

RESUMEN DE COLA CON PRIORIDAD:

* **Una “Cola con prioridad” almacena una coleccion de elementos que soporta:**

1. **Insercion de elementos arbitraria**
2. **Eliminacion de elementos en orden de prioridad** [el elemento con primera prioridad puede ser eliminado en cualquier momento]

* Una “Cola con prioridad” **almacena sus elementos de acuerdo a su prioridad relativa** y no expone una nocion de “posicion” a sus clientes
* **PRIORIDAD:** Atributo de un individuo que sirve para pesar al individuo en un conjunto de individuos [**EJEMPLO:** Promedio de un alumno para otrogar becas]

**COMPARACION DE PRIORIDADES CON ORDENES TOTALES:**

* Una cola con prioridad necesita un criterio de comparación ≤ que sea un orden total para poder resolver siempre la comparación entre prioridades.
* Sea S un conjunto y ≤ una relación binaria en S, entonces (S, ≤) es una relación de orden si y solo si:

1. Reflexivo: para todo k en S, vale k ≤ k
2. Antisimétrico: para todo k1, k2 en S, vale que si k1≤k2 y k2≤k1 entonces k1=k2
3. Transitivo: para todo k1, k2,k3 en S, vale que si k1≤k2 y k2≤k3 entonces k1 ≤ k3

* Si (S, ≤) es un orden total, todos los pares de elementos de S son comparables entre sí mediante “≤”
* “≤” en los números enteros y reales y “≤” para cadenas de texto (comparación alfabética u orden lexicográfico) son órdenes totales

**ORDEN TOTAL:**

Es un orden, que dado un conjunto S y una relación “≤” entre los elementos del conjunto, la operación debe ser reflexiva, antisimétrica y transitiva

**IMPORTANTE:**

Para comparar dos elementos por prioridad, las prioridades deben satisfacer ser un ORDEN TOTAL

* **DEFINICION:** **Una cola con prioridad es un colección de elementos, llamados valores, los cuales tienen asociada una prioridad que es provista en el momento que el elemento es insertado**
* Un par prioridad-valor insertado en un cola con prioridad se llama una **ENTRADA**

**TDA COMPARADOR:**

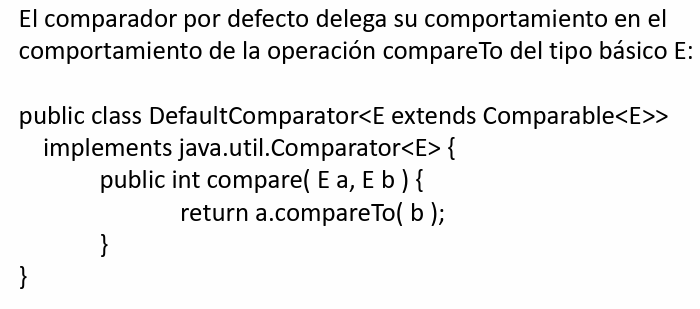
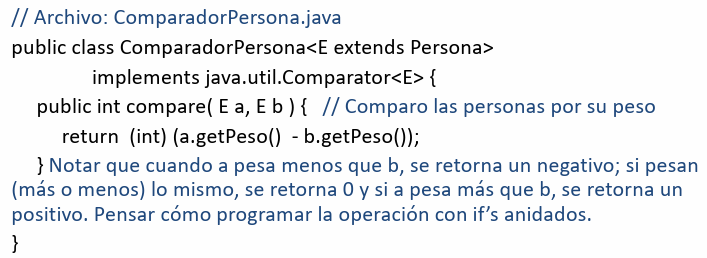
**Compare(a,b) 🡪 Retorna un entero i tal que:**

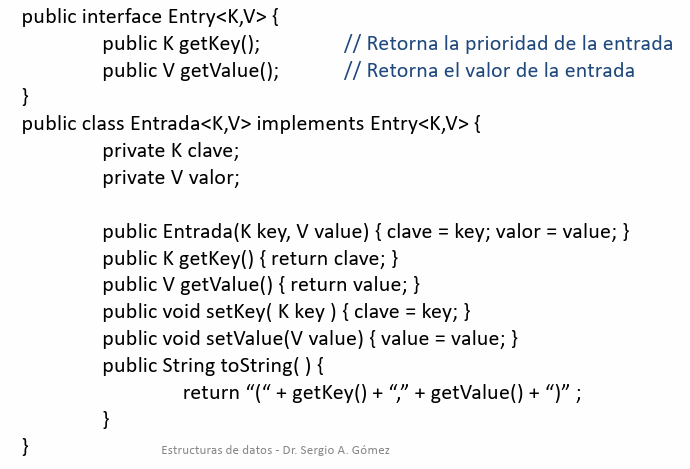
1. **i < 0 🡪 a < b**
2. **i = 0 🡪 a = b**
3. **i > 0 🡪 a > b**

**Ocurre error si a y b no pueden ser comparados, y se encuentra especificado por la interfaz “java.util.Comparator.”**

**Comparador por defecto:**

**Ejemplo de comparador para una clase especifica:**





**//Interfaz de entrada**

**Entradas de una cola con prioridad:**

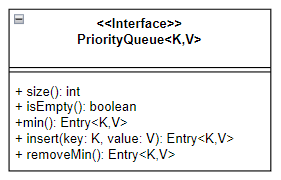
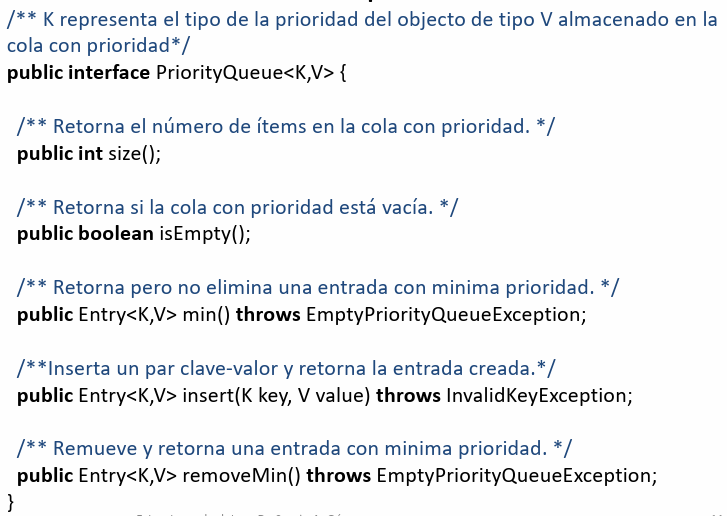
**//Clase que implementa la interfaz**

* La “Cola con Prioridad” se modela a travez de una interfaz **[TDAColaConPrioridad]**, que se encuentra modelada por las siguientes operaciones y sus restricciones [**Dada una cola con prioridad P**]:

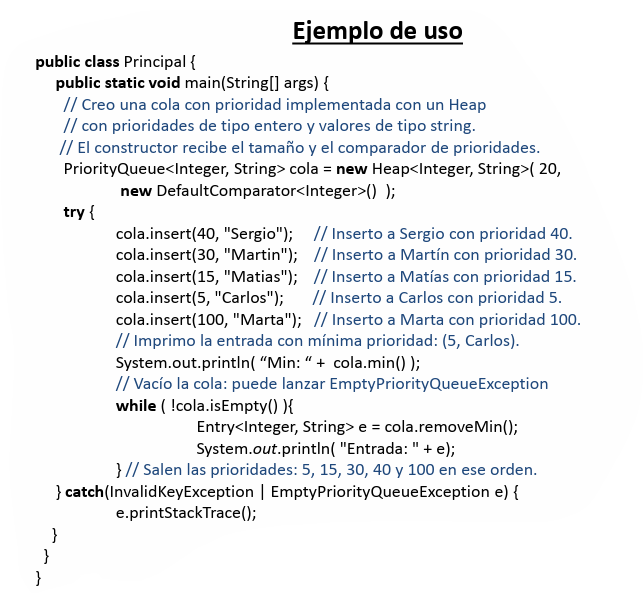
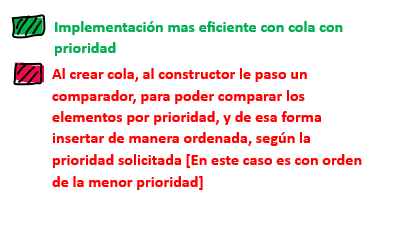
1. **Size()** 🡪 Retorna el número de entradas en P
2. **isEmpty()** 🡪 Testea si P es vacía
3. **Min()** 🡪 Retorna (pero no remueve) una entrada de P con la prioridad más pequeña; ocurre un error si P está vacía [**EmptyPriorityQueueException**] **(\*)**
4. **Insert(k,x)** 🡪 Inserta en P una entrada con prioridad k y valor x; ocurre un error si k es inválida (e.g. k es nula) [**InvalidKeyException**] **(\*)**
5. **removeMin()** 🡪 Remueve de P y retorna una entrada con la prioridad más pequeña; ocurre una condición de error si P está vacía [**EmptyPriorityQueueException**] **(\*)**

