Chapter 23
Merging Priority Queues

The Skew Heap

- Skew heap – heap-ordered binary tree without a balancing condition.
- With these, there is no guarantee that the depth of the tree is logarithmic.
- It supports all operations in logarithmic amortized time.
- It is somewhat like a splay tree.

Merging

- Many operations with heap-ordered trees can be done using merging.
- Operations:
  - *Insert* – create a one-node tree containing *x* and merge that tree into the priority queue.
  - *Find minimum* – return the item at the root of the priority queue.
  - *Delete minimum* – delete the root and merge its left and right subtrees.
  - *Decrease the value of a node* – assume that *p* points to the node in the priority queue. Lower the value of *p*'s key. Detach *p* from its parent, which yields two priority queues. Merge the two resulting priority queues.
- Thus, we need only see how merging priority queues is implemented.

Simplistic Merging of Heap-Ordered Trees

- Assume we have two heap-ordered trees, H_1 and H_2, that need to be merged.
- If either tree is empty, the other tree is the merged tree.
- Otherwise, compare the roots.
- Recursively merge the tree with the larger root into the right subtree of the tree with the smaller root.
- See Figure 1.

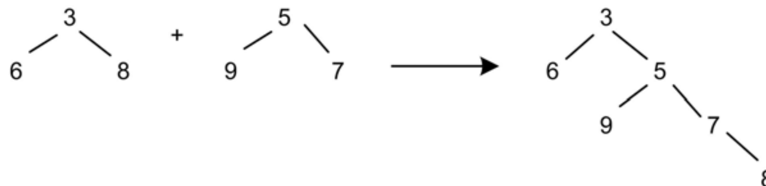

Figure 1 Simplistic merge of heap-ordered trees

- The practical effect of the above operation is in fact an ordered arrangement consisting only of a single right path.
- Thus the operations can be linear.

The Skew Heap – A Simple Modification

- We can make a simple modification to the merge operation and get better results.
- Prior to the completion of a merge, we swap the left and right children for every node in the resulting right path of the temporary tree.
- Consider the example in Figure 2.

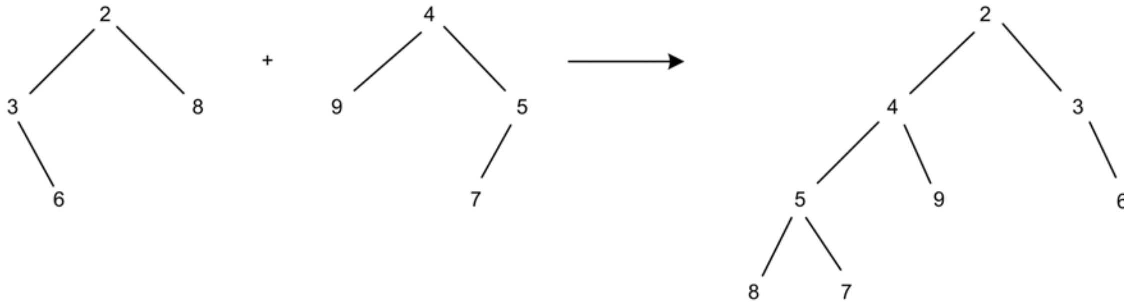

Figure 2 Merging a skew heap

- When a merge is performed in this way, the heap-ordered tree is also called a *skew heap*.
- Let's consider this operation from a recursive point of view. Let *L* be the tree with the smaller root and *R* be the other tree.
    1. If one tree is empty, the other is the merged result.
    2. Otherwise, let *Temp* be the right subtree of *L*.
    3. Make *L*'s left subtree its new right subtree.
    4. Make the result of the recursive merge of *Temp* and *R* the new left subtree of *L*.
- The result of child swapping is that the length of the right path will not be unduly large all the time.
- The amortized time needed to merge two skew heaps is $O(\log n)$.

