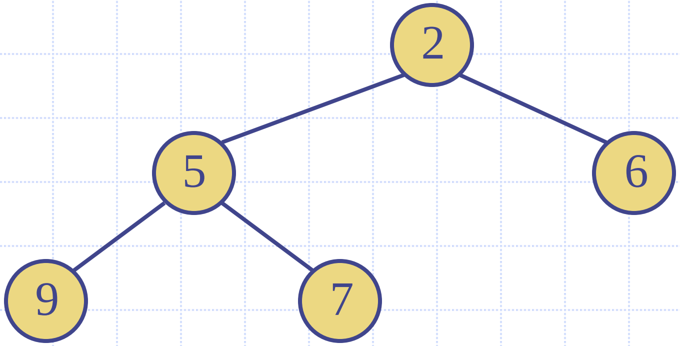
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Heaps



Recall Priority Queue ADT

- A priority queue stores a collection of entries
- Each entry is a pair (key, value)
- Main methods of the Priority Queue ADT
  - insert(k, v)
    inserts an entry with key k and value v
  - removeMin()
    removes and returns the entry with smallest key
- Additional methods
  - min()
    returns, but does not remove, an entry with smallest key
  - size(), isEmpty()
- Applications:
  - Standby flyers
  - Auctions
  - Stock market
  - …

Recall PQ Sorting

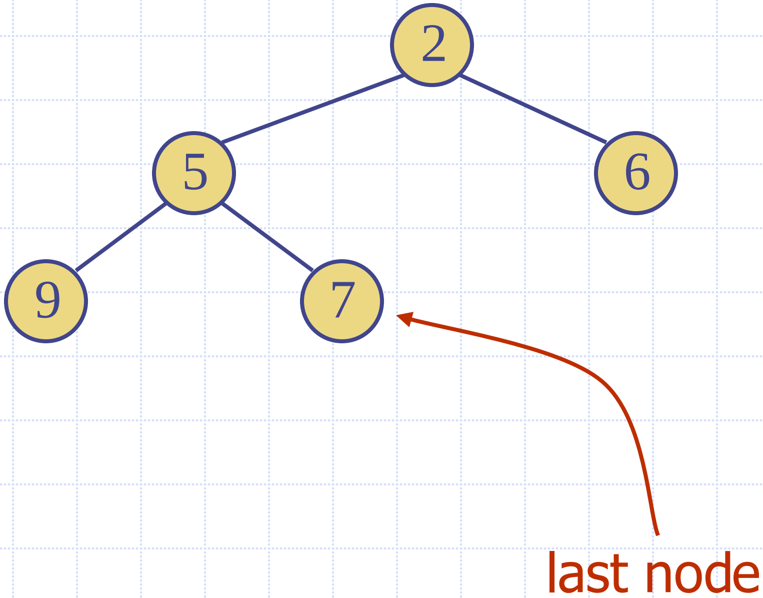
- We use a priority queue
  - Insert the elements with a series of insert operations
  - Remove the elements in sorted order with a series of removeMin operations
- The running time depends on the priority queue implementation:
  - Unsorted sequence gives selection-sort: $O(n^2)$ time
  - Sorted sequence gives insertion-sort: $O(n^2)$ time
- Can we do better?

Algorithm *PQ-Sort*(*S*, *C*)
  Input sequence *S*, comparator *C* for the elements of *S*
  Output sequence *S* sorted in increasing order according to *C*
  P ← priority queue with comparator *C*
  while ¬*S.isEmpty* ()
    e ← *S.remove* (*S. first* ())
    P.insert (*e*, *e*)
  while ¬*P.isEmpty*()
    e ← *P.removeMin*().*getKey*()
    S.addLast(*e*)

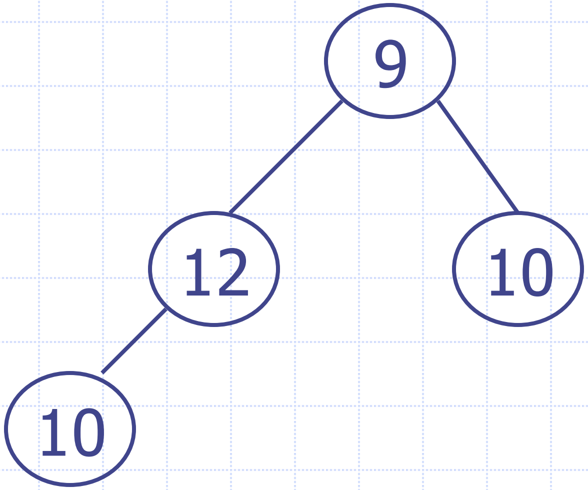
Heaps (Sec.9.3)

- A heap is a *binary tree* storing keys at its nodes and satisfying the following properties:
- Heap-Order: for every node v other than the root, $key(v) \geq key(parent(v))$
- Complete Binary Tree: let h be the height of the heap
  - levels $i = 0, \ldots, h - 1$ have the maximal number of nodes, i.e., there are 2^i nodes at depth i.
  - The remaining nodes at level/depth h reside in the leftmost possible positions at that level/depth.