**(\*) Operaciones fundamentales de una cola con prioridad P**

**IMPLEMENTACION DE LA INTERFAZ PRIORITYQUEUE**

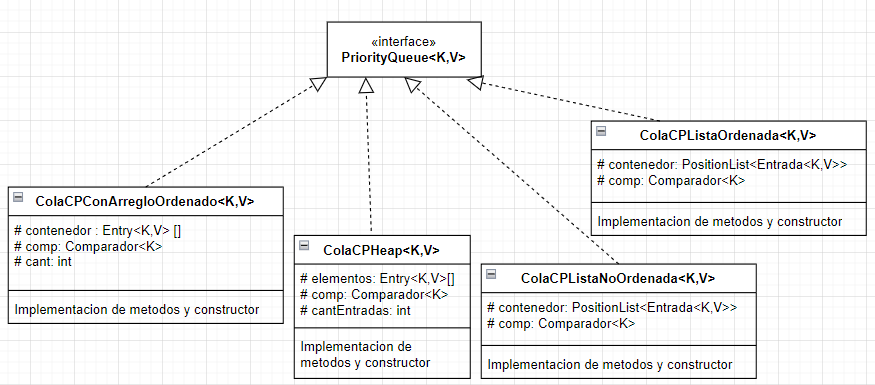


**DIAGRAMA UML DE LA INTERFAZ PRIORITYQUEUE**

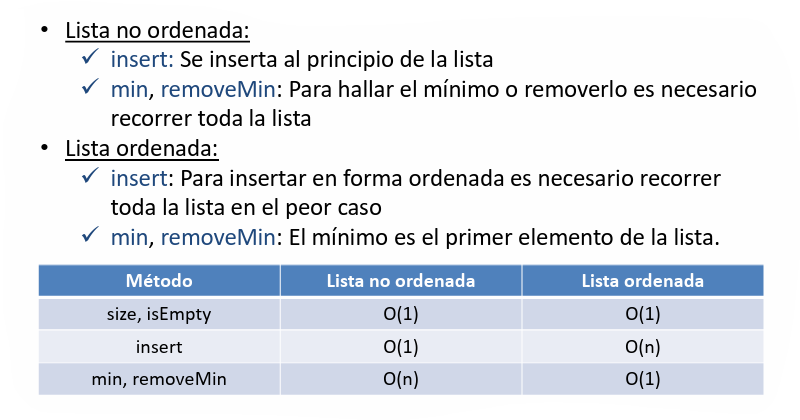




IMPLEMENTACIONES DE LA INTERFAZ PRIORITYQUEUE<K,V> [Ver implementaciones]



**IMPLEMENTACION DE COLA CON PRIORIDAD CON LISTAS [Ver implementacion]:**





**//La diferencia entre ambas implementaciones radica en la utilización de un comparador de claves de entrada para la inserción de elementos de la cola con prioridad, que puede o no ordenarlos en la ED 🡪 Esto provoca un cambio en el orden del tiempo de ejecución de algunos métodos de ambas implementaciones**

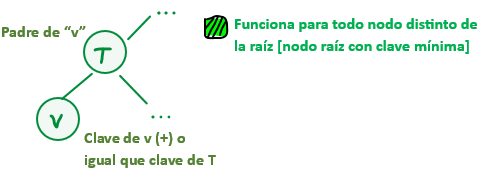
**//Utiliza estructura lineal [lista] para representar la cola con prioridad**

**//IMPLEMENTACIONES MENOS EFICIENTES**

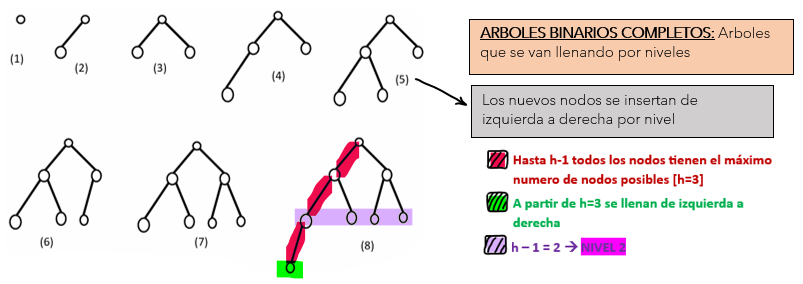
**//Ver implementacion de “Cola con Prioridad con Lista Ordenada y no Ordenada”**

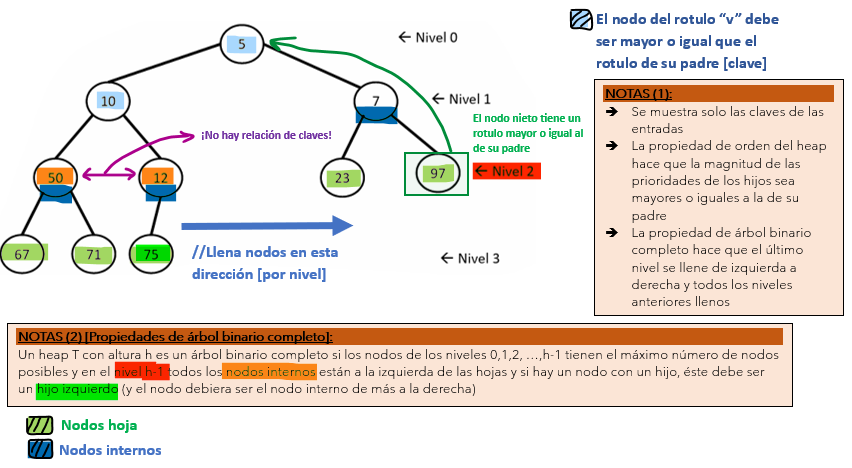
**COLA CON PRIORIDAD IMPLEMENTADA CON HEAP:**

