

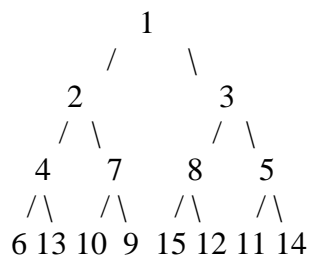
## **Ejercitación Teórica N° 2:** **Cola de Prioridades - Heap.**

### **Ejercicio 1.**

A partir de una heap inicialmente vacía, insertar, de a uno, los siguientes valores: 6, 4, 15, 2, 10, 11, 8, 1, 13, 7, 9, 12, 5, 3, 14.

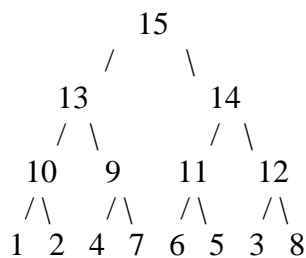
[6] - *Insert(6)*  
 [6, 4] - *Insert(4)*  
 [4, 6]  
 [4, 6, 15] - *Insert(15)*  
 [4, 6, 15, 2] - *Insert(2)*  
 [4, 2, 15, 6]  
 [2, 4, 15, 6]  
 [2, 4, 15, 6, 10] - *Insert(10)*  
 [2, 4, 15, 6, 10, 11] - *Insert(11)*  
 [2, 4, 11, 6, 10, 15]  
 [2, 4, 11, 6, 10, 15, 8] - *Insert(8)*  
 [2, 4, 8, 6, 10, 15, 11]  
 [2, 4, 8, 6, 10, 15, 11, 1] - *Insert(1)*  
 [2, 4, 8, 1, 10, 15, 11, 6]  
 [2, 1, 8, 4, 10, 15, 11, 6]  
 [1, 2, 8, 4, 10, 15, 11, 6]  
 [1, 2, 8, 4, 10, 15, 11, 6, 13] - *Insert(13)*  
 [1, 2, 8, 4, 10, 15, 11, 6, 13, 7] - *Insert(7)*  
 [1, 2, 8, 4, 7, 15, 11, 6, 13, 10]  
 [1, 2, 8, 4, 7, 15, 11, 6, 13, 10, 9] - *Insert(9)*  
 [1, 2, 8, 4, 7, 15, 11, 6, 13, 10, 9, 12] - *Insert(12)*  
 [1, 2, 8, 4, 7, 12, 11, 6, 13, 10, 9, 15]  
 [1, 2, 8, 4, 7, 12, 11, 6, 13, 10, 9, 15, 5] - *Insert(5)*  
 [1, 2, 8, 4, 7, 5, 11, 6, 13, 10, 9, 15, 12]  
 [1, 2, 5, 4, 7, 8, 11, 6, 13, 10, 9, 15, 12]  
 [1, 2, 5, 4, 7, 8, 11, 6, 13, 10, 9, 15, 12, 3] - *Insert(3)*  
 [1, 2, 5, 4, 7, 8, 3, 6, 13, 10, 9, 15, 12, 11]  
 [1, 2, 3, 4, 7, 8, 5, 6, 13, 10, 9, 15, 12, 11]  
 [1, 2, 3, 4, 7, 8, 5, 6, 13, 10, 9, 15, 12, 11, 14] - *Insert(14)*

*Min-Heap:* [1, 2, 3, 4, 7, 8, 5, 6, 13, 10, 9, 15, 12, 11, 14].



[6] - *Insert*(6)  
 [6, 4] - *Insert*(4)  
 [6, 4, 15] - *Insert*(15)  
 [15, 4, 6]  
 [15, 4, 6, 2] - *Insert*(2)  
 [15, 4, 6, 2, 10] - *Insert*(10)  
 [15, 10, 6, 2, 4]  
 [15, 10, 6, 2, 4, 11] - *Insert*(11)  
 [15, 10, 11, 2, 4, 6]  
 [15, 10, 11, 2, 4, 6, 8] - *Insert*(8)  
 [15, 10, 11, 2, 4, 6, 8, 1] - *Insert*(1)  
 [15, 10, 11, 2, 4, 6, 8, 1, 13] - *Insert*(13)  
 [15, 10, 11, 13, 4, 6, 8, 1, 2]  
 [15, 13, 11, 10, 4, 6, 8, 1, 2]  
 [15, 13, 11, 10, 4, 6, 8, 1, 2, 7] - *Insert*(7)  
 [15, 13, 11, 10, 7, 6, 8, 1, 2, 4]  
 [15, 13, 11, 10, 7, 6, 8, 1, 2, 4, 9] - *Insert*(9)  
 [15, 13, 11, 10, 9, 6, 8, 1, 2, 4, 7]  
 [15, 13, 11, 10, 9, 6, 8, 1, 2, 4, 7, 12] - *Insert*(12)  
 [15, 13, 11, 10, 9, 12, 8, 1, 2, 4, 7, 6]  
 [15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6]  
 [15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6, 5] - *Insert*(5)  
 [15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6, 5, 3] - *Insert*(3)  
 [15, 13, 12, 10, 9, 11, 8, 1, 2, 4, 7, 6, 5, 3, 14] - *Insert*(14)  
 [15, 13, 12, 10, 9, 11, 14, 1, 2, 4, 7, 6, 5, 3, 8]  
 [15, 13, 14, 10, 9, 11, 12, 1, 2, 4, 7, 6, 5, 3, 8].

