

Priority Queues

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

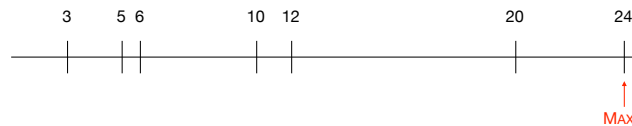
Philip Bille

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 $x.data$.
 - **MAX()**: return element with **largest** key.
 - **EXTRACTMAX()**: return **and remove** element with **largest** key.
 - **INCREASEKEY**(x, k): set $x.key = k$. (assume $k \geq x.key$)
 - **INSERT**(x): set $S = S \cup \{x\}$

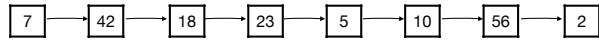


Priority Queues

- **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

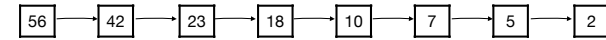
- **Solution 1: Linked list.** Maintain S in a linked list.



- MAX(): linear search for largest key.
- EXTRACTMAX(): linear search for largest key. Remove and return element.
- INCREASEKEY(x, k): set x.key = k.
- INSERT(x): add element to front of list (assume element does not exist in S beforehand).
- **Time.**
 - MAX and EXTRACTMAX in $O(n)$ time ($n = |S|$).
 - INCREASEKEY and INSERT in $O(1)$ time.
- **Space.**
 - $O(n)$.

Priority Queues

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



- MAX(): return first element.
- EXTRACTMAX(): return and 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

Data structure	MAX	EXTRACTMAX	INCREASEKEY	INSERT	Space
linked list	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
sorted linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$

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

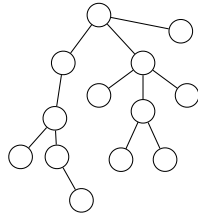
Priority Queues

- 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,...

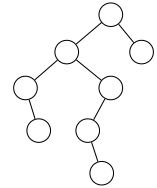
- **Depth and height.**

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

Trees

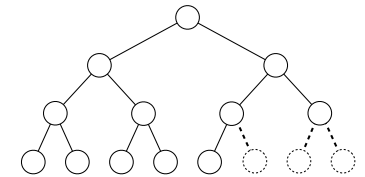
- **Binary tree.**

- 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 $\Theta(\log n)$.
- **Proof.** See exercises.

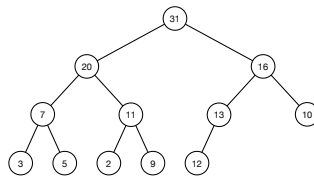


Heaps

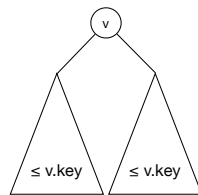
- **Heaps.** Almost complete binary tree. All nodes store one element and the tree satisfies **heap-order**.

- **Heap-order.**

- For all nodes v :
 - all keys in left subtree and right subtree are $\leq v.key$.



- **Max-heap vs min-heap.**



Priority Queues

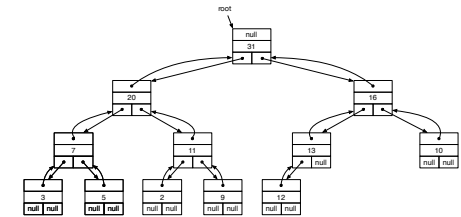
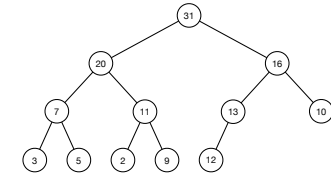
- Priority Queues
- Trees and Heaps
- **Representations of Heaps**
- Algorithms on Heaps
- Building a Heap
- Heapsort

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?

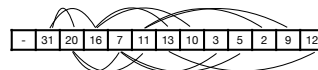
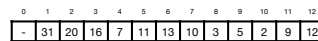
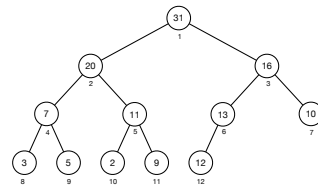
Heap

- **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**.
- PARENT(x): return $\lfloor x/2 \rfloor$
- LEFT(x) : return 2x.
- RIGHT(x): return 2x + 1
- **Time.** O(1)
- **Space.** O(n)

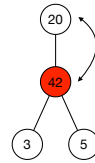


Priority Queues

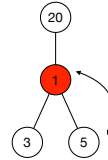
- Priority Queues
- Trees and Heaps
- Representations of Heaps
- **Algorithms on Heaps**
- Building a Heap
- Heapsort

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.

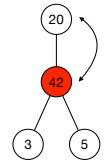


- BUBBLEDOWN(x):
 - If heap order is violated at node x because key is smaller than key at left or right child:
 - Swap x and child c with **largest** key.
 - Repeat with child until heap order is satisfied.

