# CSCB63 – Design and Analysis of Data Structures

Akshay Arun Bapat[1]

[1]based on notes by Anya Tafliovich, Anna Bretscher and Albert Lai

# Priority queue

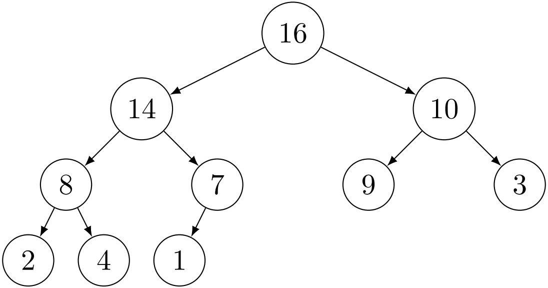
Collection of priority-job pairs; priorities must be comparable.

- `insert(`$p$`, `$j$`)`: insert job $j$ with priority $p$
- `max()`: return job with max priority
- `extract-max()`: remove and return job with max priority

# Heap

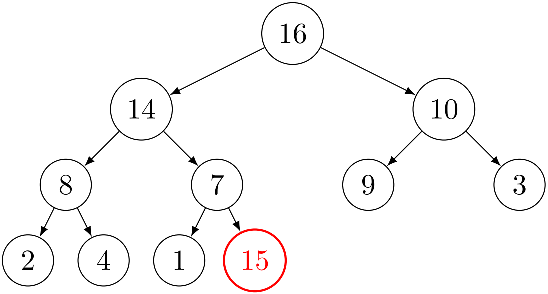
A heap is one way to store a priority queue. A heap is:

- a binary tree
- "nearly complete": every level $i$ has $2^i$ nodes, except the bottom level; the bottom nodes flush to the left
- at each node $n$: $\mathit{priority}(n) \geq \mathit{priority}(n.\mathit{left})$ and $\mathit{priority}(n) \geq \mathit{priority}(n.\mathit{right})$



# Heap insert: example

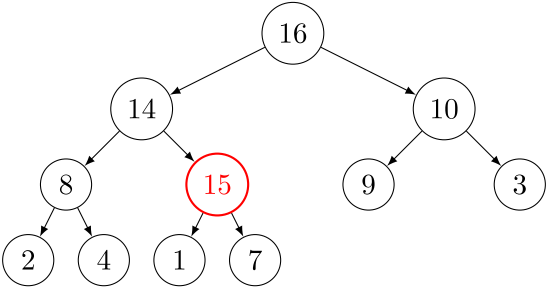
Insert job with priority 15.




- √ The tree is still "nearly-complete". But:
- ! Order of priorities bad. Fix: swap with parent.

# Heap insert: example

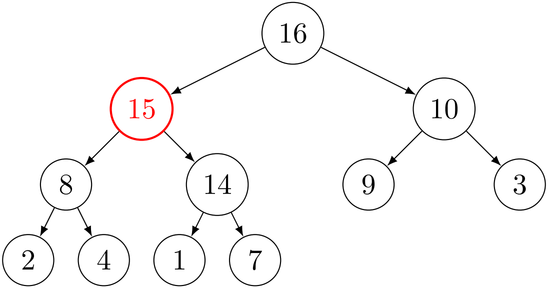
Insert job with priority 15.




- √ The tree is still "nearly-complete". But:
- ! Order of priorities bad. Fix: swap with parent.

# Heap insert: example

Insert job with priority 15.




- $\surd$ The tree is still "nearly-complete". But:
- $\surd$ Order of priorities good.

# Heap insert: algorithm

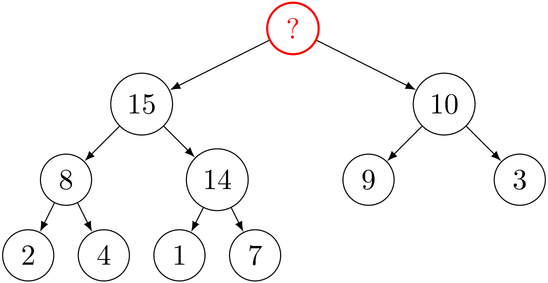
`insert(p, j)`:

1. `v := new node(p, j)`
2. insert `v` at bottom level, leftmost free place
   (keep the tree "nearly-complete")
3. while `v` has parent `p` with `p.priority < v.priority`:
   - swap `v.priority` and `p.priority`
   - swap `v.job` and `p.job`
   - `v := parent(v)`

Worst case time: $\Theta(\textit{height})$

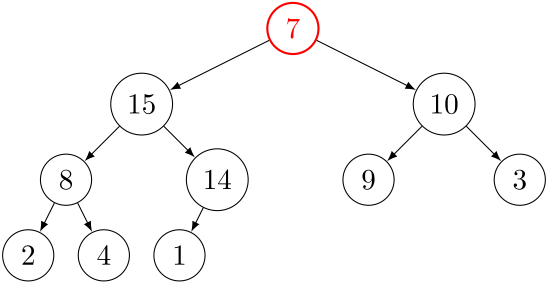
Later we will see why $\textit{height} = \lfloor \log n \rfloor + 1$. Therefore worse case time $\Theta(\log n)$.

