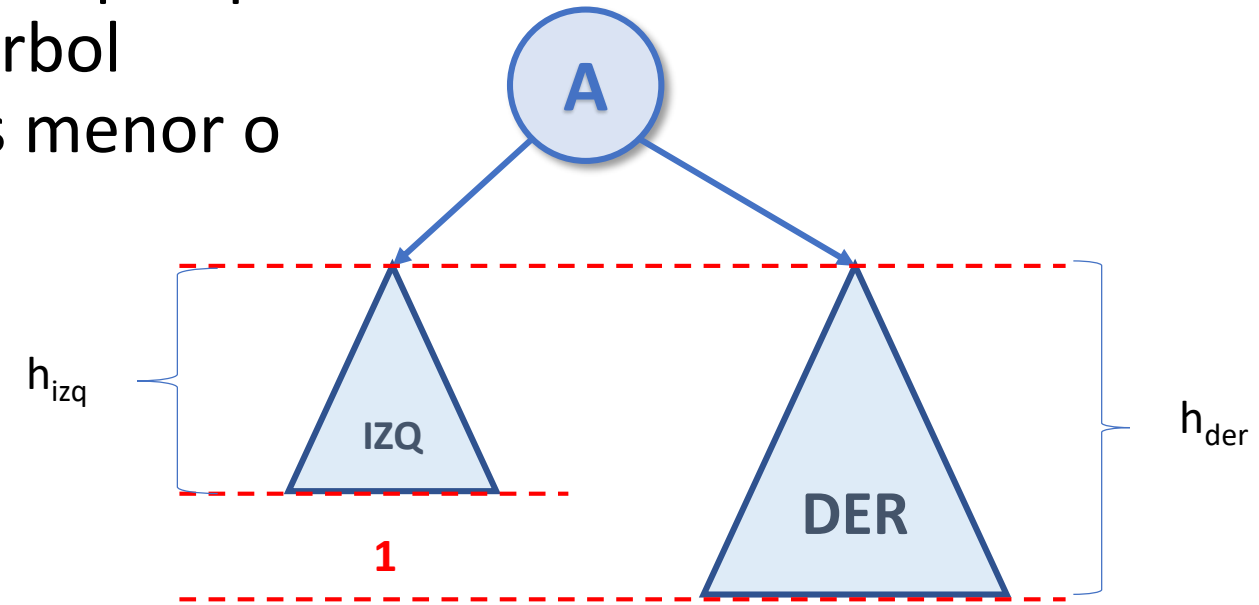
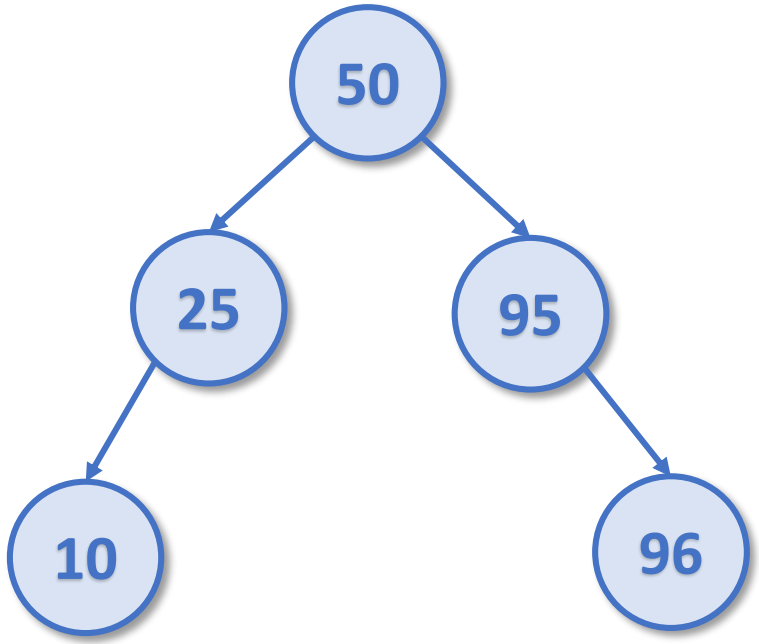
Árboles de Adelson-Velskii y Landis (AVL)

- Desarrollado por los soviéticos Georgii Adelson-Velskii y Yevgeni Landis en 1962.
- También se llaman árboles simplemente equilibrados
- Para cualquier nodo del árbol se cumple que la diferencia de alturas entre el subárbol izquierdo y el subárbol derecho es menor o igual a uno en valor absoluto
- $|h_{Izq} - h_{der}| <= 1$
  - h_{izq} → altura del subárbol izquierdo
  - h_{der} → altura del subárbol derecho

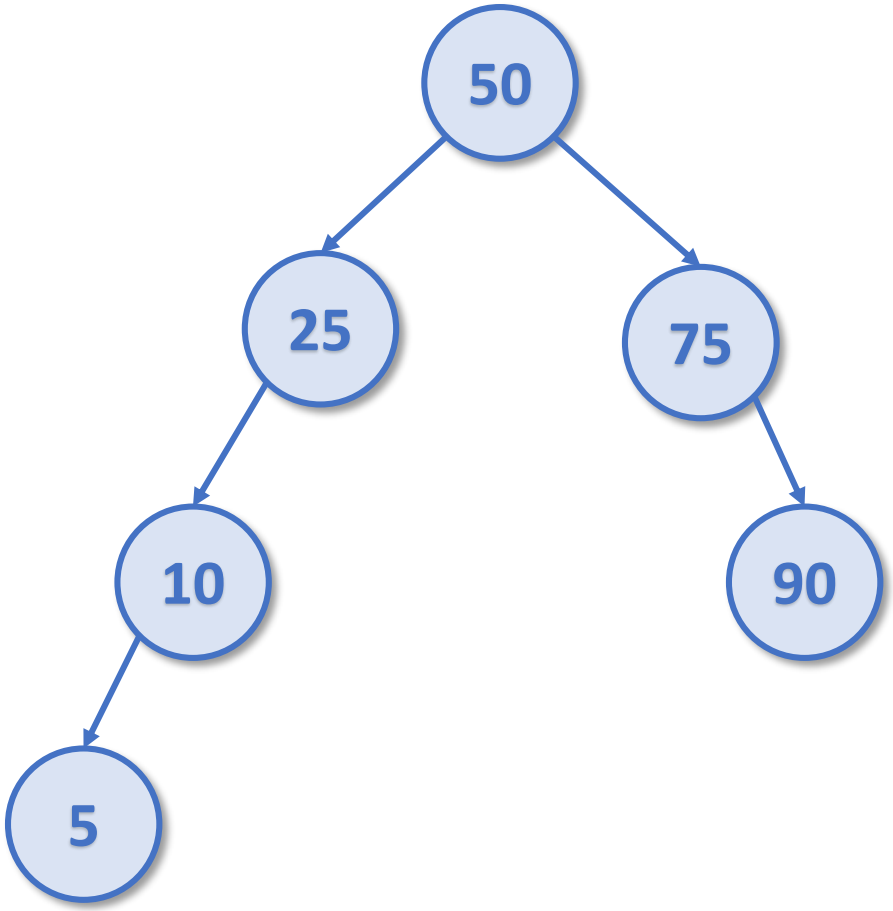


Ejercicios



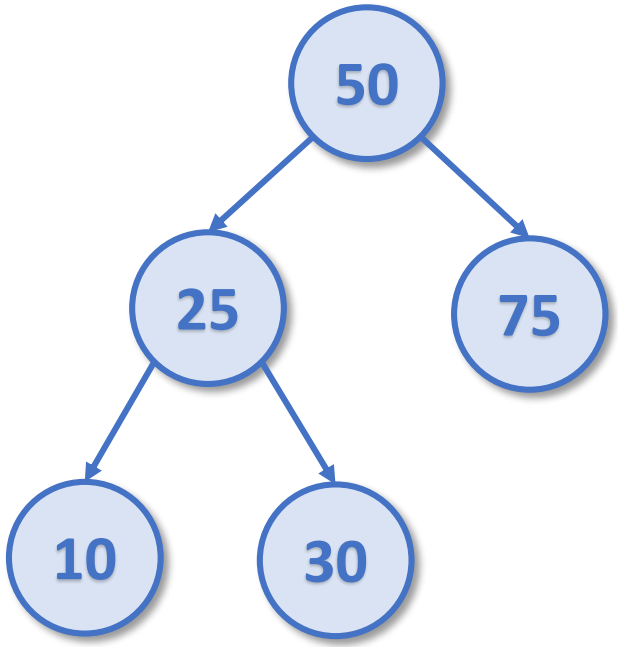
- ¿Altura mínima?
- ¿APE?
- ¿AVL?

Ejercicios



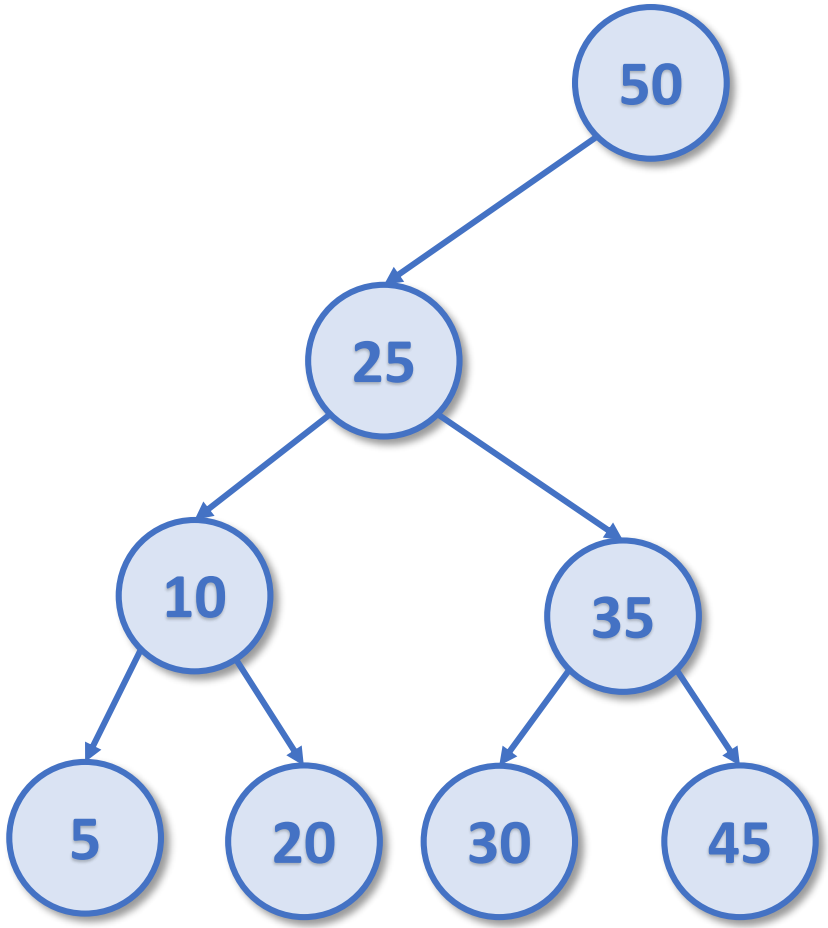
- ¿Altura mínima?
- ¿APE?
- ¿AVL?

Ejercicios



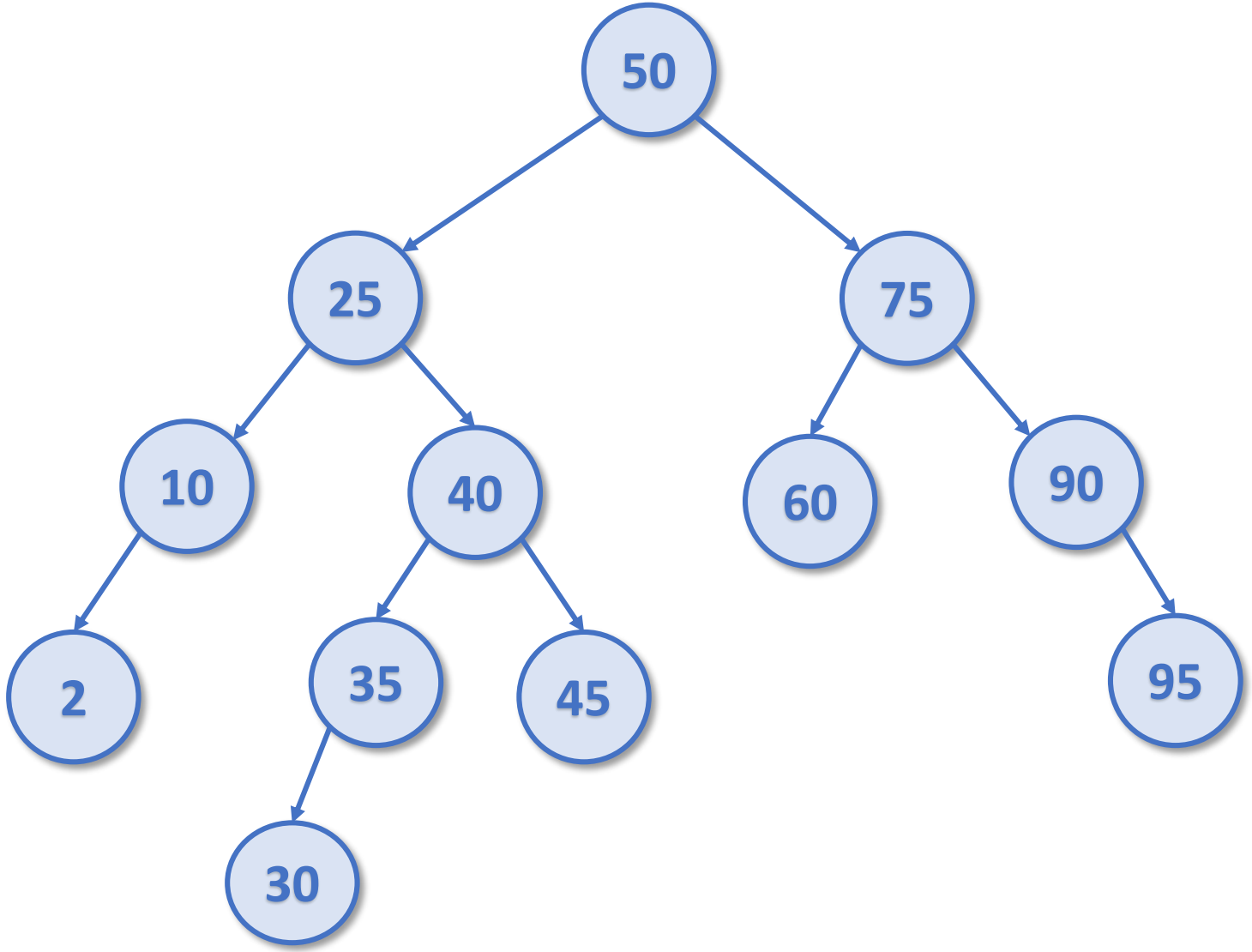
- ¿Altura mínima?
- ¿APE?
- ¿AVL?

