**EECS 311: Data Structure and Data Management Lecture 20, 21**
**Priority Queues Binary Heap, Array Based Heap, Percolation**

# Priority Queue ADT

"store key-value pairs and support operations: *insert*, *delete-min*, and *reduce-key*"

**insert**$(k, v)$

adds key-value pair $(k, v)$ to queue.

**delete-min()** $\Rightarrow (k, v)$

Removes key-value pair $(k, v)$ with minimum key value $k$ and returns it.

**reduce-key**$(k, v)$

Finds the key in queue with value $v$ and reduces its key to $k$.

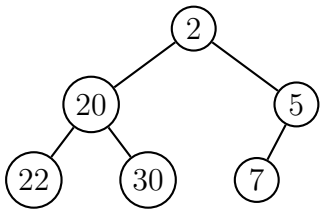
**Goal:** implement all priority queue ADT operations in $O(\log n)$.

## Binary Heaps

**Def:** a **binary heap** is a tree that satisfies

- **heap order**: the key at any node is at least the key of its parent.
- **heap structure**: it's a **complete** binary tree.

**Example:**




**Questions:**

- How to represent in memory?
- How to maintain heap properties?

**Idea:** Complete tree can be represented in array.

**Example:**

$n = 6$

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 2 | 20 | 5 | 22 | 30 | 7 | |

**Traversals**

- root() $= 1$
- leftchild$(i) = 2i$
- rightchild$(i) = 2i + 1$
- parent$(i) = \lfloor i/2 \rfloor$

**Add and Delete**

- add-key$(k)$

"add element with key $k$"

1. put key in position $n+1$.
2. set $n = n+1$.
3. fix violated heap property by "percolating up" key $k$.

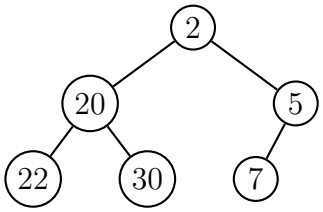
- percolate-up(i)

  "moves key at position $i$ up tree until it doesn't violate heap property."

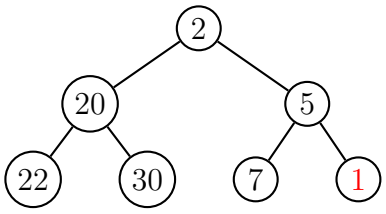
  1. if key at parent($i$) > key at $i$
     (a) swap keys.
     (b) recursively percolate-up(parent($i$))

  **Example:** add-key(1)

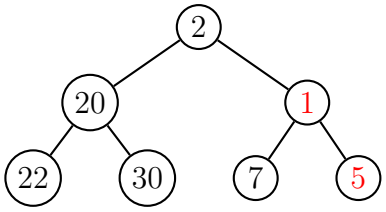
  – Initially:



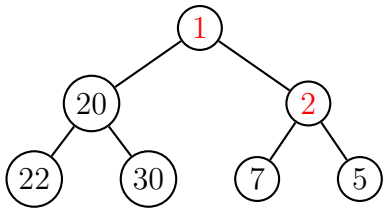

  – Add key "1" at $n+1$:




  – Percolate up at key "1":




  – Percolate up at key "1":




- delete-node($i$)

  "delete node $i$"

  1. move key at position $n$ to position $i$.
  2. Set $n = n-1$.
  3. fix violated heap order by percolate-up/down at $i$.

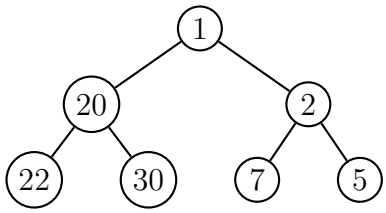
- percolate-down($i$)

  "moves key at position $i$ down tree until it doesn't violate heap property."

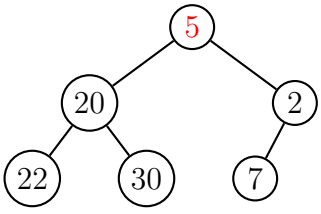
  1. Let $j$ be the child of $i$ with minimum key
  2. If the key of j < key of $i$
     (a) swap keys.
     (b) recursively percolate-down($j$)

  **Example:** delete-node(1)

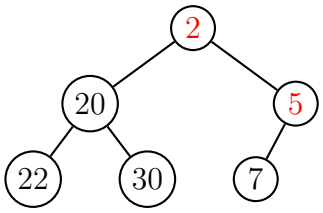
  – Initially:




  – Move key "5" to position 1



– Percolate-down key "5":




– Percolate-down key "5" ⇒ done!

## Priority Queue via Binary Heap

### P.Q. Representation

heap: "length $c$ array of key-value pairs"
aux: "length $c$ array mapping values to nodes"
n: "number of keys in queue"

### P.Q. Operations

- insert(k,v)
  1. run add-key(k) on heap.
  2. update aux with location of (k,v).
  3. update aux with location of all moved keys.
- delete-min() ⇒ (k,v)
  1. (k,v) = key and value at root.
  2. run delete-node(root()) on heap.
  3. update aux with location of all moved keys.
- reduce-key(k,v)
  1. i = aux[v]
  2. update key at $i$ to $k$.
  3. run percolate-up(i) on heap
  4. update aux with location of all moved keys.

**Runtimes:** $\Theta(logn)$ (each operation).

## Build-heap

**Fact:** sum of heights of nodes = $\Theta(n)$.

**Algorithm:** build-heap
Input: unordered array of keys, size $n$.

1. view array as complete binary tree.
2. for $i = n$ down to 1

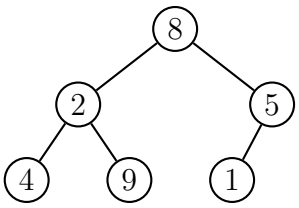
   percolate-down($i$)

**Example:**

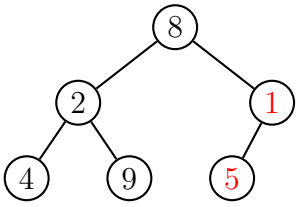
- Input unordered array.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 8 | 2 | 5 | 4 | 9 | 1 | |

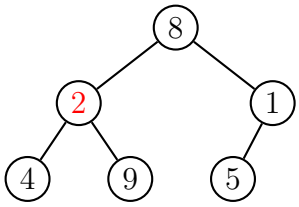
- view as tree.



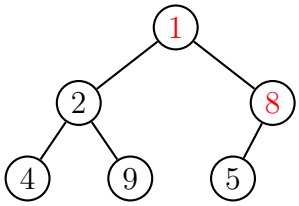

- percolate-down(3)

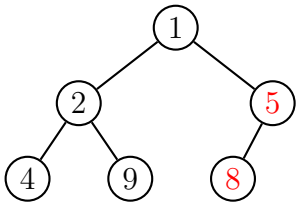


- percolate-down(2)




- percolate-down(1)




**Claim:** output of build-heap is heap-ordered.

**Proof:** by induction on $i$. trivial.

**Claim:** build-heap runs in $\Theta(n)$ time.

**Proof:** runtime = sum of node heights = $\Theta(n)$.

**Proof:** (of fact)

- runtime = "sum of heights"
- $S$ = sum of heights

$$S = \sum_{j=0}^{h} j \times \text{num. nodes at height } h$$
$$= \sum_{j=0}^{h} j \times 2^{h-j}$$

- $S = 2S - S$

$$S = \sum_{j=0}^{h} j \times 2^{h-j+1} - \sum_{j=0}^{h} j \times 2^{h-j}$$
$$= \sum_{j=0}^{h} j \times 2^{h-j+1} - \sum_{j=1}^{h+1} (j-1) \times 2^{h-j+1}$$
$$= \left[ \sum_{j=1}^{h} (j - (j-1)) \times 2^{h-j+1} \right] + 0 \times 2^{h-0+1} - h$$
$$= \left[ \sum_{j=1}^{h} 2^{h-j+1} \right] - h$$
$$= \left[ -1 + \sum_{j=0}^{h} 2^{j} \right] - h$$
$$= \left[ \sum_{j=0}^{h} 2^{j} \right] - 1 - h$$
$$= n - 1 - h$$
$$= \Theta(n).$$

□