Quake Heaps I

- Basic definitions
- Operations

Heap: Review

- A data structure storing key-value pairs
- Supports (at a minimum)
  - insert(Key x, Value v)
  - extract-min()
- Example: Binary heap used in Heapsort

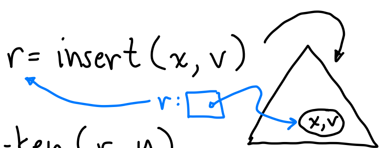
Decrease-Key:

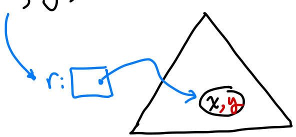
- Given an entry (x,v), decrease the key value from x to y
- How to identify the entry?
  - Heaps do not support an efficient way to find keys
- Locator: A special (abstract) object that identifies an entry of the heap.

Locator r = insert(x, v)

:

decrease-key(r, y)





- Why not just return a pointer to node (x,v)? Private information
  - Locator is a public object (eg. an inner class of the Heap)
- How about increase-key?
  - Heaps are very asymmetrical w.r.t. keys

Why decrease-key?

- Dijkstra's algorithm
- Heap tracks distances to vertices from source
- n extract-mins
- upto n^2 decrease-keys
- want decrease key fast!

History:

1984: Fibonacci Heaps (Fredman + Tarjan) ← Complex to analyze

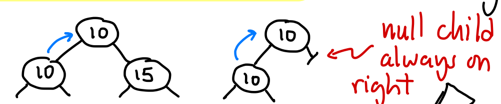
:

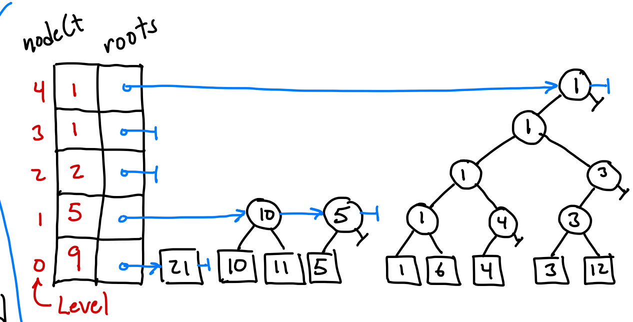
many variants

2013: Quake Heap (Timothy Chan) ← Much simpler

Quake Heap:

- Collection of binary trees
- Nodes organized in levels
- All entries are leaves at level 0
- Internal nodes have 1 or 2 children
- Parent stores smaller of child keys





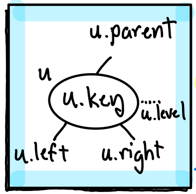
Quake Heaps II

- Utility ops
- Insert
- Decrease-key

Basic utilities:

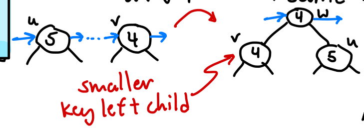
make-root (Node u): Make u a root

trivial-tree (Key x): Create 1-node tree with key x

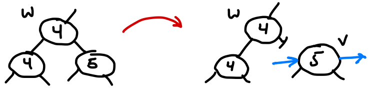


link (Node u, Node v): Link u & v

- u & v roots on same level



cut (Node w): Assuming w has right child – cuts it off as new root




```
void make-root (Node u)
    u.parent ← null
    add u to roots[u.level]
```

```
Node trivial-tree (Key x)
    Node u ← new Node key x & level 0
    nodeCt[0] += 1
    make-root(u)
    return u
```

```
Node link(Node u, Node v)
    int lev ← u.level+1   (== v.level+1)
    if(u.key ≤ v.key)
        w ← new Node(u.key, lev, u, v)
    else  w ← new Node(v.key, lev, v, u)
    nodeCt[lev] += 1
    u.parent ← v.parent ← w
    return w
```

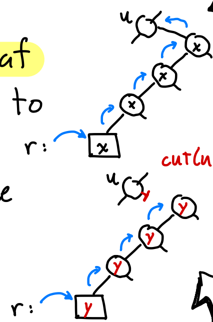
left child
right child

```
void cut (Node w)
    Node v ← w.right
    if (v ≠ null)
        w.right ← null
        make-root (v)
```

We'll apply these utilities to implement operations

Decrease Key:

- Use locator to access leaf
- Follow left-child path to highest ancestor
- Cut (u): Now we are free to change key
- In code, we'll change up order of ops




```
void decrease-key(Locator r, Key y)
    Node u ← r.getNode()  // get leaf node
    Node uChild ← null
    do {
        u.key ← y  // update key value
        uChild ← u; u ← u.parent  // go up
    } while (u ≠ null && uChild == u.left)
    if (u ≠ null) cut(u)  // cut subtree
```

Insert:

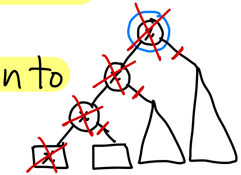
Super lazy! Just make a single node tree

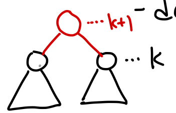
```
Locator insert(Key x)
    Node u ← new trivial-tree(x)
    return new Locator(x)
```

Quake Heap III
- Extract Min

Extract-Min:

- Find the root with smallest key (brute force)
- Delete all nodes down to leaf - many trees
- Merge trees together
  - Work bottom-up
  - Merge 2 trees at level k to form tree at lev k+1
- Too "stringy"? → Flatten QUAKE!