Ejercicios



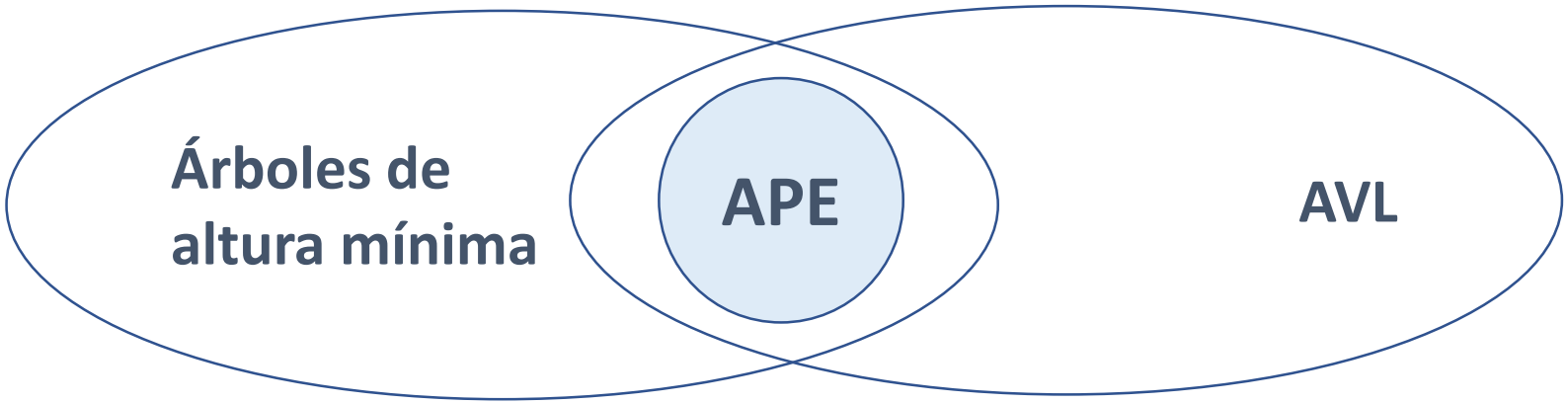
- ¿Altura mínima?
- ¿APE?
- ¿AVL?

Ejercicios



- ¿Altura mínima?
- ¿APE?
- ¿AVL?

Resumiendo



- Todo árbol APE es de altura mínima y AVL
- No todo árbol AVL es de altura mínima
- No todo árbol de altura mínima es AVL

Árboles de Fibonacci

- Surgen como respuesta a la siguiente pregunta
  - Si un árbol AVL no tiene porque ser de altura mínima ¿cuál es su altura máxima?
  - ¿Cuánto difiere la altura máxima de la altura mínima?
- Los árboles de ***Fibonacci*** se construyen de la mejor manera posible para alcanzar la altura máxima respetando la condición AVL
- Dependiendo de la altura h tenemos:
  - Si h=0 → árbol T_0 (vacío)
  - Si h=1 → árbol T_1 (de un solo nodo)
  - Si h>1 → árbol T_h (T_{h-1}, nodo, T_{h-2})

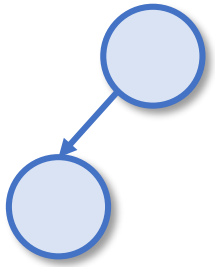
Fibonacci. Ejemplo

h=0

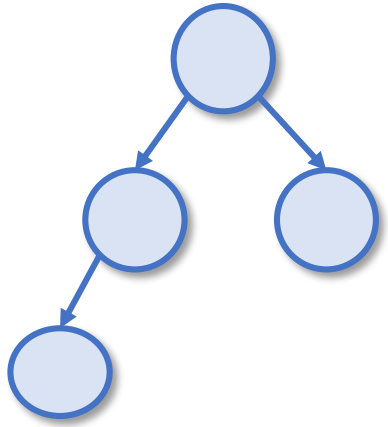
null

h=1

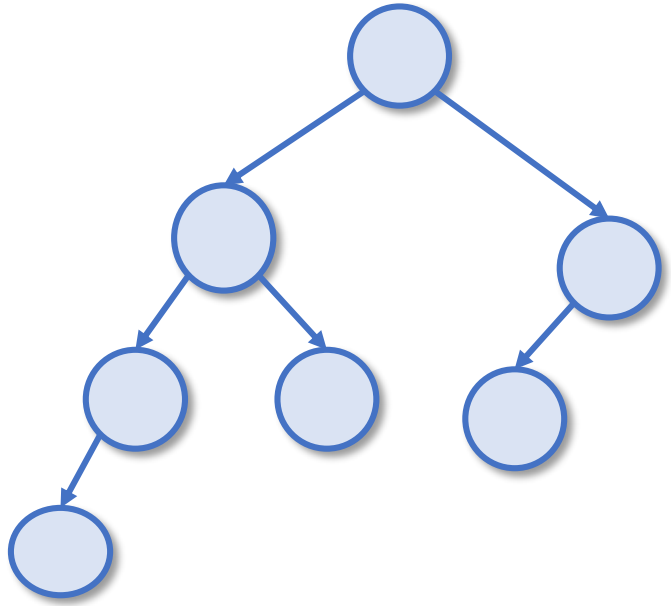
h=2 (h=1, nodo, h=0)



h=3 (h=2, nodo, h=1)



h=4 (h=3, nodo, h=2)



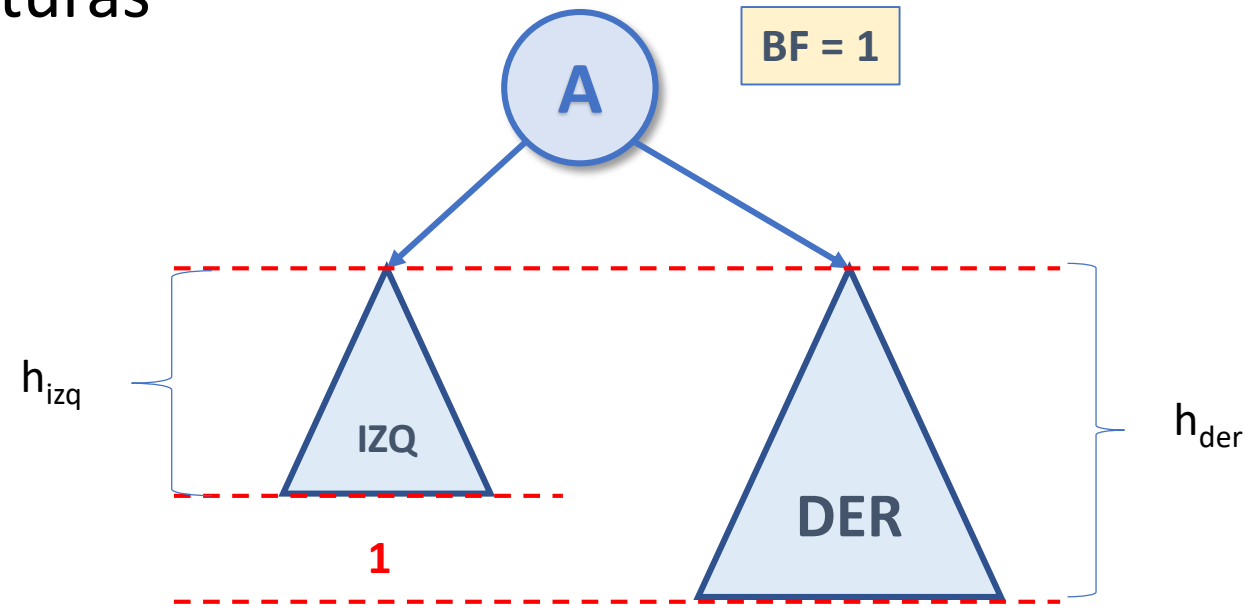
Resultados

- Cota para la altura máxima de un árbol de Fibonacci
  - $h_{MaxFib}(n) <= 1{,}44 \log_2 n$
- Rango de altura para un AVL
  - $h_{APE}(n) <= h_{AVL}(n) <= h_{MaxFib}(n)$
  - $\log_2 n <= h_{AVL}(n) <= 1{,}44 \log_2 n$
- Complejidad para las tres operacines
  - $O(\log_2 n) <= O(h_{AVL}(n)) <= O(1{,}44 \log_2 n)$

Implementation de un AVL

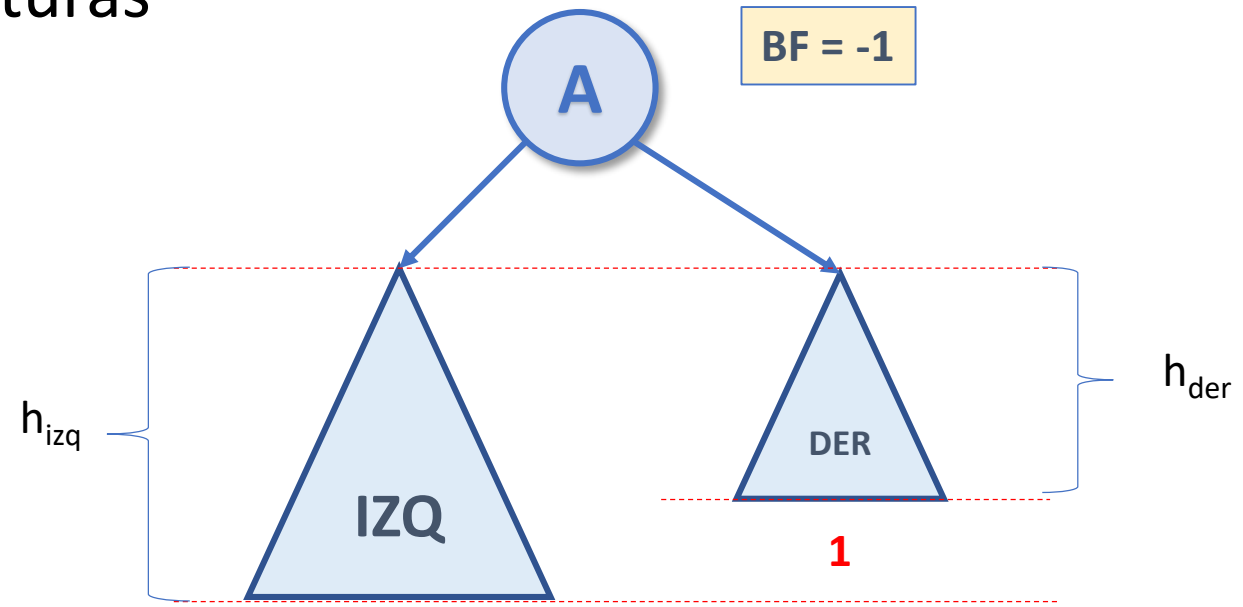
AVL. Factor de balance (BF)

- Indica si un árbol AVL esta equilibrado o no
- Se calcula como la diferencia de alturas entre el subárbol derecho y el izquierdo
  - $BF_n = h_{der} - h_{izq}$
- Situaciones posibles
  - $h_{izq} > h_{der}$ ($BF_n = -1$)
  - $h_{izq} = h_{der}$ ($BF_n = 0$)
  - $h_{izq} < h_{der}$ ($BF_n = 1$)
- Desequilibrio si $|BF_n| > 1$



