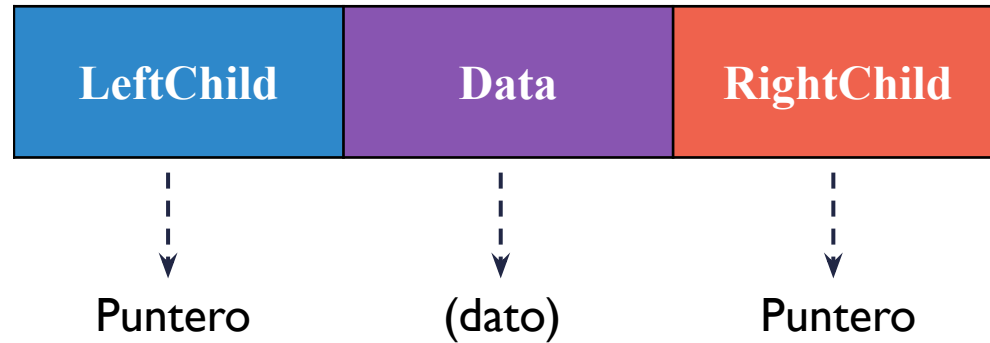


d) Árboles binarios enhebrados

Árboles binarios donde los punteros a NULL son reemplazados por “**hebras**” (*threads*) que apuntan al nodo sucesor y/o predecesor en un recorrido *inorden*.

d) Árboles binarios enhebrados

- Formato del nodo:



d) Árboles binarios enhebrados

- Formato del nodo:

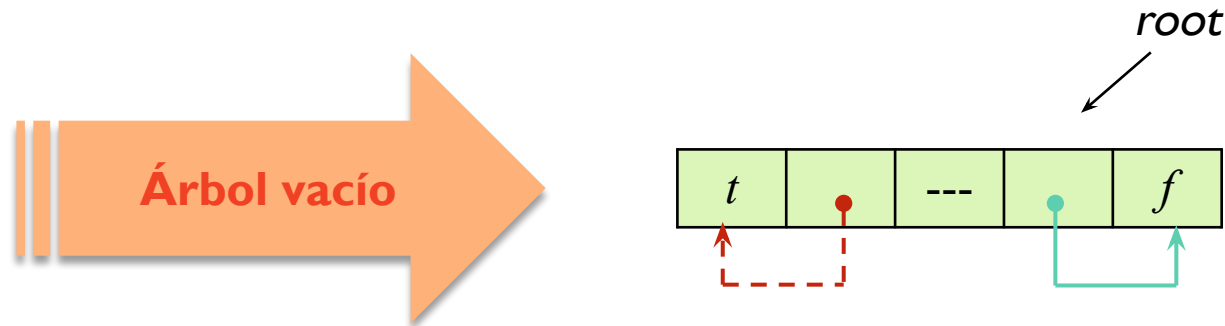
| | |
|-------------|-----------------------------|
| LeftThread | • Boolean |
| LeftChild | • Puntero a nodo izquierdo |
| Data | • Dato que almacena el nodo |
| RightChild | • Puntero a nodo derecho |
| RightThread | • Boolean |

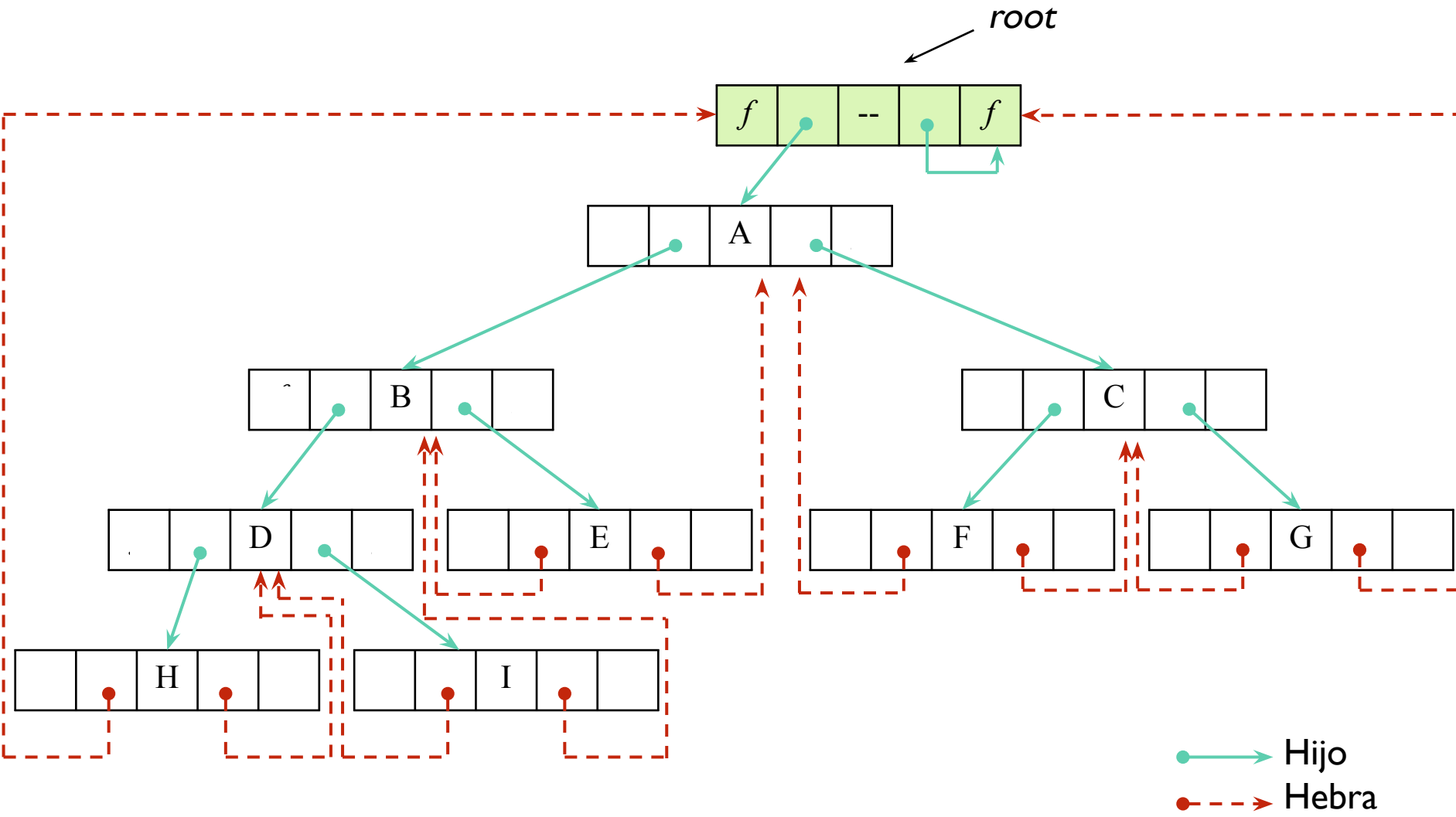
TRUE: LeftChild es hebra
FALSE: LeftChild es hijo

TRUE: RightChild es hebra
FALSE: RightChild es hijo

d) Árboles binarios enhebrados

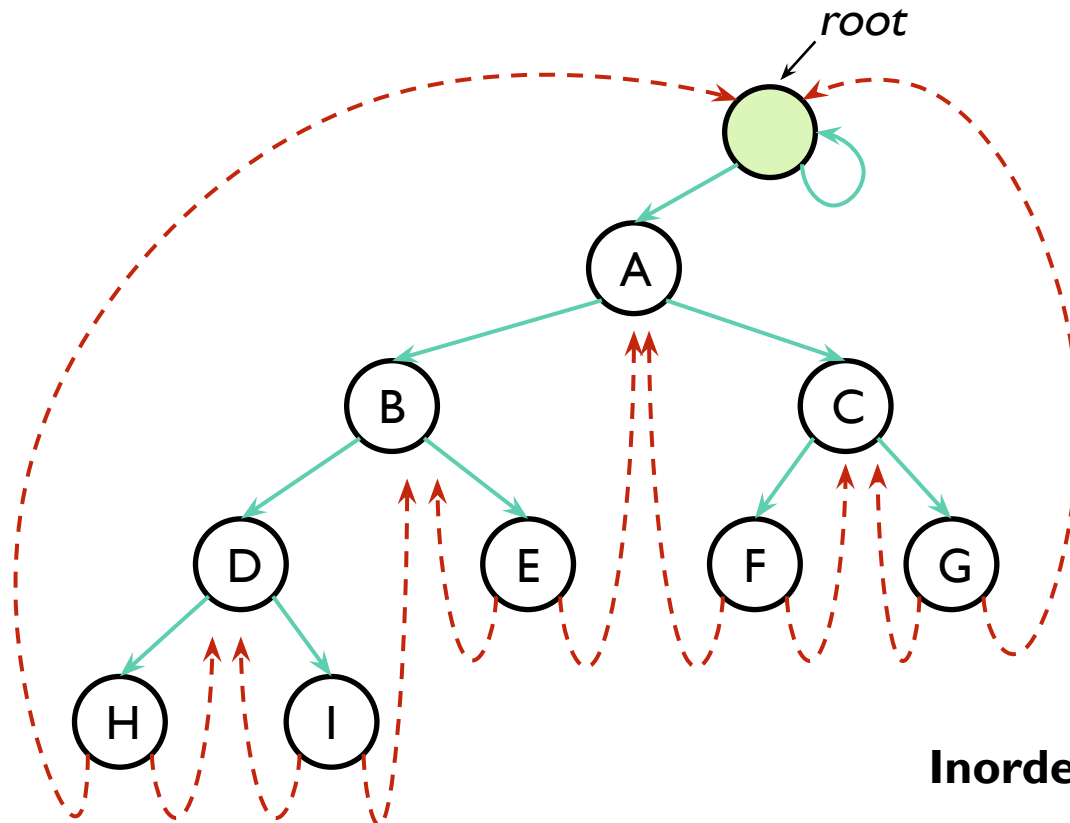
- El árbol siempre tiene un nodo de encabezamiento.
- Si el árbol no está vacío, el nodo de encabezamiento apunta al primer elemento (por su hijo **izquierdo**).





