UNIVERSIDAD NACIONAL
GUILLERMO BROWN

CLASE 9 - Unidad 5

Cola de Prioridades-HEAP

ESTRUCTURAS DE DATOS (271)

Clase N. 9. Unidad 5.

Clase 9: AGENDA

AGENDA

- **Temario:**
  - Heap binaria. Implementaciones y operaciones: Operaciones de inserción, borrado y construcción
  - Aplicaciones: Selección y Ordenación (Heapsort).
  - Análisis de la eficiencia de cada algoritmo.

- **Ejemplos** en Lenguajes **Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

Cola de Prioridades(HEAP):

Una **cola de prioridad** es una estructura de datos que permite al menos dos operaciones:

- Insertar un elemento en la estructura
- Buscar, recuperar y eliminar el elemento mínimo

Implementaciones:

- Lista Ordenada

  Insertar tiene un tiempo de ejecución **O(N)** operaciones.

  Borrar el mínimo tiene un tiempo de **O(1)** operaciones.

- Lista no Ordenada

  Insertar tiene un tiempo de ejecución **O(1)** operaciones.

  Borrar el mínimo tiene un tiempo de **O(N)** operaciones.

- Árbol binario de búsqueda

  Insertar y *Borrar el mínimo* tienen un tiempo de ejecución promedio de **O(log N)** operaciones.

Heap Binaria:

- Es una implementación de colas de prioridad que no usa punteros y permite implementar ambas operaciones con **O(logN)** operaciones en el peor caso.

- Cumple con dos propiedades:

  Propiedad estructural

  Propiedad de orden

Propiedad Estructural:

Una Heap es un árbol binario completo.

- En un árbol binario lleno de *altura h*, los nodos internos tienen exactamente 2 hijos y las hojas tienen la misma profundidad

- Un árbol binario completo de *altura h* es un árbol binario lleno de *altura h-1* y en el nivel h, los nodos se completan de izquierda a derecha

Clase 9: Heap Binaria

Propiedad Estructural:

El número de nodos ***n*** de un árbol binario completo de altura ***h***, satisface:

$$2^h \leq n \leq (2^{h+1}-1)$$

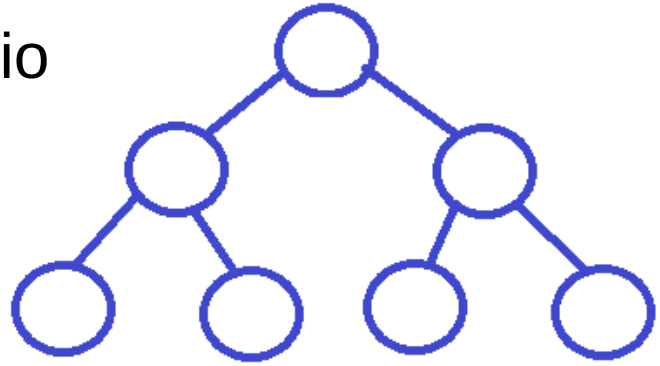
- Si el árbol es lleno, $\mathbf{n= 2^{h+1}-1}$

- Si no, el árbol es lleno en la altura h-1 y tiene por lo menos un nodo en el nivel h:

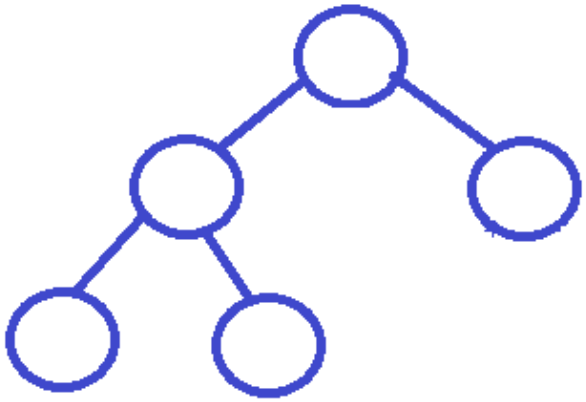
$$n= 2^{h-1+1}-1+1=2h$$

La altura h del árbol es de **O(log n)**

Ârbol Binario LLeno



Àrbol Binario Completo



Clase 9: Heap Binaria

Propiedad Estructural:

Dado que un árbol binario completo es una estructura de datos regular, puede almacenarse en un arreglo, tal que:

- La raíz está almacenada en la posición 1
- Para un elemento que está en la posición i:

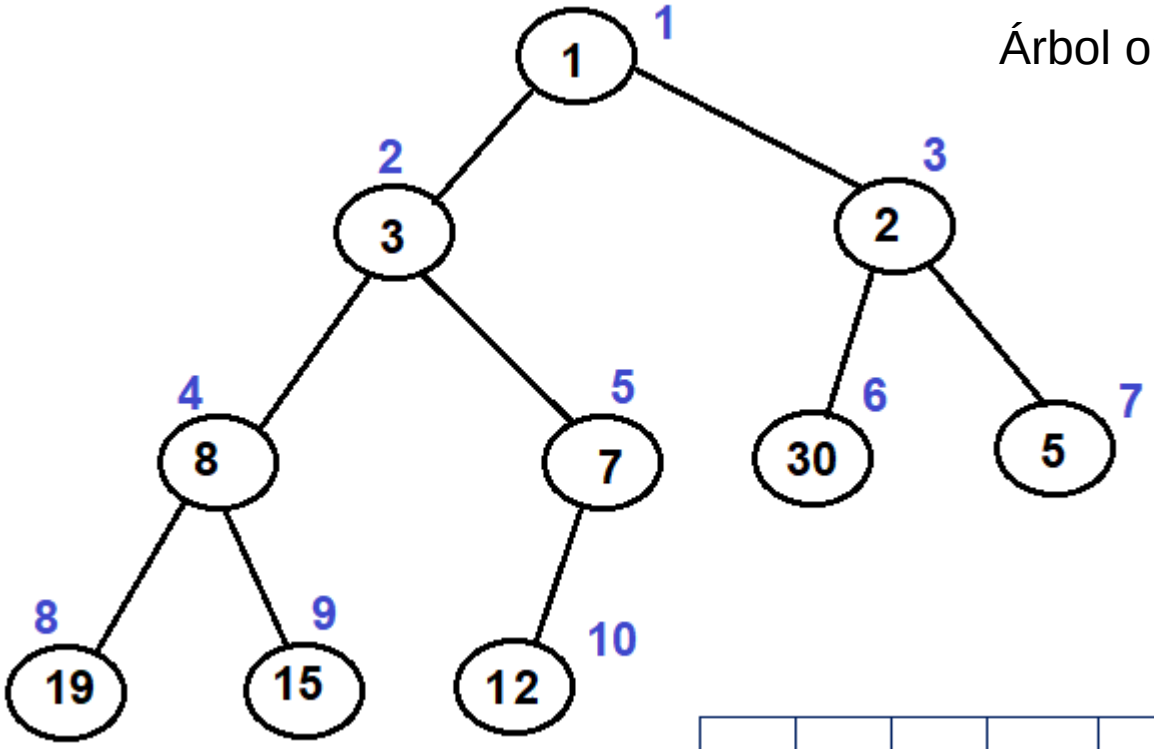
  El hijo izquierdo está en la posición 2*i

  El hijo derecho está en la posición 2*i+1

  El padre está en la posición [i/2]

Clase 9: Heap Binaria

Propiedad Estructural:

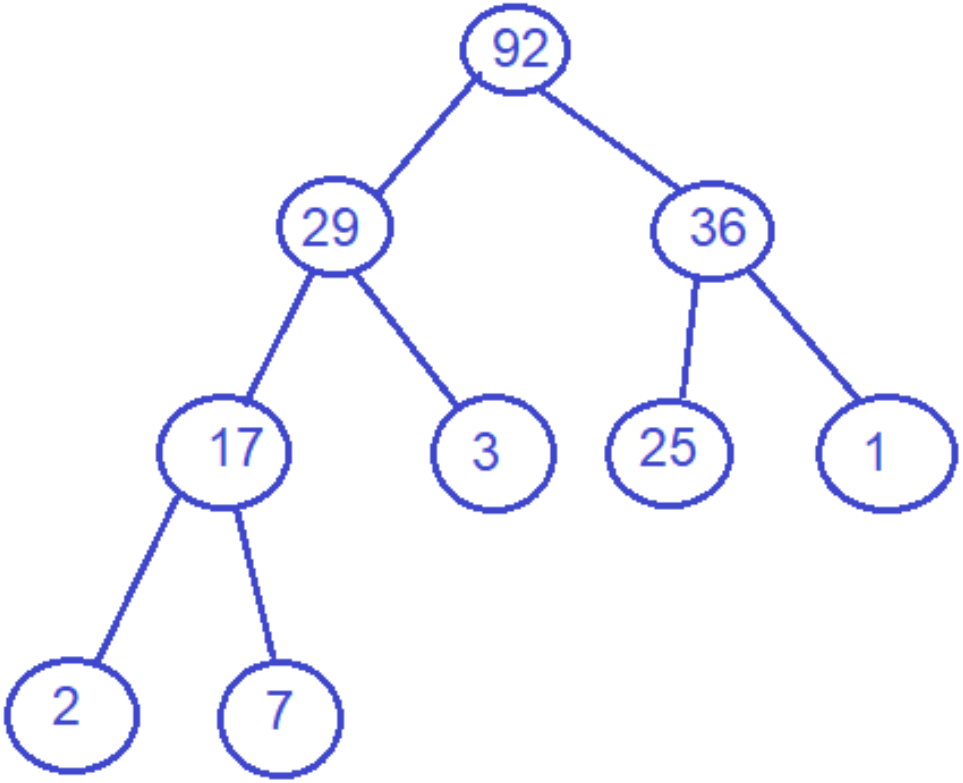


Árbol ordenado en una Heap

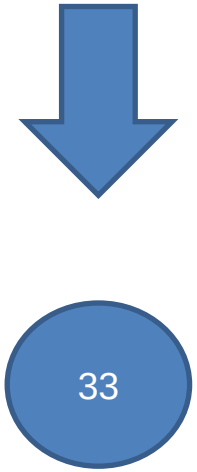
1	3	2	8	7	30	5	19	15	12
1	2	3	4	5	6	7	8	9	10

Clase 9: Heap Binaria

Operaciones:



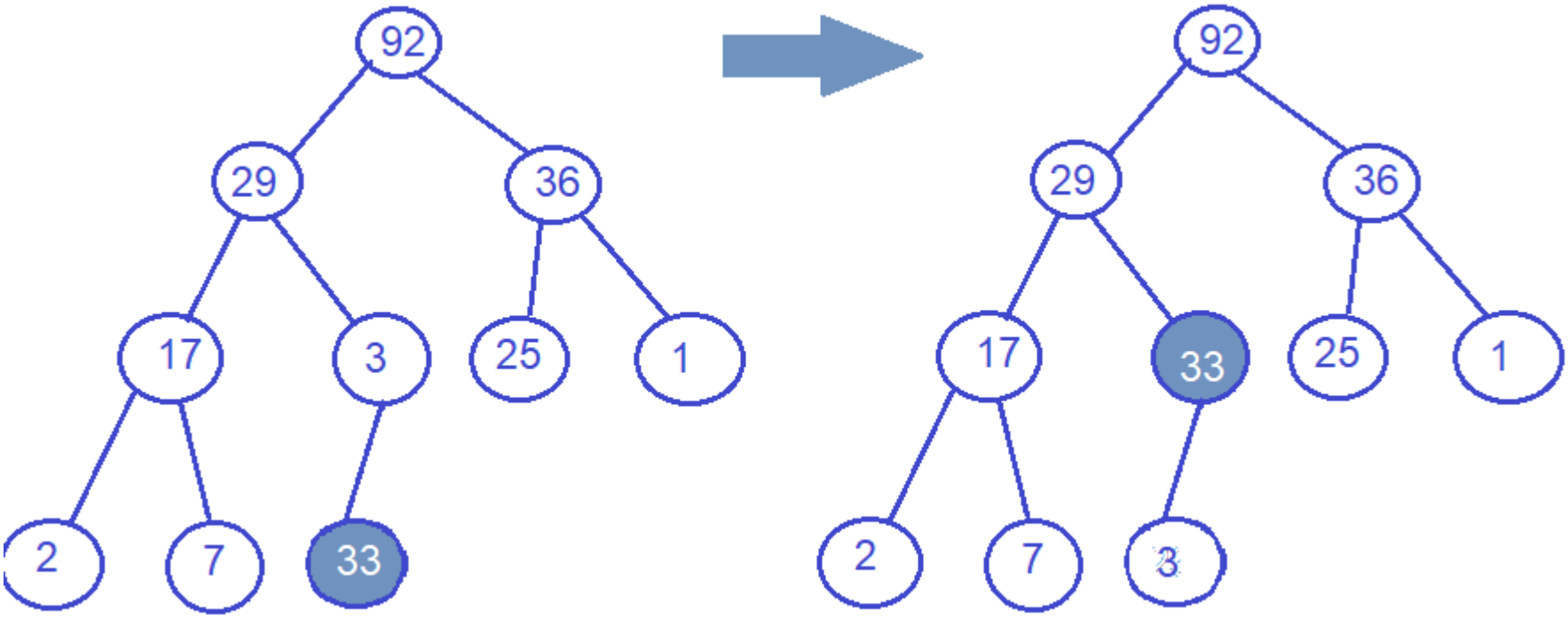
Insertar un nodo en el árbol



Clase 9: Heap Binaria

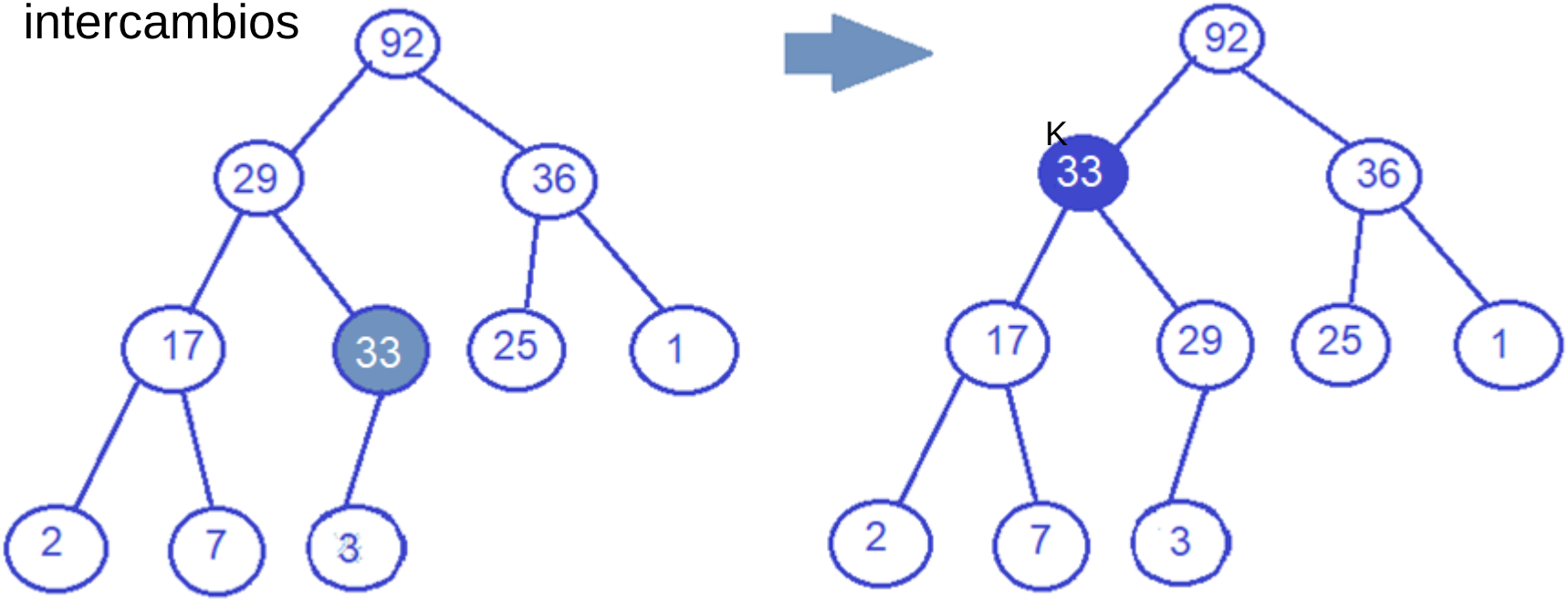
Operaciones:

El nodo se ***inserta*** al final del árbol y luego se debe verificar que el árbol quede ordenado, si no es así, se intercambian los elementos: hijo por el padre, sucesivamente hasta que quede ordenado.



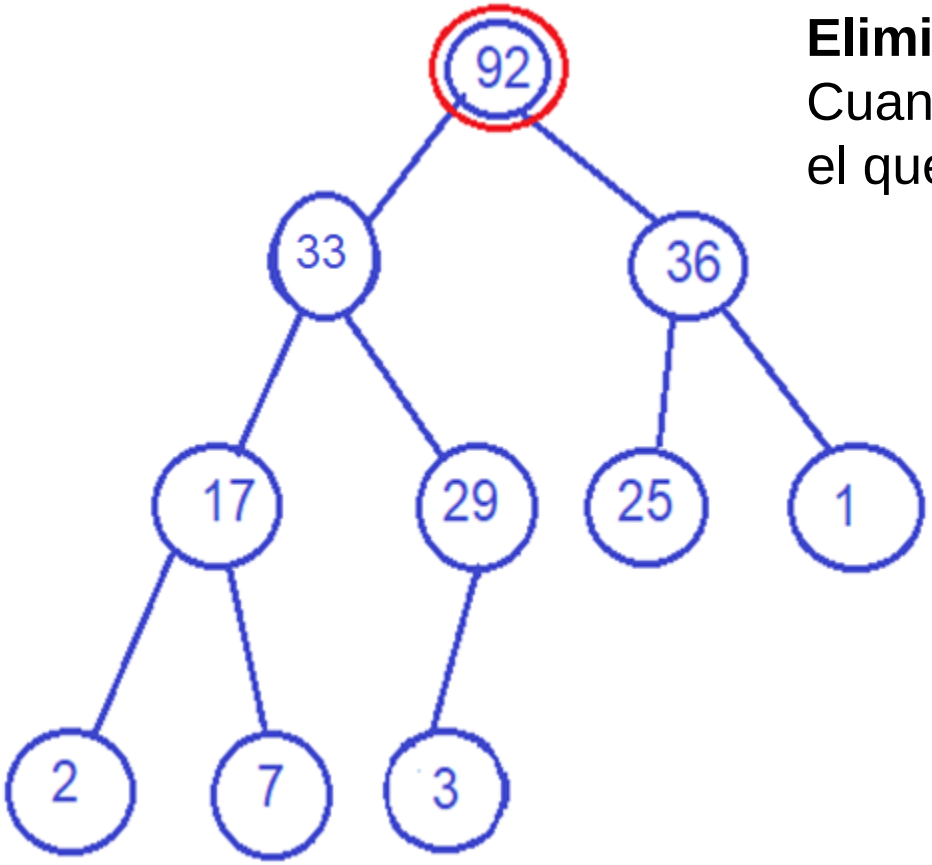
Operaciones:

- El filtrado hacia arriba restaura la propiedad de orden intercambiando ***k*** a lo largo del camino hacia arriba desde el lugar de inserción
- El filtrado termina cuando la clave ***k*** alcanza la raíz o un nodo cuyo padre tiene una clave menor
- Ya que el algoritmo recorre la altura de la heap, tiene ***O(log n)*** intercambios

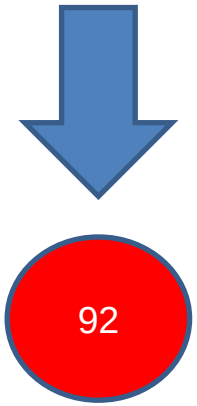


Clase 9: Heap Binaria

Operaciones:



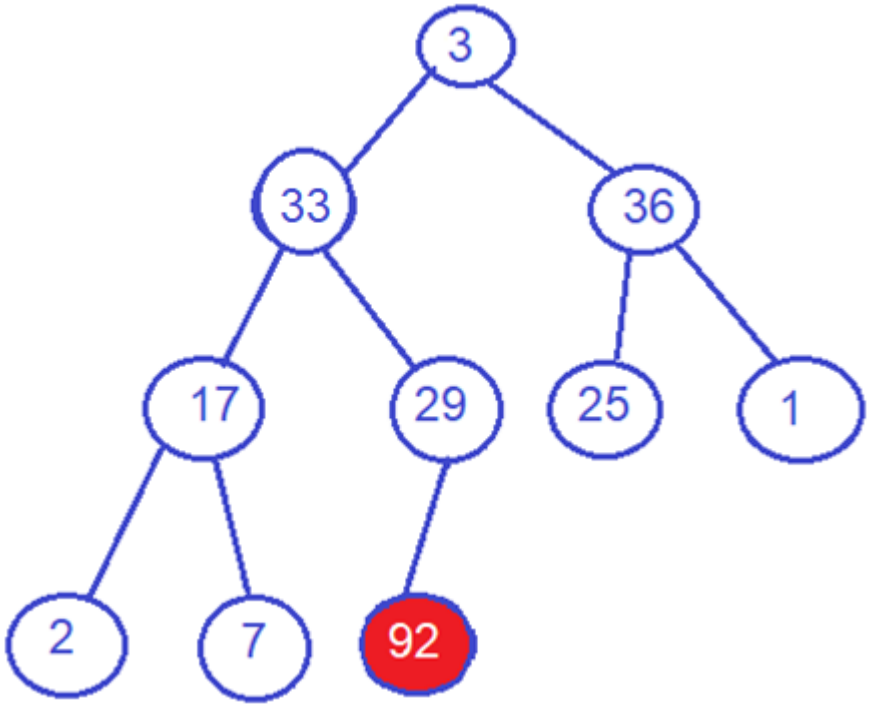
Eliminar un nodo del árbol.
Cuando se elimina un nodo siempre es el que se encuentra en la raíz del árbol

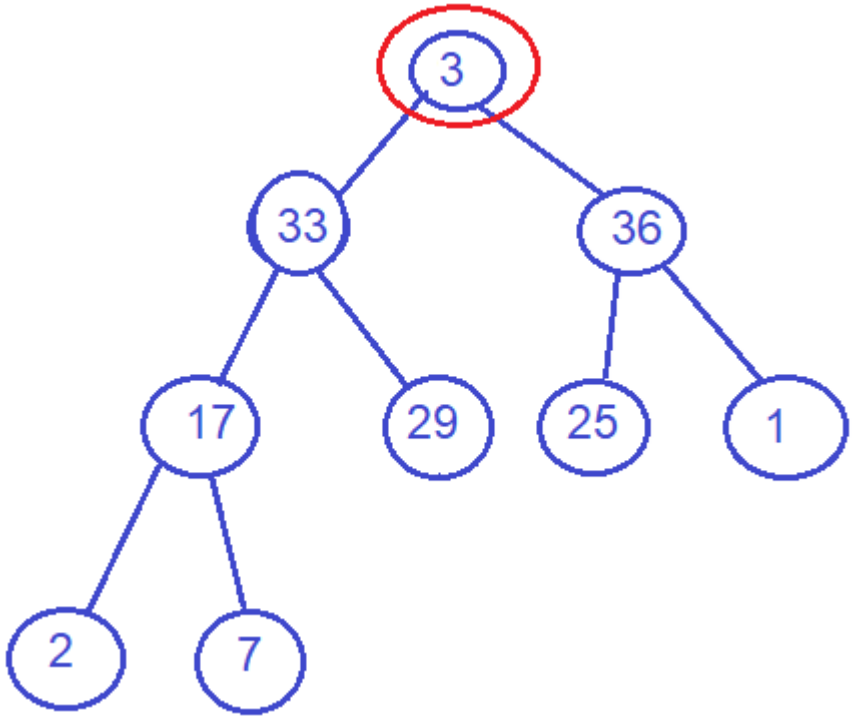


Clase 9: Heap Binaria

Operaciones:

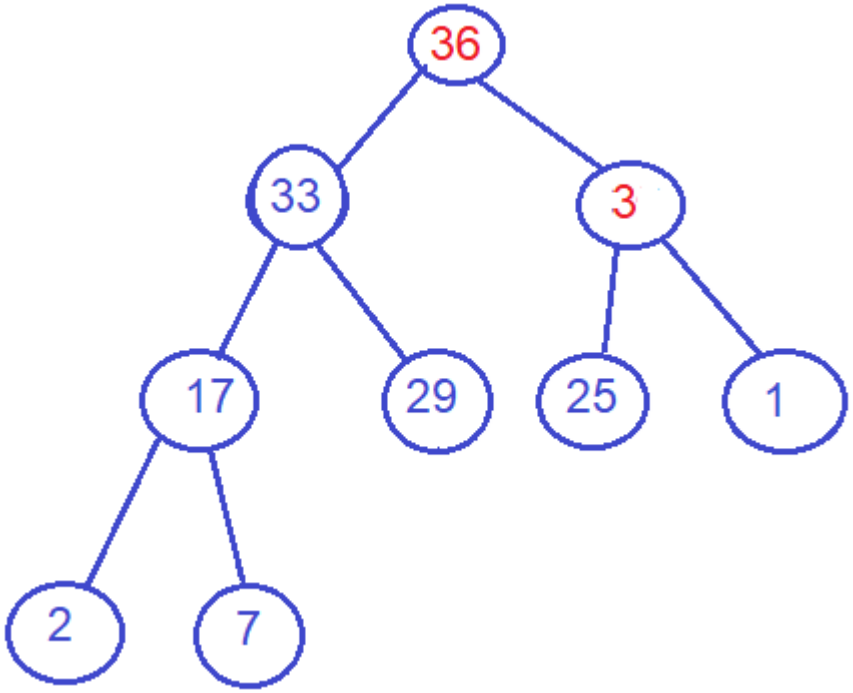
Se **Elimina** el nodo 92 del árbol.
Ahora debemos reordenar los nodos



