# CSE 230: Data Structures

Lecture 8.2: Priority Queues

Dr. Vidhya Balasubramanian



Amrita School of Engineering
Amrita Vishwa Vidyapeetham

# Priority Queues

- Is an abstract data type which is a collection of items like other ADTs
  - Additionally there is a priority associated with each item
  - An element with high priority is served before an element with lower priority

| | Silver Card | Gold Card | Platinum Card |
|---|---|---|---|
| Priority Check-in | ✓ | ✓ | ✓ |
| Priority Boarding | ✓ | ✓ | ✓ |
| Additional Baggage Allowance | 10 kg | 15 kg | 20 kg |
| Lounge Access | - | ✓ | ✓ |

Src:airberlin.com

# Priority Queue ADT

- An item in a priority queue P is represented as follows
  - (key, element), key is the priority
- Operations
  - insertItem(*k*, *o*): inserts an item with key *k* and element *o*
  - removeMin() : removes the item with the smallest key
  - minKey() : returns, but does not remove, the smallest key of P
  - minElement(): returns, but does not remove, the element of an item with smallest key
  - size(), isEmpty()

# Priority Queue Example

| Operation | Output | Priority Queue |
|---|---|---|
| insertItem(5,A) | - | {(5,A)} |
| insertItem(9,C) | - | {(5,A), (9,C)} |
| insertItem(3,B) | - | {(3,B),(5,A), (9,C)} |
| insertItem(7,D) | - | {(3,B),(5,A),(7,D) (9,C)} |
| minElement() | B | {(3,B),(5,A),(7,D) (9,C)} |
| minKey() | 3 | {(3,B),(5,A),(7,D) (9,C)} |
| removeMin() | (3,B) | {(5,A),(7,D) (9,C)} |
| minElement() | A | {(5,A),(7,D) (9,C)} |
| removeMin() | (5,A) | {(7,D) (9,C)} |
| removeMin() | (7,D) | {(9,C)} |
| size() | 1 | {(9,C)} |

# Total Order Relation

- Keys in a priority queue follow a total ordered relation
  - Two distinct items in a priority queue can have the same key
- A relation <= is a total order on a set S ("<= totally orders S") if the following properties hold.
  - Reflexivity: a<=a for all a in S.
  - Antisymmetry: a<=b and b<=a implies a=b.
  - Transitivity: a<=b and b<=c implies a<=c.
  - Comparability: For any a,b in S, either a<=b or b<=a

# Comparator ADT

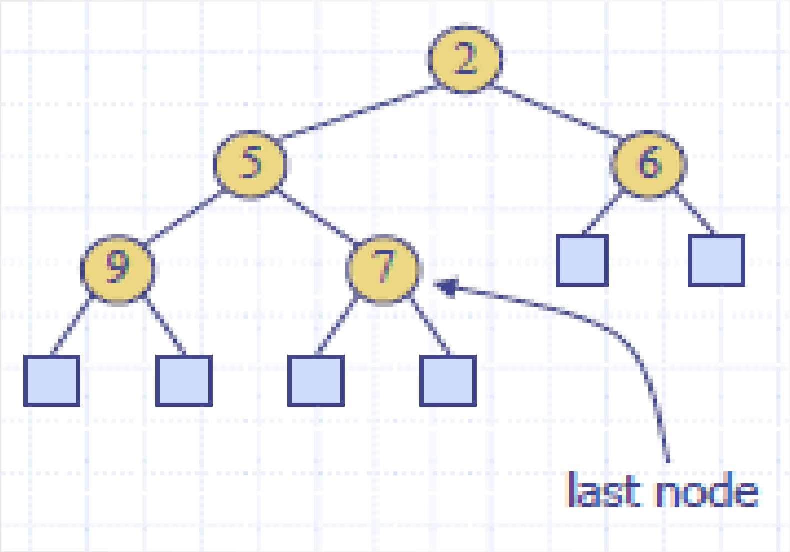
- comparator encapsulates the action of comparing two objects according to a given total order relation
- A generic priority queue uses a comparator as a template argument, to define the comparison function (<,=,>)
- Function
  - comp(a,b)
    - Returns integer i, such that i<0, i=0 or i>0
    - Value of i depends on whether a<b, a=b or a>b respectively
  - When the priority queue needs to compare two keys, it uses its comparator

# Sequence based Priority Queue

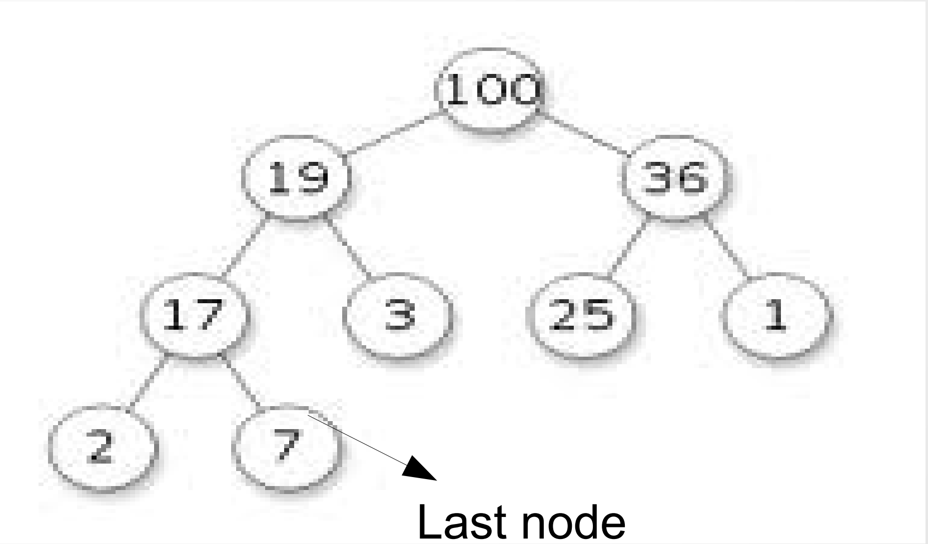
- Unsorted Sequence
  - Store items in a list based sequence in an arbitrary order
  - Performance
    - insertItem: O(1) time since it can be inserted anywhere
    - removeMin: O(n) to find the smallest key in the array
- Sorted Sequence
  - Store items sorted by key
    - insertItem: O(n) to find and insert item at right place
    - removeMin: O(1): element is at front of sequence

