UNIVERSIDAD DEL BÍO-BÍO

Estructura de Datos

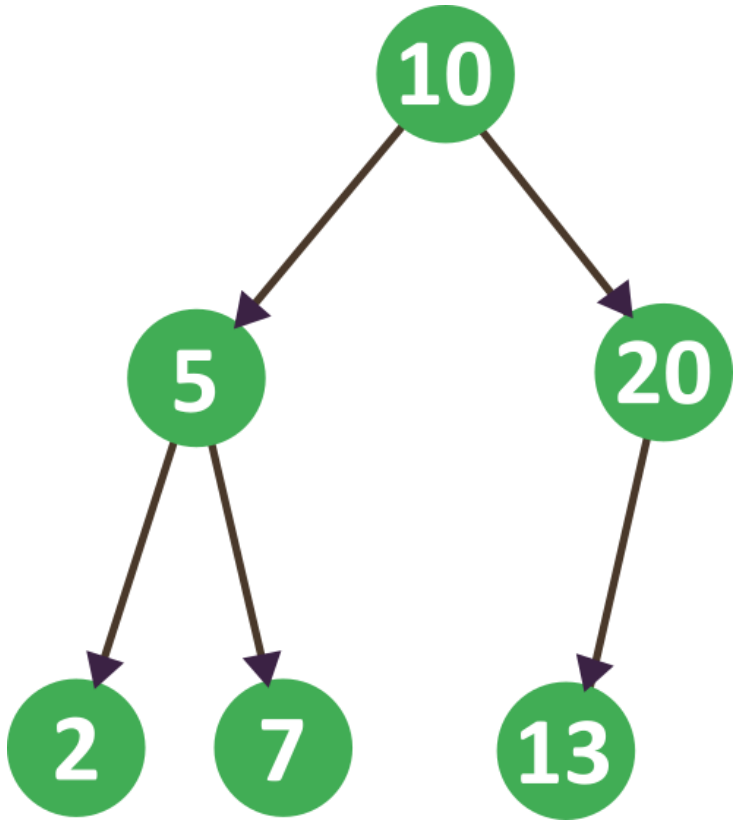
HeapSort

¿Qué es HeapSort?

HeapSort, es un algoritmo de clasificación utilizado en árboles binarios.

Se trabaja utilizando un **arreglo**, el cual se **representa** bajo un árbol binario **completo o semi-completo**.

Un árbol se dice semi-completo cuando todos su niveles (Menos el último) se encuentran con su capacidad máxima de nodos. Para el último nivel todos los nodos se encuentran desde la posición más a la izquierda sin dejar un "campo vacío"



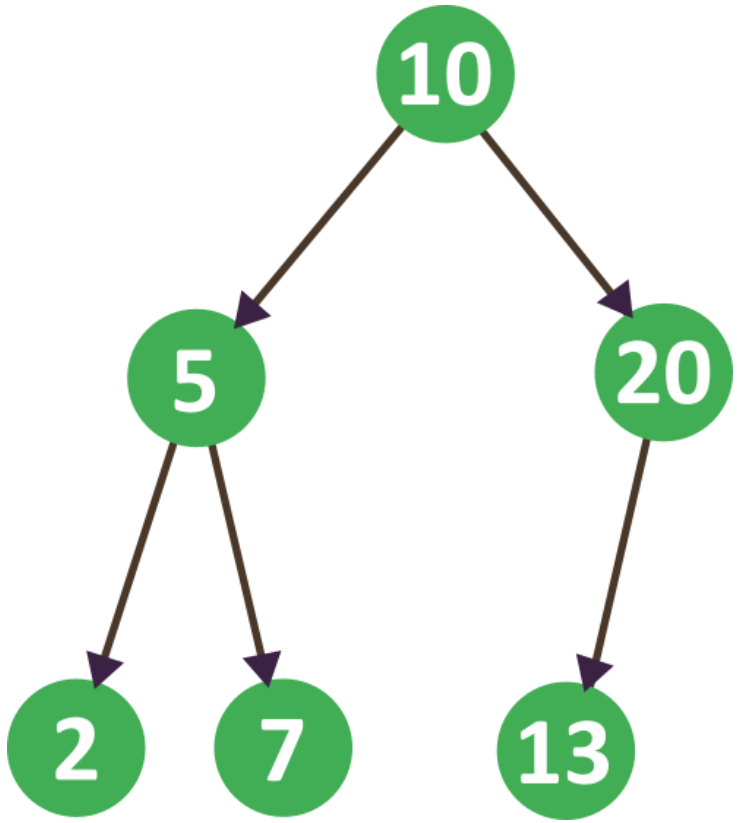
¿Para qué usar Árbol Heap?

