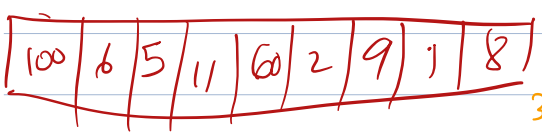
429 Binomial Heaps 10.4.

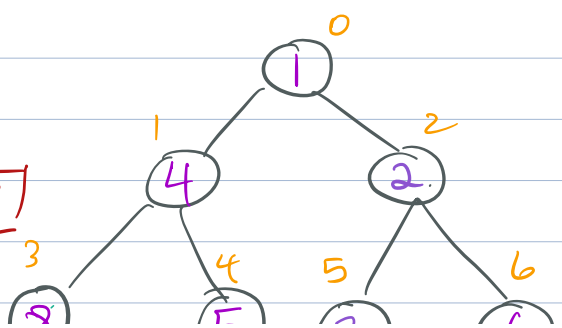
We are interested in Priority Queue ADT and will explore other options for their implementation.

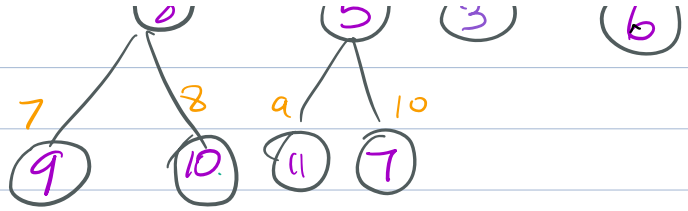
Procedure	Binary Heap: WC	Fibonacci Heap: Amortized	
MakeHeap	$\Theta(1)$	$\Theta(1)$	
Insert	$\Theta(\lg n)$	$\Theta(1)$	
Min	$\Theta(1)$	$\Theta(1)$	
ExtractMin	$\Theta(\lg n)$	$\Theta(\lg n)$	
Union	$\Theta(n)$	$\Theta(1)$	
DecreaseKey	$\Theta(\lg n)$	$\Theta(1)$	
Delete	$\Theta(\lg n)$	$\Theta(\lg n)$	

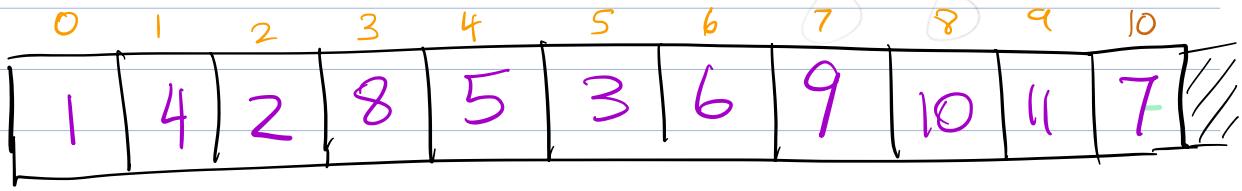
Recall: Binary Heap was studied in 260

- keep a complete binary tree in heap order



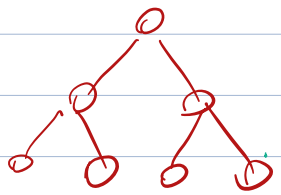






$left(i) = 2i + 1$

$right(i) = 2i + 2$



Binary Heaps are great, but they are not easily **Merged**

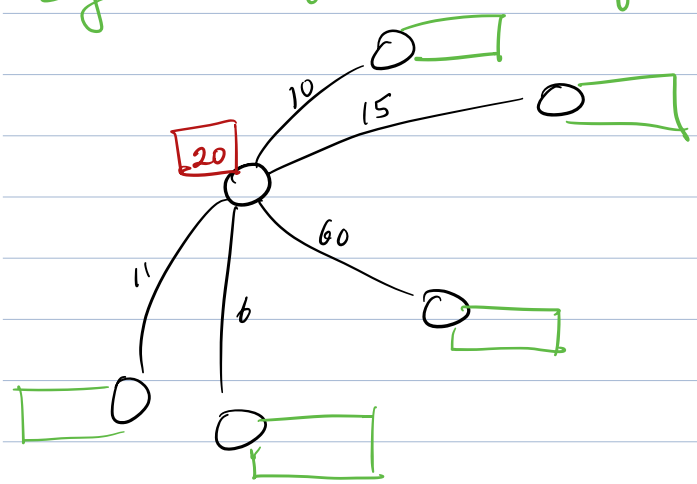
Many applications of Priority Queues require ability to take two PQs and merge them into one.

For the next few lectures, we will focus on this topic ... **Mergeable Heaps.**

Cormen, Leiserson, Rivest + Stein, 3rd Ed, Ch 19
(Fibonacci Heaps)

For applications such as in many graph algorithm (e.g. Single-Source Shortest Paths (Dijkstra's) the number of Decrease Key operations may far outnumber the ExtractMin operations.

Eg in Dijkstra's Alg



A dense graph may have O(n) edges to update (DecreaseKey's) for each of the n-1 "permanent" labels determined.