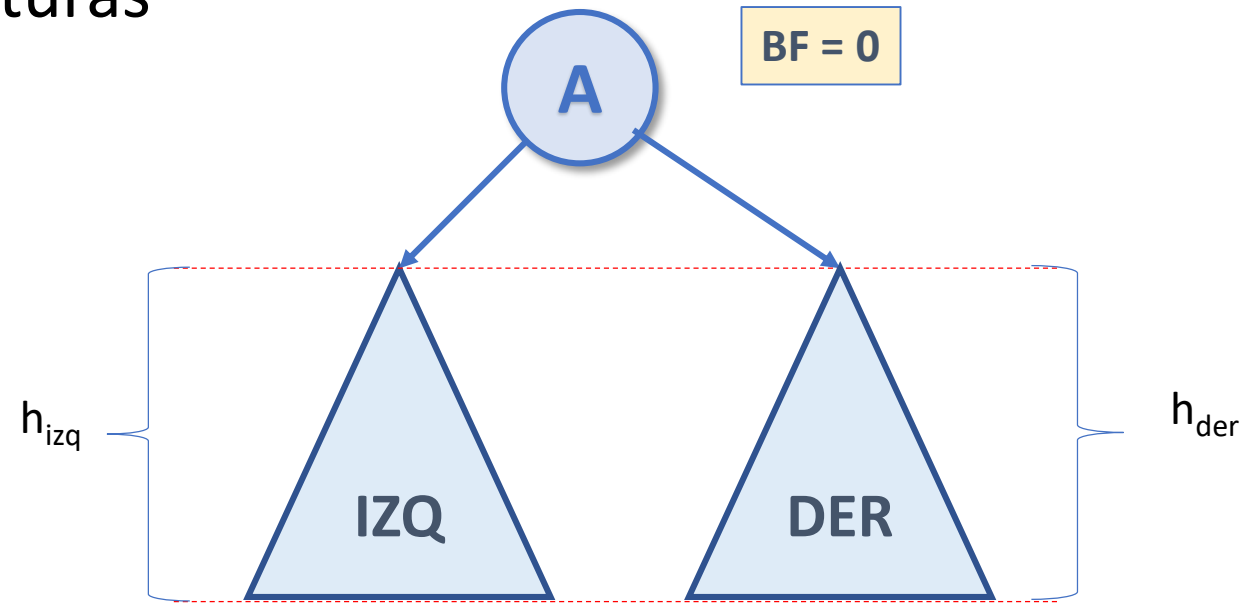
AVL. Factor de balance (BF)

- Indica si un árbol AVL esta equilibrado o no
- Se calcula como la diferencia de alturas entre el subárbol derecho y el izquierdo
  - $BF_n = h_{der} - h_{izq}$
- Situaciones posibles
  - $h_{izq} > h_{der}$ ($BF_n = -1$)
  - $h_{izq} = h_{der}$ ($BF_n = 0$)
  - $h_{izq} < h_{der}$ ($BF_n = 1$)
- Desequilibrio si $|BF_n| > 1$



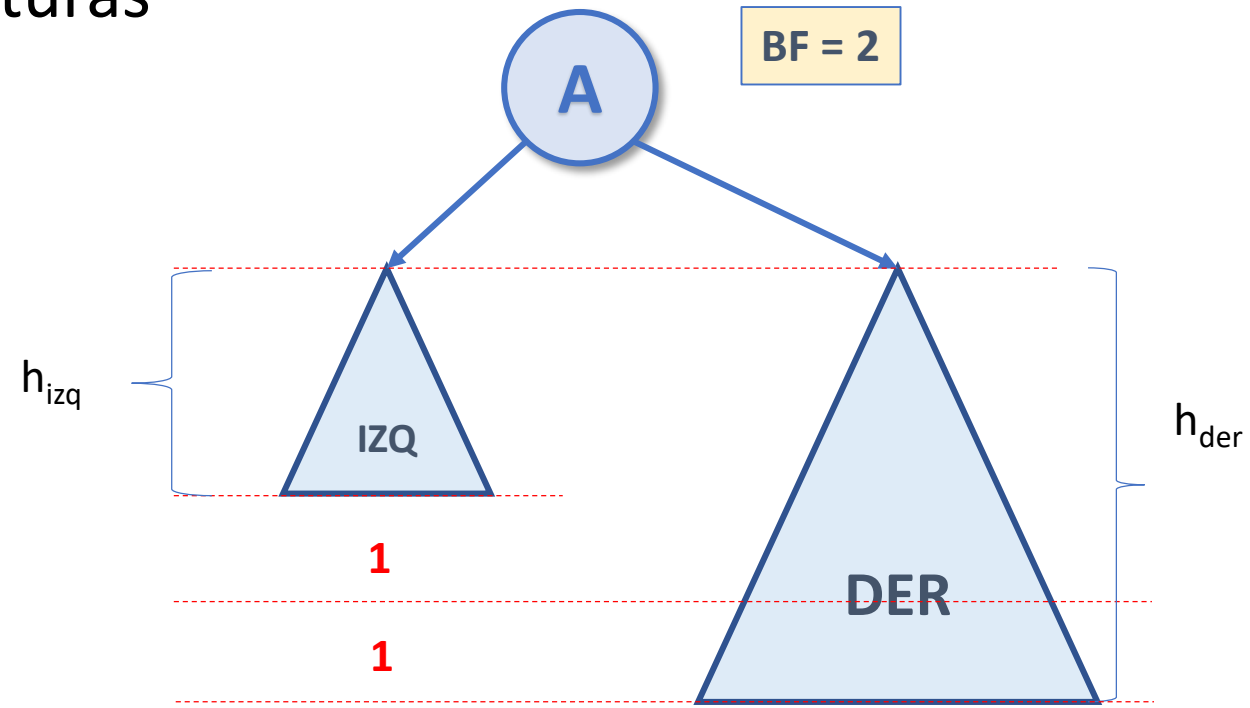
AVL. Factor de balance (BF)

- Indica si un árbol AVL esta equilibrado o no
- Se calcula como la diferencia de alturas entre el subárbol derecho y el izquierdo
  - $BF_n = h_{der} - h_{izq}$
- Situaciones posibles
  - $h_{izq} > h_{der}$ ($BF_n = -1$)
  - $h_{izq} = h_{der}$ ($BF_n = 0$)
  - $h_{izq} < h_{der}$ ($BF_n = 1$)
- Desequilibrio si $|BF_n| > 1$



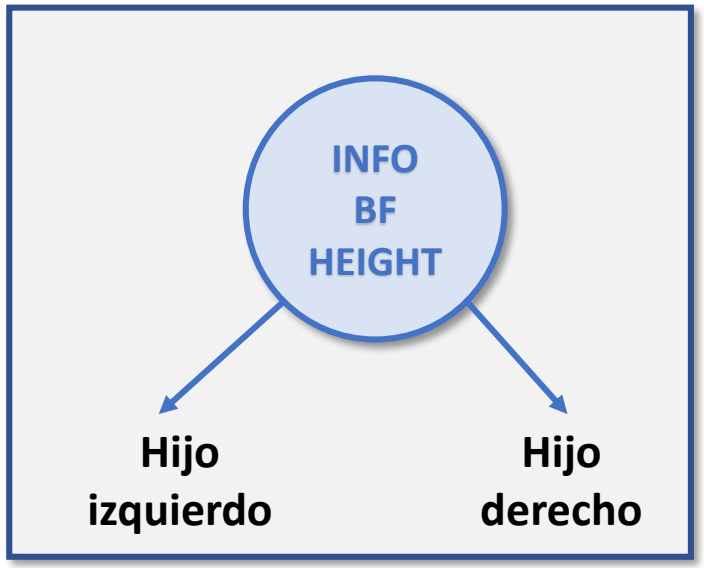
AVL. Factor de balance (BF)

- Indica si un árbol AVL esta equilibrado o no
- Se calcula como la diferencia de alturas entre el subárbol derecho y el izquierdo
  - $BF_n = h_{der} - h_{izq}$
- Situaciones posibles
  - $h_{izq} > h_{der}$ ($BF_n = -1$)
  - $h_{izq} = h_{der}$ ($BF_n = 0$)
  - $h_{izq} < h_{der}$ ($BF_n = 1$)
- Desequilibrio si $|BF_n| > 1$



La estructura de datos para definir un nodo

- Clase genérica que define un nodo de un árbol AVL con sus atributos y sus métodos
- Atributos que definen un nodo de un árbol
  - La **información** almacenada en el **nodo** que será de tipo genérico
  - La información sobre quien es su **hijo izquierdo**
  - La información sobre quien es su **hijo derecho**
  - El **factor de balance** del nodo
  - La **altura** del nodo
- Métodos. Todos aquellos necesarios para gestionar los atributos del nodo



La estructura de datos para definir un árbol

- Clase genérica de un árbol AVL con sus atributos y sus métodos
- Atributos que definen un árbol AVL
  - El nodo raíz del árbol
- Métodos básicos (recursividad)
  - Añadir → addNode
  - Buscar → findNode
  - Borrar → removeNode
  - Mostrar el árbol → toString
- Cualquier otro método que resulte útil

AVL. Añadir una clave a un árbol

- Si el árbol está vacío crea un nuevo nodo con la información nueva y termina
- Si la clave a insertar ya existe termina
- **Método recursivo**
  - Si la clave del nodo a insertar **es menor** que la clave del nodo actual entonces
    Si el subárbol izquierdo es null entonces
    crear un nodo con la clave y asignarlo por la izquierda
    Sino volver a llamar al método insertar recursivo pero esta vez con el subárbol **izquierdo**
  - Si la clave del nodo a insertar **es mayor** que la clave del nodo actual entonces
    Si el subárbol derecho es null entonces
    crear un nodo con la clave y asignarlo por la derecha
    Sino volver a llamar al método insertar recursivo pero esta vez con el subárbol **derecho**
- **¿Complejidad del algoritmo?**

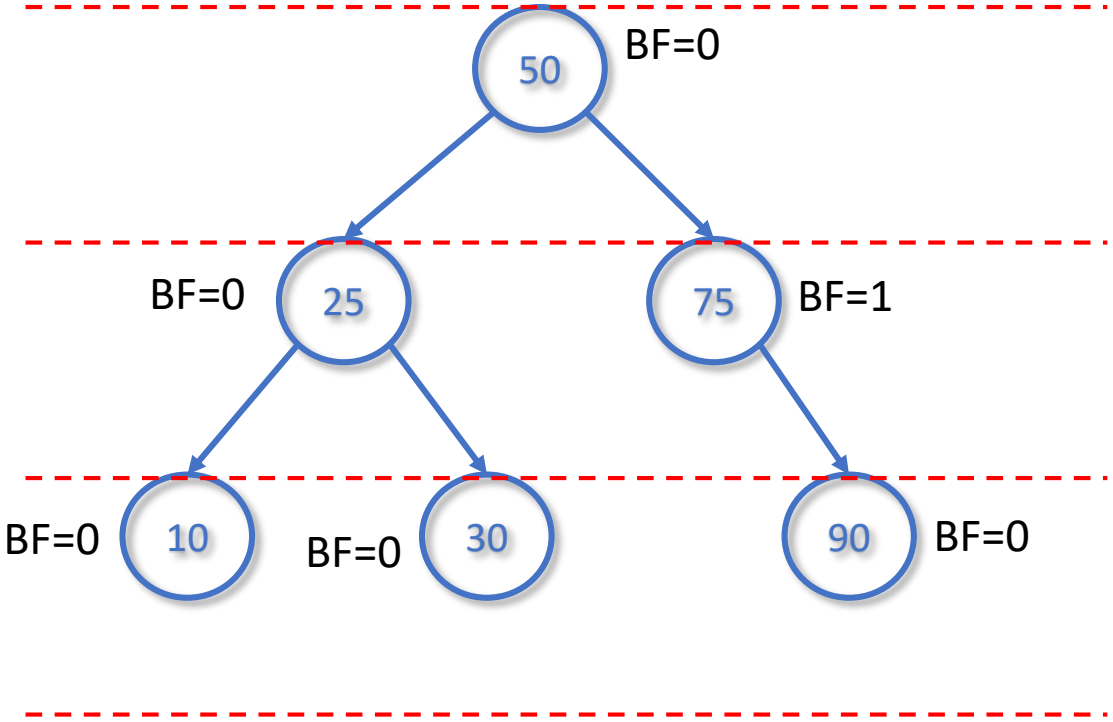
AVL. Añadir una clave a un árbol

- Diferencia con respecto a los árboles de búsqueda binarios
  - Hay que calcular el factor de balance y la altura de cada uno de los nodos que forman parte del camino de búsqueda para insertar
  - Esto se realiza al regresar de la recursividad
  - Si el $|FB_n| > 1$ para algún nodo **n** entonces hay que reequilibrar

