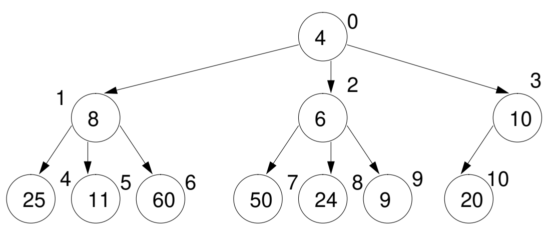
CSE 326
Lecture 25

Spring Quarter 1995

d-Heaps

- Use a complete d-ary tree.
- Each node has d children instead of just 2.
- Store the root at 0.
- The children of i are stored in di+1, di+2, ..., di+d.
- The parent of i is stored in (i-1)/d

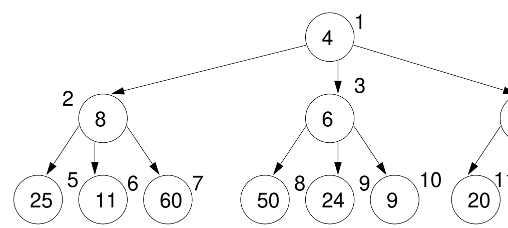
3-Heap



0	1	2	3	4	5	6	7	8	9	10	
4	8	6	10	25	11	60	50	24	9	20	

Alternative d-Heap Implementation

- Store the root at 1.
- The children of i are stored in di-1, di, di+1,..., di+d - 2.
- The parent of i is stored in (i+1)/d



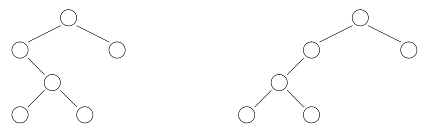
Leftist Trees

- A leftist tree is a binary tree with two properties.

  1. Heap property - key at a node is less than or equal to the keys of its children.

  2. Leftist property - For each node the null path length of the left subtree is greater than the null path length of the right subtree.
- Null path length - The null path length of a node (subtree) is the shortest path to a node without two children. A null has null path length = -1.

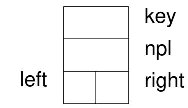
not a leftist tree   leftist tree



Data Structure for Leftist Trees

- Assume that a node of a leftist tree contains the following

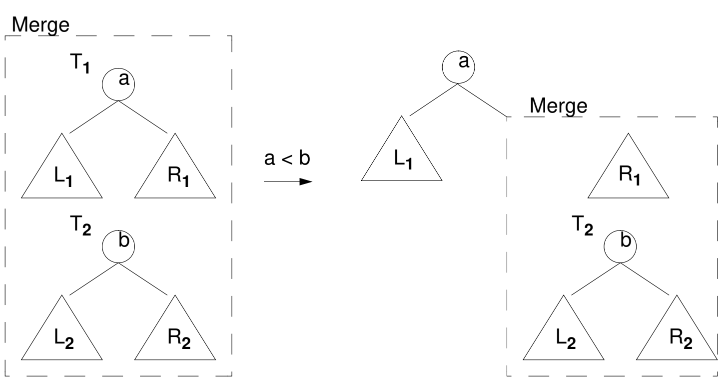
npl = null path length



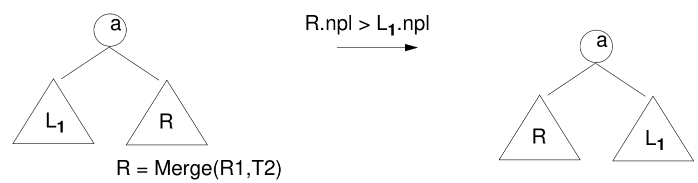
- The basic operation is the merging of two leftist trees. O(log n)
- Insert by merging a single node with a leftist tree. O(log n)
- Delete_min by merging the left and right subtrees of the root. O(log n)

Merging Two Leftist Trees

- Merge(T_1,T_2) returns a new leftist tree assuming T_1 and T_2 are two distinct leftist trees.



Merge Continued



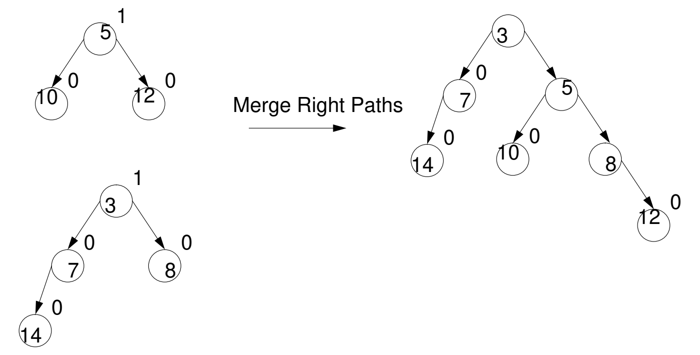
Iterative Leftist Tree Merging

- Two passes to merge T_1 and T_2.