* Un **(MIN)HEAP** es un arbol binario que almacena una coleccion de entradas en sus nodos y satisfice dos propiedades adicionales

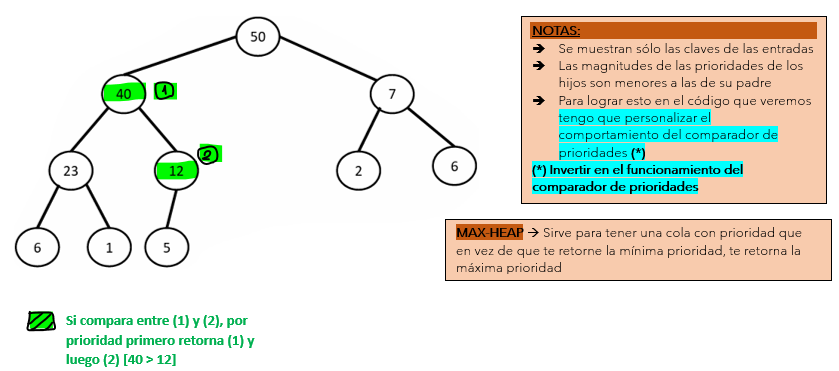
1. **PROPIEDAD DE ORDEN DEL HEAP (ARBOL PARCIALMENTE ORDENADO):** En un heap T, para cada nodo v distinto de la raíz, la clave almacenada en v es mayor o igual que la clave almacenada en el padre de v
2. **PROPIEDAD DE ARBOL BINARIO COMPLETO**: Un heap T con altura h es un árbol binario completo si los nodos de los niveles 0,1,2,…,h-1 tienen el máximo número de nodos posibles y en el nivel h-1 todos los nodos internos están a la izquierda de las hojas y si hay un nodo con un hijo, éste debe ser un hijo izquierdo (y el nodo debiera ser el nodo interno de más a la derecha)

**EJEMPLO DE ARBOLES BINARIOS COMPLETOS**





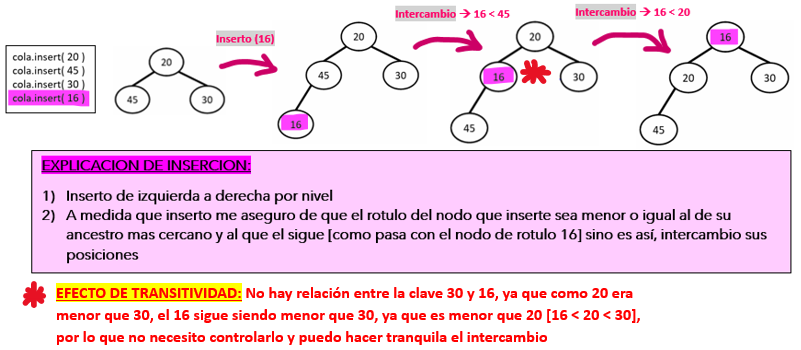
**EJEMPLO DE MIN-HEAP CON ALTURA 3:**



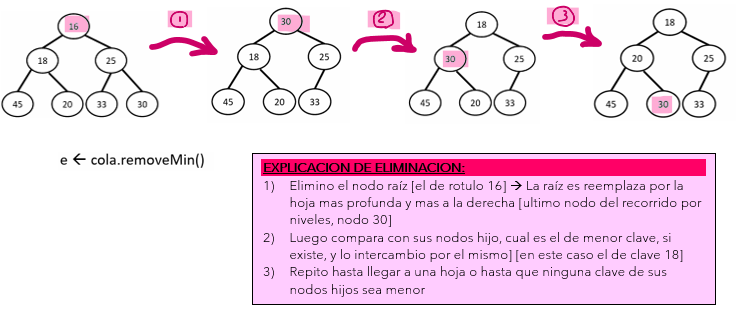
**El comparador que le paso al constructor de la cola con prioridad compara los elementos por mayor prioridad**

