



UNIVERSIDAD NACIONAL DE
INGENIERÍA

Monticulos a la Izquierda

Leftist Heap

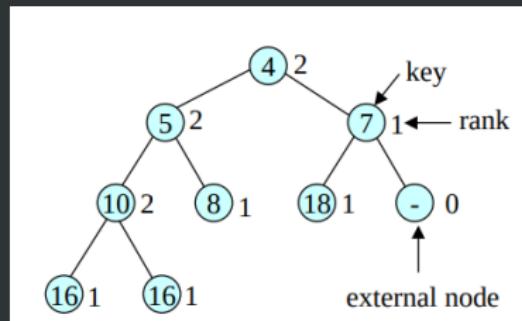
Felix Romo, Carlos

Leftist Heap

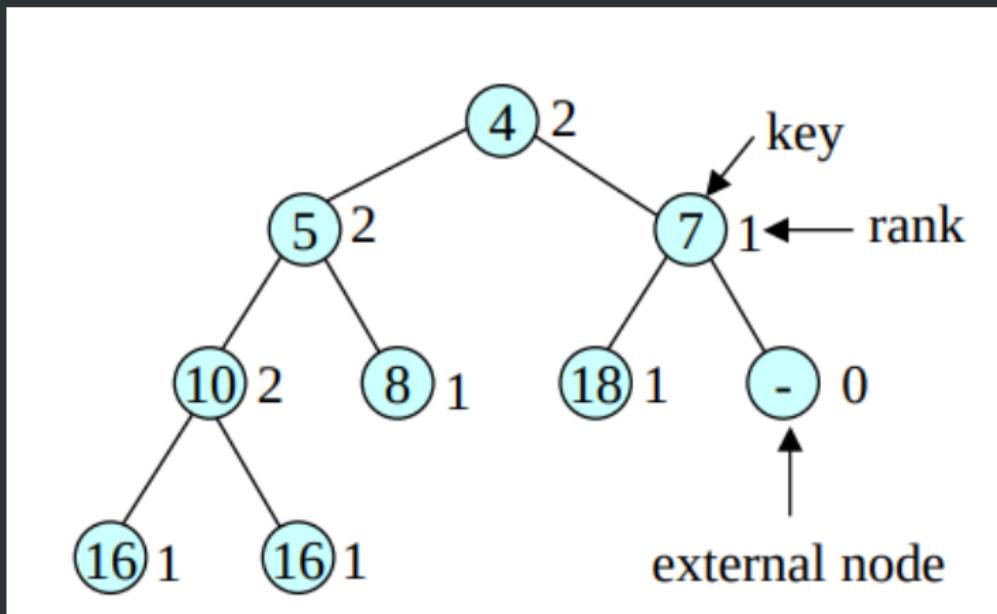
Un montículo a la izquierda o leftist heap es una cola de prioridad implementado como una variante de un montículo binario.

Cada nodo tiene un “rank” o “S-value” que es la menor distancia del nodo a un nodo externo.

A diferencia del montículo, el montículo a la izquierda puede ser muy desbalanceado ya que tiende a ser más pesado a la izquierda.