AVL. Equilibrado del árbol

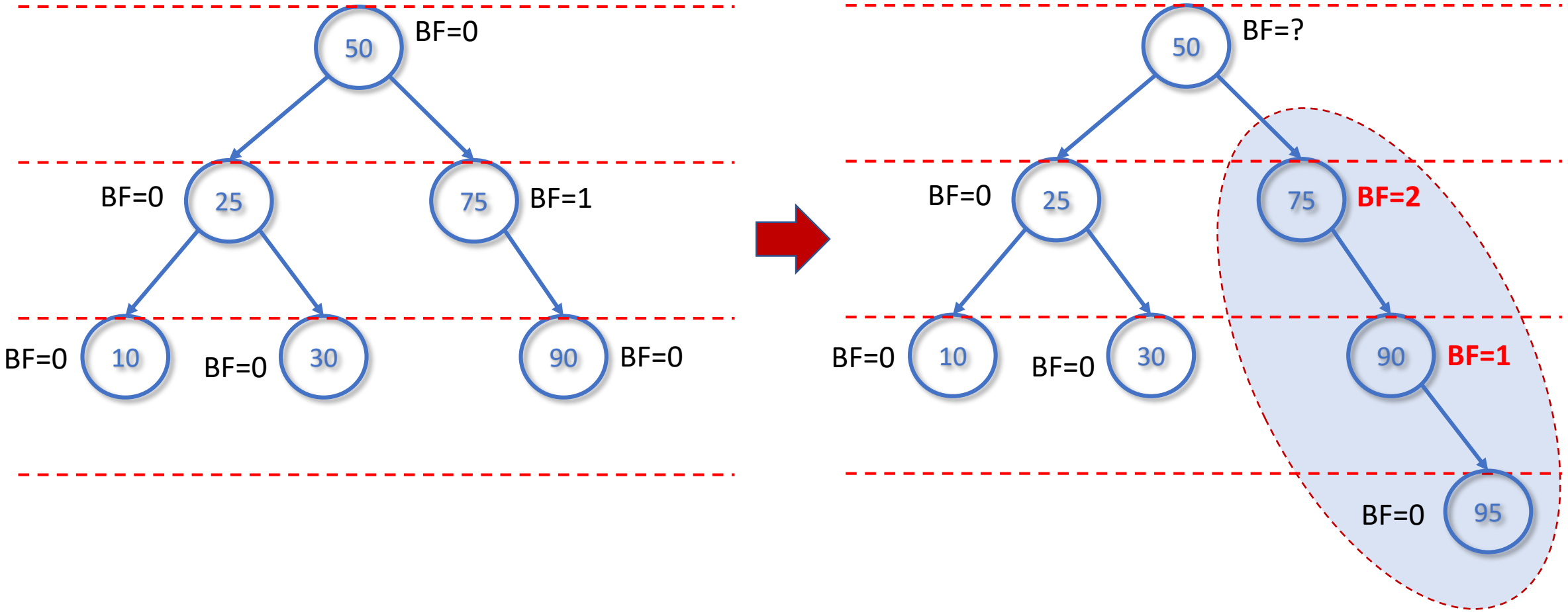
- Para el nodo a tratar hay que actualizar
  - Factor de balance (BF)
  - La altura
- Si el BF del nodo tiene un valor de -2 → rotación izquierda
  - Si el BF del subárbol izquierdo del nodo tiene un valor de 1 → rotación doble
  - En otro caso → rotación simple
- Si el BF del nodo tiene un valor de 2 → rotación derecha
  - Si el BF del subárbol derecho del nodo tiene un valor de -1 → rotación doble
  - En otro caso → rotación simple
- Al finalizar → devolver el nodo actualizado

AVL. Rotación simple derecha (RSD)

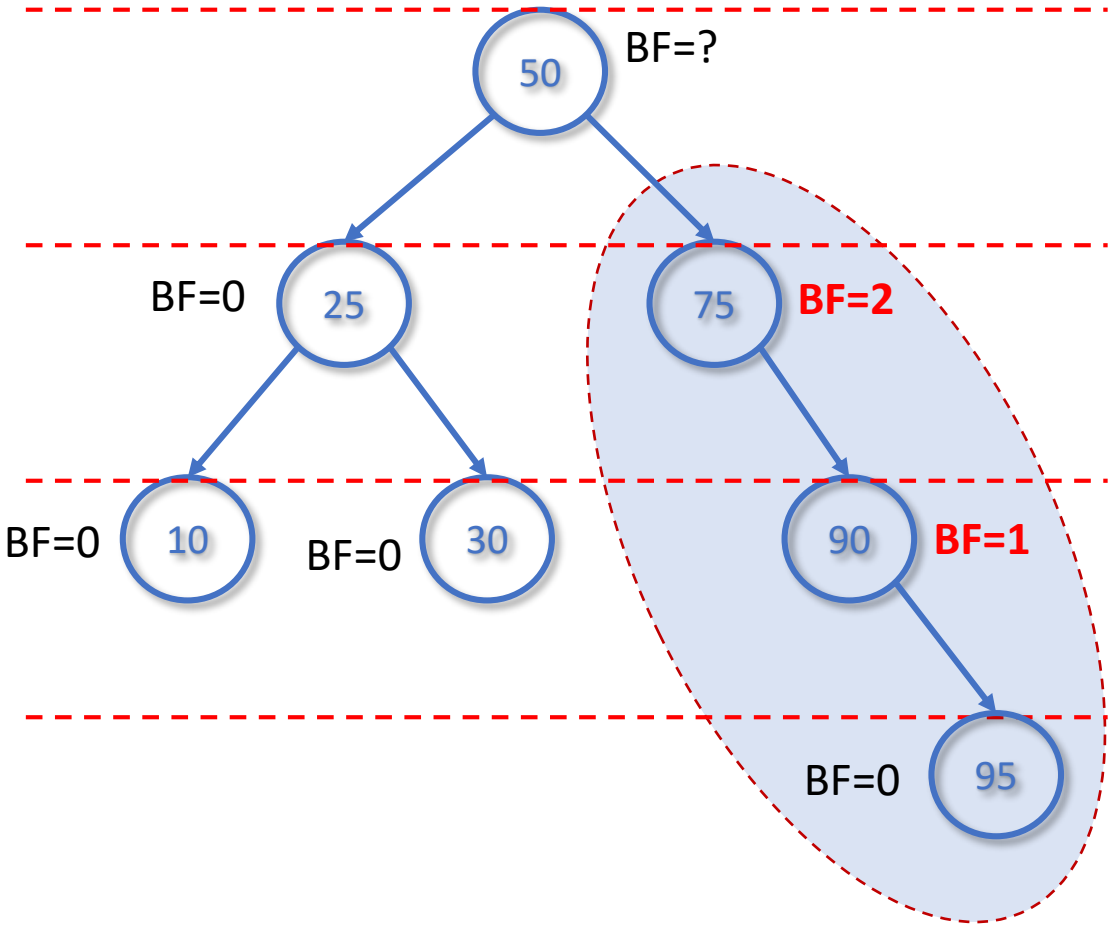


Añadir la clave 95

AVL. Rotación simple derecha (RSD)



AVL. Rotación simple derecha (RSD)



RSD sobre el nodo 75

Se utiliza un nodo auxiliar **aux**

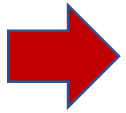
Para cualquier nodo N

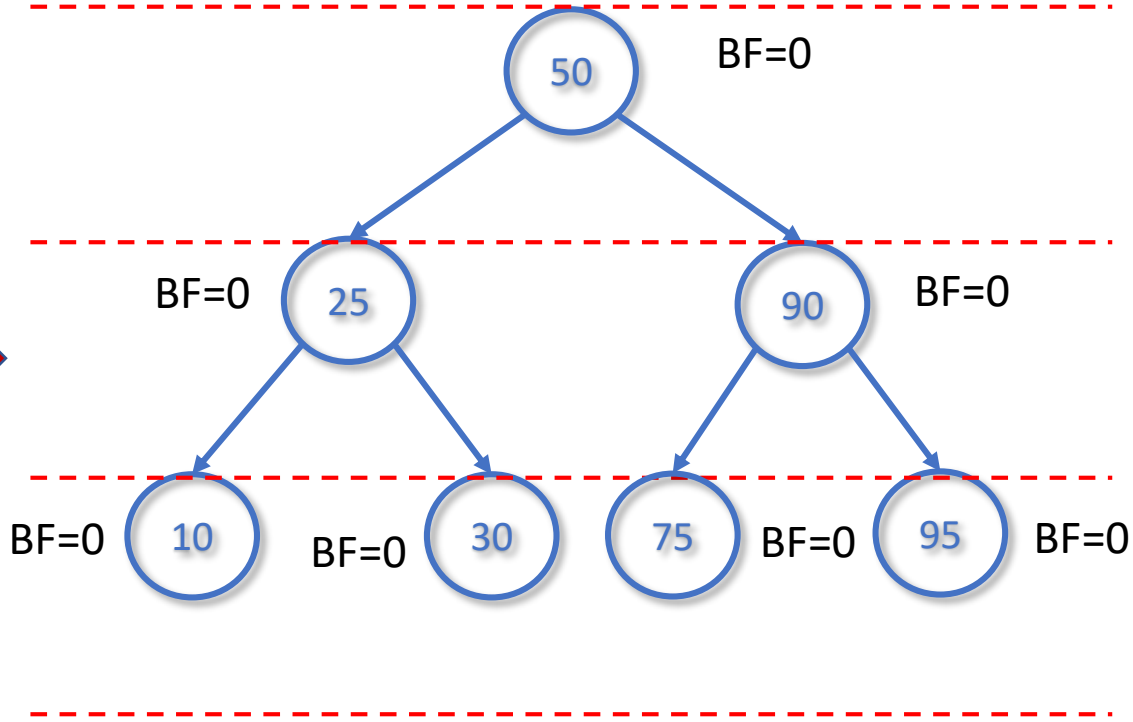
- **aux** = subárbol derecho del nodo N
- nodo N por la derecha = subárbol izquierdo de **aux**
- **aux** por la izquierda = nodo N

AVL. Rotación simple derecha (RSD)

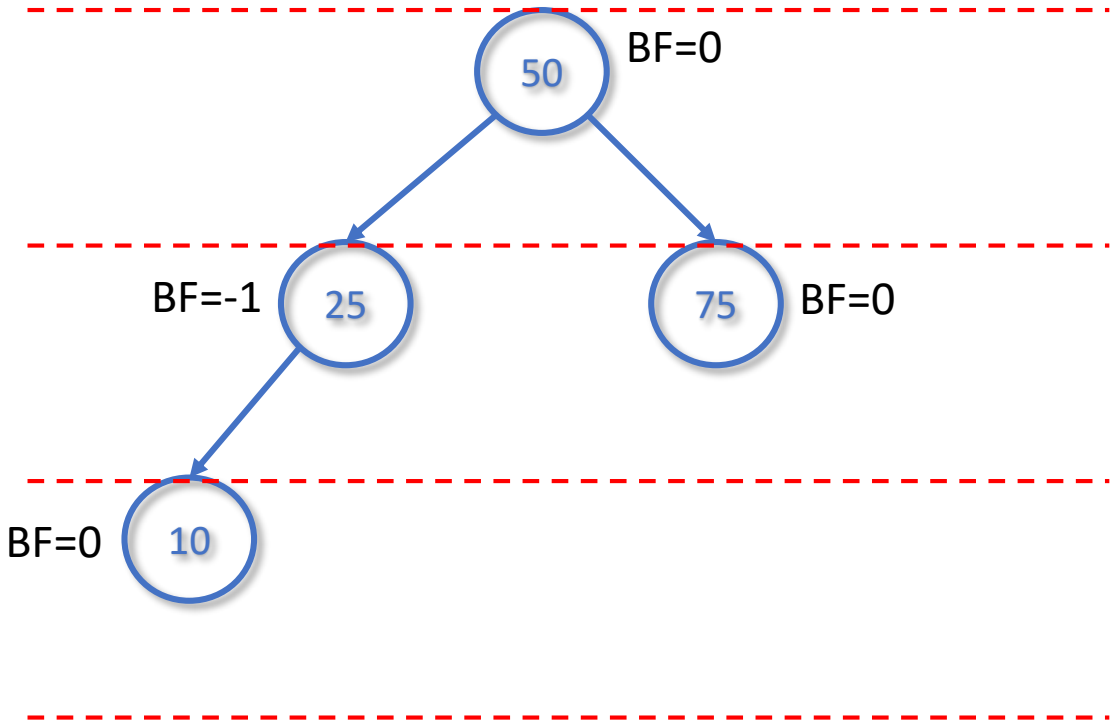
RSD sobre el nodo 75
Se utiliza un nodo auxiliar **aux**

Para cualquier nodo N
- **aux** = subárbol derecho del nodo N
- nodo N por la derecha = subárbol izquierdo de **aux**
- **aux** por la izquierda = nodo N