d) Árboles binarios enhebrados

- El árbol anterior es:



Inorder: HDIBEAFCG

d) Árboles binarios enhebrados

```
void threadedInorder(){
    if(root->LeftThread) return;
    Node current = leftMost(root->LeftChild);

    while(current <> root){
        print(current->data);

        if(current->RightThread) //Hebra
            current = current->RightChild;
        else //Hijo
            current = leftMost(current->RightChild);
    }
}
```

```
Node leftMost(Node p){
    if(!p) return NULL;

    Node aux = p;
    while(!aux->LeftThread)
        aux = aux->LeftChild;

    return aux;
}
```

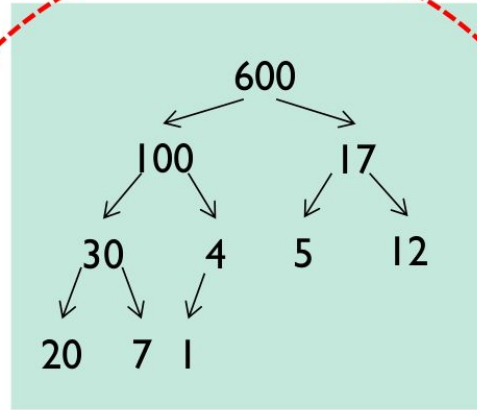
Contenidos

- a) Conceptos
- b) Árboles binarios
- c) Recorrido de árboles binarios
- d) Árboles binarios enhebrados
- e) **Heaps**
- f) Árboles de búsqueda binarios
- g) Árboles de búsqueda binarios balanceados
- h) Familia de árboles B (B , B^* , B^+)



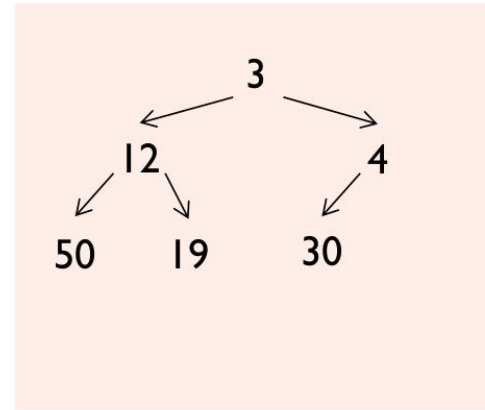
e) Heaps

- Es un caso especial de árbol binario **completo**.



MAX heap

Cada nodo almacena un valor **mayor o igual** a las claves de sus hijos. Por lo tanto, la raíz contiene el dato **mayor** del árbol.



MIN heap

Cada nodo almacena un valor **menor o igual** a las claves de sus hijos. Por lo tanto, la raíz contiene el dato **menor** del árbol.



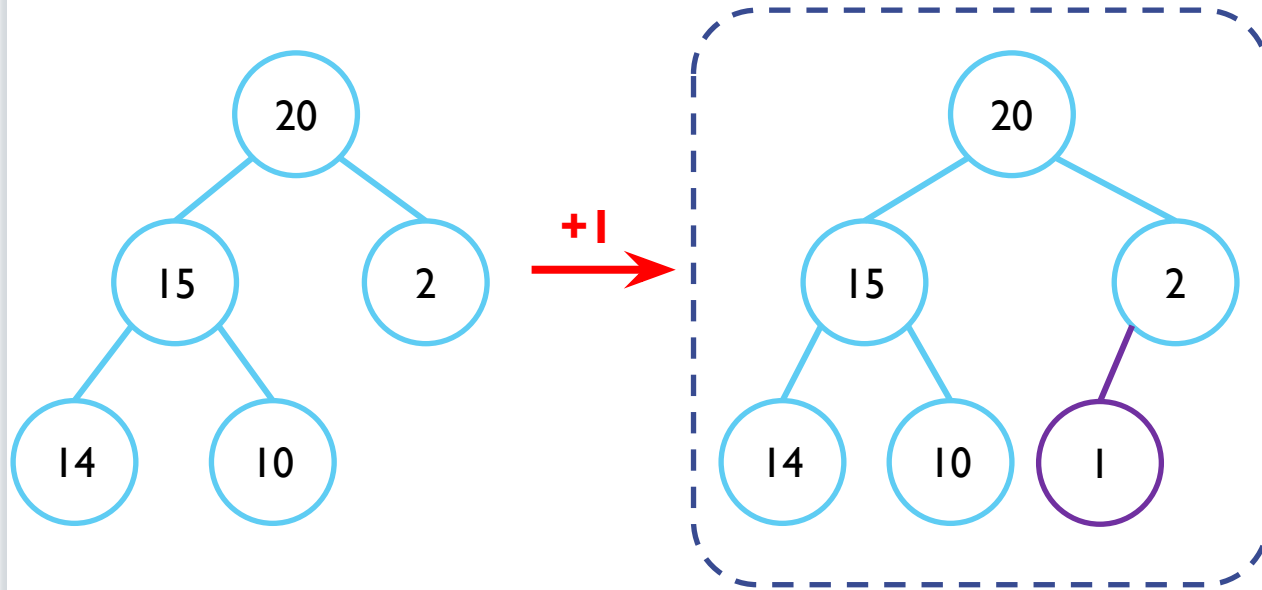
e) Heaps

- Insertar en un *heap*

e) Heaps

- Insertar en un *heap*

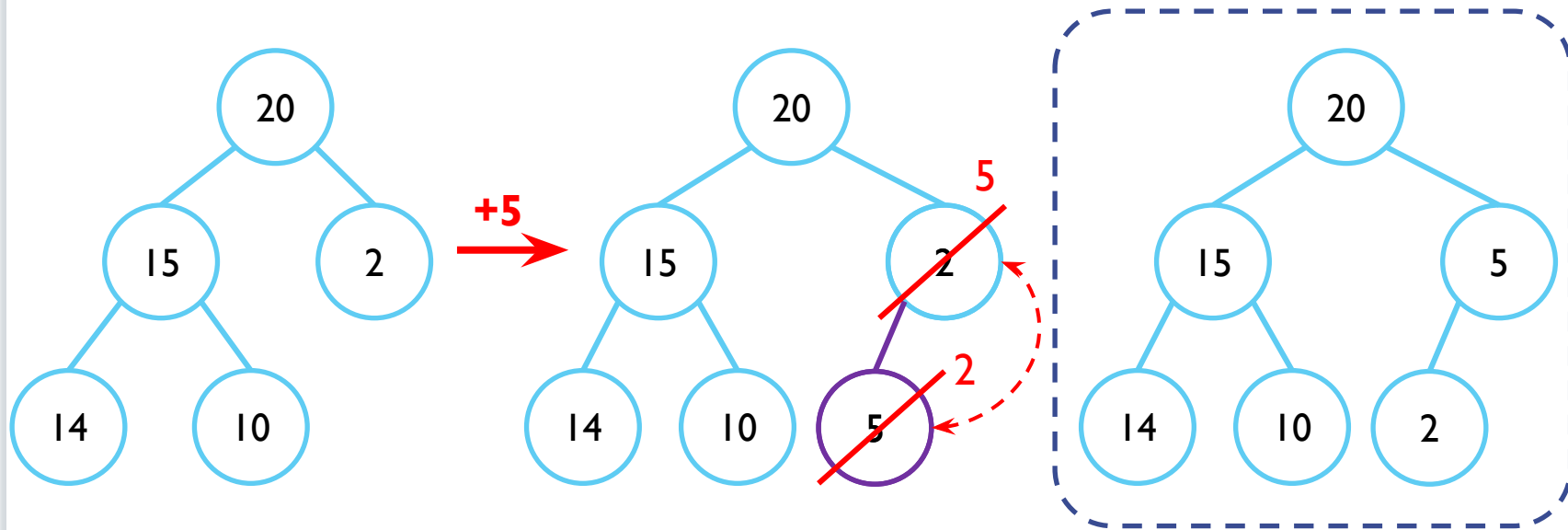
- Ejemplo 1:



No hay problema 😊

e) Heaps

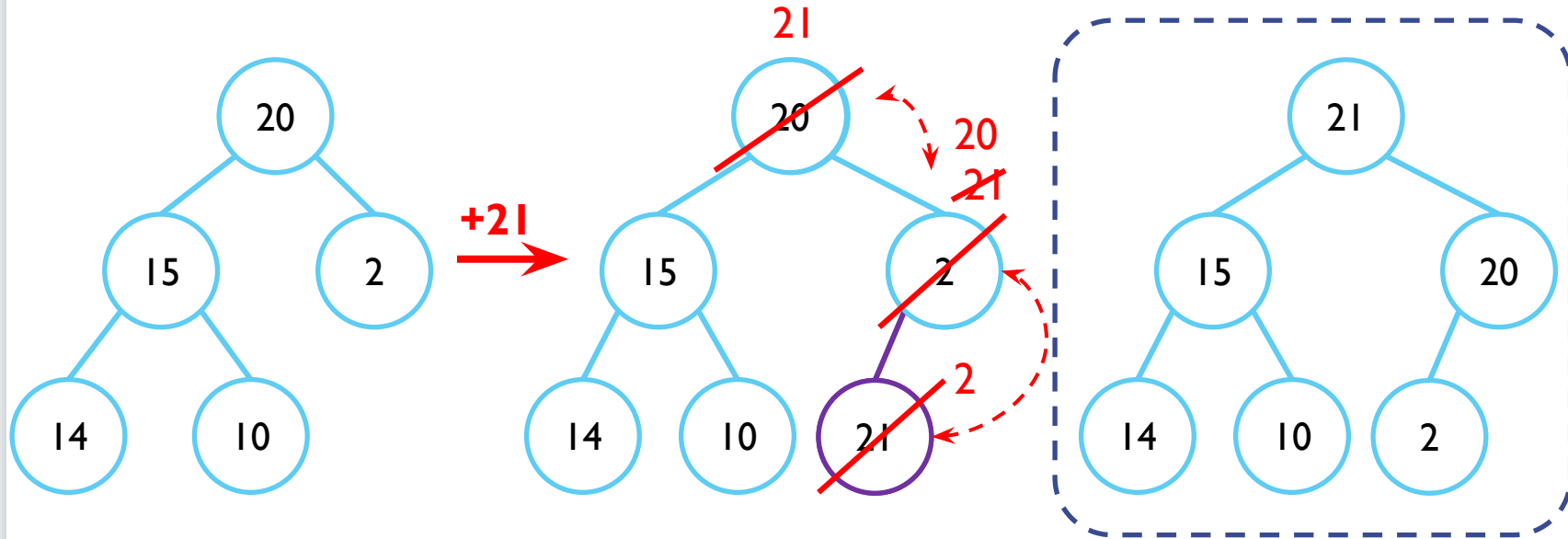
- Insertar en un *heap*
 - Ejemplo 2:



e) Heaps

- Insertar en un *heap*

- Ejemplo 3:





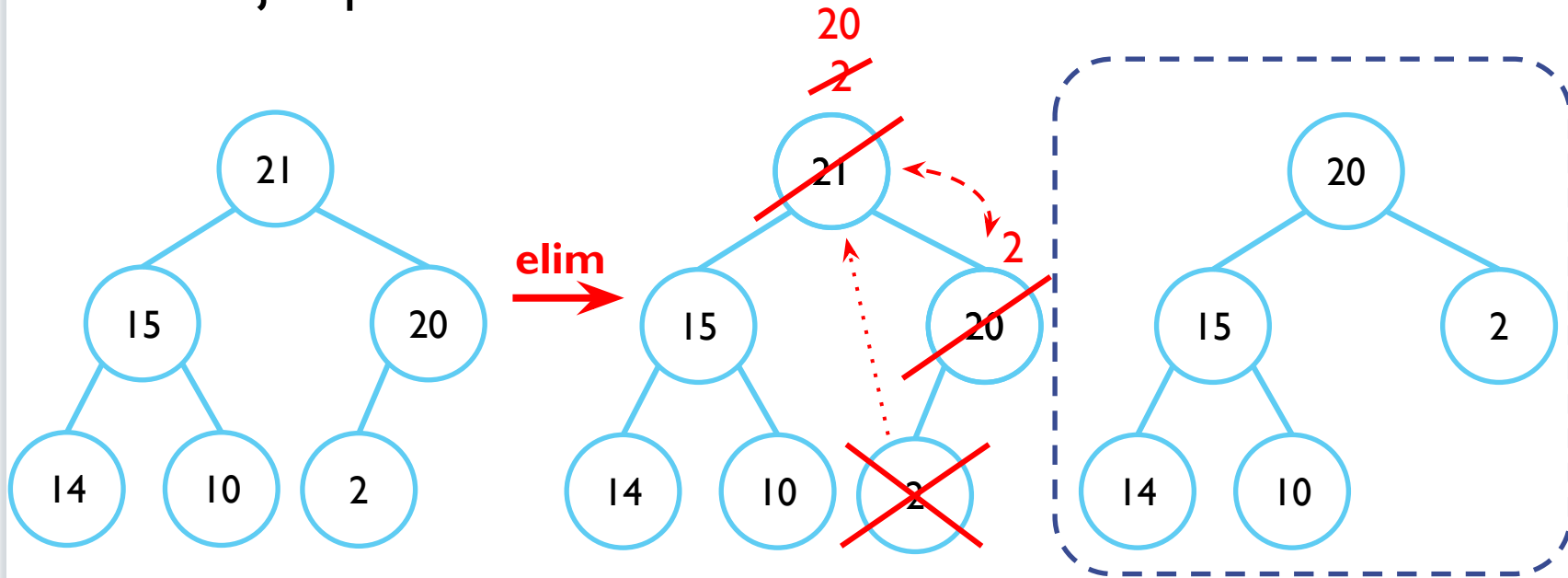
e) Heaps

- Eliminar en un *heap*

e) Heaps

- Eliminar en un *heap*

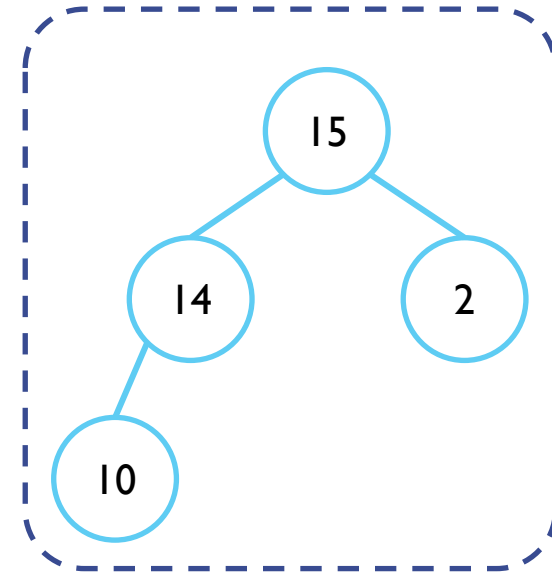
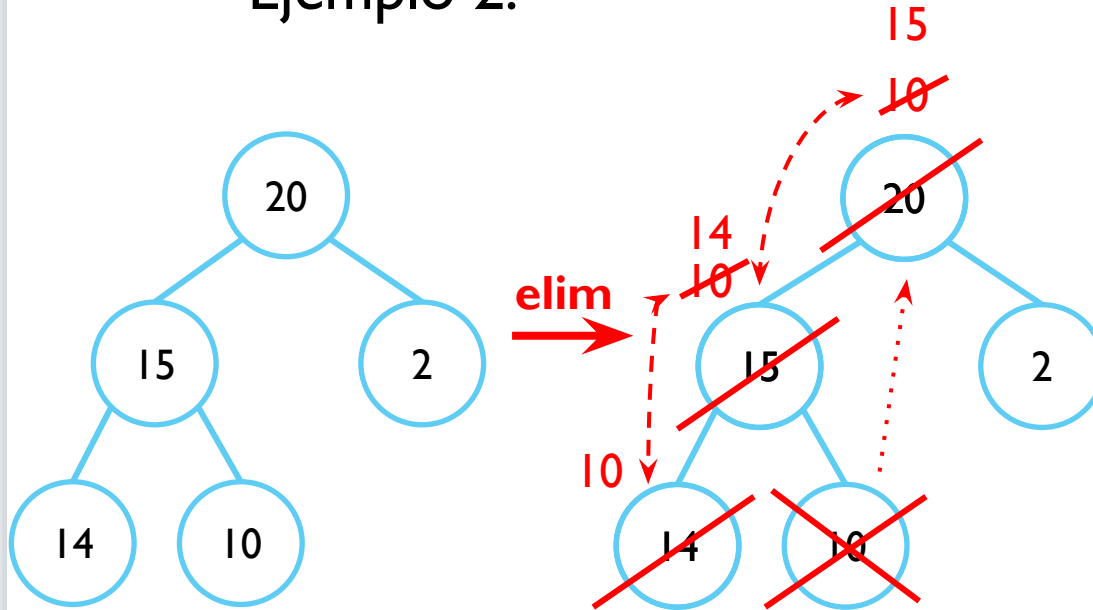
- Ejemplo 1:



e) Heaps

- Eliminar en un *heap*

- Ejemplo 2:



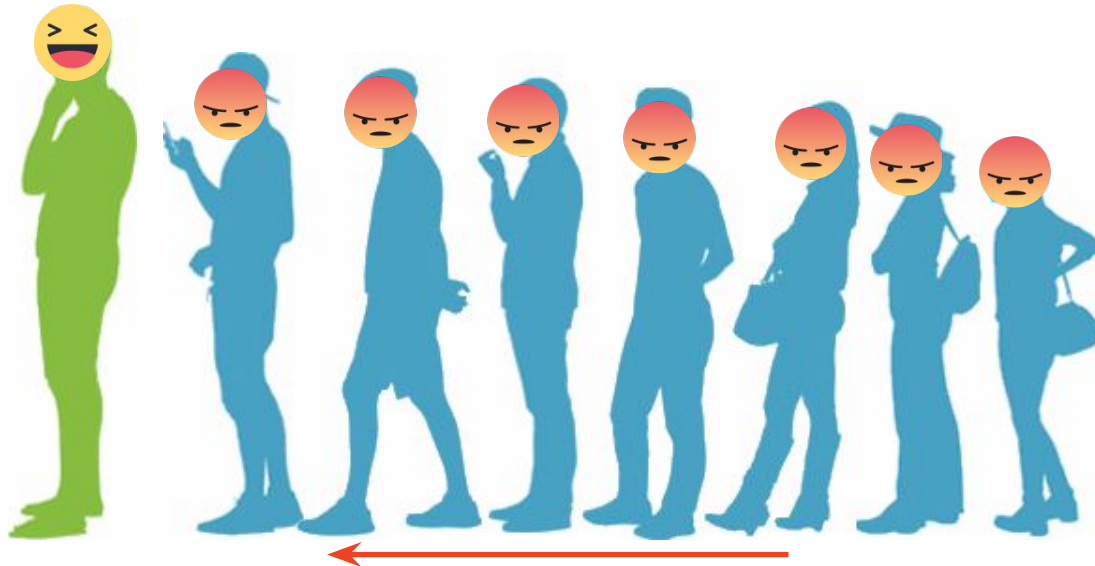
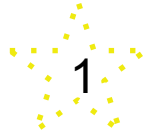
e) Heaps

- Comportamiento BIFO (*Best In First Out*)
- Colas con prioridad
 - El nodo que sale es aquel con mayor valor (posición [1])



e) Heaps

- Ejemplo: BEST = más alto

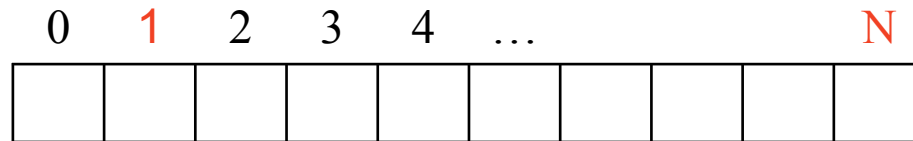


e) Heaps

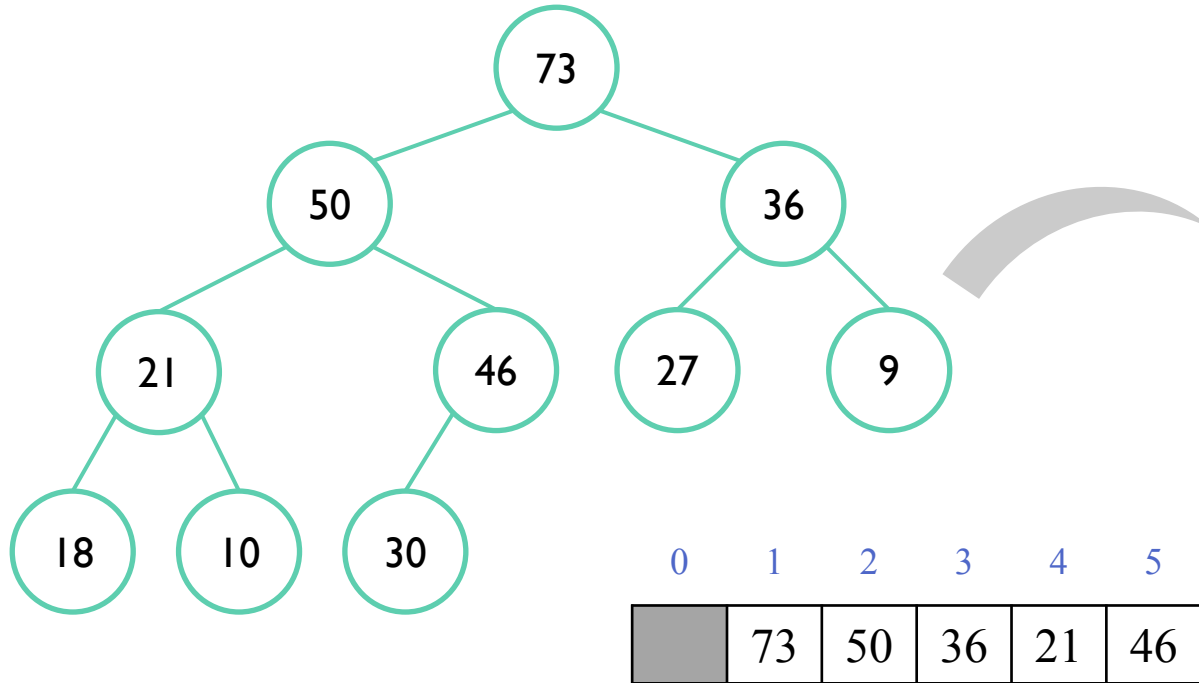
- Representación como arreglo: (secuencial)

- La posición 0 se deja vacía.
- Datos desde $A[1]$ hasta $A[N]$.
- El valor en $A[i]$ es el padre de los valores en

$A[2i]$ y $A[2i+1]$
h. izq h. der



e) Heaps



- $A[1]$ es padre de $A[2*1]$ y $A[2*1+1]$
- $A[4]$ es padre de $A[2*4]$ y $A[2*4+1]$

e) Heaps

- Para ir del hijo al padre hay que dividir en 2.

- El padre de $A[x]$ está en $A\left[\left\lfloor \frac{x}{2} \right\rfloor\right]$

| | | | | | | | | | | |
|---|----|----|----|----|----|----|---|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 73 | 50 | 36 | 21 | 46 | 27 | 9 | 18 | 10 | 30 |

- $A[10]$ es hijo de $A\left[\left\lfloor \frac{10}{2} \right\rfloor\right]$
- $A[3]$ es hijo de $A\left[\left\lfloor \frac{3}{2} \right\rfloor\right]$

e) Heaps

● HeapSort

- **Algoritmo de ordenamiento** con $O(n * \log n)$ en el peor caso.

- Consta de dos pasos:

Convertir a *heap*



Intercambios

- Ordenar de menor a mayor: *MAX heap*.
Ordenar de mayor a menor: *MIN heap*.

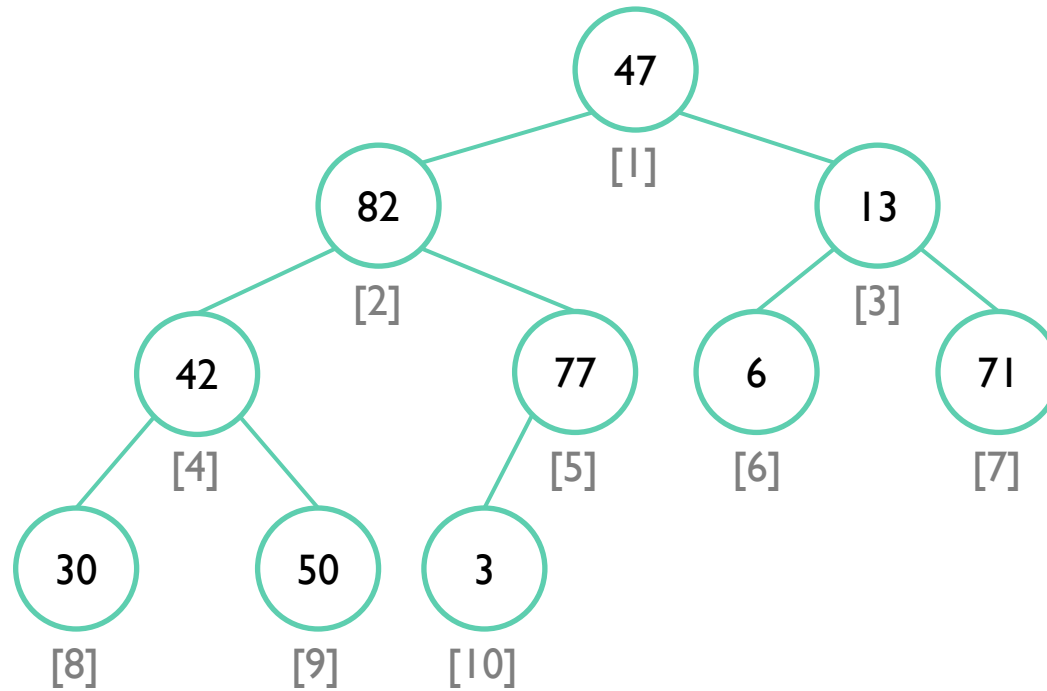
e) Heaps

- HeapSort

- Ejemplo: ordenar el siguiente arreglo de menor a mayor:

| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 42 | 77 | 6 | 71 | 30 | 50 | 3 |

| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 42 | 77 | 6 | 71 | 30 | 50 | 3 |



*[Vista como
árbol]*

| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 42 | 77 | 6 | 71 | 30 | 50 | 3 |