*Max-Heap*: [15, 13, 14, 10, 9, 11, 12, 1, 2, 4, 7, 6, 5, 3, 8].



6, 4, 15, 2, 10, 11, 8, 1, 13, 7, 9, 12, 5, 3, 14

La raíz está almacenada en la posición 1. Para un elemento que está en la posición  $i$ , se tiene:

- El hijo izquierdo está en la posición  $2i$ .
- El hijo derecho está en la posición  $2i + 1$ .
- El padre está en la posición  $\left\lfloor \frac{i}{2} \right\rfloor$ .

## **Ejercicio 2.**

(a) *¿Cuántos elementos hay, al menos, en una heap de altura  $h$ ?*

En una *heap* de altura  $h$ , hay, al menos,  $2^h$  elementos.

(b) *¿Dónde se encuentra ubicado el elemento mínimo en una max-heap?*

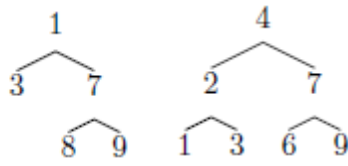
El elemento mínimo en una *max-heap* se encuentra ubicado en alguna de las hojas.

(c) *¿El siguiente arreglo es una max-heap: [23, 17, 14, 6, 13, 10, 1, 5, 7, 12]?*

No, este arreglo no es una *max-heap*, ya que el nodo con valor 6 es padre del nodo con valor 7, no cumpliendo con que cada nodo sea mayor o igual que sus hijos.

**Ejercicio 3.**

Dados los siguientes árboles, indicar si representan una *heap*. Justificar la respuesta.



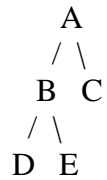
El primer árbol no es una *heap*, ya que no es un árbol binario completo.

El segundo árbol no es una *heap*, ya que, si bien es completo, cada nodo no es menor o igual (o mayor o igual) que sus hijos.

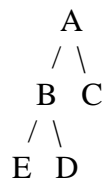
**Ejercicio 4.**

*Dibujar todas las min-heaps posibles para este conjunto de claves: {A, B, C, D, E}.*

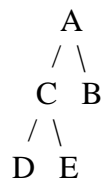
Opción 1:



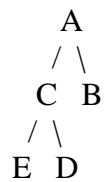
Opción 2:



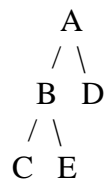
Opción 3:



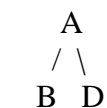
Opción 4:



Opción 5:



Opción 6:



/ \  
E C

Opción 7:

A  
 / \  
B E  
 / \  
C D

Opción 8:

A  
 / \  
B E  
 / \  
D C

**Ejercicio 5.**

A partir de una min-heap inicialmente vacía, dibujar la evolución del estado de la heap al ejecutar las siguientes operaciones:

*Insert(5), Insert(4), Insert(7), Insert(1), DeleteMin(), Insert(3), Insert(6), DeleteMin(), DeleteMin(), Insert(8), DeleteMin(), Insert(2), DeleteMin(), DeleteMin().*

Insert(5):

5

Insert(4):

5  
/  
4

4  
/  
5

Insert(7):

4  
/ \  
5 7

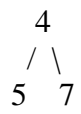
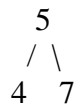
Insert(1):

4  
/ \  
5 7  
/  
1

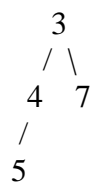
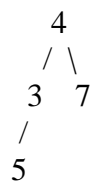
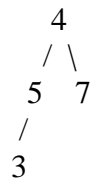
4  
/ \  
1 7  
/  
5

1  
/ \  
4 7  
/  
5

DeleteMin():



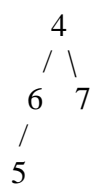
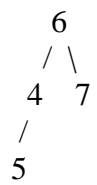
Insert(3):



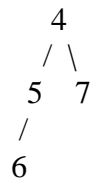
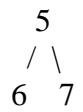
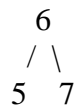
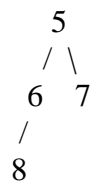
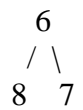
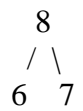
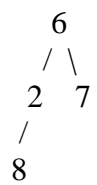
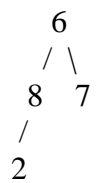
Insert(6):



DeleteMin():





DeleteMin():Insert(8):DeleteMin():Insert(2):

2

```

    / \
   6  7
  /
 8

```

DeleteMin():

```

    8
   / \
  6  7

```

```

    6
   / \
  8  7

```

DeleteMin():

```

    7
   /
  8

```

**Ejercicio 6.**

Aplicar el algoritmo *BuildHeap* para construir una *min-heap* en tiempo lineal, con los siguientes valores: {150, 80, 40, 10, 70, 110, 30, 120, 140, 60, 50, 130, 100, 20, 90}.

Se parte de:

[150, 80, 40, 10, 70, 110, 30, 120, 140, 60, 50, 130, 100, 20, 90].

Se empieza en la posición  $i = \left\lfloor \frac{n}{2} \right\rfloor = 7$  del arreglo, el 30, y se filtra:

[150, 80, 40, 10, 70, 110, 20, 120, 140, 60, 50, 130, 100, 30, 90].

Se avanza a la posición anterior del arreglo, el 110, y se filtra:

[150, 80, 40, 10, 70, 100, 20, 120, 140, 60, 50, 130, 110, 30, 90].

Se avanza a la posición anterior del arreglo, el 70, y se filtra:

[150, 80, 40, 10, 50, 100, 20, 120, 140, 60, 70, 130, 110, 30, 90].

Se avanza a la posición anterior del arreglo, el 10, y se filtra:

[150, 80, 40, 10, 50, 100, 20, 120, 140, 60, 70, 130, 110, 30, 90].

Se avanza a la posición anterior del arreglo, el 40, y se filtra:

[150, 80, 20, 10, 50, 100, 40, 120, 140, 60, 70, 130, 110, 30, 90]

[150, 80, 20, 10, 50, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90].

Se avanza a la posición anterior del arreglo, el 80, y se filtra:

[150, 10, 20, 80, 50, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90].

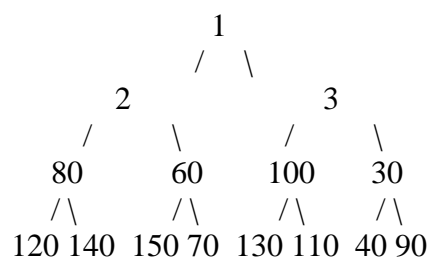
Se avanza a la posición anterior del arreglo, el 150, y se filtra:

[10, 150, 20, 80, 50, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90]

[10, 50, 20, 80, 150, 100, 30, 120, 140, 60, 70, 130, 110, 40, 90]

[10, 50, 20, 80, 60, 100, 30, 120, 140, 150, 70, 130, 110, 40, 90].

*Min-Heap*: [10, 50, 20, 80, 60, 100, 30, 120, 140, 150, 70, 130, 110, 40, 90].



**Ejercicio 7.**

*Aplicar el algoritmo HeapSort para ordenar, descendentemente, los siguientes elementos: {15, 18, 40, 1, 7, 10, 33, 2, 140, 500, 11, 12, 13, 90}. Mostrar, paso a paso, la ejecución del algoritmo sobre los datos.*

Se parte de:

*Min-Heap:* [1, 2, 10, 7, 11, 12, 33, 18, 140, 500, 15, 40, 13, 90].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 90:

[90, 2, 10, 7, 11, 12, 33, 18, 140, 500, 15, 40, 13 | 1]  
[2, 90, 10, 7, 11, 12, 33, 18, 140, 500, 15, 40, 13 | 1]  
[2, 7, 10, 90, 11, 12, 33, 18, 140, 500, 15, 40, 13 | 1]  
[2, 7, 10, 18, 11, 12, 33, 90, 140, 500, 15, 40, 13 | 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 13:

[13, 7, 10, 18, 11, 12, 33, 90, 140, 500, 15, 40 | 2, 1]  
[7, 13, 10, 18, 11, 12, 33, 90, 140, 500, 15, 40 | 2, 1]  
[7, 11, 10, 18, 13, 12, 33, 90, 140, 500, 15, 40 | 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 40:

[40, 11, 10, 18, 13, 12, 33, 90, 140, 500, 15 | 7, 2, 1]  
[10, 11, 40, 18, 13, 12, 33, 90, 140, 500, 15 | 7, 2, 1]  
[10, 11, 12, 18, 13, 40, 33, 90, 140, 500, 15 | 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 15:

[15, 11, 12, 18, 13, 40, 33, 90, 140, 500 | 10, 7, 2, 1]  
[11, 15, 12, 18, 13, 40, 33, 90, 140, 500 | 10, 7, 2, 1]  
[11, 13, 12, 18, 15, 40, 33, 90, 140, 500 | 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 500:

[500, 13, 12, 18, 15, 40, 33, 90, 140 | 11, 10, 7, 2, 1]  
[12, 13, 500, 18, 15, 40, 33, 90, 140 | 11, 10, 7, 2, 1]  
[12, 13, 33, 18, 15, 40, 500, 90, 140 | 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 140:

[140, 13, 33, 18, 15, 40, 500, 90 | 12, 11, 10, 7, 2, 1]  
[13, 140, 33, 18, 15, 40, 500, 90 | 12, 11, 10, 7, 2, 1]  
[13, 15, 33, 18, 140, 40, 500, 90 | 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 90:

[90, 15, 33, 18, 140, 40, 500 | 13, 12, 11, 10, 7, 2, 1]  
[15, 90, 33, 18, 140, 40, 500 | 13, 12, 11, 10, 7, 2, 1]  
[15, 18, 33, 90, 140, 40, 500 | 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 500:

[500, 18, 33, 90, 140, 40 | 15, 13, 12, 11, 10, 7, 2, 1]  
[18, 500, 33, 90, 140, 40 | 15, 13, 12, 11, 10, 7, 2, 1]  
[18, 90, 33, 500, 140, 40 | 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 40:

[40, 90, 33, 500, 140 | 18, 15, 13, 12, 11, 10, 7, 2, 1]  
[33, 90, 40, 500, 140 | 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 140:

[140, 90, 40, 500 | 33, 18, 15, 13, 12, 11, 10, 7, 2, 1]  
[40, 90, 140, 500 | 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 500:

[500, 90, 140 | 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1]  
[90, 500, 140 | 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 140:

[140, 500 | 90, 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Intercambiar el primero con el último, decrementar el tamaño:

[500 | 140, 90, 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

Arreglo ordenado descendentemente:

[500, 140, 90, 40, 33, 18, 15, 13, 12, 11, 10, 7, 2, 1].

**Ejercicio 8.**

Construir una max-heap con los siguientes datos: {5, 8, 12, 9, 7, 10, 21, 6, 14, 4}.

(a) Insertándolos de a uno.

[5] - *Insert(5)*  
 [5, 8] - *Insert(8)*  
 [8, 5]  
 [8, 5, 12] - *Insert(12)*  
 [12, 5, 8]  
 [12, 5, 8, 9] - *Insert(9)*  
 [12, 9, 8, 5]  
 [12, 9, 8, 5, 7] - *Insert(7)*  
 [12, 9, 8, 5, 7, 10] - *Insert(10)*  
 [12, 9, 10, 5, 7, 8]  
 [12, 9, 10, 5, 7, 8, 21] - *Insert(21)*  
 [12, 9, 21, 5, 7, 8, 10]  
 [21, 9, 12, 5, 7, 8, 10]  
 [21, 9, 12, 5, 7, 8, 10, 6] - *Insert(6)*  
 [21, 9, 12, 6, 7, 8, 10, 5]  
 [21, 9, 12, 6, 7, 8, 10, 5, 14] - *Insert(14)*  
 [21, 9, 12, 14, 7, 8, 10, 5, 6]  
 [21, 14, 12, 9, 7, 8, 10, 5, 6]  
 [21, 14, 12, 9, 7, 8, 10, 5, 6, 4] - *Insert(4)*

Max-Heap: [21, 14, 12, 9, 7, 8, 10, 5, 6, 4].

(b) Usando el algoritmo BuildHeap.

Se parte de:

[5, 8, 12, 9, 7, 10, 21, 6, 14, 4].