# Heaps

- A heap implements a priority queue
- Stores elements in a binary tree
  - insertions and deletions logarithmic time




Src: Goodrich notes



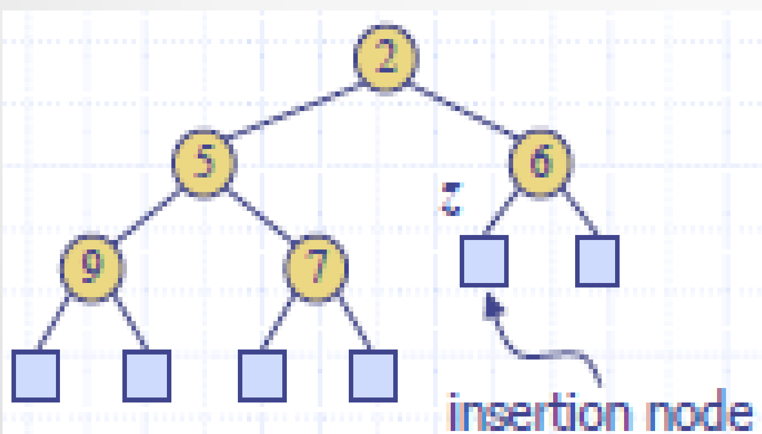

http://upload.wikimedia.org/wikipedia/commons/thumb/3/38/Max-Heap.svg/240px-Max-Heap.svg.png

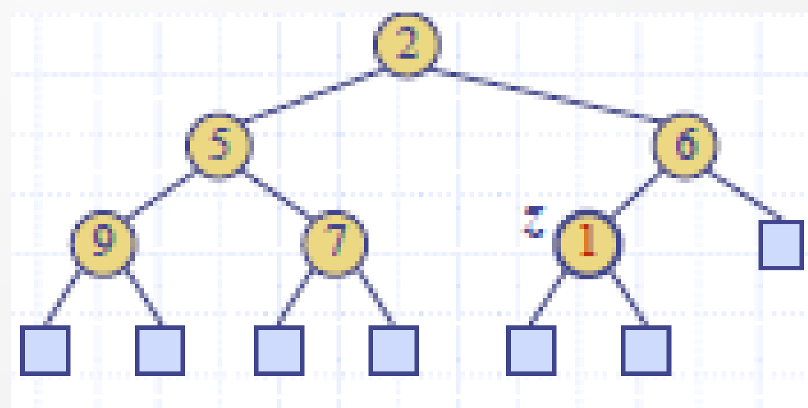
# Heap: Properties

- Heap-Order Property
  - For every node v other than the root, the key stored at v is greater than or equal to the key stored at v's parent
  - key(v) ≥ key(parent(v)) (min-heap)
  - Or vice versa for a max-heap
- Complete Binary tree
  - A binary tree with height h is complete if the levels 0,1,2,...h-1 have the maximum number of nodes possible and
  - All internal nodes are to the left of the external nodes
  - Helps keep the height of the heap small

# Insertion in a Heap

- Corresponds to insertion in a priority queue
  - Find the insertion node z (the new last node)
  - Store k at z and expand z into an internal node
    - Expanding means adding a child
  - Restore the heap-order property