Con un árbol Heap, podemos trabajar colas de prioridades a un bajo costo.

En la raíz siempre se encontrará el valor más bajo o más alto, dependiendo del criterio de prioridad.

Una de sus características es que el árbol no se encuentra "ordenado", pero bajo la lógica de cómo trabaja siempre en la raíz se encontrará el valor más bajo o más alto.

Observación: Este tipo de árbol puede tener valores repetidos.



Representación

Para representar un arreglo en un árbol binario, se distribuyen los datos como *"si fueran obtenidos en el recorrido en anchura".*

Ejercicio

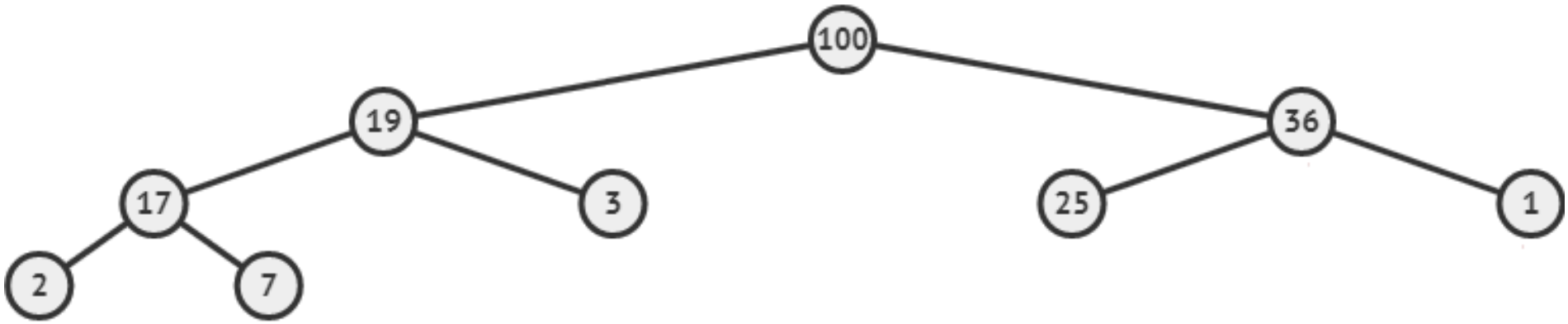
Teniendo el siguiente arreglo, representarlo en un árbol binario semi-completo.

100	19	36	17	3	25	1	2	7			
0	1	2	3	4	5	6	7	8	9	10	11

Observaciones:

1. La Raíz está en la posición 0.
2. El Hijo izquierdo de su padre se encuentra ubicado en : **2*i +1**
3. El Hijo derecho de su padre se encuentra ubicado en : **2*i +2**

Solución



Criterio de Orden (Prioridad)