**EJEMPLO DE MAX-HEAP CON ALTURA 3:**

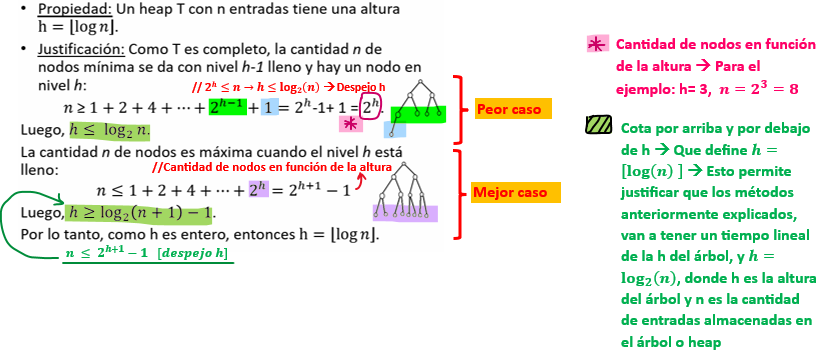
* **EXPLICACION GRAFICA DEL INSERT(K,X) [MEAN-HEAP]:**



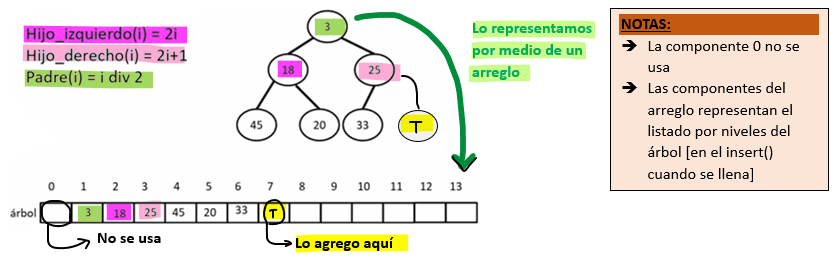
* **EXPLICACION GRAFICA DEL REMOVEMIN(K,X) [MEAN-HEAP]:**



* **ALTURA DEL HEAP:**



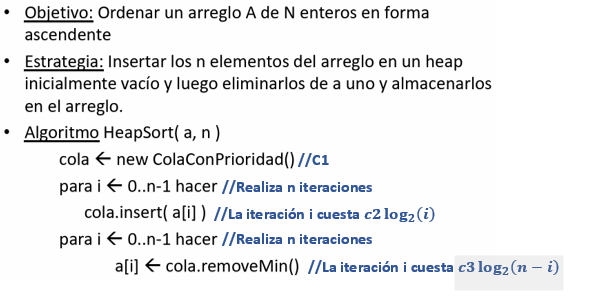
* **REPRESENTACION CON ARREGLOS DEL ARBOL BINARIO (HEAP):**



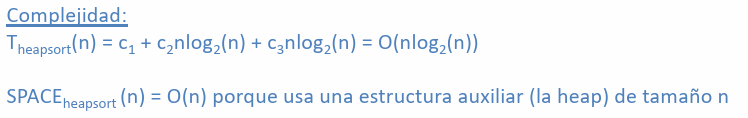
**//Ver implementacion de “Cola con Prioridad con Heap” [HEAP]**