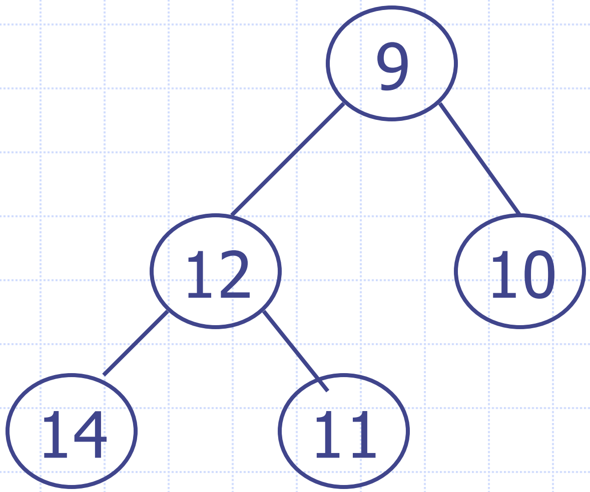
- The last node of a heap is the rightmost node of maximum depth



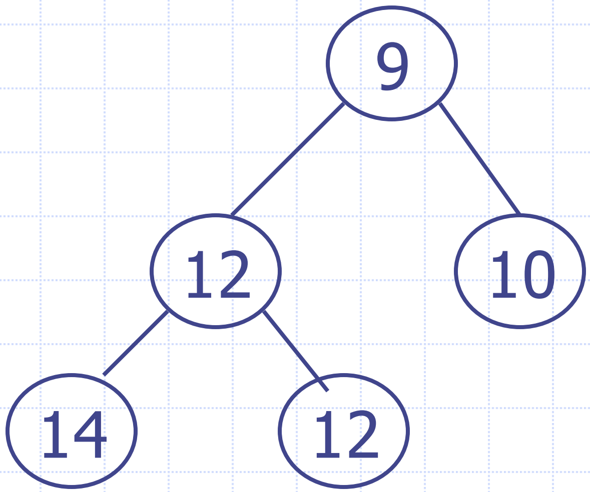
Exercise: is this a heap?



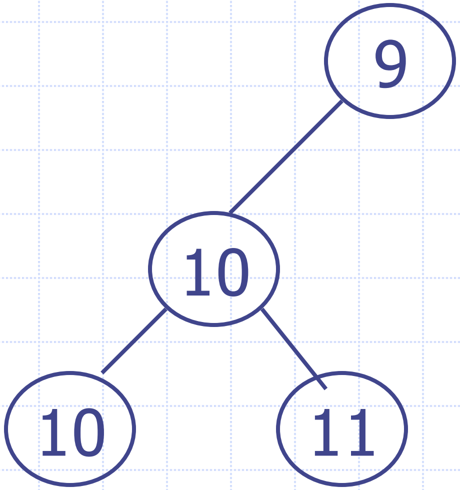
Exercise: is this a heap?



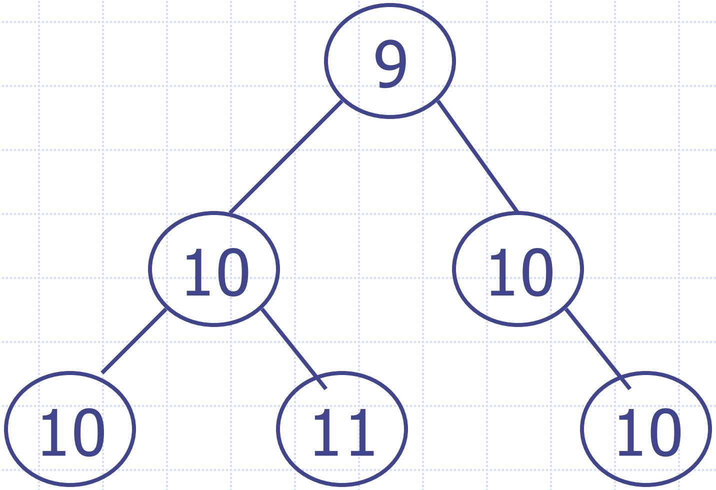
Exercise: is this a heap?



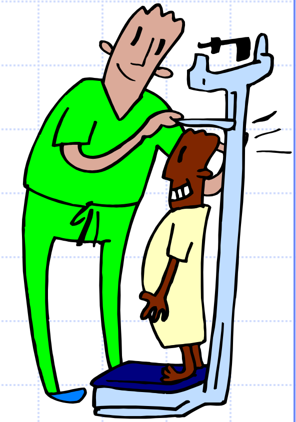
Exercise: is this a heap?



Exercise: is this a heap?

