# Priority Queues

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

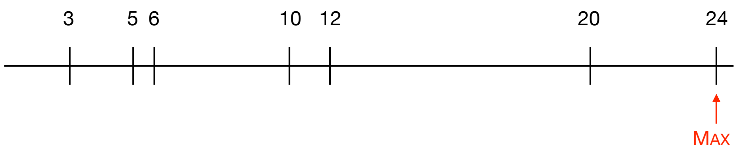
Philip Bille

# 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 med largest key.
  - ExtractMax(): return and remove element with largest key.
  - IncreaseKey(x, k): set x.key = k. (assume k ≥ x.key)
  - Insert(x): set S = S ∪ {x}



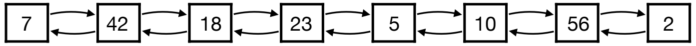

## Priority Queues

- 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

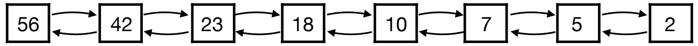
- Solution 1: Linked list. Maintain S in a doubly-linked list.




- 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 doubly-linked list.




- 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

| 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) |

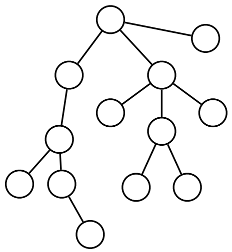
- Challenge. Can we do significantly better?

# Priority Queues

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

## Trees

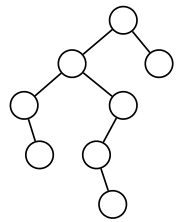
- Rooted trees.
  - Nodes (or vertices) connected with edges.
  - Connected and acyclic.
  - Designated root node.
  - Special type of graph.



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

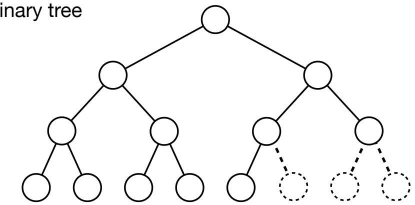
## Trees

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



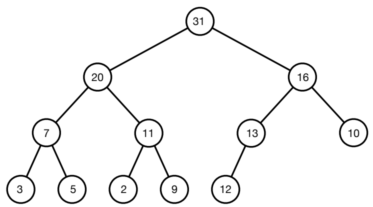
- Complete binary tree. Binary tree where all levels of tree are full.
- Almost complete binary tree. Complete binary tree with 0 or more rightmost leaves deleted.

- Lemma. Height af an (almost) complete binary tree with n nodes is Θ(log n).
- Pf. See exercises.



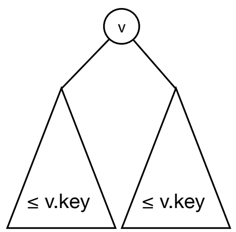
## Heaps

- 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 ≤ 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.

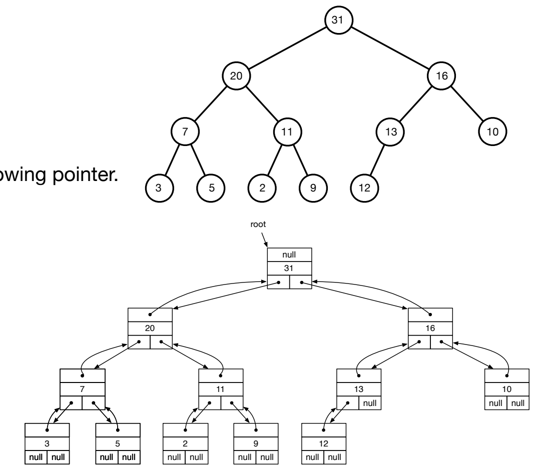
- Challenge. How can we represent a heap compactly to support fast navigation?

# 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)

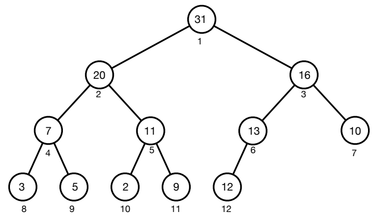


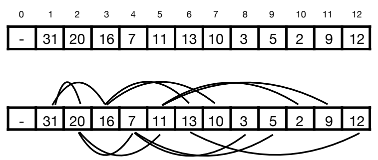

# Heap

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

- PARENT(x): return ⌊x/2⌋
- LEFT(x) : return 2x.
- RIGHT(x): return 2x + 1

- Time. O(1)
- Space. O(n)



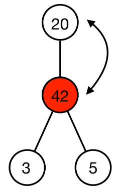




# Priority Queues

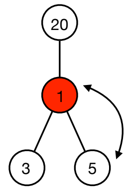
- Priority Queues
- Trees and Heaps
- Representations of Heaps
- **Algorithms on Heaps**
- Building a Heap
- Heapsort

## 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.

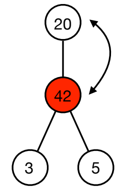


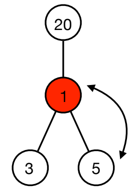

- BUBBLEDOWN(x):
  - If heap order is violated at node x because key is smaller than key at left or right child:
  - Swap x and child c with largest key.
  - Repeat with child until heap order is satisfied.