# Heap extract-max: example




new root?

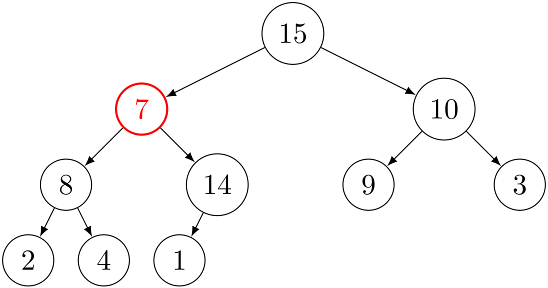
# Heap extract-max: example




replace by the bottom level, rightmost item.

- $\surd$ The tree is still "nearly-complete".
- ! Order of priorities bad. Fix: swap with the larger child. (Why not the smaller child?)

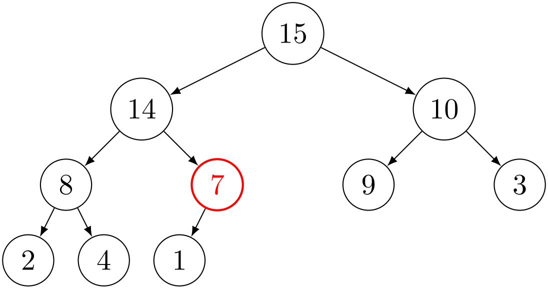
# Heap extract-max: example




replace by the bottom level, rightmost item.

- $\surd$ The tree is still "nearly-complete".
- ! Order of priorities bad. Fix: swap with the larger child. (Why not the smaller child?)

# Heap extract-max: example




replace by the bottom level, rightmost item.

- $\surd$ The tree is still "nearly-complete".
- $\surd$ Order of priorities good.

# Heap extract-max: algorithm

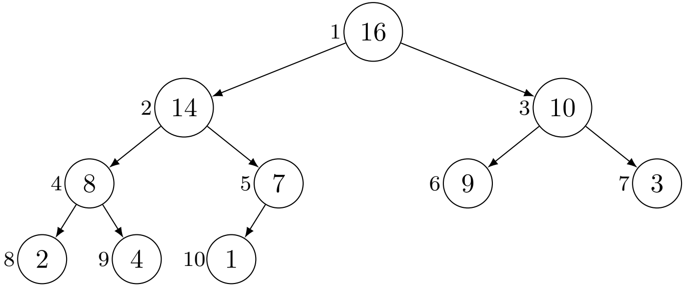
`extract-max()`:

1. `max_p, max_j = root.priority, root.job`
2. move `(priority, job)` from last (bottom, rightmost) node into root
3. remove last node
4. `v := root`
5. while `v` has child `c` with `c.priority > v.priority`:
   - `c :=` child of `v` with largest priority
   - swap `v.priority` and `c.priority`
   - swap `v.job` and `c.job`
   - `v := c`
6. return `max_p, max_j`

Worst case time: $\Theta(height)$

Later we will see why $height = \lfloor \log n \rfloor + 1$. Therefore worse case time $\Theta(\log n)$.

# Heap in array/vector




| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

## Heap in array/vector

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1 | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Easy:

- where to insert/remove? simply at the end
- saves space: no pointers to store

Where are children/parents?

- left child of node at index $i$: at index $2 \times i$
- right child of node at index $i$: at index $2 \times i + 1$
- parent of index node at $i$: at index $\lfloor i/2 \rfloor$

Downside?

# Heap: height

Let $n$ be the number of nodes, $h$ be the height.

- largest $n$: bottom level is full
  - $n = 2^h - 1$
- smallest $n$: only 1 node at bottom level
  - $h - 1$ levels are full
  - $n = (2^{h-1} - 1) + 1$

$$
\begin{aligned}
(2^{h-1} - 1) + 1 \leq & \quad n & \leq 2^h - 1 \\
2^{h-1} \leq & \quad n & < 2^h \\
h - 1 \leq & \quad \log_2 n & < h \\
h \leq & \quad (\log_2 n) + 1 & < h + 1 \\
h = & \quad \lfloor \log_2 n \rfloor + 1 &
\end{aligned}
$$