**//Se utiliza la clase Entry<K,V> y Entrada<K,V> de Mapeos y Diccionarios**

* **APLICACION QUE UTILIZA LA COLA CON PRIORIDAD CON HEAP: “HEAP SORT”**



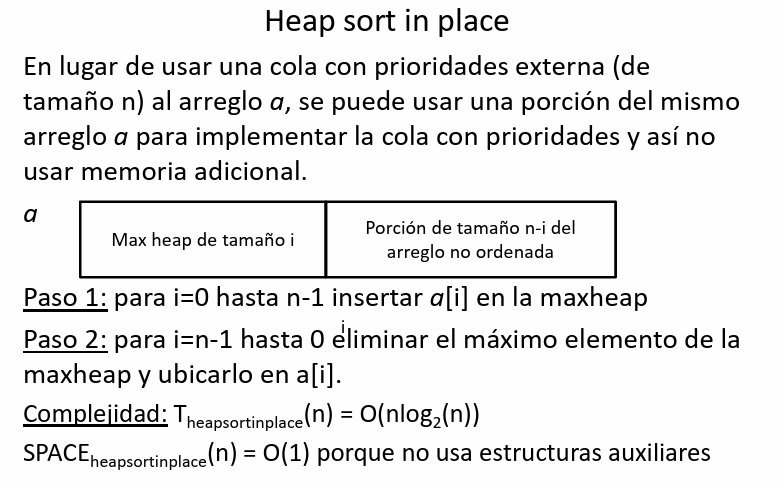


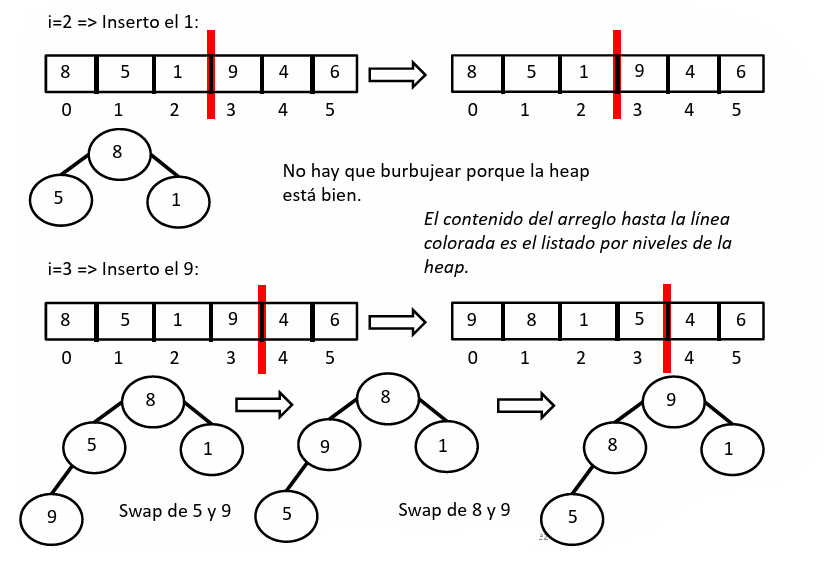




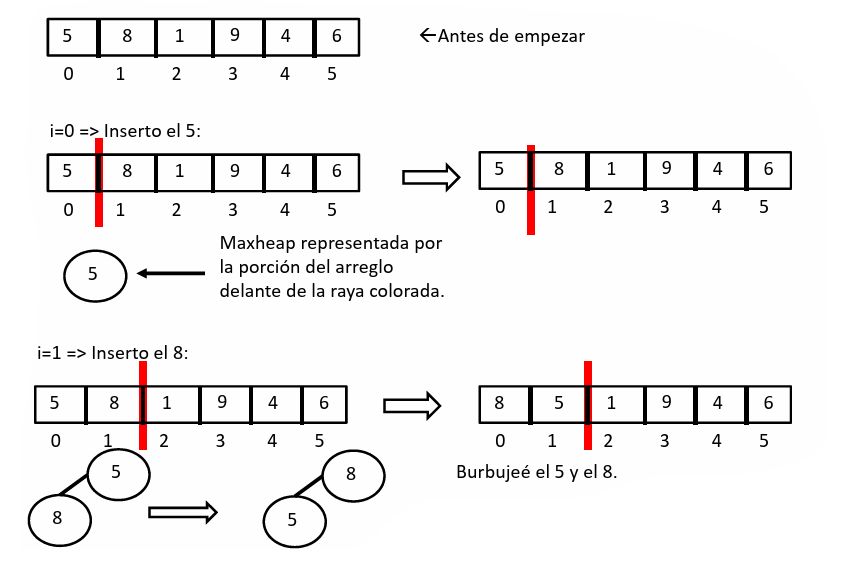


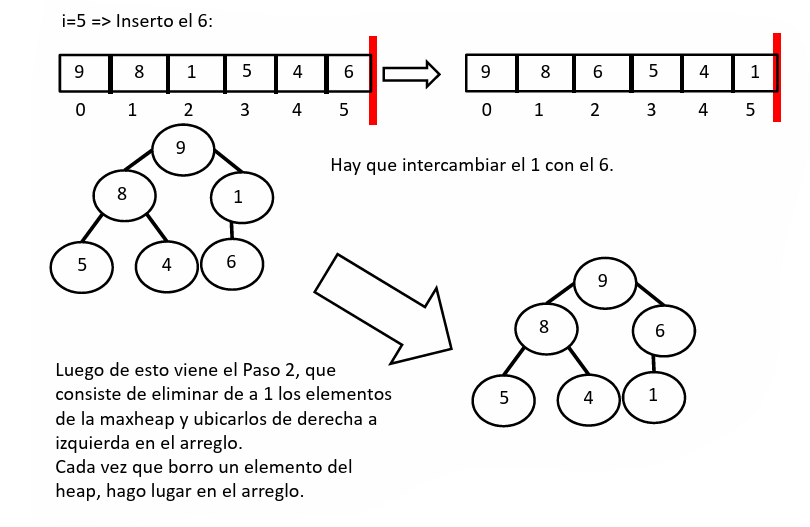
**Requiere una memoria auxiliar de tamaño n [arreglo] 🡪 Para implementar la cola con prioridades auxiliares**