Translated into code:

```
Node * SkewHeap::merge(Node * t1, Pair * t2)
{ Node *small, *big;
  if (t1==NULL) return t2;
  if (t2==NULL) return t1;
  if (t1->pri < t2->pri)
  {  small = t1; big = t2;
  }
  else
  {small = t2; big = t1;
  }
  Node * temp = small->right;
  small->right = small->left;
  small->left = merge(temp, big);
  return small;
}
```

The Pairing Heap

- The *pairing heap* is a structurally unconstrained heap-ordered M-ary tree for which all operations, except deletion, take constant worst-case time.
- Deletion could take linear worst-case time.
- Consider the pairing heap shown in Figure 3.

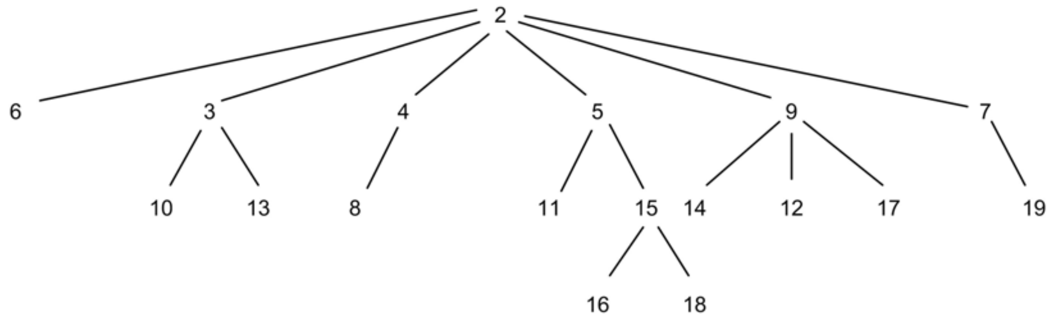

Figure 3 Abstract pairing heap

- The actual implementation, using a left child/right sibling representation is shown in Figure 4.

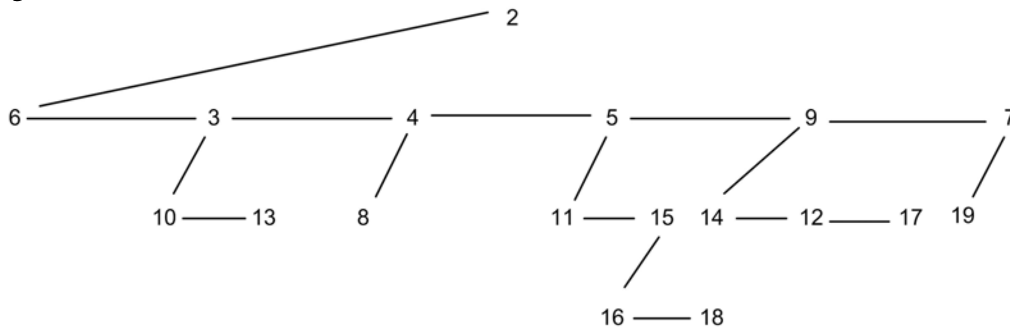

Figure 4 Actual representation of pairing heap

- Constant time operations on a pairing heap
  - Merging
    - Make the heap with the larger root the new first child of the heap with the smaller root
  - Insertion
    - A special case of merge
  - Decrease key
    - Decrease the value of the requested node
    - Detach the adjusted node from its parent and merge the two pairing heaps that result
- Deletion
  - Remove the root of the tree creating a collection of heaps
  - If there are c children of the root, combining these heaps requires $c-1$ merges
  - Consequently, this operation can take $\mathrm{O}(n)$ time.
  - The order of the merging is important.
  - A two-pass merge has been proposed
    - First scan – merges pairs of children from left to right

- Second scan – merge the rightmost tree that remains from the first scan with the current merged result.
- Suppose we have children c_1 through c_8.
    - The first pass merges c_1 and c_2, c_3 and c_4, c_5, and c_6, and c_7, and c_8.
    - The result is d_1, d_2, d_3, and d_4.
    - The second pass merges d_3 and d_4; d_2 is then merged with the result, and d_1 is then merged with the result of that merge.
- See Figure 5 after deleting 2.