Convertir a *heap*



Intercambios

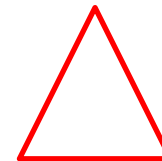
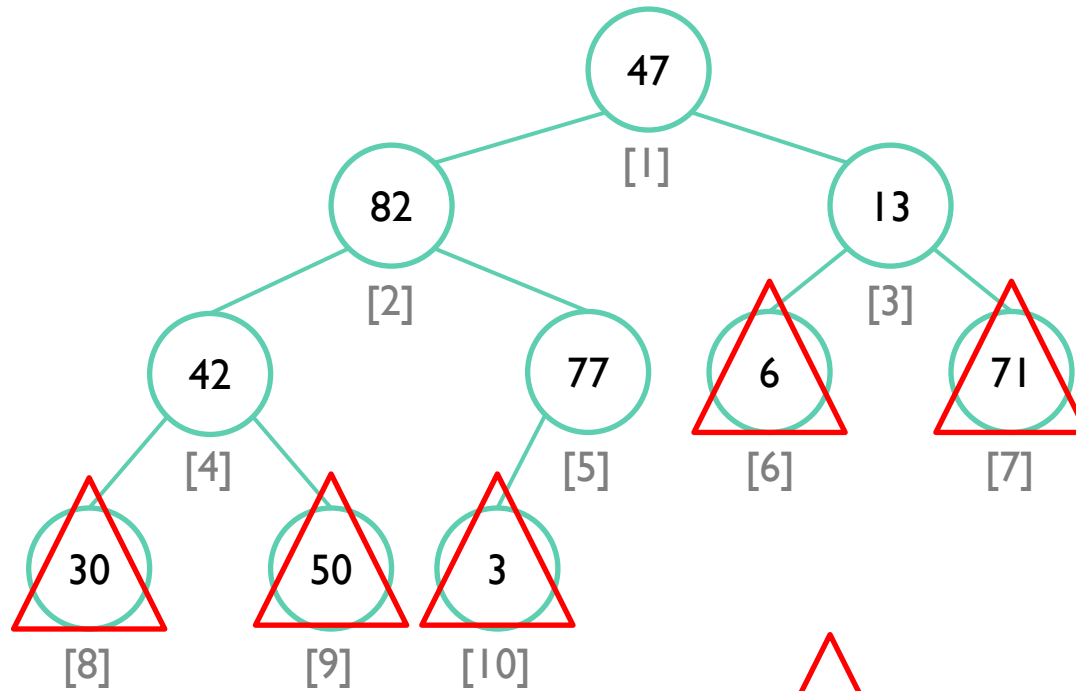
1° Convertir a *heap*

- $N = 10$
- Los elementos terminales del árbol son *heap* por definición.
- Es decir, desde $\left\lfloor \frac{N}{2} \right\rfloor + 1$ hasta N son *heap*.



6 hasta 10
- Hay que analizar desde $\left\lfloor \frac{N}{2} \right\rfloor$ hasta 1 (en ese orden).

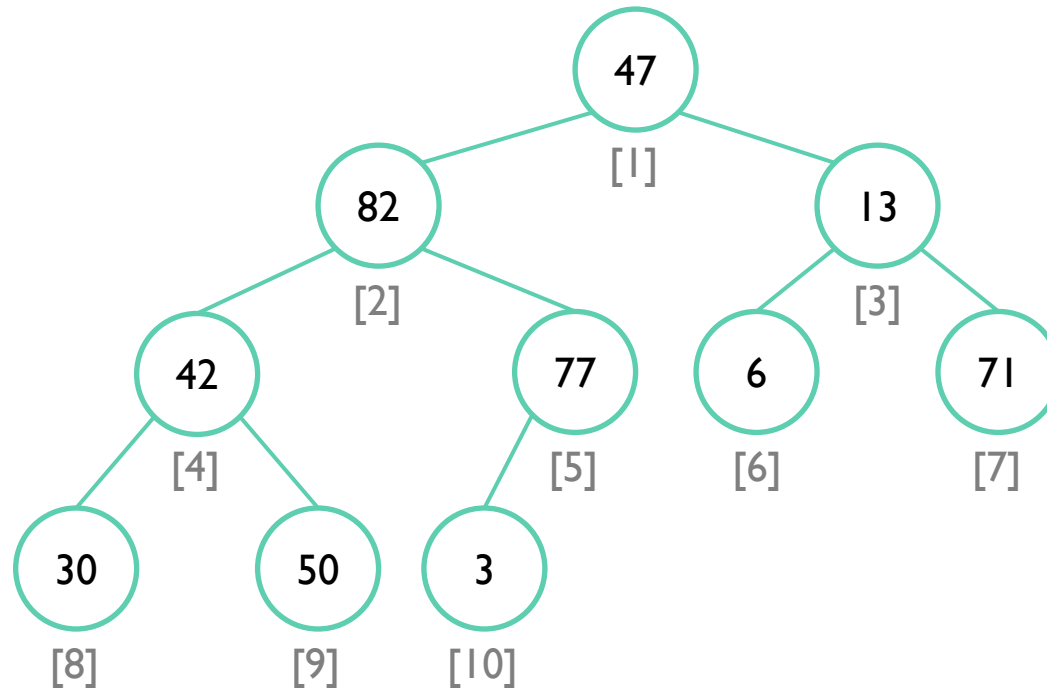
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 42 | 77 | 6 | 71 | 30 | 50 | 3 |



Nodos terminales ya son *heap*.

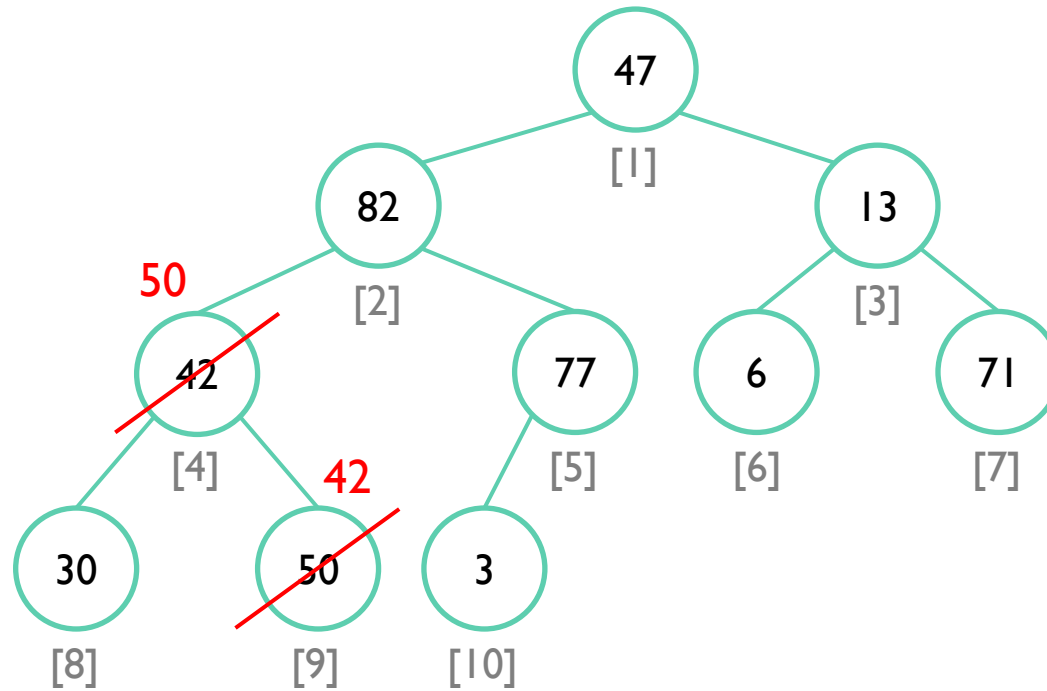
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 42 | 77 | 6 | 71 | 30 | 50 | 3 |

Posición [5]: es *heap*



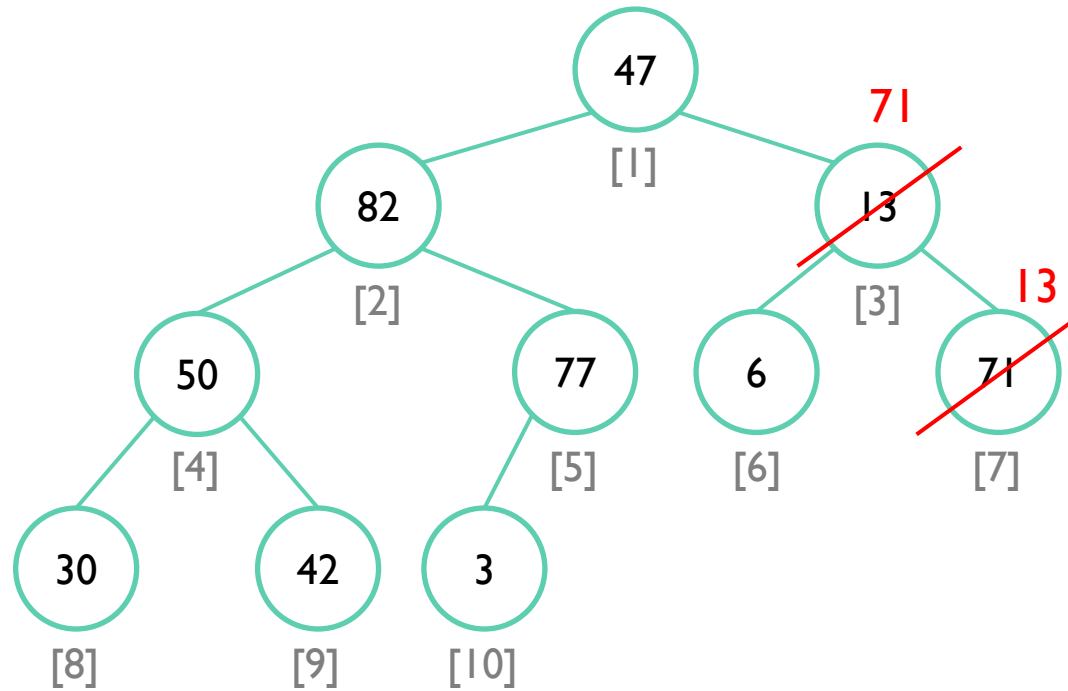
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 42 | 77 | 6 | 71 | 30 | 50 | 3 |