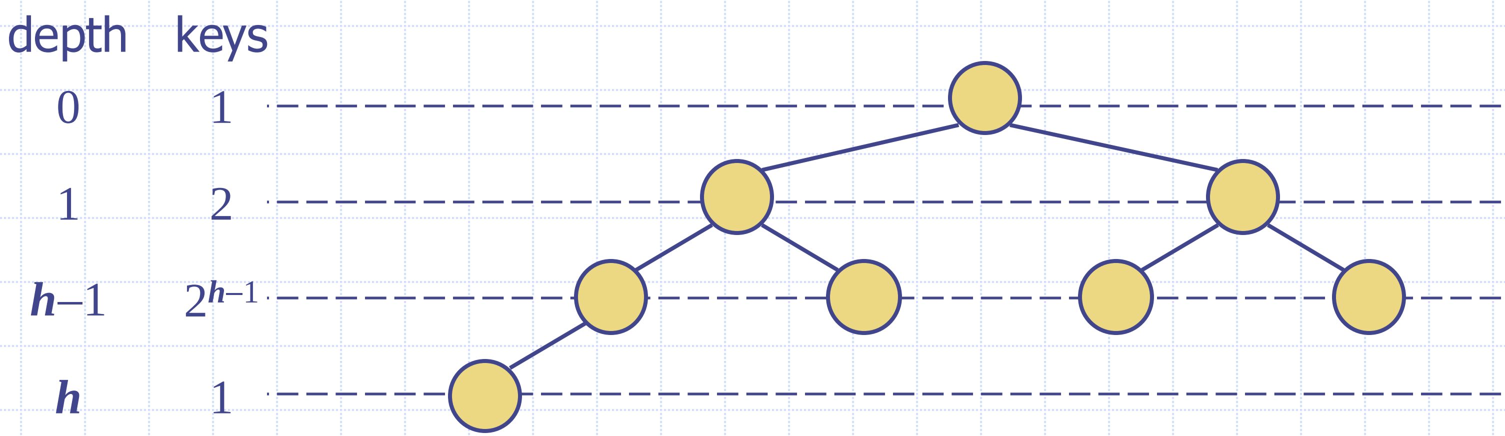


Height of a Heap



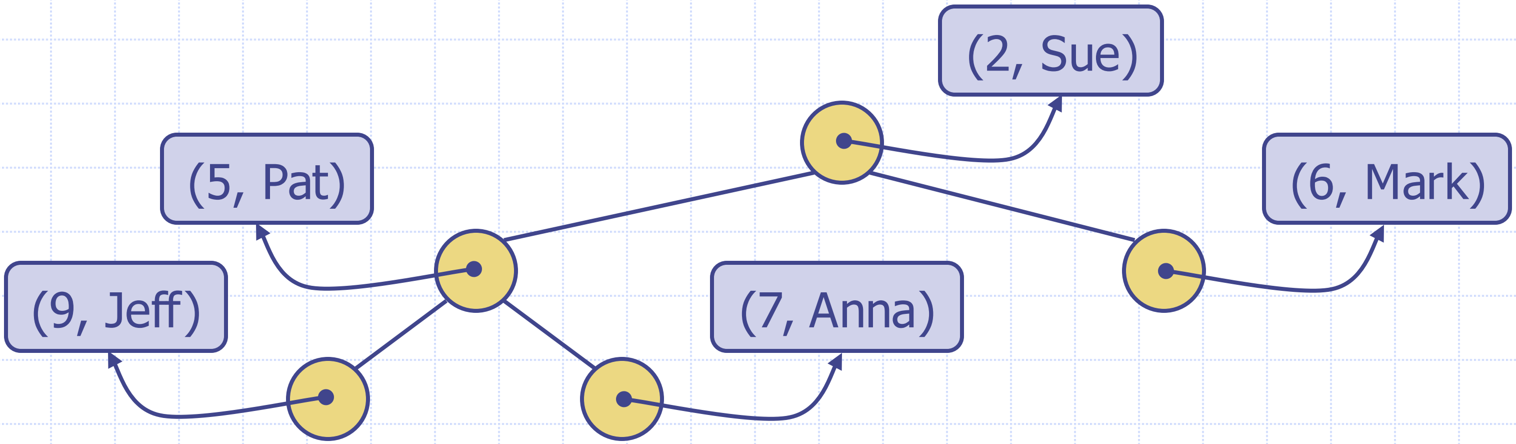
- **Theorem:** A heap storing n keys has height $O(\log n)$

  Proof: (we apply the complete binary tree property)
  - Let h be the height of a heap storing n keys
  - Since there are 2^i keys at depth $i = 0, \ldots, h-1$ and at least one key at depth h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2^h$, i.e., $h \leq \log n$



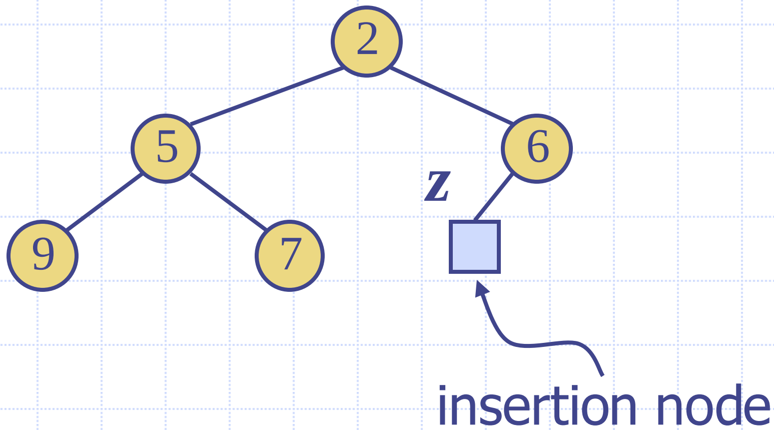
Heaps and Priority Queues

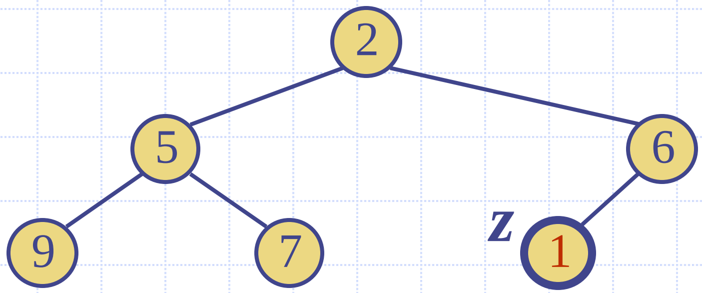
- We can use a heap to implement a priority queue
- We store a (key, element) item at each internal node
- We keep track of the position of the last node



Insertion into a Heap

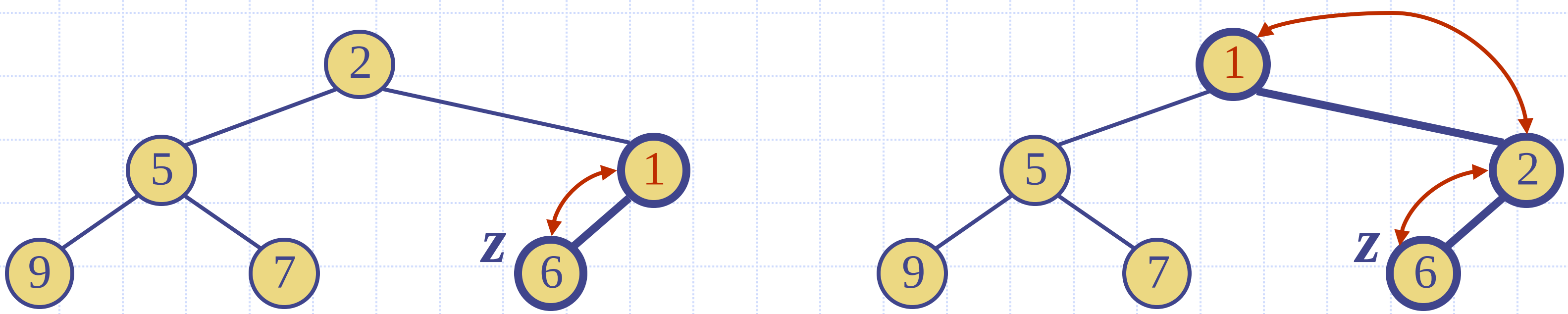
- Method *insertItem* of the priority queue ADT corresponds to the insertion of a key k to the heap.
- The insertion algorithm consists of three steps
  - Find the insertion node z (the new last node)
  - Store k at z
  - Restore the heap-order property (discussed next)





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.

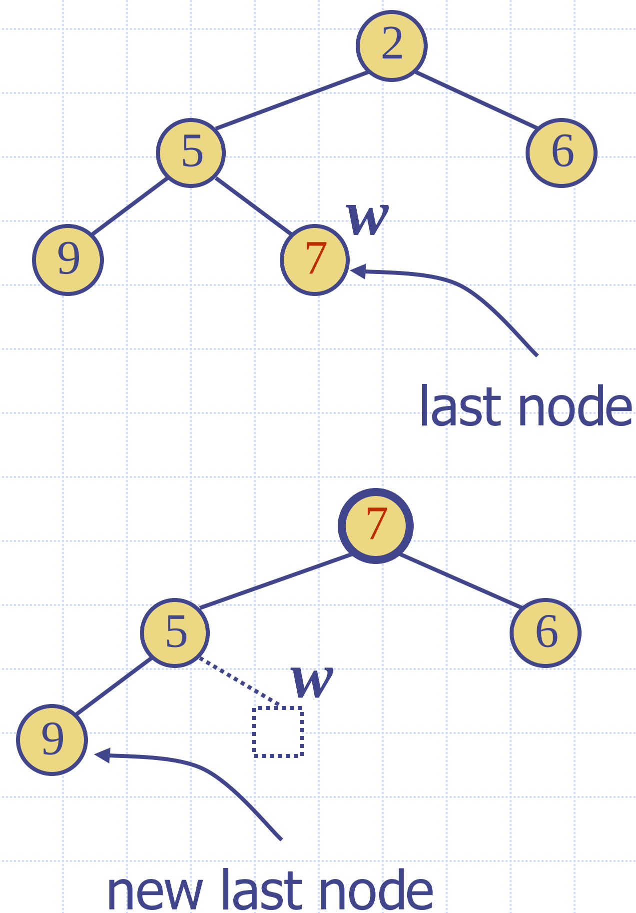


Exercise

- Convince yourself that the upheap method is correct.
- This is, after the upheap method is completed, the result is a heap.

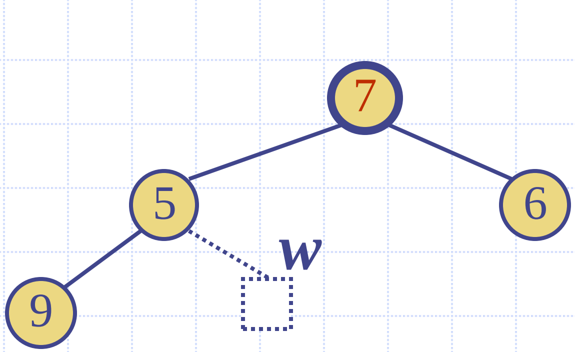
Removal from a Heap

- Method *removeMin* of the priority queue ADT corresponds to the removal of the root key from the heap.
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node w
  - Remove w
  - Restore the heap-order property (discussed next)



Downheap

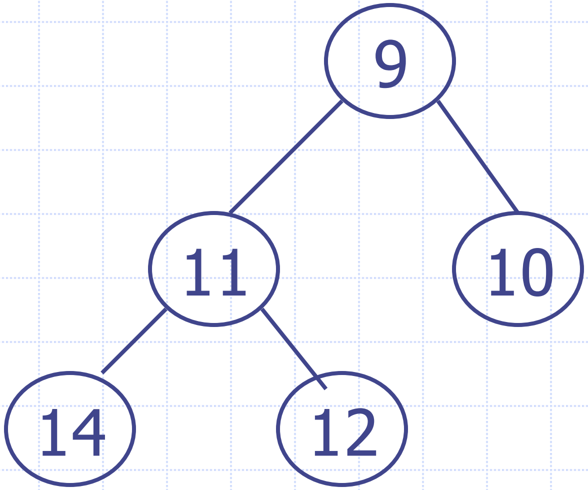
- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping the key k along a downward path from the root
- Downheap terminates when the key k reaches a leaf or a node whose children have keys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



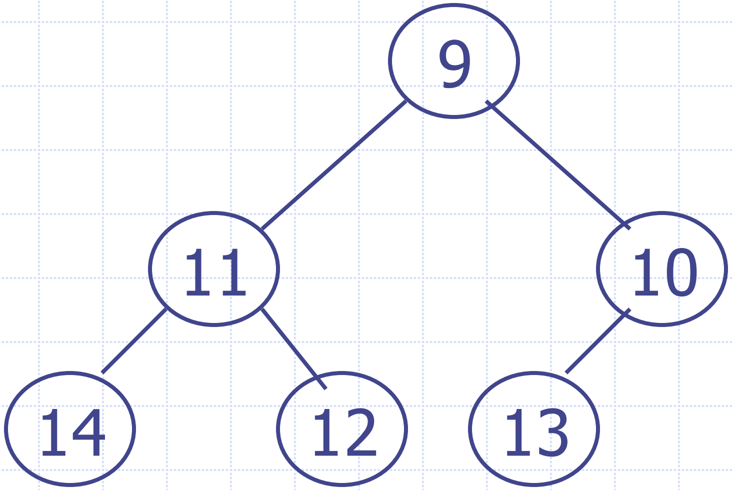
Exercise

- Convince yourself that the downheap method is correct.
- Do we need to scan the entire tree? Why?

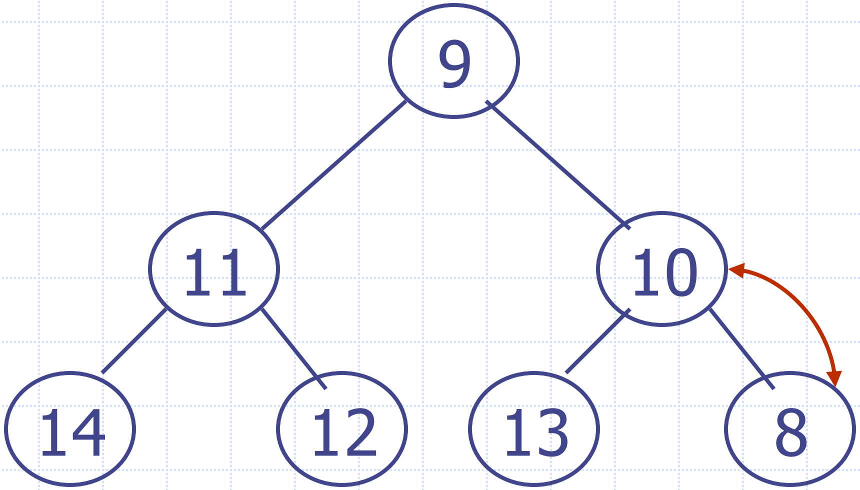
Exercise: insert 13, 8, 7



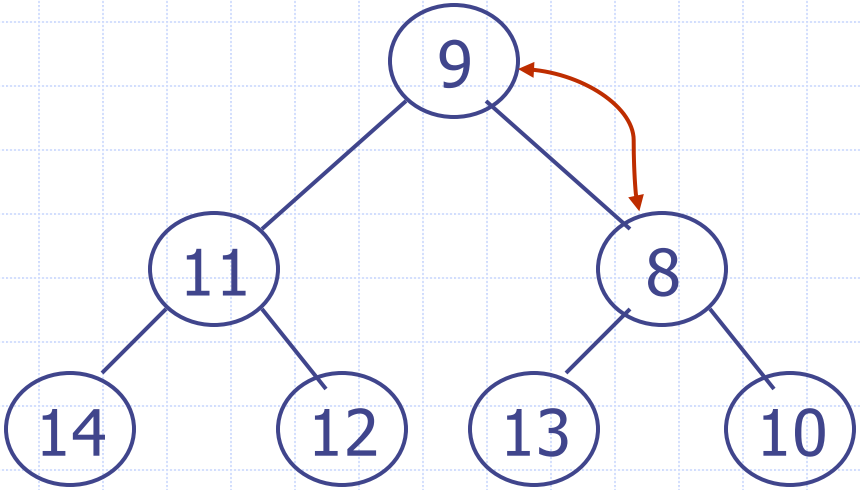
Exercise: insert 13



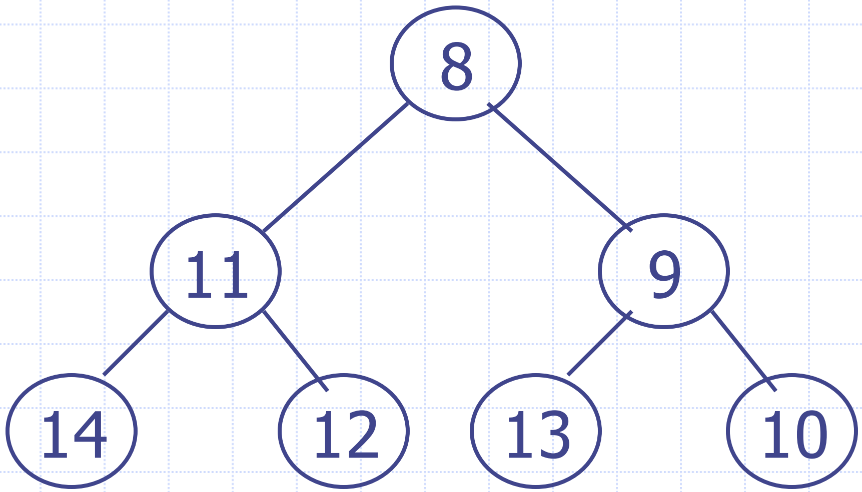
Exercise: insert 8



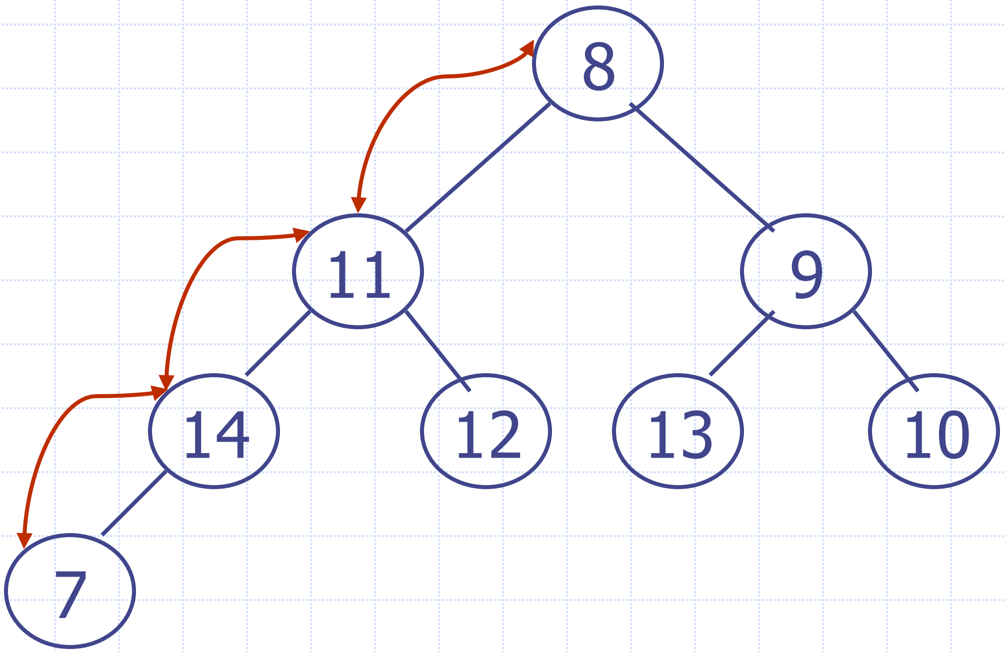
Exercise: insert 8



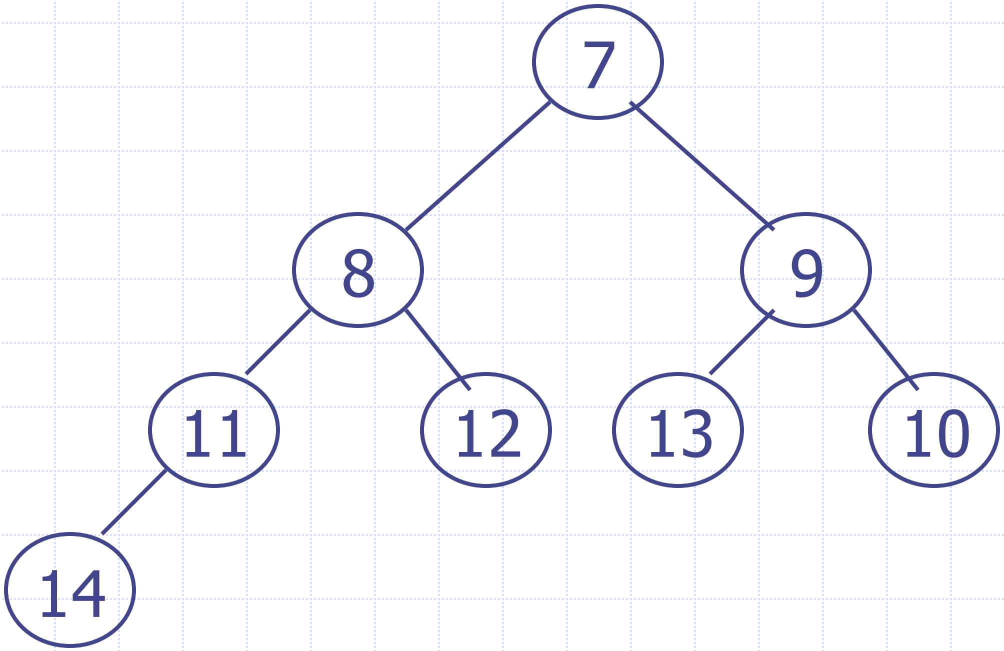
Exercise: insert 8



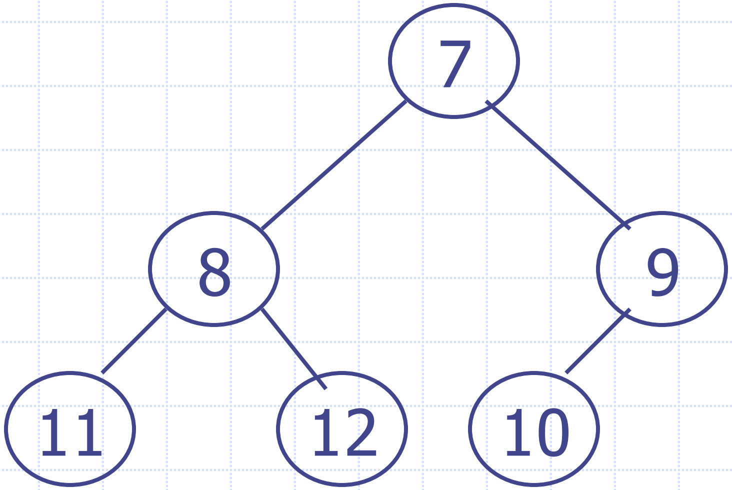
Exercise: insert 7



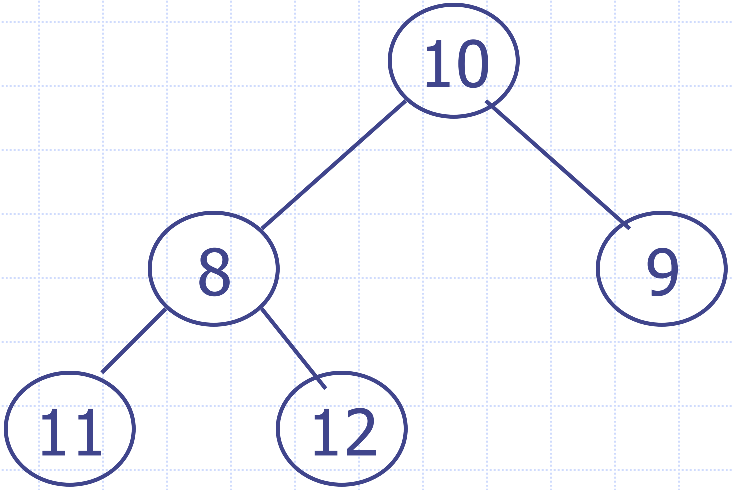
Exercise: insert 7



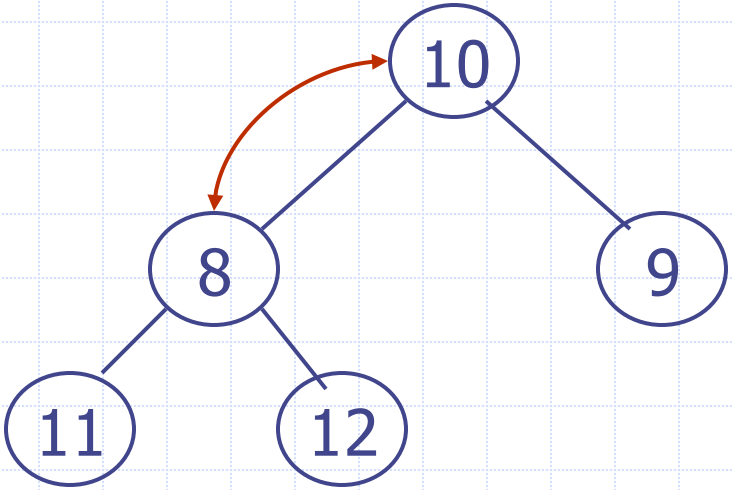
Exercise: remove 7



Exercise: remove 7

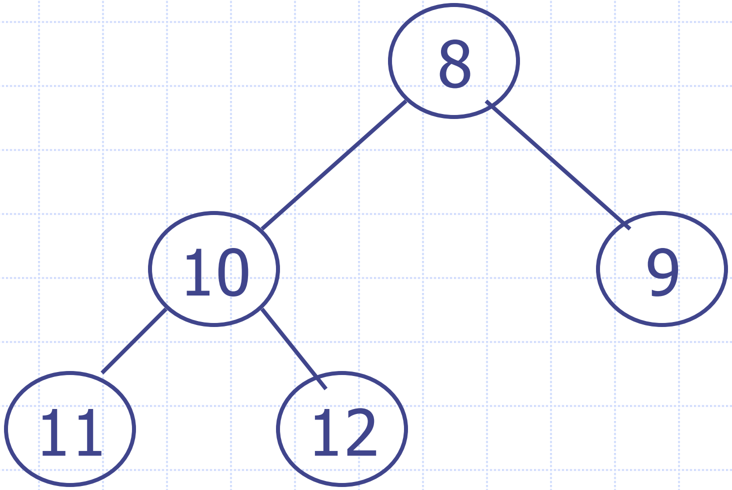


Exercise: remove 7



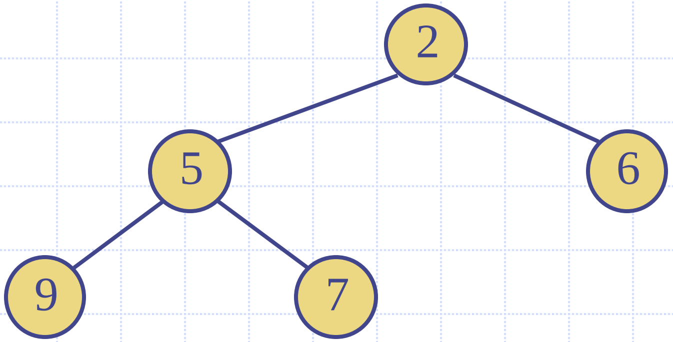
Always swap with the smaller child!

Exercise: remove 7



Array-based Heap Implementation