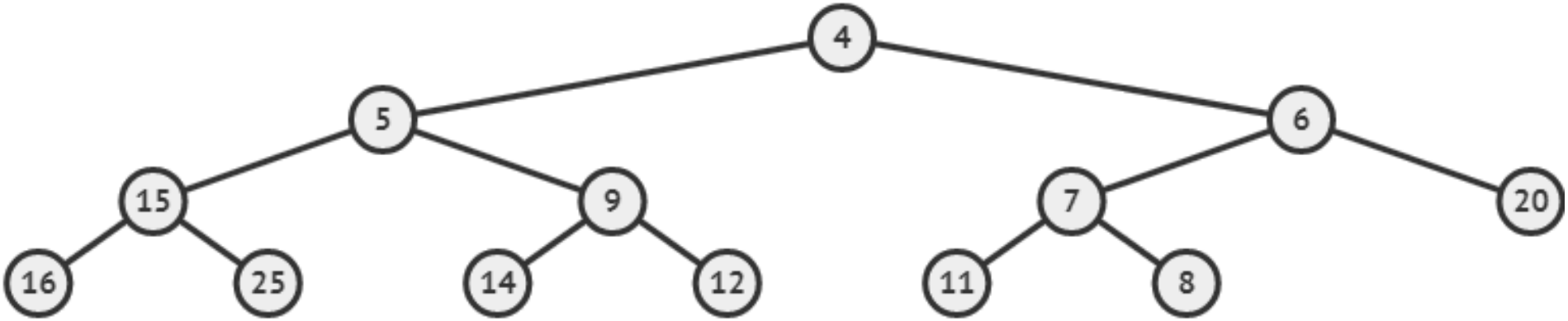
Mayor valor indica mayor prioridad : Será un árbol binario casi completo de n nodos en el que el contenido de cada nodo es menor o igual que el contenido de su padre.

Menor valor indica mayor prioridad : Es un árbol binario casi completo de n nodos en el que el contenido de cada nodo es mayor o igual que el contenido de su padre.

Criterio de Orden (Menor valor)

*Nodo Padre tiene el valor **más bajo** que sus hijos directos*

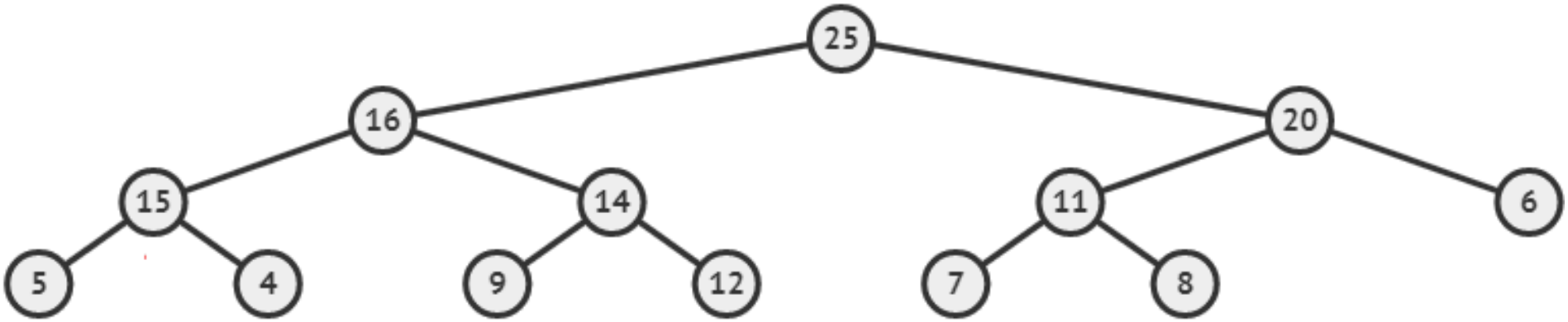
4	5	6	15	9	7	20	16	25	14	12	11	8
0	1	2	3	4	5	6	7	8	9	10	11	12



Criterio de Orden (Mayor valor)

*Nodo Padre tiene el valor **más alto** que sus hijos directos*

25	16	20	15	14	11	6	5	4	9	12	7	8
0	1	2	3	4	5	6	7	8	9	10	11	12



Propiedades

1. Es un árbol binario lleno, con **excepción** del nivel más bajo, el cual se rellena **de izquierda a derecha**.

2. Todo nodo es más prioritario que sus descendientes. Entonces, como la máxima prioridad está en la raíz, su búsqueda y eliminación es rápida.