Posición [4]: rotar



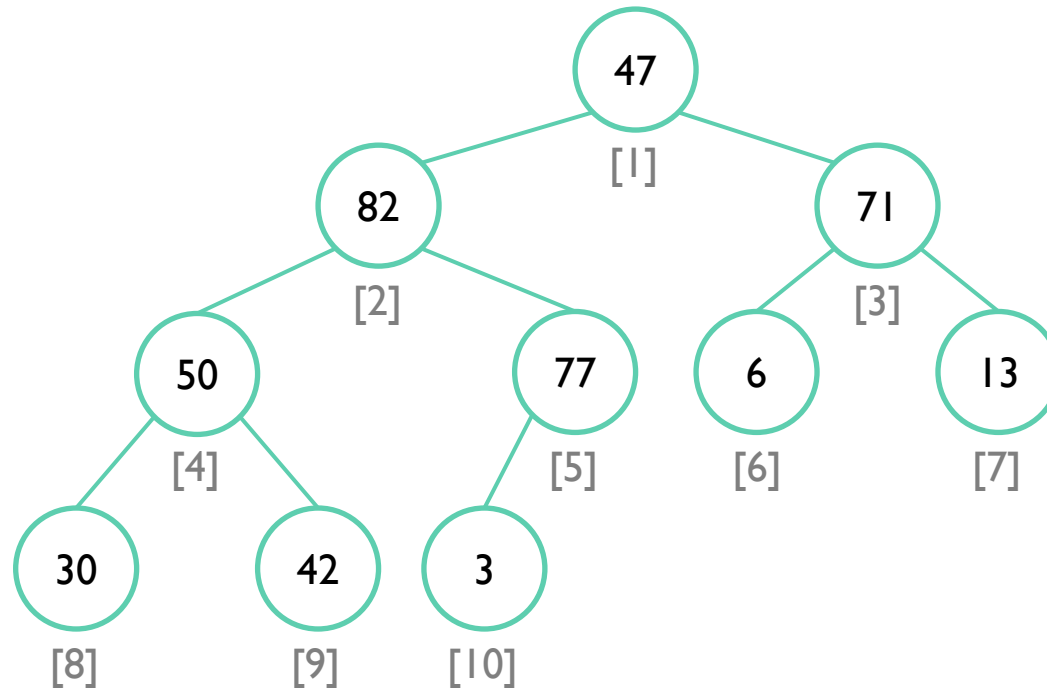
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 13 | 50 | 77 | 6 | 71 | 30 | 42 | 3 |

Posición [3]: rotar



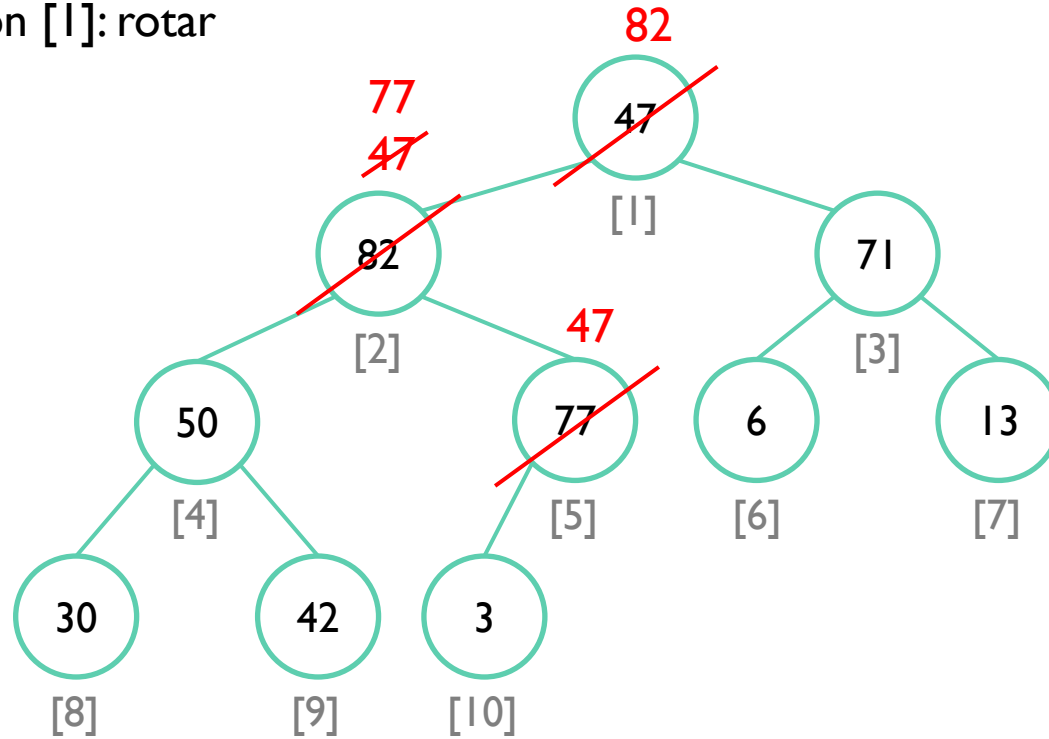
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 71 | 50 | 77 | 6 | 13 | 30 | 42 | 3 |

Posición [2]: es *heap*



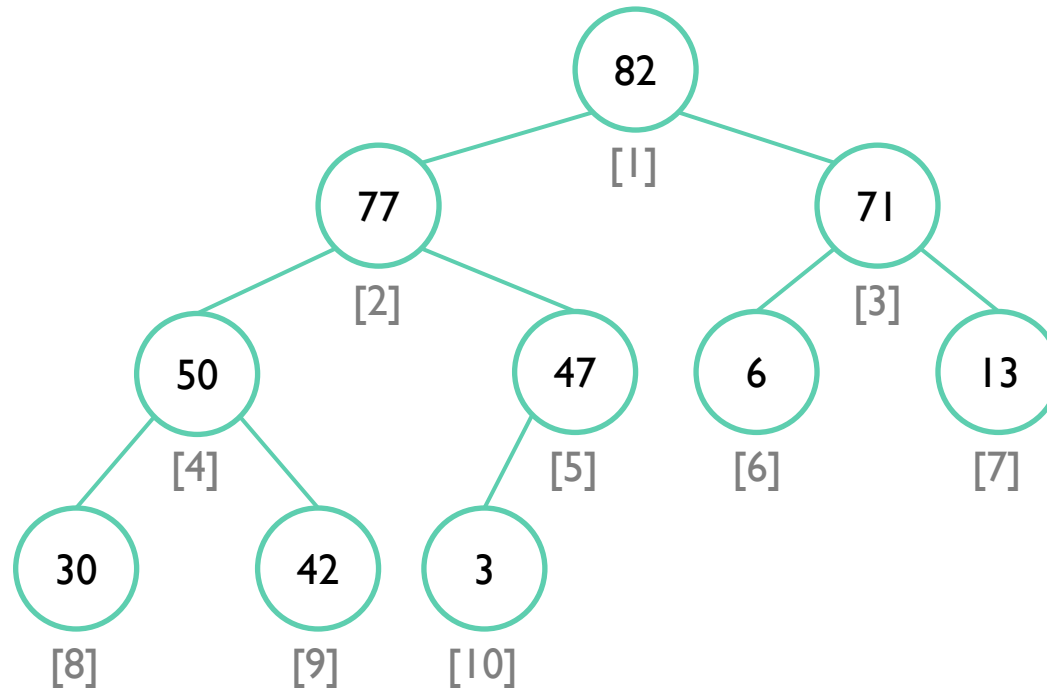
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 47 | 82 | 71 | 50 | 77 | 6 | 13 | 30 | 42 | 3 |


Posición [1]: rotar



| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 82 | 77 | 71 | 50 | 47 | 6 | 13 | 30 | 42 | 3 |

Entonces nos queda el *MAX heap*:





| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 82 | 77 | 71 | 50 | 47 | 6 | 13 | 30 | 42 | 3 |

