Tecnología de la programación
Sesión 25

Ejercicios para la evaluación continua

1. Calcular la altura (profundidad) de un árbol binario

Objetivos de la sesión

1. Montículos (transparencias hasta el final del tema).
2. EJERCICIO: calcular el máximo de un árbol binario.
3. EJERCICIO: comprobar si un árbol es de búsqueda.
4. EJERCICIO: ver si un árbol es zurdo.

Guion

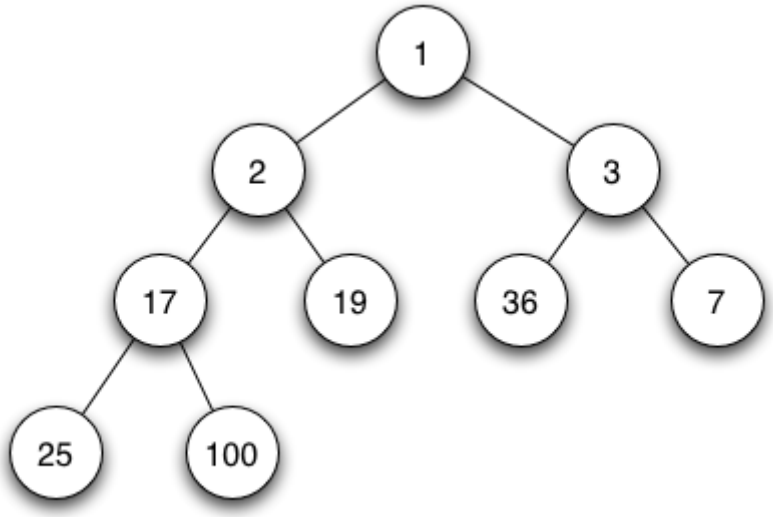
INTRODUCCIÓN

Un montículo (en inglés, heap) es otro tipo de árbol binario.

Concretamente, un montículo (de mínimos) es un árbol que:

- Es casi completo (no hay huecos)
- El elemento raíz es menor o igual que el resto de los elementos del árbol
- Los subárboles izquierdo y derecho son montículos

Un ejemplo:

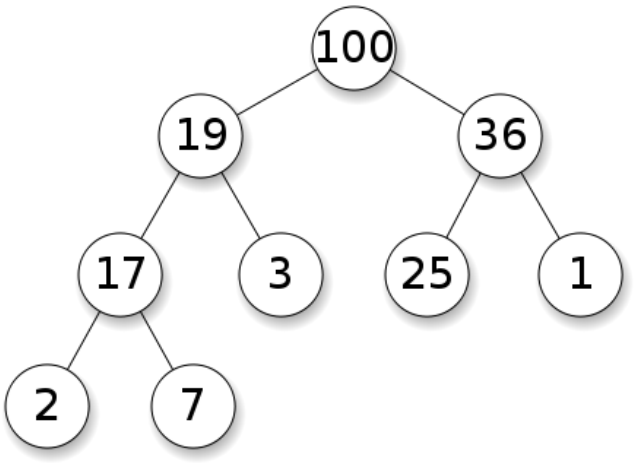


De nuevo hay que entender bien la definición: no basta con que todos los elementos que están por debajo de la raíz sean mayores, sino que también los subárboles izquierdo y derecho sean montículos. Es decir, que la propiedad se vuelva a cumplir para todos los subárboles, como si los tratásemos por separado.

Si nos damos cuenta, la raíz del árbol contendrá al mínimo.

También podríamos variar la definición y hacer un montículo de máximos. Simplemente hay que cambiar la segunda propiedad por "la raíz es mayor o igual que el resto de elementos".

Esto es un ejemplo de montículo de máximos:



La ventaja de los montículos es que es muy fácil conseguir el mínimo (o el máximo, si tenemos un montículo de máximos). Por esta razón, los montículos son útiles para implementar colas de prioridad.

