Estructuras de Datos y Algoritmos 1

Tema 2

- Colecciones:

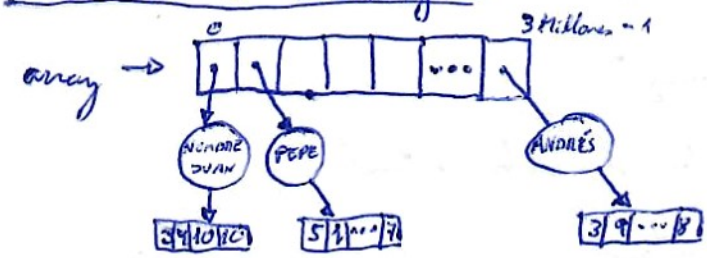
¿Cómo realizamos la búsqueda?

O(n) >>>>>>>>>> O(log n) >>>>>>>>>> O(1)

- Forma 1: Bruta (Iteramos con Foreach sobre la solución).
- Forma 2: int pos ← IndexOf(T aux)
  Boolean ← contains(T aux)

  } Preferible contains por tiempo de ejecución

- Semántica de IndexOf()




```
ArrayList<Persona> ...
public class Persona {
  private String nombre; // CLAVE
  private ArrayList<Integer> notas;
  ...
}
```

¿Cómo buscamos para cambiar la nota 3 a 8 de Andrés?

```
int pos = array.indexOf("Andrés");
```
→ NO

```
Persona aux = new Persona("Andrés");
int pos = array.indexOf(aux);
```
→ SI

Dato importante, Java no sabe comparar objetos dados por nosotros

Para comparar utilizamos @Override //Sobreescribe la clase Padre y "Añado mis cosas".

```
boolean equals (Objects otro) {
  // Claves
}
```

Ejemplo:

```
public boolean equals (Object otro) {
  return this.nombre.equals ((Cast) otro.nombre);
}
```

T = K + V → Persona = Nombre + Notas

¿Cómo ordenamos un array?

```
array.sort (); // Respeta el orden natural
```

Ejemplo de todo lo anterior:

```
public class Persona implements Comparable<Persona> {
```

INTERFAZ

```
  private String nombre;
  private ArrayList<Integer> notas;
  ...

  @Override
  public int compareTo (Persona otro) {
    return this.compareTo() → NO
    return this.nombre.compareTo (otro.nombre); → SI
  }
```

Cola de Prioridad (Priority Queue <T>) T = K + V

Implementación:

- Opción 1: Estructura de Datos Lineal ArrayList<T>
  Inserción O(1). Extraer costoso O(n) + Comparación

  Descartemos esta opción

- Opción 2: Heap de máximos / Heap de mínimos } O(log n)

Heap es un Árbol Binario Completo. ABC es ⇨ 

¿Cómo implementa un Heap?

Por referencias y por (ArrayList) → Tratamiento elaborado

Lista enlazada lista → [8] → [9] → [10] → ⏚

```
Clase Node {
  T dato;
  Clase Node siguiente;
}

Clase Lista {
  Clase Node lista;
}
```

¿Boolean add(T dato)?

Si, devuelve true si NO está el elemento;

Una interfaz es una colección de métodos

Reposerros:

indexOf et etmaylost me devuelve;

int pos = array.indexOf(T aux) → (Solo K Key)

array.get(pos)^{+-1} ← consecuent

Clase Persona (InsertaNota (double Nota))

ArrayList<Persona> datos;

Método (Persona nota). Si ∄ Persona → new Persona + nota

Si ∃ Persona → add (nota)

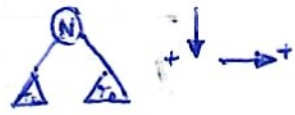
```
public boolean add (Persona persona, double nota) {
  int pos = this.datos.indexOf(Persona);
  if (pos != -1) {
   this.datos.get(pos).insertaNota(nota);
  } else {
   persona.insertaNota(nota);
   this.datos.add(persona);
  }
  return pos == -1;
}
```

Compactado:

```
public boolean add (Persona persona, double nota) {
  int pos = this.datos.indexOf(persona);
  if (pos == -1) this.datos.add(persona);
  this.datos.get(pos == -1 ? this.datos.size() -1 : pos).insertaNota(nota);
  return pos == -1;
}
```

-Heaps:

Es un ABC Árbol Binario Completo + Órden



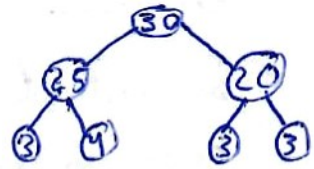
$T \begin{cases} \emptyset \\ N < T_{IZQ} \; T_{DECH} \end{cases}$ T_{IZQ} y T_{DECH} son AB

Criterio de órden: Dado un nodo con subárboles N_i $\left.\begin{matrix} N < N_2 \\ N < N_0 \end{matrix}\right\}$ Heap de mínimos

También se exige $\left.\begin{matrix} N < N_i \\ N < N_0 \end{matrix}\right\}$ Heap de máximos

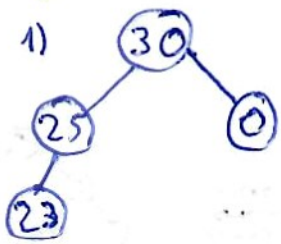
¿Es posible que en un Heap los nodos se repitan? Sí, se pueden repetir mientras cumplan los criterios de órden, de hijo e padre

Representación
- Nodo raíz posición $\emptyset$
- Nodo $\rightarrow$ IZQ $2i+1$, DECH $2i+2$
- Nodo i: PADRE $\rightarrow (i-1)/2$