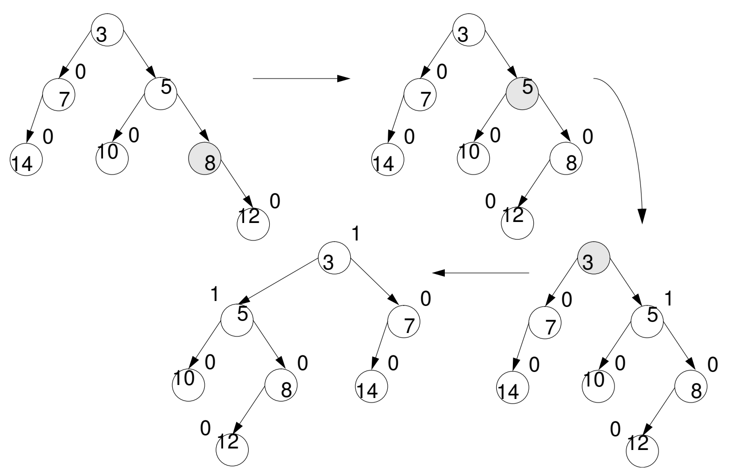
  Downward pass: merge the right paths of T_1 and T_2 forming a new tree T.

  Upward pass: Return along the right path of T computing the null path lengths and interchanging the left and right subtrees to maintain the leftist property.

Iterative Leftist Tree

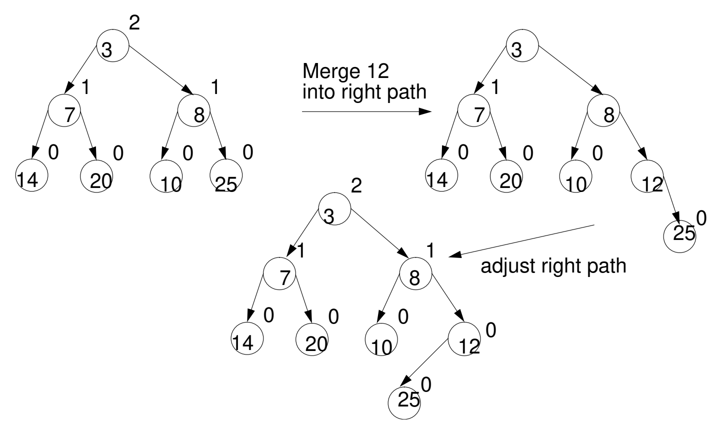


Upwards Pass

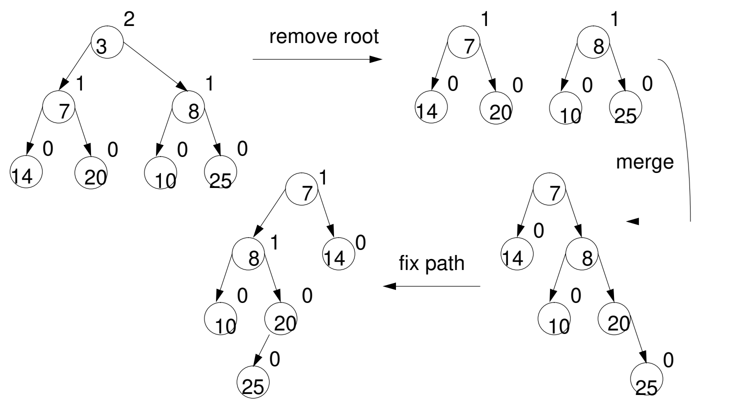


Insertion in leftist trees

- Insert 12



Delete_min

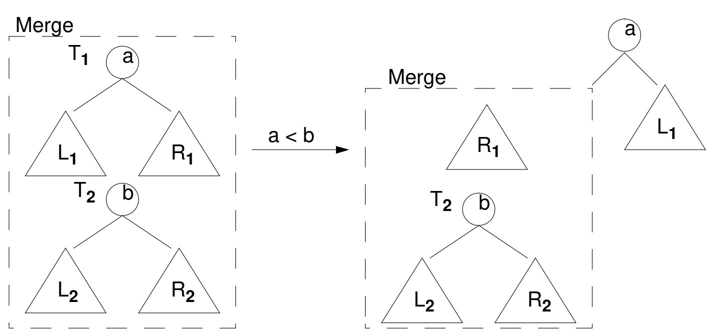


Skew Heaps

- Self adjusting version of leftist heaps (priority queues)
- Amortized time to merge, insertion, and delete_min is O(log n)
- Worst case time is O(n) for these operations.
- Merging

  Always switch children on merging except for the final step.

