(materia, diccLog(alumno, nota))

dn = diccLog(materia, array[11]: $\mathbb{Z}$)

3 y 4 →

proc nuevoSistema() O(1) tipos

dm = new diccLog< materia, diccLog(alumno, nota)) O(1)

dn = new diccLog< materia, array[11]> O(1)

dm = diccionarioVacio() O(1)

dn = diccionarioVacio() O(1)

proc registrarMateria (inout s: sistema, in m: materia)

→ s.dm.definir(m, new diccLog(alumno, nota)) $O(\log m) + o(1)$

→ s.dn.definir(m, new array[11]) $O(\log m) + o(1)$

$O(\log m) + o(1) + O(\log m) + o(1) =$

$= \max\{O(\log m), o(1), O(\log m), o(1)\} = O(\log m)$

proc registrarNota (inout s: sistema, in m: materia, in n: nota, in a: alumne)

s.dm.obtener(m) $O(\log m)$. definir(a, n) $O(\log n)$ $O(\log m) + O(\log n) = O(\log m + \log n)$

aux = s.dn.obtener(m) $O(\log m) + o(1)$

aux[n] = aux[n] + 1 $O(1)$

s.dn.definir(m, aux) $O(\log m)$

$\max\{O(\log m + \log n), O(\log m) + o(1), O(1), O(\log m)\} = O(\log m + \log n)$

proc NotaAlumno (in s: sistema, in m: materia, in a: alumno): $\mathbb{Z}$

int res = s.dm.obtener(m).obtener(a) $O(\log m + \log n)$

return res $O(1)$

$\max\{O(1), O(\log m + \log n)\} = O(\log m + \log n)$

proc cantAlumnosConNota (in s: sistema, in m: materia, in n: nota): $\mathbb{Z}$

int res = s.dn.obtener(m)[n] $O(\log m) + o(1)$

return res $O(1)$

$\max\{O(1), O(\log m)\} = O(\log m)$

```
proc cantAlumnosAprobados (in s: sistema, in m: materia): ℤ
    aux[] = s.dn.obtener(m)
    int res = aux[7] + aux[8] + aux[9] + aux[10]
    return res
```

¿cómo se relacionan las var del módulo?

2. Su invariante de representación y su función de abstracción

¿cómo se relacionan las var del módulo con los obs del TAD?

Módulo sistema

```
dm = dicc Log (materia, dicc Log (alumno, nota))
dn = dicc Log (materia, array [11..7]: ℤ)
```

TAD Sistema {
obs notas: dict⟨*materia*, dict⟨*alumno*, ℤ⟩⟩

```
pred invRep (s: sistema)
    1°  claves(dm) = claves(dn) ←
    2°  ∀ m:materia (∀ n:nota (0 ≤ n ≤ 10 ∧ m ∈ claves(dm)
⇒ ∃ c: conj(alumnos) (∀ a:alumno) (a ∈ c ⇔ dm[m][a] = n
    ∧ tamanio(c) = dn[m][n]

pred funAbs (s: sistema [módulo], s': Sistema [TAD]):
    s.dm = s'.notas
```

cantElemEnRango (C, 2, 7) ⇝ 1
cantElemEnRango (C, 2, 8) ⇝ 2
cantElemEnRango (C, 20, 100) ⇝ 2
cantElemEnRango (C, 3, 28) ⇝ 3

Ejercicio 4. Considerar la siguiente extensión sobre el TAD Conjunto⟨Z⟩

```
TAD Conjunto⟨Z⟩ {
obs elems: conj⟨Z⟩
proc cantElementosEnRango(in c : Conjunto⟨Z⟩, in desde : Z, in hasta : Z) : Z
   requiere { desde ≤ hasta }
   asegura { (∃ c': conj⟨Z⟩ )((∀ e: Z) ( e ∈ c' ⟺ e ∈ c ∧ desde ≤ e < hasta) ∧ res = |c'| ) } }
```