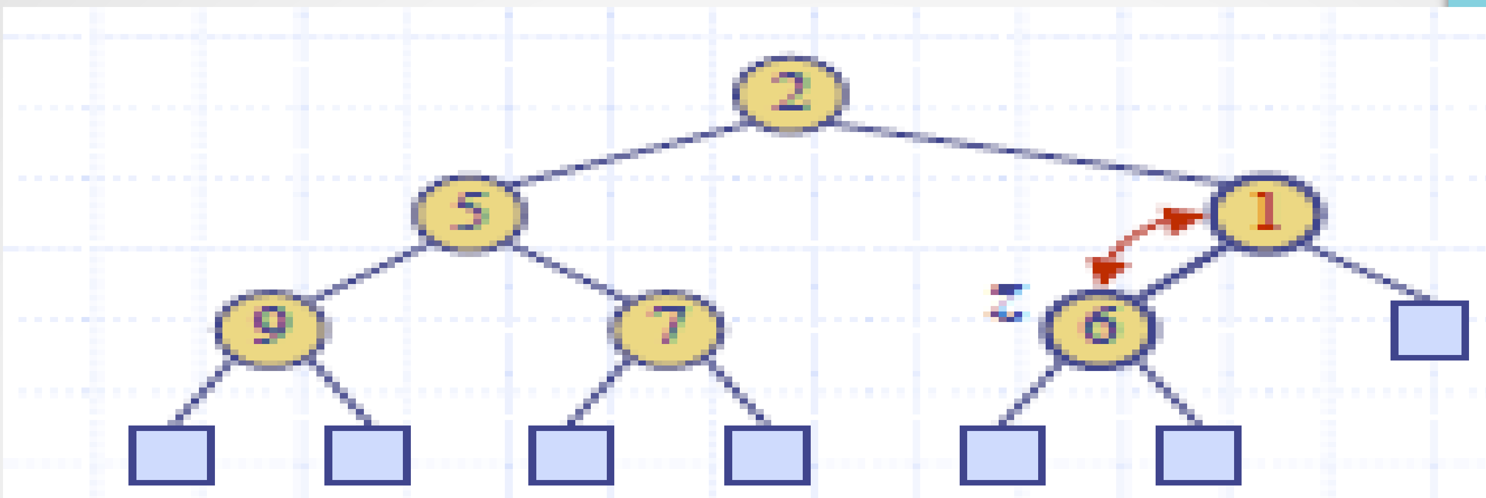



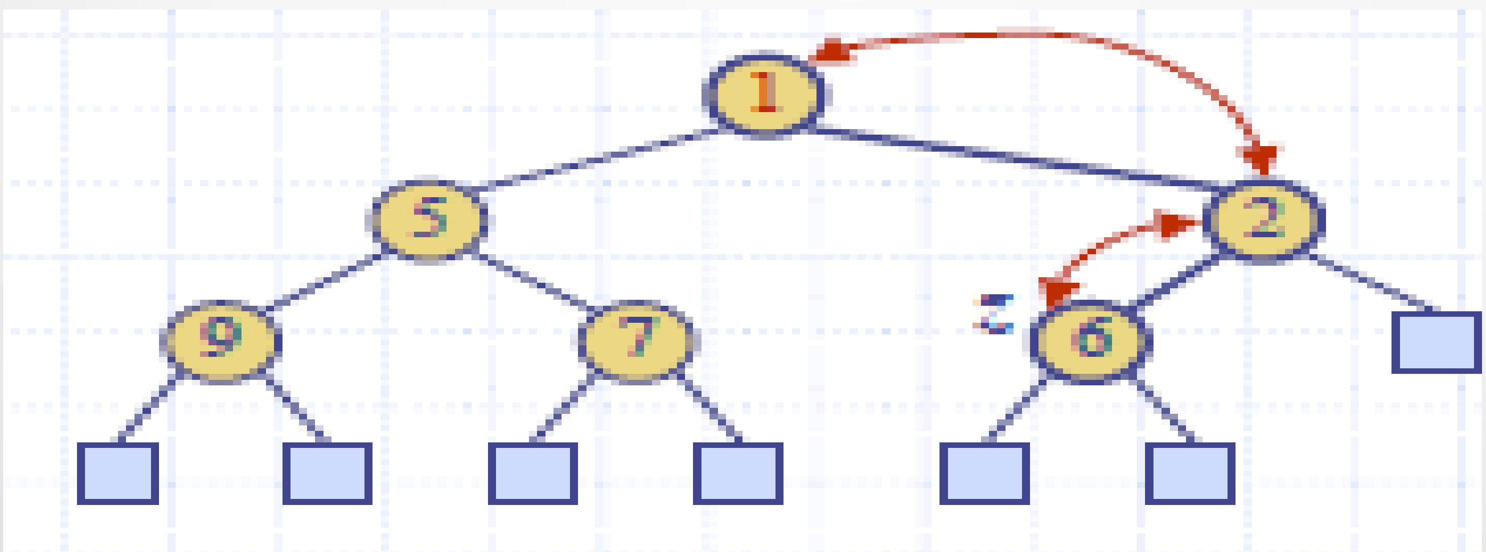

Src: Goodrich notes

# Upheap

- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height O(log n), upheap runs in O(log n) time

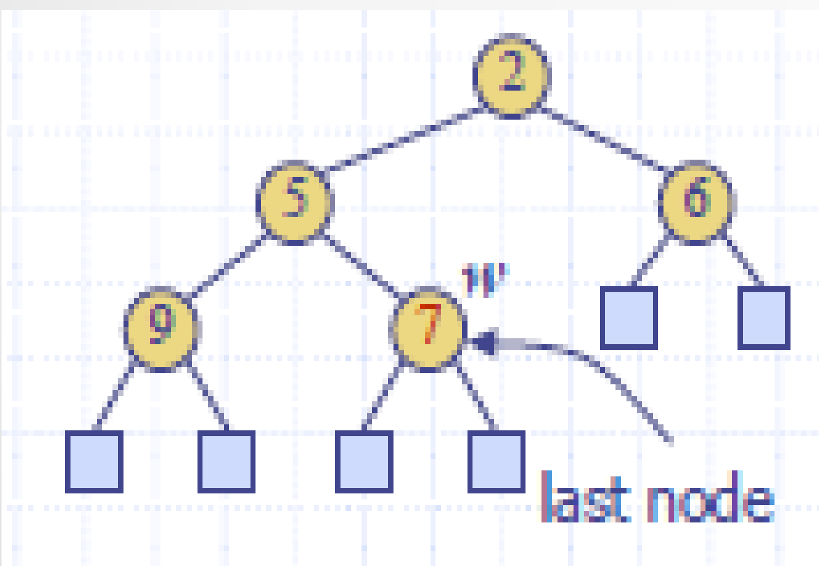
# Upheap

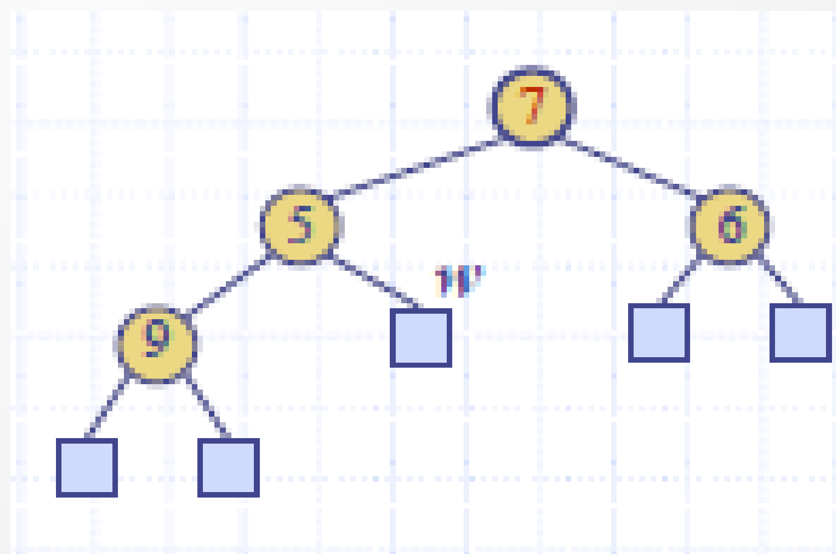


# Removal from a heap

- Removes root from the heap
- Replace the root key with the key of the last node w
- Compress w and its children into a leaf
- Restore the heap-order property using down-heap