Skew Heap

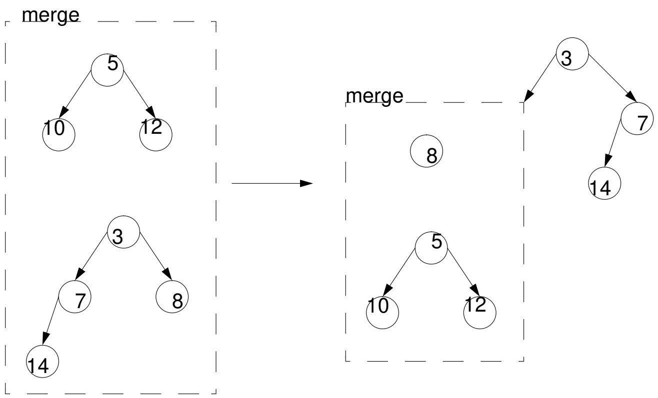


Skew Heap Code

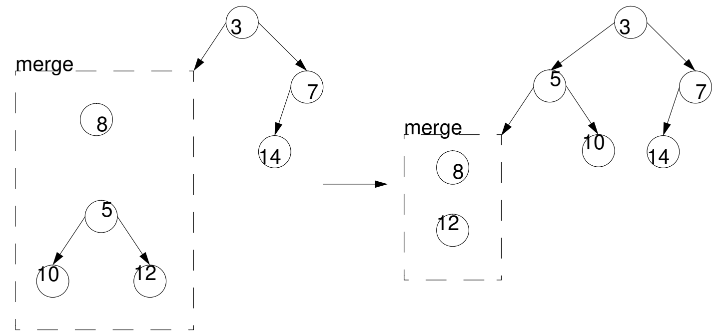
- merge(S,T) {

```
local pointer temp;
case
  S=null: return(T);
  T=null: return(S);
  S.key < T.key: {
      temp := S.left;
      S.left := merge(S.right,T);
      S.right := temp;
      return(S) }
  T.key ≤ S.key: {
      temp := T.left;
      T.left := merge(T.right,S);
      T.right := temp;
      return(T) }
endcase }
```

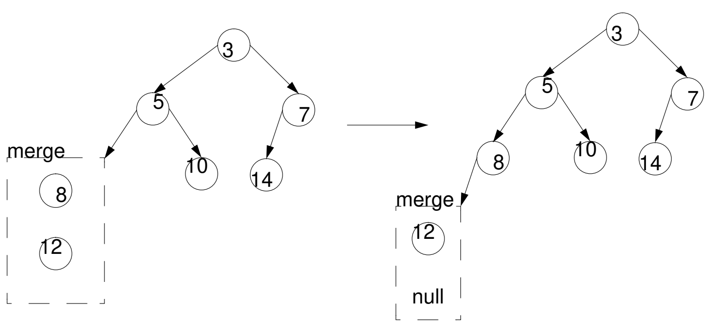
Skew Heap Example



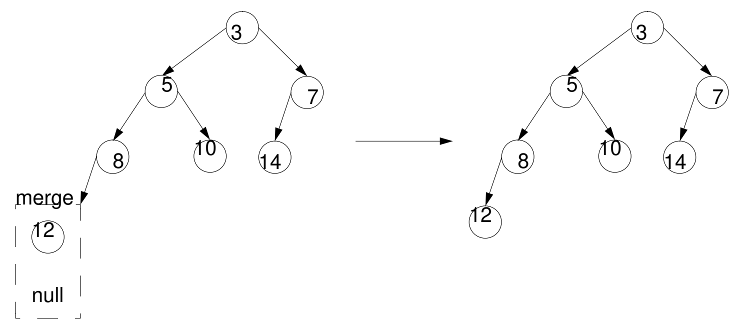
Example Continued



Example Continued



Example Continued



Summary of Priority Queues

- There are many - d-heaps, leftist trees, skew heaps, binomial queues, Fibonacci heaps, ...
- At a minimum they support insert and delete_min.
- Additional operations of increase, decrease, delete, merge can often be supported.
- Most of the data structures have worst case time O(log n) for these operations. An exception is skew heaps which take amortized time O(log n) time for merge insertion and deletion.
- For discrete event simulation and other similar problems the calendar queue can achieve average time O(1) for insert and delete_min.