NPTEL MOOC,JAN-FEB 2015
Week 5, Module 4

DESIGN AND ANALYSIS OF ALGORITHMS

Heaps

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Priority queue

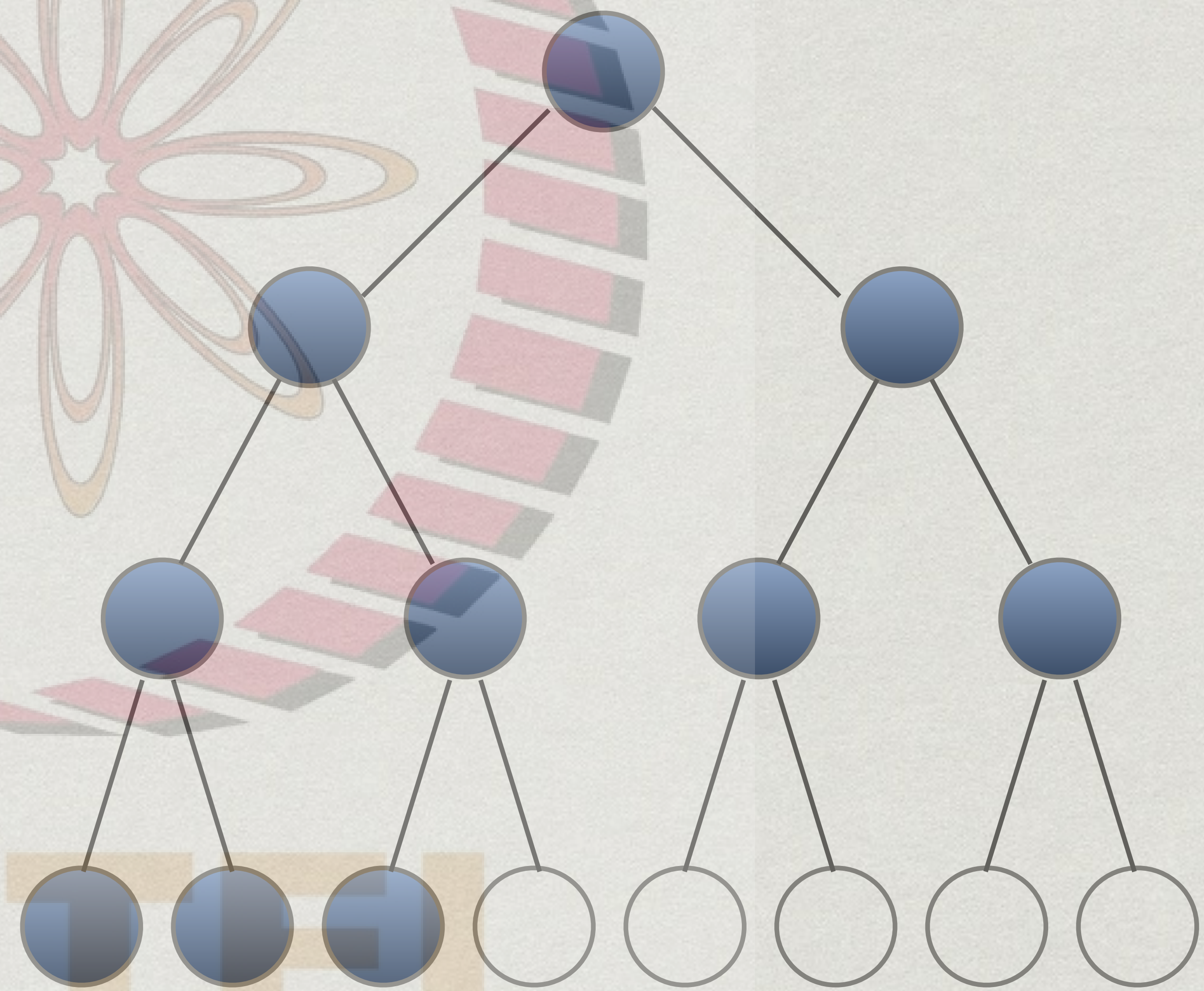
- Need to maintain a list of jobs with priorities to optimise the following operations
  - delete_max( )
    - Identify and remove job with highest priority
    - Need not be unique
  - insert( )
    - Add a new job to the list

Trees

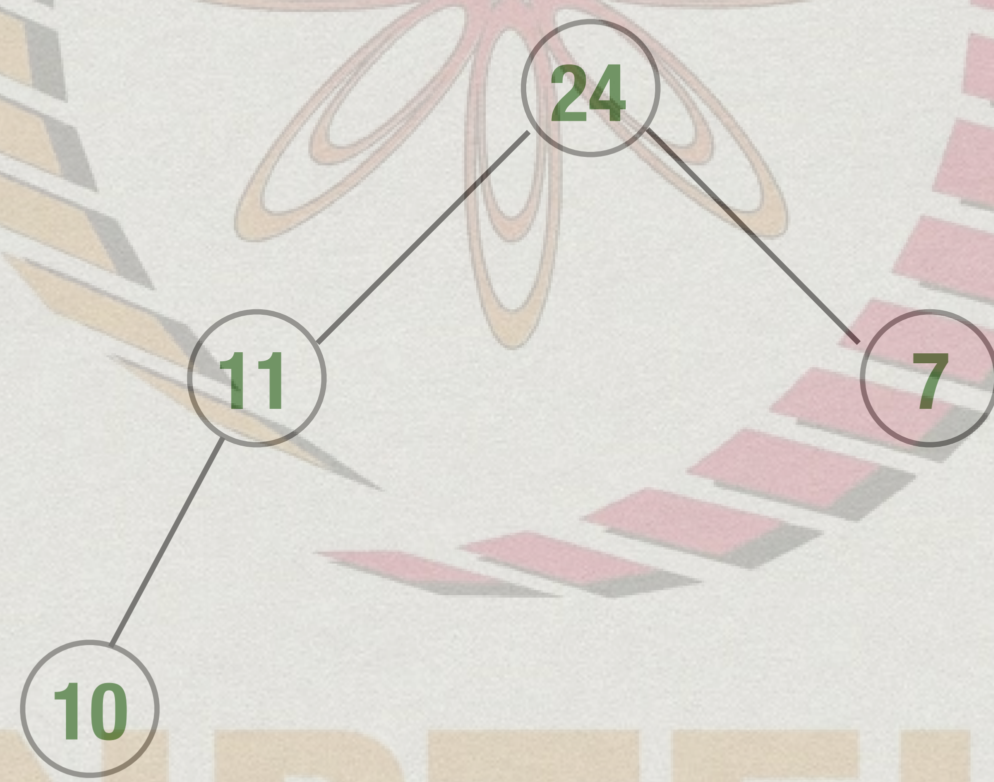
- Maintain a special kind of binary tree called a heap
  - Balanced: N node tree has height log N
- Both insert( ) and delete_max( ) take O(log N)
  - Processing N jobs takes time O(N log N)
- Truly flexible, need not fix upper bound for N in advance

Heaps

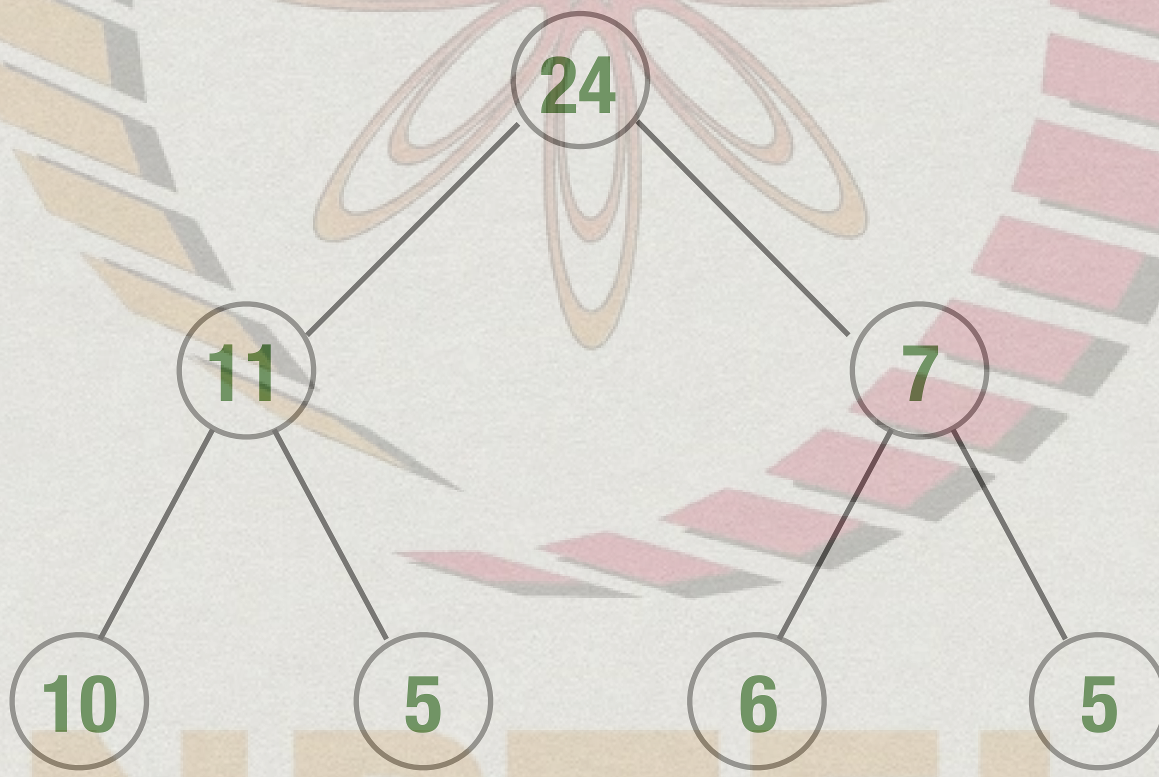
- Binary tree filled level by level, left to right
- At each node, value stored is bigger than both children
  - (Max) Heap PropertyBinary tree filled level by level, left to right



