Binary heap

insert \(O(\log n)\)
deleteMin \(O(\log n)\)

conceptually a

complete

heap-ordered

binary tree
implemented as an array
delete, decrease key still \( O(\log n) \)
find Min : \( O(1) \)
use this in practice, but in theory...
Merge: combine elements from two priority queues (elements only belong to new queue afterward)

with binary heaps, need to call BuildHeap

$O(m+n) \text{ time}$
Binomial trees (of order $k = 0$)

\[ B_k \text{ (not binary)} \]

\[ B_0 \]

$B_k$: Take two copies of $B_{k-1}$ and make one the child of the other's root.
$B_k$ has $2^k$ nodes

exactly $\binom{k}{d}$ nodes at depth $d$
Binomial Heap
(binomial queue)
A collection of heap-ordered binomial trees, no two of the same order
Maybe a link to smallest root.

You have $B_k$, its binary representation of $n$ has a 1 in position $k$. 
merge:
combine both collections of trees

Repeatedly

\[ k \text{ smallest int s.t.} \]

\[ \text{two copies of } B_j \text{ in our collection have smaller root} \]

link to larger root so you have a $B_{k+1}$
(just like binary addition)

\[ H_2 \cdot (13) \]

\[ H_2 + H_2 \cdot (13) \]

\[ 14 \times 26 \]

\[ \text{No valid combination} \]
Time to merge trees of size ≤ n, like adding two O(\log n) bit numbers; O(\log n) time.
\textbf{insert}(x): merge in one-element heap with just $x$, $O(\log n)$ time.

delete Min:
Remove min root, add child sub-trees to collection finish merge
\(O(\log n)\) time

decrease key is still a percolate up.

time \(O(\log k) = O(\log n)\)
In 1986, Fredman and Tarjan needed faster insert and decrease key to speed up some graph algorithms. Fibonacci heap and a lazy binomial queue.
Collection of heap-order trees.
Link to smallest root.
No longer require they are distinct.
insert: add a one-node tree to the collection.
stop.

worst-case: \( O(1) \)

find min: use that min
root link
worst-case: \( O(1) \)
deleteMin; use link to min root
delete that root node & add children subtrees to collection
Merging of trees
For node $x$:

$\text{rank}(x)$: # of children for $x$

For heap $H$:

$\text{rank}(H)$: largest $\text{rank}(x)$ among all nodes in $H$

$\text{Trees}(H)$: # of trees in our collection
to merge after delete Min:
repeatedly combine
trees where roots
have equal rank
$\emptyset(H) := \text{trees}(H)$

insert: $O(1)$ time

$\emptyset$ increases by 1

so amortized cost is

$2 = O(1)$

delete Min: we have

$\leq \text{rank}(H) + \text{trees}(H)$

trees right after we delete root
Then spend

rank(H) +

trees(H) time

to merge trees

after merges

H: heap

vankc(H)

+ troos(H)

It's heap after mangos

heap after merges

trees(\(H_i\)) \equiv rank(\(H_i\)) + 1
amortized cost of delete Min:

\[(\text{rank}(H) + \text{trees}(H)) + (\text{rank}(H') + 1) - \text{trees}(H) =\]

\[O(\text{rank}(H') + \text{rank}(H))\]

If we only insert and delete Min, we only have binomial trees
\[ \Rightarrow \quad \text{rank} = \Theta(\log n) \]

\[ \Rightarrow \quad \text{deleteMin takes } O(\log n) \text{ amortized time} \]

decreaseKey \( (v) \):

- make \( v \) a root in the collection of trees
- if old parent of \( v \) is marked, recursively make
parent a root

( to check its parent)

else

mark parent

also, remove mark

Whenever a node becomes a tree root
Change $\emptyset$ so

$$\emptyset(H) = \text{trees}(H) + 2 \text{ marks } (H) \uparrow \text{ total } \uparrow \text{ marks}$$

if we move c nodes during decrease key, we lose c-1 marks + gain $\leq 1$ mark
am.

cost of decrease key is

\((O(1) + c) +

\left(c + 2\left(1 - (c - 1)\right)\right)\)

new trees

= \(O(1) + 4\)

= \(O(1)\)
deleteMin has cost $\mathcal{O}(\text{rank}(H))$.

\[
\begin{align*}
\text{added as children in left-to-right order} \\
\text{rank}(v_j) &\geq 0 \\
\text{rank}(v_i) &\geq i - 1
\end{align*}
\]
8

\[ f_k : \text{min number of nodes in a subtree of a root k node} \]

\[ f_0 = 1 \]

\[ f_1 = 2 \]
\[ f = 1 + 1 + f_0 + f_1 + \ldots + f_{k-3} + f_{k-2} \]

shifted Fibonacci number

\[ \Rightarrow \text{rank}(H) = O(\log n) \]
merge; combine lists
stop
0(1) am.