

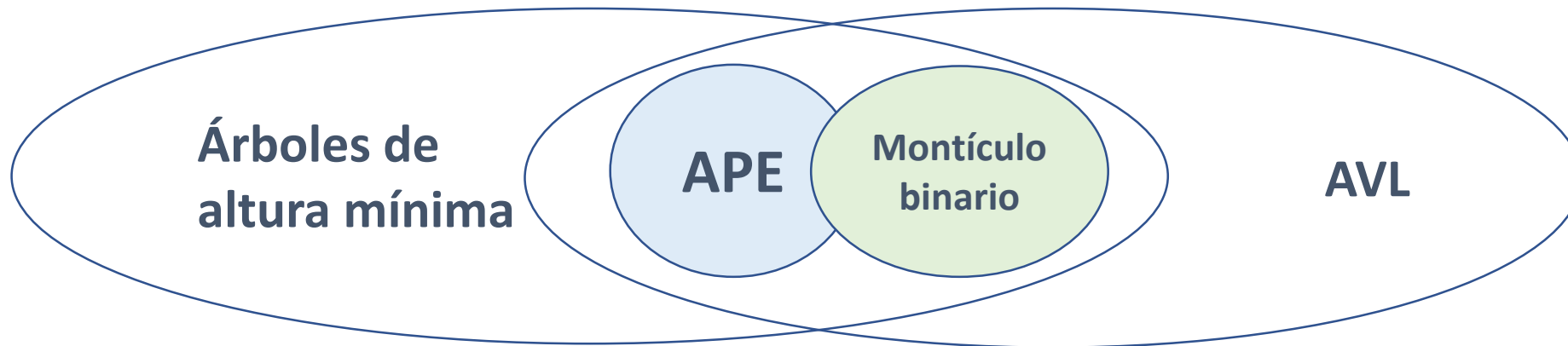
Colas de prioridad

- Objetivo
 - Modelar estructuras lineales en las que los elementos se atienden en el orden indicado por una prioridad asociada
 - Ejemplos de aplicación
 - Colas de impresión de documentos
 - Gestión de tráfico aéreo
 - Planificación de procesos en CPU
 - Supervisión planes estratégicos y de emergencia
 - Colas de Espera para Servicios Médicos
- Problema a resolver
 - Optimizar las operaciones de insertar y sacar el elemento de máxima prioridad
- Solución
 - Utilizar Montículos binarios para su implementación
 - Proporcionan complejidad logarítmica



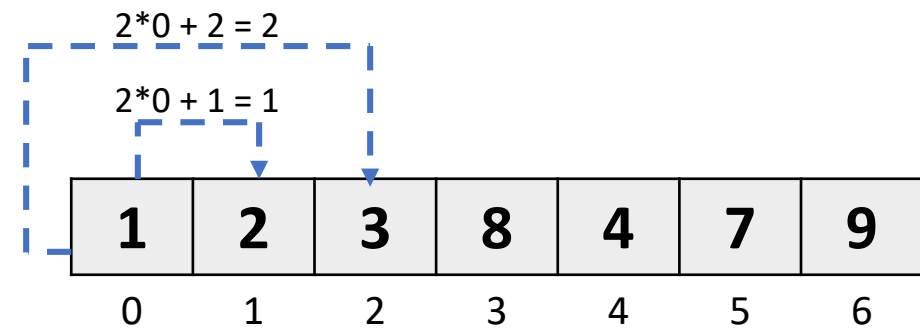
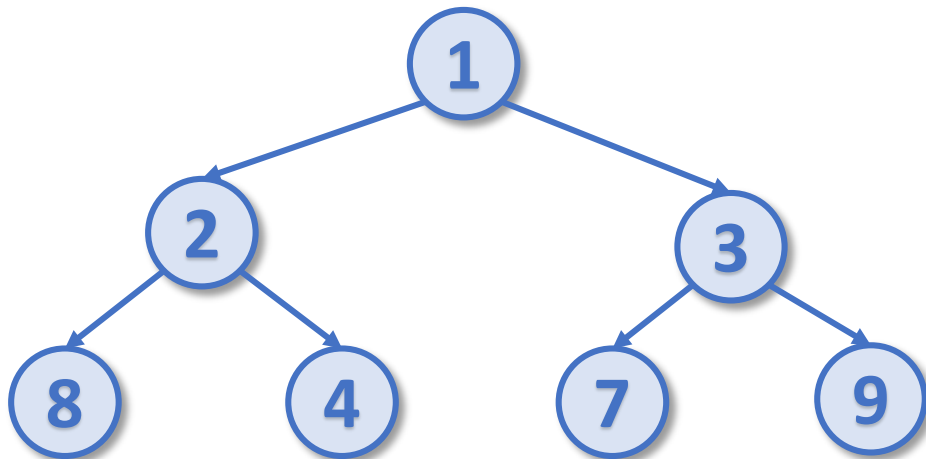
Montículo binario