Fibonacci Heaps useful for

- Shortest Paths
- Min Spanning Tree

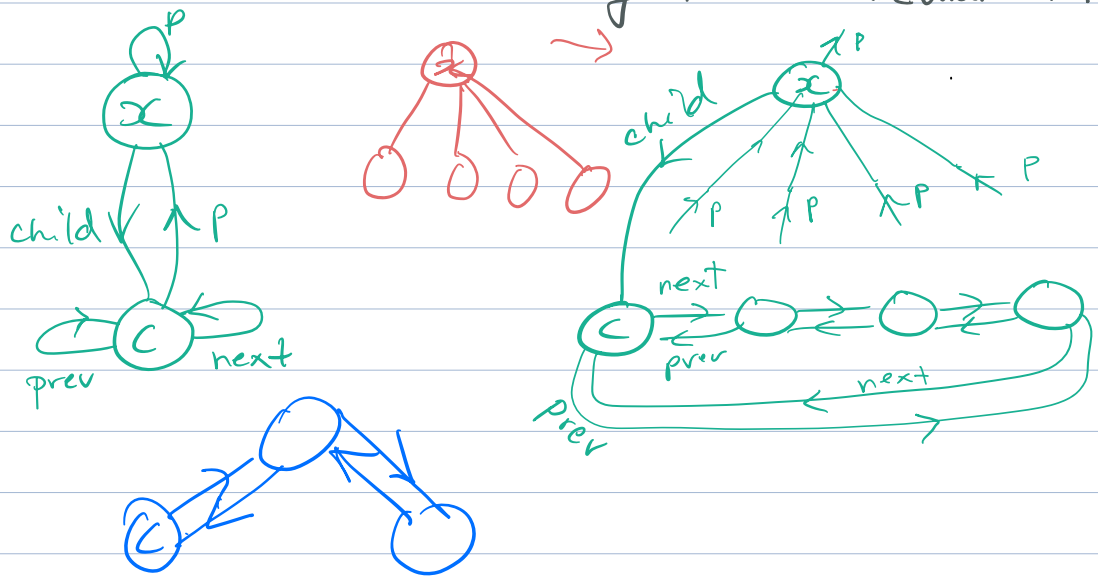
Would like simpler DS with same bounds, though.

Fibonacci Heap

- collection of rooted trees
  that are min-heap ordered.
  (key at node ≤ keys in left- or right- subtree)

- each node has
  - pointer to parent $x \to p$
  - pointer to child $x \to$ child

The children are in a doubly-linked circular list.



Circular doubly-linked child lists can be
inserted into in time $\Theta(1)$
combined in time $\Theta(1)$

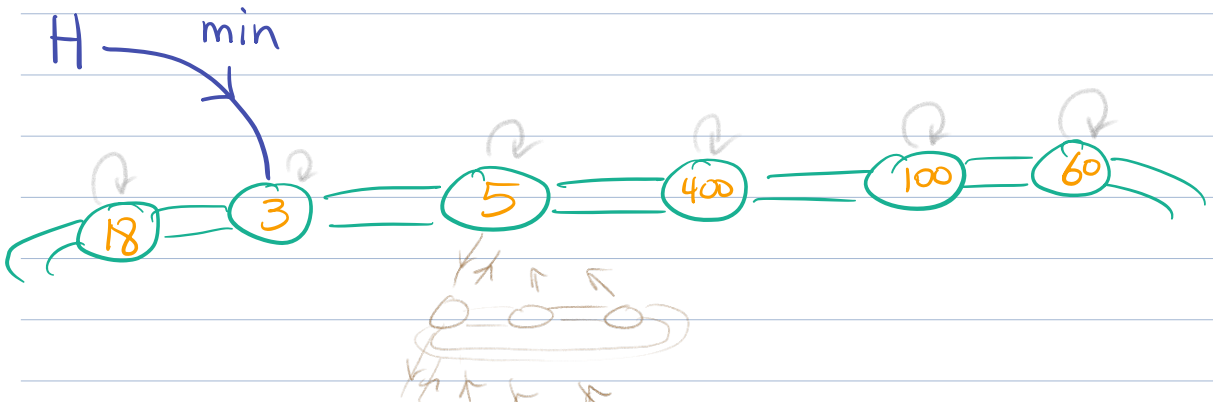
A Fibonacci Heap is given as a pointer to a root list.

- the root pointed to has the min key value of all the roots in the root list.
- Furthermore, all the trees in the root list are trees formed by Fib-merges (see below)
- the root list at the top of a Fib heap is just like the child list of a node, except

$x \to p = x$ $\forall$ roots x in root list

$x \to$ degree = # of children in x's child list

$x \to$ mark = "x has lost a child since x was made a child of another node"



for Fib Heap H: $H \to n$ = # nodes in H.

$t(H)$ = # roots in H's root list

$m(H)$ = # marked nodes in H

We will use the Potential Function method for amortized analysis

$$\Phi(H) = t(H) + 2m(H)$$

$\Delta\Phi$ = net potential changes to collection of heaps.

- a unit of potential can pay for a constant amount of work.