Se empieza en la posición  $i = \left\lfloor \frac{n}{2} \right\rfloor = 5$  del arreglo, el 7, y se filtra:

[5, 8, 12, 9, 7, 10, 21, 6, 14, 4].

Se avanza a la posición anterior del arreglo, el 9, y se filtra:

[5, 8, 12, 14, 7, 10, 21, 6, 9, 4].

Se avanza a la posición anterior del arreglo, el 12, y se filtra:

[5, 8, 21, 14, 7, 10, 12, 6, 9, 4].

Se avanza a la posición anterior del arreglo, el 8, y se filtra:

[5, 14, 21, 8, 7, 10, 12, 6, 9, 4]  
[5, 14, 21, 9, 7, 10, 12, 6, 8, 4].

Se avanza a la posición anterior del arreglo, el 5, y se filtra:

[21, 14, 5, 9, 7, 10, 12, 6, 8, 4]  
[21, 14, 12, 9, 7, 10, 5, 6, 8, 4].

*Max-Heap*: [21, 14, 12, 9, 7, 10, 5, 6, 8, 4].

**Ejercicio 9.**

Suponer que una heap que representa una cola de prioridades está almacenada en el arreglo  $A$  (se comienza de la posición  $A[1]$ ). Si se inserta la clave 16, ¿en qué posición quedará?

$i$	1	2	3	4	5	6	7	8	9	10	11	12
$A[i]$	11	21	27	37	36	34	32	43	44	42	51	62

- (a)  $A[2]$ .      (b)  $A[3]$ .      (c)  $A[6]$ .      (d)  $A[7]$ .      (e)  $A[12]$ .

[11, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51, 62]

[11, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51, 62, 16] - *Insert(16)*

[11, 21, 27, 37, 36, 16, 32, 43, 44, 42, 51, 62, 34]

[11, 21, 16, 37, 36, 27, 32, 43, 44, 42, 51, 62, 34].



**Ejercicio 10.**

Suponer que una heap que representa una cola de prioridades está almacenada en el arreglo  $A$  (se comienza de la posición  $A[1]$ ). Si se aplica un delete-min, ¿en qué posición quedará la clave 62?

$i$	1	2	3	4	5	6	7	8	9	10	11	12
$A[i]$	11	21	27	37	36	34	32	43	44	42	51	62

(a)  $A[1]$ .      (b)  $A[2]$ .      (c)  $A[10]$ .      (d)  $A[11]$ .      (e)  $A[12]$ .

[11, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51, 62]  
[62, 21, 27, 37, 36, 34, 32, 43, 44, 42, 51] - DeleteMin()  
[21, 62, 27, 37, 36, 34, 32, 43, 44, 42, 51]  
[21, 36, 27, 37, 62, 34, 32, 43, 44, 42, 51]  
[21, 36, 27, 37, 42, 34, 32, 43, 44, 62, 51].

**Ejercicio 11.**

**(a)** Ordenar, en forma creciente, los datos del ejercicio anterior, usando el algoritmo *HeapSort*.

Se parte de:

*Max-Heap*: [62, 51, 42, 44, 43, 32, 34, 37, 36, 27, 21, 11].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 11:

[11, 51, 42, 44, 43, 32, 34, 37, 36, 27, 21 | 62]  
[51, 11, 42, 44, 43, 32, 34, 37, 36, 27, 21 | 62]  
[51, 44, 42, 11, 43, 32, 34, 37, 36, 27, 21 | 62]  
[51, 44, 42, 37, 43, 32, 34, 11, 36, 27, 21 | 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 44, 42, 37, 43, 32, 34, 11, 36, 27 | 51, 62]  
[44, 21, 42, 37, 43, 32, 34, 11, 36, 27 | 51, 62]  
[44, 43, 42, 37, 21, 32, 34, 11, 36, 27 | 51, 62]  
[44, 43, 42, 37, 27, 32, 34, 11, 36, 21 | 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 43, 42, 37, 27, 32, 34, 11, 36 | 44, 51, 62]  
[43, 21, 42, 37, 27, 32, 34, 11, 36 | 44, 51, 62]  
[43, 37, 42, 21, 27, 32, 34, 11, 36 | 44, 51, 62]  
[43, 37, 42, 36, 27, 32, 34, 11, 21 | 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 37, 42, 36, 27, 32, 34, 11 | 43, 44, 51, 62]  
[42, 37, 21, 36, 27, 32, 34, 11 | 43, 44, 51, 62]  
[42, 37, 34, 36, 27, 32, 21, 11 | 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 11:

[11, 37, 34, 36, 27, 32, 21 | 42, 43, 44, 51, 62]  
[37, 11, 34, 36, 27, 32, 21 | 42, 43, 44, 51, 62]  
[37, 36, 34, 11, 27, 32, 21 | 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 36, 34, 11, 27, 32 | 37, 42, 43, 44, 51, 62]  
[36, 21, 34, 11, 27, 32 | 37, 42, 43, 44, 51, 62]  
[36, 27, 34, 11, 21, 32 | 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 32:

[32, 27, 34, 11, 21 | 36, 37, 42, 43, 44, 51, 62]  
[34, 27, 32, 11, 21 | 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 27, 32, 11 | 34, 36, 37, 42, 43, 44, 51, 62]  
[32, 27, 21, 11 | 34, 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 11:

[11, 27, 21 | 32, 34, 36, 37, 42, 43, 44, 51, 62]  
[27, 11, 21 | 32, 34, 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 21:

[21, 11 | 27, 32, 34, 36, 37, 42, 43, 44, 51, 62].

Intercambiar el primero con el último, decrementar el tamaño:

[11 | 21, 27, 32, 34, 36, 37, 42, 43, 44, 51, 62].

Arreglo ordenado ascendentemente:

[11, 21, 27, 32, 34, 36, 37, 42, 43, 44, 51, 62].

**(b)** *¿Cuáles serían los pasos a seguir si se quiere ordenar en forma decreciente?*

Los pasos a seguir si se quiere ordenar en forma decreciente sería partir de una *Min-Heap*, intercambiar el primer con el último elemento del arreglo, decrementar el tamaño de éste y filtrar correspondientemente en cada caso.

### **Ejercicio 12.**

*¿Cuáles de los siguientes arreglos representan una max-heap, min-heap o ninguna de las dos?*

**(a)** *arreglo 1: 0 1 2 0 4 5 6 7 8 9.*

Ninguna de las dos.

**(b)** *arreglo 2: 9 8 7 6 5 4 3 2 1 0.*

*Max-Heap.*

**(c)** *arreglo 3: 5 5 5 6 6 6 6 7 7 1.*

Ninguna de las dos.

**(d)** *arreglo 4: 9 3 9 2 1 6 7 1 2 1.*

*Max-Heap.*

**(e)** *arreglo 5: 8 7 6 1 2 3 4 2 1 2.*

Ninguna de las dos.

**Ejercicio 13.**

*Un arreglo de 7 enteros se ordena ascendentemente usando el algoritmo HeapSort. Luego de la fase inicial del algoritmo (la construcción de la heap), ¿cuál de los siguientes es un posible orden del arreglo?*

- (a) 85 78 45 51 53 47 49.
- (b) 85 49 78 45 47 51 53.
- (c) 85 78 49 45 47 51 53.
- (d) 45 85 78 53 51 49 47.
- (e) 85 51 78 53 49 47 45.

**Ejercicio 14.**

*En una Heap, ¿para un elemento que está en la posición  $i$ , su hijo derecho está en la posición ...?*

- (a)  $\left\lfloor \frac{i}{2} \right\rfloor$ .
- (b)  $2i$ .
- (c)  $2i + 1$ .
- (d) Ninguna de las anteriores.

### **Ejercicio 15.**

*¿Siempre se puede decir que un árbol binario lleno es una Heap?*

(a) *Sí.*

(b) *No.*

**Ejercicio 16.**

*La operación que agrega un elemento a la heap que tiene  $n$  elementos, en el peor caso, es de ...*

- (a)  $O(n)$ .      (b)  $O(n \log n)$ .      (c)  $O(\log n)$ .      (d) Ninguna de las anteriores.



**Ejercicio 17.**

Se construyó una Max-Heap con las siguientes claves: 13, 21, 87, 30, 25, 22, 18. ¿Cuál de las siguientes opciones corresponde al resultado de realizar la construcción insertando las claves una a una?

- (a) 87, 30, 25, 22, 21, 18, 13.
- (b) 87, 30, 22, 21, 25, 13, 18.
- (c) 87, 30, 25, 13, 22, 18, 21.
- (d) 87, 30, 22, 13, 25, 21, 18.

[13] - *Insert(13)*

[13, 21] - *Insert(21)*

[21, 13]

[21, 13, 87] - *Insert(87)*

[87, 13, 21]

[87, 13, 21, 30] - *Insert(30)*

[87, 30, 21, 13]

[87, 30, 21, 13, 25] - *Insert(25)*

[87, 30, 21, 13, 25, 22] - *Insert(22)*

[87, 30, 22, 13, 25, 21]

[87, 30, 22, 13, 25, 21, 18] - *Insert(18)*

Max-Heap: [87, 30, 22, 13, 25, 21, 18].

**Ejercicio 18.**

Se construyó una *Max-Heap* con las siguientes claves: 13, 21, 87, 30, 25, 22, 18. ¿Cuál de las siguientes opciones corresponde al resultado de realizar la construcción aplicando el algoritmo *Build-Heap*?

- (a) 87, 30, 25, 22, 21, 18, 13.
- (b) 87, 30, 22, 21, 25, 13, 18.
- (c) 87, 30, 25, 13, 22, 18, 21.
- (d) 87, 30, 22, 13, 25, 21, 18.

Se parte de:

[13, 21, 87, 30, 25, 22, 18].

Se empieza en la posición  $i = \left\lfloor \frac{n}{2} \right\rfloor = 3$  del arreglo, el 87, y se filtra:

[13, 21, 87, 30, 25, 22, 18].

Se avanza a la posición anterior del arreglo, el 21, y se filtra:

[13, 30, 87, 21, 25, 22, 18].

Se avanza a la posición anterior del arreglo, el 13, y se filtra:

[87, 30, 13, 21, 25, 22, 18]  
[87, 30, 22, 21, 25, 13, 18].

*Max-Heap*: [87, 30, 22, 21, 25, 13, 18].

**Ejercicio 19.**

*El algoritmo HeapSort consta de dos etapas:*

- (1) Se construye una heap y*
- (2) Se realizan los intercambios necesarios para dejar ordenados los datos.*

*Asumir que la heap ya está construida y es la siguiente: 58 38 53 23 28 40 35 18.*

*¿Cómo quedan los datos en el arreglo después de ejecutar sólo 2 pasos de la segunda etapa del HeapSort?*

- (a)** 40 38 23 28 35 18 53 58.
- (b)** 53 38 40 23 28 18 35 58.
- (c)** 40 38 23 35 28 18 53 58.
- (d)** 40 38 35 23 28 18 53 58.

Se parte de:

*Max-Heap:* [58, 38, 53, 23, 28, 40, 35, 18].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 18:

[18, 38, 53, 23, 28, 40, 35 | 58]  
[53, 38, 18, 23, 28, 40, 35 | 58]  
[53, 38, 40, 23, 28, 18, 35 | 58].

Intercambiar el primero con el último, decrementar el tamaño y filtrar el 35:

[35, 38, 40, 23, 28, 18, 35 | 53, 58]  
[40, 38, 35, 23, 28, 18 | 53, 58].

**Ejercicio 20.**

*Dada la Min-Heap 3, 8, 5, 15, 10, 7, 19, 28, 16, 25, 12. ¿En qué posición está ubicado el hijo derecho de la clave 15?*

- (a) 7.
- (b) 8.
- (c) 9.
- (d) 10.

$$2i + 1 = 2 * 4 + 1$$

$$2i + 1 = 8 + 1$$

$$2i + 1 = 9.$$

**Ejercicio 21.**

Construir una min-heap con las siguientes claves: 15, 25, 23, 13, 18, 2, 19, 20, 17, insertándose una a una. Indicar en qué posiciones quedaron ubicadas las claves: 2, 18 y 25.