3. Toda lista ordenada es un heap.

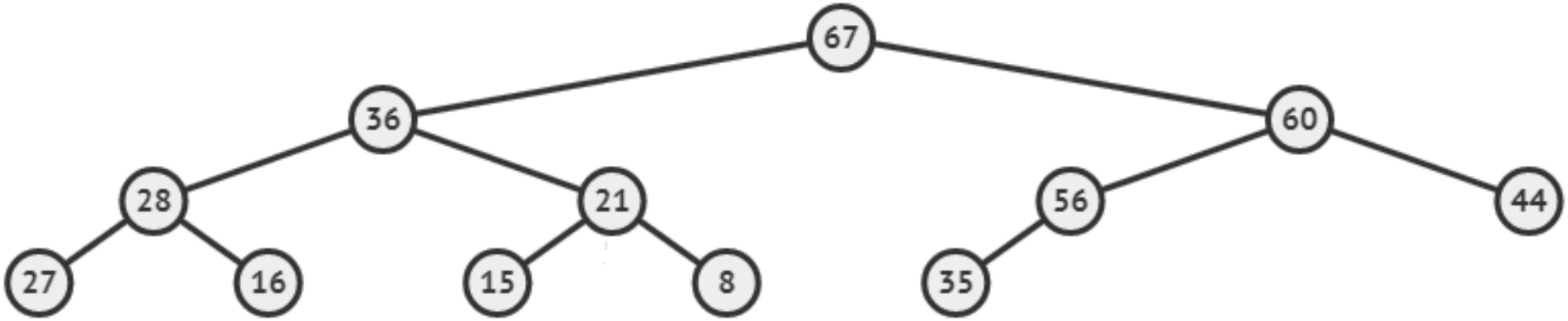
4. Complejidad computacional O(n Log n)

Ejercicio

Reconstruya el siguiente Heap

67	36	60	28	21	56	44	27	16	15	8	35
0	1	2	3	4	5	6	7	8	9	10	11

Solución



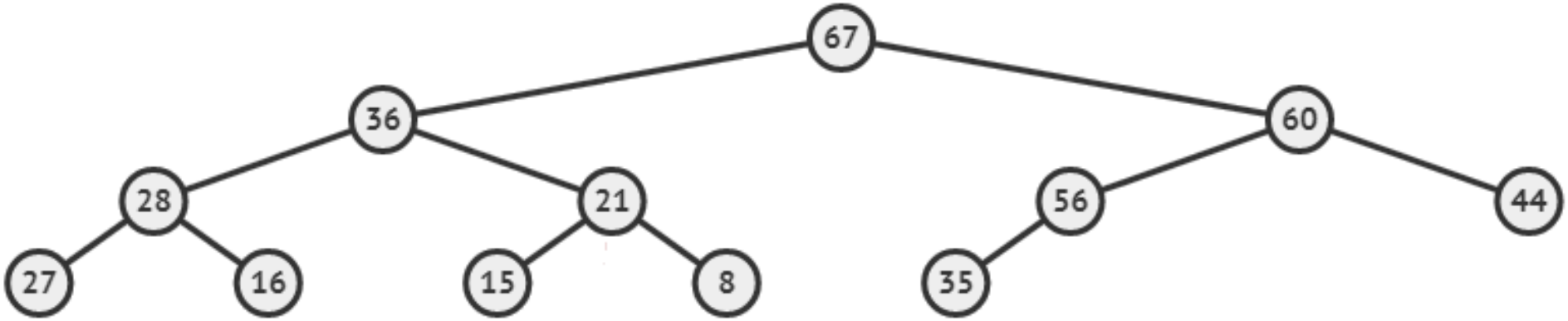
Inserción de un Nodo

1. Agregar el nodo en la primera posición disponible (lo más a la derecha del último nivel).

2. Comparar con su padre (Usar criterio correspondiente al Heap).
    1. Mayor valor: Si el nodo es mayor a su Padre intercambiar valores y repetir hasta llegar a su posición definitiva (donde es menor o igual al padre).
    2. Menor valor: Si el nodo es menor a su Padre intercambiar valores y repetir hasta llegar a su posición definitiva (donde es mayor o igual al padre).