Convertir a
heap

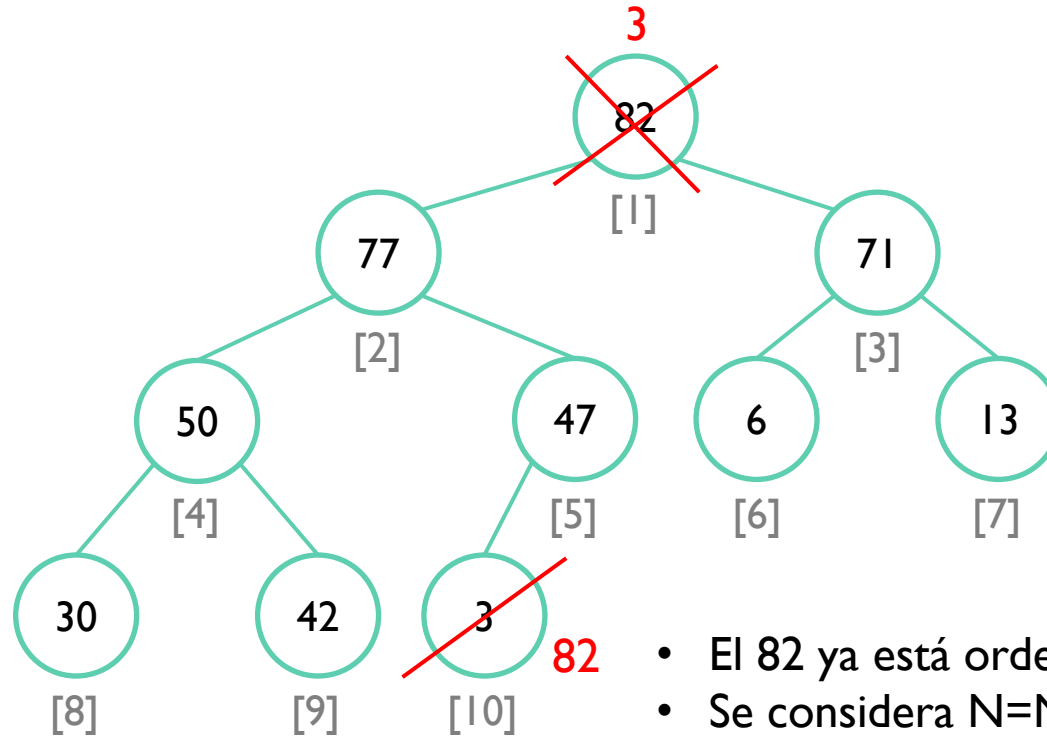


Intercambios

- 2º Intercambios

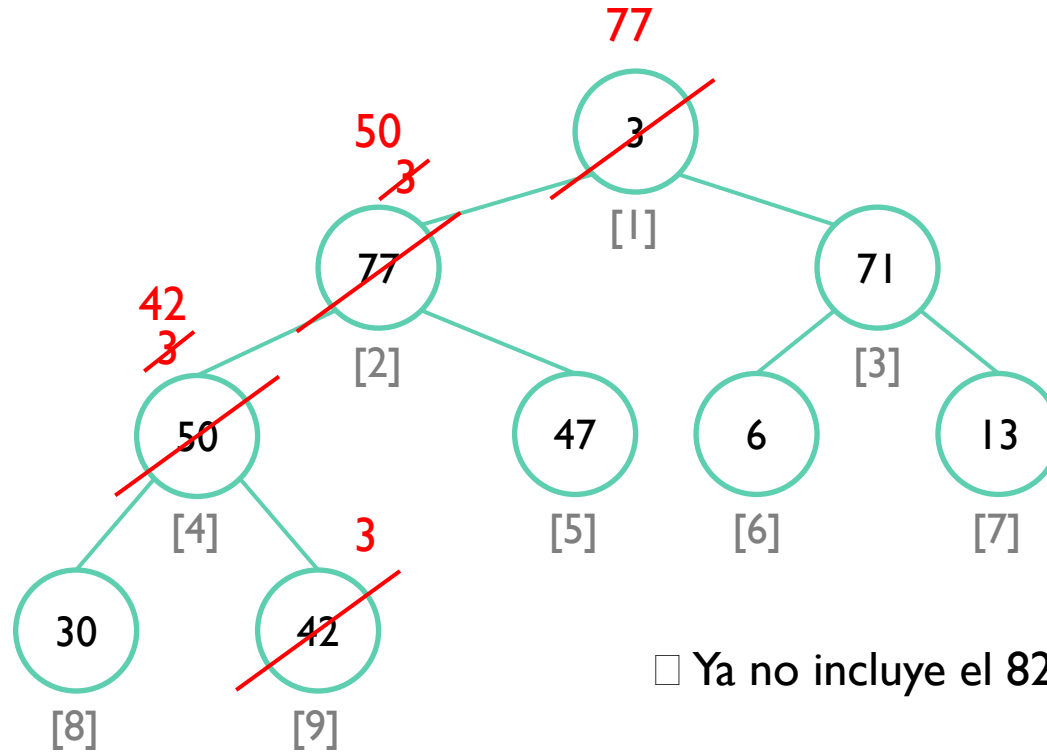
- Intercambio de $A[1]$ con $A[\text{último}]$.
- Asegurar que se mantiene la condición de *heap*.

| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 82 | 77 | 71 | 50 | 47 | 6 | 13 | 30 | 42 | 3 |



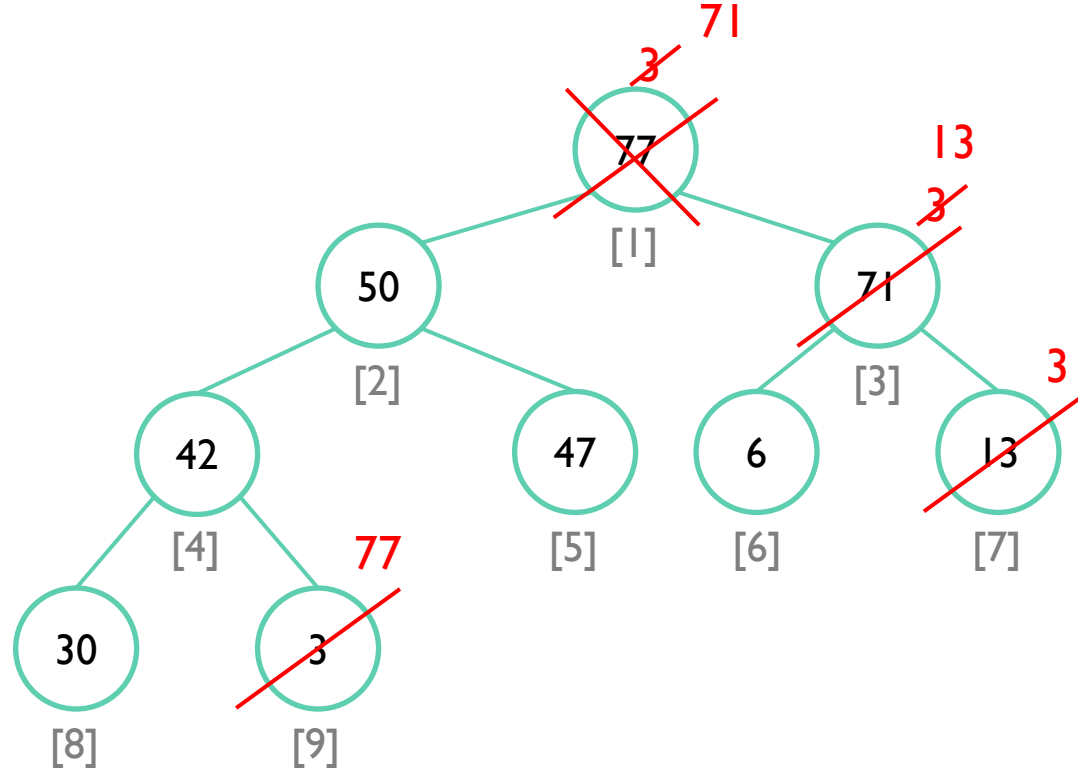
- El 82 ya está ordenado.
- Se considera $N=N-1$
- Ahora se verifica que no se pierda la condición de *heap*.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|----|----|----|----|---|----|----|----|----|
| | 3 | 77 | 71 | 50 | 47 | 6 | 13 | 30 | 42 | 82 |