- Árbol binario completo con excepción del nivel inferior que se rellena de izquierda a derecha
- Altura de un montículo binario
 - $h = E[\log_2 n] + 1$
- Propiedades



Montículo binario - Propiedades

- Dada la estructura fija del árbol binario, éste puede representarse en un vector sin necesidad de referencias
 - La raíz del árbol se almacenará en la primera posición del vector
 - Para cualquier nodo situado en un posición i , se cumple que:
 - Su hijo izquierdo se almacenará en la posición $2*i + 1$
 - Su hijo derecho se almacenará en la posición $2*i + 2$



Montículo binario – Relación de orden

- Suponemos claves repetidas
- Montículo de mínimos
 - Todo nodo tiene una clave **menor** o igual a la de sus hijos
 - El **menor elemento** se encuentra en la raíz (posición 0 del vector)
 - Optimiza las operaciones de añadir y obtener el mínimo
- Montículo de máximos
 - Todo nodo tiene una clave **mayor** o igual a la de sus hijos
 - El **mayor elemento** se encuentra en la raíz (posición 0 del vector)
 - Optimiza las operaciones de añadir y de obtener el máximo

Montículo binario – Relación de orden

- Suponemos claves repetidas
- **Montículo de mínimos**
 - Todo nodo tiene una clave **menor** o igual a la de sus hijos
 - El **menor elemento** se encuentra en la raíz (posición 0 del vector)
 - Optimiza las operaciones de añadir y obtener el mínimo
- Montículo de máximos
 - Todo nodo tiene una clave **mayor** o igual a la de sus hijos
 - El **mayor elemento** se encuentra en la raíz (posición 0 del vector)
 - Optimiza las operaciones de añadir y de obtener el máximo