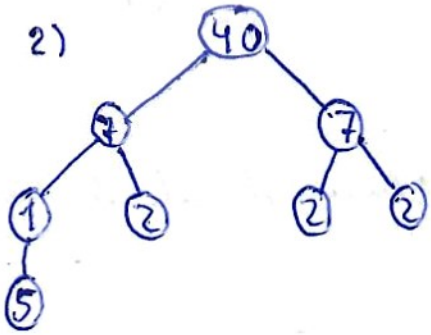
Ejemplos:

1)



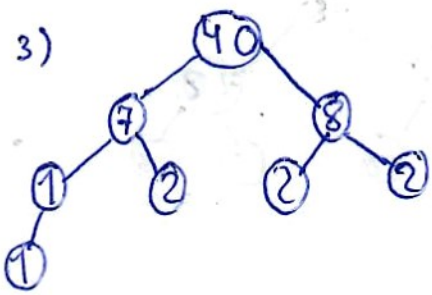
0	1	2	3
30	25	0	23

2)



NO es un HEAP, NO cumple criterio de órden, pero sí es AB

3)



SÍ es un HEAP

0	1	2	3	4	5	6	7
40	7	8	1	2	2	2	1

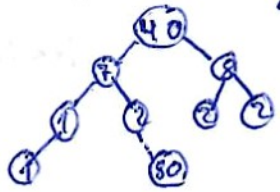
- Recorrido en anchura:

A la hora de insertar, siempre ha de tener un Heap.

Se inserta dicho en la siguiente hoja, se compara y se pone donde corresponda, subiendo hacia el padre.

El coste del algoritmo es igual a la altura del árbol $\boxed{\text{altura}(ABC) \text{ es } h}$

$O(\log n) <<<< \infty <<<< O(n)$



¿Cómo elimino?

Se intercambia y se elimina.

El número a eliminar se intercambia por el último y se elimina.

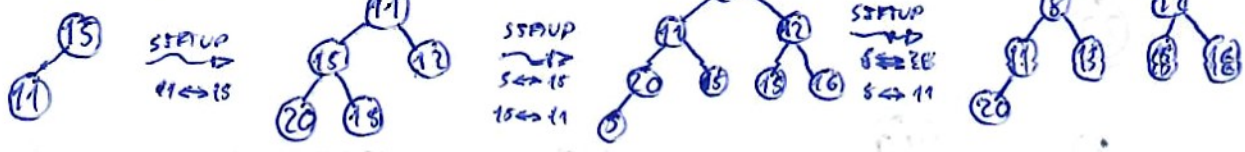
Se reestructura el árbol, quedando arriba el valor de máxima prioridad.

SIFT UP (Flotar)

- Heap de mínimos

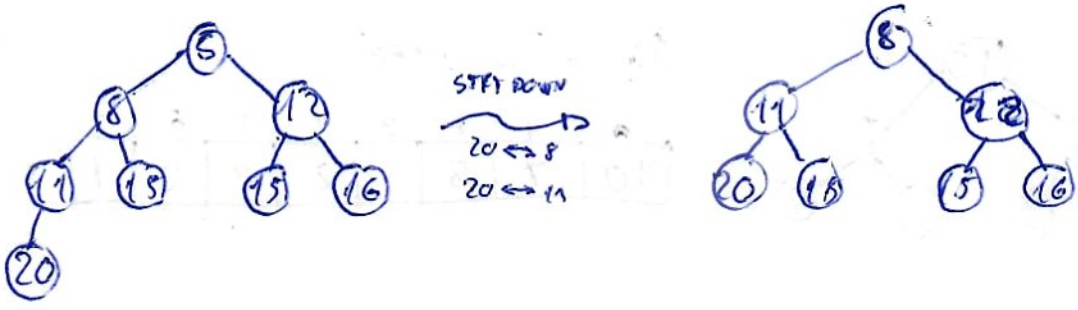
E = [15, 11, 12, 20, 5, 15, 16, 8]

Obtener Heap de mínimos:



¿Elimina el 8?

No puede elegir el valor que yo quiera. Elimina el valor de máxima prioridad (5 en nuestro caso)



Ejemplo:

```
Método {
  ArrayList<String> aux = new ArrayList<>();
  // Insertamos 10M palabras
  PriorityQueue<String> pQ = new PriorityQueue<>();
  for (String palabra : aux) {
    pQ.add(palabra);
  }                                   } O(n log n)
  aux.clear();                        +
  While (!pQ.isEmpty()) {
    aux.add(pQ.poll());               } O(n)
  }
  Bigoo(aux);
}
```

COSTE MAYOR $O(n \log n)$

¿Qué hace el método? Ordena (SortHeap)

¿Coste? { ESPACIAL / TEMPORAL

coste (sortHeap) = $O(n \log n)$

- Iteración

```
public class Diccionario implements Iterable<String> {
    private String nombre;  // Clave
    private LinkedList<String> registroPalabras;
    :
    @Override
    public Iterator<String> iterator() {
        return this.registroPalabras.iterator();
    }
}
```

Fuera de Diccionario puedo acceder a registroPalabras.

1ª Opción: Poner public en la colección. DESCARTADO
2ª Opción: get() DESCARTADO
3ª Opción: Iterable

```
Diccionario Dic01...
for (String palabra : dic01) {
    ...
}
... private ArrayList<String> aux; ...
@Override
public Iterator<String> iterator() {
    ArrayList<String> result = new ArrayList<>(this.aux);
    for (int i = 0; i < result.size(); i++) {
        result.set(i, i%2 == 0 ? result.get(i) + "0" : result.get(i) + "1");
    }
    result.iterator();
}
```

Sirve para aux, pero en posiciones pares añade 0 e impares 1

```
result.get(i) + (i%2 == 0 ? "0" : "1");
```