Max degree in our n-node Fib Heap will be denoted $D(n)$

When only the mergeable heap ops are supported, (no DecreaseKey or Delete)

$D(n) \leq \lfloor \lg n \rfloor$

When DecreaseKey and Delete are also used,

$D(n) \in O(\lg n)$

Fib Heap Ops

FibHeap.MaKe

$H \to n = 0$

$H \to min = NULL$

$\Delta \Phi(H) = 0.$

Cost $= \Theta(1)$

AmCost $= \Theta(1).$

FibHeap.Insert(x)

// x→key is already filled in

$x \to degree = 0$

$x \to p = NULL$

$x \to child = NULL$

$x \to marked = FALSE$

if $H \to min == NULL$

create a root list that just contains x

$H \to min = x$

Cost $= \Theta(1)$

$\Delta \Phi = 1$

$= \Theta(1).$

else

insert x into H's root list

if $x \to key < H \to min \to key$ $H \to min = x$.

$H \to n = H \to n + 1$

FibHeap.Min

return H→min

Cost = 1

ΔΦ = 0

O(1) amortized.

FibHeap.Union(H_1, H_2)

// H_1 and H_2 will not be usable in future

// H will be their union

H = FibHeap.Make()

H→min = H_1→min

Concatenate root list of H_2 with root list of H.

if (H→min == NULL) or

(H_2→min ≠ NULL and H_2→min < H→min)

H→min = H_2→min

H→n = H_1→n + H_2→n

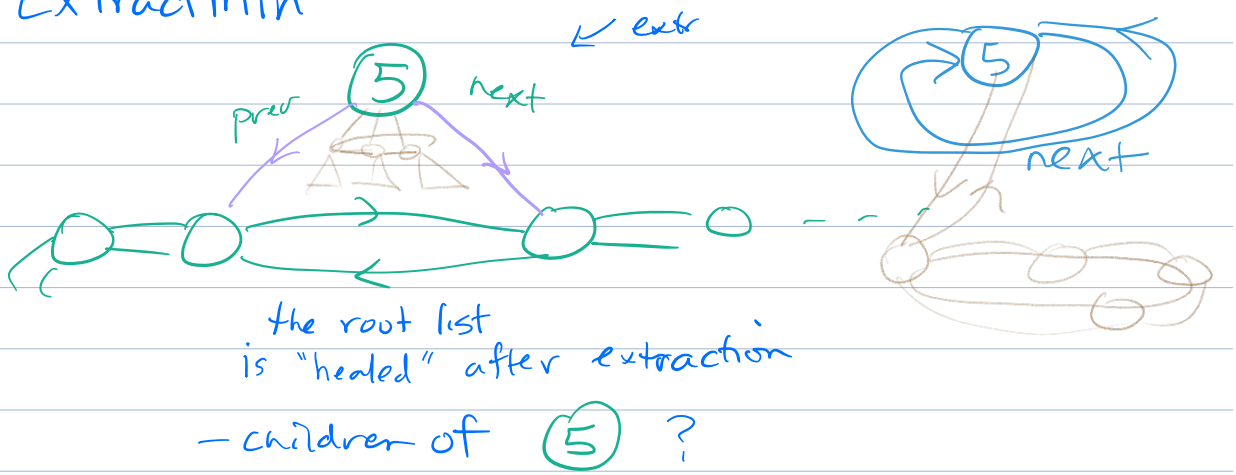
return H.

Cost = O(1)

ΔΦ = 0

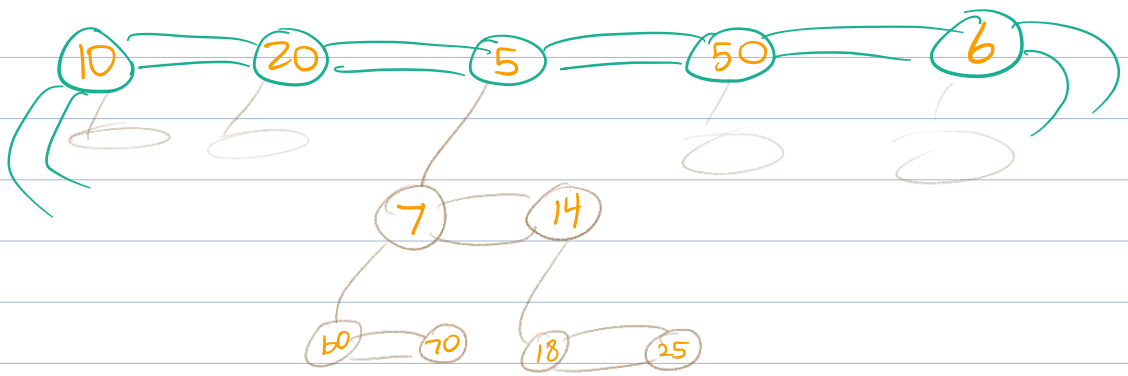
O(1) amortized

ExtractMin



FibHeap.ExtractMin

```
Z = H -> min
if Z ≠ NULL
    for each child x of z
        add x to H's root list
        x->p = NULL
    remove z from rootlist of H
    if Z == Z -> next
        H->min = NULL
    else H-> min = Z-> next
        CONSOLIDATE (H)
    H->n = H->n -1
return Z
```



H → min.