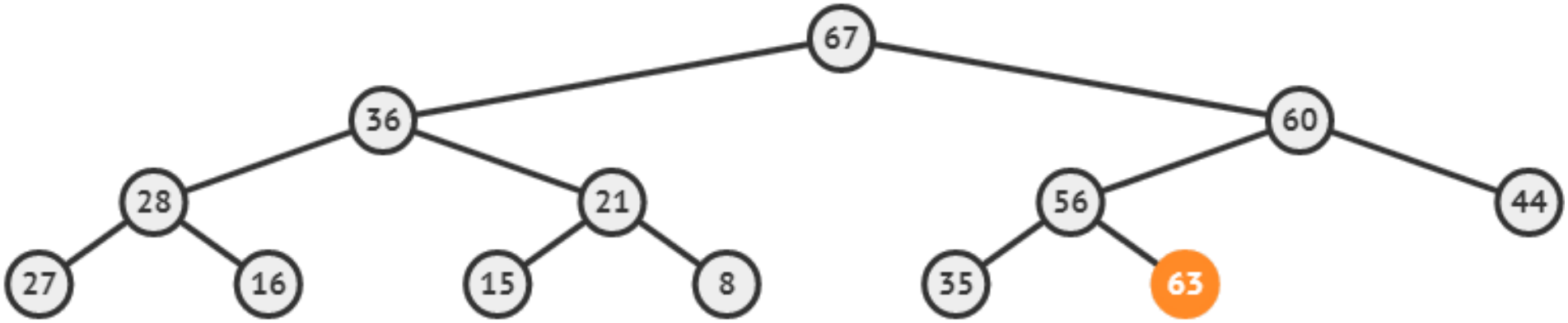
Ejercicio

*Utilizando el ejercicio anterior, inserte el valor **63***



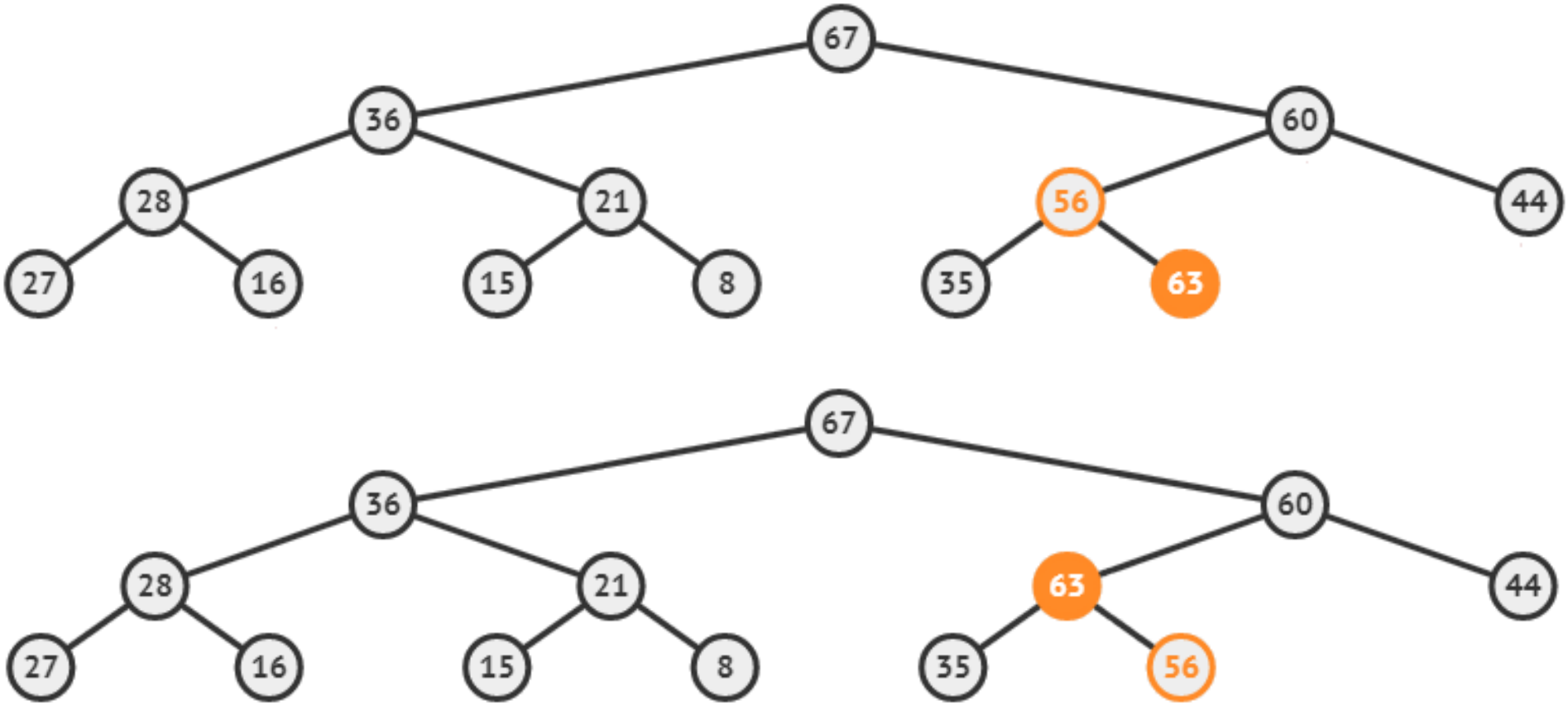
Ejercicio - Desarrollo

*Insertando el valor **63***