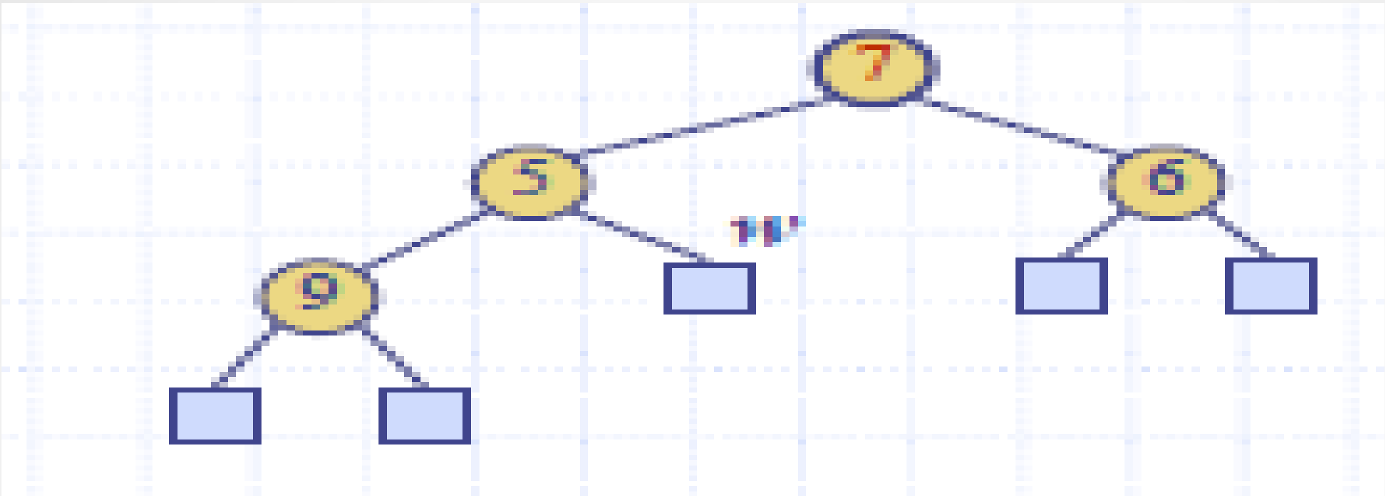


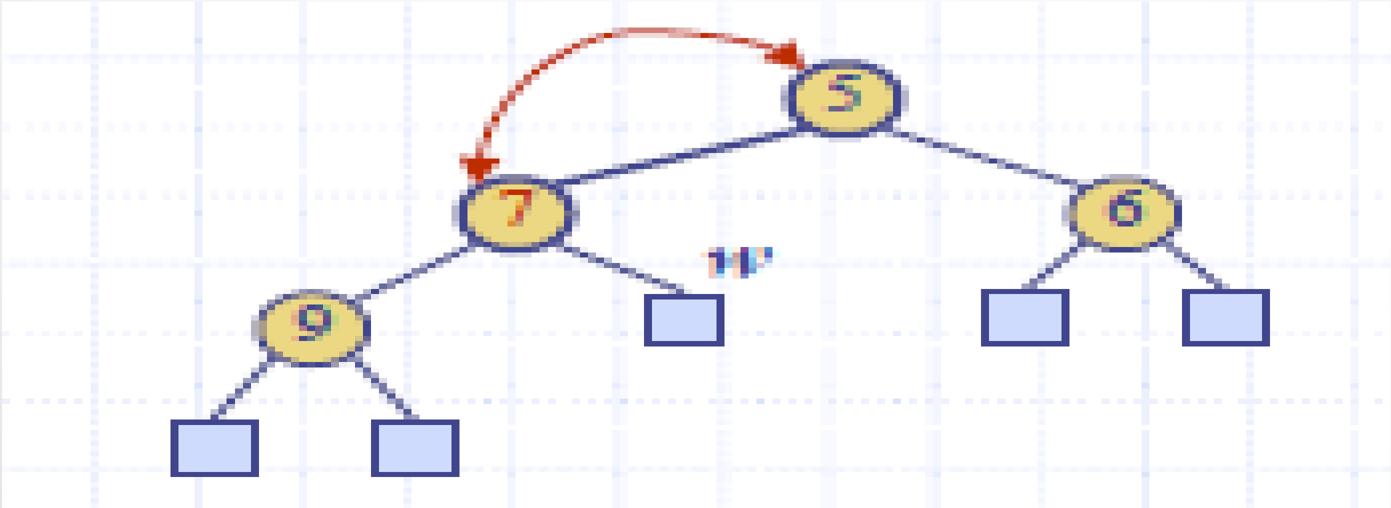



# Downheap

- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Upheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height O(log n), downheap runs in O(log n) time

