

Traza — Heap_H clásico (completo por niveles)

Construcción de un heap con Insert 5, 3, 8, 1 y borrado del mínimo; ejercita las tres reglas de Insert y el DeleteMin clásico (último BFS + sift-down).

Términos a evaluar:

```
Heap_DeleteMin(  
  Heap_Insert(  
    Heap_Insert(  
      Heap_Insert(  
        Heap_Insert(Heap_Empty() as Heap_H, 5),  
        3),  
      8),  
    1))
```

Spec: Heap_H_complete.nereus. Convención: · = Heap_Empty() as Heap_H.

Fase 1 — Construcción del heap

Insert 5 — caso base

Sobre Heap_Empty, axioma base: produce singleton.

```
BTree_Create(·, ·, 5) as Heap_H
```

Estado

5

Insert 3 — Regla A ($|L|=|R|=0$)

Sobre BTree_Create(·, ·, 5): $|L|=0, |R|=0, \text{not}(0>0) \checkmark \rightarrow$ Regla A, descender a L.
Recurción: Heap_Insert(·, 3) = BTree_Create(·, ·, 3).
Heap_CombineL(BTree_Create(·, ·, 3), ·, 5): $m=3 < x=5 \rightarrow$ swap.

```
BTree_Create(BTree_Create(·, ·, 5) as Heap_H, ·, 3) as Heap_H
```

Estado

3
/
5

Insert 8 — Regla C ($|L|=1>|R|=0$, L perfecto)

Sobre `BTree_Create(BTree_Create(·, ·, 5), ·, 3)`: $|L|=1$, $|R|=0$, $|L|>|R|$ ✓, `perfect(L)` ✓ (singleton) → Regla C, descender a R.

Recursión: `Heap_Insert(·, 8) = BTree_Create(·, ·, 8)`.

`Heap_CombineR(L, BTree_Create(·, ·, 8), 3)`: $m=8 \geq x=3 \rightarrow$ **no swap**.

```
BTree_Create(  
  BTree_Create(·, ·, 5) as Heap_H,  
  BTree_Create(·, ·, 8) as Heap_H,  
  3) as Heap_H
```

Estado

```
  3  
 / \  
5  8
```

Insert 1 — Regla A con sift-up doble

Sobre `BTree_Create(BTree_Create(·, ·, 5), BTree_Create(·, ·, 8), 3)`: $|L|=1$, $|R|=1$, `not(1>1)` ✓ → Regla A, descender a L.

Sub-paso: `Heap_Insert(BTree_Create(·, ·, 5), 1)`:

- Sobre `BTree_Create(·, ·, 5)`: $|L'|=|R'|=0 \rightarrow$ Regla A, descender.
- `Heap_Insert(·, 1) = BTree_Create(·, ·, 1)`.
- `Heap_CombineL(BTree_Create(·, ·, 1), ·, 5)`: $1 < 5 \rightarrow$ **swap**: `BTree_Create(BTree_Create(·, ·, 5), ·, 1)`.

Vuelta al outer: `Heap_CombineL(BTree_Create(BTree_Create(·, ·, 5), ·, 1), BTree_Create(·, ·, 8), 3)`: $m=1 < x=3 \rightarrow$ **swap**.

```
BTree_Create(  
  BTree_Create(BTree_Create(·, ·, 5) as Heap_H, ·, 3) as Heap_H,  
  BTree_Create(·, ·, 8) as Heap_H,  
  1) as Heap_H
```

Estado (heap completo de 4 nodos)

```
  1  
 / \  
3  8  
/  
5
```

Forma OK: posiciones 1, 2, 3, 4 llenas; el "1" subió desde la posición 4 hasta la raíz. Propiedad heap OK: $1 \leq 3$, $1 \leq 8$, $3 \leq 5$.

Fase 2 — DeleteMin sobre el heap construido

Aplicamos `Heap_DeleteMin(T)` donde `T` es el heap del paso anterior. Como `T` no es singleton, se aplica el axioma:

```
Heap_DeleteMin(BTree_Create(L, R, x) as Heap_H) =
  Heap_SiftDown(
    Heap_ReplaceRoot(
      Heap_RemoveLast(BTree_Create(L, R, x) as Heap_H),
      Heap_Last(BTree_Create(L, R, x) as Heap_H)
    )
  );
```

Paso D1 — Calcular `Heap_Last(T)`

$T = \text{BTree_Create}(L, R, 1)$ con $|L|=2, |R|=1. |L| > |R| \checkmark$.
 $\text{perfect}(L)? L = \text{BTree_Create}(\text{BTree_Create}(\cdot, \cdot, 5), \cdot, 3): |LL|=1 \neq |LR|=0 \rightarrow \text{no perfecto}$.
 Condición "último en L ": $|L| > |R| \wedge (\neg \text{perfect}(L) \vee \text{perfect}(R)) = T \wedge (T \vee T) = T \rightarrow \text{último en } L$.

Recursión: `Heap_Last(L)` con $L = \text{BTree_Create}(\text{BTree_Create}(\cdot, \cdot, 5), \cdot, 3)$:
 $|L'|=1, |R'|=0, |L'| > |R'|, \text{perfect}(L')$ (singleton). Último en L' .
`Heap_Last(BTree_Create( $\cdot, \cdot, 5$ ))`: singleton $\rightarrow 5$.