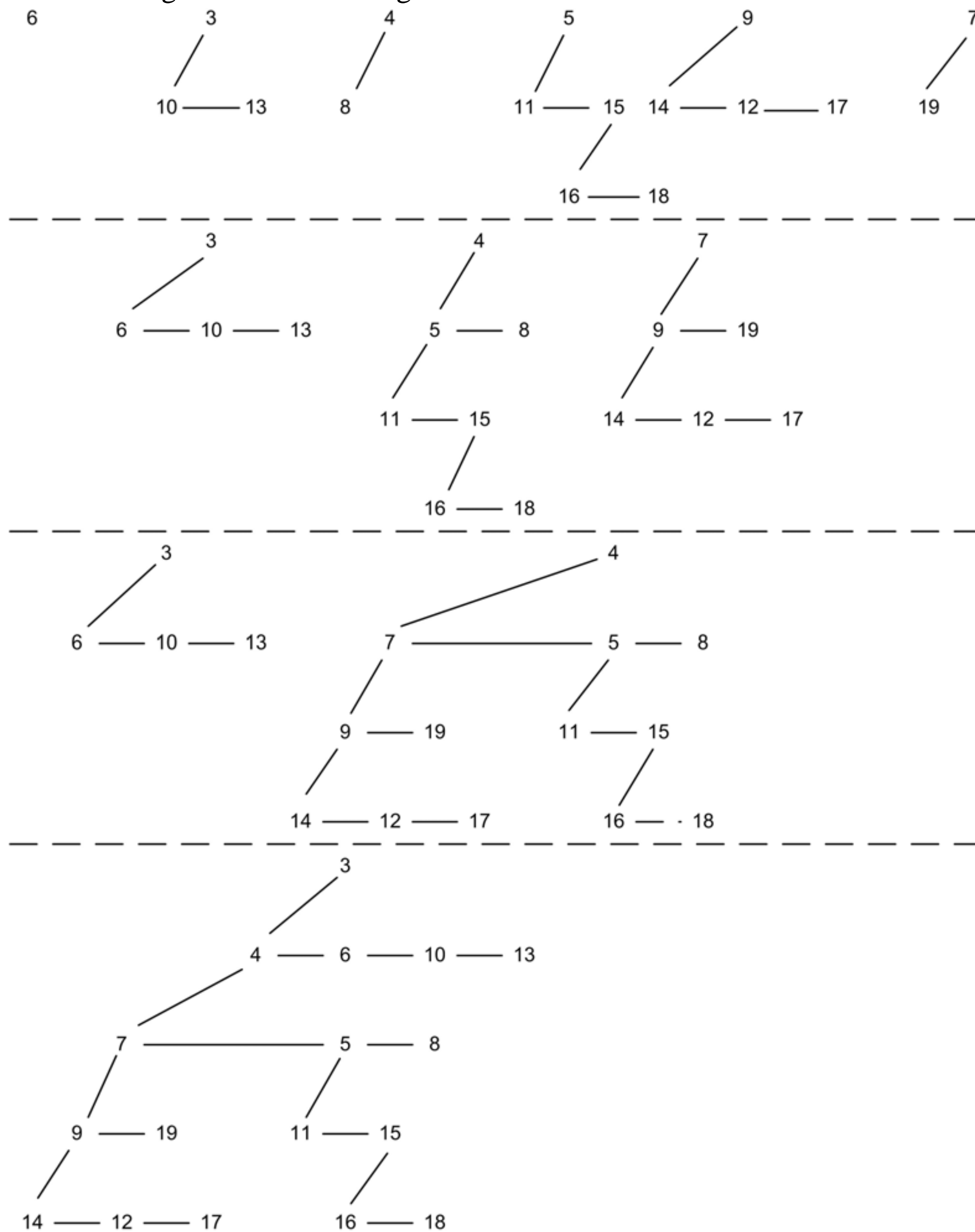

Figure 5 Merging pairing heaps