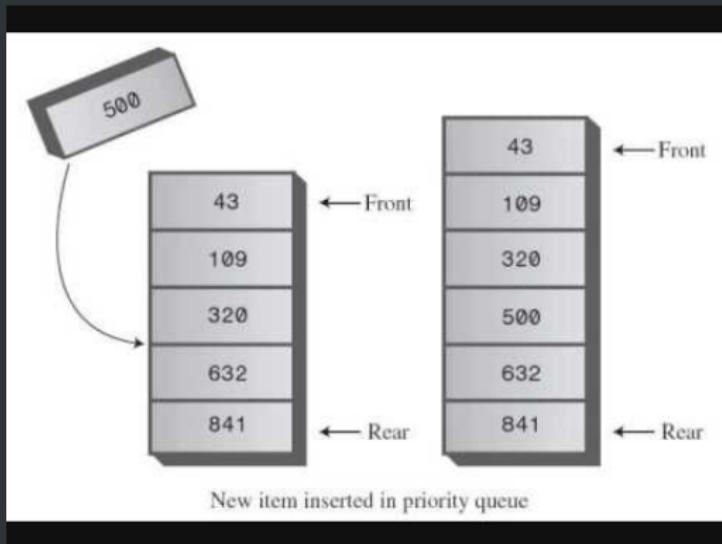


Leftist Heap



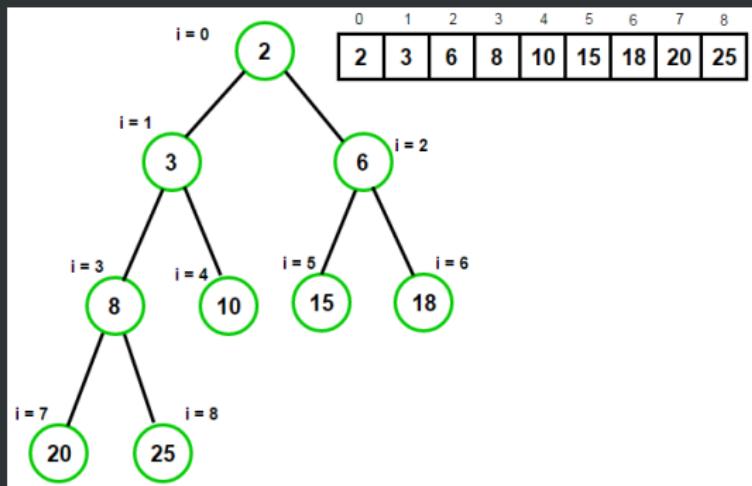
Definiciones Previas

- Colas de prioridad: Es un TDA similar a la cola donde los elementos tienen adicionalmente una prioridad asignada. Así en una cola de prioridad el elemento con mayor prioridad (el menor numero) sera desencolado antes.



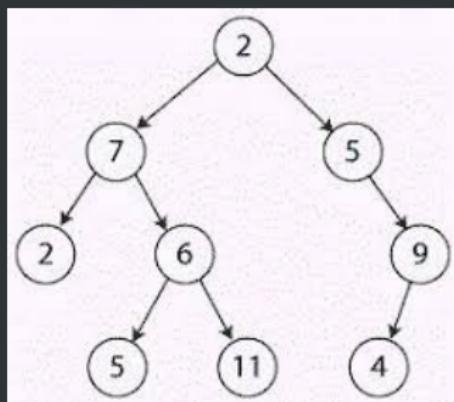
Definiciones Previas

- Min-heap: Los montículos son arboles binarios completos que cumplen la condicion de montículo, en este caso es que el valor del nodo padre es siempre menor o igual al de sus nodos hijos. Son útiles para implementar colas de prioridad.



Definiciones Previas

- Nodo externo: Un nodo externo en un arbol binario es aquel que tiene menos de dos hijos.



- 2, 5, 11, 4, 9 y 5 son nodos externos.

Propiedades.

Un leftist heap cumple 3 propiedades:

- Es un árbol binario.

Propiedades.

Un leftist heap cumple 3 propiedades:

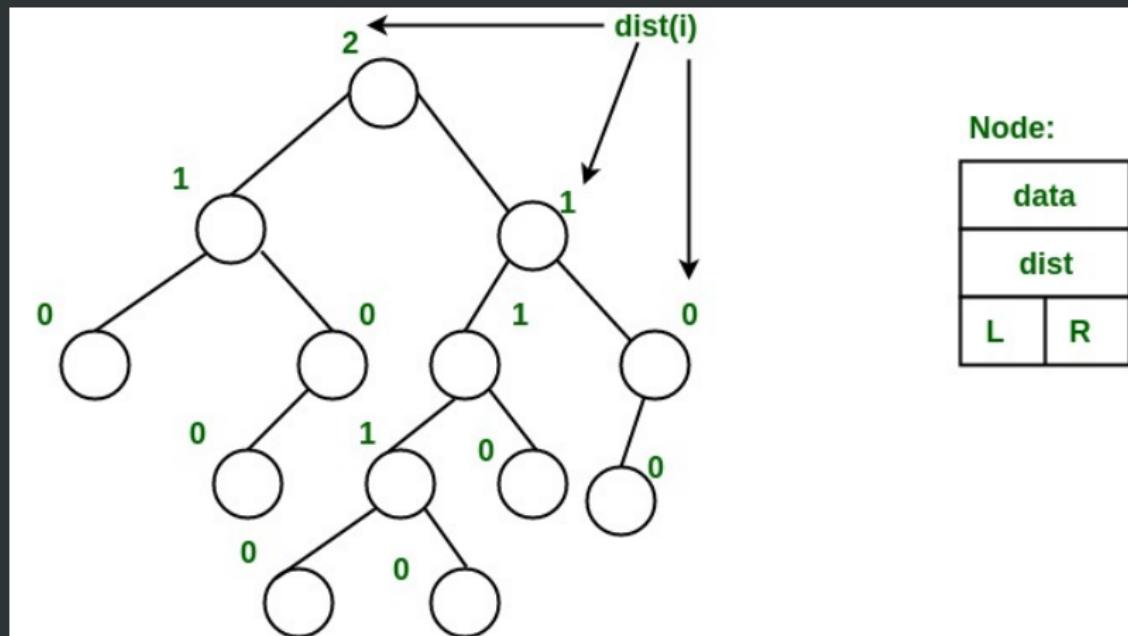
- Es un árbol binario.
- Cumple la condición de montículo (min-heap).