# Downheap



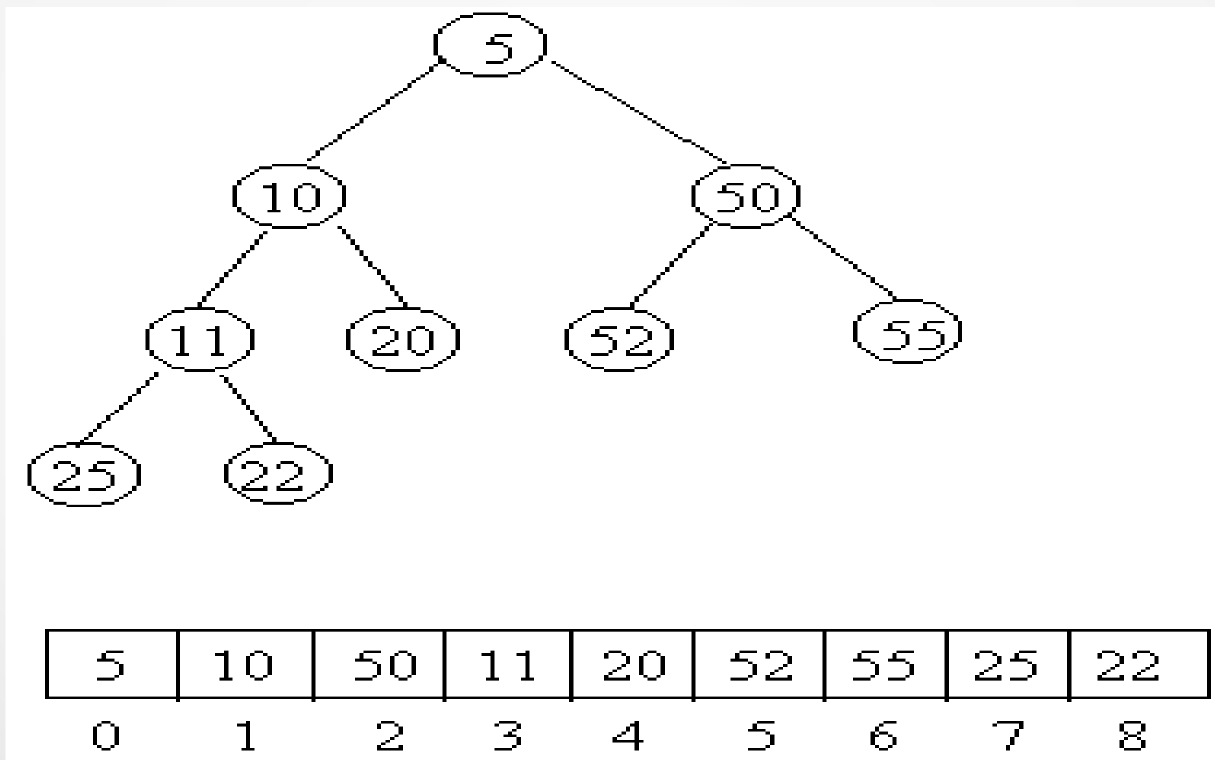



# Exercise

- Create a heap by inserting the following elements in order
  - 2,5,16,4,10,23,39,18,26,15, 9, 8
  - What is the height of the heap
  - Demonstrate the deletion operation
    - Remove min element thrice and demonstrate how the heap changes
- Is there a heap T storing seven distinct elements such that the preorder traversal of T yields the elements in sorted order?
  - What about the other traversals

# Heap Implementation

- Implemented using vector representation
- The last node is the rightmost node in the last level

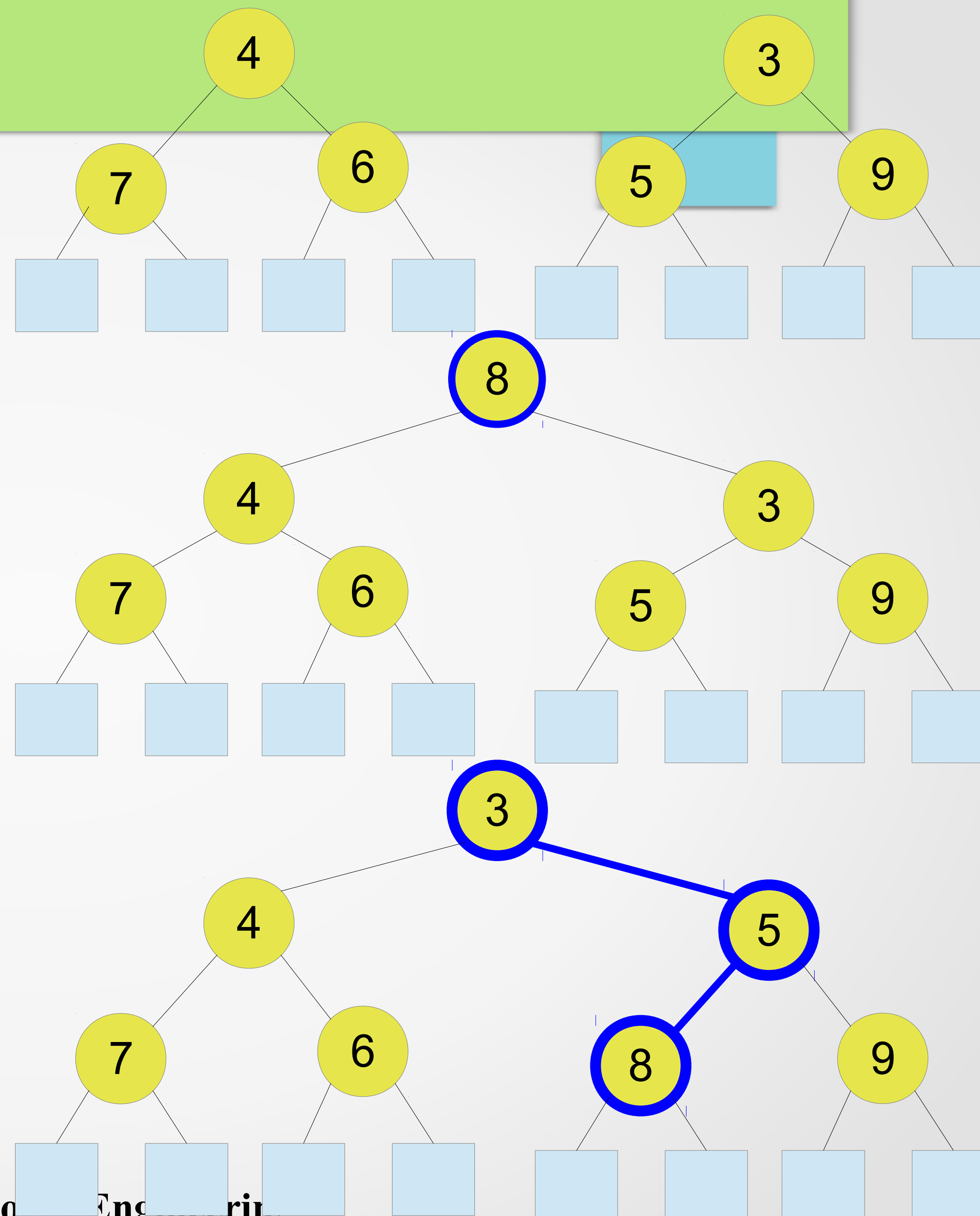


# Analysis of Heaps

- Insertion
  - Element inserted in the last position
  - Up-heap restores the heap-order property by swapping inserted element along an upward path from the insertion node
  - Worst case O(log n)
- Deletion
  - Remove root and replace with last node
  - Down-heap restores the heap-order property by swapping key k along a downward path from the root
  - Worst case O(log n)