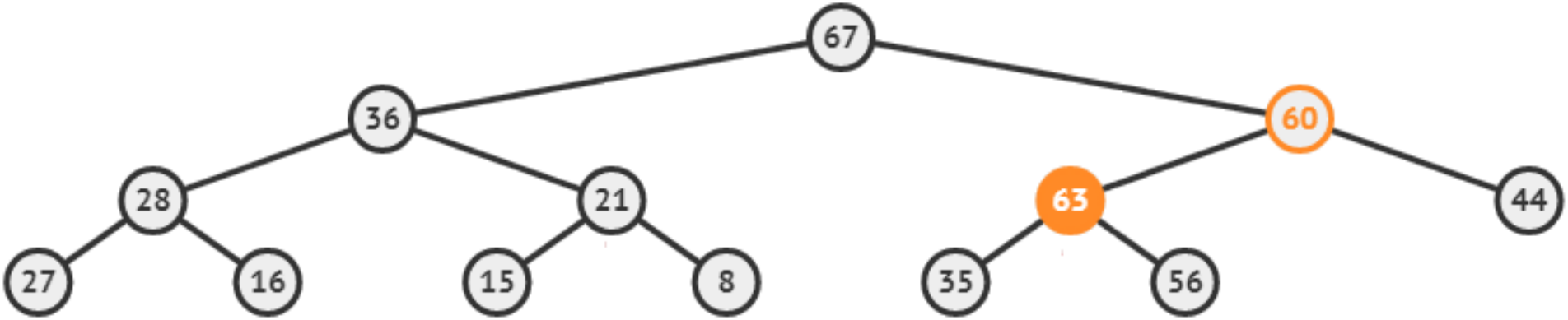
Ejercicio – Intercambiando valores

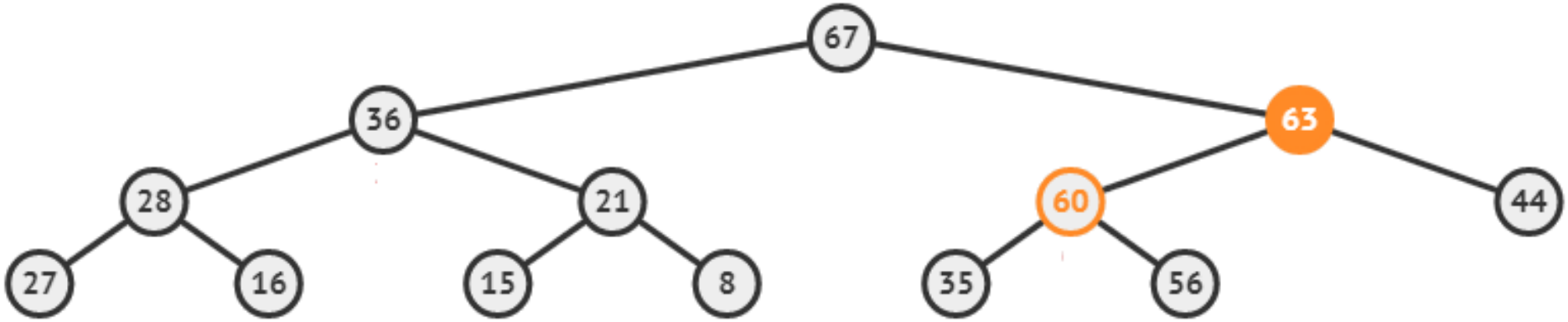
Intercambiando 56 - 63



Ejercicio – Intercambiando valores

Intercambiando 63 - 60