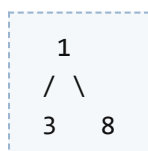
Resultado: `Heap_Last(T) = 5`.

Paso D2 — Calcular `Heap_RemoveLast(T)`

Mismo análisis que `Heap_Last`: último en L . Aplico el axioma:
`Heap_RemoveLast(BTree_Create(L, R, x)) = BTree_Create(Heap_RemoveLast(L), R, x) as Heap_H`.
Recursión: `Heap_RemoveLast(L)` con $L = \text{BTree_Create}(\text{BTree_Create}(\cdot, \cdot, 5), \cdot, 3)$:
 último en $L' \rightarrow \text{BTree_Create}(\text{Heap_RemoveLast}(\text{BTree_Create}(\cdot, \cdot, 5)), \cdot, 3)$.
`Heap_RemoveLast(BTree_Create( $\cdot, \cdot, 5$ )) = Heap_Empty` (caso singleton).
 Resultado interno: `BTree_Create( $\cdot, \cdot, 3$ ) as Heap_H`.

```
BTree_Create(
  BTree_Create( $\cdot, \cdot, 3$ ) as Heap_H,
  BTree_Create( $\cdot, \cdot, 8$ ) as Heap_H,
  1) as Heap_H
```

`T` sin la última posición (raíz sigue siendo 1)



Paso D3 — Heap_ReplaceRoot(..., 5)

Axioma: $\text{Heap_ReplaceRoot}(\text{BTree_Create}(L, R, x), m) = \text{BTree_Create}(L, R, m)$.
Reemplazo 1 (la raíz vieja) por 5 (el "último" que retiramos):

```
BTree_Create(  
  BTree_Create(·, ·, 3) as Heap_H,  
  BTree_Create(·, ·, 8) as Heap_H,  
  5) as Heap_H
```

Antes de sift-down (la raíz 5 puede violar la propiedad heap)



La propiedad de heap se violó ($5 > 3$, raíz mayor que su hijo izquierdo). Heap_SiftDown la restaura.

Paso D4 — Heap_SiftDown

Sobre $\text{BTree_Create}(\text{BTree_Create}(\cdot, \cdot, 3), \text{BTree_Create}(\cdot, \cdot, 8), 5)$: ambos hijos no vacíos $\rightarrow$ $\text{Heap_SDBoth}(L, R, 5)$.

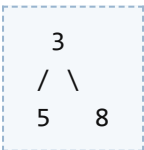
SDBoth con $m_1 = 3$ (raíz de L), $m_2 = 8$ (raíz de R), $x = 5$:

Caso	Condición	Aplicable
(a)	$\neg(m_1 < x) \wedge \neg(m_2 < x) = \neg(3 < 5) \wedge \neg(8 < 5) = F \wedge T$	No
(b)	$(m_1 < x) \wedge \neg(m_2 < m_1) = (3 < 5) \wedge \neg(8 < 3) = T \wedge T$	Sí $\leftarrow$
(c)	$(m_2 < x) \wedge (m_2 < m_1) = (8 < 5) \wedge (8 < 3) = F \wedge F$	No

Caso (b): el menor de los hijos es $m_1=3$. Swap con izquierda y recursar sobre el nuevo L.
Sub-paso: $\text{Heap_SiftDown}(\text{BTree_Create}(\cdot, \cdot, 5))$ — hoja $\rightarrow$ no hace nada. Devuelve $\text{BTree_Create}(\cdot, \cdot, 5)$.

```
BTree_Create(  
  BTree_Create(·, ·, 5) as Heap_H,  
  BTree_Create(·, ·, 8) as Heap_H,  
  3) as Heap_H
```

Forma normal: heap tras DeleteMin



Resumen y verificación

Antes / Después	Heap	Mín	Tamaño	Forma	Heap order
Antes (4 nodos)	{1, 3, 5, 8}	1	4	✓ (posiciones 1–4)	✓
Después (3 nodos)	{3, 5, 8}	3	3	✓ (posiciones 1–3)	✓ ($3 \leq 5$, $3 \leq 8$)

Lectura semántica: el heap contenía {1, 3, 5, 8} con 1 a la raíz. DeleteMin sacó el mínimo (1) y dejó el heap clásico con 3 elementos: 3 a la raíz, 5 a la izquierda, 8 a la derecha. El "5" venía de la última posición y trepó vía el sift-down hacia donde estaba 3, mientras que 3 (el nuevo mínimo) subió a la raíz.

Verificación de invariantes:

- **Forma:** 3 nodos en posiciones 1, 2, 3 — completo por niveles. ✓
- **Heap order:** $3 \leq 5$ (hijo izq), $3 \leq 8$ (hijo der). ✓
- **Tamaño:** exactamente uno menos que antes ($4 \rightarrow 3$). ✓

Traza generada para la spec `Heap_H_complete.nereus`. Pasos totales: 4 (construcción) + 4 (DeleteMin) = 8 reducciones de alto nivel; muchas más si se desglosan las sub-llamadas internas.