# Merging heaps

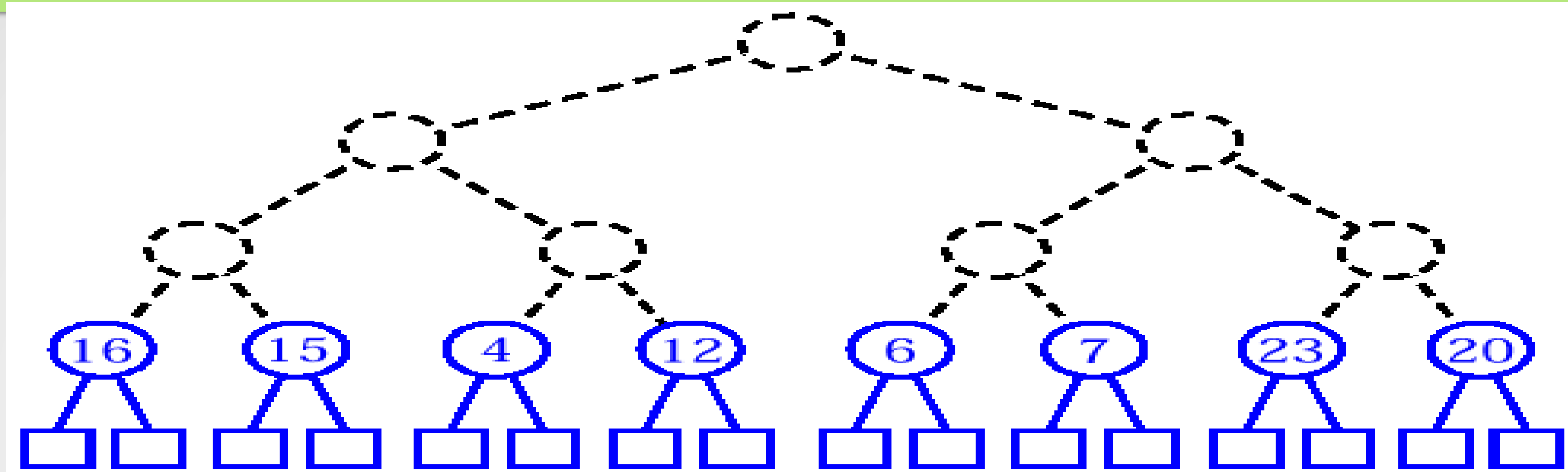
- Given two heaps and a key k
  - Create a new heap with k as root, and the two heaps as subtrees
  - Down-heap to restore heap order property

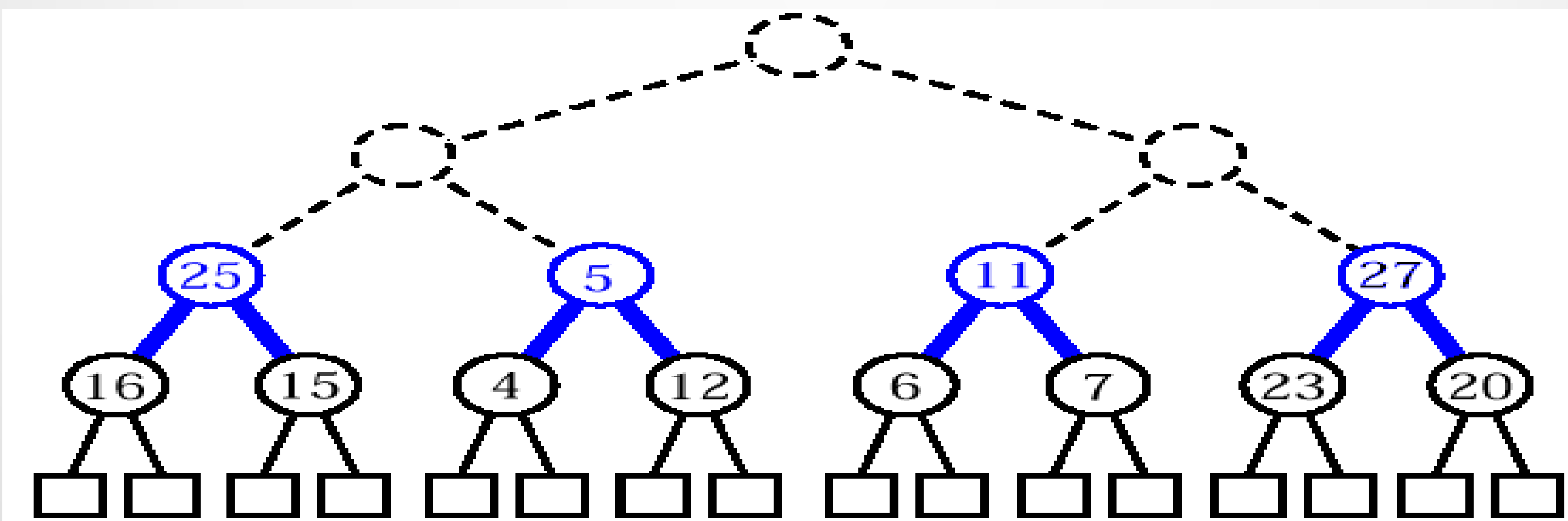


# Building the heap

- Bottom up building of the heap takes O(n) time
  - Construct (n+1)/2 elementary heaps composed of one key each.
  - Construct (n+1)/4 heaps, each with 3 keys, by joining pairs of elementary heads and adding a new key as the root.
    - Swap if heap-order not satisfied
  - In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys
  - i.e form $(n+1)/2^i$ heaps, each storing $2^i$-1 keys, by joining pairs of heaps storing $(2^{i+1}$-1) keys.