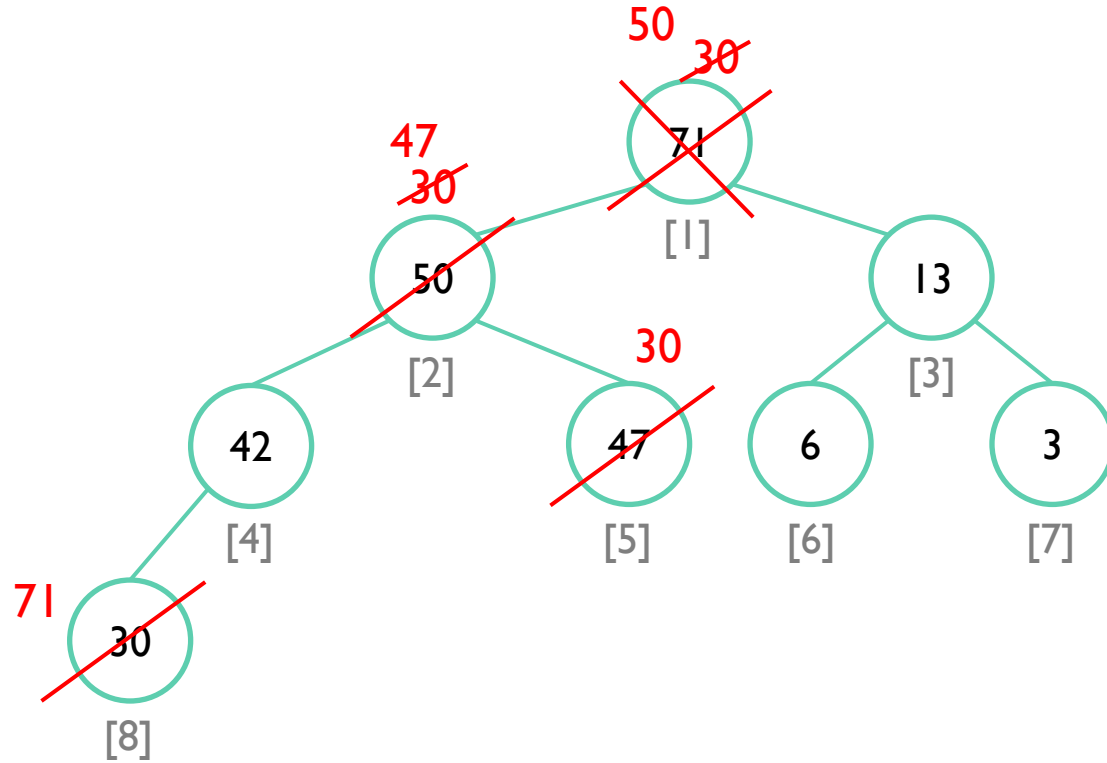


☐ Ya no incluye el 82 en la representación

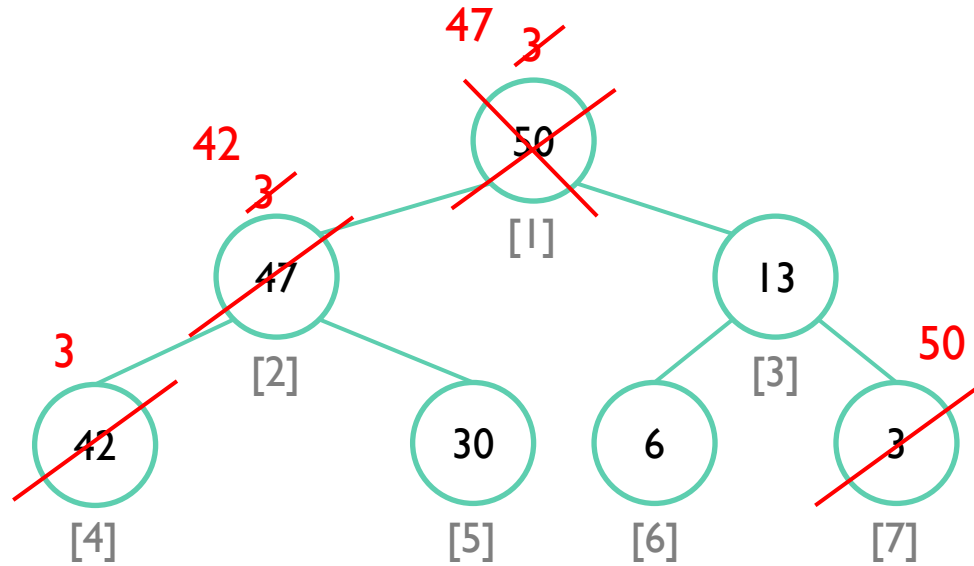
| | | | | | | | | | | |
|---|----|----|----|----|----|---|----|----|---|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 77 | 50 | 71 | 42 | 47 | 6 | 13 | 30 | 3 | 82 |



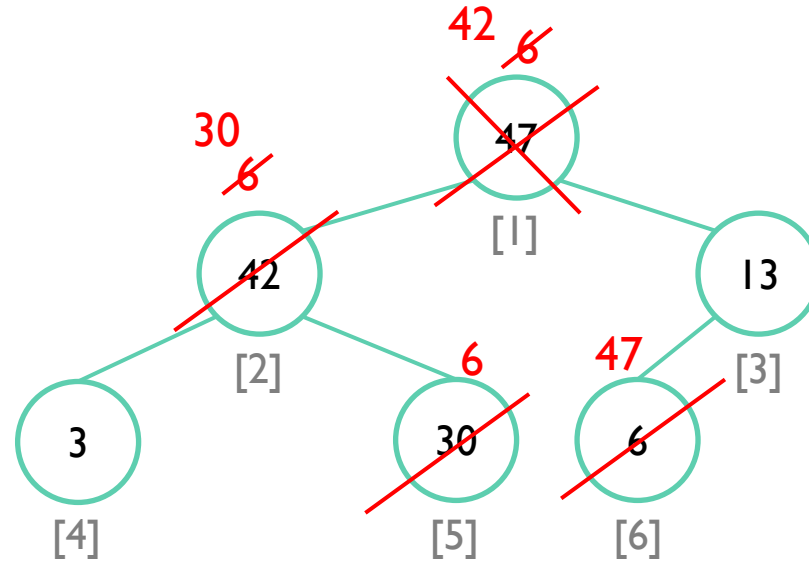
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|----|----|----|----|----|---|---|----|----|----|
| | 71 | 50 | 13 | 42 | 47 | 6 | 3 | 30 | 77 | 82 |



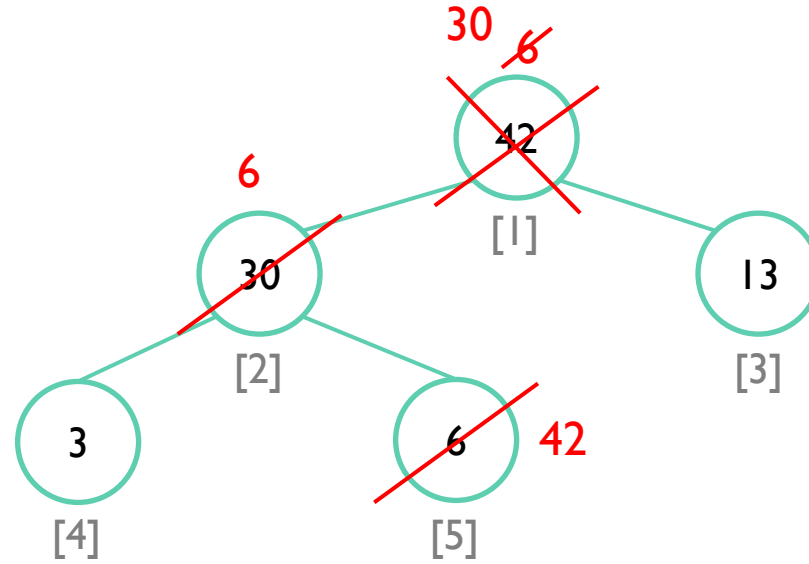

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|----|----|----|----|----|---|---|----|----|----|
| | 50 | 47 | 13 | 42 | 30 | 6 | 3 | 71 | 77 | 82 |



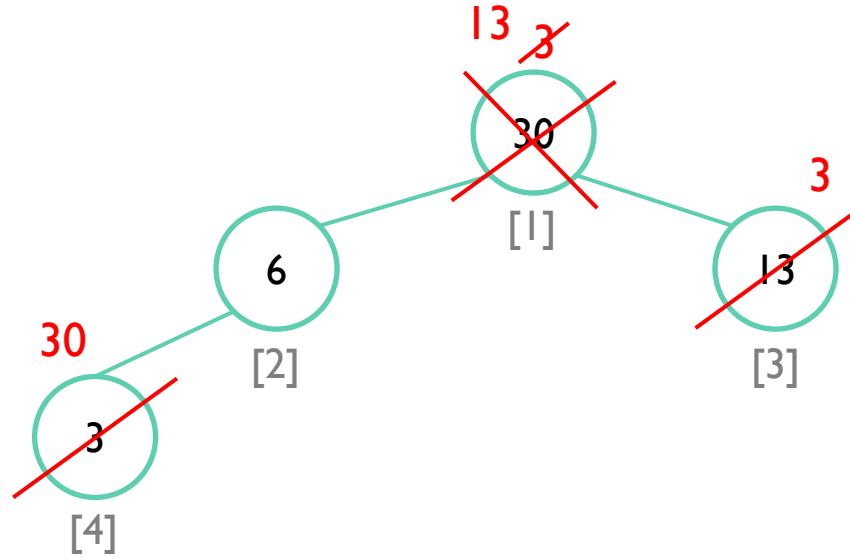
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|----|----|----|---|----|---|----|----|----|----|
| | 47 | 42 | 13 | 3 | 30 | 6 | 50 | 71 | 77 | 82 |



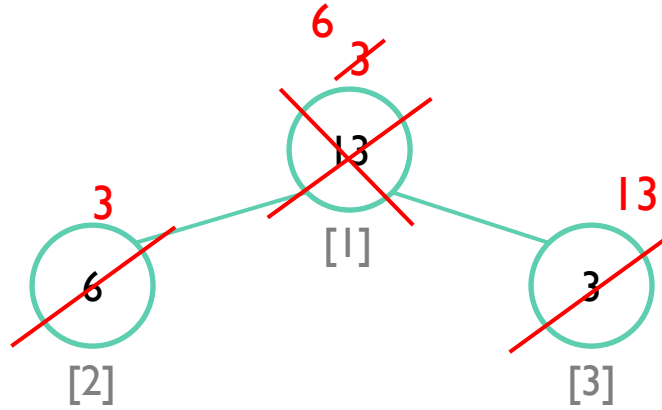
| | | | | | | | | | | |
|---|----|----|----|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 42 | 30 | 13 | 3 | 6 | 47 | 50 | 71 | 77 | 82 |



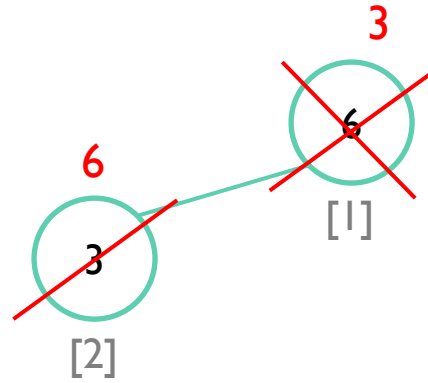

| | | | | | | | | | | |
|---|----|---|----|---|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 30 | 6 | 13 | 3 | 42 | 47 | 50 | 71 | 77 | 82 |



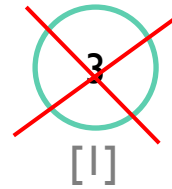
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|----|---|---|----|----|----|----|----|----|----|
| | 13 | 6 | 3 | 30 | 42 | 47 | 50 | 71 | 77 | 82 |



| | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 6 | 3 | 13 | 30 | 42 | 47 | 50 | 71 | 77 | 82 |



| | | | | | | | | | | |
|---|---|---|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | 3 | 6 | 13 | 30 | 42 | 47 | 50 | 71 | 77 | 82 |



¡ORDENADO!