Quake:

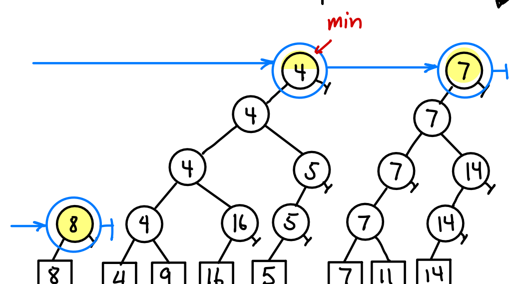
```
for (k=0,1,2,...,nLevels-2) {
    if (nodeCt[k+1] > 0.75 * nodeCt[k])
        - remove all nodes at level k+1
              and higher
        - make all nodes at level k roots
```

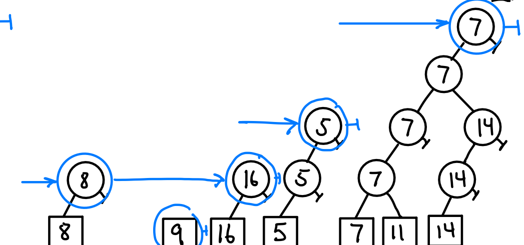
Intuition: Tree becomes "stringy" after many extractions.
- This is evidenced by the fact that node counts do not decrease
- When this happens - we flatten so we can build up later

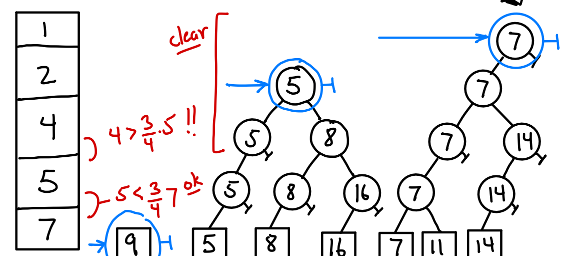
So far:

- insert + decrease-key - lazy!
  - Don't worry about
    - tree balance
    - number of roots
- insert - $O(1)$ time
- dec-key - $O(\log n)$ [later: $O(1)$]

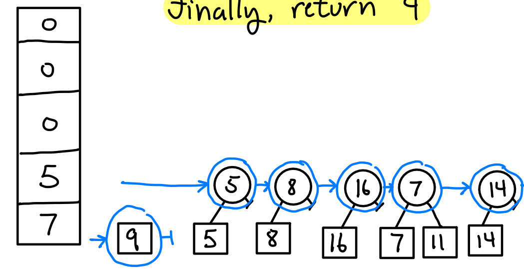
Extract Min Example:







Finally, return 4



Quake Heaps IV

- Extract min (cont)
- Faster decrease key

Key extract-min()

```
Node u ← find root (all levels)
      with smallest key
Key result ← u.key
delete-left-path(u)
remove u from roots[u.level]
merge-trees()
quake()
return result
```

Extract-min: Recap

- find root with min key
- delete left-chain to leaf
- merge trees
- quake (if needed)
- return result

void delete-left-path(u)

```
while (u ≠ null)
    cut(u)
    nodeCt[u.level] -= 1
    u ← u.left
```

void merge-trees()

```
for (lev ← 0 .. nLevels-2)
    while (roots[lev].size >= 2)
        Node u,v ← remove any 2
                   from roots[lev]
        make-root(link(u,v))
```

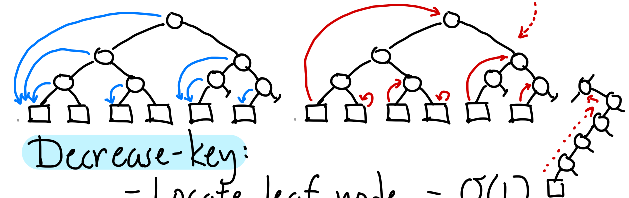
void quake()

```
for (lev ← 0 .. nLevels-2)
    if (nodeCt[lev+1] > 3/4 · nodeCt[lev])
        clear-all-above(lev)
```

Clear-all-above (lev) removes all nodes in levels lev+1 .. nLevels-1 and makes nodes of lev into roots

Faster Decrease-key:

- Each node stores pointer to leaf with key (only one change)
- Each leaf stores highest left chain ancestor (path trace $O(1)$ time)



Decrease-key:

- Locate leaf node - $O(1)$
- Trace path up left-child links
- Cut $O(1)$
- Change key

$O(\text{height}) = O(\log n)$

Times:

Insert - $O(1)$

Decrease-key - $O(\log n)$

Extract-min - ??

Can we do better? $O(1)$?

will show $O(\log n)$ amortized

Quake Heaps V
- Analysis
(Quick + Dirty)

Why is Quake Heap efficient?

- insert: $O(1)$ worst case
- decrease-key: $O(1)$ worst case (assuming enhancements)
- extract-min: As bad as $O(n)$ [no. of roots]

Amortized Analysis:

- Can show that extract-min runs in $O(\log n)$ amortized time
- Given any sequence of ops (starting from empty heap) time to do m ops (insert, dec-key, extract-min) is $O(m \cdot \log n)$

n = max no. of keys

Potential-Based Analysis:

- Each instance of the data structure assigned a potential Ψ

- Low potential ⇒ good structure
- High potential ⇒ bad structure

Idea: The amortized cost of an operation defined to be (actual-cost) + (change in Ψ)

Intuition: Expensive ops okay if they improve structure
actual = high $\Delta\Psi$ = negative

Intuition:

Extract min actual cost is high ⇒
- Tree height > $O(\log n)$
  - Quake will flatten
- Many more roots than $O(\log n)$
  - Merge trees will reduce no. to $O(\log n)$

Potential decrease compensates for high actual cost

Lemma: Amortized cost of
insert / dec-key = $O(1)$
extract-min = $O(\log n)$

Quake Heap Potential:

Let N = no. of nodes
R = no. of roots
B = no. of nodes with 1 child (bad nodes)

$$\Psi = N + 2R + 4B$$