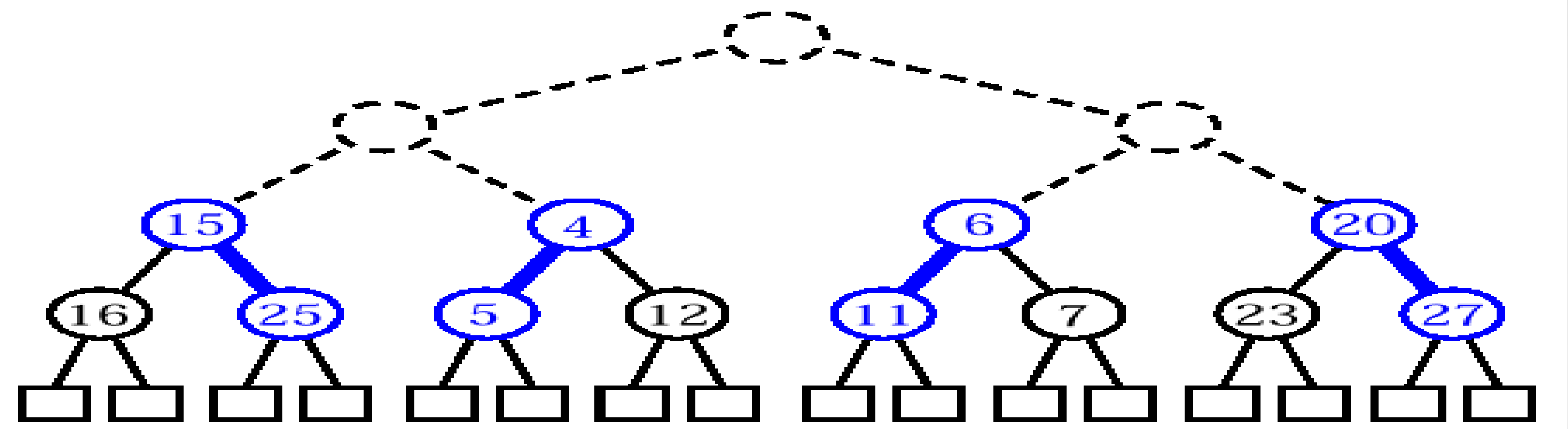
# Example contd

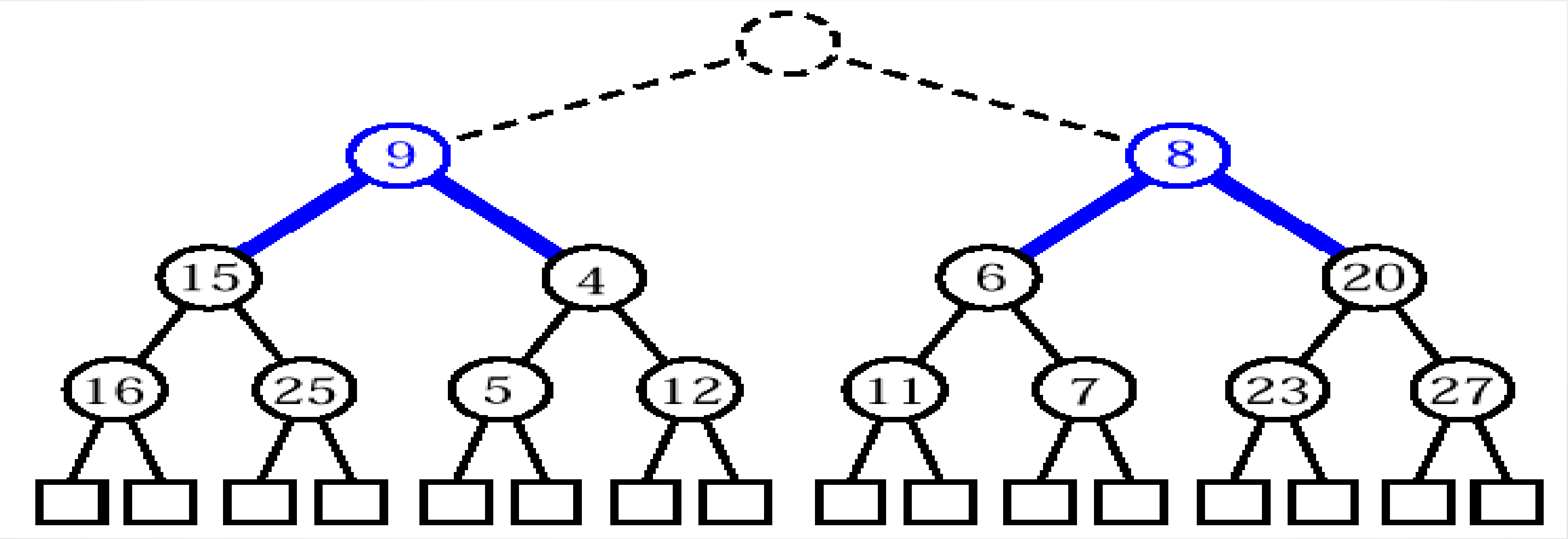



http://www.apl.jhu.edu/Classes/605202/felikson/lectures/L8/L8.html

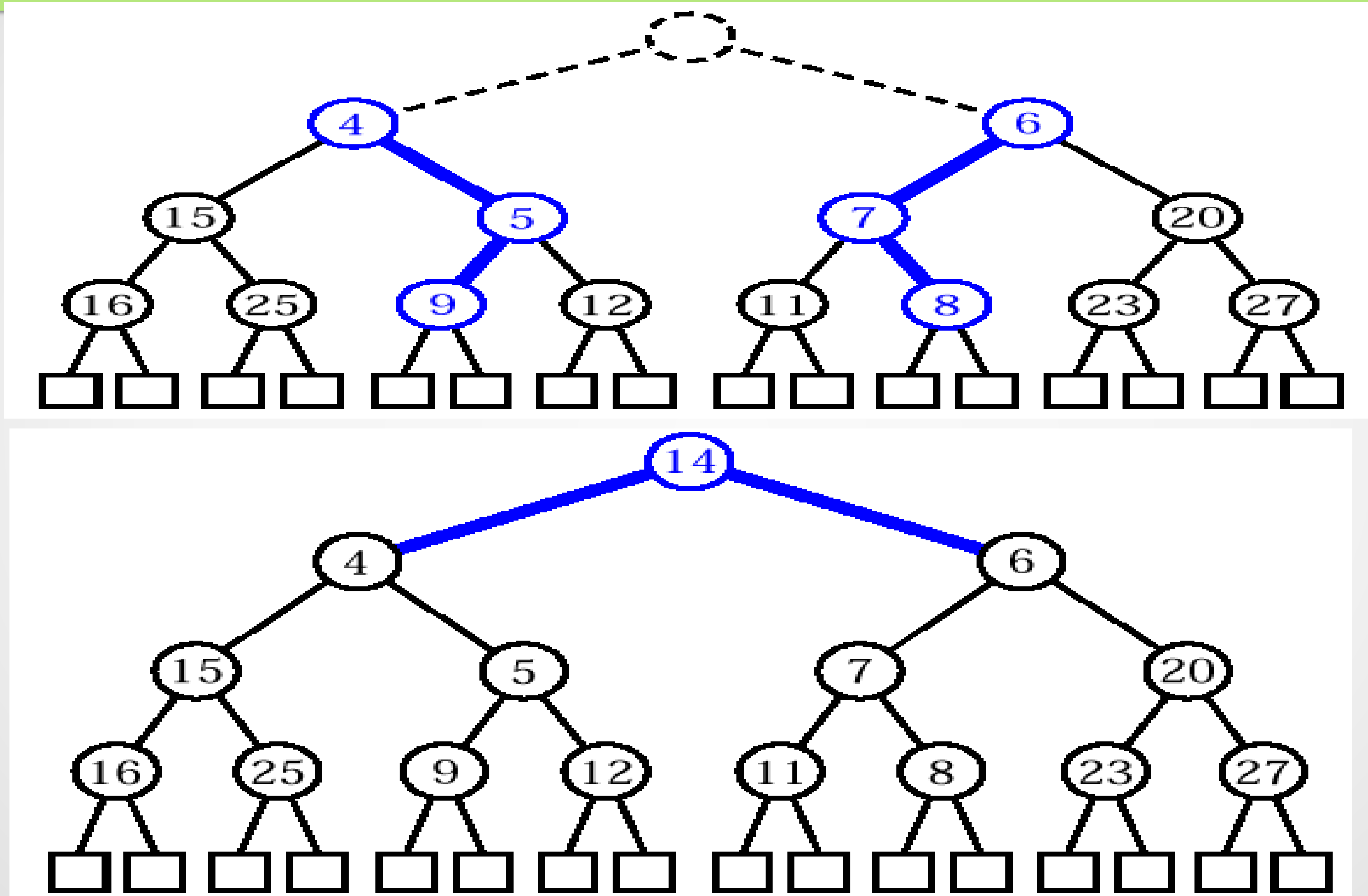
# Example contd



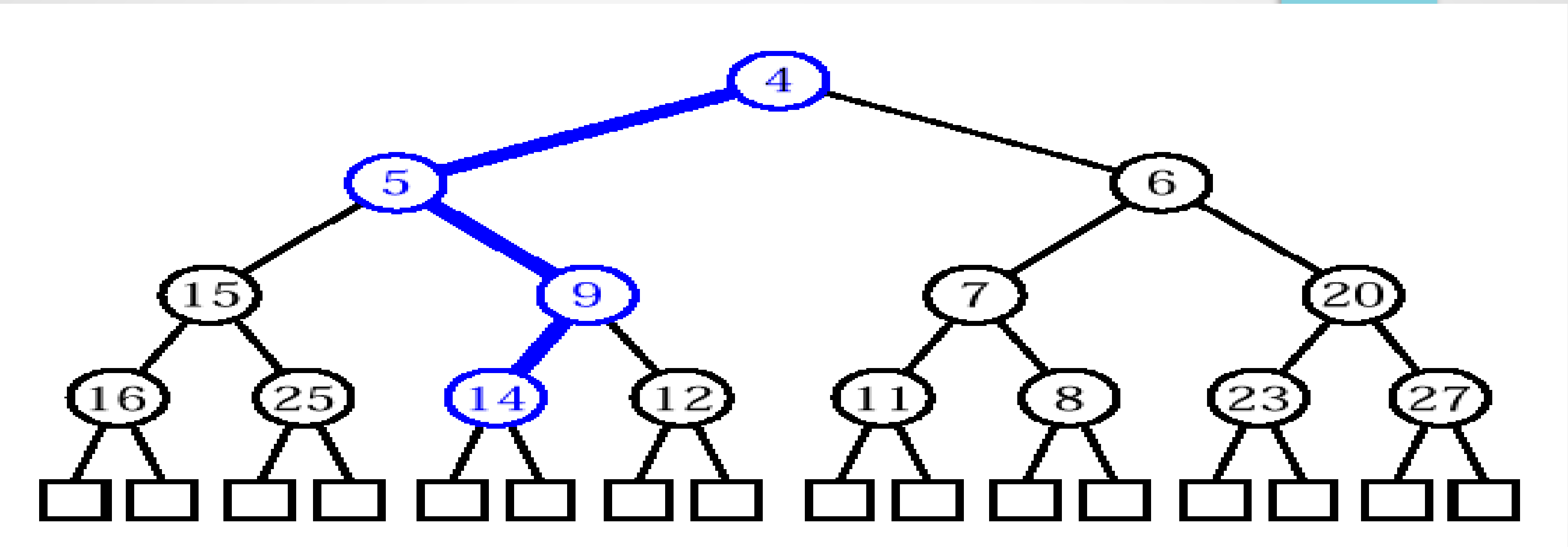

http://www.apl.jhu.edu/Classes/605202/felikson/lectures/L8/L8.html

# Example contd




http://www.apl.jhu.edu/Classes/605202/felikson/lectures/L8/L8.html

# Example contd




- Atmost n nodes in the path of down-heap
- Hence cost of heap building is O(n)

# Exercise

- Create a heap for the following data using the bottom-up approach
  - 2,5,16,4,10,23,39,18,26,15, 9, 8, 3, 22, 34
- Draw an example of a heap whose keys are all odd numbers from 1 to 59 (no repeat), such that the insertion of an item with key 32 causes up-heap bubbling to proceed all the way up to a child of the root
- Will the preorder traversal of a heap always yield the sorted order? Give an example to show it need not always be so.