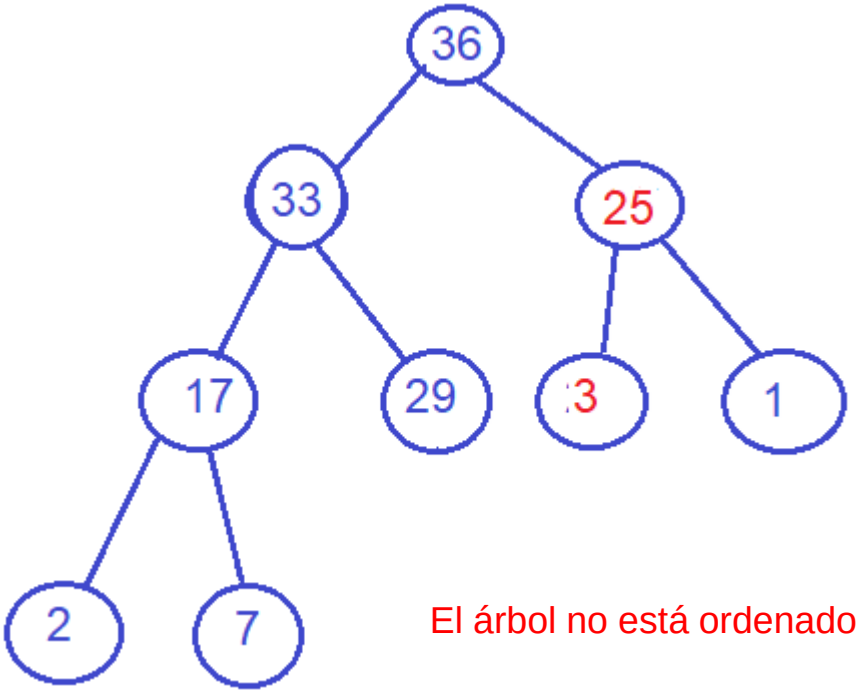


Clase 9: Heap Binaria

Operaciones:



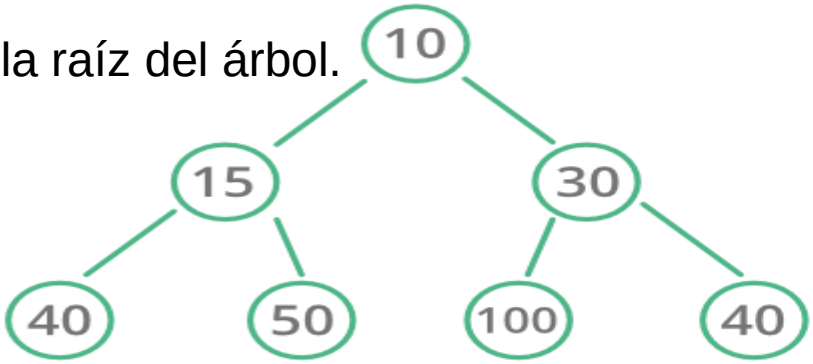
Intercambiamos el 3 por el 36
Reordenamos nuevamente
Realizando el último intercambio



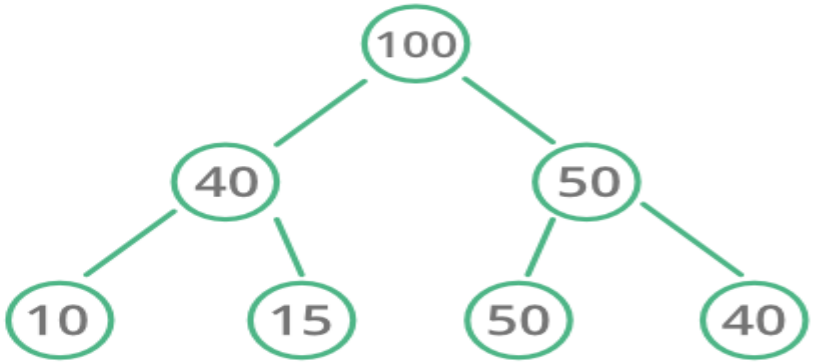
El árbol no está ordenado

Propiedad de Orden:

Es una heap o sea es un árbol binario ordenado por una condición de orden específica, lo que hace que sea un **"min-heap" o un "max-heap".** En un **min-heap**, cada nodo tiene un valor que es menor o igual que el valor de sus hijos. Esto implica que el nodo con el valor más pequeño siempre estará en la raíz del árbol. Por otro lado, en un **max-heap**, cada nodo tiene un valor mayor o igual que el valor de sus hijos, lo que significa que el nodo con el valor más grande siempre estará en la raíz del árbol.