- We can represent a heap with n keys by means of a vector or ArrayList of length $n + 1$
- The cell of at index 0 is not used
- Links between nodes are not explicitly stored
- For the node at index i
  - the left child is at index $2i$
  - the right child is at index $2i + 1$
- Operation *insert* corresponds to inserting at index $n + 1$
- Operation *removeMin* corresponds to moving index n *to* index 1

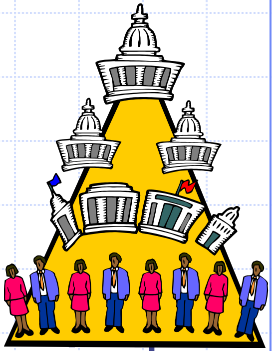


	2	5	6	9	7
0	1	2	3	4	5

Implementing Priority Queue with a Heap

- To create a priority queue, initialise a heap
- To insert in the priority queue, insert in the heap
- To get the value with the minimal key, ask for the value of the root of the heap
- To dequeue the highest priority item, remove the root and return the value stored there.

Heap-Sort



- Consider a priority queue with n items implemented by means of a heap
  - the space used is $O(n)$
  - methods insert and removeMin take $O(\log n)$ time
  - methods size, isEmpty, and min take time $O(1)$ time
- Using a heap-based priority queue, we can sort a sequence of n elements in $O(n \log n)$ time.
- The resulting algorithm is called *heap-sort*.
- Heap-sort is much faster than quadratic sorting algorithms, such as insertion-sort and selection-sort.

Conclusion

- Priority Queue ADT can be implemented using an unsorted list, a sorted list, or a heap.
- In the first two cases, one of the methods has to run in $O(n)$ time. For the heap implementation, all methods run in $O(\log n)$.