DS286 | 2016-11-04

L24: Heap & Priority Queue

Yogesh Simmhan
simmhan@cds.iisc.ac.in

Slides courtesy:
S223, EECS, WSU
CS233, CSE, POSTECH

Priority Queue

- Queues used to order items on a FIFO basis
  - Last item inserted is first item removed
  - Order of insertion forms the “priority”
- But cases where items in queue may need to be removed in different order, based on “importance”
  - e.g. boarding a flight by zone, process queue with priorities, shortest distance metric for vertices in SSSP
- Priority Queue
  - Each item in the queue has a value and a priority
  - Priorities have relative ordering, highest to lowest
  - When removing, item in queue with highest priority returned

Priority Queue ADT

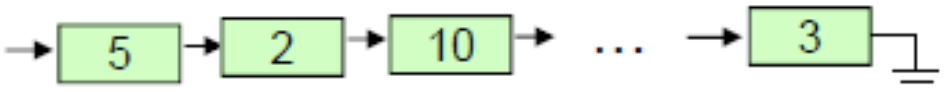
- void **insert**(item, priority)
  - Priority: 0 (high), 1, 2, ..., k (low)
  - Multiple items can have same priority
- item **remove**()
  - Returns the item with highest priority
  - smallest priority value [Min Priority Queue]
  - largest priority value [Max Priority Queue]
- Goal is to perform “fast” operations

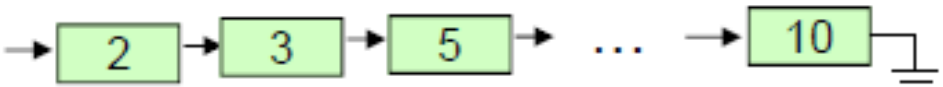
Priority Queue

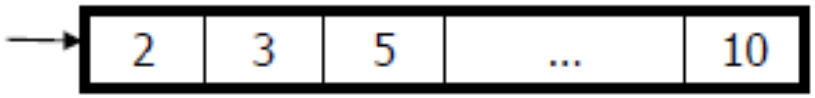
- Naïve solutions
- Sort items in queue every time you insert them to the queue
  - Ensures highest priority at the head
  - O(n.log n) for insert, O(1) for remove
- Sort items in queue just before you remove item from queue
  - Ensures highest priority at the head
  - O(n.log n) for insert, O(1) for remove…but avoid pre-sorting unecessarilly

Priority Queue

- Simple solutions
- Unordered linked list
  - O(1) insert
  - O(n) remove
- Ordered linked list
  - O(n) insert
  - O(1) remove
- Ordered array
  - O(lg n + n) insert
  - O(n) remove







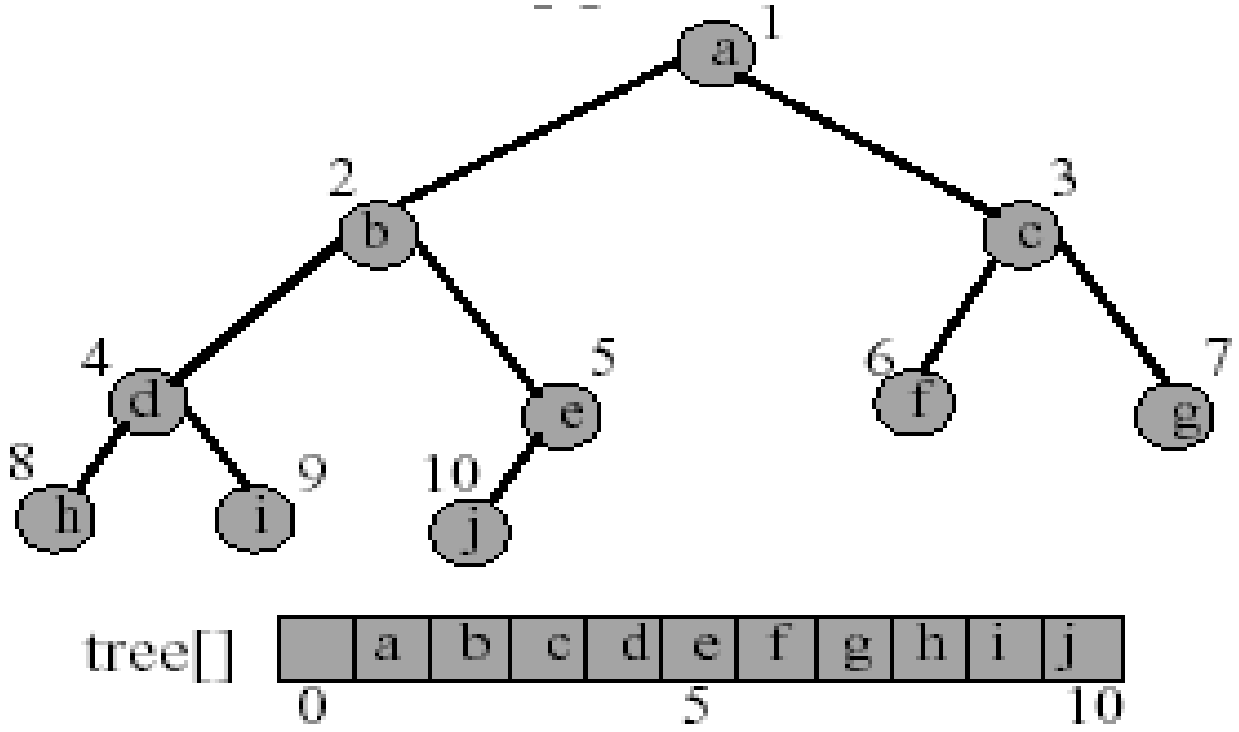
Binary Heap

- Used to implement priority queues
- Binary tree with two properties
  - Structure property
  - Heap order property

Structure Property

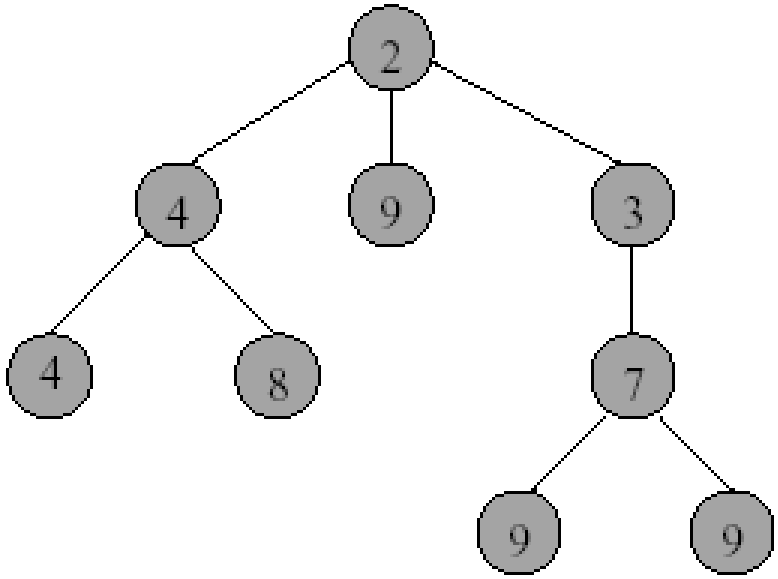
- A binary heap is a complete binary tree
  - Each level (except possibly the bottom most level) is completely filled
  - The bottom most level may be partially filled (from left to right)
- Height of a complete binary tree with N elements is `floor(log2N)`

Recall: Complete Binary Tree & Array Representation



Heap Order Property

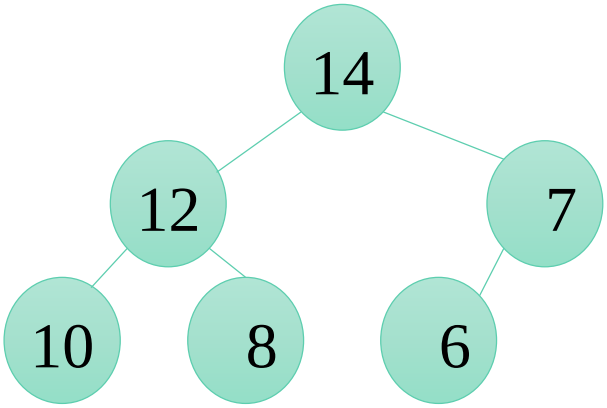
- A max tree (min tree) is a tree in which the value in each node is greater (less) than or equal to those in its children (if any)
  - Nodes of a max or min tree may have more than two children (i.e., may not be binary tree)