Examples

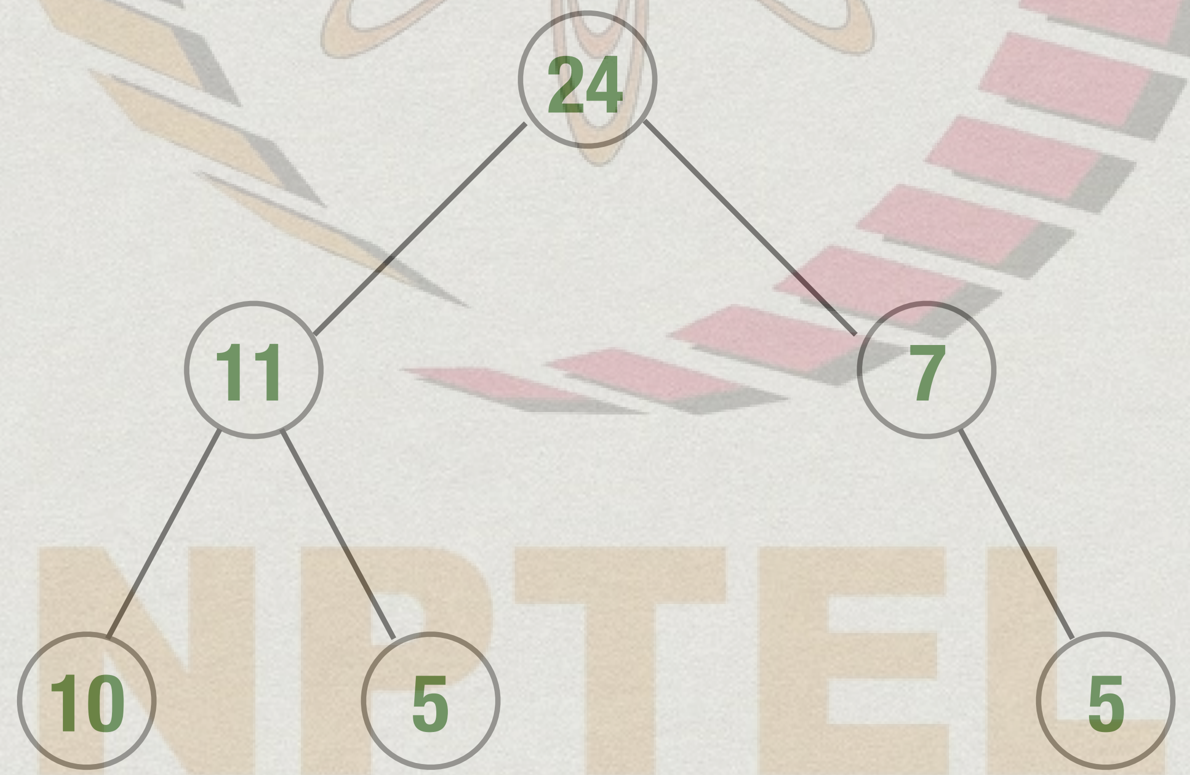


Examples



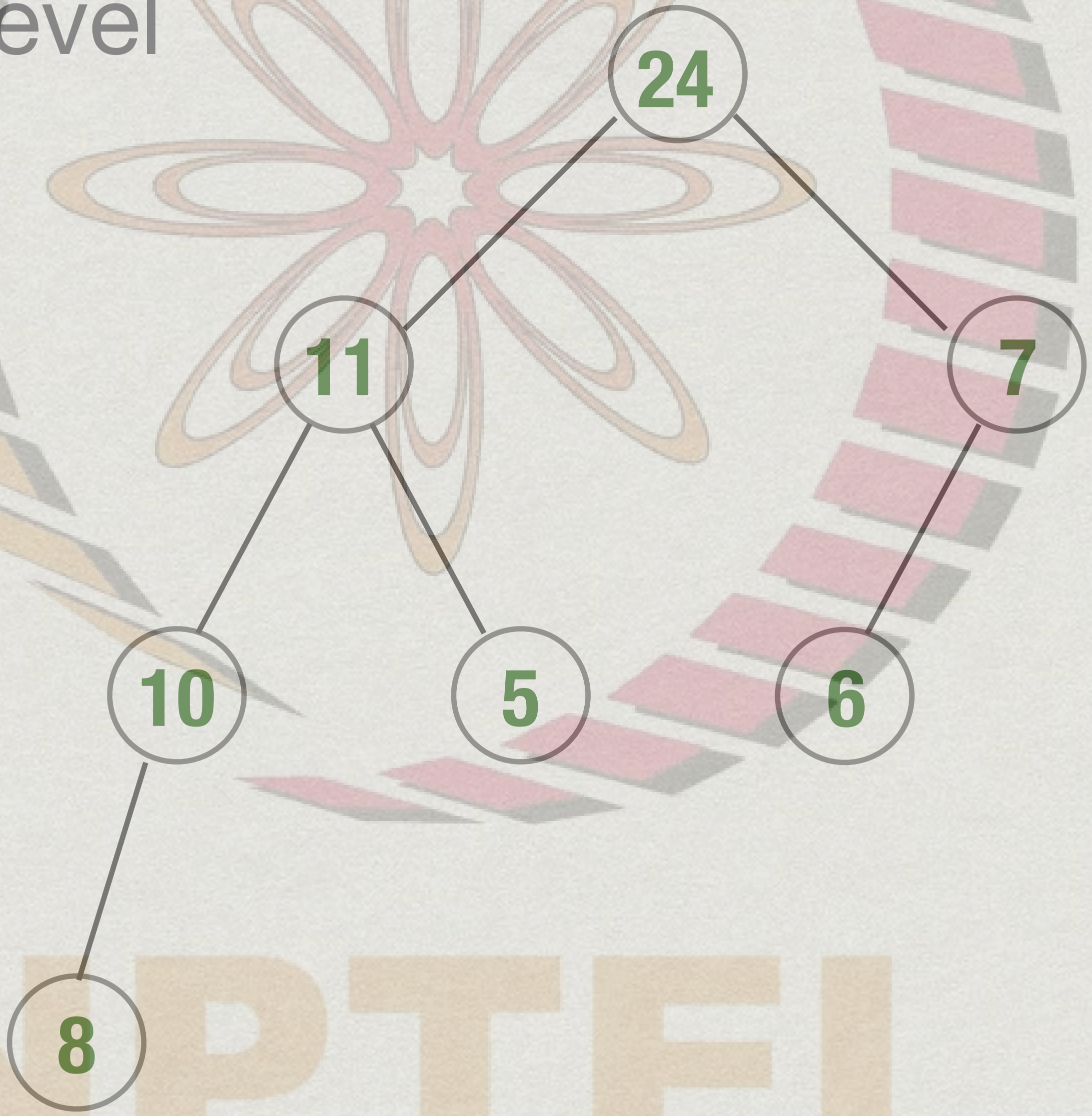
Non-examples

* No “holes” allowed



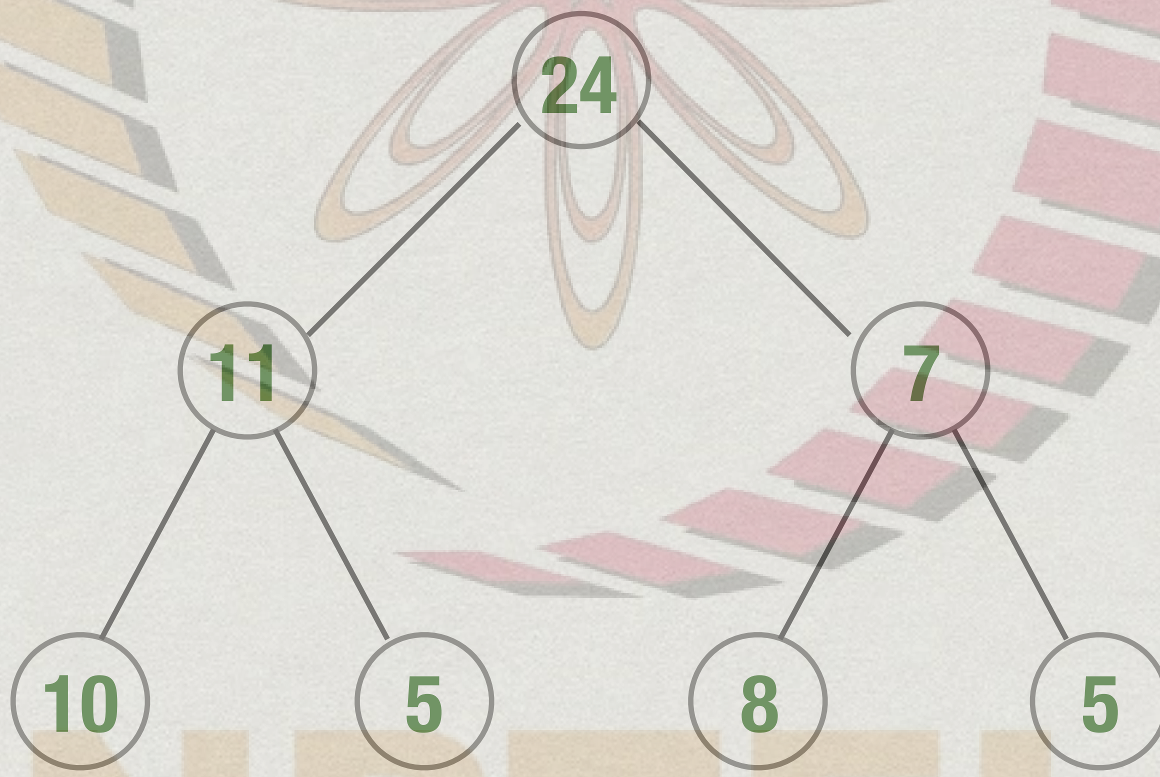
Non-examples

* Can't leave a level incomplete



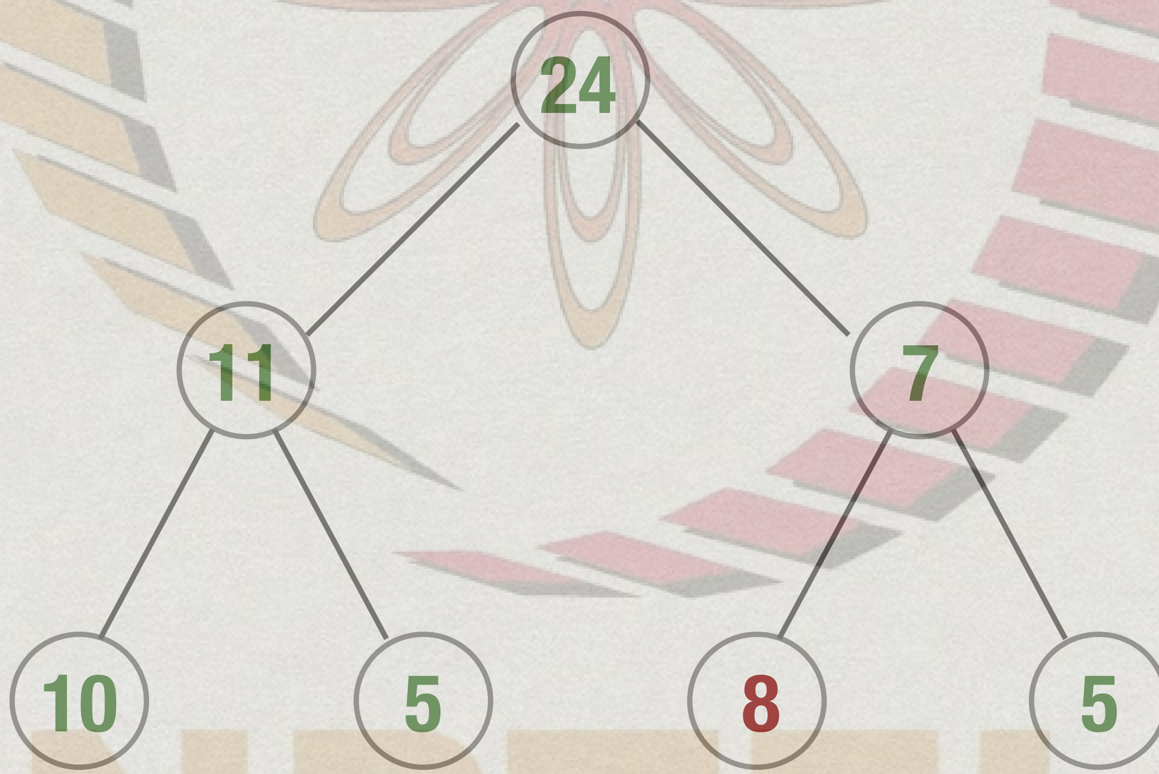