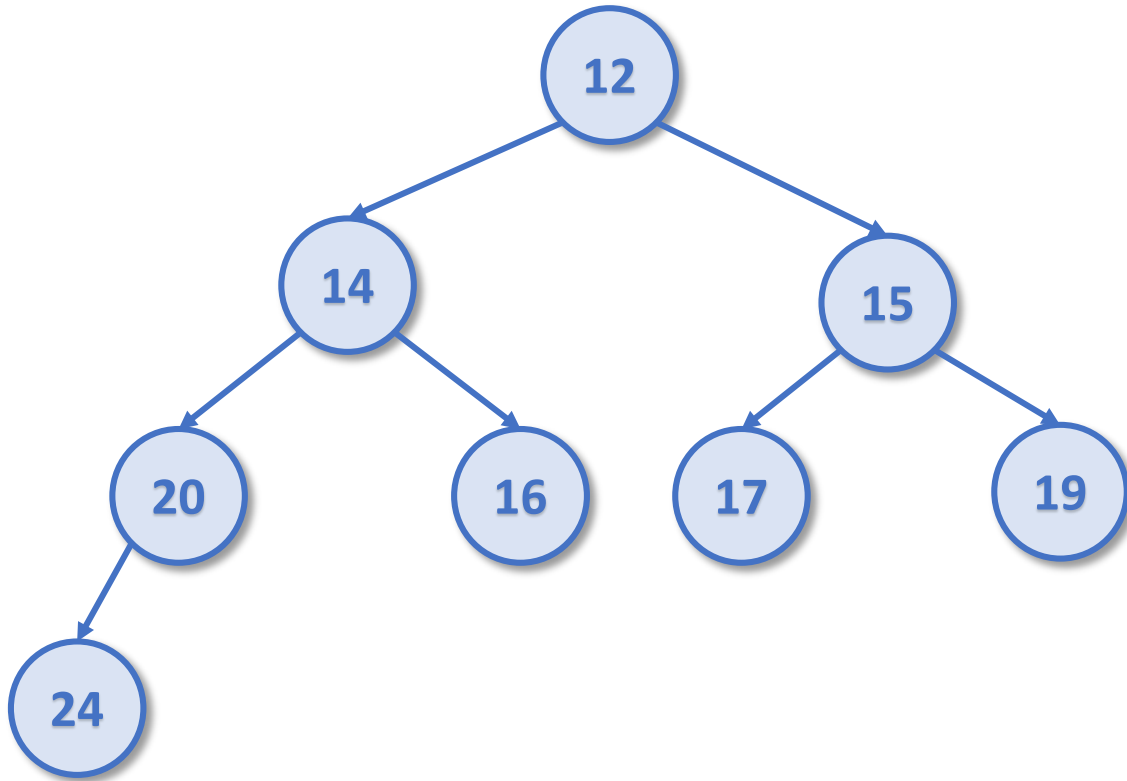
Montículo binario – Operaciones

- Añadir un elemento al montículo
- Sacar el elemento situado en la raíz
- Cambiar la prioridad de un elemento
- Eliminar un elemento del montículo

Montículo binario – Insertar

- Colocar el elemento a insertar en la última posición del vector
- Realizar un ***filtrado ascendente***
 - Si el elemento insertado es menor que su padre entonces intercambiarlos
 - Realizar la operación mientras el padre sea mayor que el hijo o elemento insertado y no hayamos alcanzado la raíz
 - El padre estará en la posición $E[(i - 1)/2]$
- Complejidad
 - Caso mejor: $O(1)$
 - Caso peor: $O(\log_2 n)$

Insertar – Ejercicio1



Insertar 18

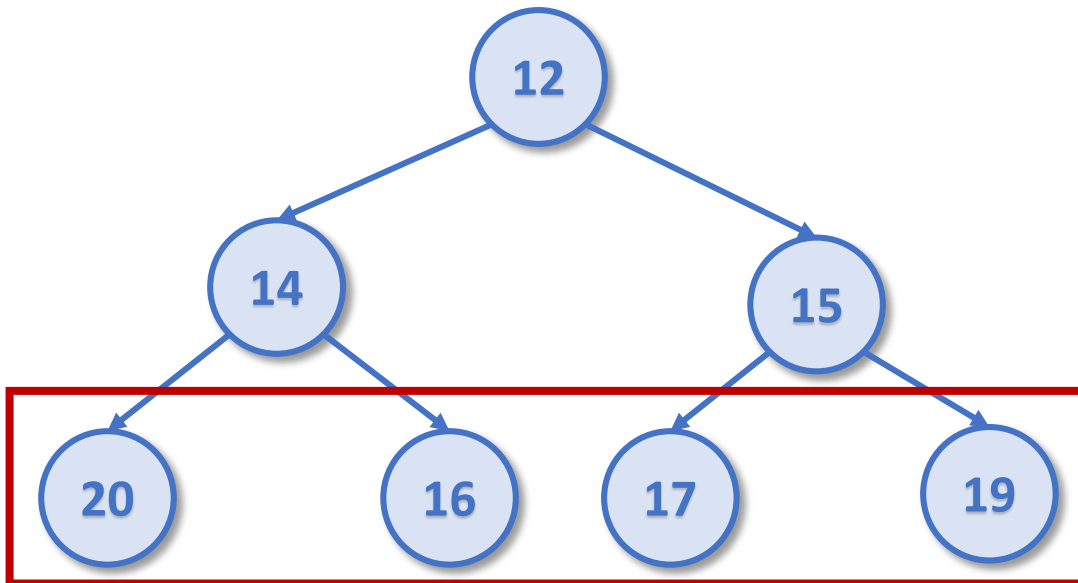
0	1	2	3	4	5	6	7	8	9
12	14	15	20	16	17	19	24		