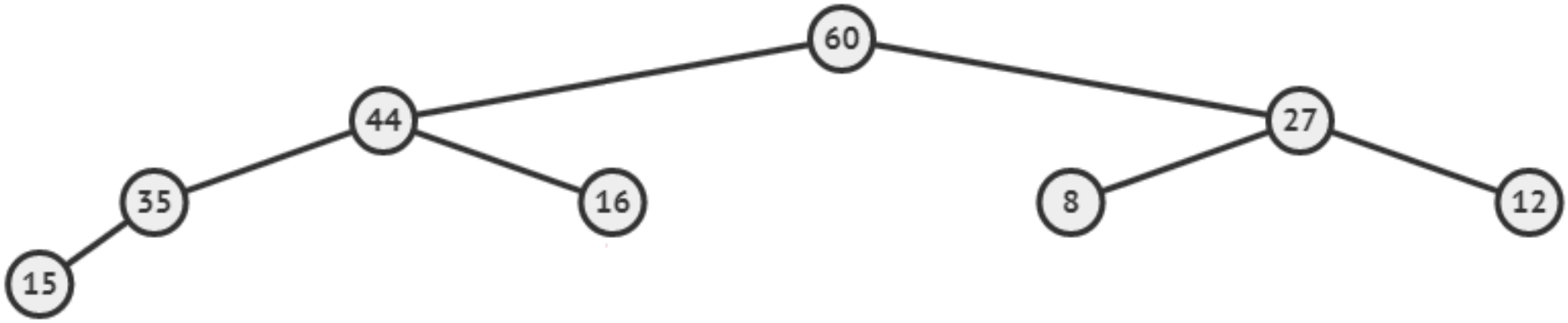
Resultado Final

Ejercicio - Inserción

Ingrese ***(No reconstruir)*** los siguiente valores en un heap vacío. Por último indique el arreglo resultado.
Observación: Prioridad Valor mayor