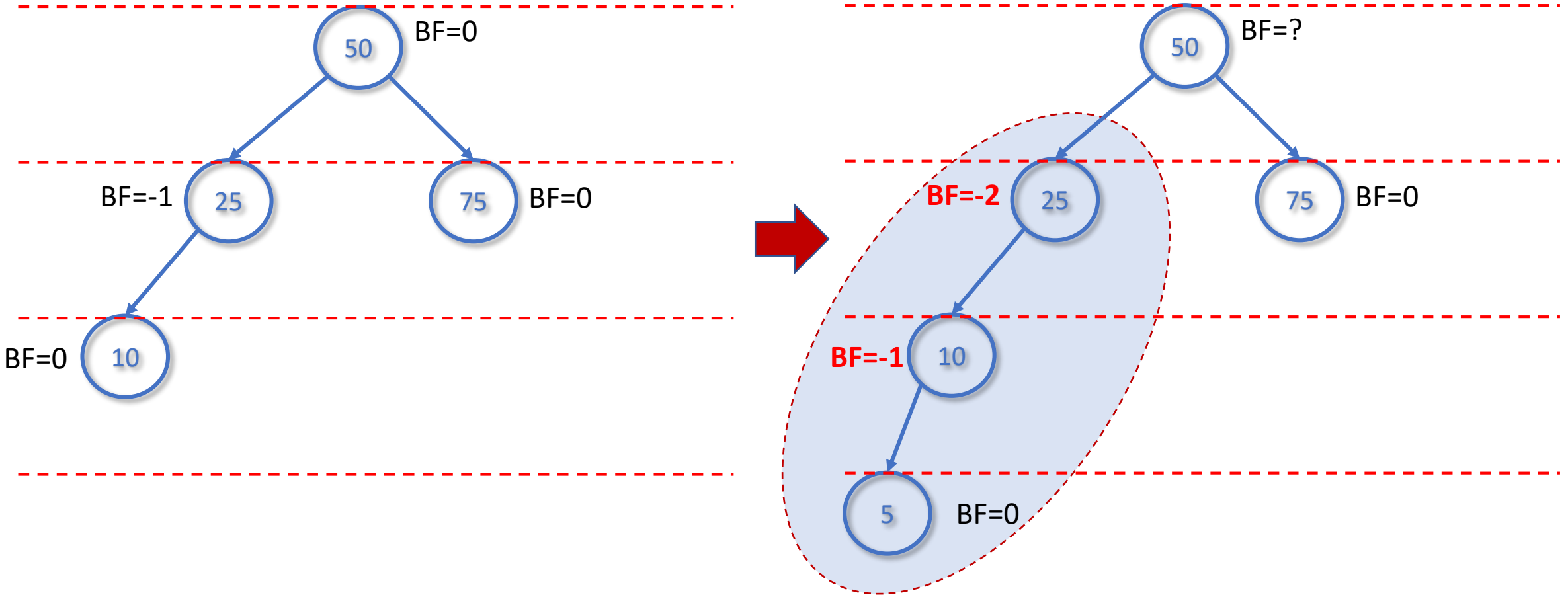


AVL. Rotación simple izquierda (RSI)

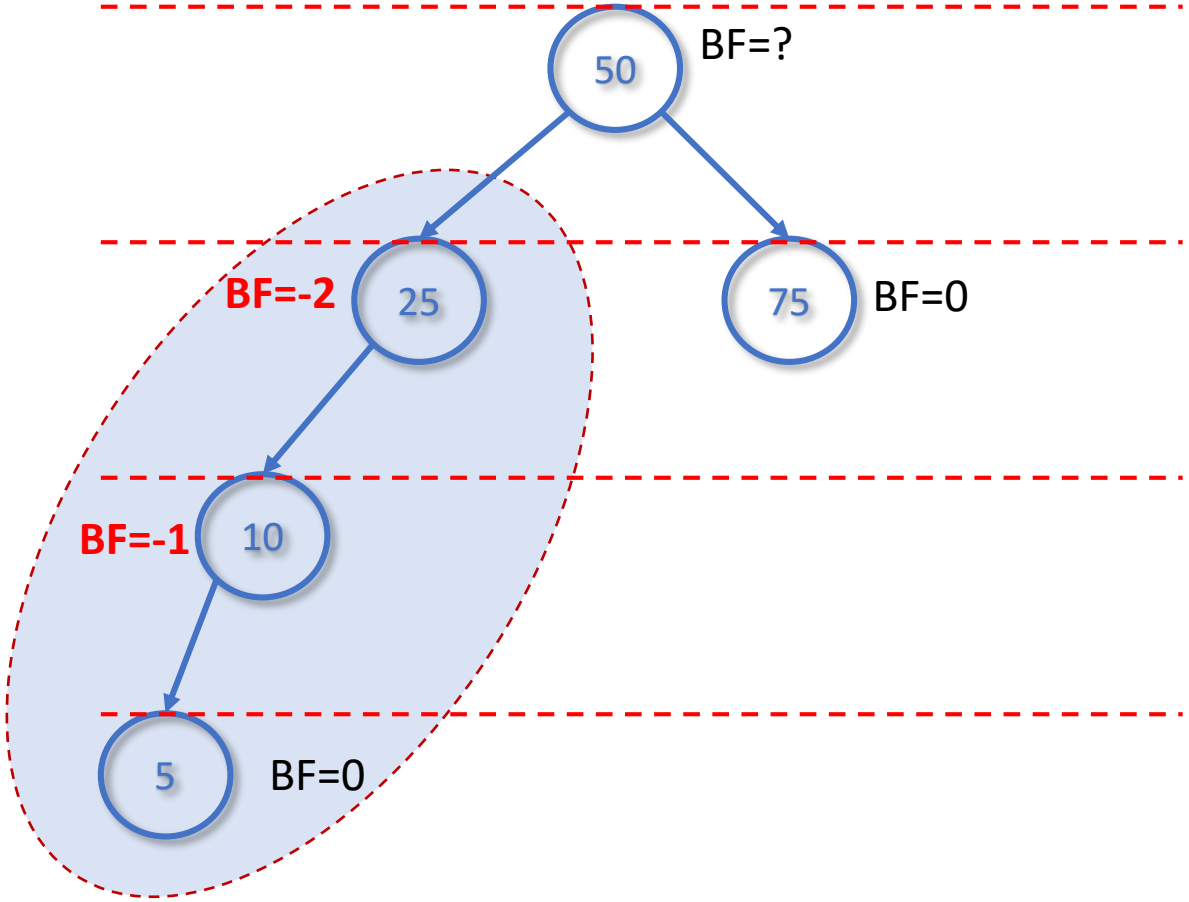


Añadir la clave 5

AVL. Rotación simple izquierda (RSI)



AVL. Rotación simple izquierda (RSI)



RSI sobre el nodo 25

Se utiliza un nodo auxiliar **aux**

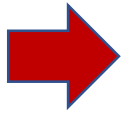
Para cualquier nodo N

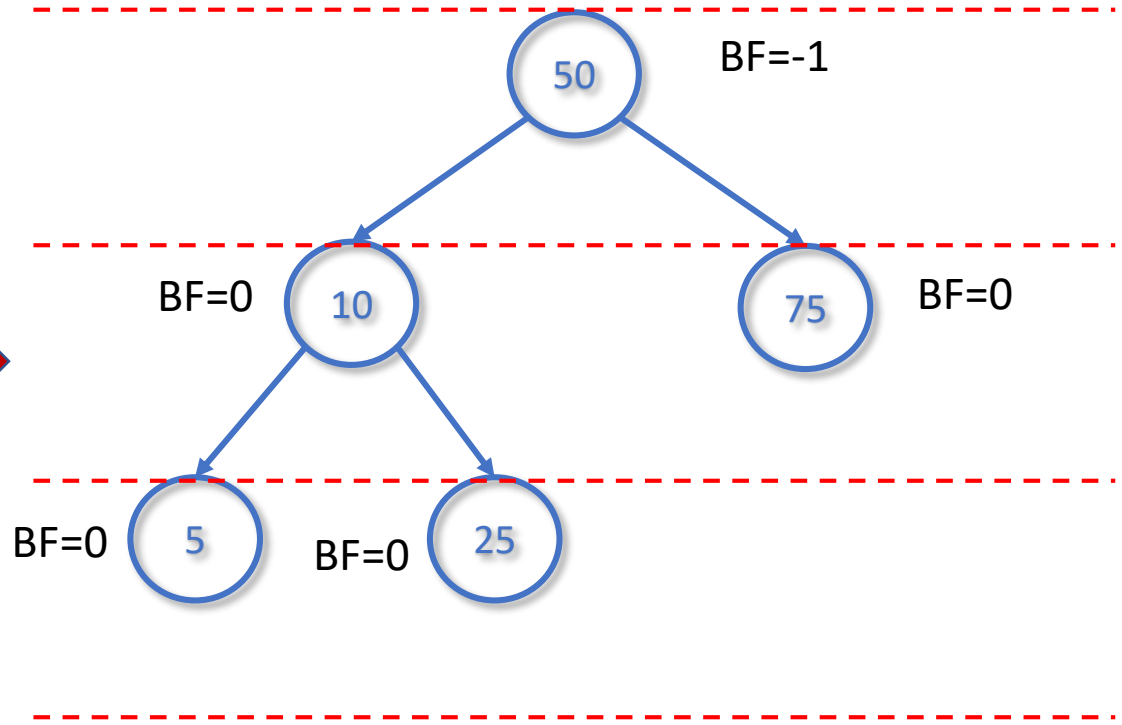
- **aux** = subárbol izquierdo del nodo N
- nodo N por la izquierda = subárbol derecho de **aux**
- **aux** por la derecha = nodo N

AVL. Rotación simple izquierda (RSI)

RSi sobre el nodo 75
Se utiliza un nodo auxiliar **aux**

Para cualquier nodo N
- **aux** = subárbol izquierdo del nodo N
- nodo N por la izquierdo = subárbol derecho de **aux**
- **aux** por la derecha = nodo N



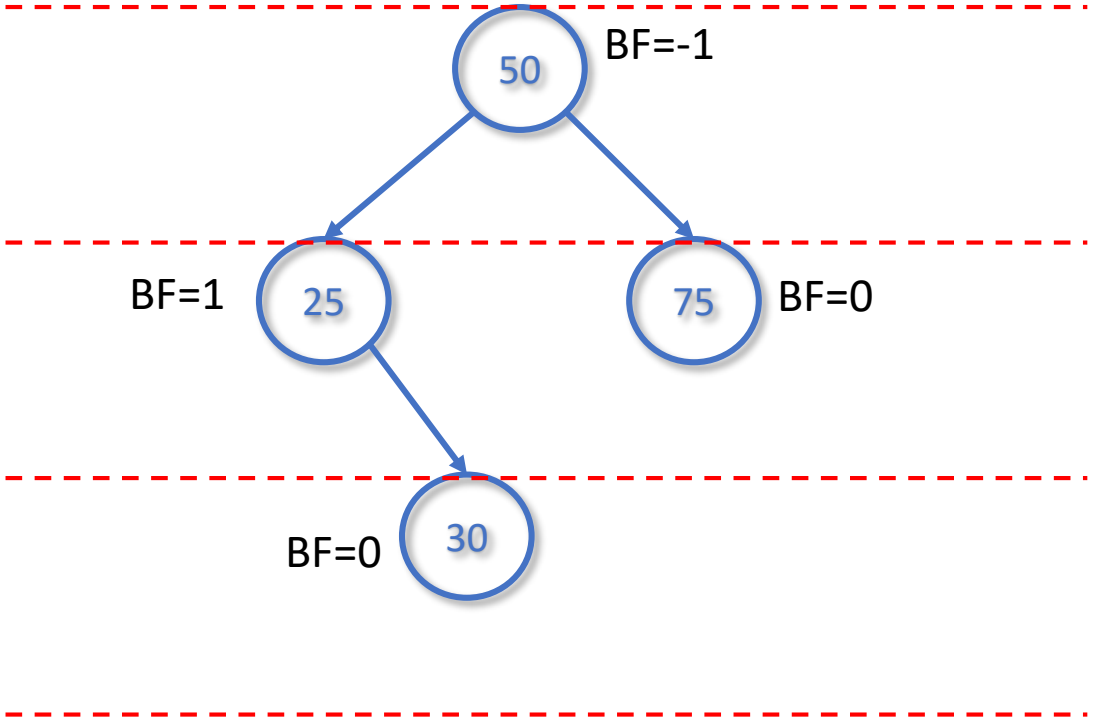


AVL. Rotación Simple. Ejercicios

- **Ejercicio1**
  - Partiendo de un AVL vacío insertar la secuencia de claves: 7, 6, 5, 4, 3, 2, 1

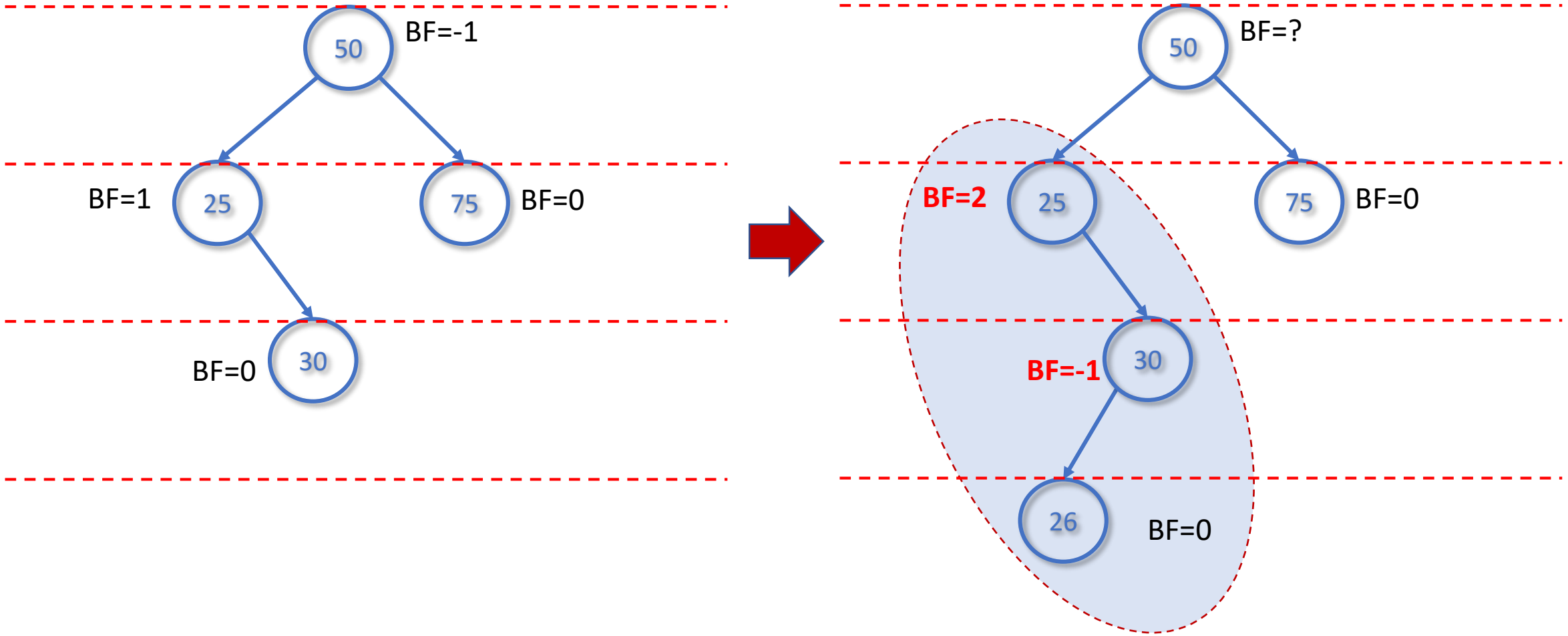
- **Ejercicio2**
  - Añadir 8, 9, 10
  - ¿Cuál es la complejidad temporal de cada inserción?