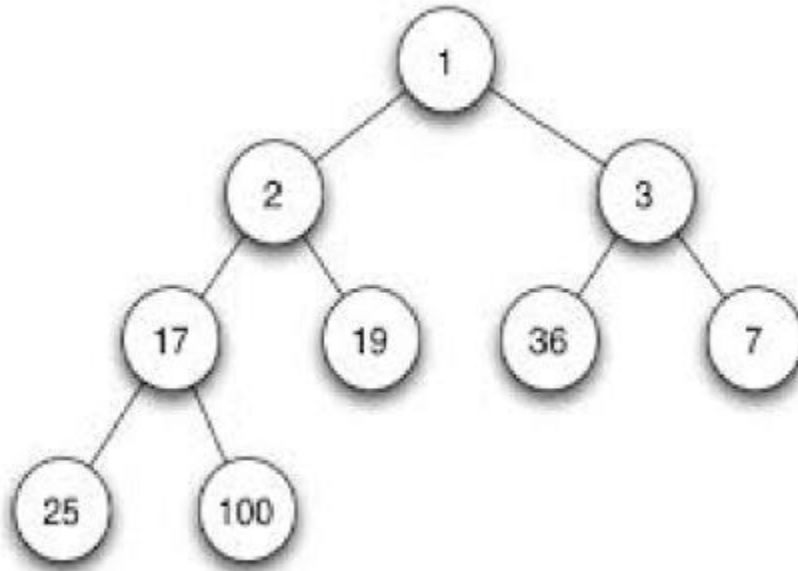
[15] - *Insert(15)*  
[15, 25] - *Insert(25)*  
[15, 25, 23] - *Insert(23)*  
[15, 25, 23, 13] - *Insert(13)*  
[13, 15, 23, 25]  
[13, 15, 23, 25, 18] - *Insert(18)*  
[13, 15, 23, 25, 18, 2] - *Insert(2)*  
[13, 15, 2, 25, 18, 23]  
[2, 15, 13, 25, 18, 23]  
[2, 15, 13, 25, 18, 23, 19] - *Insert(19)*  
[2, 15, 13, 25, 18, 23, 19, 20] - *Insert(20)*  
[2, 15, 13, 20, 18, 23, 19, 25]  
[2, 15, 13, 20, 18, 23, 19, 25, 17] - *Insert(17)*  
[2, 15, 13, 17, 18, 23, 19, 25, 20].

*Min-Heap:* [2, 15, 13, 17, 18, 23, 19, 25, 20].

Las claves 2, 18 y 25 quedaron ubicadas en las posiciones 1, 5 y 8, respectivamente.

**Ejercicio 22.**

Luego de insertar la clave 15 en la siguiente min-heap, ¿cuántas de las claves que ya estaban en la heap han mantenido su lugar (es decir, ocupan en la min-heap resultante la misma posición que ocupaban antes de la inserción)?



- (a) Ninguna.
- (b) Seis.
- (c) Ocho.
- (d) Nueve.

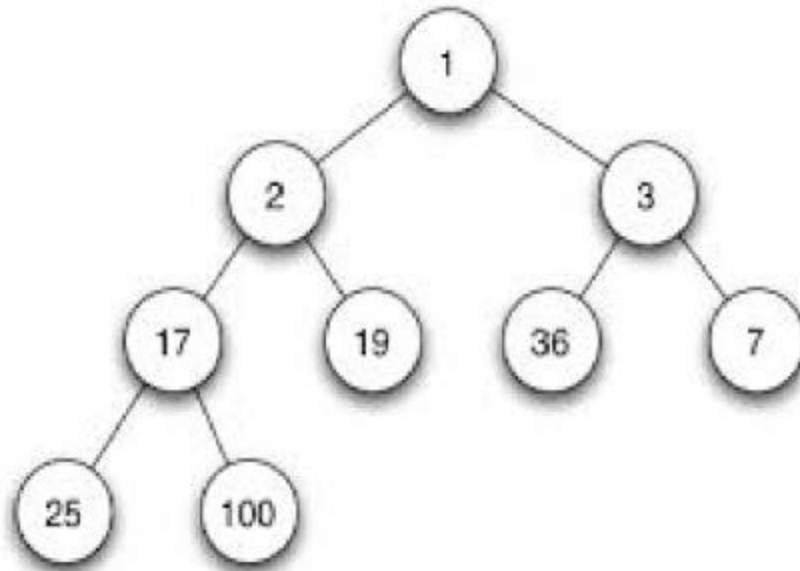
[1, 2, 3, 17, 19, 36, 7, 25, 100]

[1, 2, 3, 17, 19, 36, 7, 25, 100, 15] - Insert(15)

[1, 2, 3, 17, 15, 36, 7, 25, 100, 19].

**Ejercicio 23.**

Luego de una operación de borrado del mínimo en la siguiente min-heap, ¿cuántas claves han cambiado de lugar (es decir, ocupan en la min-heap resultante un lugar diferente al que ocupaban en la min-heap antes del borrado) ? (No contar la clave borrada, ya que no pertenece más a la heap).



- (a) Ninguna.
- (b) Dos.
- (c) Tres.
- (d) Cuatro.

[1, 2, 3, 17, 19, 36, 7, 25, 100]  
[100, 2, 3, 17, 19, 36, 7, 25] - DeleteMin()  
[2, 100, 3, 17, 19, 36, 7, 25]  
[2, 17, 3, 100, 19, 36, 7, 25]  
[2, 17, 3, 25, 19, 36, 7, 100].