Non-examples

- Violates heap property



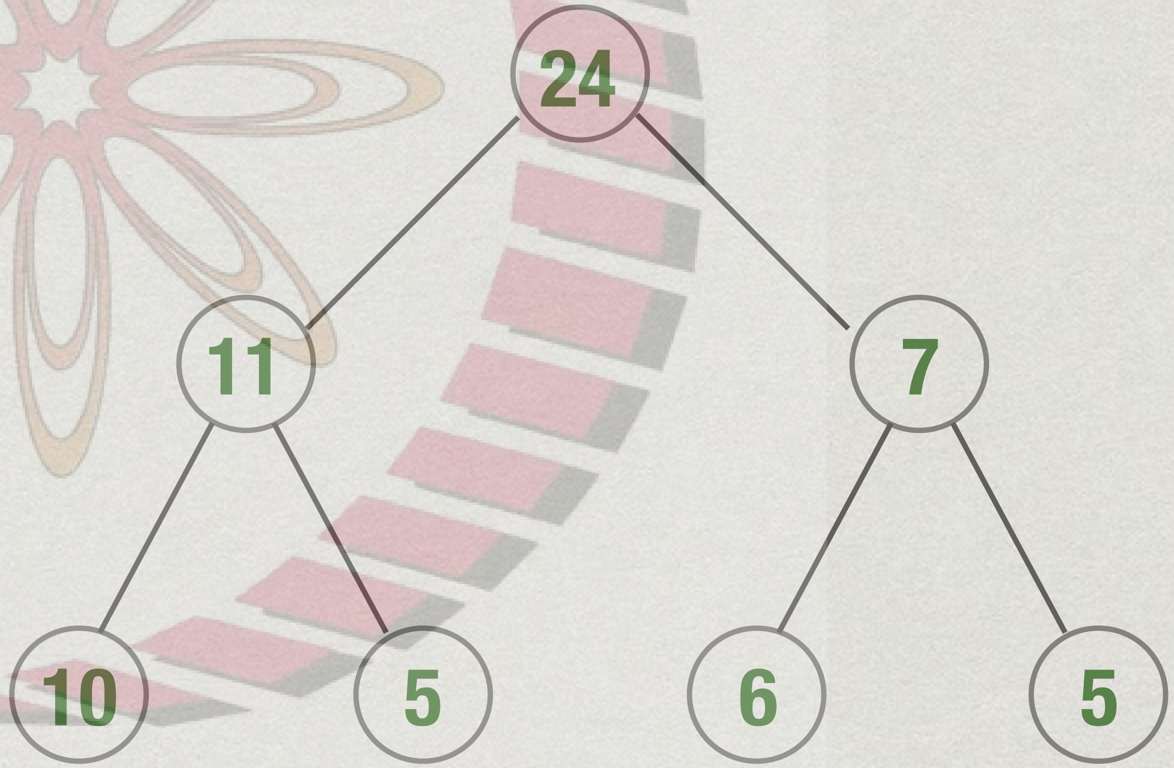
Non-examples

- Violates heap property



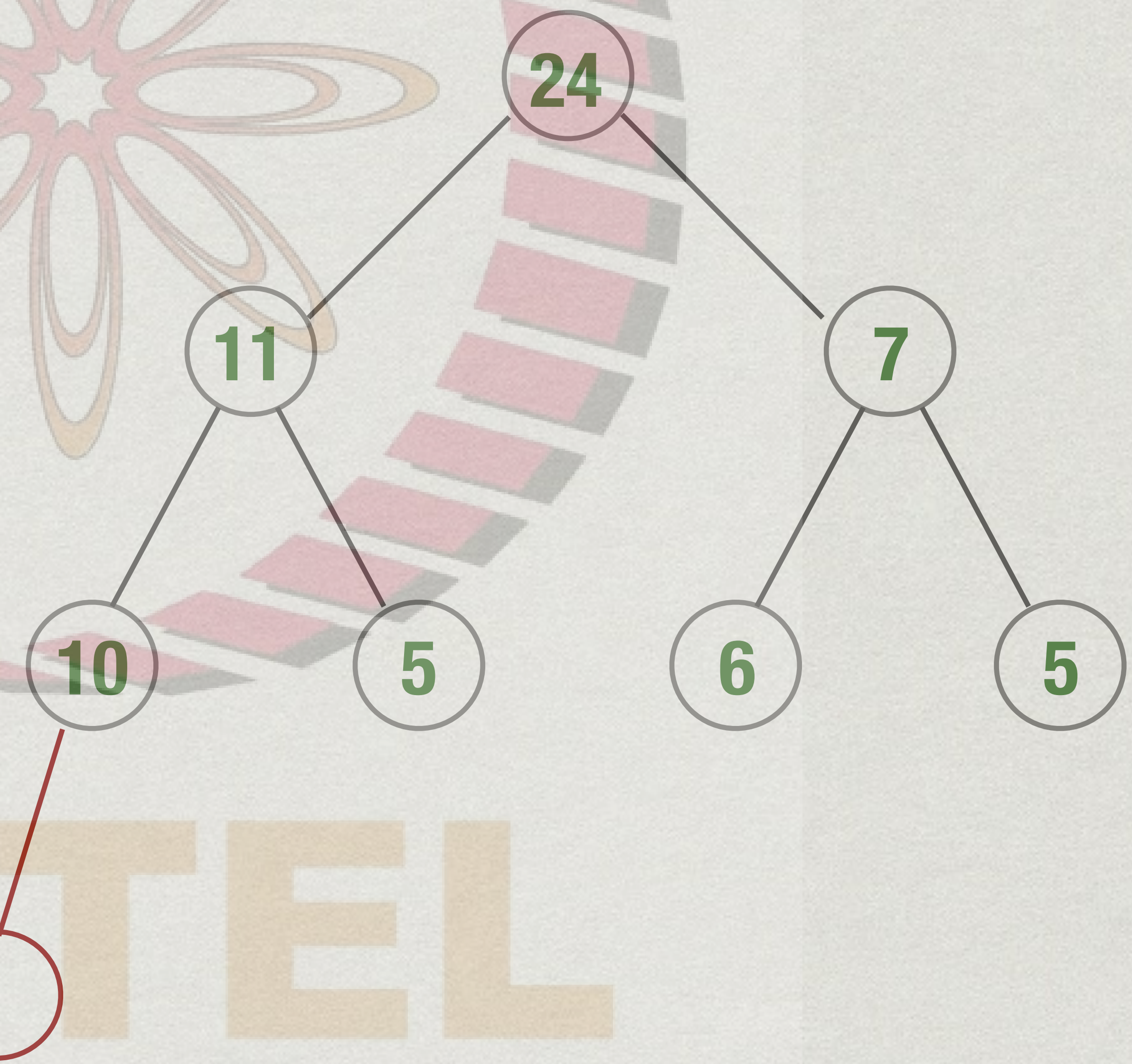
insert( )

- insert 12
- Position of new node is fixed by structure
- Restore heap property along the path to the root