## 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 $\Theta(\log n)$ time.
- How can we use them to implement a priority queue?



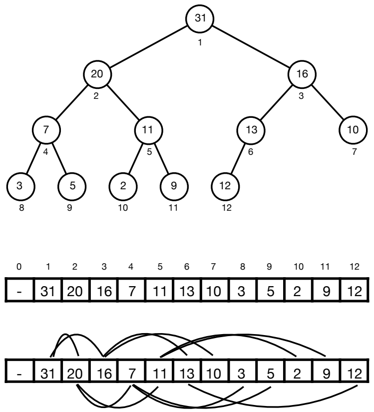

## Priority Queues

```
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)
```




- 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.

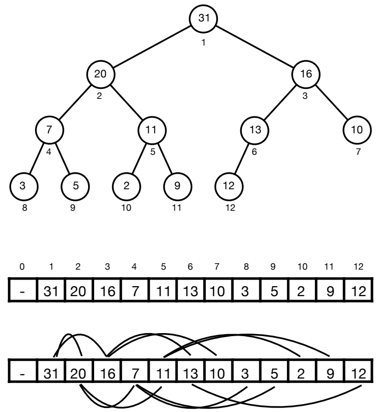
## Priority Queues

```
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 $\Theta(1)$ time.
  - EXTRACTMAX, INCREASEKEY, and INSERT in $\Theta(\log n)$ 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.

## Prioritetskøer

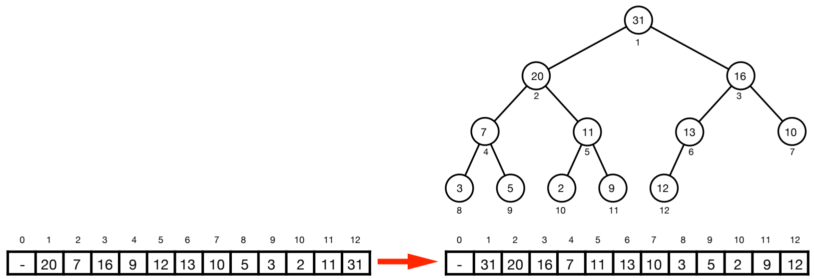
- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

## Priority Queues

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

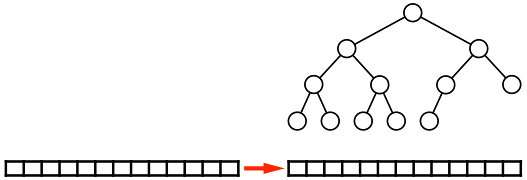
## Building a Heap

- Building a heap. Given n integers in a array H[0..n], convert array to a heap.



## Building a Heap

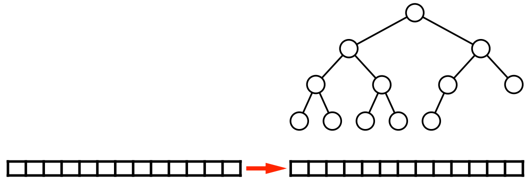
- Solution 1: top-down construction
  - For all nodes in increasing level order apply BubbleUp.



- 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, ..., ~n/2 nodes of depth log n.
  - ⇒ 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.



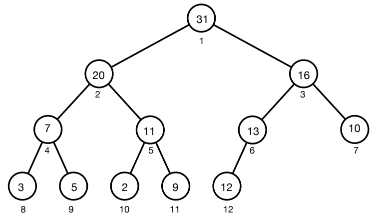
- Time.
  - For each node of height h we use O(h) time.
  - 1 node of height log n, 2 nodes of height log n - 1, ..., n/4 nodes of height 1, n/2 nodes of height 0.
  - ⇒ total time is $\Theta$(n) (see exercise)

# 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.



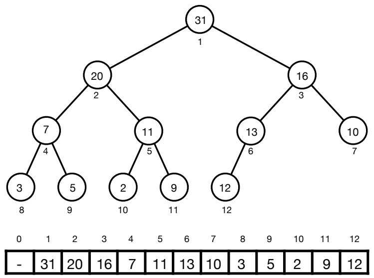

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 31 | 20 | 16 | 7 | 11 | 13 | 10 | 3 | 5 | 2 | 9 | 12 |

- Time.
  - Heap construction in $\Theta$(n) time.
  - n ExtractMax in $\Theta$(nlog n) time.
  - ⇒ total time is $\Theta$(nlog n).

# Heapsort

- Theorem. We can sort an array in Θ(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