Propiedades.

Un leftist heap cumple 3 propiedades:

- Es un árbol binario.
- Cumple la condición de montículo (min-heap).
- El rank o S-value del hijo izquierdo de cualquier nodo es mayor o igual al de su hijo derecho.

Estructura.



Operaciones.

Sus operaciones se basan en el merging (combinar dos leftist heap):

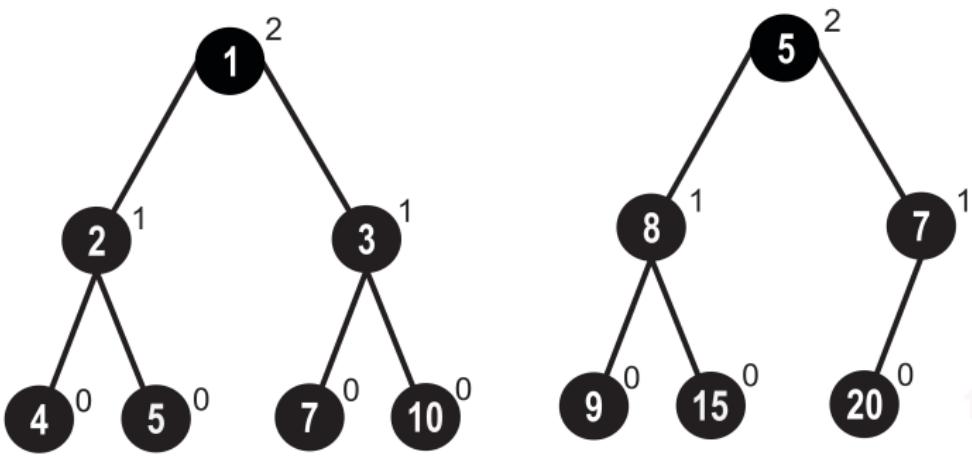
- La principal operacion es Merge().
- deleteMin() se puede hacer removiendo la raíz y llamando a Merge() de sus subarboles izquierdo y derecho.
- Insert() se puede hacer creando un nuevo leftist heap con un solo elemento(a insertar) y llamando a Merge() entre el arbol principal y el nuevo creado.

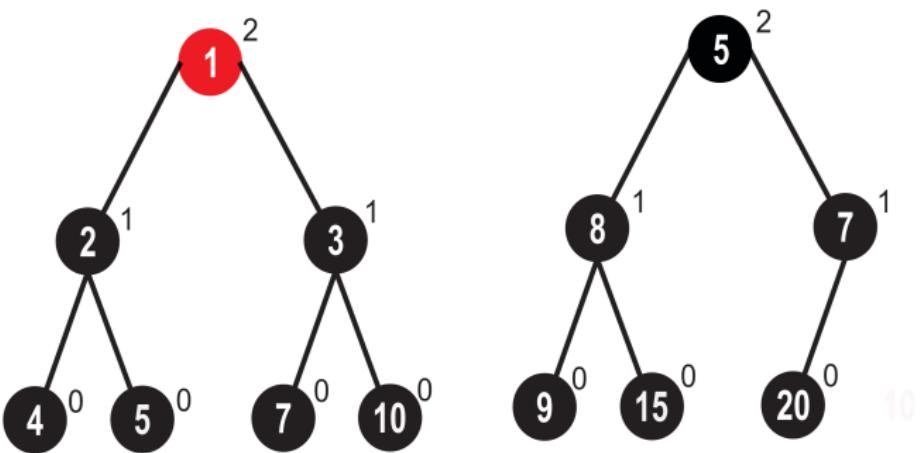
Merge.

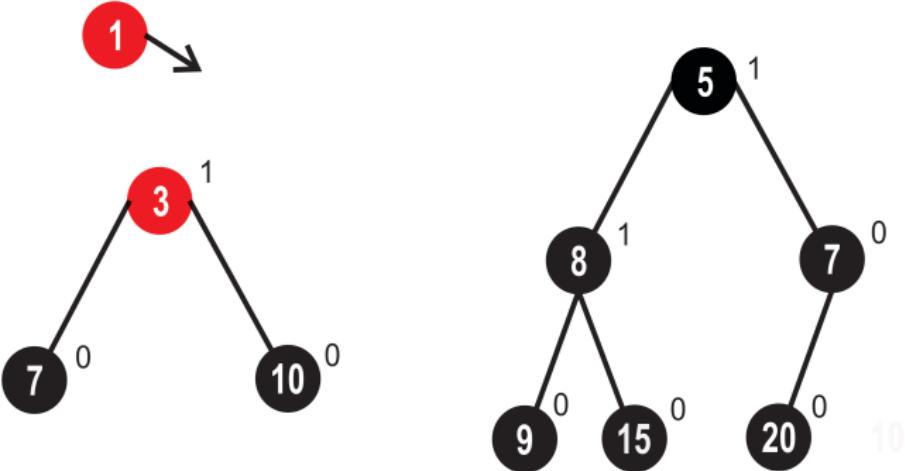
Algoritmo Recursivo para el Merge:

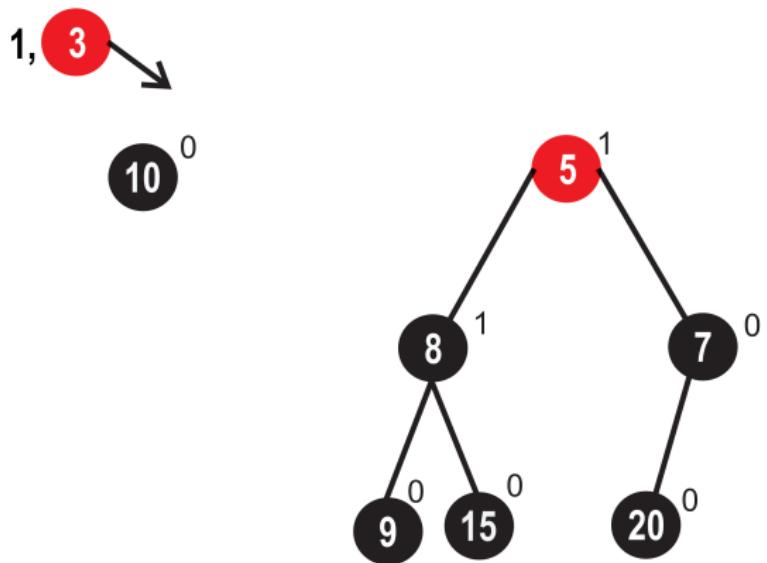
Merge(r_1, r_2)

1. Si r_1 o r_2 es NULL retorna la la raíz no nula.
2. Si el hijo derecho de la raiz con menor valor es NULL, la otra raíz(de mayor valor) pasa a ser el hijo derecho de la primera.
3. Sino el hijo derecho de la raíz con menor valor será el resultante del merge entre el subarbol derecho(hijo derecho) de esta y el arbol con la raíz de mayor valor.
4. intercambiamos los hijos de la raiz del heap resultante si el s-value del derecho es mayor que el del izquierdo.
5. Se actualiza el s-value de la raiz del heap resultante y se retorna esta raiz.





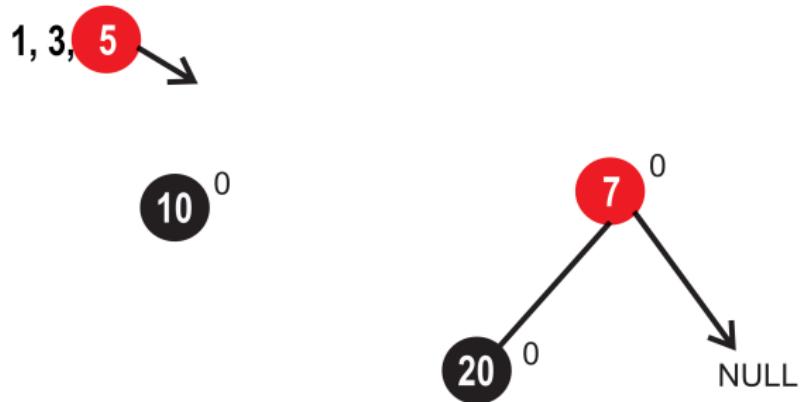


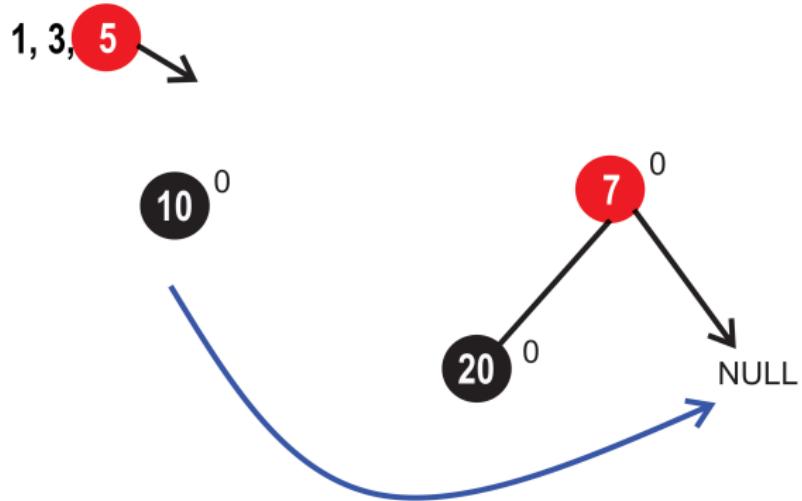


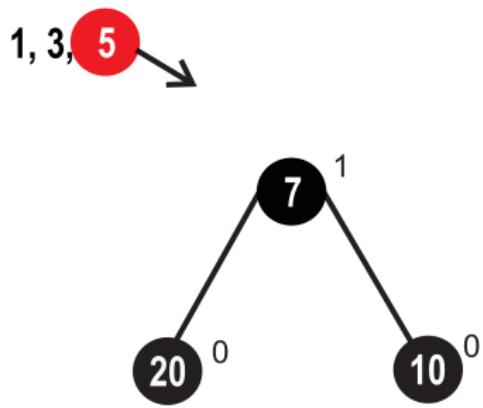
1, 3, 5

10⁰

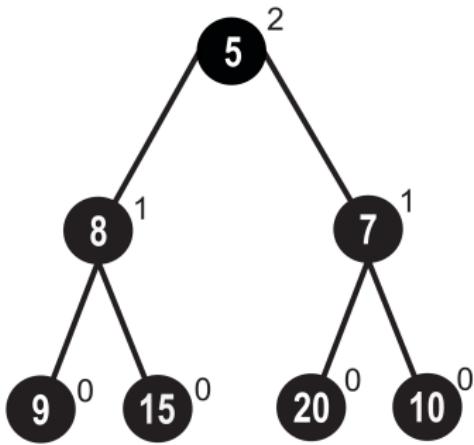
20⁰

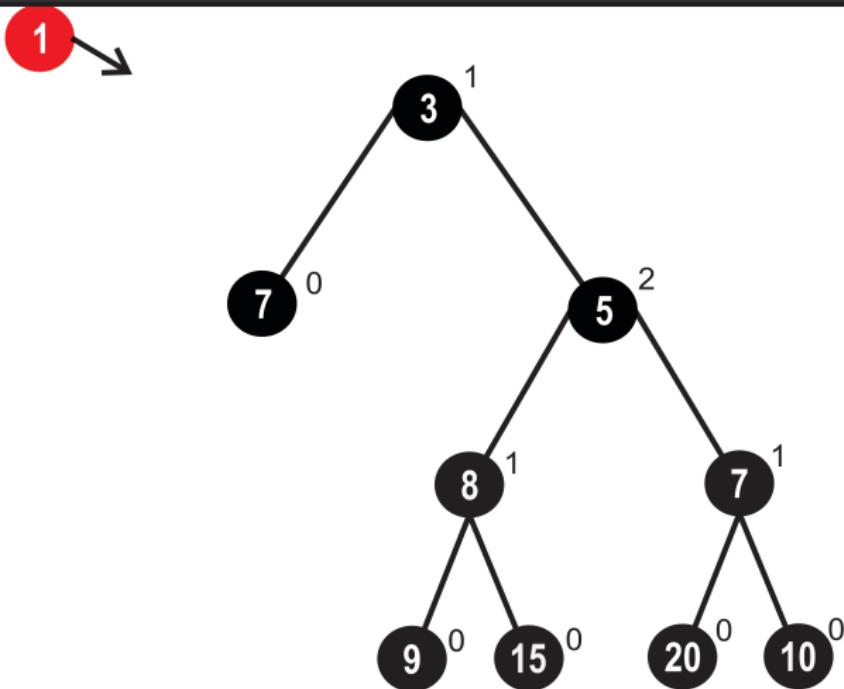


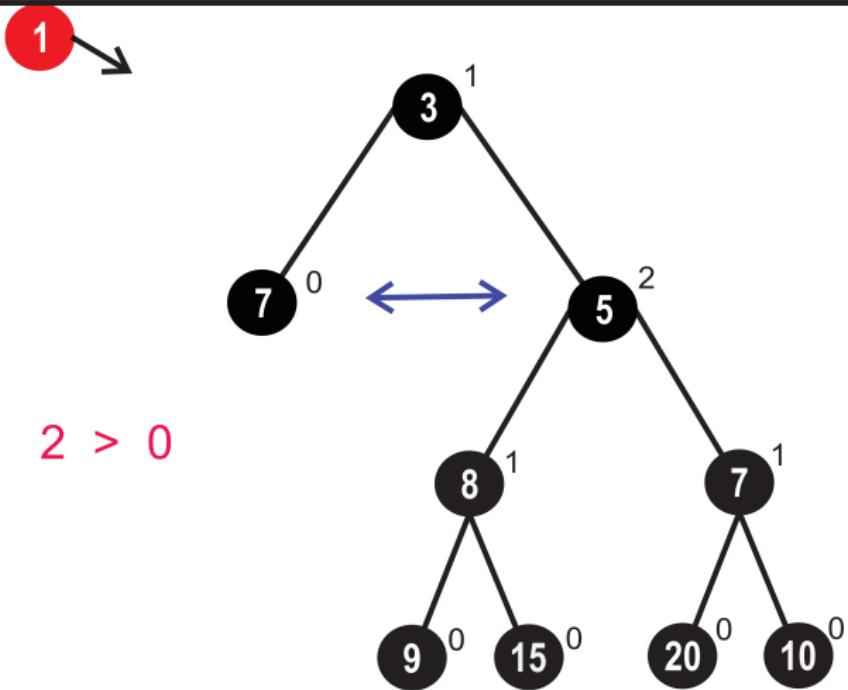