Montículo binario – Sacar

- Devolver el mínimo, esto es, el elemento situado en la raíz (posición cero del vector)
- Colocar el último elemento del vector en la raíz o posición cero
- Realizar un ***filtrado descendente*** desde la raíz
 - Si el padre es mayor que uno de sus hijos entonces intercambiarlo
 - Si el padre tiene dos hijos y es mayor que los dos entonces intercambiarlo por el menor de los hijos
 - Realizar la operación anterior mientras el padre sea mayor que alguno de sus hijos o los dos y además no sea hoja
 - El hijo izquierdo, si existe, estará en la posición $2*i + 1$
 - El hijo derecho, si existe, estará en la posición $2*i + 2$
- Complejidad
 - Caso mejor: $O(1)$
 - Caso peor: $O(\log_2 n)$

Montículo binario – Devolver el máximo

- Los elementos de mayor peso se encuentran en las hojas por lo tanto sólo es necesario explorar la mitad del vector
- El máximo se encuentra en la zona del vector comprendida entre $[\text{numElementos}/2, \text{numElementos}]$
- Complejidad: $O(n)$

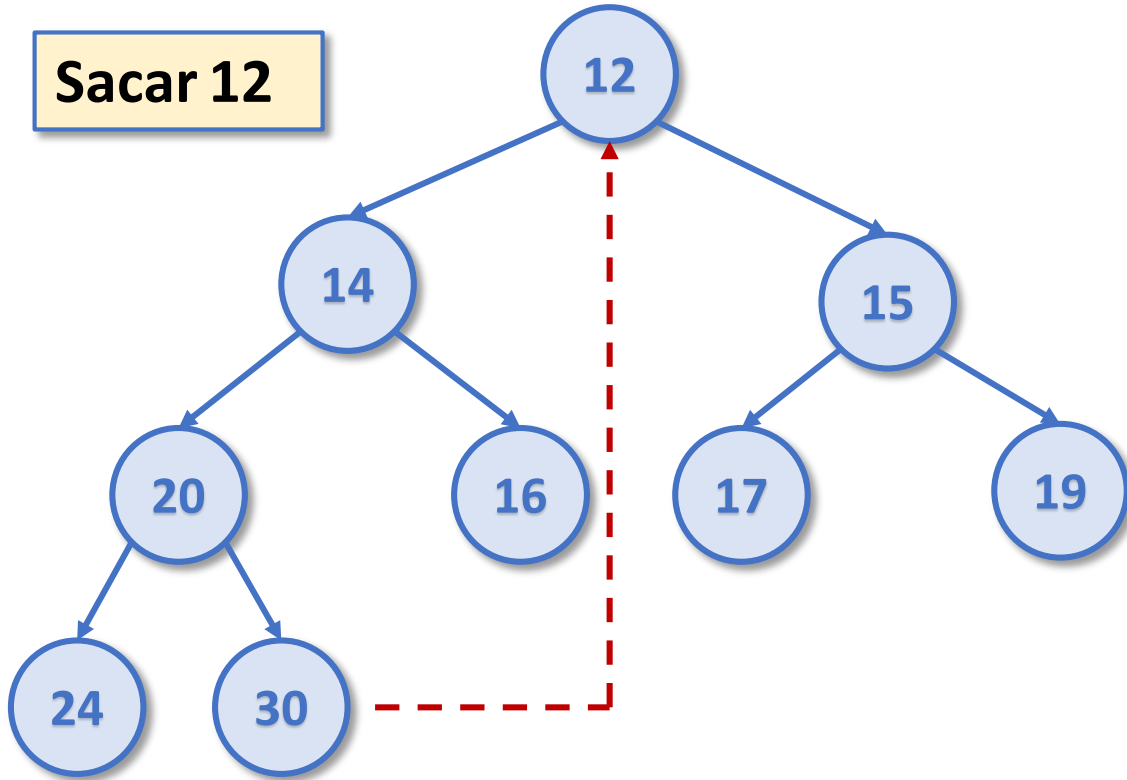


Rango: $[7/2, 7] = [3, 7]$

0	1	2	3	4	5	6	7	8	9
12	14	15	20	16	17	19			

Sacar – Ejercicio2

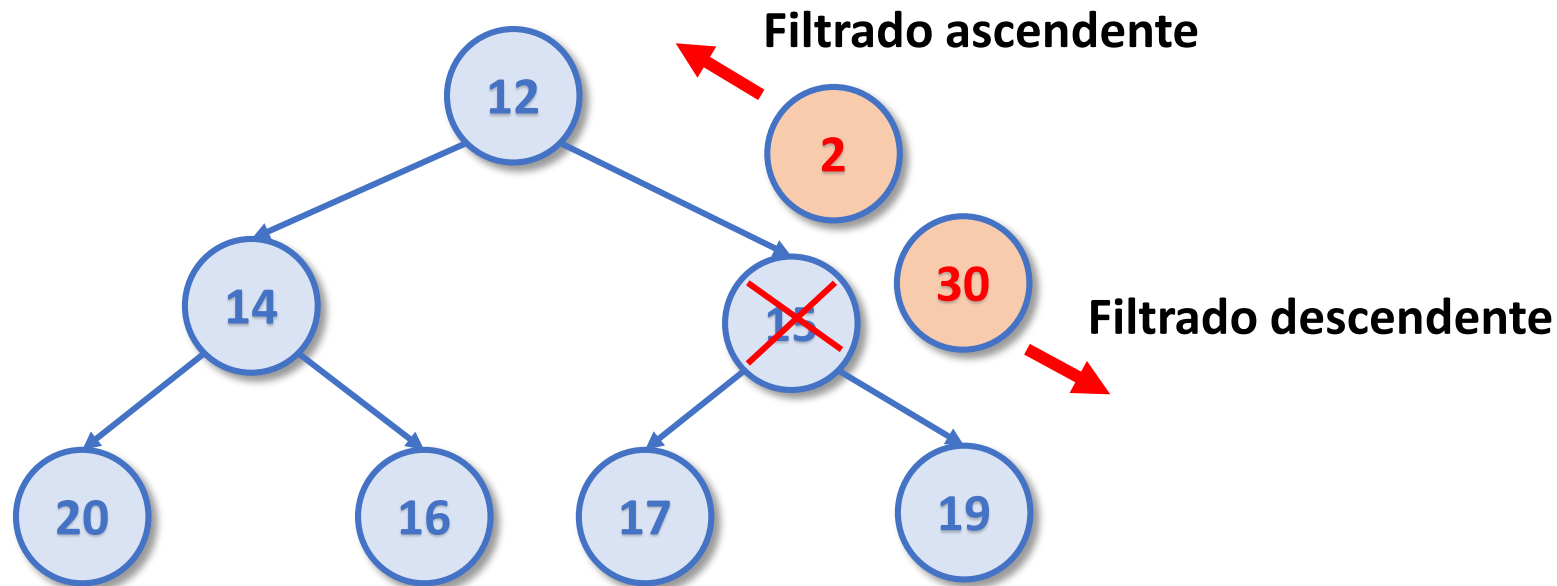
Sacar 12



0	1	2	3	4	5	6	7	8	9
12	14	15	20	16	17	19	24	30	

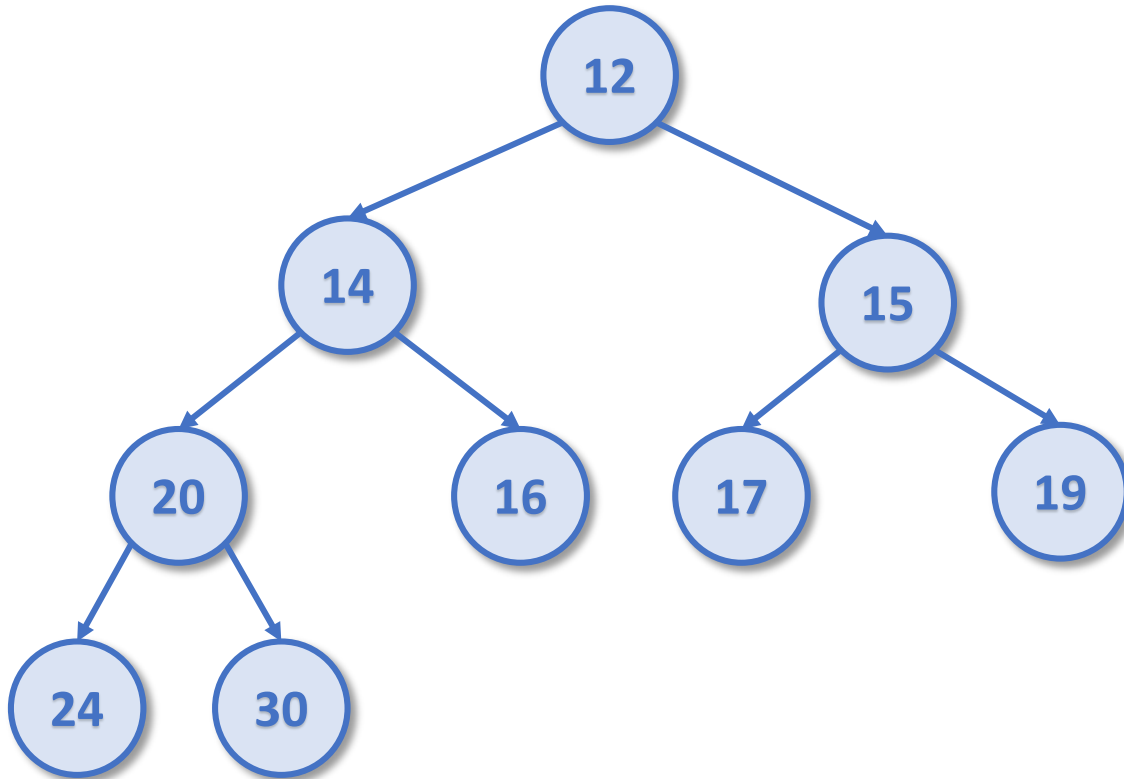
Montículo binario – Cambiar prioridad

- Si el nuevo valor es menor que el original \rightarrow filtrado ascendente
- Si el nuevo valor es mayor que el original \rightarrow filtrado descendente
- Complejidad: $O(\log_2 n)$



Cambiar prioridad – Ejercicio3

Cambiar prioridad de 20 a 5



0	1	2	3	4	5	6	7	8	9
12	14	15	20	16	17	19	24	30	

Montículo binario – Eliminar un elemento

- Dos métodos
 - Método1
 - Colocar el último elemento del vector en la posición del elemento a borrar
 - Realizar un ***filtrado descendente*** o ***filtrado ascendente*** desde la posición del elemento a borrar, dependiendo de lo siguiente:
 - Si el valor del último elemento es menor que el original → filtrado ascendente
 - Si el valor del último elemento es mayor que el original → filtrado descendente
 - Método2
 - Cambiar su prioridad a $-\infty$ (menor prioridad) para subirlo a la raíz
 - Invocar al método ***sacar()***
- Complejidad: $O(\log_2 n)$