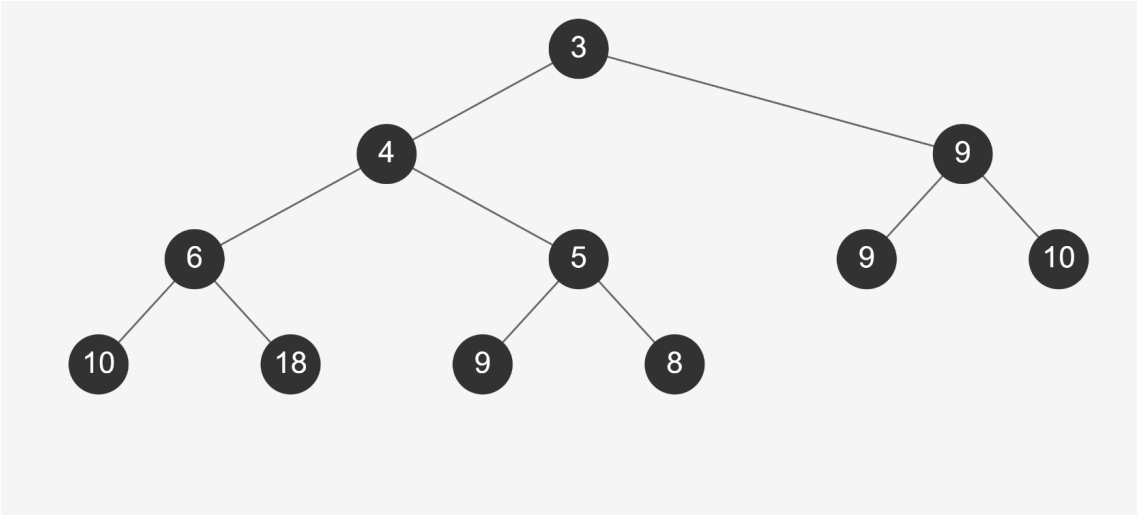
ESPECIFICACIÓN DEL TAD MONTÍCULO

```
especificación TAD Montículo
parámetros
    géneros
        tElemento
    operaciones
    función esMenor(d1: tElemento, d2: tElemento) dev booleano
    {Pre:}
    {Post: Devuelve VERDAD si d1 es menor que d2, y FALSO en caso
contrario}

    acción permutar(e/s d1: tElemento, e/s d2: tElemento)
    {Pre:}
    {Post: Intercambia los valores de d1 y d2}
géneros
      montículo
operaciones
operaciones
acción iniciarMontículo(sal/ montículo M)
{Pre:}
{Post: inicia M como el montículo vacío}

acción insertar(e/s montículo M, e/ tElemento d)
{Pre: M ha sido iniciado previamente}
{Post: inserta en el montículo M el elemento d}
```

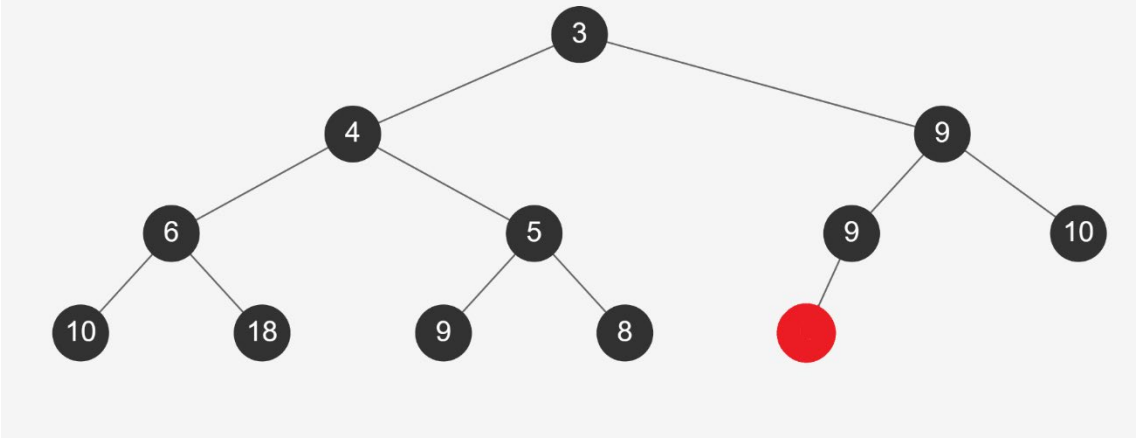
```
función mínimo (montículo M) dev tElemento
{Pre: M ha sido iniciado previamente}
{Post: devuelve el elemento más pequeño del montículo M}

acción eliminarMinimo(e/s montículo M)
{Pre: M ha sido iniciado previamente}
{Post: elimina del montículo M el elemento mínimo}

función monticuloVacio (montículo M) dev booleano
{Pre: M ha sido iniciado previamente}
{Post: devuelve VERDAD si M está vacío y FALSO en caso contrario}

función alturaMontículo (montículo M) dev entero
{Pre: M ha sido iniciado previamente}
{Post: devuelve la altura del montículo M}
fin_especificación
```

Tal y como aparece en la transparencia 62, está muy claro dónde toca añadir un elemento en un montículo.

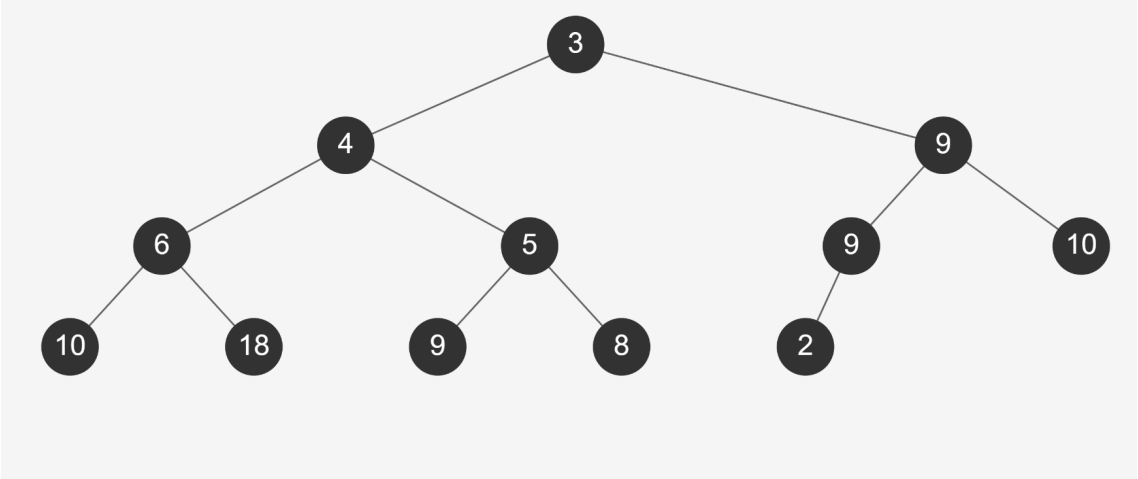
Por ejemplo, lo siguiente es un montículo (de mínimos):



Si ahora queremos insertar un nuevo nodo, está claro cuál es el hueco que le corresponde (el rojo):



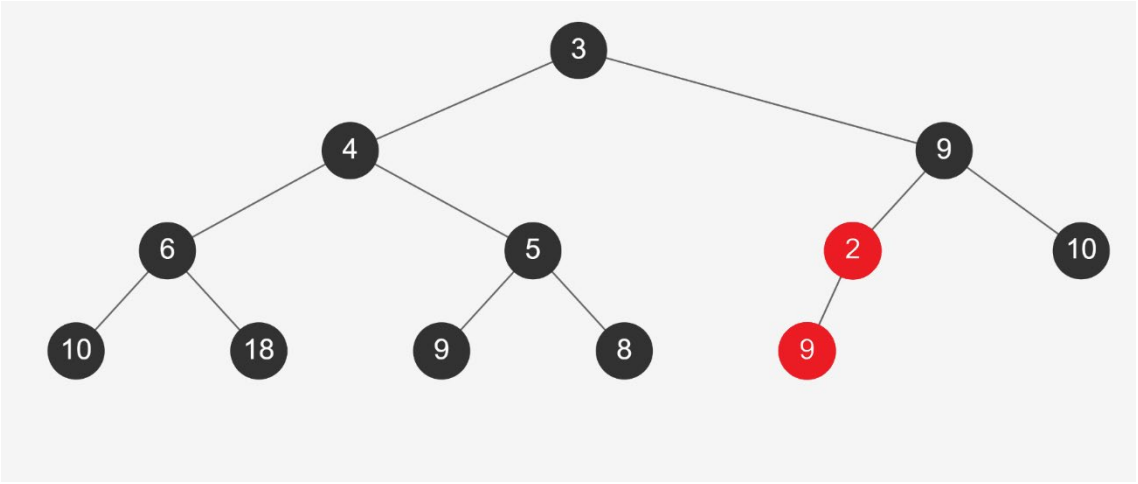
Sin embargo, lo que casi siempre ocurre es que al insertar perdamos la propiedad de montículo. Por ejemplo, si insertamos un 2, ya no tenemos un montículo.



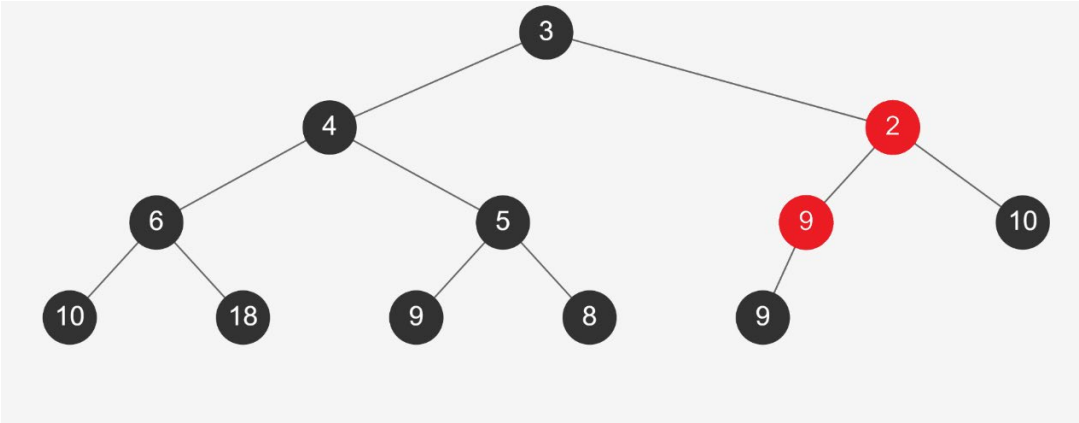
Para solucionar esto, nos tendremos que hacer una operación llamada flotar, que lo que haga sea "subir" el elemento en el árbol hasta donde le corresponda.

Para ello iremos intercambiando el 2 con su nodo padre, hasta conseguir la propiedad de montículo.

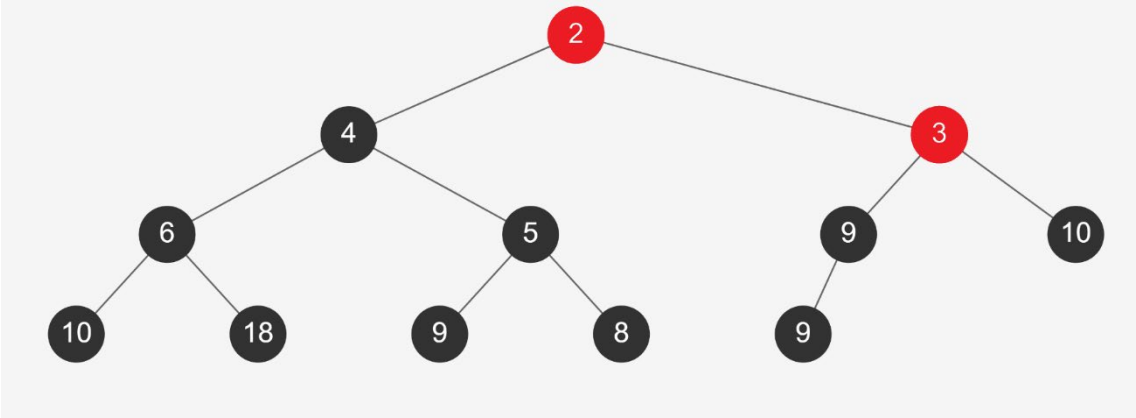
Paso 1: intercambiamos el 2 con el 9 (pongo en rojo el intercambio)



Paso 2: intercambiamos el 2 con el otro 9 (pongo en rojo el intercambio)



Paso 3: intercambiamos el 2 con el otro 3 (pongo en rojo el intercambio)



De esta forma, conseguimos volver a tener la propiedad de montículo (en este ejemplo lo he forzado mucho intencionadamente, insertando un elemento que debería ir arriba del todo).