Min Heap

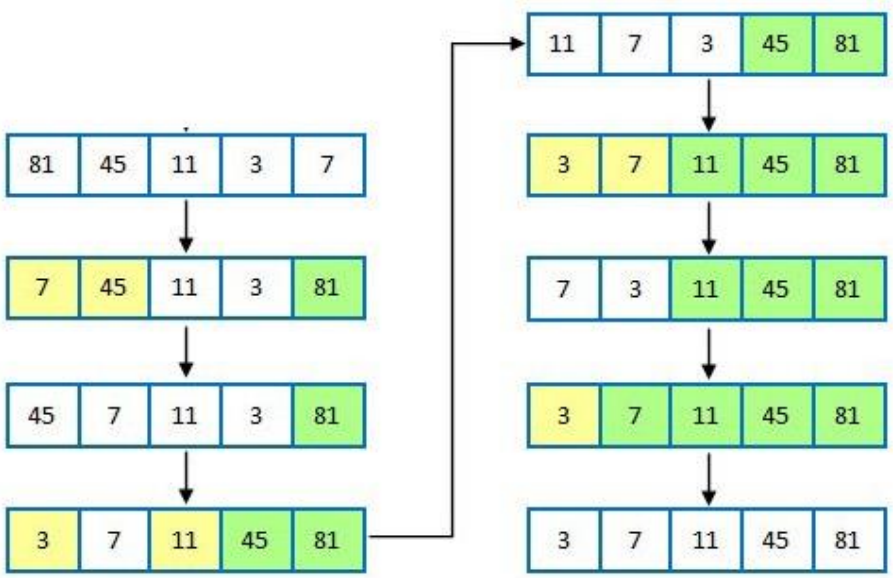


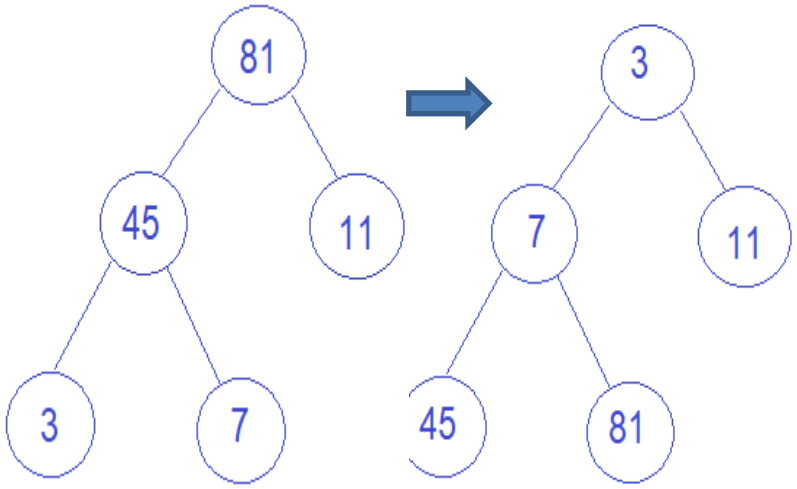
Max Heap

Clase 9: Heapsort

Heapsort presentada por un vector ordenado:

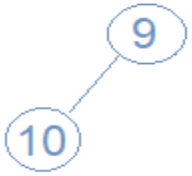
El Algoritmo creado por William Joseph Williams en 1964 para ordenar los elementos toma el mayor o menor (dependiendo si es un máximo o mínimo), es decir el que esta situado en la primer posición del vector y lo intercambia con el ultimo elemento del vector. Luego reubica el elemento colocado en la primera posición lo intercambia(para restablecer la propiedad de orden) y marca el ultimo elemento como ordenado decrementando el tamaño del vector. Y se repite sucesivamente este proceso con el resto de los elementos del vector hasta que el tamaño del vector desordenado es uno. Se puede observar el funcionamiento del algoritmo para ordenar un vector.



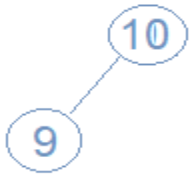


Heapsort ejemplo paso a paso:

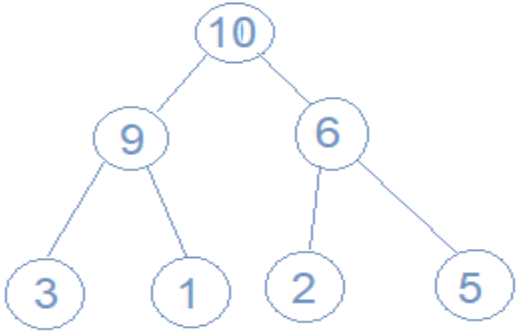
9	10	6	3	1	2	5



10	9	6	3	1	2	5

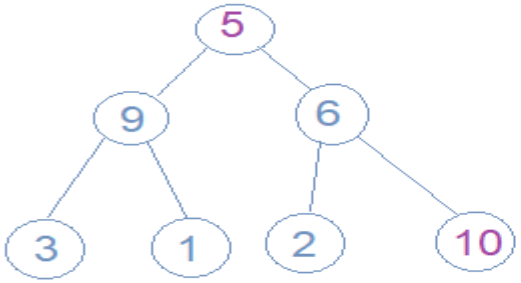


10	9	6	3	1	2	5

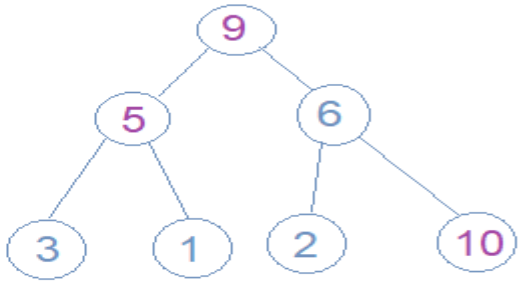


-Intercambiamos la raíz con el último elemento del vector
-Luego ordenamos la raíz con respecto a los hijos
-Intercambiamos nuevamente la raíz con el anteúltimo elemento del vector

