Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort
Priority Queues

• Priority Queues
• Trees and Heaps
• Representations of Heaps
• Algorithms on Heaps
• Building a Heap
• Heapsort
Priority Queues

- **Priority queues.** Maintain dynamic set $S$ supporting the following operations. Each element has key $x.key$ and satellite data $x.data$.
  - $\text{MAX}()$: return element med largest key.
  - $\text{EXTRACTMAX}()$: return and remove element with largest key.
  - $\text{INCREASEKEY}(x, k)$: set $x.key = k$. (assume $k \geq x.key$)
  - $\text{INSERT}(x)$: set $S = S \cup \{x\}$
Priority Queues

- **Applications**.
  - Scheduling
  - Shortest paths in graphs (Dijkstra's algorithm)
  - Minimum spanning trees in graphs (Prim's algorithm)
  - Compression (Huffman's algorithm)
  - ...

- **Challenge.** How can we solve problem with current techniques?
Priority Queues

• **Solution 1: Linked list.** Maintain $S$ in a doubly-linked list.

  ![Linked list diagram]

  - $\text{MAX}()$: linear search for largest key.
  - $\text{EXTRACTMAX}()$: linear search for largest key. Remove and return element.
  - $\text{INCREASEKEY}(x, k)$: set $x.\text{key} = k$.
  - $\text{INSERT}(x)$: add element to front of list (assume element does not exist in $S$ beforehand).

• **Time.**
  - $\text{MAX}$ and $\text{EXTRACTMAX}$ in $O(n)$ time ($n = |S|$).
  - $\text{INCREASEKEY}$ and $\text{INSERT}$ in $O(1)$ time.

• **Space.**
  - $O(n)$. 
Priority Queues

- **Solution 2: Sorted linked list.** Maintain S in a sorted doubly-linked list.

```
56 <-> 42 <-> 23 <-> 18 <-> 10 <-> 7 <-> 5 <-> 2
```

- **MAX():** return first element.
- **EXTRACTMAX():** return og remove first element.
- **INCREASEKEY(x, k):** set x.key = k. Linear search to move x to correct position.
- **INSERT(x):** linear search to insert x at correct position.

- **Time.**
  - MAX and EXTRACTMAX in O(1) time.
  - INCREASEKEY and INSERT in O(n) time.

- **Space.**
  - O(n).
Priority Queues

<table>
<thead>
<tr>
<th>Data structure</th>
<th>MAX</th>
<th>EXTRACTMAX</th>
<th>INCREASEKEY</th>
<th>INSERT</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>sorted linked list</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

- **Challenge.** Can we do significantly better?
Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort
Trees

- **Rooted trees.**
  - Nodes (or vertices) connected with edges.
  - Connected and acyclic.
  - Designated root node.
  - Special type of graph.

- **Terminology.**
  - Children, parent, descendant, ancestor, leaves, internal nodes, path,..

- **Depth and height.**
  - **Depth** of v = length of path from v to root.
  - **Height** of v = length of path from v to descendant leaf.
  - Depth of T = height of T = length of longest path from root to a leaf.
Trees

• Binary tree.
  • Rooted tree.
  • Each node has at most two children called the left child and right child

• Complete binary tree. Binary tree where all levels of tree are full.

• Almost complete binary tree. Complete binary tree with 0 or more rightmost leaves deleted.

• Lemma. Height of an (almost) complete binary tree with n nodes is $\Theta(\log n)$.

• Pf. See exercises.
Heaps

- **Heaps.** Almost complete binary tree that satisfies heap-order.

- **Heap-order.**
  - All nodes store one element.
  - For all nodes $v$.
    - all keys in left subtree and right subtree are $\leq v.key$.

- **Max-heap vs min-heap.**
Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort
Heap

- **Data structure.** We need the following navigation operations on a heap.
  - PARENT(x): return parent of x.
  - LEFT(x): return left child of x.
  - RIGHT(x): return right child of x.

- **Challenge.** How can we represent a heap compactly to support fast navigation?
Heap

- **Linked representation.** Each node stores
  - v.key
  - v.parent
  - v.left
  - v.right

- **PARENT, LEFT, RIGHT** by following pointer.

- **Time.** $O(1)$
- **Space.** $O(n)$
Heap

- Array representation.
  - Array $H[0..n]$
  - $H[0]$ unused
  - $H[1..n]$ stores nodes in level order.