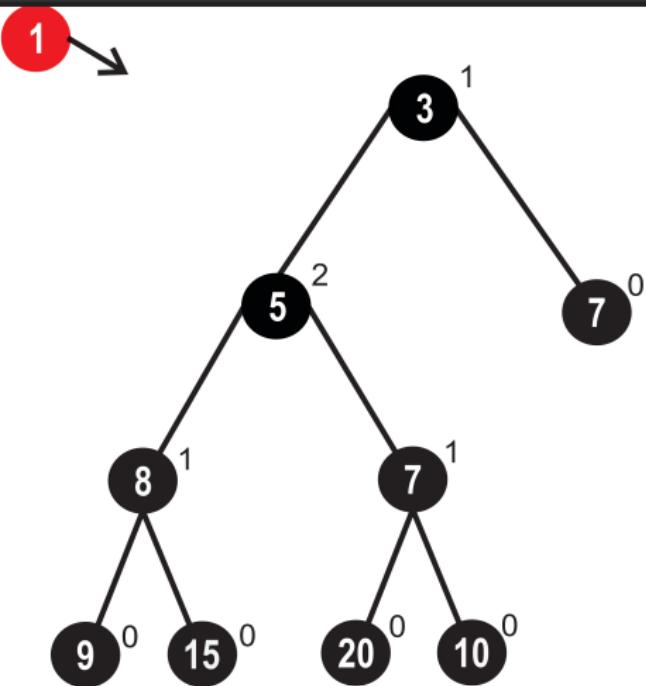
1, 3

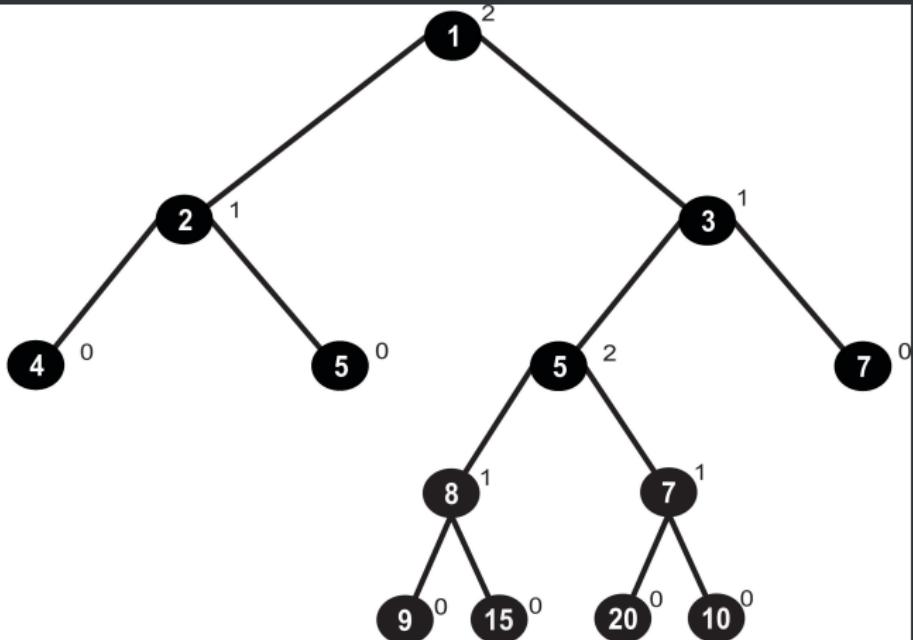






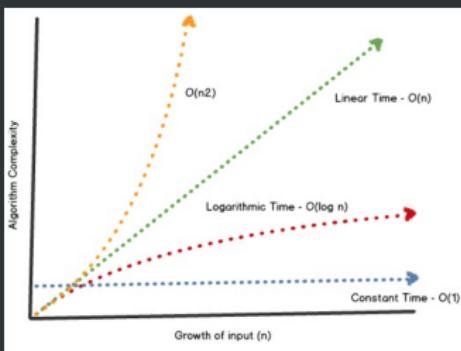
$2 > 0$



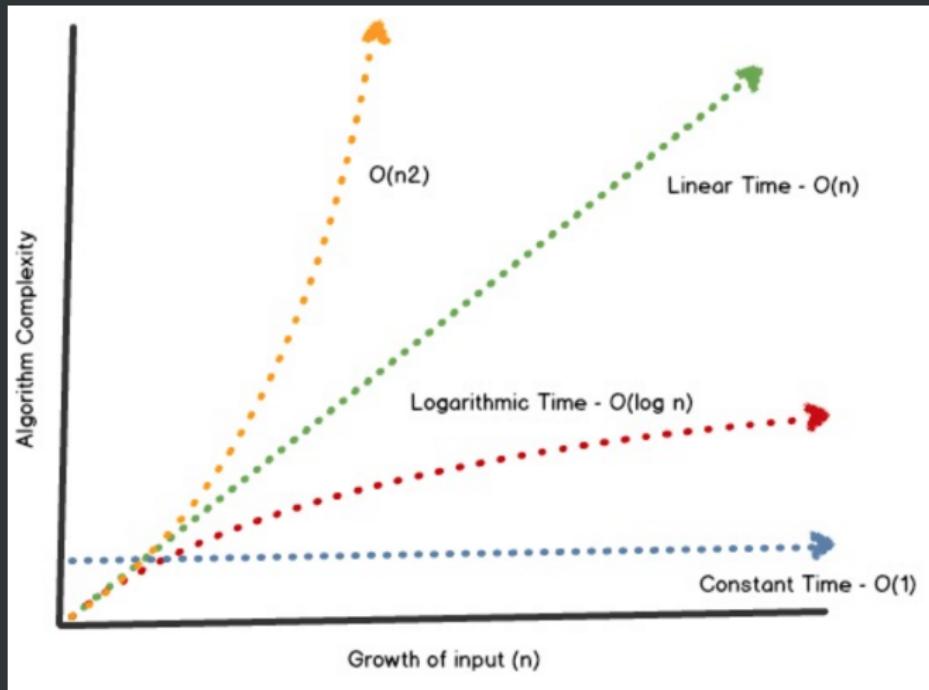


Ventajas:

El leftist heap es ventajoso debido a que es un mergeable heap (heap fusionable) muy rápido comparado con los montículos binarios (binary Heap) que es la implementación más común de las colas de prioridad el cual demora $\Theta(n)$, mientras que el leftist heap en el peor de los casos tiene una complejidad de $O(\log n)$.