15, 60, 08, 16, 44, 27, 12, 35

Ejercicio - Solución



60	44	27	35	16	8	12	15				
0	1	2	3	4	5	6	7	8	9	10	11

Eliminación de un Nodo

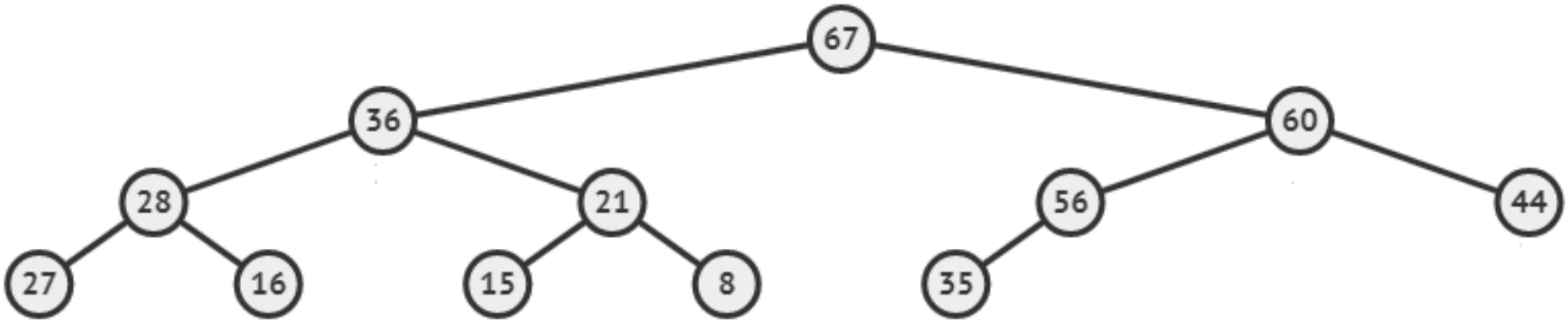
1. **Para cada eliminación siempre se elimina la raíz del heap.**
2. Una vez eliminada remplazar la raíz con el último nodo del último nivel.
3. Comparamos si los hijos de la nueva raíz son menores.
4. Si son menores no se hace ninguna permutación
5. Si son mayores (o uno de ellos) se hace permutación con el hijo mayor.

Ejercicio

Reconstruya *el siguiente Heap*

67	36	60	28	21	56	44	27	16	15	8	35
0	1	2	3	4	5	6	7	8	9	10	11

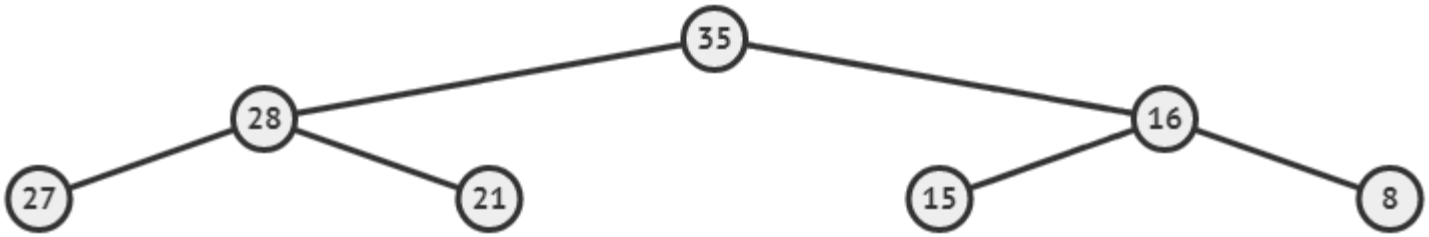
Ejercicio - Solución



Ejercicio - Eliminación

1. Para el heap solución, realice 5 eliminaciones.
   Observación: *Cuando se borra un elemento de un Heap, siempre se borra la Raíz.*
2. Genere una lista de elementos eliminados e indique cuál es la característica observada en la lista.

Ejercicio - Solución



35	28	16	27	21	15	8					
0	1	2	3	4	5	6	7	8	9	10	11

Arreglo Original

67	36	60	28	21	56	44	27	16	15	8	35
0	1	2	3	4	5	6	7	8	9	10	11

Lista de elementos eliminados

67	60	56	44	36							

Respuesta: Al eliminar un valor siempre me entregará el mayor, por lo tanto, si aplico eliminar hasta dejar el heap en **NULO**, obtendré una lista ordenada descendente.

Los ejemplos de esta clase están tomados del libro
"Estructuras de datos" de los autores Cairó & Guardati.
Ejemplares disponibles en biblioteca.

Para continuar practicando con ejemplos visuales animados visitar la página:

https://visualgo.net/en/heap