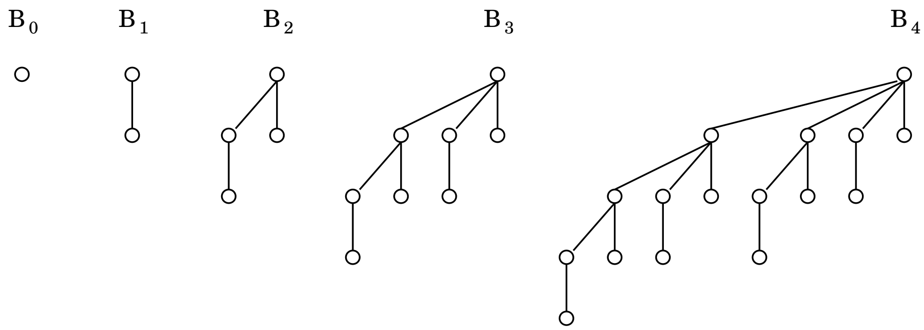
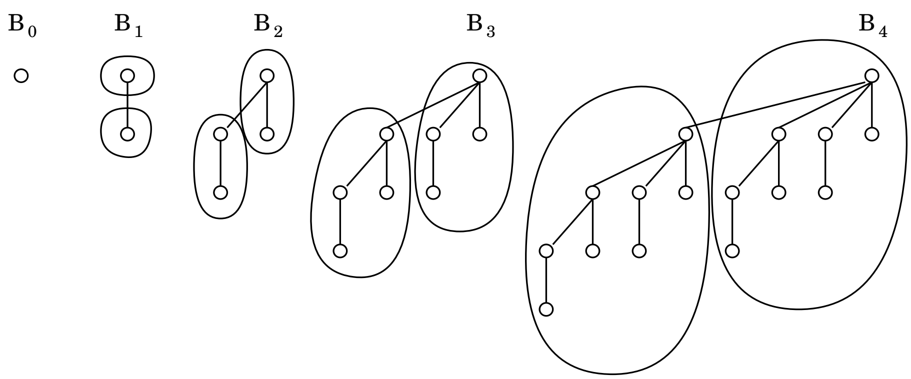
13-0: **Binomial Trees**

- B_0 is a tree containing a single node
- To build B_k:
    - Start with B_{k-1}
    - Add B_{k-1} as left subtree

13-1: **Binomial Trees**



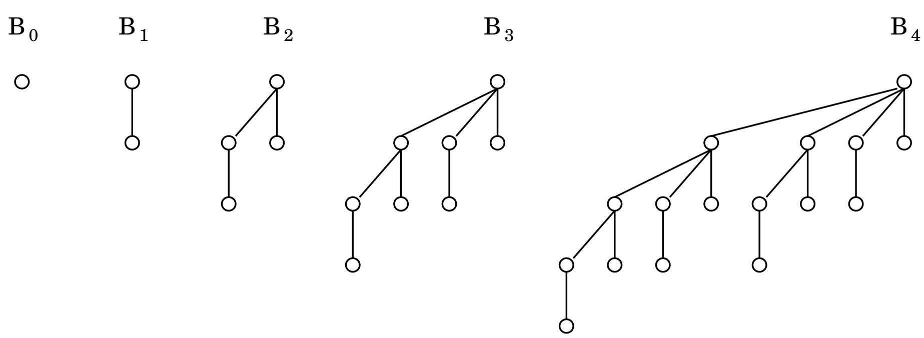
13-2: **Binomial Trees**



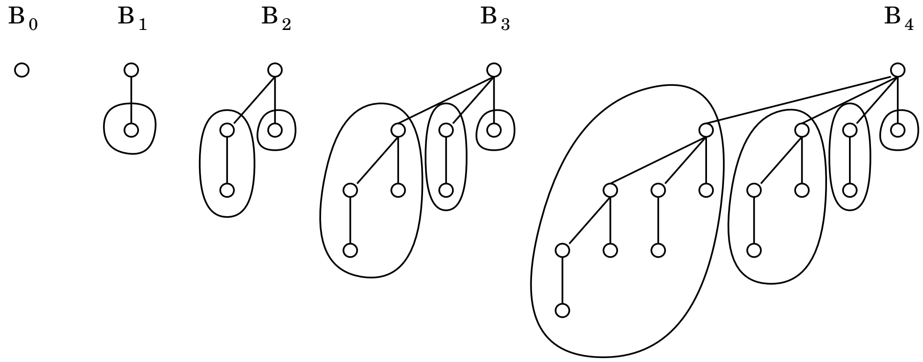
13-3: **Binomial Trees**

- Equivalent defintion
    - B_0 is a binomial heap with a single node
    - B_k is a binomial heap with k children:
        - $B_0 \ldots B_{k-1}$

13-4: **Binomial Trees**



13-5: **Binomial Trees**



13-6: **Binomial Trees**

- Properties of binomial trees B_k
  - Contains 2^k nodes
  - Has height k
  - Contains $\binom{k}{i}$ nodes at depth i for $i = 0 \ldots k$

13-7: **Binomial Trees**

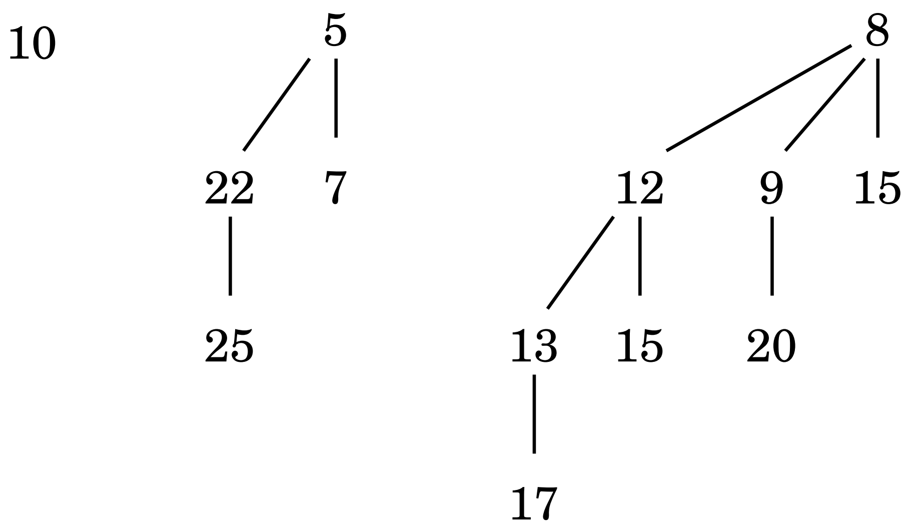
- B_k contains $\binom{k}{i}$ nodes at depth i
  - $D(k, i)$ # of nodes at depth i in B_k
  - $D(k, i) = D(k-1, i) + D(k-1, i-1)$ (why?)

$$\begin{aligned} D(k,i) &= D(k-1,i) + D(k-1,i-1) \\ &= \binom{k-1}{i} + \binom{k-1}{i-1} \\ &= \binom{k}{i} \end{aligned}$$

13-8: **Binomial Heaps**

- A Binomial Heap is:
  - Set of binomial trees, each of which has the heap property
    - Each node in every tree is <= all of its children
  - All trees in the set have a different root degree
    - Can't have two B_3's, for instance

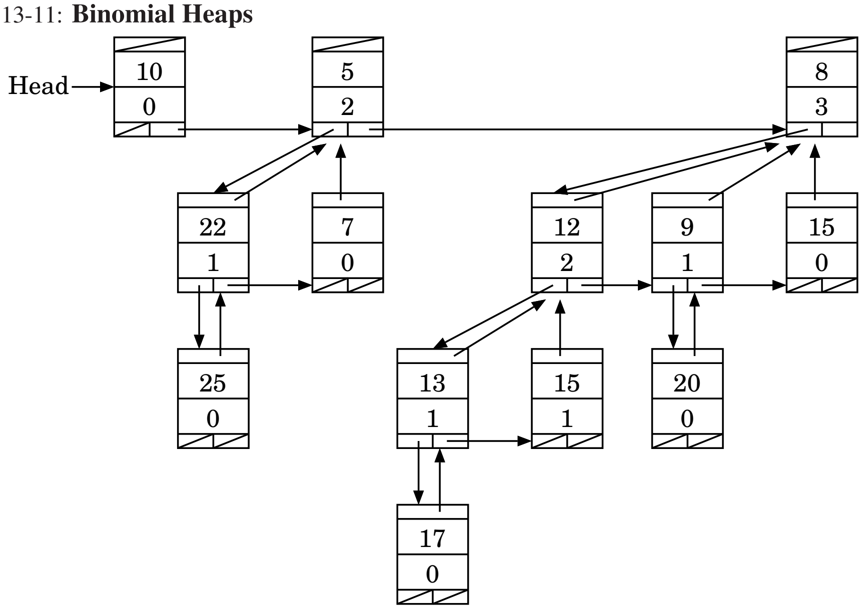
13-9: **Binomial Heaps**



13-10: **Binomial Heaps**

- Representing Binomial Heaps
  - Each node contains:
    - left child, right sibling, parent pointers
    - degreee (is the tree rooted at this node B_0, B_1, etc.)
    - data
  - Each list of children sorted by degree

13-11: **Binomial Heaps**



13-12: **Binomial Heaps**

- How can we find the minimum element in a binomial heap?
- How long does it take?