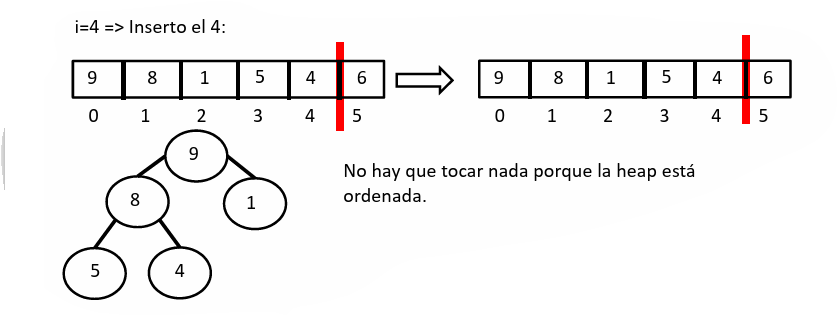


* **TRAZA DE HEAPSORT IN PLACE:**

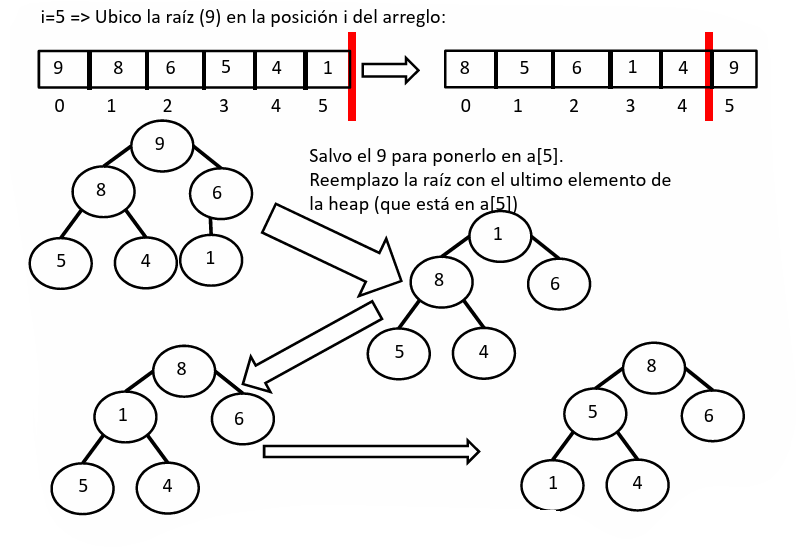
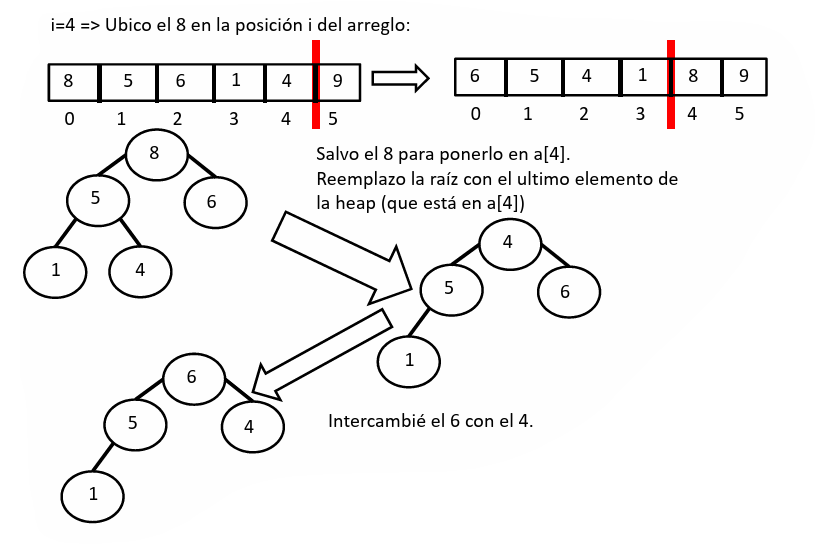
1. **Construir una maxheap con los elementos del arreglo**