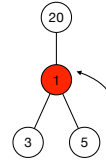


Algorithms on Heaps

- 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.



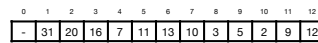
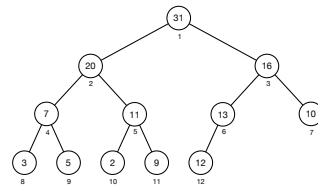
- BUBBLEDOWN(x):
 - If heap order is violated at node x because key is smaller than key at left or right child:
 - Swap x and child c with **largest** key.
 - Repeat with child until heap order is satisfied.



- Time.
 - BUBBLEUP and BUBBLEDOWN in $O(\log n)$ time.
 - How can we use them to implement a priority queue?

Priority Queues

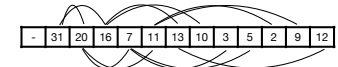
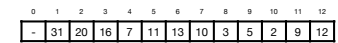
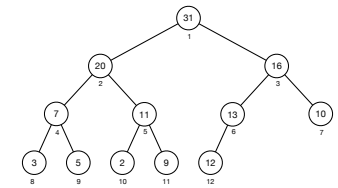
MAX() return H[1]	INSERT(x) n = n + 1 H[n] = x BUBBLEUP(n)
EXTRACTMAX() r = H[1] H[1] = H[n] n = n - 1 BUBBLEDOWN(1) return r	INCREASEKEY(x, k) H[x] = k BUBBLEUP(x)



- Exercise. 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

MAX() return H[1]	INSERT(x) n = n + 1 H[n] = x BUBBLEUP(n)
EXTRACTMAX() r = H[1] H[1] = H[n] n = n - 1 BUBBLEDOWN(1) return r	INCREASEKEY(x, k) H[x] = k BUBBLEUP(x)



- Time.
 - MAX in $O(1)$ time.
 - EXTRACTMAX, INCREASEKEY, and INSERT in $O(\log n)$ time.

Priority Queues

Data structure	MAX	EXTRACTMAX	INCREASEKEY	INSERT	Space
linked list	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
sorted linked list	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
heap	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$

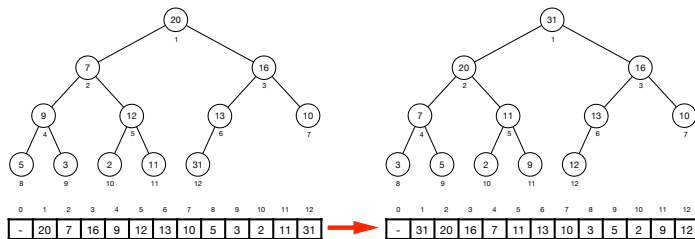
- Heaps with array data structure is an example of an **implicit data structure**.

Priority Queues

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

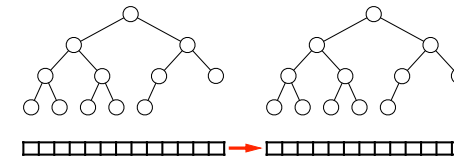
Building a Heap

- **Building a heap.** Given n integers in a array $H[1..n]$, convert array to a heap.



Building a Heap

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

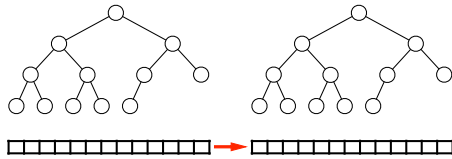


- **Time.**
 - For each node of depth d , we use $O(d)$ time.
 - 1 node of depth 0, 2 nodes of depth 1, 4 nodes of depth 2, ..., $\sim n/2$ nodes of depth $\log n$.
 - \Rightarrow total time is $O(n \log n)$
- **Challenge.** Can we do better?

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 $O(n)$ (see exercise)

Priority Queues

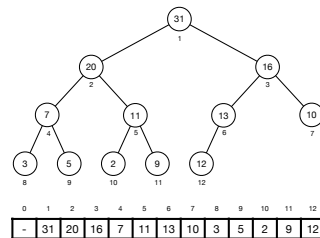
- Priority Queues
- Trees and Heaps
- Representations of Heaps
- Algorithms on Heaps
- Building a Heap
- Heapsort

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 .



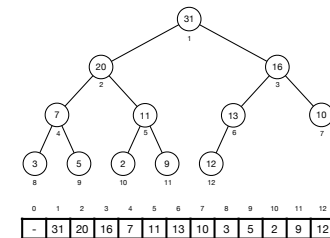
- **Time.**

- Heap construction in $O(n)$ time.
- n EXTRACTMAX in $O(\log n)$ time.
- \Rightarrow total time is $O(n \log n)$.

Heapsort

- **Theorem.** We can sort an array in $O(n \log n)$ time.

- Uses only $O(1)$ extra space.
- **In-place** sorting algorithm.



Priority Queues

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