13-13: **Binomial Heaps**

- How can we find the minimum element in a binomial heap?
    - Look at the root of each tree in the list, find smallest value
- How long does it take?
    - Heap has n elements
    - Represent n as a binary number
    - B_k is in heap iff kth binary digit of n is 1
    - Number of trees in heap $\in O(\lg n)$

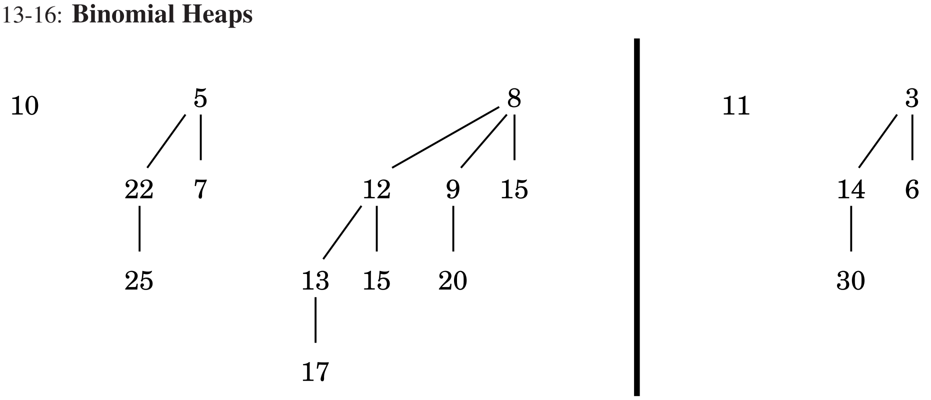
13-14: **Binomial Heaps**

- Merging Heaps H_1 and H_2
    - Merge root lists of H_1 and H_2
    - What property of binomial heaps may be broken?
    - How do we fix it?

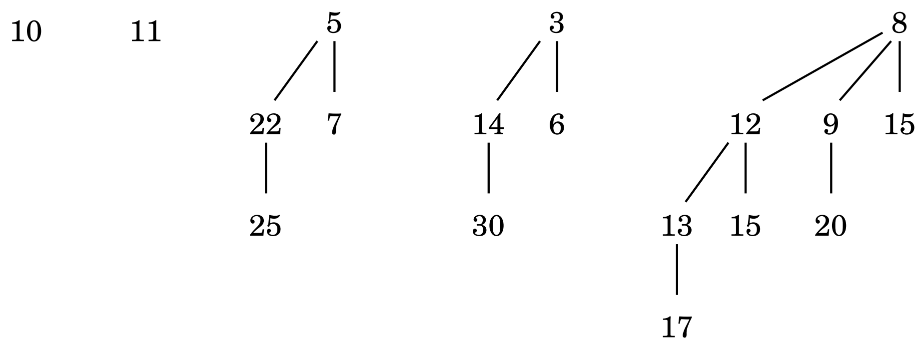
13-15: **Binomial Heaps**

- Merging Heaps H_1 and H_2
    - Merge root lists of H_1 and H_2
        - Could now have two trees with same degree
    - Go through list from smallest degree to largest degree
        - If two trees have same degree, combine them into one tree of larger degree
        - If three trees have same degree (how can this happen?) leave one, combine other two into tree of larger degree

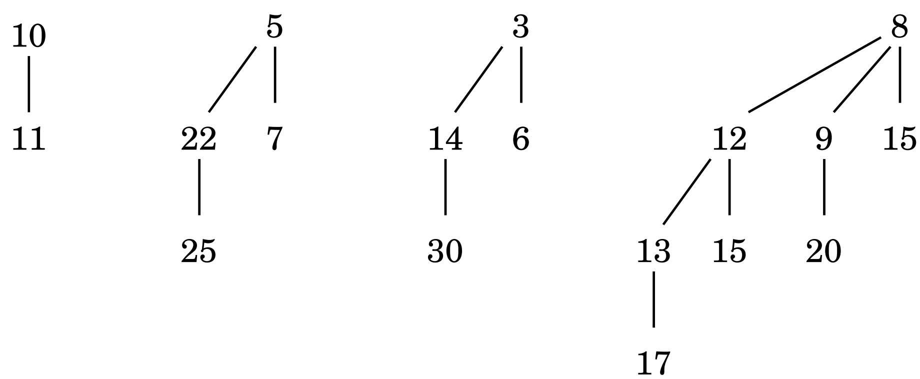
13-16: **Binomial Heaps**



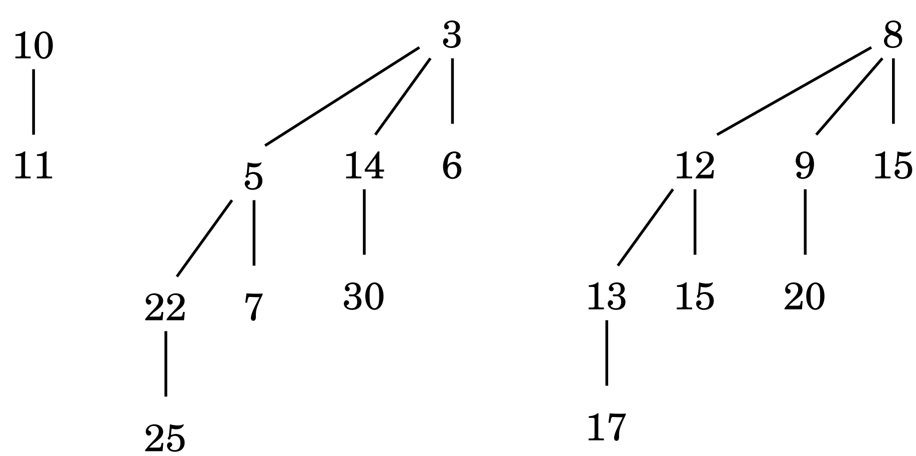
13-17: **Binomial Heaps**



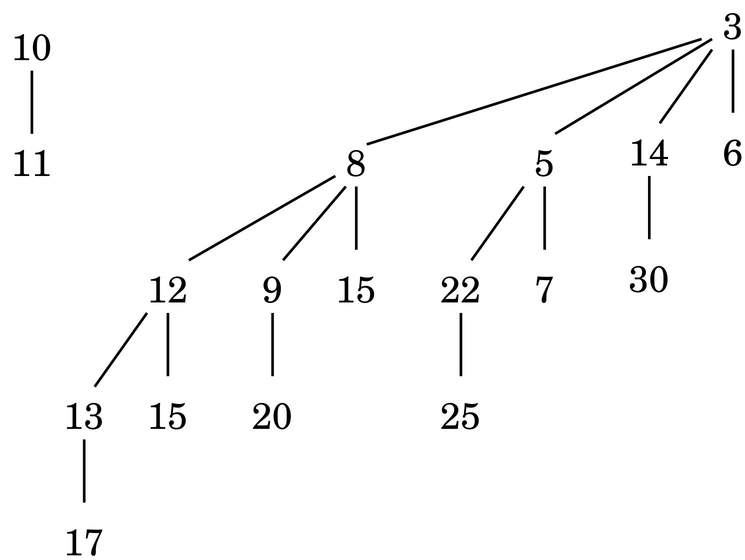
13-18: **Binomial Heaps**



13-19: **Binomial Heaps**



13-20: **Binomial Heaps**



13-21: **Binomial Heaps**

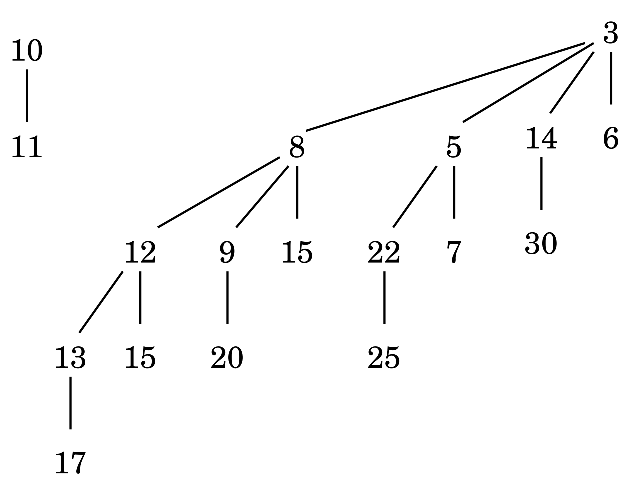
- Removing minimum element
  - How can we remove the minimum element
  - *HINT:* Be lazy – use operations that we already have

13-22: **Binomial Heaps**

- Removing minimum element
  - Find tree T that has minimum value at root, remove T from the list
  - Remove the root of T
    - Leaving a list of smaller trees
  - Reverse list of smaller trees
  - Merge two lists of trees together

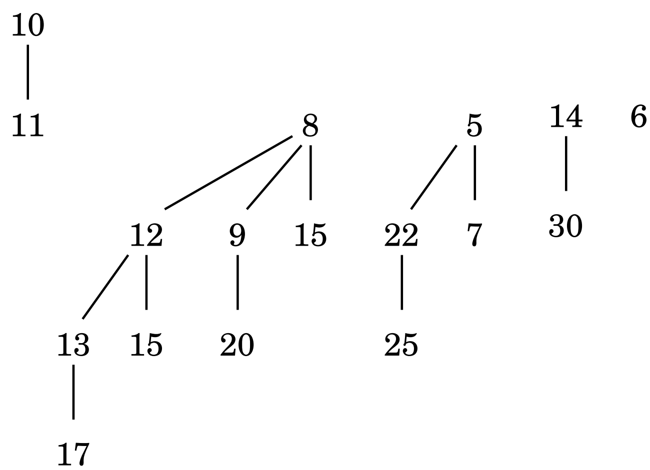
13-23: **Binomial Heaps**

- Removing minimum element



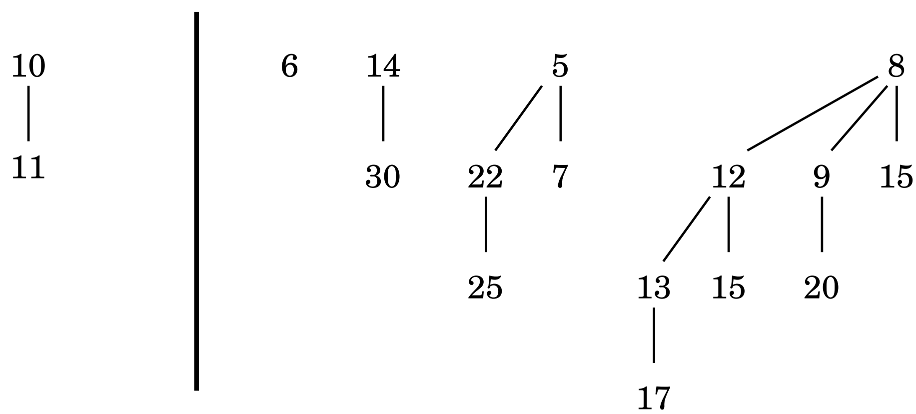
13-24: **Binomial Heaps**

- Removing minimum element



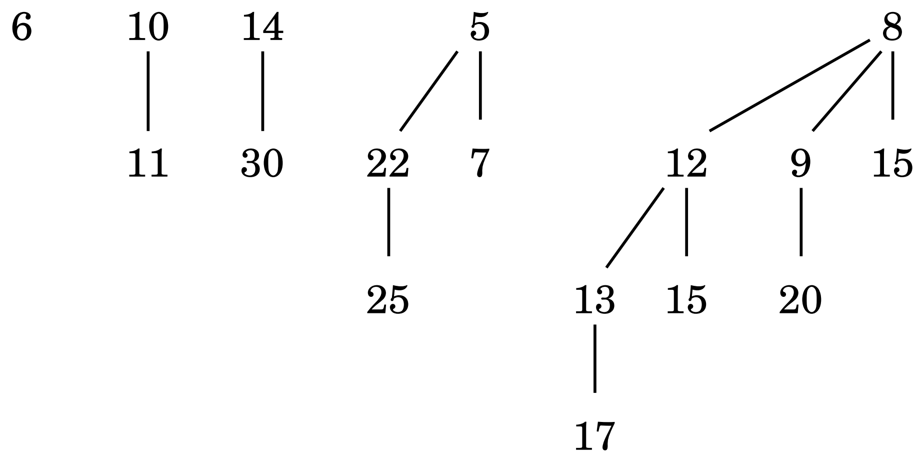
13-25: **Binomial Heaps**

- Removing minimum element



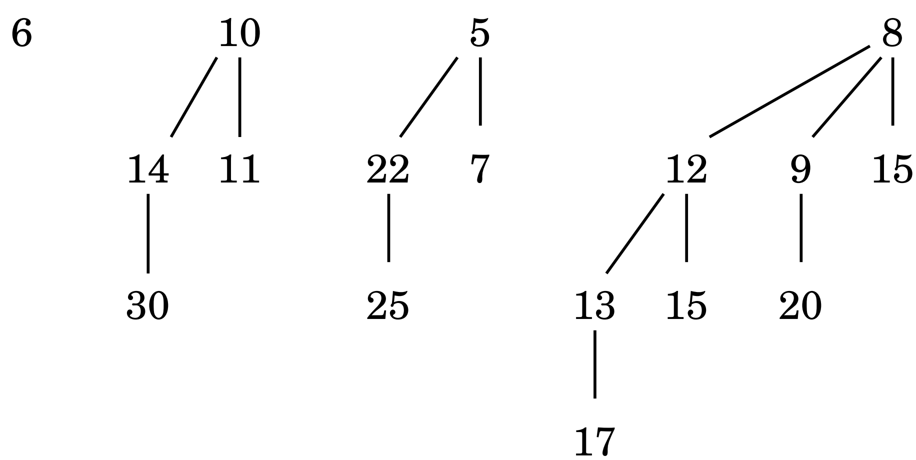
13-26: **Binomial Heaps**

- Removing minimum element



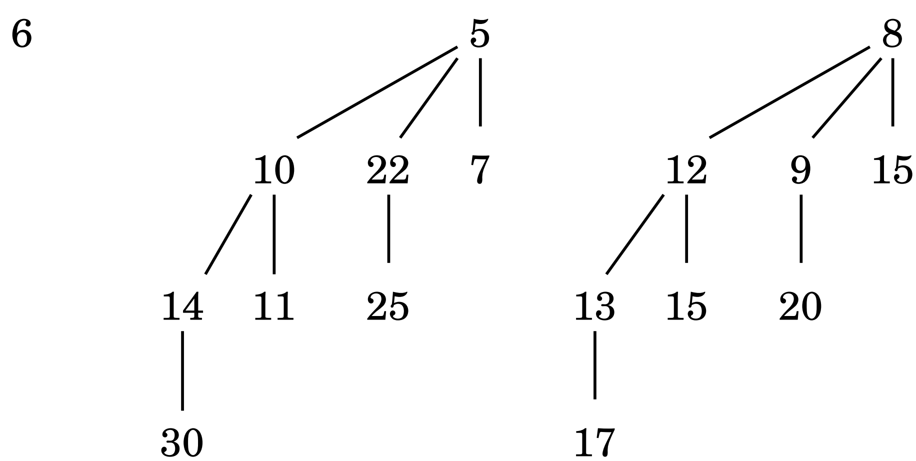
13-27: **Binomial Heaps**

- Removing minimum element



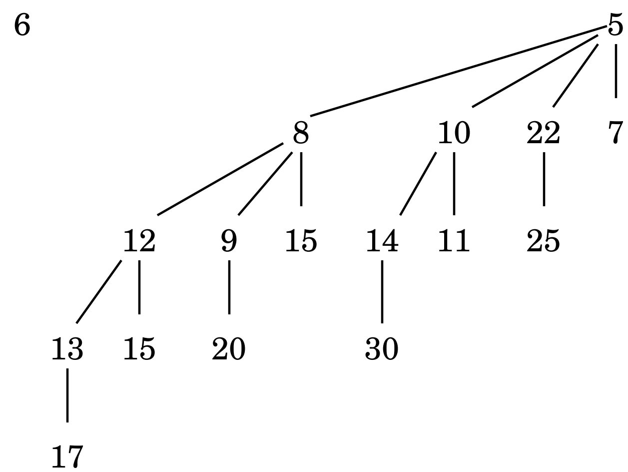
13-28: **Binomial Heaps**

- Removing minimum element



13-29: **Binomial Heaps**

- Removing minimum element



13-30: **Binomial Heaps**

- Removing minimum element
  - Time?

13-31: **Binomial Heaps**

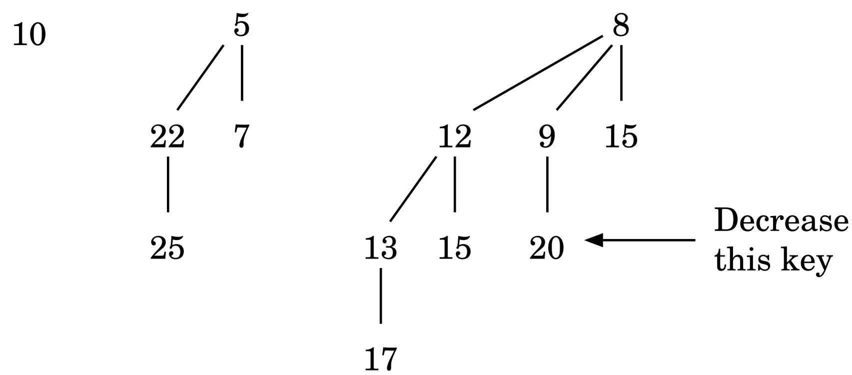
- Removing minimum element
  - Time?
    - Find the smallest element:
    - Reverse list of children
    - Merge heaps

13-32: **Binomial Heaps**

- Removing minimum element
  - Time?
    - Find the smallest element: $O(\lg n)$
    - Reverse list of children $O(\lg n)$
    - Merge heaps $O(\lg n)$

13-33: **Binomial Heaps**

- Decreasing the key of an element (assuming you have a pointer to it)

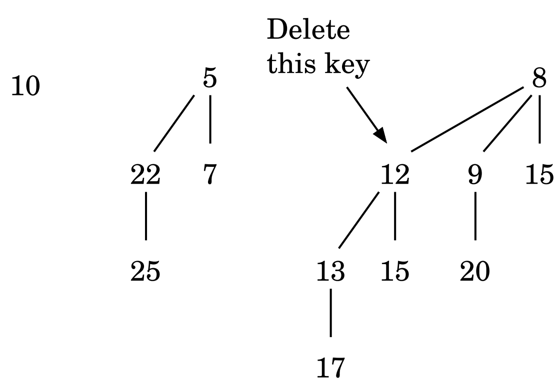


13-34: **Binomial Heaps**

- Decreasing the key of an element (assuming you have a pointer to it)
  - Decrease key value
  - While value < parent, swap with parent
    - Exactly like standard, binary heaps
- Time: $O(\lg n)$

13-35: **Binomial Heaps**

- How could we delete an arbitrary element (assuming we had a pointer to this element)?



13-36: **Binomial Heaps**

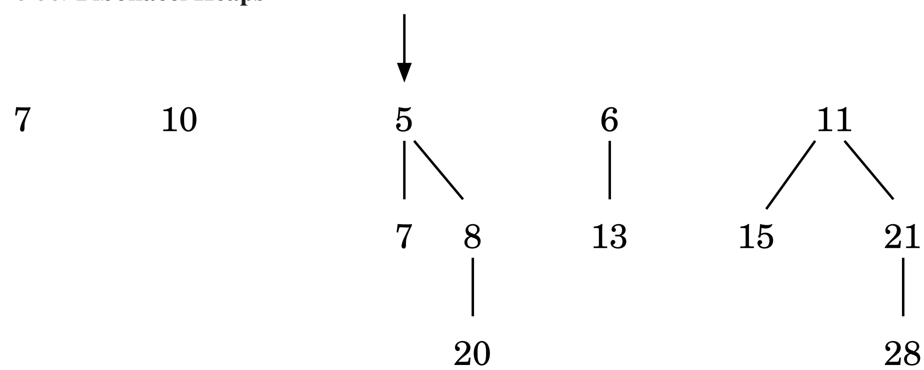
- How could we delete an arbitrary element (assuming we had a pointer to this element)?
  - Decrease key to $-\infty$, Time $O(\lg n)$
  - Remove smallest, Time $O(\lg n)$

13-37: **Fibonacci Heaps**

- A Fibonacci Heap, like a Binomial Heap, is a collection of min-heap ordered trees
  - No restriction on the # of trees of the same size
  - (We'll relax some of the other restrictions later ...)

- Maintain a pointer to tree with smallest root

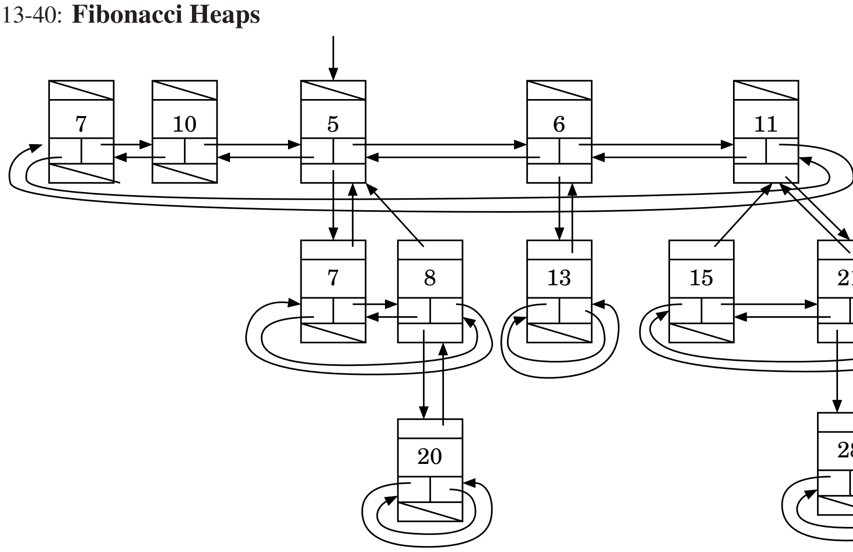
13-38: **Fibonacci Heaps**



13-39: **Fibonacci Heaps**

- Implementation
  - Each node has pointer to parent
  - Children are stored in circular linked list
    - No ordering among the children
  - Maintain a pointer to the tree with the smallest root

13-40: **Fibonacci Heaps**



13-41: **Fibonacci Heaps**

- We will use amortized analysis, using the potential method, to analyze Fibonacci heaps
- $\Phi = c * t(H)$
  - $t(H)$ = # of trees in the heap
  - (We will modify this Φ in a bit ...)

13-42: **Fibonacci Heaps -Min**

- Finding the minimum element

13-43: **Fibonacci Heaps - Min**

- Finding the minimum element
    - Look at the element pointed to by minimum pointer
        - Potential not changed
        - Takes time $O(1)$

13-44: **Fibonacci Heaps - Merge**

- Merging two heaps H_1 and H_2
    - Combine their root lists into one list
        - Takes a constant # of pointer changes (example on board)
    - Set minimum pointer
    - Change in potential:

$$\begin{aligned} \Phi(H) - (\Phi(H_1) + \Phi(H_2)) &= t(H) - (t(H_1) + t(H_2)) \\ &= 0 \end{aligned}$$

13-45: **Fibonacci Heaps - Delete Min**

- To delete the minimum node:
    - Remove smallest node
    - Add its children to root list
    - Consolidate root list
        - Link together nodes of the same degree until there is at most one node of each degree
        - Make it back into a Binomial Heap
        - Common practice when you only care about amortized running time – put off work, and do it all at once

13-46: **Fibonacci Heaps - Delete Min**

```
Consolidate
    Create an array A[], initially empty
        // Eventually, A[i] will hold tree of degree i
    For each node w in the root list
        x ← w
        d ←degree(x)
        while A[d]! =nil do
            y ← A[d]
            x ← link(x, y)
            A[d] ← nil
            d ← d + 1
        A[d] ← x
    Link elements of A together as new root list
    Recalculate min
```

13-47: **Fibonacci Heaps - Delete Min**

- Amortized cost to remove min:

$$\begin{aligned} am(c_{rem-min}) &= c_{rem-min} + \Phi(H_{new}) - \Phi(H_{old}) \\ &= (C_1 * t + c_2 * max\_deg) + c * max\_deg - c * t \\ &\in O(max\_deg) \end{aligned}$$

- $max\_deg \in O(\lg n)$
- $am(c_{rem-min}) \in O(\lg n)$

13-48: **Fib. Heaps - Decrease Key**

- Like to implement decrease key in amortized time $O(1)$
  - Add a new "Mark" field to each node in the tree
    - Mark is true if node has lost a child since parent pointer changed
  - New Potential function $\Phi(H) = t(H) + 2 * m(H)$
    - (extra constant c left out for clarity)

13-49: **Fib. Heaps - Decrease Key**

- With new potential function, merge and find still have amortized running time $O(1)$, and remove-min still has amortized running time $O(\lg n)$
  - Since none of those operations increase $m(H)$
- We can use the marks to make decrease-key work in time $O(1)$

13-50: **Fib. Heaps - Decrease Key**

- Decreasing a key can break the heap property
- Cut: Move Decreased node to root list
  - Now the heap property still holds
- Cascading cut:
  - If parent is not marked, mark parent
  - If parent is marked, cut parent, Cascading cut parent
- Examples (on board)

13-51: **Fib. Heaps - Decrease Key**

- Amortized cost for Decrease Key:
  - Actual Cost + Change in potential
  - Actual Cost:
    - $O(1)$ to move element to root list
    - # of cascading cuts c

- Change in Potential
  - # of added trees - 2 * # of nodes unmarked
  - 4 - c
- Amortized cost: $O(1) + c + 4 - c \in O(1)$

13-52: **Fib. Heaps - Decrease Key**

- Fibonacci heaps are no longer binomial heaps
- Analysis of Extract-min used the fact that they are binomial heaps to show that maximum degree of any node $\in O(\lg n)$
- Even with cuts/cascading cuts, maximum degree of any node is still $\in O(\lg n)$
  - See textbook, section 20.4 for details
- Previous analysis still correct