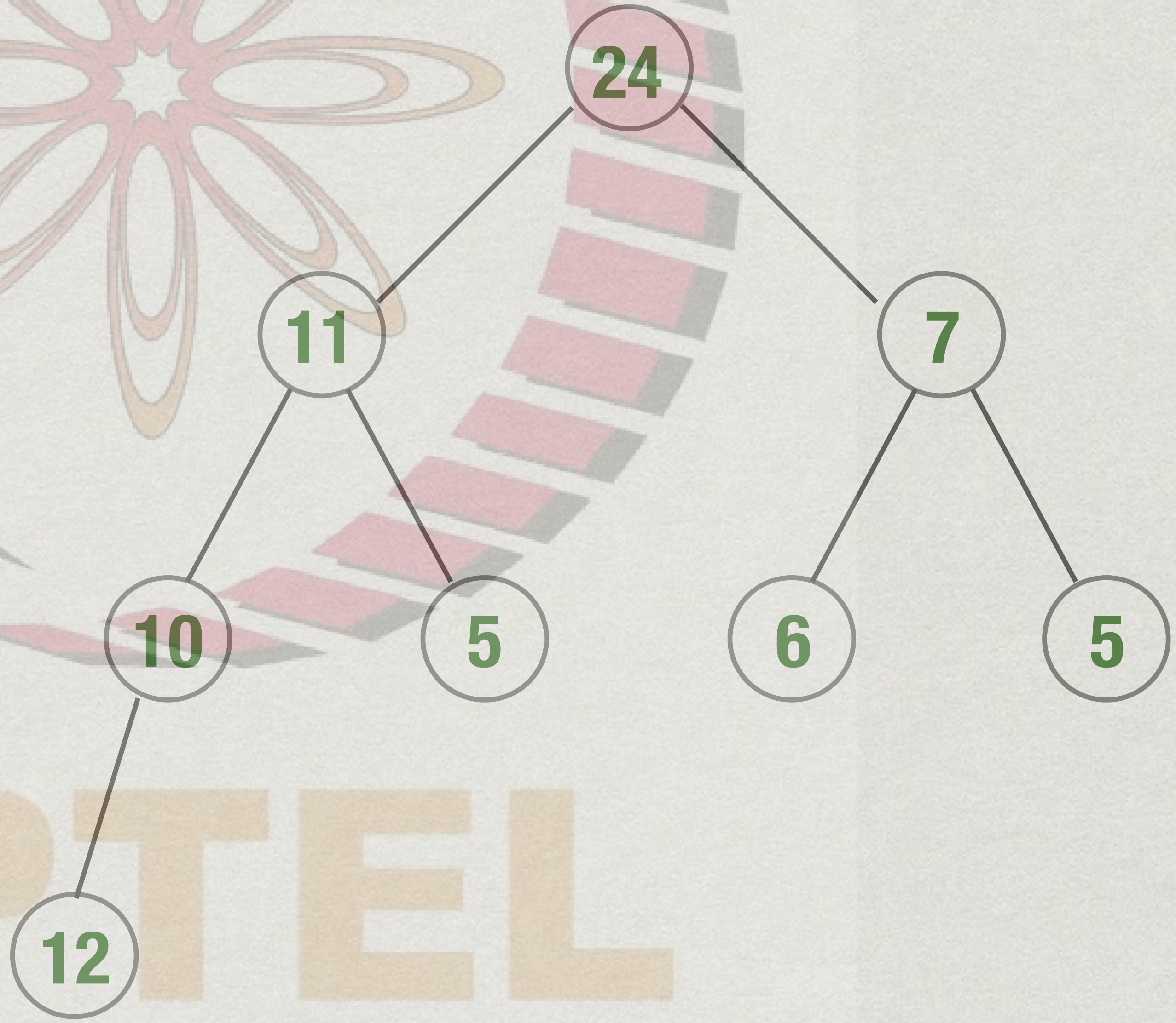
insert( )

- insert 12
- Position of new node is fixed by structure
- Restore heap property along the path to the root



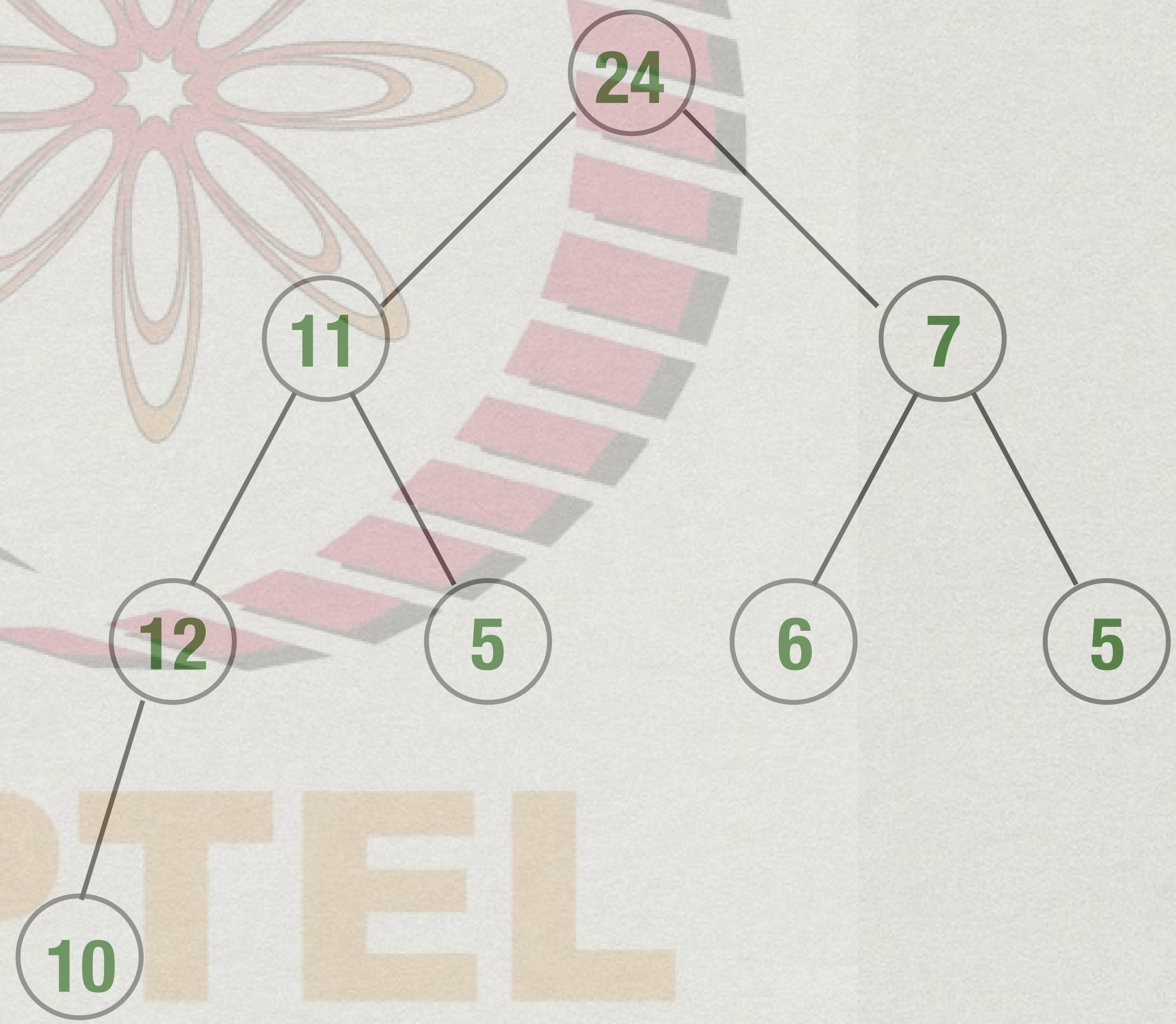
insert( )

- insert 12
- Position of new node is fixed by structure
- Restore heap property along the path to the root



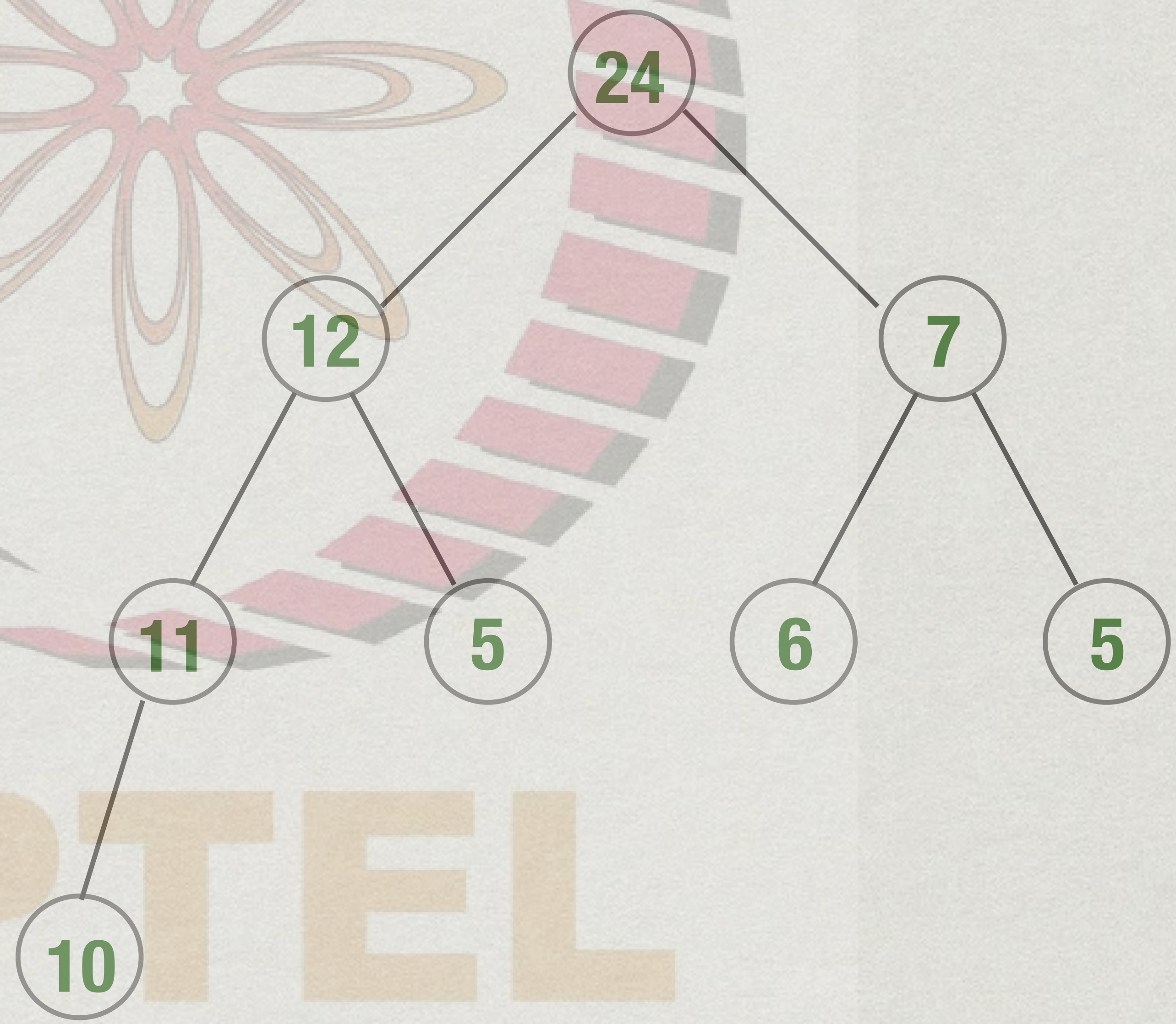
insert( )

- insert 12
- Position of new node is fixed by structure
- Restore heap property along the path to the root



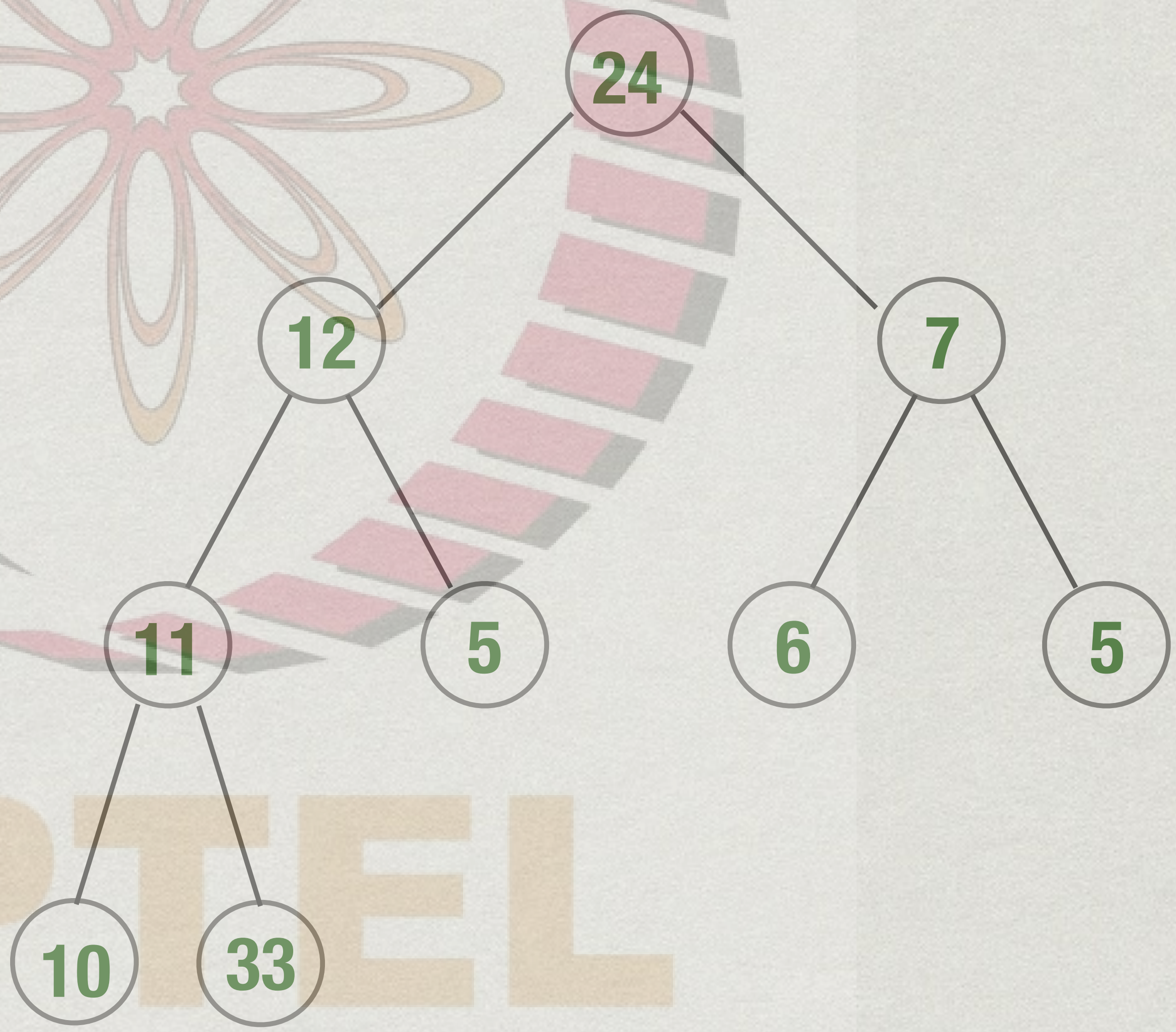
insert( )

- insert 12
- Position of new node is fixed by structure
- Restore heap property along the path to the root



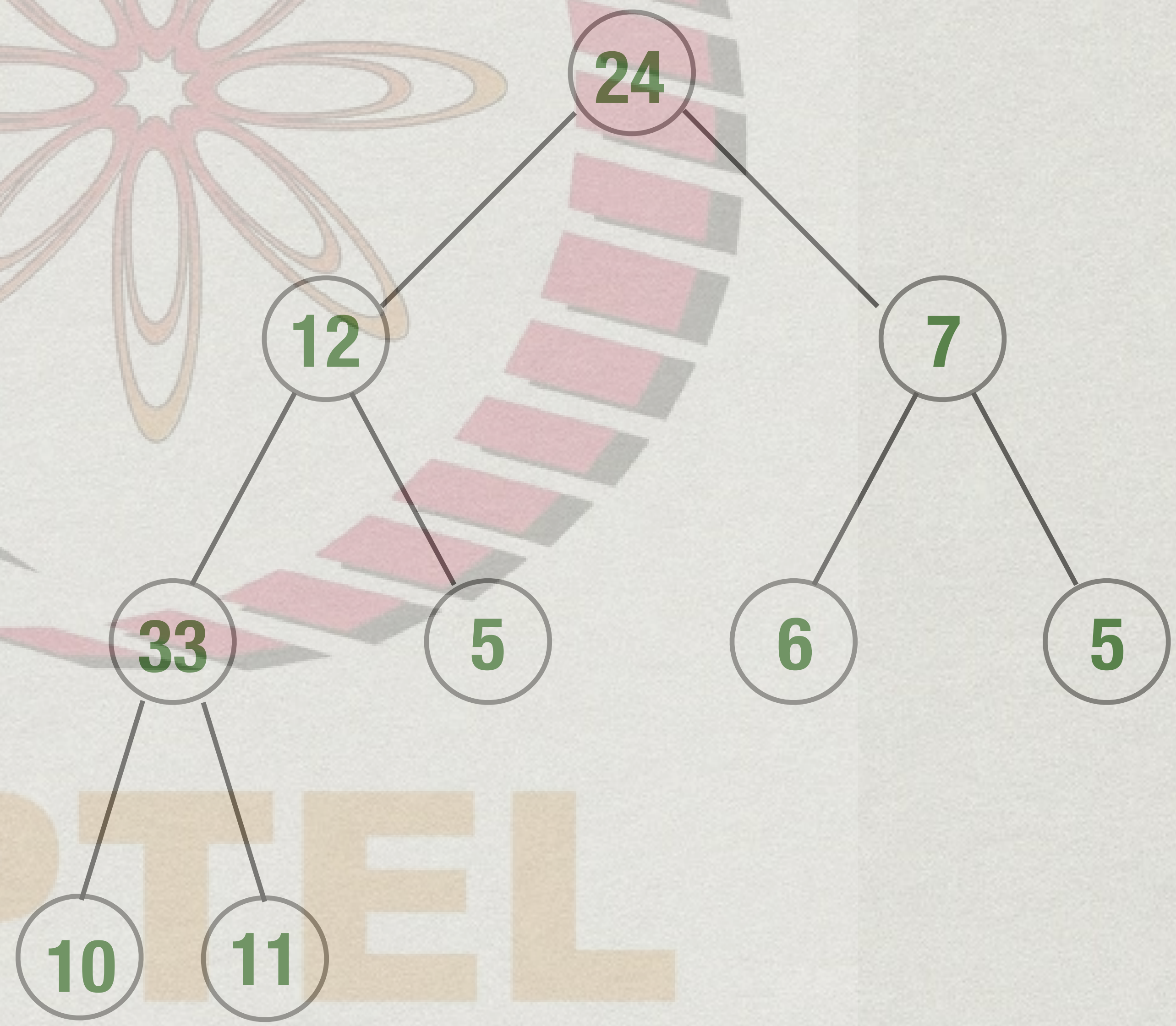
insert( )

- insert 33



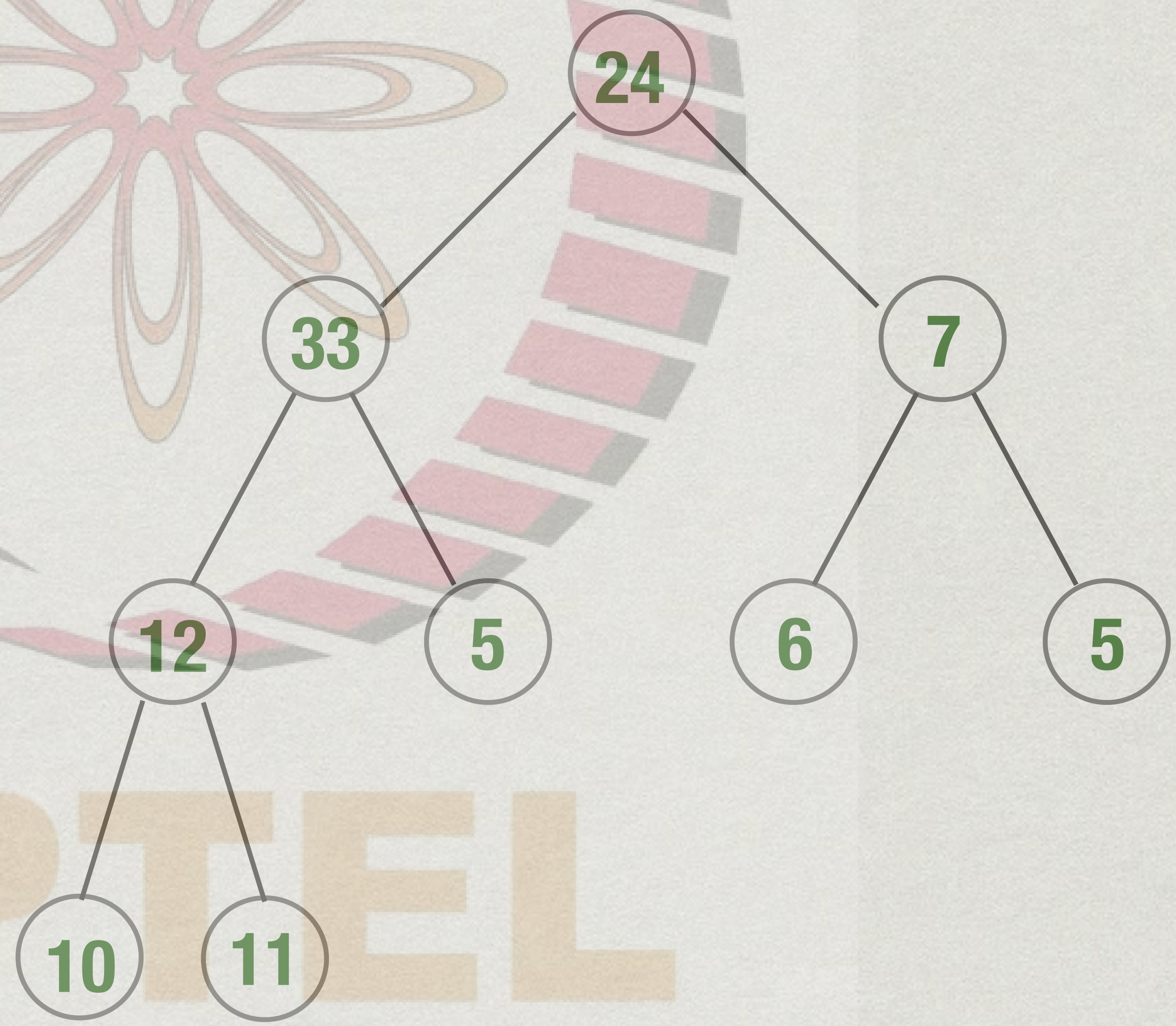
insert( )

* insert 33



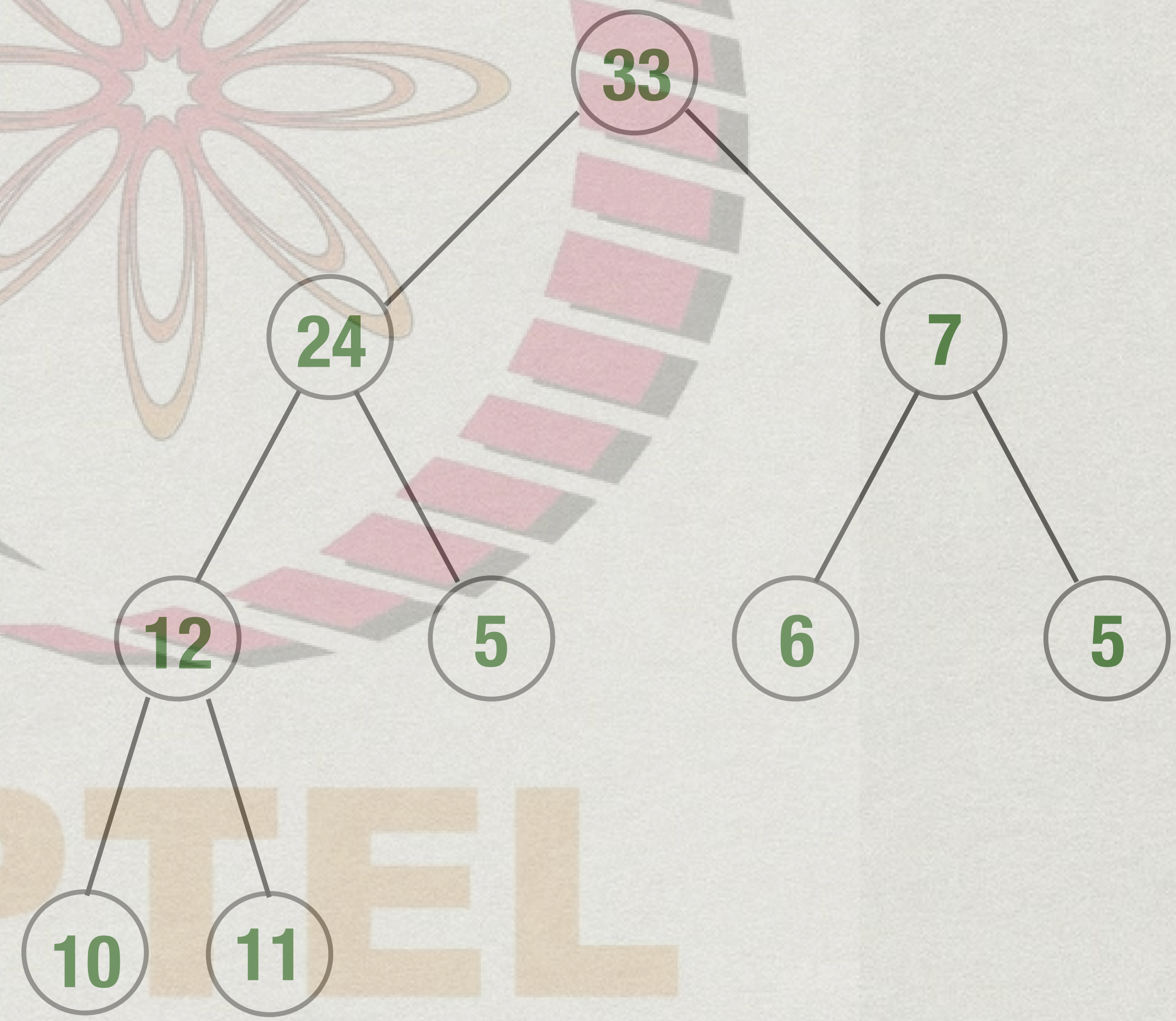
insert( )

- insert 33



insert( )

- insert 33

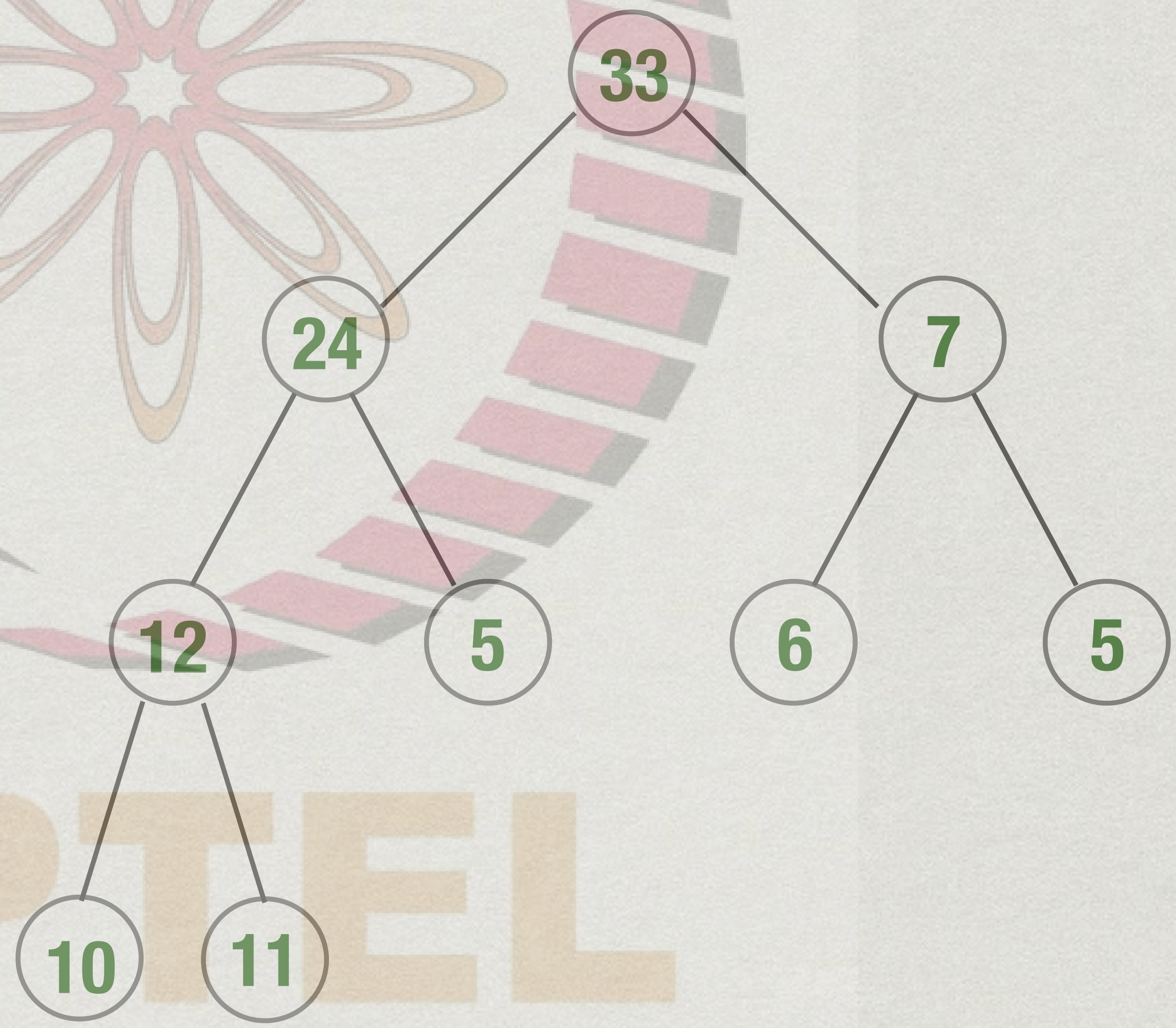


Complexity of insert( )

- Need to walk up from the leaf to the root
  - Height of the tree
- Number of nodes at level 0,1,…,i is $2^0, 2^1, \ldots, 2^i$
- K levels filled : $2^0+2^1+ \ldots +2^{k-1} = 2^k - 1$ nodes
- N nodes : number of levels at most log N + 1
- insert( ) takes time O(log N)

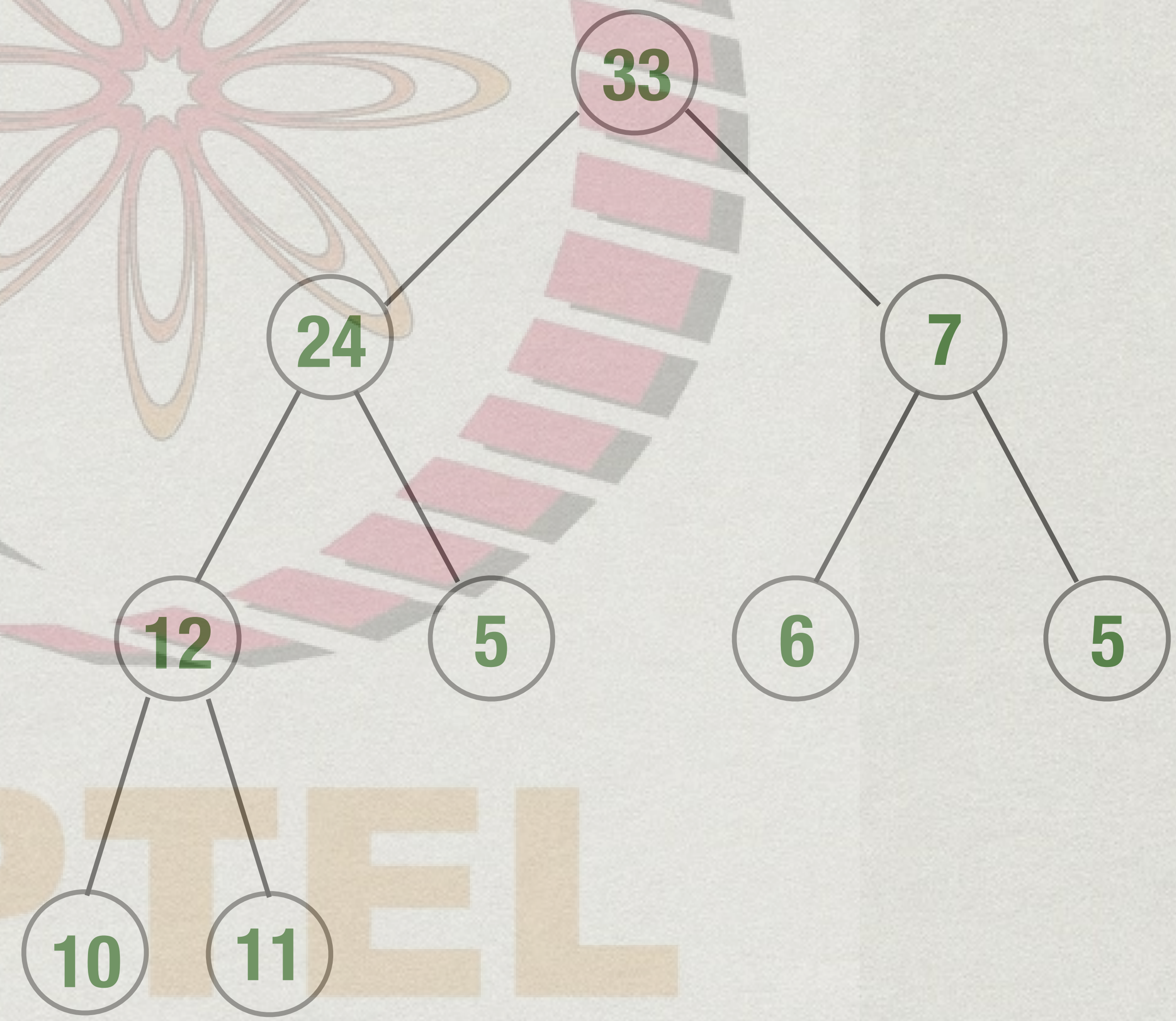
delete_max( )

- Maximum value is always at the root
    - From heap property, by induction
- How do we remove this value efficiently?



delete_max( )

- Removing maximum value creates a "hole" at the root
- Reducing one value requires deleting last node
- Move "homeless" value to root

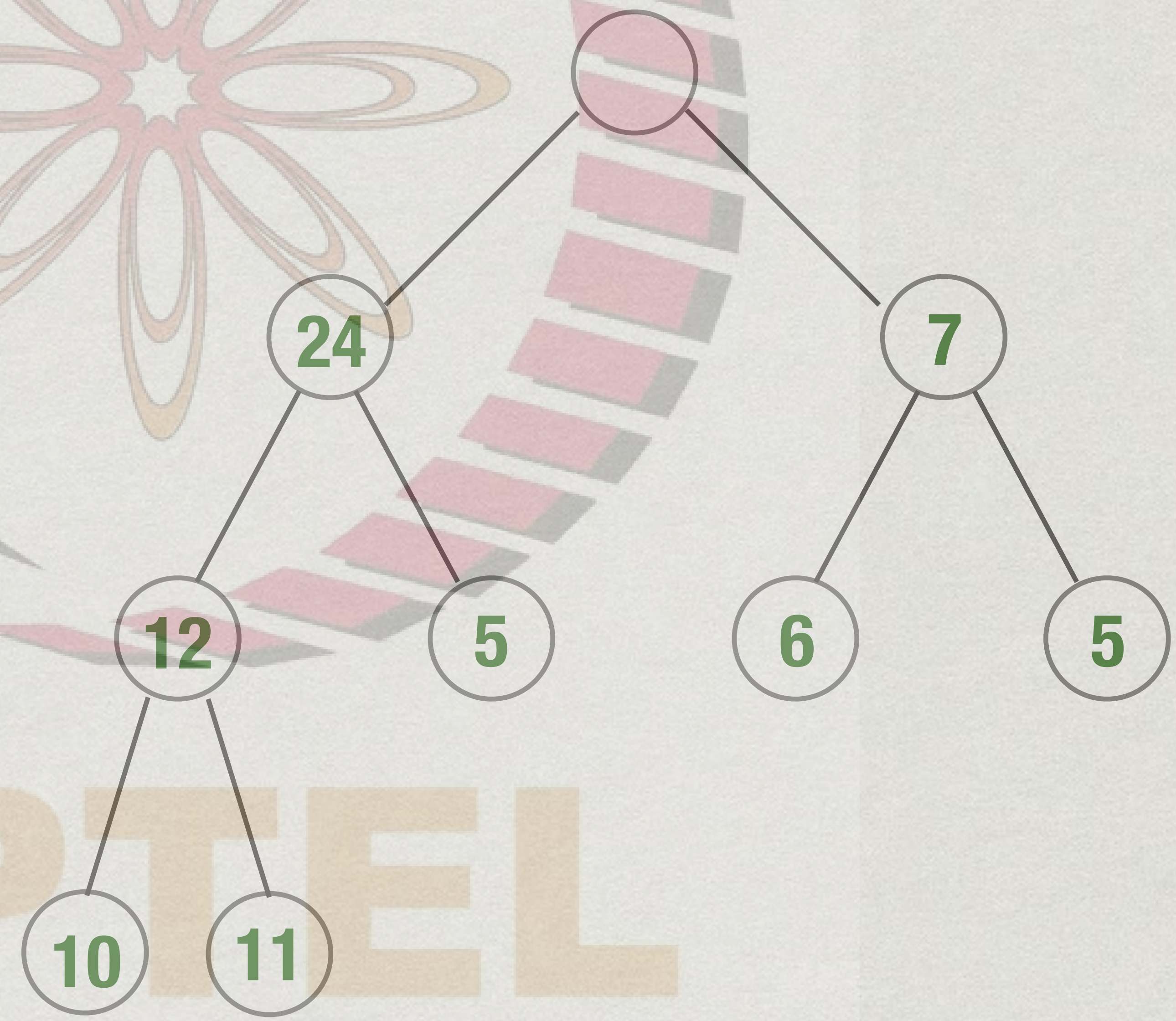


delete_max( )