- $\text{PARENT}(x)$: return $\lfloor x/2 \rfloor$
- $\text{LEFT}(x)$: return $2x$.
- $\text{RIGHT}(x)$: return $2x + 1$

- Time. $O(1)$
- Space. $O(n)$
Priority Queues

• Priority Queues
• Trees and Heaps
• Representations of Heaps
• Algorithms on Heaps
• Building a Heap
• Heapsort
Algorithms on Heaps

**BubbleUp(x):**
- If heap order is violated at node x because key is larger than key at parent:
  - Swap x and parent
  - Repeat with parent until heap order is satisfied.

**BubbleDown(x):**
- If heap order is violated at node x because key is smaller than key at left or right child:
  - Swap x and child c with largest key.
  - Repeat with child until heap order is satisfied.
Algorithms on Heaps

• **BUBBLEUP(x):**
  - If heap order is violated at node x because key is larger than key at parent:
    • Swap x and parent
    • Repeat with parent until heap order is satisfied.
• **BUBBLEDOWN(x):**
  - If heap order is violated at node x because key is smaller than key at left or right child:
    • Swap x and child c with largest key.
    • Repeat with child until heap order is satisfied.
• **Time.**
  • **BUBBLEUP** and **BUBBLEDOWN** in $\Theta(\log n)$ time.
• How can we use them to implement a priority queue?
Priority Queues

\[ \text{Max}() \]
\[
\text{return } H[1] 
\]

\[ \text{ExtractMax}() \]
\[
\begin{align*}
  r &= H[1] \\
  H[1] &= H[n] \\
  n &= n - 1 \\
  \text{BubbleDown}(1) \\
  \text{return } r
\end{align*}
\]

\[ \text{Insert}(x) \]
\[
\begin{align*}
  n &= n + 1 \\
  H[n] &= x \\
  \text{BubbleUp}(n)
\end{align*}
\]

\[ \text{IncreaseKey}(x,k) \]
\[
\begin{align*}
  H[x] &= k \\
  \text{BubbleUp}(x)
\end{align*}
\]

- **Ex.** Trace execution of following sequence in initially empty heap: 2, 5, 7, 6, 4, E, E
- Numbers mean **Insert** og E is **ExtractMax**. Draw heap after each operation.
Priority Queues

\[
\text{MAX}() \\
\quad \text{return } H[1]
\]

\[
\text{EXTRACTMAX}() \\
\quad \text{r} = H[1] \\
\quad H[1] = H[n] \\
\quad n = n - 1 \\
\quad \text{BUBBLEDOWN}(1) \\
\quad \text{return } r
\]

\[
\text{INSERT}(x) \\
\quad n = n + 1 \\
\quad H[n] = x \\
\quad \text{BUBBLEUP}(n)
\]

\[
\text{INCREASEKEY}(x, k) \\
\quad H[x] = k \\
\quad \text{BUBBLEUP}(x)
\]

- **Time.**
  - **Max** in \(\Theta(1)\) time.
  - **EXTRACTMAX**, **INCREASEKEY**, and **INSERT** in \(\Theta(\log n)\) time.
Heaps with array data structure is an example of an implicit data structure.
Prioritetskøer

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering
Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort
Building a Heap

- **Building a heap.** Given n integers in a array \( H[0..n] \), convert array to a heap.
Building a Heap

- **Solution 1: top-down construction**
  - For all nodes in increasing level order apply BUBBLEUP.

- **Time.**
  - For each node of depth $d$, we use $O(d)$ time.
  - $1$ node of depth $0$, $2$ nodes of depth $1$, $4$ nodes of depth $2$, ..., $\sim n/2$ nodes of depth $\log n$.
  - $\implies$ total time is $\Theta(n \log n)$

- **Challenge.** Can we do better?
Building a Heap

• **Solution 2: bottom-up construction**
  
  • For all nodes in decreasing level order apply **BUBBLEDOWN**.

• **Time.**
  
  • For each node of height \( h \) we use \( O(h) \) time.
  
  • 1 node of height \( \log n \), 2 nodes of height \( \log n - 1 \), ..., \( n/4 \) nodes of height 1, \( n/2 \) nodes of height 0.

  • \( \Rightarrow \) total time is \( \Theta(n) \) (see exercise)
Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort
Heapsort

• **Sorting.** How can we sort an array $H[1..n]$ using a heap?

  • **Solution.**
    - Build a heap for $H$.
    - Apply $n$ EXTRACTMAX.
      - Insert results in the end of array.
    - Return $H$.

• **Time.**
  - Heap construction in $\Theta(n)$ time.
  - $n$ EXTRACTMAX in $\Theta(n\log n)$ time.
  - $\Rightarrow$ total time is $\Theta(n\log n)$.
Theorem. We can sort an array in $\Theta(n \log n)$ time.

- Uses only $O(1)$ extra space.
- In-place sorting algorithm.
- Equivalence of sorting and priority queues.
Priority Queues

- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort