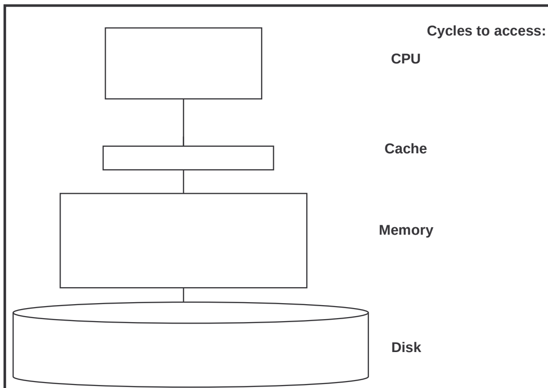
Priority Queues

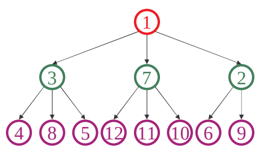
(Leftist Heaps & Skew Heaps)

Chapter 6 in Weiss



A Solution: *d*-Heaps

- Each node has *d* children
- Still representable by array
- Good choices for *d*:
  - (choose a power of two for efficiency)
  - fit one set of children in a cache line
  - fit one set of children on a memory page/disk block



12	1	3	7	2	4	8	5	12	11	10	6	9

Operations on *d*-Heap

- Insert : runtime =
- deleteMin: runtime =

Does this help insert or deleteMin more?

One More Operation

- Merge two heaps. Ideas?

New Operation: Merge

Given two heaps, merge them into one heap

- first attempt: insert each element of the smaller heap into the larger.

  runtime:

- second attempt: concatenate binary heaps' arrays and run buildHeap.

  runtime:

Leftist Heaps

Idea:

Focus all heap maintenance work in one small part of the heap

Leftist heaps:

1. Most nodes are on the left
2. All the merging work is done on the right

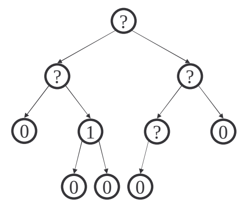
Definition: Null Path Length

null path length (npl) of a node *x* = the number of nodes between *x* and a null in its subtree

OR

npl(x) = min distance to a descendant with 0 or 1 children

- *npl*(null) = -1
- *npl*(leaf) = 0
- *npl*(single-child node) = 0



Equivalent definitions:

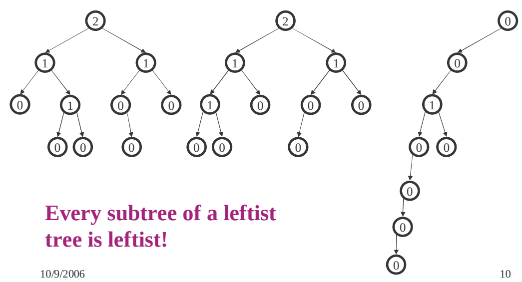
1. *npl(x)* is the height of largest complete subtree rooted at *x*
2. *npl(x)* = 1 + min{*npl*(left(x)), *npl*(right(x))}

Leftist Heap Properties

- Heap-order property
  - parent's priority value is ≤ to childrens' priority values
  - <u>result</u>: minimum element is at the root
- Leftist property
  - For every node *x*, *npl*(left(*x*)) ≥ *npl*(right(*x*))
  - <u>result</u>: tree is at least as "heavy" on the left as the right

Are leftist trees…
complete?
balanced?

Are These Leftist?



Every subtree of a leftist tree is leftist!

Right Path in a Leftist Tree is Short (#1)

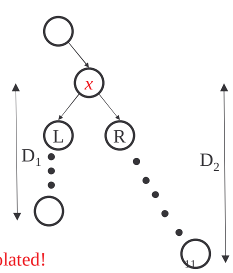
<u>Claim</u>: The right path is as short as *any* in the tree.

<u>Proof</u>: (By contradiction)

Pick a shorter path: $D_1 < D_2$
Say it diverges from right path at *x*

npl(L) ≤ D_1-1 because of the path of length D_1-1 to null

npl(R) ≥ D_2-1 because every node on right path is leftist



Leftist property at *x* violated!

Right Path in a Leftist Tree is Short (#2)

<u>Claim</u>: If the right path has `r` nodes, then the tree has at least `2^r-1` nodes.

<u>Proof</u>: (By induction)

Base case : `r=1`. Tree has at least `2^1-1 = 1` node

Inductive step : assume true for `r' < r`. Prove for tree with right path at least `r`.

1. Right subtree: right path of `r-1` nodes
 ⇒ `2^(r-1)-1` right subtree nodes (by induction)
2. Left subtree: also right path of length at least `r-1` (by previous slide)
 ⇒ `2^(r-1)-1` left subtree nodes (by induction)

Total tree size: $(2^{r-1}-1) + (2^{r-1}-1) + 1 = 2^r-1$

Why do we have the leftist property?

Because it guarantees that:

- the *right path is really short* compared to the number of nodes in the tree
- A leftist tree of N nodes, has a right path of at most log (N+1) nodes

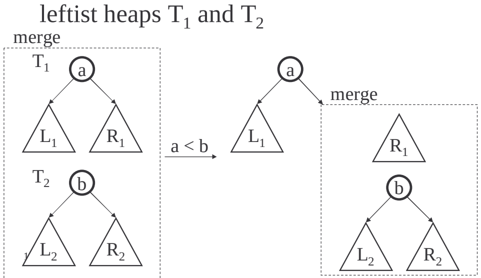
Idea – perform all work on the right path

Merge two heaps (basic idea)

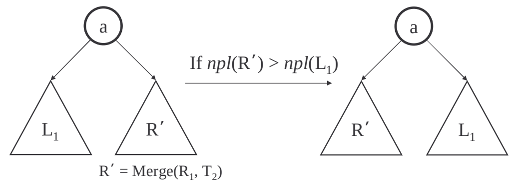
- Put the smaller root as the new root,
- Hang its left subtree on the left.
- Recursively merge its right subtree and the other tree.

Merging Two Leftist Heaps

- merge(T_1,T_2) returns one leftist heap containing all elements of the two (distinct) leftist heaps T_1 and T_2



Merge Continued



runtime:

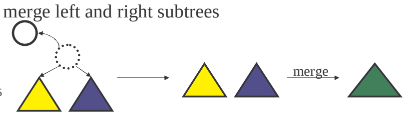
Let's do an example, but first… Other Heap Operations

- insert ?
- deleteMin ?

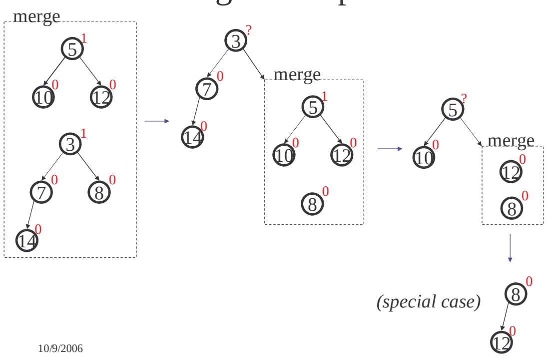
Operations on Leftist Heaps

- merge with two trees of total size n: O(log n)
- insert with heap size n: O(log n)
  - pretend node is a size 1 leftist heap
  - insert by merging original heap with one node heap

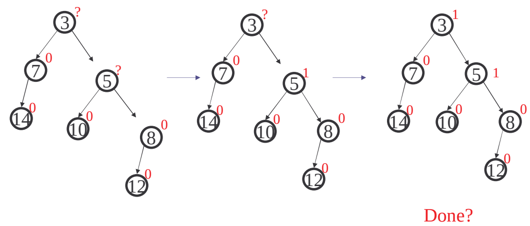
- deleteMin with heap size n: O(log n)
  - remove and return root
  - merge left and right subtrees



Merge Example



Sewing Up the Example



Finally…