1) $T(n) = T(n-1) + 1$, con $T(1) = 0$

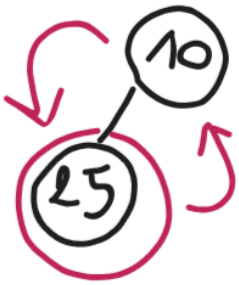
$T(n-1) = T(n-2) + 1 \Rightarrow T(n) = [T(n-2) + 1] + 1 = T(n-2) + 2$
$T(n-2) = T(n-3) + 1 \Rightarrow T(n) = [T(n-3) + 1] + 2 = T(n-3) + 3$
De donde:
$T(n) = T(n-k) + k$, siendo k un número natural, $k >= 1$.
Si $k == n$ se tiene que $T(n) = T(0) + n = n + 1$. Entonces
$T(n) \in O(n)$. Como $O(n) \subseteq O(n^2)$, la proposición es V.

$T(n) = 2T(n/2) + n^2$, con $T(1) = 1$.
Entonces, usando el Teorema Maestro de reducción por división,
se tiene que:
$a = 2, b = 2, k = 2 \Rightarrow b^{**}k = 2^2 = 4 > a = 2 \Rightarrow T(n) \in O(n^{**}k) = O(n^2)$.
Con lo cual, y siguiendo con el razonamiento anterior, la prop. es V.

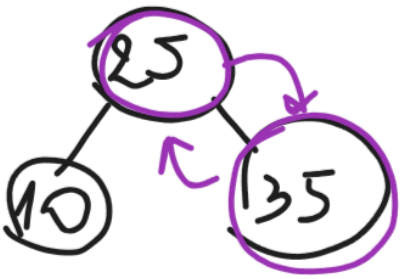
2)
a)

```
public class ABB<T> {
    private Nodo<T> raiz;

    public int contarNodosSinHijos() {
        return contarNodosSinHijos(this.raiz);
    }

    private int contarNodosSinHijos(Nodo<T> nodo) {
        if (nodo == null) {
            return 0;
        }
        if ((nodo.izq == null) && (nodo.der == null)) {
            return 1;
        }
        return contarNodosSinHijos(nodo.izq) + contarNodosSinHijos(nodo.der);
    }

}
```

b)
Valores = [10, 25, 35, 30, 2, 1]

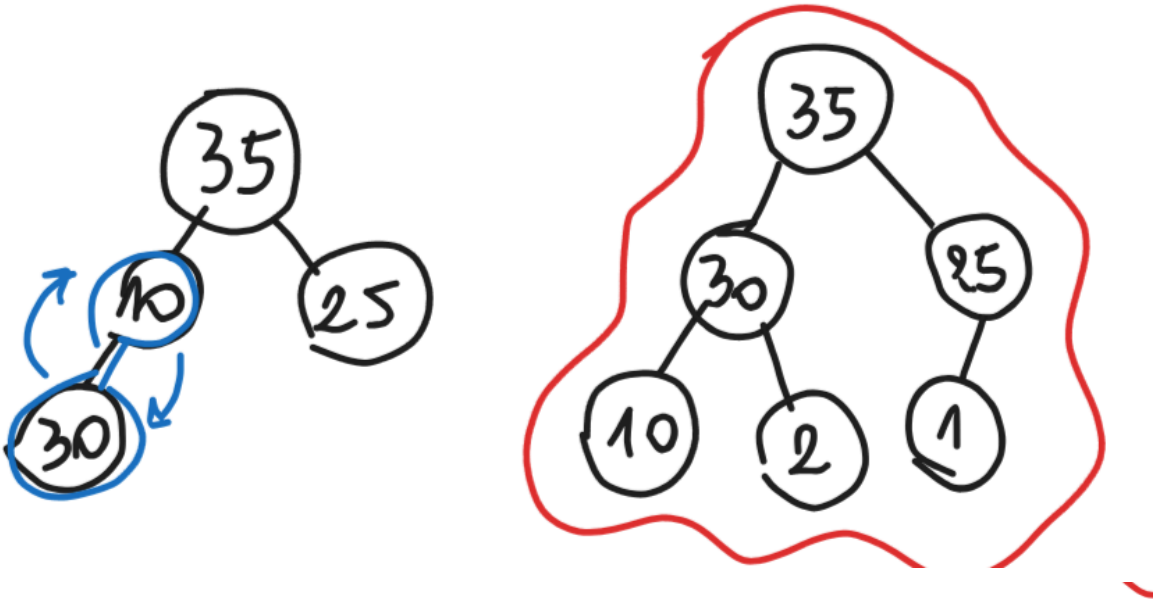


Al insertar, como el nodo padre 10 es menor que el hijo 25, se produce un swap(10, 25) (upheap!)