Ventajas:





UNIVERSIDAD NACIONAL DE
INGENIERÍA

Montículos oblicuos

skew Heap

Felix Romo, Carlos

Skew Heap

Un monticulo oblicuo es una forma autoajustable de un leftist Heap. El monticulo oblicuo intenta mantener el balance intercambiando hijos en cada paso(sin ninguna condicion).

Es la version simplificada (menos costosa en memoria) del leftist heap ya que ya no se requiere de S-Value.

Aunque el intercambio se hace en cada paso al analizar la complejidad amortizada es la misma que el peor caso para cualquier operación del leftist heap.

Merge.

Algoritmo Recursivo para el Merge del montículo oblicuo:

Merge(r_1, r_2)

1. Si r_1 o r_2 es NULL retorna la la raíz no nula.
2. Si el hijo derecho de la raiz con menor valor es NULL, la otra raíz(de mayor valor) pasa a ser el hijo derecho de la primera.
3. Sino el hijo derecho de la raíz con menor valor será el resultante del merge entre el subarbol derecho(hijo derecho) de esta y el arbol con la raíz de mayor valor.
4. intercambiamos los hijos de la raiz del heap resultante y se retorna esta raiz.

Skew Heap Complexity

Operation	Amortized
MakeHeap	$O(1)$
FindMin	$O(1)$
Union	$O(\log n)$
Insert	$O(\log n)$
DeleteMin	$O(\log n)$
DecreaseKey	$O(\log n)$
Delete	$O(\log n)$

Binomial Heap (discussed next) improves these amortized times.

Conclusiones

El skew Heap es probablemente la implementacion más simple de un montículo(o cola de prioridad) que garantiza una complejidad de $O(\ln(n))$ (amortizada) para la combinacion de dos montículos.

Operation	find-min	delete-min	insert	decrease-key	meld
Binary ^[13]	$\Theta(1)$	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$	$\Theta(n)$
Leftist	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$
Binomial ^{[13][14]}	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)^{[c]}$	$\Theta(\log n)$	$O(\log n)^{[d]}$
Fibonacci ^{[13][15]}	$\Theta(1)$	$O(\log n)^{[c]}$	$\Theta(1)$	$\Theta(1)^{[c]}$	$\Theta(1)$
Pairing ^[16]	$\Theta(1)$	$O(\log n)^{[c]}$	$\Theta(1)$	$O(\log n)^{[c][e]}$	$\Theta(1)$
Brodal ^{[19][f]}	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Rank-pairing ^[21]	$\Theta(1)$	$O(\log n)^{[c]}$	$\Theta(1)$	$\Theta(1)^{[c]}$	$\Theta(1)$
Strict Fibonacci ^[22]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2-3 heap ^[23]	$O(\log n)$	$O(\log n)^{[c]}$	$O(\log n)^{[c]}$	$\Theta(1)$?

Conclusiones

Operation	find-min	delete-min	insert	decrease-key	meld
Binary ^[13]	$\Theta(1)$	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$	$\Theta(n)$
Leftist	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$	$\Theta(\log n)$
Binomial ^{[13][14]}	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)^{[c]}$	$\Theta(\log n)$	$O(\log n)^{[d]}$
Fibonacci ^{[13][15]}	$\Theta(1)$	$O(\log n)^{[c]}$	$\Theta(1)$	$\Theta(1)^{[c]}$	$\Theta(1)$
Pairing ^[16]	$\Theta(1)$	$O(\log n)^{[c]}$	$\Theta(1)$	$o(\log n)^{[c][e]}$	$\Theta(1)$
Brodal ^{[19][f]}	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Rank-pairing ^[21]	$\Theta(1)$	$O(\log n)^{[c]}$	$\Theta(1)$	$\Theta(1)^{[c]}$	$\Theta(1)$
Strict Fibonacci ^[22]	$\Theta(1)$	$O(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2-3 heap ^[23]	$O(\log n)$	$O(\log n)^{[c]}$	$O(\log n)^{[c]}$	$\Theta(1)$?