AVL. Rotación doble derecha (RDD)

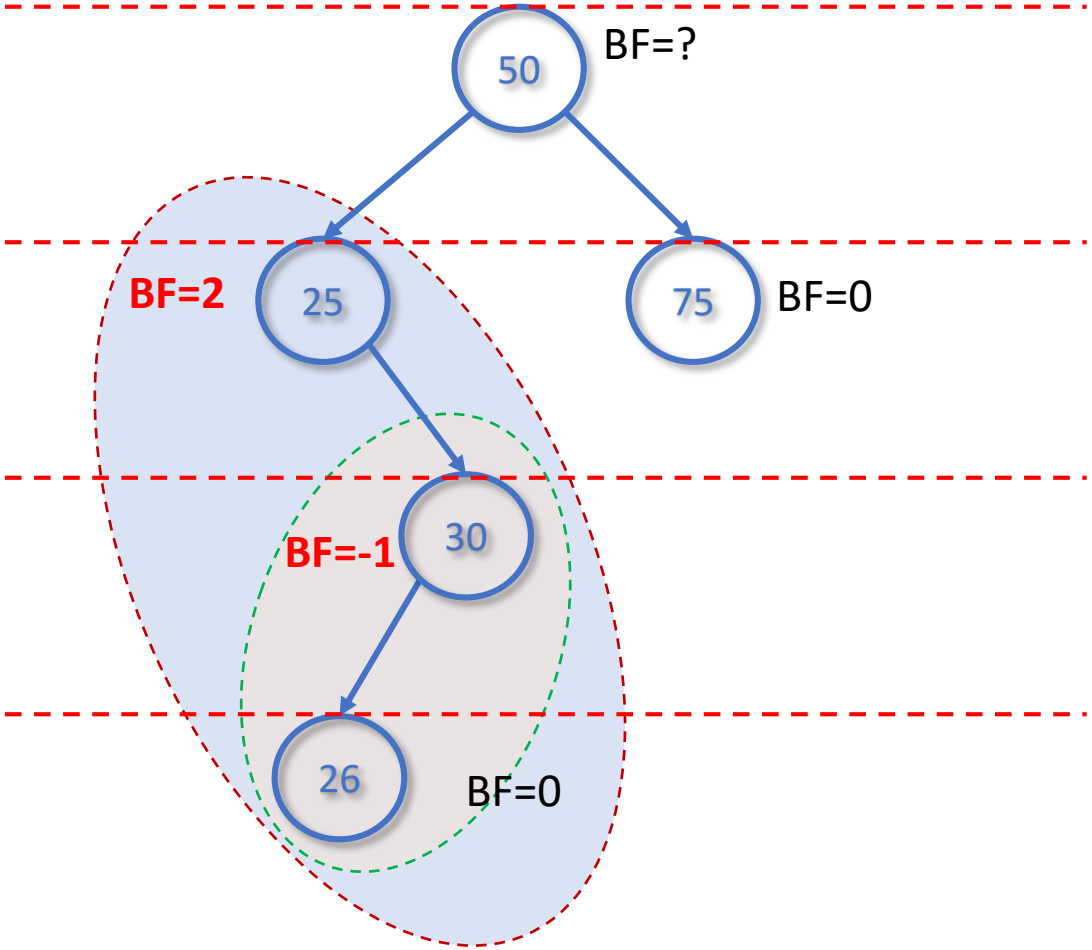


Añadir la clave 26

AVL. Rotación doble derecha (RDD)



AVL. Rotación doble derecha (RDD)



RDD sobre el nodo 25
Se soluciona con dos rotaciones simples

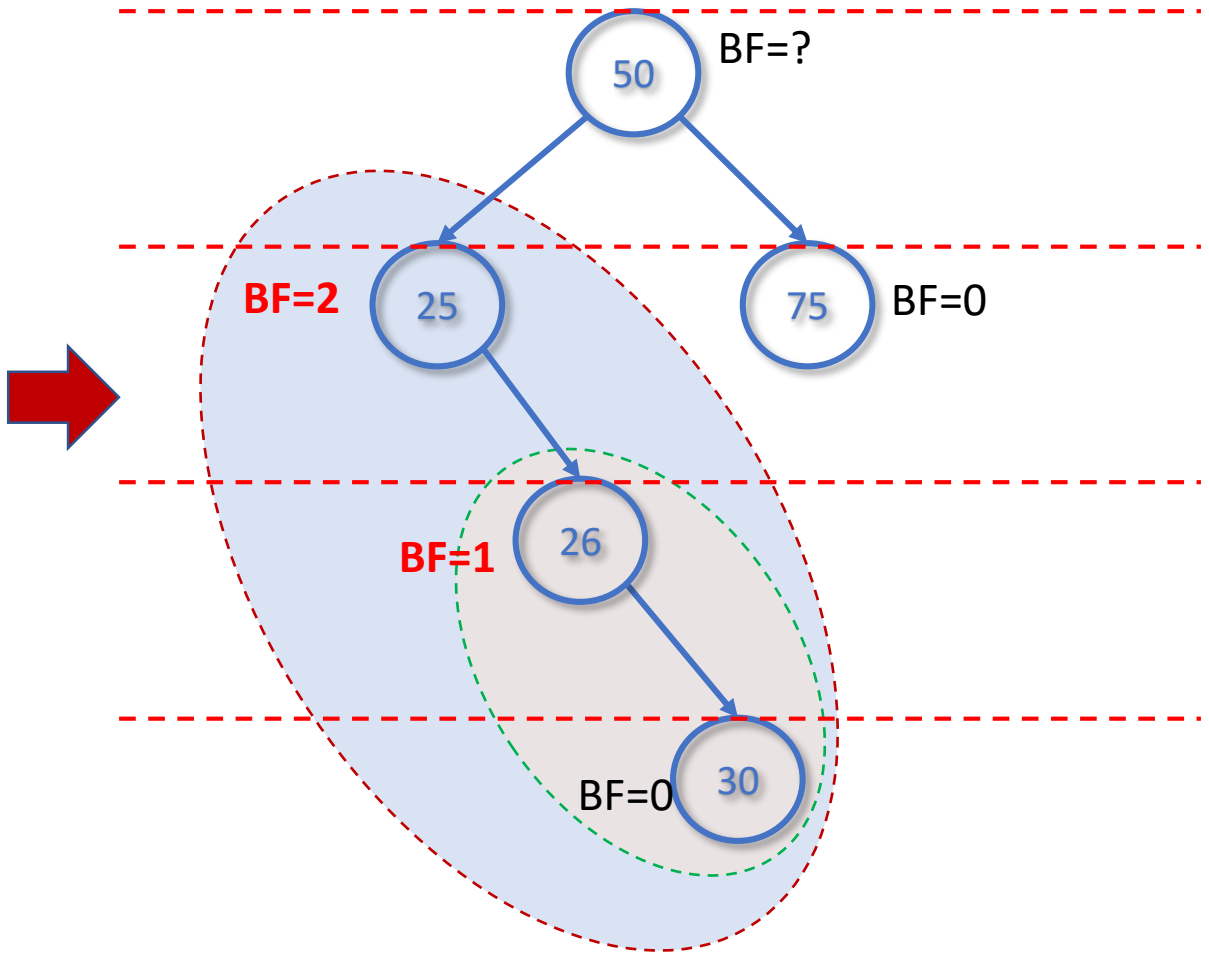
Para cualquier nodo N
- RSI sobre el subárbol derecho de N
- RSD sobre el nodo N

AVL. Rotación doble derecha (RDD)

RDD sobre el nodo 25
Se soluciona con dos rotaciones simples

Primera parte: RSI sobre el subárbol derecho de N

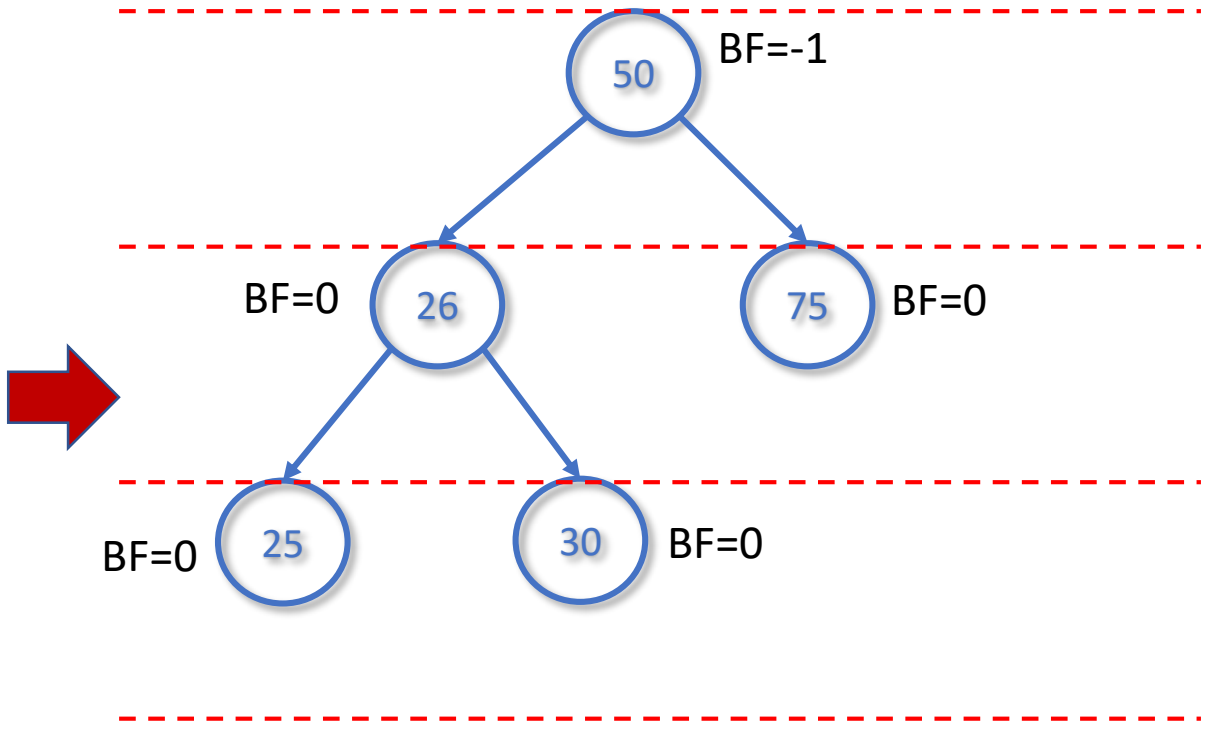
- **aux** = subárbol izquierdo del nodo N
- nodo N por la izquierdo = subárbol derecho de **aux**
- **aux** por la derecha = nodo N



