Leftist Heaps

Summary of complexity of operations on Binary Heaps:

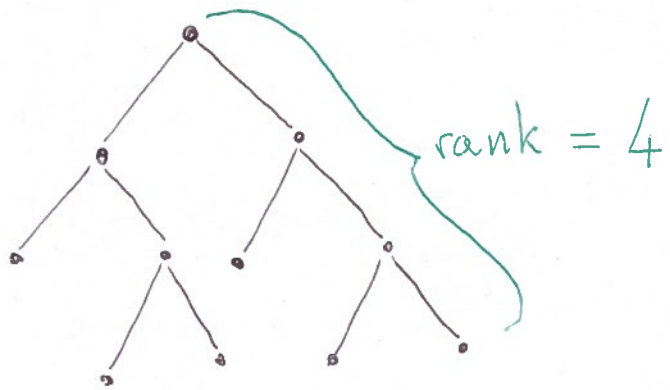
- insert: $O(\log n)$
- minimum: $O(1)$
- extract: $O(\log n)$

We haven't seen how to perform union.
The naive way is to extract the elements from one heap and insert them into the second.
The best algorithm can do it in $O(n)$.

Leftist Heaps are a structure for which we can do union in $O(\log n)$

rank of a binary tree:
length of its rightmost spine

ex:

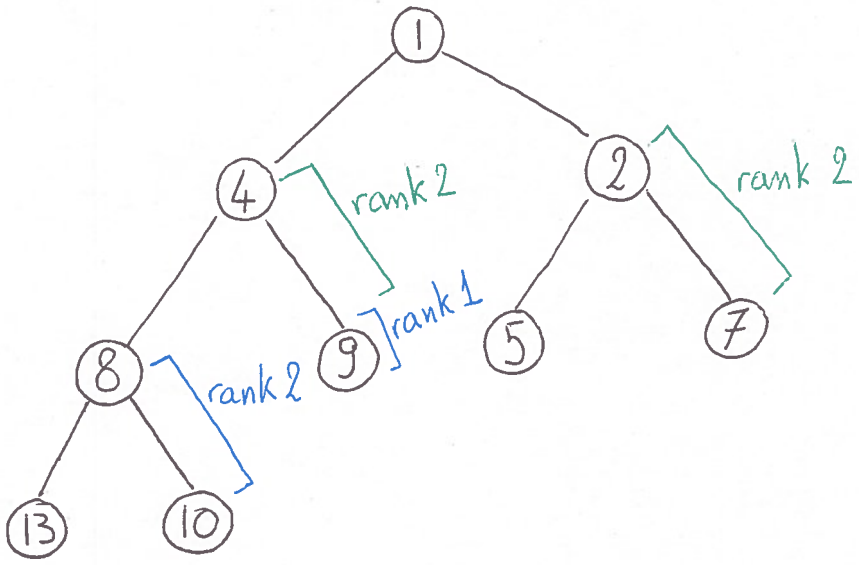


A leftist heap is a binary tree such that:

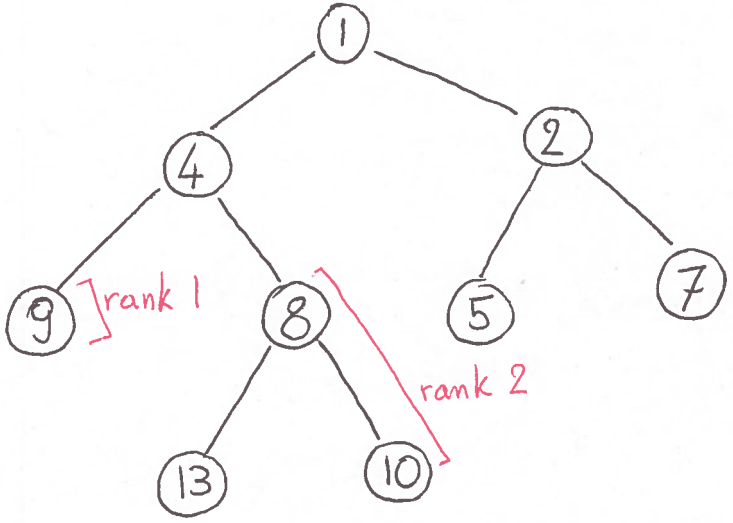
- It has the heap property: every node is smaller or equal to its children;
- For every node, the rank of the left child is larger or equal to the rank of the right child.

Example:

This is a correct Leftist Heap:



This is not a correct Leftist Heap:

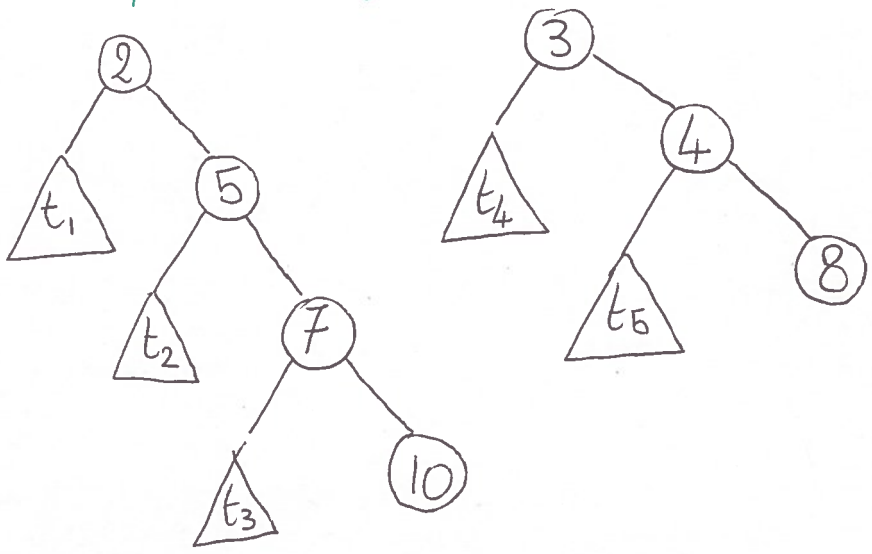


So a leftist heap is allowed to be unbalanced to the left.

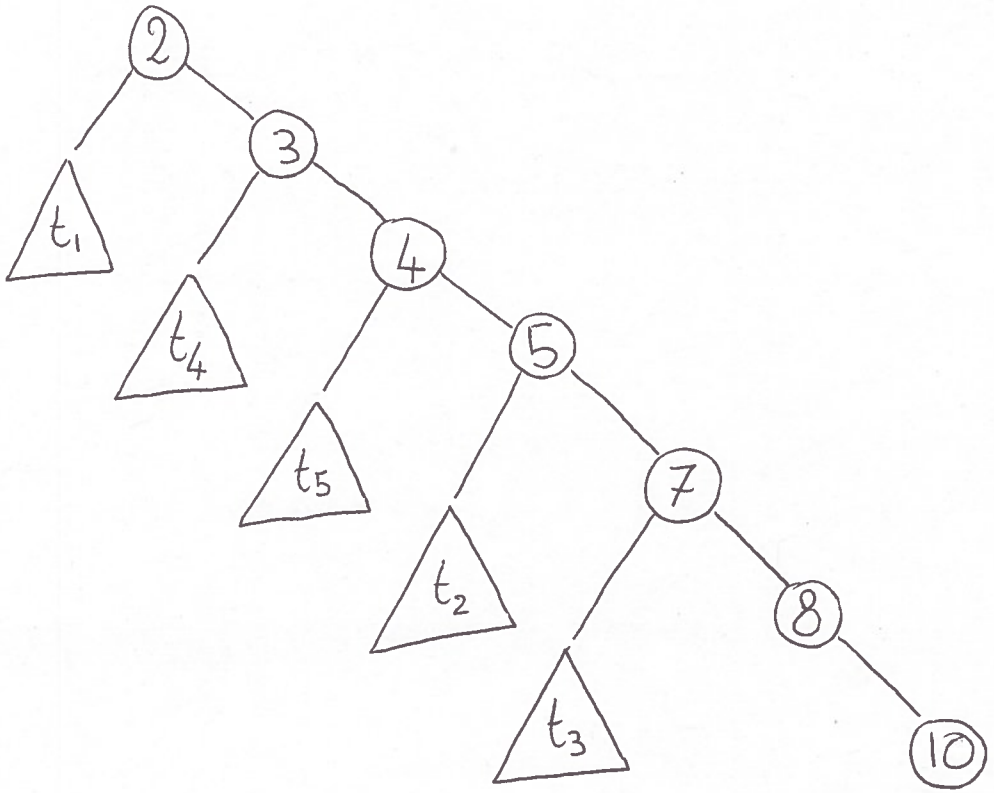
This makes it easy to implement union.

Idea:
- merge the two trees along the right spine (as in merge sort for list)
- then rearrange the children to preserve the leftist property:
  when the right child has higher rank than the left child, swap them

Example: Merge these two trees



Merge them along the right spines:



Then travel up the right spine and fix the violations of the leftist property by swapping children

We represent Leftists Heaps by keeping track of the rank in each node

```
data LHeap =
  Empty | Node ℕ Key LHeap LHeap
```

↑ a natural number giving the rank of the tree

The auxiliary function to "repair" the leftist property, constructs a Leftist tree from a key and the two children:

```
makeLH :: Key → LHeap → LHeap → LHeap
makeLH x h1 h2 =
  if (rank h1) ≥ (rank h2)
    then Node (rank h2 + 1) x h1 h2
    else Node (rank h1 + 1) x h2 h1
```

makeLH puts the child with the higher rank on the left, that with the lower rank on the right.

The new rank is one more than the rank of the right child.

Union of two Leftist Heaps:

merge them along the right spines

apply makeLH at each step:

```
union :: LHeap ⟶ LHeap ⟶ LHeap
union h, Empty = h1
union Empty h2 = h2
union h1@(Node r1 x1 h11 h12) h2@(Node r2 x2 h21 h22)
  = if x1 ≤ x2
    then makeLH x1 h11 (union h12 h2)
    else makeLH x2 h21 (union h1 h22)
```

What is the complexity of union?

makeLH has complexity $O(1)$ because it performs a fixed set of operations with no recursive calls

union makes one recursive call where

- one of the arguments remains the same
- the other is replaced by its right child.

So union does recursion along the right spine. The length of the right spine is the rank.

Observation:

Because of the leftist property, the rank is $O(\log n)$

(Exercise: prove this)

So the complexity of union is $O(\log n)$

(if n is a bound of the size of the two arguments)

Other operations can be defined in terms of union, so they also have complexity $O(\log n)$

```
insert x h = union (Node 1 x Empty Empty) h

extract (Node r x h1 h2) =
        (x, union h1 h2)
```