5	9	6	3	1	2	10



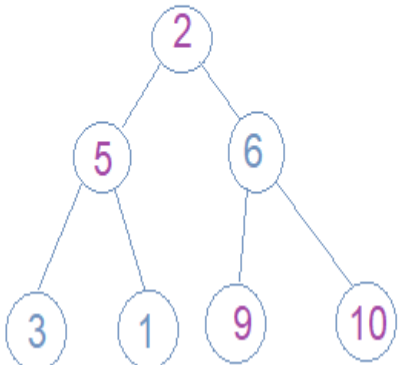
9	5	6	3	1	2	10



Clase 9: Heapsort

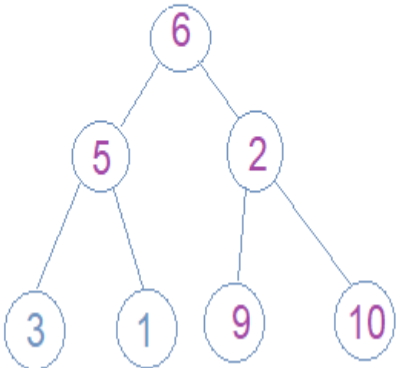
2 5 6 3 1 9 10

Se intercambia el 9 x el 2



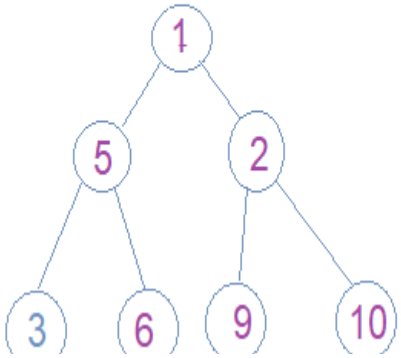
6 5 2 3 1 9 10

Ordeno la raíz con los hijos

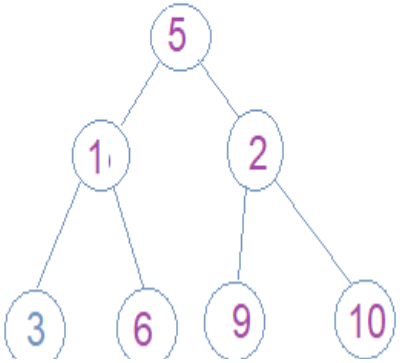


1 5 2 3 6 9 10

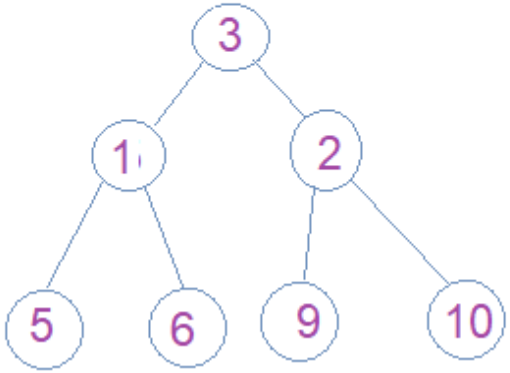
Intercambio la raíz con el 6



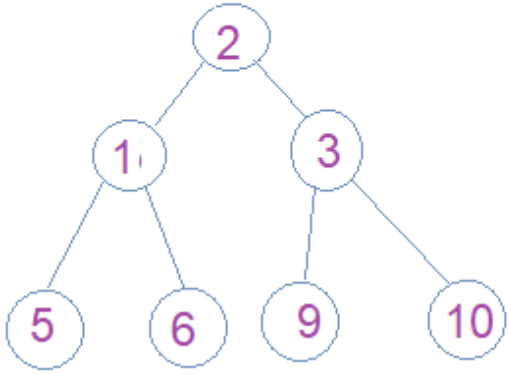
5 1 2 3 6 9 10



3 1 2 5 6 9 10

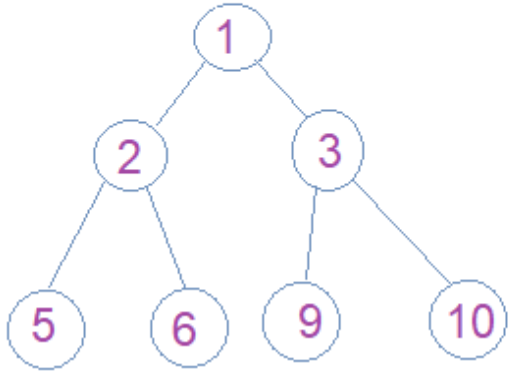


2 1 3 5 6 9 10



1 2 3 5 6 9 10

No quedan elementos por ordenar



Implementación de la Heap:

```
class Heap(object):
    def __init__(self,tamanio):
        self.tamanio = 0
        self.vector = [None]*tamanio
```

Una heap H consta de:

- *Un arreglo que contiene los datos*
- *Un valor que me indica el número de elementos almacenados*

Ventaja:

- No se necesita usar punteros
- Fácil implementación de las operaciones

Clase 9

unab

Consultas

Temas a desarrollar la próxima clase

- Análisis asintótico, comportamiento en el mejor caso, caso promedio y peor caso.
- Modelo computacional.
- Concepto de tiempo de ejecución. Notación O(), Ω, Θ. Reglas generales para el cálculo del tiempo de ejecución.
- Cálculo de tiempo y orden de ejecución en algoritmos iterativos y recursivos.
- Comparación de distintas estrategias de diseño de algoritmos.