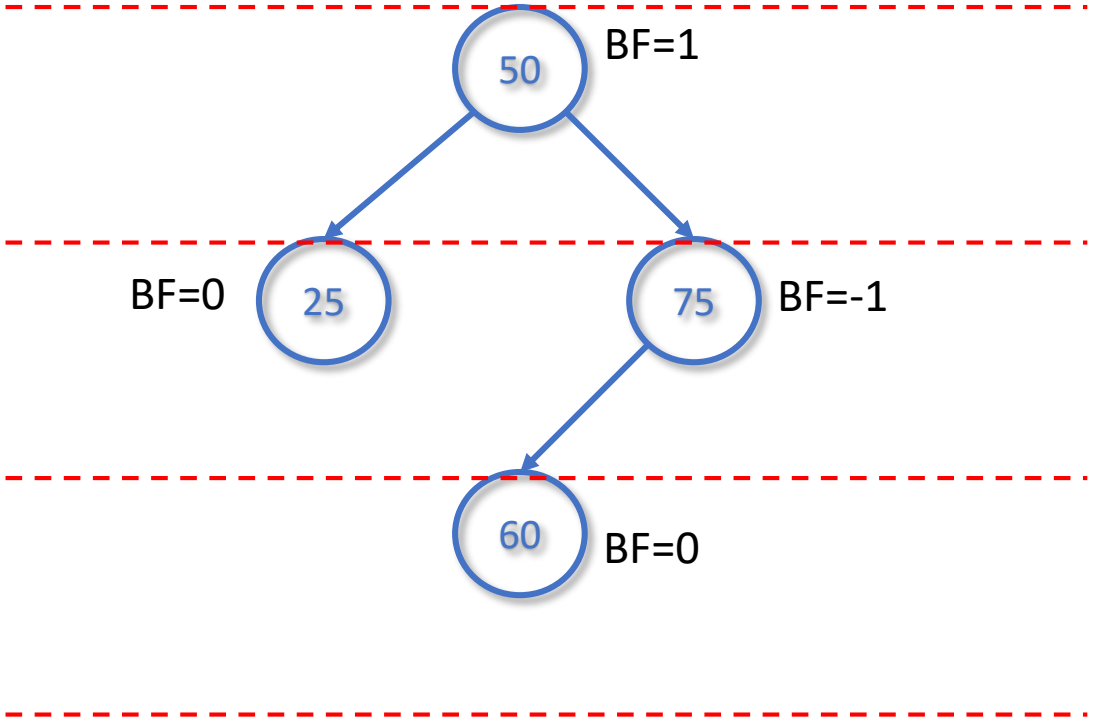
AVL. Rotación doble derecha (RDD)

RDD sobre el nodo 25
Se soluciona con dos rotaciones simples

Segunda parte: RSD sobre el nodo N
- **aux** = subárbol derecho del nodo N
- nodo N por la derecha = subárbol izquierdo de **aux**
- **aux** por la derecha = nodo N

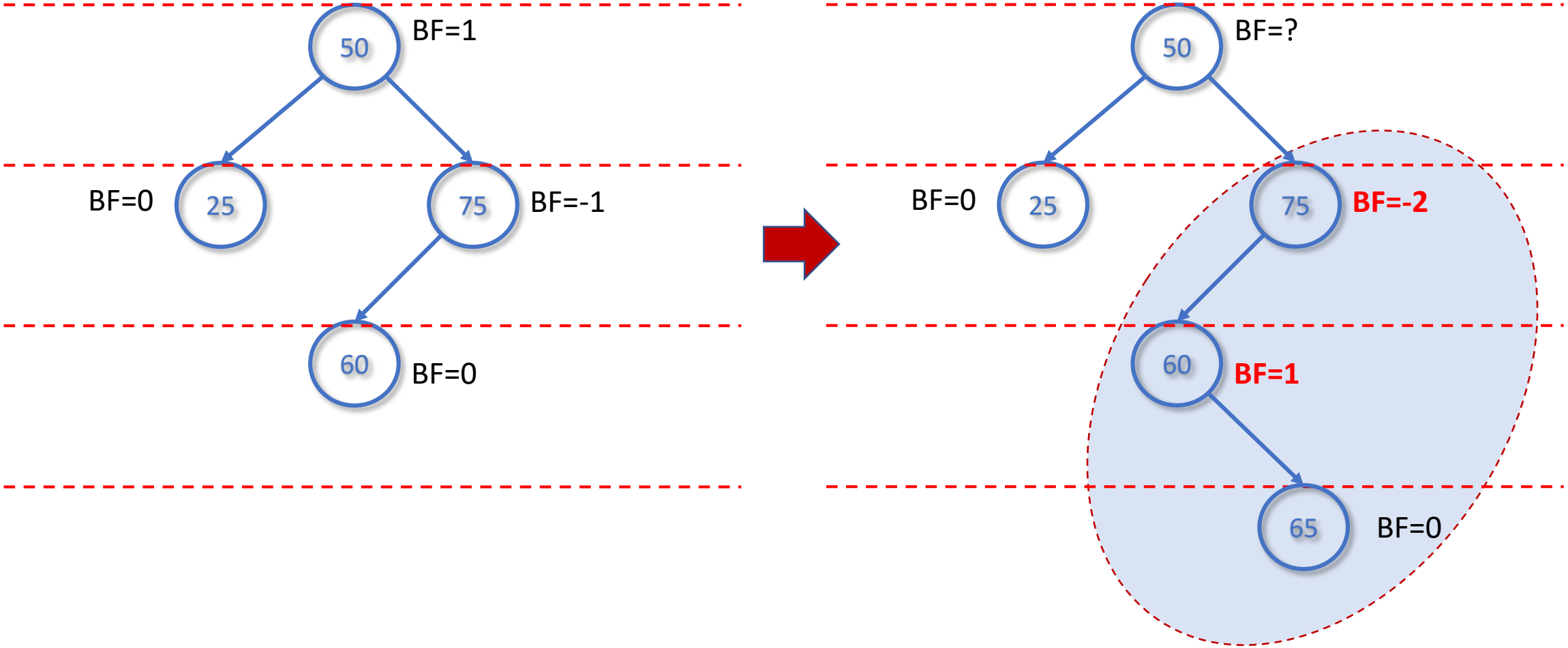


AVL. Rotación doble izquierda (RDI)

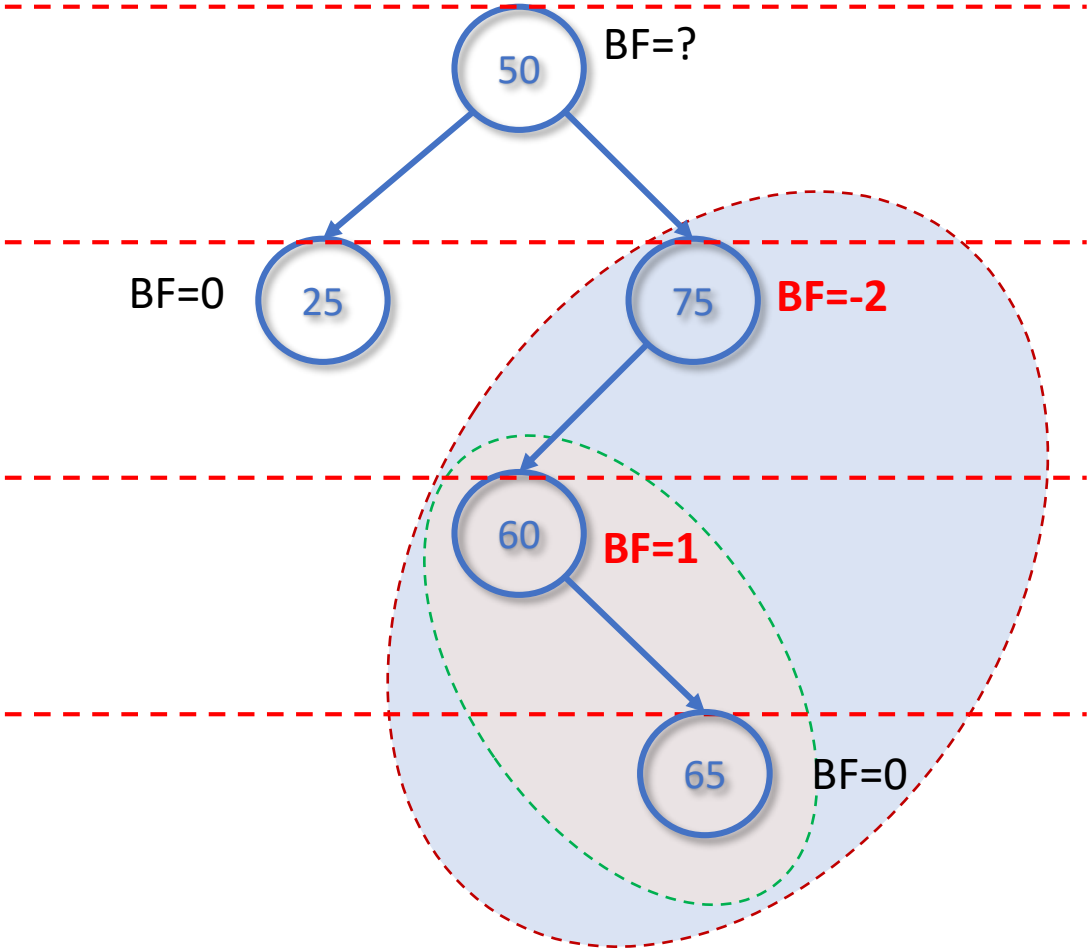


Añadir la clave 65

AVL. Rotación doble izquierda (RDI)



AVL. Rotación doble izquierda (RDI)



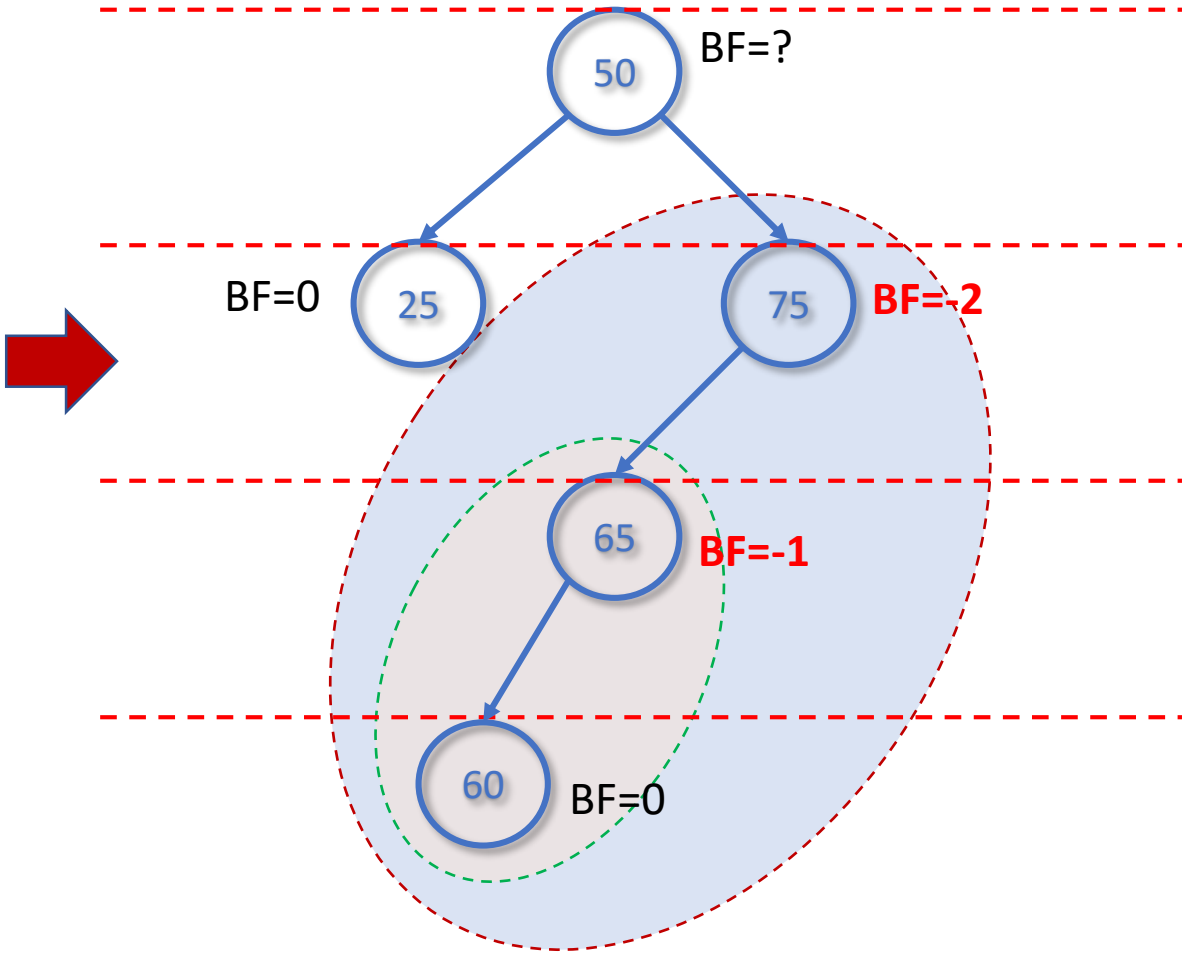
RDI sobre el nodo 75
Se soluciona con dos rotaciones simples

Para cualquier nodo N
- RSD sobre el subárbol izquierdo de N
- RSI sobre el nodo N

AVL. Rotación doble izquierda (RDI)