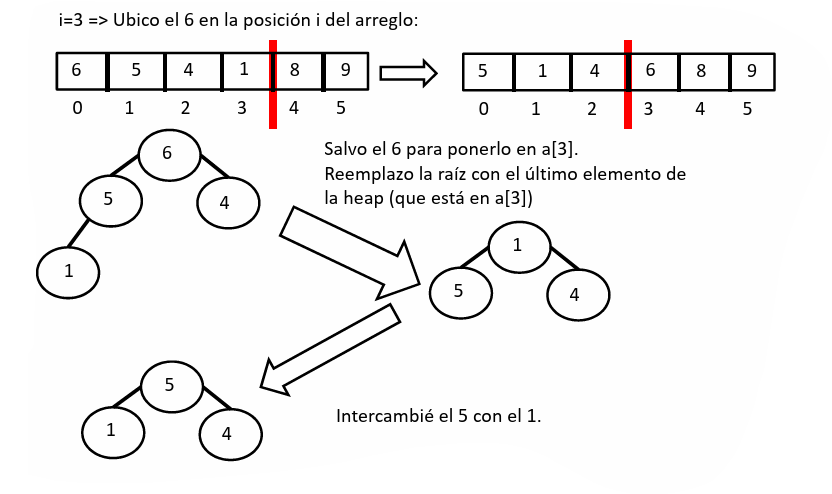
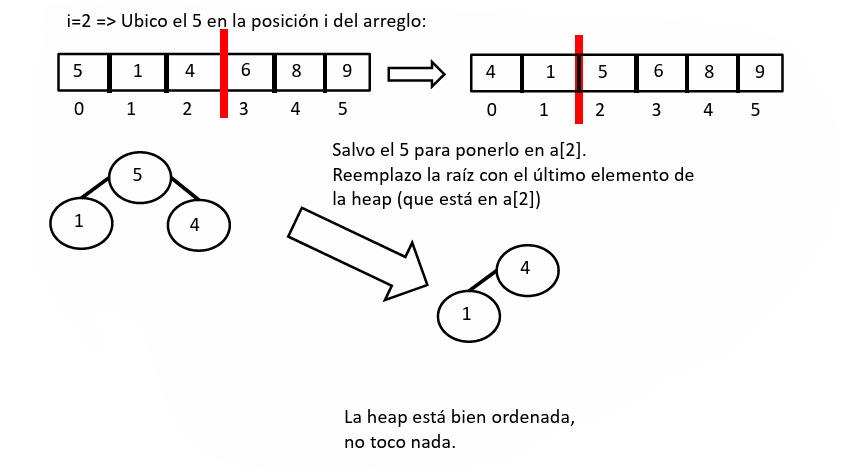


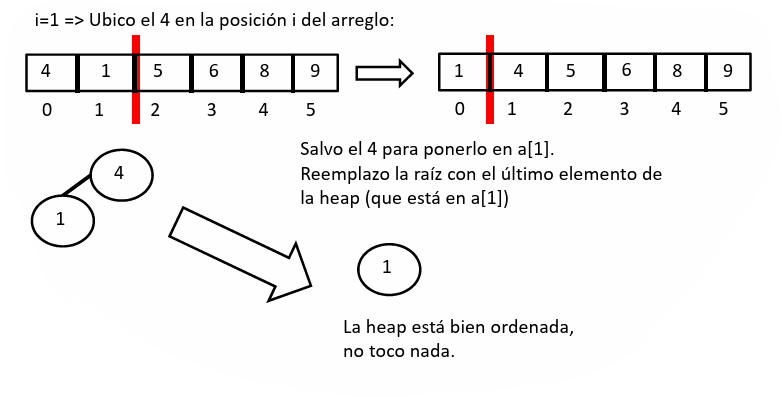
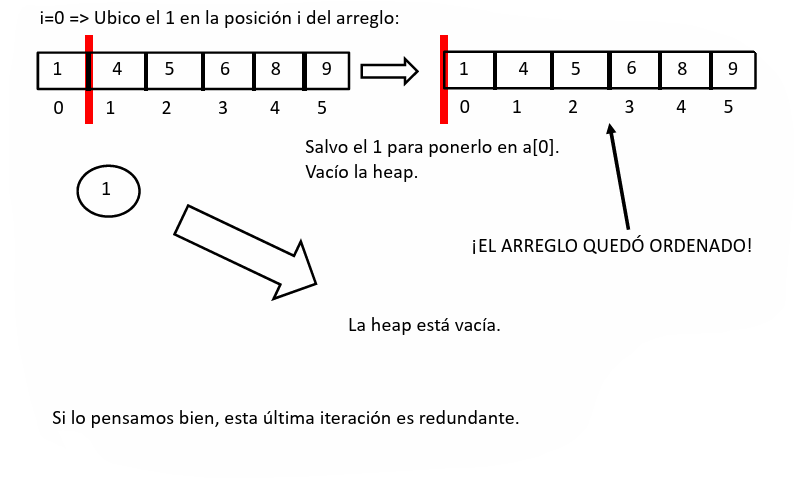




1. **Eliminar y ubicar el elemento eliminado en el arreglo**







**IMPLEMENTACION CON HEAP:** Asegura que insert() y remove() se hace en 🡪 Garantizo con:

1. Haciendo un algoritmo que es lineal con la altura de datos [altura del árbol binario] 🡪 “h proporcional al algoritmo” (1)
2. Como garantizo (1), el árbol es lleno, se llena de izquierda a derecha y los datos de la raíz son “” que los datos de sus hijos directos, y entre nodos hermanos no hay comparación

**ORDENES DE LOS TIEMPOS DE EJECUCION DE LOS METODOS DE LA IMPLEMENTACION CON HEAP:**

|  |  |
| --- | --- |
| **METODOS** | **ORDEN DEL TIEMPO DE EJECUCION** |
| **Size()** | **O(1)** |
| **isEmpty()** | **O(1)** |
| **Min()** | **O(1)** |
| **Insert(k,v)** |  |
| **removeMin()** |  |