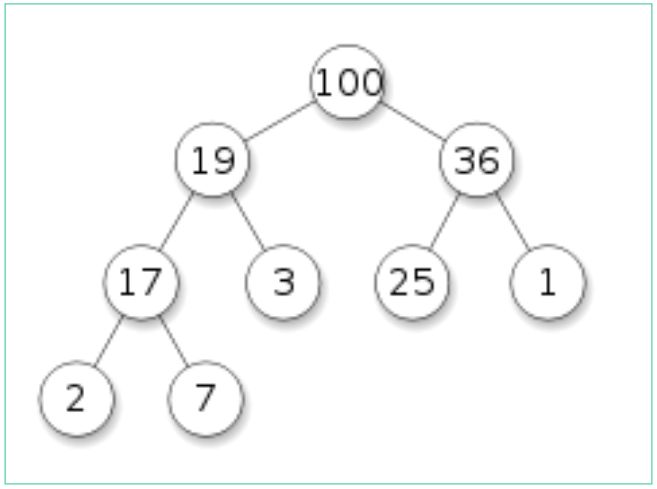


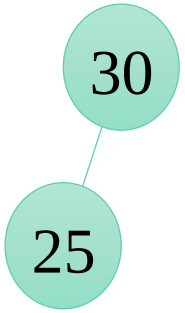
Heap Order Property

- Binary (Min) Heap is a Min Tree
- Heap-order property (for a "Min Heap")
  - For every node X, key(parent(X)) ≤ key(X)
  - Except root node, which has no parent
- Thus, minimum key always at root
  - Alternatively, for a "Max Heap", always keep the maximum key at the root
- **insert** & **remove** must maintain heap-order property
- Duplicates priority values are allowed
- No order implied for elements which do not share ancestor-descendant relationship

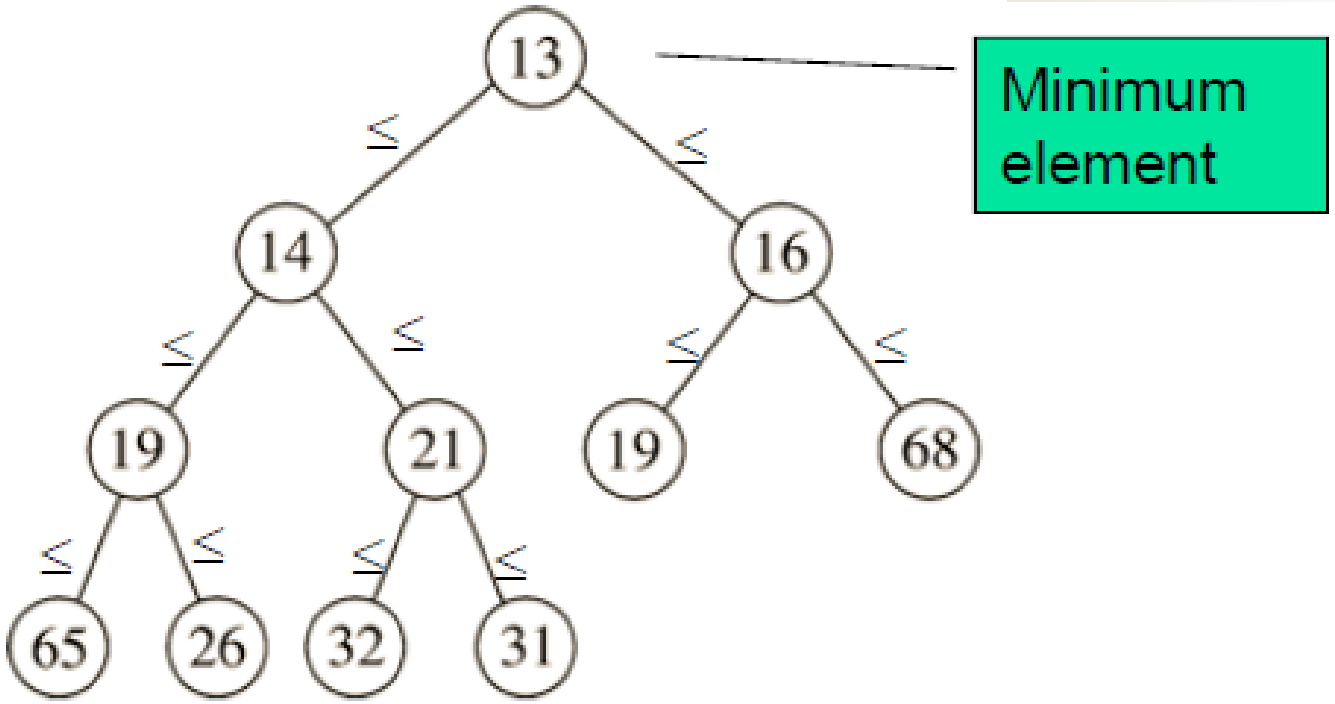
Binary (Max) Heap





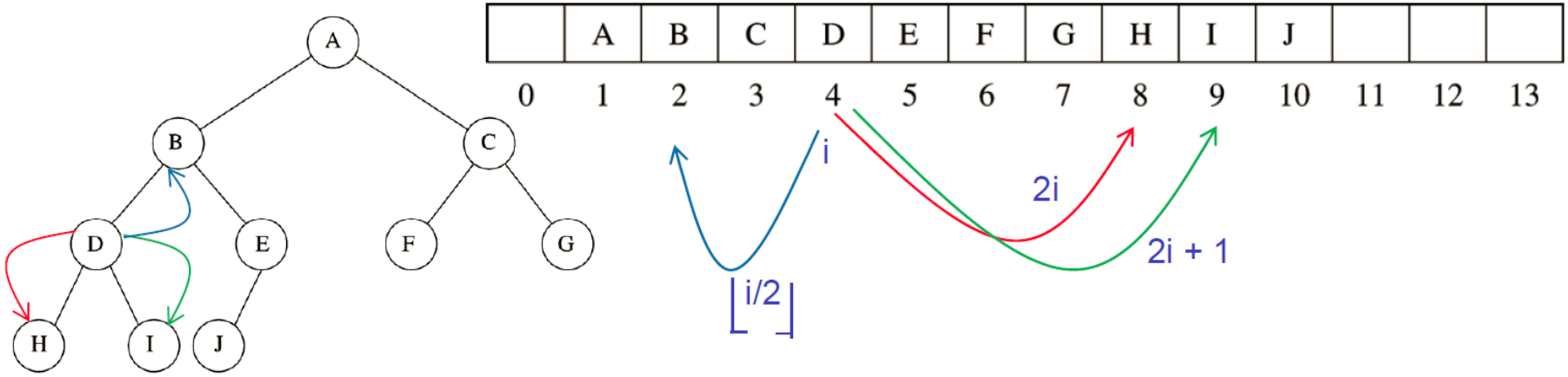


Binary (Min) Heap

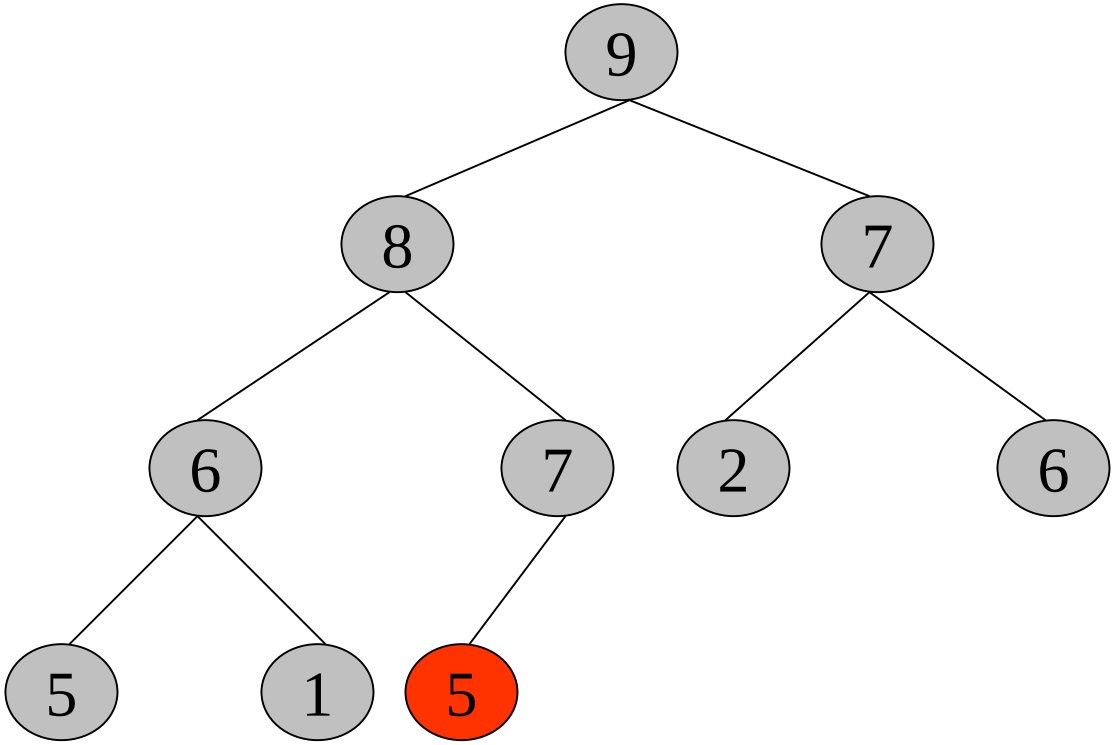


Heap Representation as Array

- Given element at position i in the array
  - Left child(i) = at position 2i
  - Right child(i) = at position 2i + 1
  - Parent(i) = at position floor(*i* / 2)



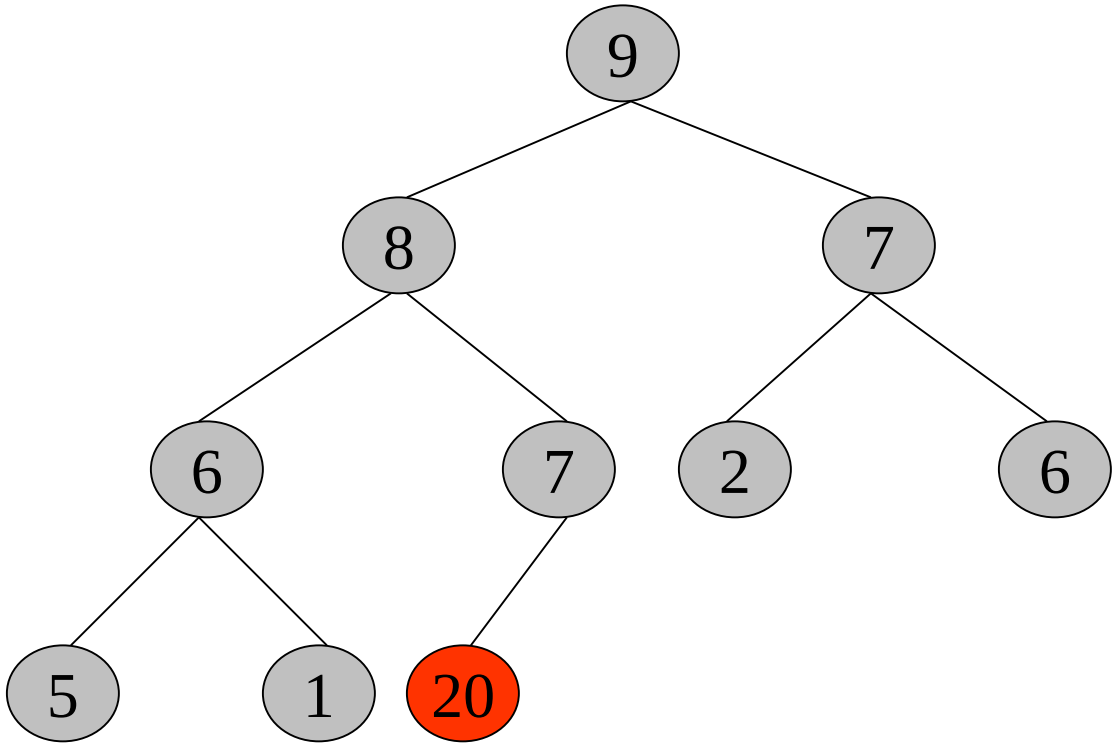
Max Heap Insert



Insert 5

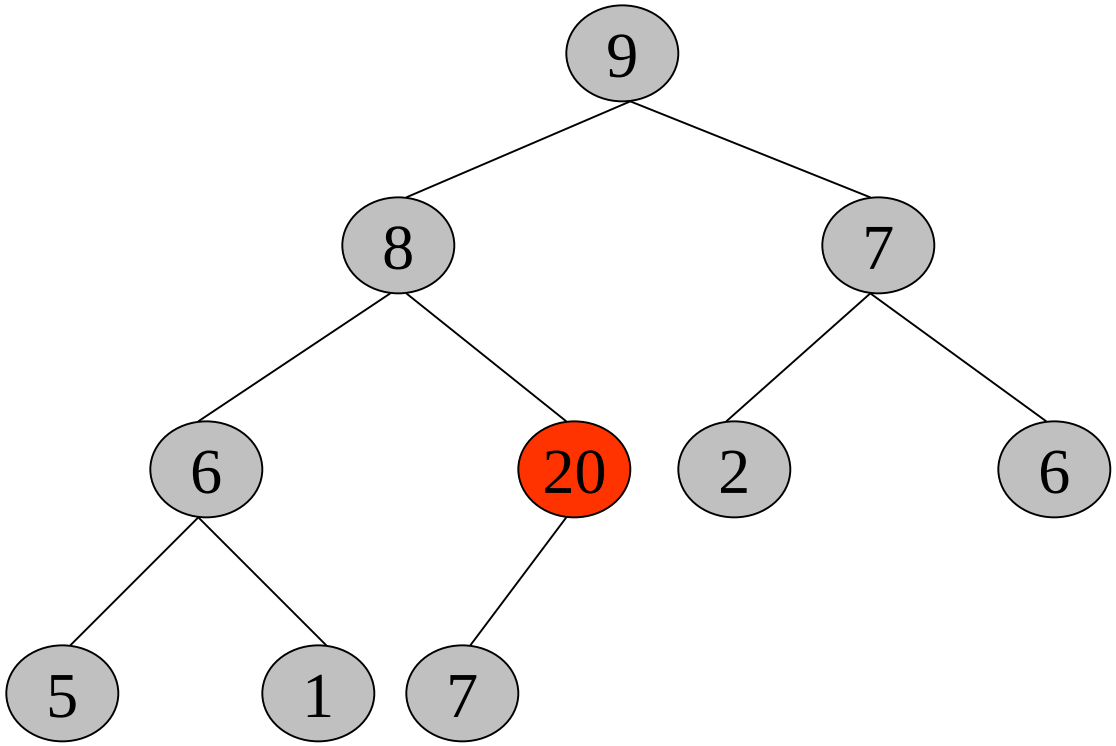
- Insert at next available slot, maintaining a complete tree
- "percolate up" to ensure heap order property

Max Heap Insert



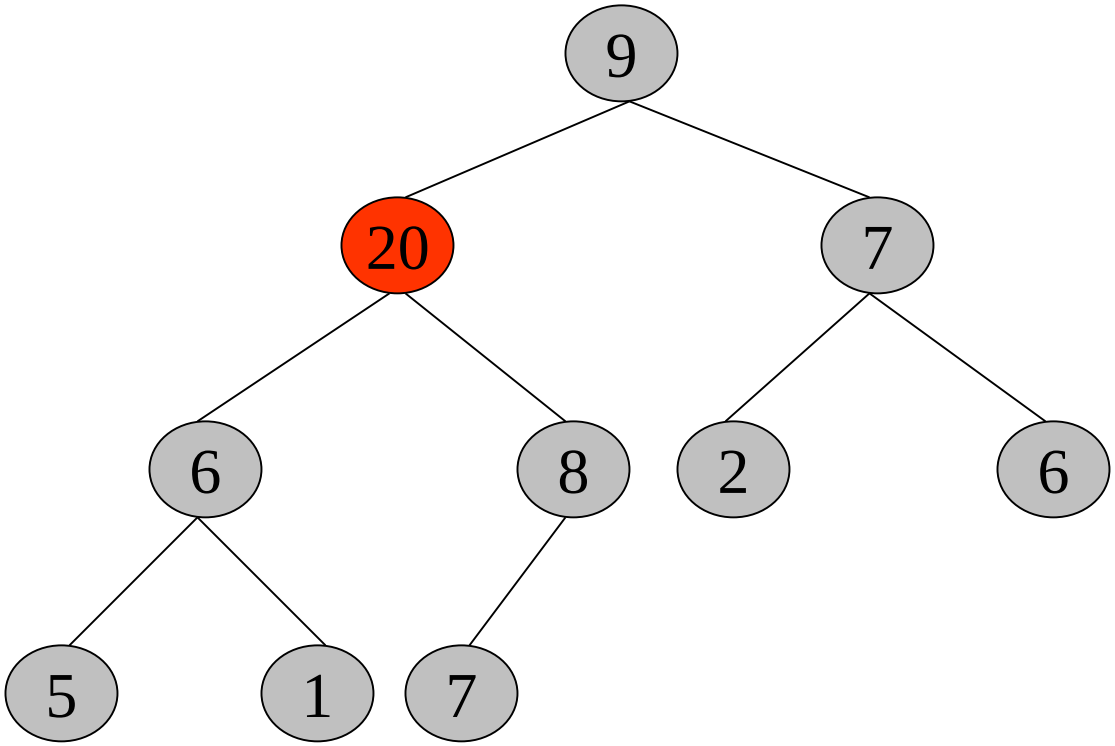
Insert 20

Max Heap Insert



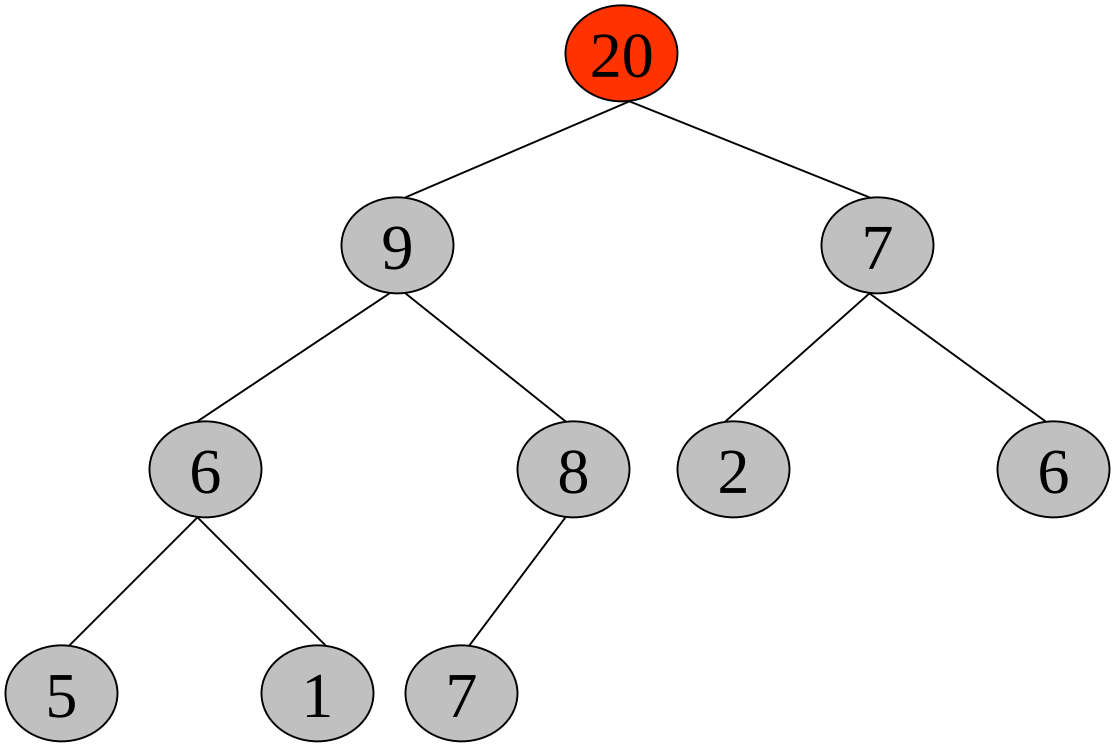
- Percolate up
- Exchange the positions with 7

Max Heap Insert



- Percolate up
- Exchange the positions with 8

Max Heap Insert



Done! Structure and Heap-order property met

- Percolate up
- Exchange the positions with 9

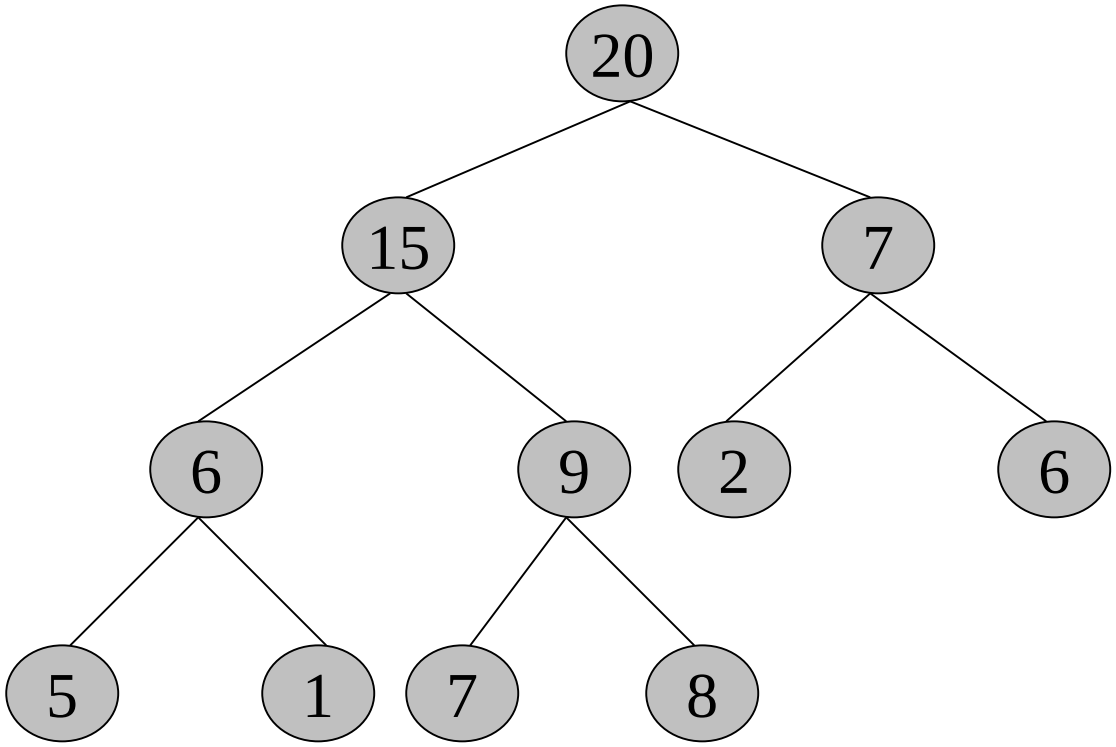
Insert Complexity

- At each level, we do O(1) work
- Thus the time complexity is **O(height) = O($log_2 n$)**, where n is the heap size

Max Heap Remove

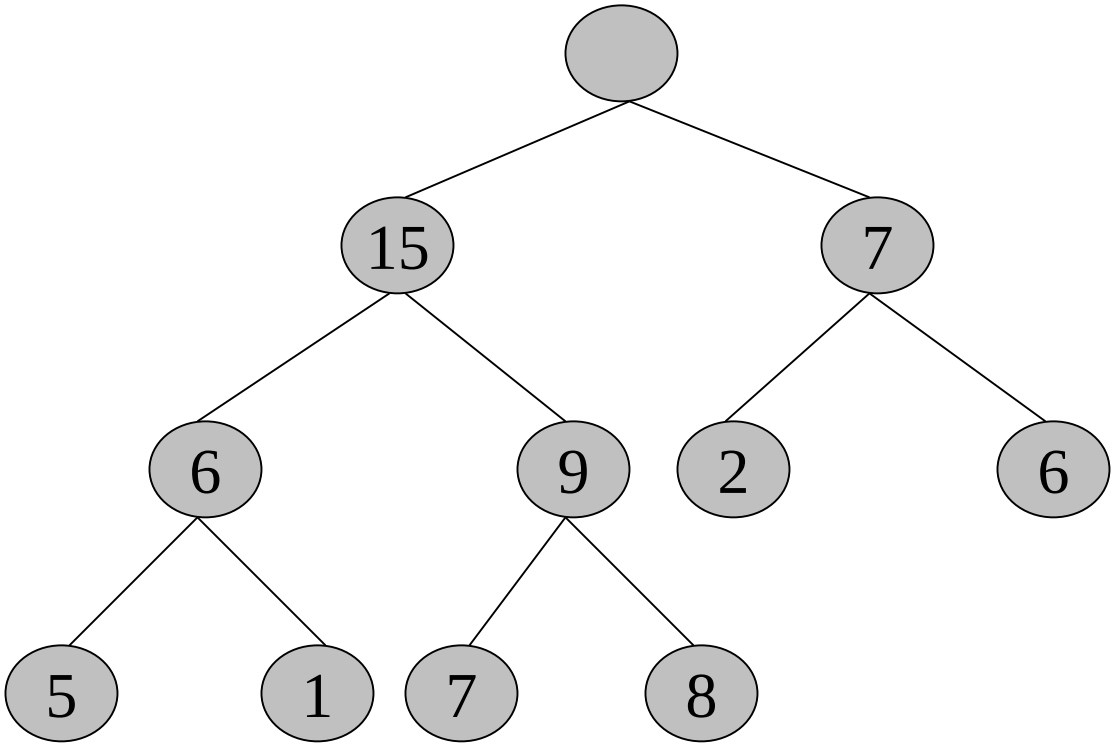
- Minimum element is always at the root
- Heap decreases by one in size
- Move last element into hole at root
- **Percolate down** while heap-order property not satisfied

Max Heap Remove



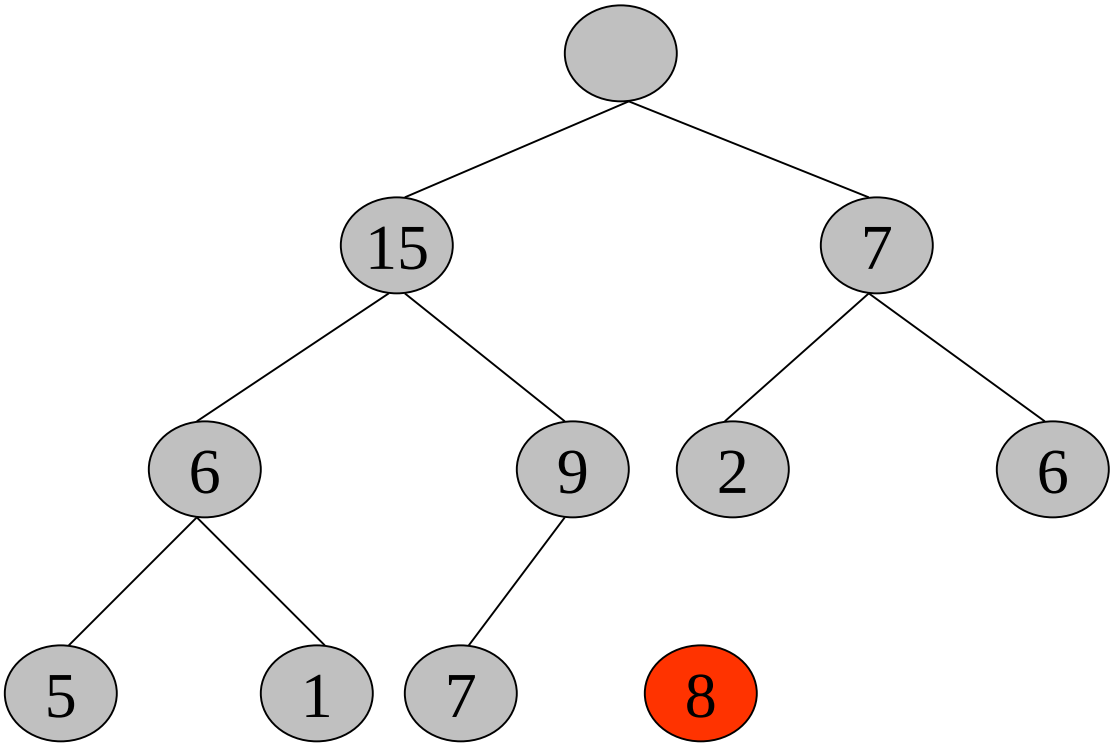
Remove root (Max item)

Max Heap Remove



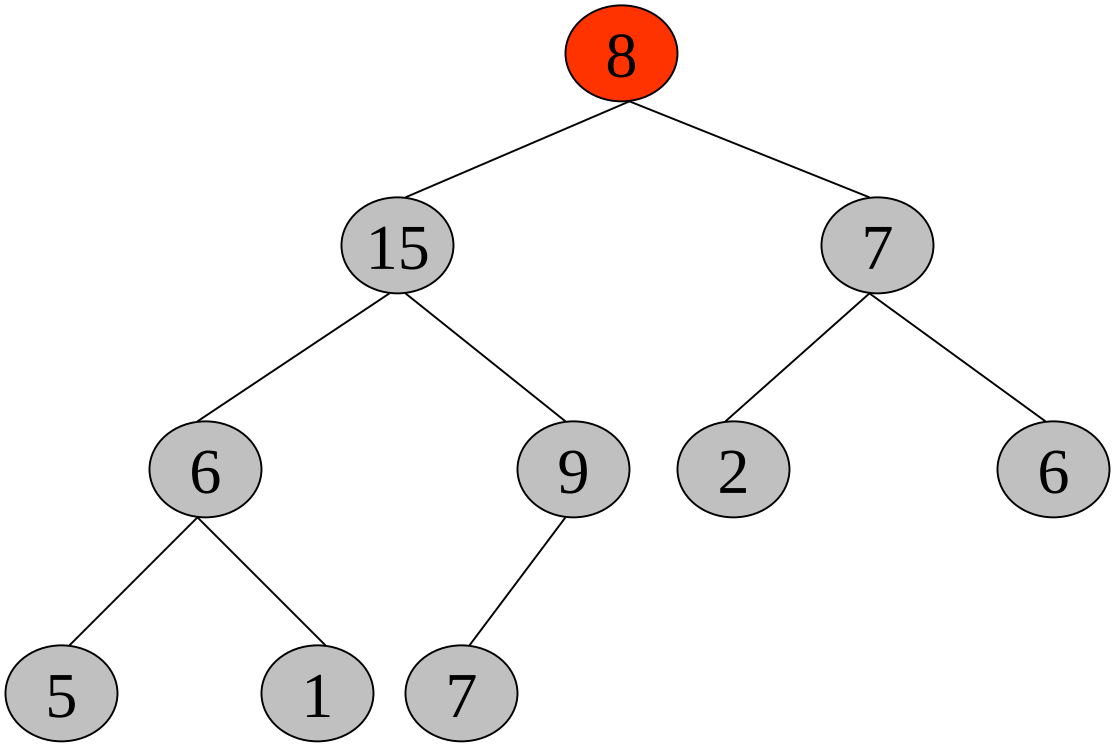
Hole in the root

Max Heap Remove



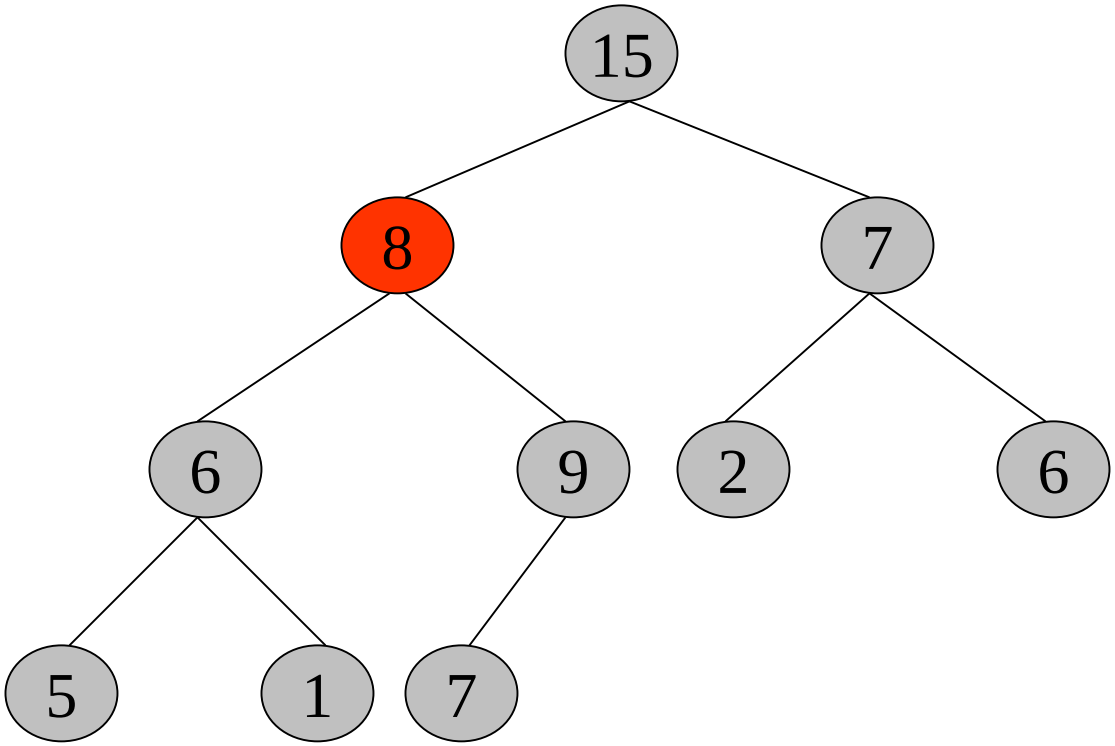
Remove last item in heap
Replace hole with last item

Max Heap Remove



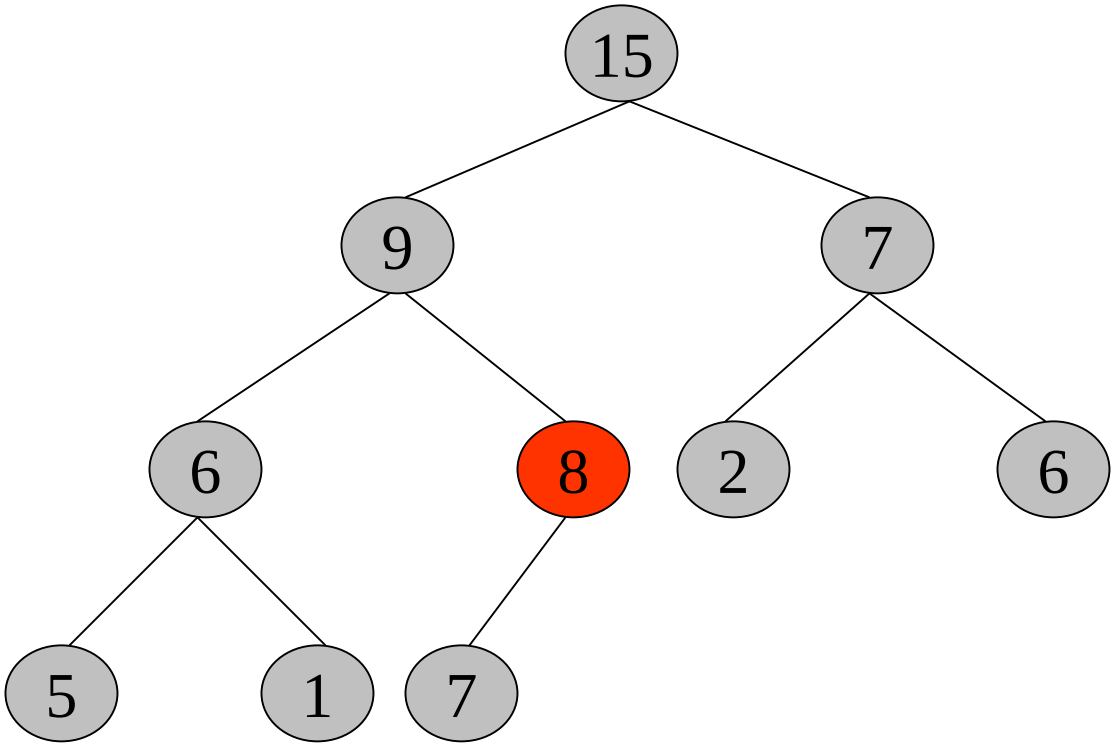
Replace hole with last item

Max Heap Remove



Percolate item down
Exchange item with 15

Max Heap Remove



Done! Structure and Heap-order property met

Percolate item down
Exchange item with 9

Remove Complexity

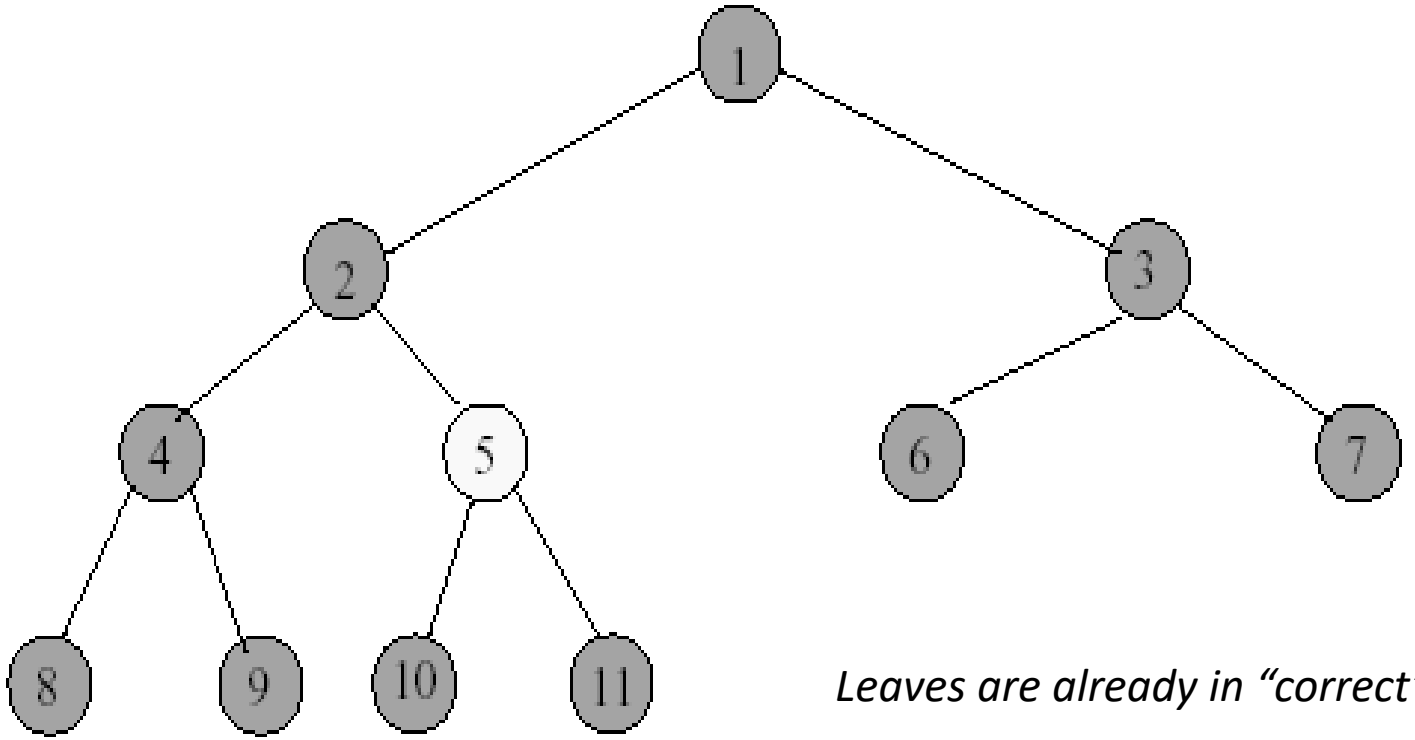
- The time complexity of deletion is the same as insertion
- At each level, we do O(1) work
- Thus the time complexity is **O(height) = O($log_2 n$)**, where *n* is the heap size

Max Heap Initialization

- Construct a new heap from an array of items
- Example: input array = [-,1,2,3,4,5,6,7,8,9,10,11]
- Default method takes O(N lg N) time
  - Perform N inserts
  - O(N log2 N) worst-case
- Can we improve this?
  - Start with given array as the complete tree
  - Adjust the tree (array) as necessary to meet heap order property

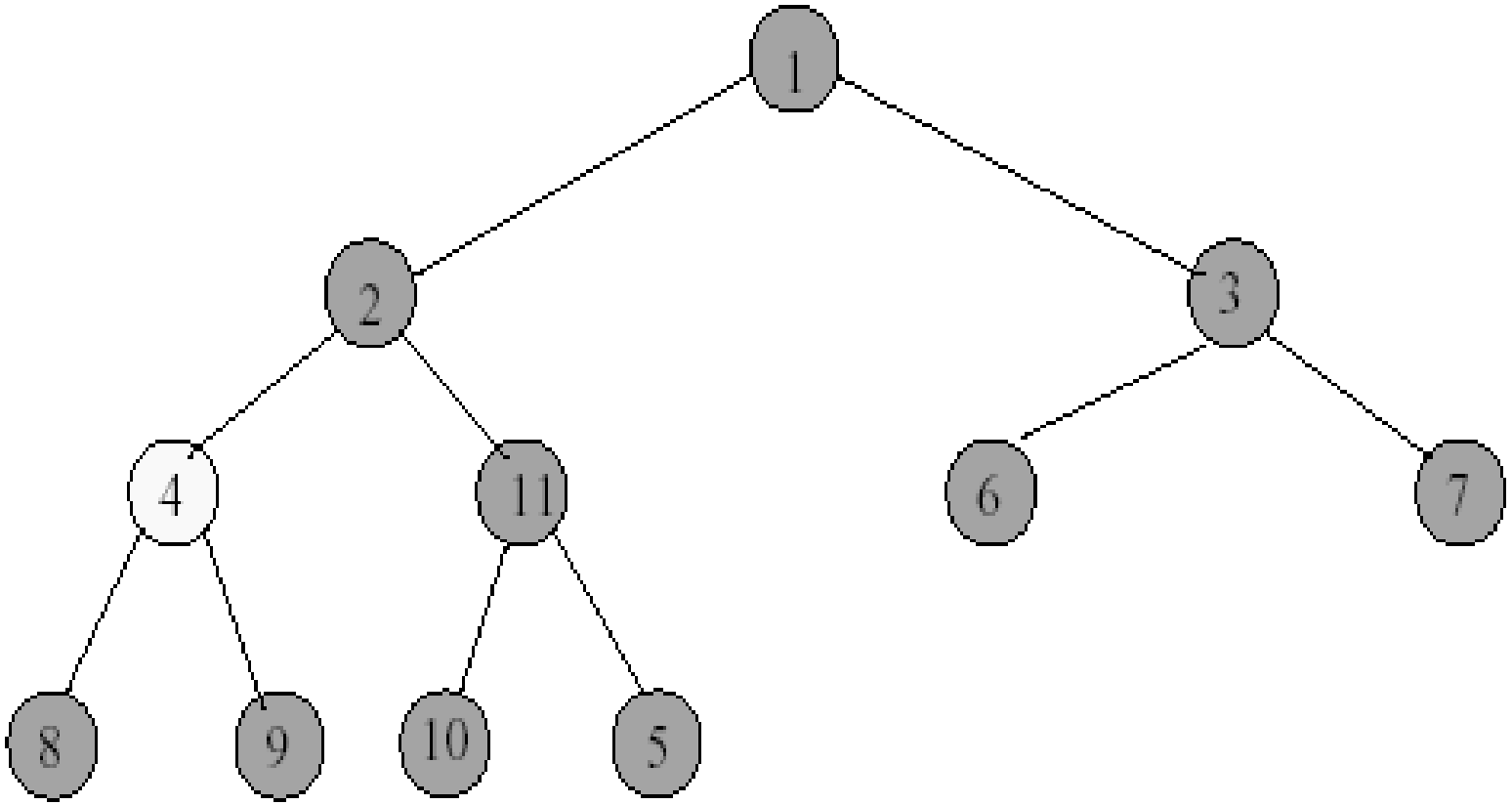
Max Heap Initialization

- Check each internal node from the bottom right
  - Start at rightmost array position having child, **floor(n/2)**
- Test children and swap to ensure heap order

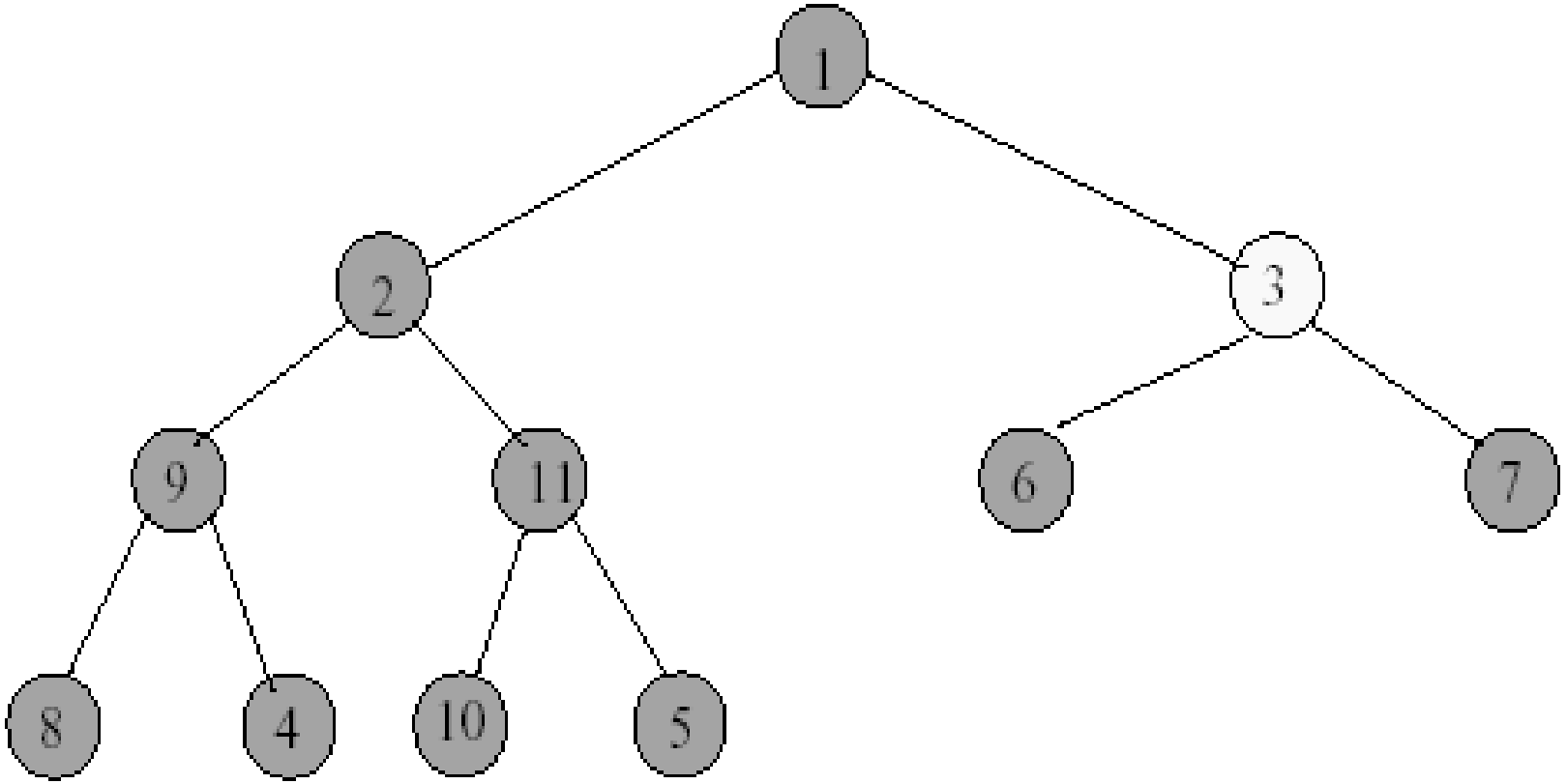


Leaves are already in "correct" position

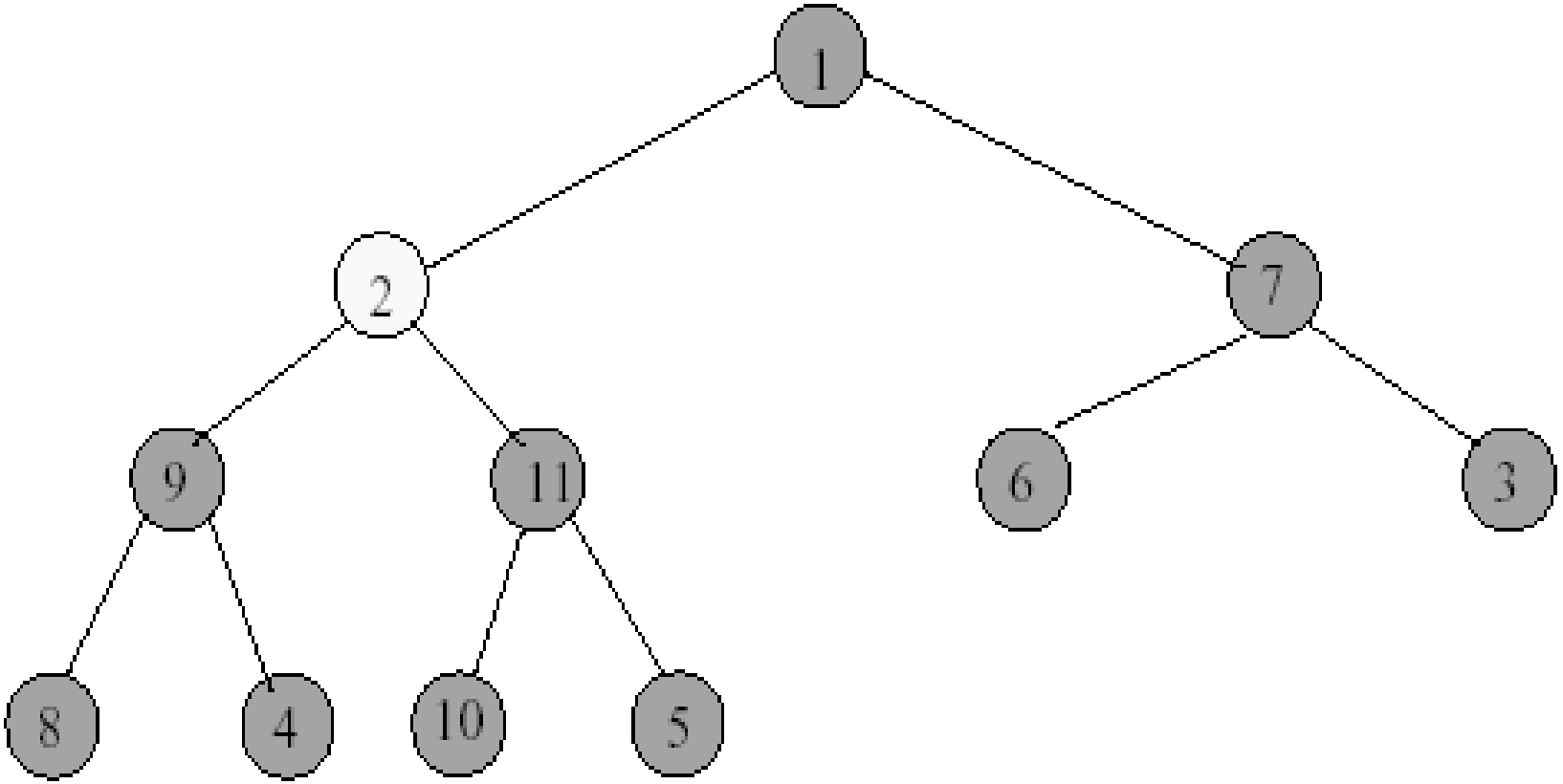
Max Heap Initialization



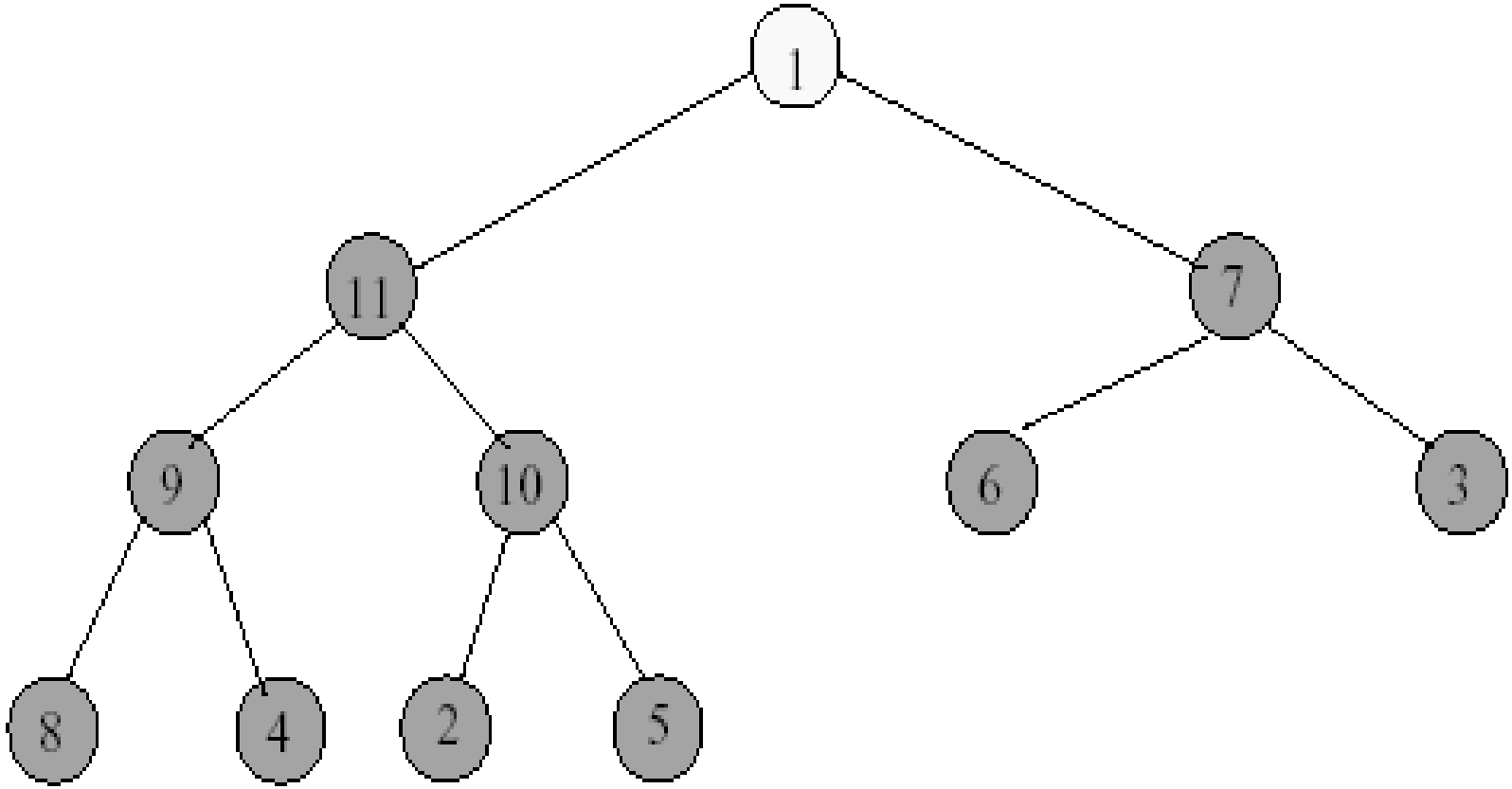
Max Heap Initialization



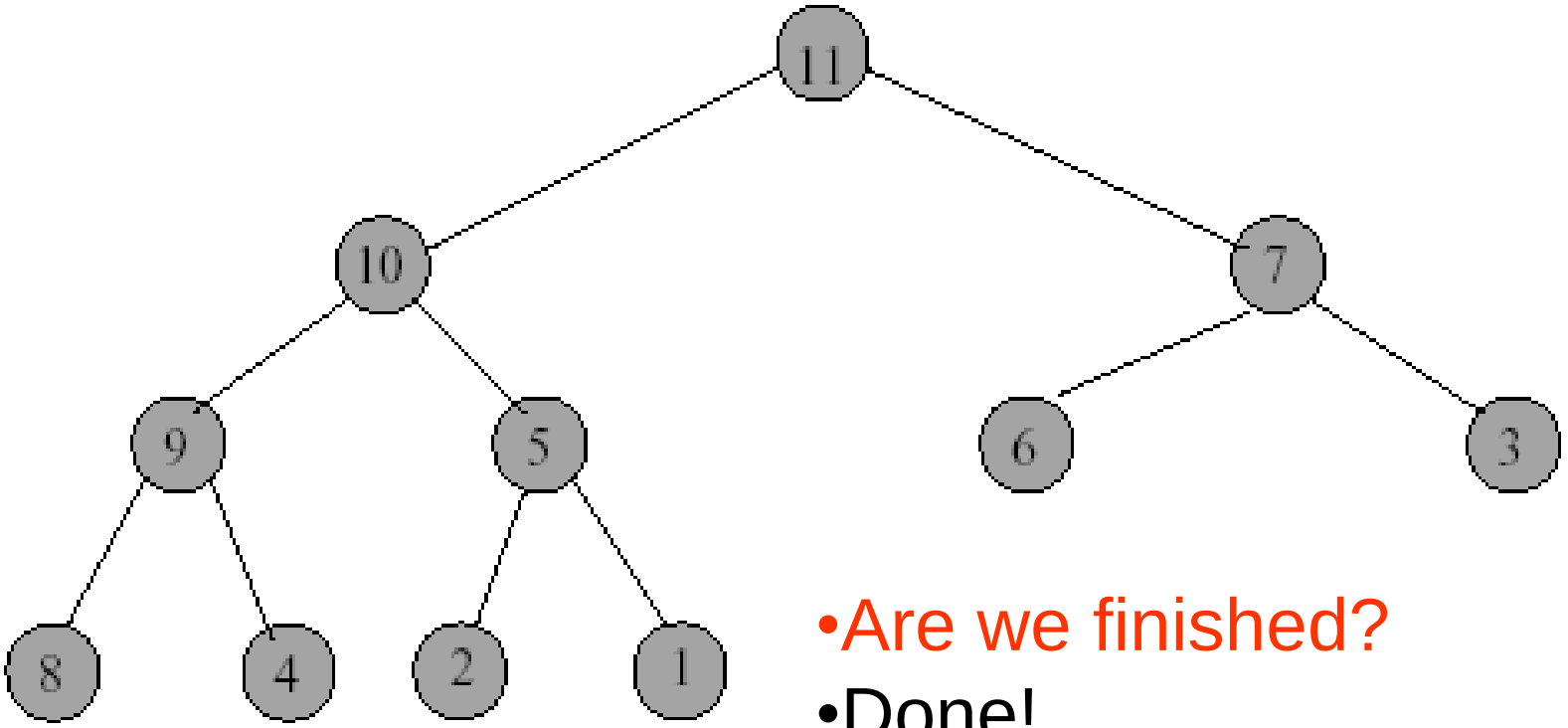
Max Heap Initialization



Max Heap Initialization



Max Heap Initialization



- Are we finished?
- Done!

Complexity of Initialization

- Bulk insert method takes O(N) time
  - Height of heap = h
  - Number of nodes at level *j* is <= 2^{j-1}
  - Time for each node at level *j* is O(h-j+1)
  - Time for all nodes at level j is <= $2^{j-1}(h-j+1) = t(j)$
  - Total time is $t(1) + t(2) + \ldots + t(h) = O(2^h) = O(n)$

Tasks

- Self study
  - **Read**: Heap and Priority Queue (online sources)
- Finish Assignment 5 by Mon Nov 14 *(100 points)*
- Make progress on CodeChef (100 points)

Questions?