Al insertar el 35, el nodo padre 25, que es menor swapea con éste.

con este:



Un Heap es un árbol binario que satisface dos condiciones fundamentales:
Propiedad de Heap: En un Max-Heap (heap de máximos), el valor de cada nodo padre es mayor o igual al de sus hijos, asegurando que la raíz sea siempre el elemento mayor. De forma recursiva, cada subárbol debe cumplir también esta propiedad. (En un Min-Heap, el padre es menor o igual a sus hijos).

Propiedad de Estructura (Árbol Completo): El árbol debe estar completamente lleno en todos sus niveles, a excepción del último, el cual debe completarse de izquierda a derecha.
Debido a su eficiencia para acceder al elemento de mayor (o menor) prioridad, el Heap es fundamental en:
TDA Cola de Prioridad: Permite inserciones y extracciones del elemento óptimo en tiempo logarítmico (O(logn)).
Algoritmos Greedy: Es la estructura auxiliar clave para optimizar algoritmos como Dijkstra (caminos mínimos) y Prim (árbol de expansión mínima).
HeapSort: Un algoritmo de ordenamiento eficiente que aprovecha la estructura para ordenar arreglos in-place con una complejidad de O(nlogn).

La Programación Dinámica es una técnica de diseño de algoritmos que se utiliza para resolver problemas complejos dividiéndolos en subproblemas más simples. A diferencia de la estrategia de "Dividir y Conquistar", se aplica específicamente cuando estos subproblemas se solapan o se repiten.

Algoritmo de Floyd-Warshall: Para encontrar los caminos mínimos entre todos los pares de nodos en un grafo.
Cálculo de la Sucesión de Fibonacci: En su versión optimizada para evitar la explosión exponencial de llamadas recursivas.

```
Numeracion(v: nodo, var nd: int){
    nd = nd + 1;
    v.numero = nd;
    for (nodo w: adyacentes a v) {
        if (w.numero == 0) { // el nodo NO fue visitado
            Numeracion(w, nd);
        }
    }
}

DFS(g: grafo) {
    int nd = 0; // contador global
    for (nodo v: vertice de g) {
        // Marcarlo como NO visitado
        v.numero = 0;
    }

    for (nodo v: vertice de g) {
        if (v.numero == 0) {
            Numeracion(v, nd);
        }
    }
```

recorrido: [4, 2, 1, 3, 5, 6]
numeración: [0, 1, 2, 3, 4, 5]

a) ¿Es conexo? ¿Cuáles son sus componentes conexas?

El grafo no es fuertemente conexo, pero sí es débilmente conexo.
Es débilmente conexo porque su grafo subyacente (eliminando la dirección de las aristas) permite conectar cualquier par de nodos. No es fuertemente conexo debido a que no existe un camino orientado entre todos los pares de vértices; por ejemplo, el nodo 6 es una "fuente" (solo tiene aristas salientes), lo que impide que sea alcanzado desde cualquier otro nodo.
Hay dos componentes fuertemente conexas:
CF1 = {1, 2, 3, 4, 5}, CF2 = {6}.

c) ¿Es posible recorrerlo respetando las precedencias?

No, no es posible realizar un ordenamiento topológico que respete las precedencias.
Para que un grafo dirigido admita un orden topológico, debe ser un DAG (Grafo Dirigido Acíclico). Este grafo contiene, al menos, el ciclo cerrado entre los nodos 1, 3 y 2 (1→3→2→1), lo cual genera una dependencia circular que impide establecer una secuencia lineal de precedencia única3. b) Puntos de articulación (Vértices de corte)

Un punto de articulación o vértice de corte es un nodo tal que, al ser eliminado junto con todas sus aristas incidentes, provoca un incremento en el número de componentes conexas del grafo. En términos prácticos, su eliminación "desconecta" el grafo o una parte del mismo.

El nodo 5 es un punto de articulación: Si eliminamos el nodo 5, el nodo 6 quedaría totalmente aislado del resto de la estructura (suponiendo el análisis sobre el grafo subyacente para puntos de articulación en grafos no dirigidos).