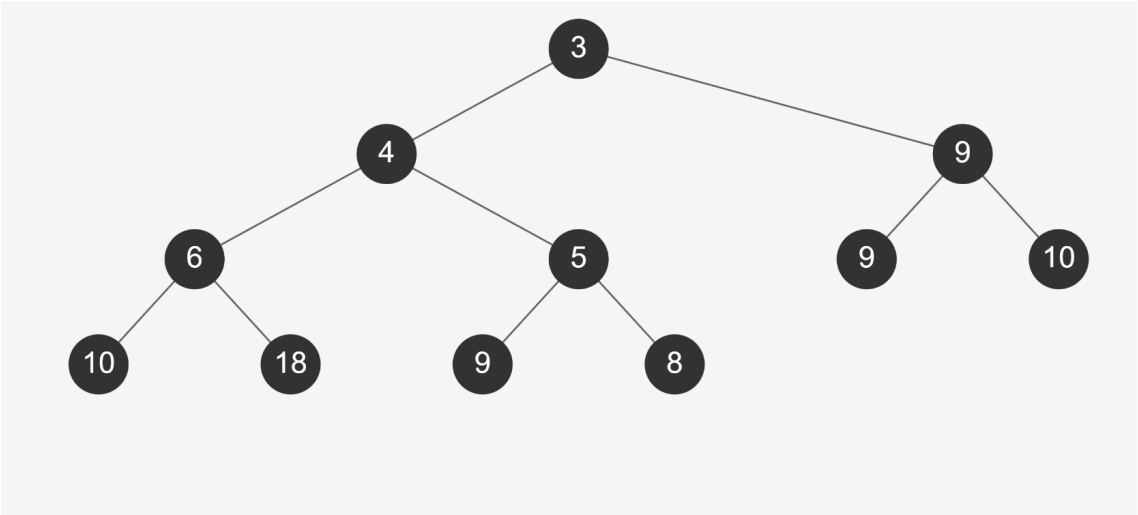
¿Y para eliminar?

En el TAD, la operación eliminar lo que hace es quitar el mínimo (el elemento raíz). El problema es que otra vez tienes que "reconstruir" el montículo. ¿Qué ponemos ahora de elemento raíz, una vez eliminamos lo que había?

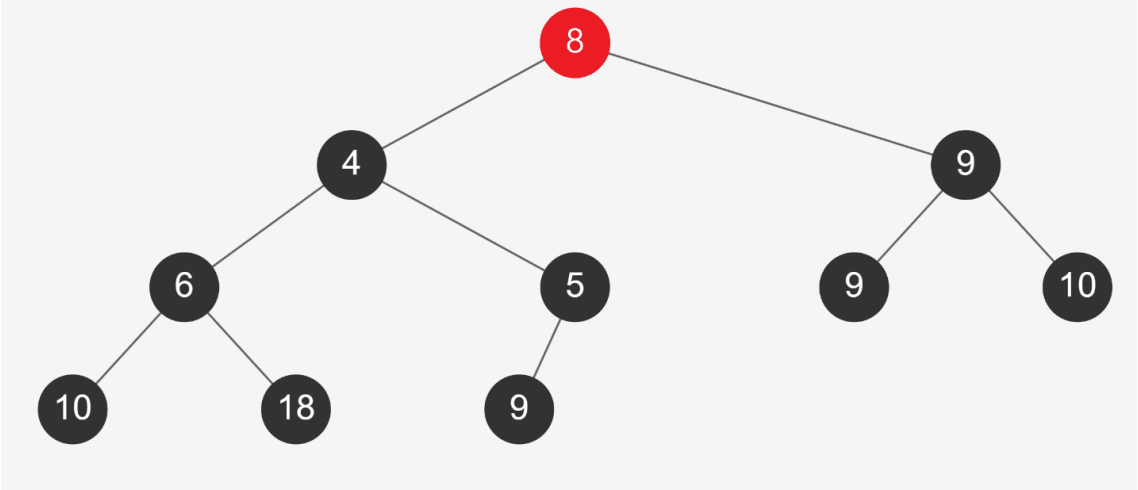
Lo más fácil es lo siguiente:

- Eliminamos la raíz
- Ponemos de raíz el último hijo
- Lo "hundimos" en el árbol, hasta ponerlo en su posición correcta, por ejemplo, intercambiándolo con el menor de sus hijos recursivamente (estilo el insertar, pero esta vez bajando en el árbol).

