Laboratorio A.E.D.

Lars-Åke Fredlund
lfredlund@fi.upm.es
Tonghong Li
tonghong@fi.upm.es
Manuel Carro Liñares
mcarro@fi.upm.es
Germán Puebla Sánchez
german@fi.upm.es
Pablo Nogueira
pnogueira@fi.upm.es

Viernes 11:00-13:00

Entrega

- La fecha límite para optar a la máxima nota es
  Viernes 14 de diciembre de 2012, a las 13:00 horas
- Los ficheros que hay que subir son `Heap.java` y `Sort.java` (son dos ficheros)
- La entrega se hace a través de la siguiente URL:
  `http://lml.ls.fi.upm.es/~entrega`
- El paquete `heaps` esta documentado con Javadoc en
  `http://babel.ls.fi.upm.es/~fred/courses/aed/heaps/`
- El proyecto debe compilar sin errores, cumplir la especificación y pasar el `Tester`.

Configuración

- Arrancad Eclipse.
- Cread un paquete `heaps` en el proyecto `aed`, dentro de `src`.
- Aula Virtual → AED → Sesiones de laboratorio → Laboratorio 9 → codigo_lab9.zip (formato zip).
- Importad al paquete `heaps` los fuentes que habéis descargado
- Ejecutad `Tester`. Veréis que lanza una excepción:

```
Beginning testing of class Heap...

*** Error: the minimal element in the queue should have key 5
but has key 9
...
```

Tareas para hoy

Hoy trabajaremos con montículos (heaps).

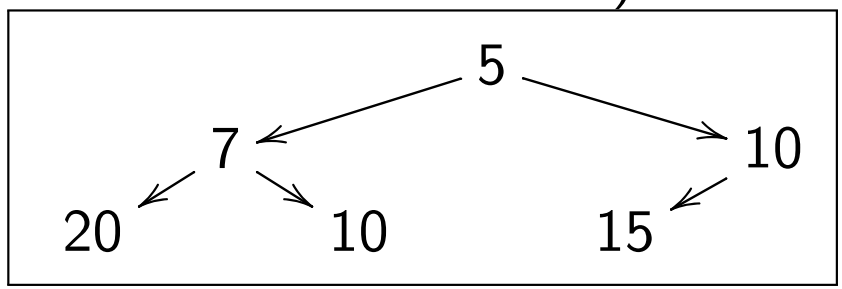
- Debéis completar el método `void downHeap()` dentro la clase `Heap<K,V>`. Dicho método es llamado desde `removeMin()` que borra la entrada con clave mínima del montículo.
- La clase `Sort` implementa un método

  `static <K> void sort(K[] arr, Comparator<K> comp)`

  que usa los métodos de la clase `Heap<K,V>` para ordenar el array `arr` en orden ascendente.

Montículos

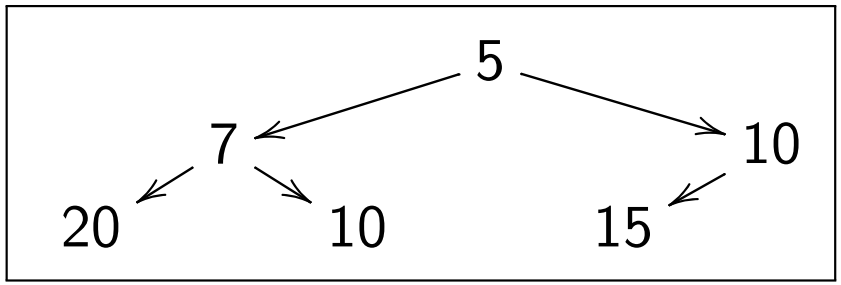
- Un montículo mínimo es un árbol (casi)completo en el que la clave de la entrada almacenada en un nodo es mayor o igual que la clave de la entrada almacenada en su nodo padre (si tiene padre).
- Ejemplo (**sólo mostramos las claves**):