RDD sobre el nodo 75
Se soluciona con dos rotaciones simples

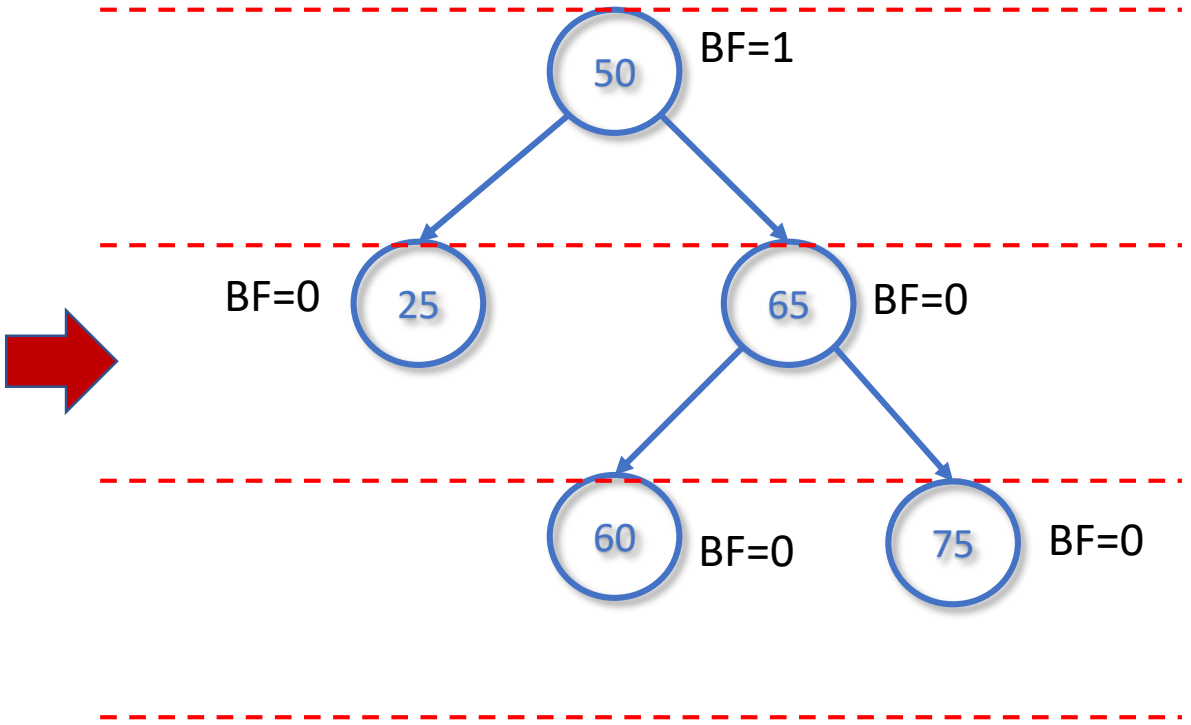
Primera parte: RSD sobre el subárbol izquierdo de N
- **aux** = subárbol derecho del nodo N
- nodo N por la derecha = subárbol izquierdo de **aux**
- **aux** por la izquierda = nodo N



AVL. Rotación doble izquierda (RDI)

RDD sobre el nodo 75
Se soluciona con dos rotaciones simples

Segunda parte: RSI sobre el nodo N
- **aux** = subárbol izquierdo del nodo N
- nodo N por la izquierda = subárbol derecho de **aux**
- **aux** por la izquierda = nodo N



AVL. Rotación Doble. Ejercicios

- **Ejercicio1**
  - Partiendo de un AVL vacío insertar la secuencia de claves: 1, 2, 3, 4, 5, 6, 10, 11, 8, 7

- **Ejercicio2**
  - Partiendo de un AVL vacío insertar la secuencia de claves: 5, 2, 10, 15, 12, 9, 7, 8, 6

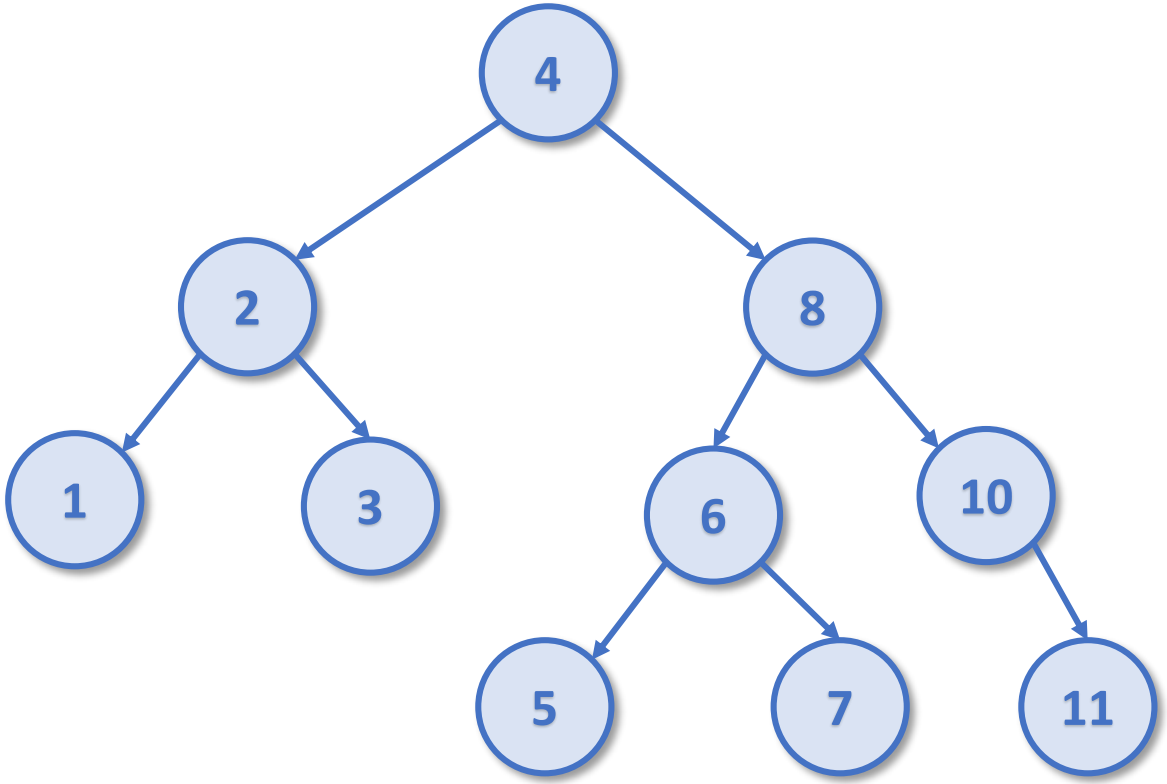
- **Ejercicio3**
  - ¿Cuál es la complejidad temporal de cada inserción?

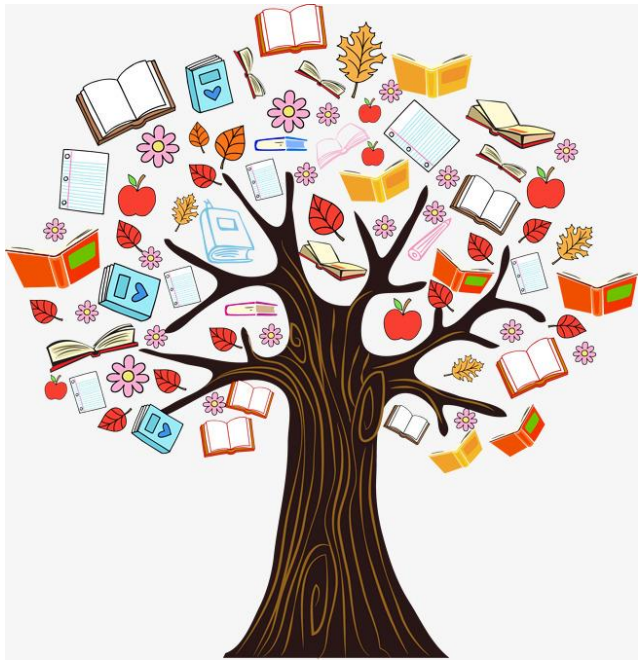
AVL. Borrar una clave a un árbol

- Realizar el borrado como se hacía en los árboles BST
- Si ha cambiado al altura del árbol
  - Recalcular el FB al regresar de la recursividad (actualizando los BF de los nodos que forman parte del camino de búsqueda)
  - Si $|BF_n| > 1$ para algún ***n*** entonces reequilibrar (detectando caso)
- En el borrado, el reequilibrado no es puntual
  - El reequilibrado de un subárbol **no garantiza** el equilibrio del árbol
  - A diferencia de la inserción, en el borrado es preciso **continuar reequilibrando siempre** hasta la raíz del árbol

AVL. Borrado. Ejercicios

- **Ejercicio1**
  - Partiendo del siguiente árbol AVL, borrar las siguientes claves: 1, 3, 4, 7, 11, 10

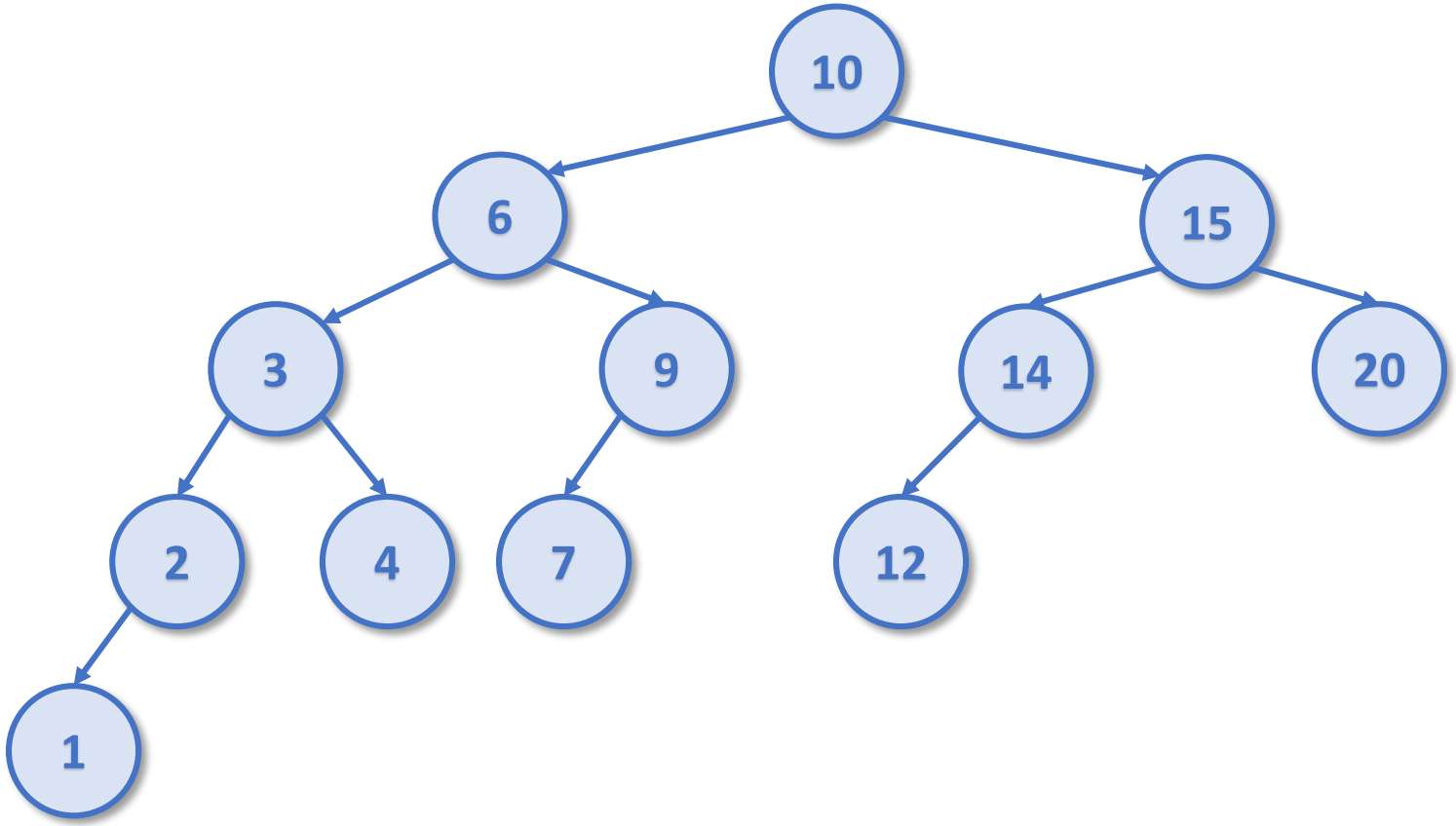




¿Cuál es la complejidad temporal de cada borrado?

AVL. Borrado. Ejercicios

- **Ejercicio2**
  - Partiendo del siguiente árbol AVL, borrar las siguientes claves: 20, 4, 10, 9, 6, 3



¿Cuál es la complejidad temporal de cada borrado?

AVL – Eficiencia

- Caso peor
  - El reequilibrado en un AVL solo afecta al camino de búsqueda
    - Su longitud es del orden de $\log_2 n$
  - $\text{Log}_2 n <= \text{Longitud del Camino de Búsqueda} <= 1{,}44\text{Log}_2 n$
- Resumiendo

MÉTODO	COMPLEJIDAD	
	APE	AVL
Insertar	O(n)	$O(\log_2 n)$
Buscar	$O(\log_2 n)$	$O(\log_2 n)$
Borrar	O(n)	$O(\log_2 n)$
Recorridos	O(n)	O(n)