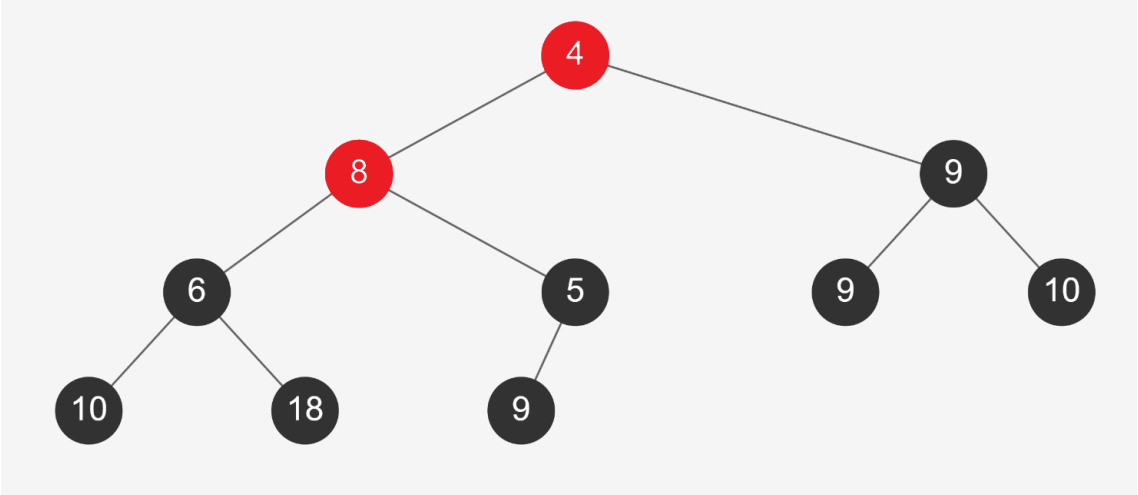
Por ejemplo, en el árbol inicial:



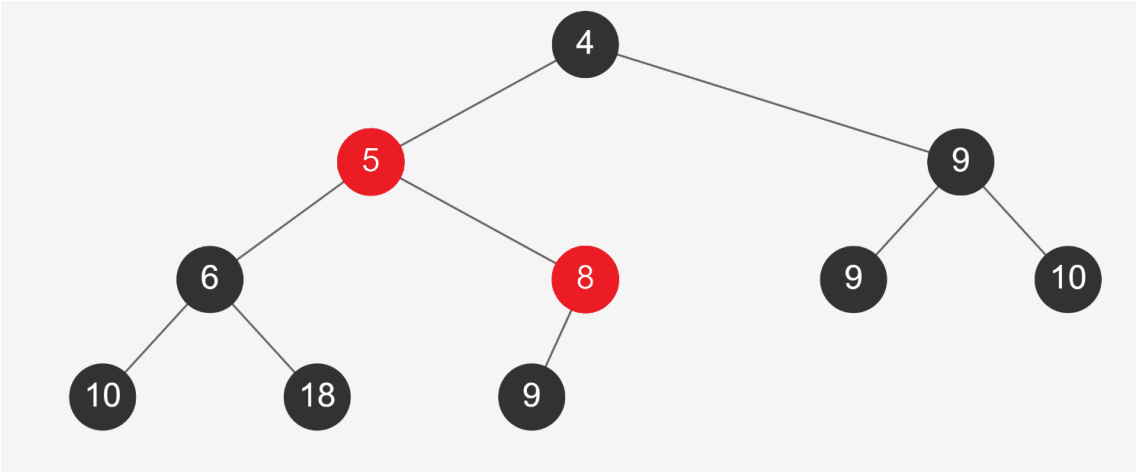
Eliminamos el 3. Entonces, colocamos el 8 arriba del todo:



Y ahora lo hundimos intercambiándolo con el menor de sus hijos.



Y volvemos a intercambiar hasta que ninguno de los hijos sea menor:



SOBRE MÉTODOS DE ORDENACIÓN

Si nos dan un vector desordenado, si lo pasamos a un montículo (insertando cada dato con la operación insertar) y una vez construido extraemos cada vez el mínimo, podemos construir un vector ordenado muy fácilmente. ¡¡Esencialmente eso hace el Heapsort que tanto nos costó entender a principio de curso!!

Fijémonos que las operaciones flotar y hundir se pueden hacer en O(logn), porque cada vez recorres solo una rama (te olvidas de la otra mitad del árbol en cada paso). Esto hace que el insertar y hundir se puedan hacer en tiempo O(logn).