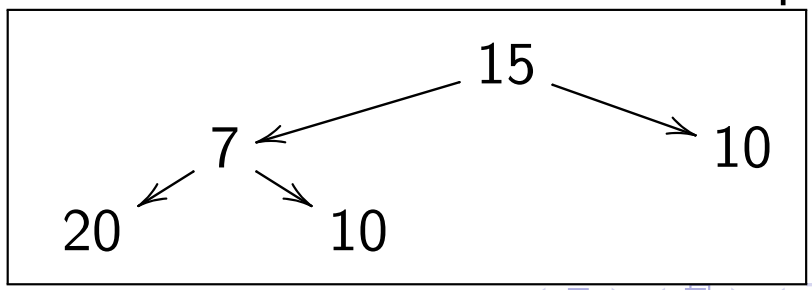
  - El árbol es (casi)completo. Se cumple: a) todos los niveles del árbol están llenos excepto el último nivel, b) en el penúltimo nivel los nodos internos están a la izquierda de las hojas (si las hubiera), y c) como máximo hay un único nodo con un solo hijo que debe ser su hijo izquierdo.
  - El árbol cumple la propiedad de montículo: la clave de la entrada almacenada en un nodo es mayor o igual que la clave de la entrada almacenada en su nodo padre (si tiene padre).

Borrar la entrada con clave mínima

- La entrada con clave mínima está en la raíz del montículo.
- Para borrar dicha entrada se mueve la entrada de la última hoja (la más a la derecha del último nivel) a la raíz, y se borra ésta.
- Después hay que restablecer la propiedad de ordenación: esa es la tarea de `heapDown` que se pide.
- Un ejemplo (**sólo mostramos las claves**): Si borra el valor mínimo 5 de éste árbol:



Se mueve la entrada de
la última hoja a la raíz. El árbol no es un montículo pues $15 > 7$:

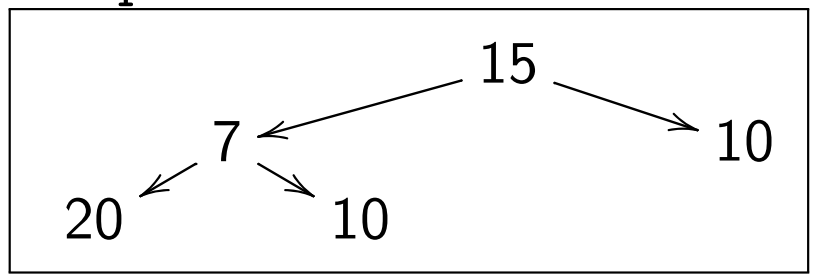


`void downHeap()`

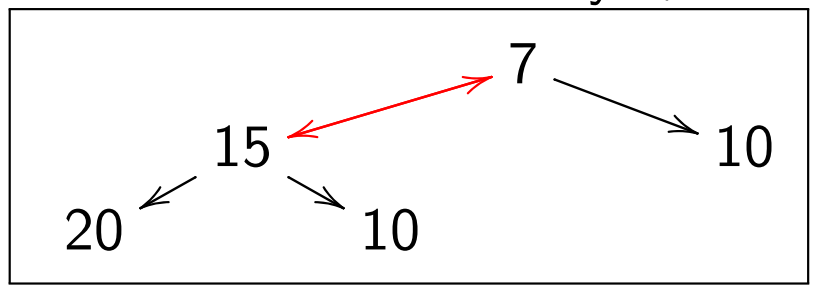
- El método, que *tenéis que completar*, es responsable de convertir un árbol de nuevo en montículo.
- Las entradas en los nodos son objetos de una clase que implementa el interfaz `Entry<K,V>`. Sólo necesitáis el interfaz.
- Para comparar *las claves* debéis usar el comparador en el atributo `comp`. El método `getKey()` devuelve la clave de un objeto de tipo `Entry<K,V>`.
- Si la clave del la entrada de la raíz es mayor de la clave de alguna entrada de sus hijos, el método debería intercambiar la entrada de la raíz con la entrada del hijo de menor clave.
- Después de cada intercambio, `downHeap` debe continuar con el hijo cambiado, y comparar su clave con las claves de sus hijos. Este procedimiento debe continuar hasta que se llega a una hoja del árbol, o no hace faltar intercambiar entradas.

`void downHeap()`: ejemplo

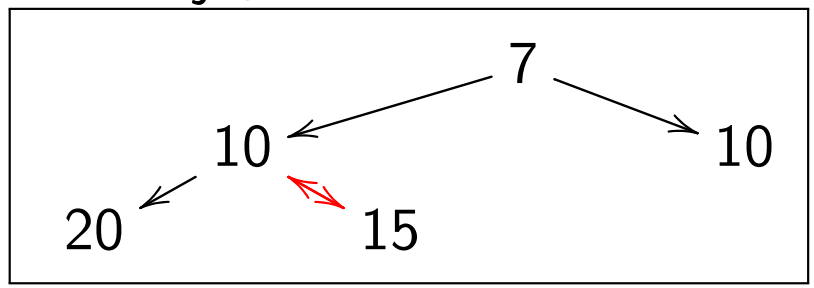
- Si llamamos a `downHeap` con el árbol:



- `downHeap` debería primero intercambiar 15 y 7, con el resultado:

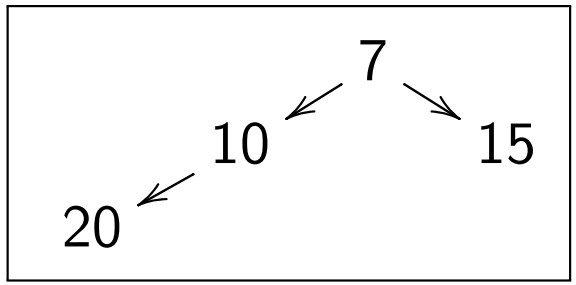


- Después `downHeap` intercambia 15 y 10, y como hemos llegado a una hoja, el árbol es un montículo otra vez:



Representación de un montículo

- Para almacenar los nodos del montículo se usa un array `Entry<K,V> a[]` que es un atributo de la clase `Heap`.
- La raíz del montículo siempre se almacena en `a[0]`.
- Un nodo con índice i tiene a sus hijos en los índices $i * 2 + 1$ (izquierdo) y $i * 2 + 2$ (derecho).
- Un nodo con índice i tiene a su padre en la posición $(i - 1)/2$.
- Como ejemplo, los elementos del montículo:



están guardados en el array:

7	10	15	20

El índice del hijo izquierdo del nodo 10 (con índice 1) se calcula como $1 * 2 + 1 = 3$. El índice del padre del nodo `a[2]` se calcula como $(2 - 1)/2 = 0$, es decir esta en `a[0]`.

`static <E> void sort(E[] arr)`

- El método ordena el array `arr` en orden *ascendente*
- Para ordenar el array es obligatorio usar la clase `Heap`.
- Para comparar dos elementos en el método `sort` se puede usar el comparador `comp` que el `sort` recibe como parámetro.
- Algoritmo:
  - inserta todos los elemento de `arr` en un nuevo montículo `m`
  - usa el montículo `m` para insertar los elementos *en orden ascendente* en el array `arr` otra vez
- Ejemplo:

  Si llamamos a `Sort.sort(arr)` con el array `arr`:

15	7	20	10

  después la llamada `arr` debe estar ordenado:

7	10	15	20

Complejidad – Ejercicios opcionales

Estos ejercicios no dan puntos extra, y no son entregables, pero son útiles para aprender mas (y para aprobar exámenes `:-`).
Muestra las soluciones a los profesores durante tutorías.

- Considera la implementación del método `sort` dentro la clase `Sort`, ¿cual es la complejidad del método `sort`?
- Considera la implementación de otro algoritmo para ordenar en la clase `BubbleSort`, ¿cual es su complejidad?
- ¿Cual es la altura máxima de un heap?
- ¿Cual es la complejidad de buscar un elemento en un heap?

Complejidad – Ejercicios opcionales practicas

- Mide las implementaciones de ordenación en la practica:
- Usad el siguiente codigo para medir el tiempo gastado en una llamada (en nano segundos):

```
long startTime = System.nanoTime();
Sort.sort(arr,comp);   // o BubbleSort.sort(arr,comp);
long elapsedTime = System.nanoTime() - startTime;
```

- Cambiad el tamaño del array, y también cambiad los contenidos del array (usad la clase `Random` para generar enteros aleatoriamente).
- ¿Qué diferencias de comportamiento observas en las implementaciones?
- ¿Coinciden los resultados prácticos con las medidas teóricas?