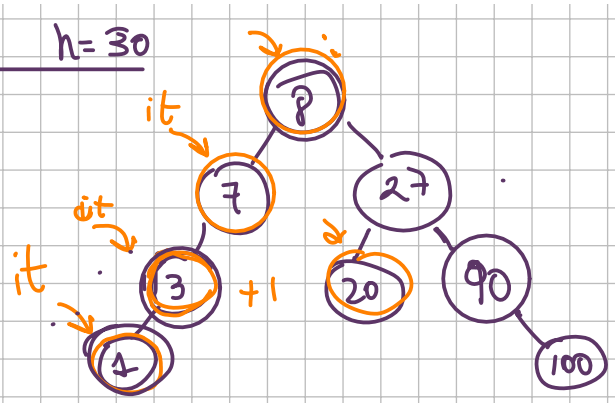
1. Escriba el algoritmo para el `cantElementosEnRango` si el TAD Conjunto⟨Z⟩ está implementado sobre ABB. ¿Qué complejidad tiene?

```
private class Nodo {
    T valor;
    Nodo izq;
    Nodo der;
    Nodo padre;

    Nodo(T v) {
        valor = v;
        izq = null;
        der = null;
        padre = null;
    }
}
```

→ # hijos

d = 2 h = 30



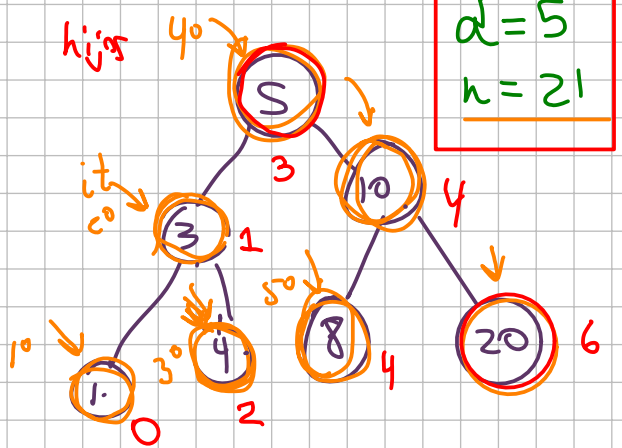

```
public class ABB<T extends Comparable<T>> implements Conjunto<T> {
    private Nodo _raiz;

    public ABB() {
        _raiz = null;
    }
}
```

proc CantElemEnRango(in c:conj(Z), d:int, h:int)

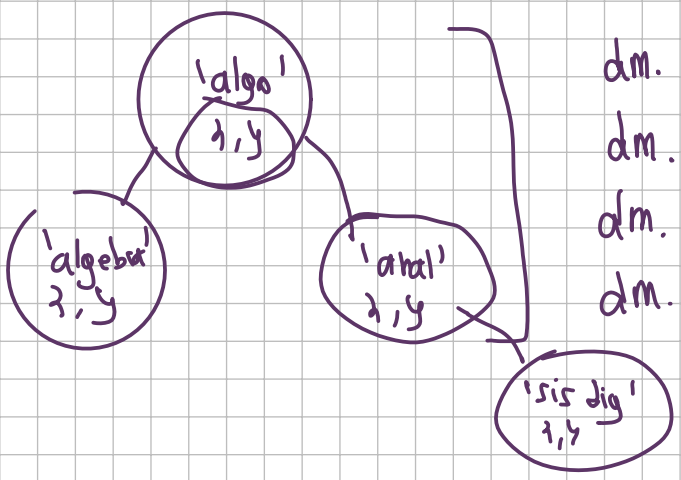
```
int cont = 0
it = c.iterador()
while ( it.haySiguiente() && it.valor < d)        // 5  < 5
    it.siguiente
while ( it.haySiguiente() && it.valor < h)        // 20
    cont ++
    it.siguiente()
return cont
```

IteradorBidireccional	
proc	complejidad
haySiguiente	$O(\log n)$
hayAnterior	$O(\log n)$
siguiente	$O(\log n)$
anterior	$O(\log n)$



d = 5
h = 21

AVL dm: <materia, da>



dm. definir ('algo', ?, ?)
dm. definir ('algebra', ?, ?)
dm. definir ('anal', ?, ?)
dm. definir ('sis dig', ?, ?)