Así que para ordenar un vector v de tamaño n, podemos hacer lo siguiente:

```
para i = 0 hasta n-1 hacer
      insertar(M, v[i])
fpara
```

Construir el montículo.

Complejidad O(n) * O (logn) = O(nlogn)

```
para i = 0 hasta n-1 hacer
      v[i] = minimo(M)
      eliminarMinimo(M)
fpara
```

Extraer el mínimo todo el rato y guardarlo en el vector.

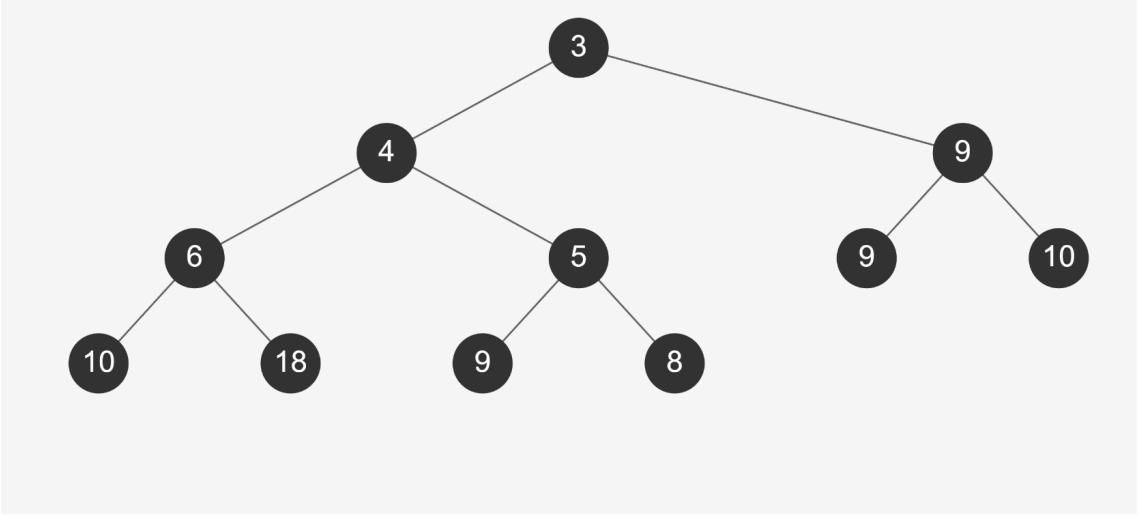
Complejidad O(n) * (O(1) + O (logn)) = O(nlogn)

DESVENTAJA: necesitamos por una parte tener el vector, y por otra el montículo. Es decir, necesitamos el doble de memoria.

Normalmente, para representar los montículos no se suele utilizar una representación dinámica, sino estática basada en vectores. De esta forma, además, podremos hacer las operaciones directamente sobre el vector de forma que la ordenación no necesite espacio extra en memoria

Podemos fácilmente representar los montículos como vectores, fundamentalmente gracias a la primera propiedad de los montículos: son un árbol casi completo, sin huecos.

La idea de representar un montículo como un vector es la siguiente. Dado el montículo



Su representación es el vector [3,4,9,6,5,9,10,10,18,9,8]. Es decir, hemos ido cogiendo los elementos nivel a nivel y de izquierda a derecha.

Si tenemos n elementos, los nodos del árbol ocupan las posiciones desde la 0 hasta la n-1 del vector. Los hijos del elemento que ocupa la posición i, están en 2i+1 y 2i+2; el padre del que ocupa la posición k, está en (k-1) div 2.

Por ejemplo:

- Los hijos del 6 (posición 3 en el vector), están en las posiciones 7 (que se corresponde con 2*3+1) y 8 (que se corresponde con 2*3+2), es decir, son el 10 y el 18.
- El padre del 8, que ocupa la posición 10, será el que esté en la posición 4 (que se corresponde con la operación (10 – 1) DIV 2). Es decir, será el 5.

Tenéis la implementación habitual de montículos (estática) en una hoja que está en la carpeta del Tema 7. Echarle un ojo, no os lo tenéis que aprender de memoria pero sí entenderlos.

<u>EJERCICIOS (intentarlos primero, pondré las soluciones más adelante)</u>

1. Calcular el máximo de un árbol binario.
2. Comprobar si un árbol es de búsqueda.
3. Hoja 8 ejercicio 12: ver si un árbol es zurdo.

¡¡CON ESTO ACABA LA TEORÍA!!!