- Removing maximum value creates a “hole” at the root

- Reducing one value requires deleting last node

- Move “homeless” value to root

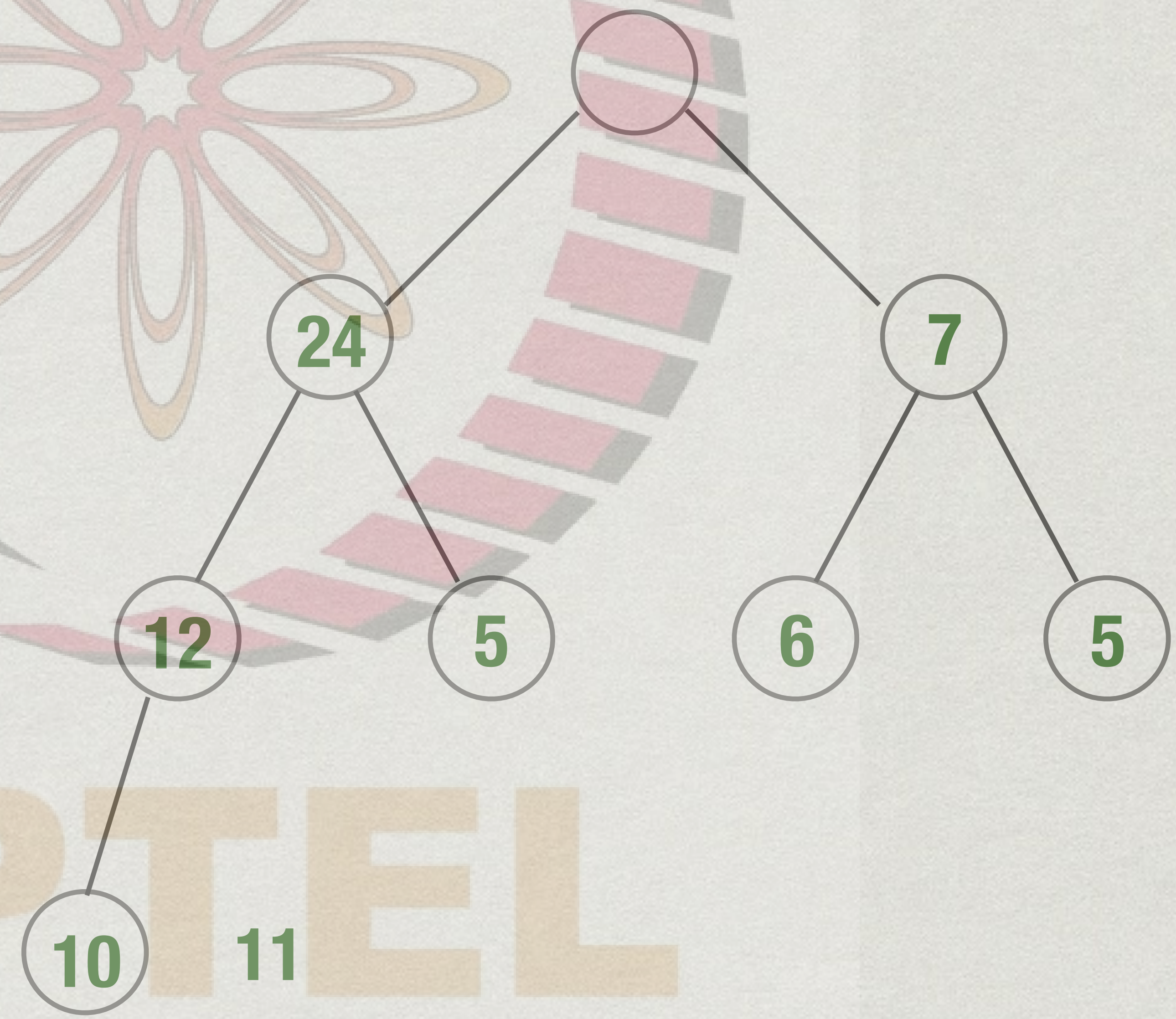


delete_max( )

- Removing maximum value creates a "hole" at the root

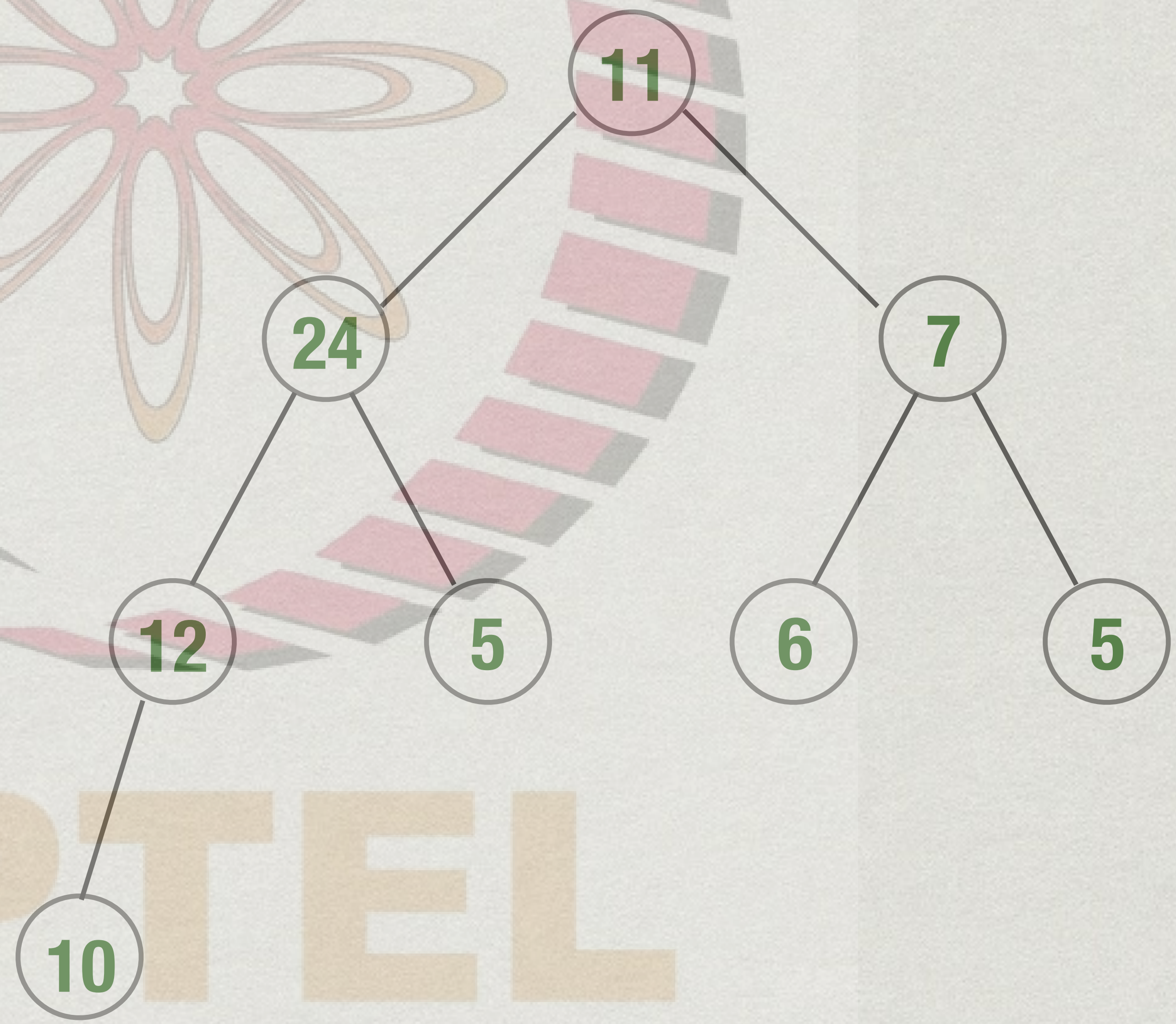
- Reducing one value requires deleting last node

- Move "homeless" value to root



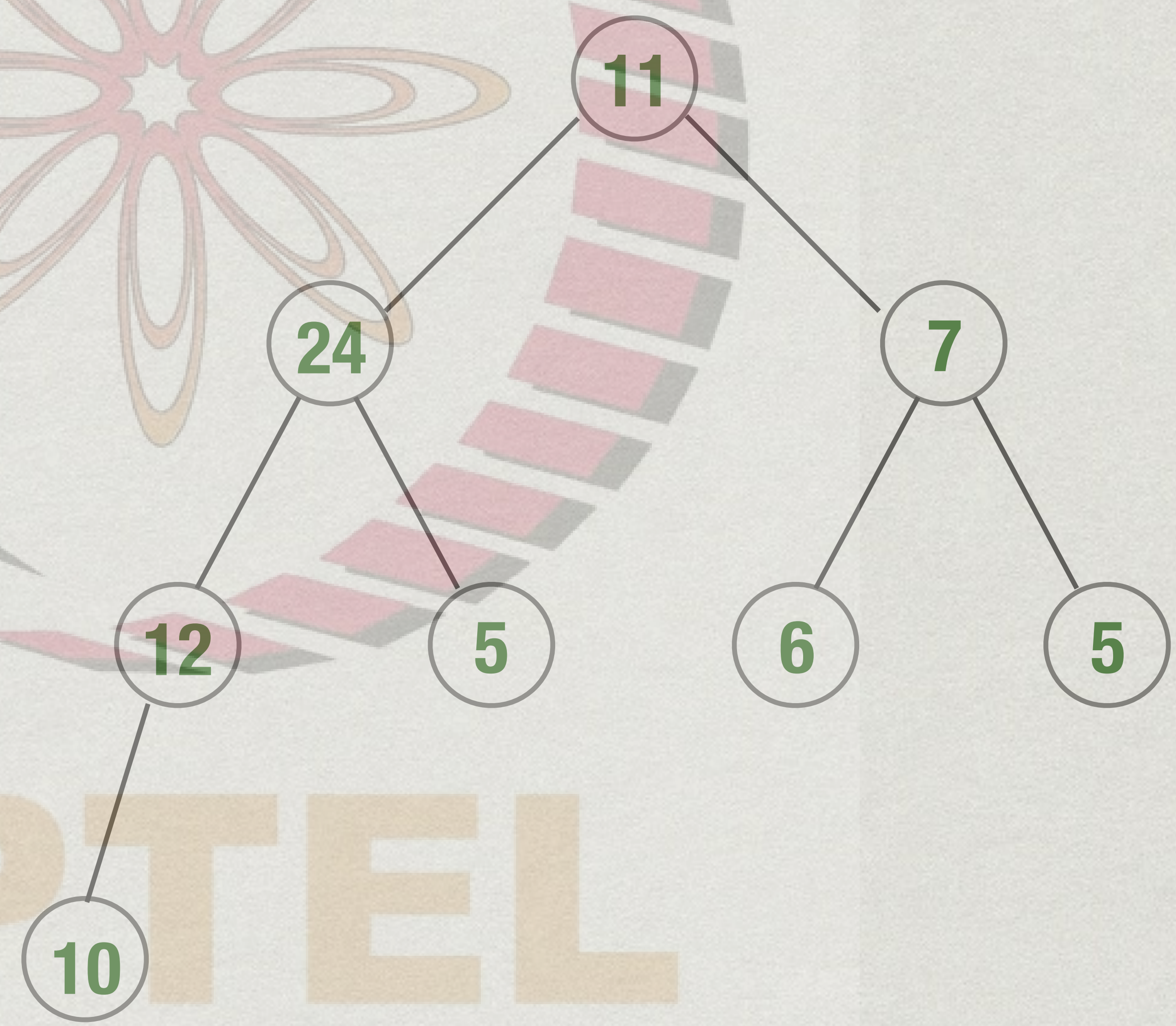
delete_max( )

- Removing maximum value creates a “hole” at the root
- Reducing one value requires deleting last node
- Move “homeless” value to root



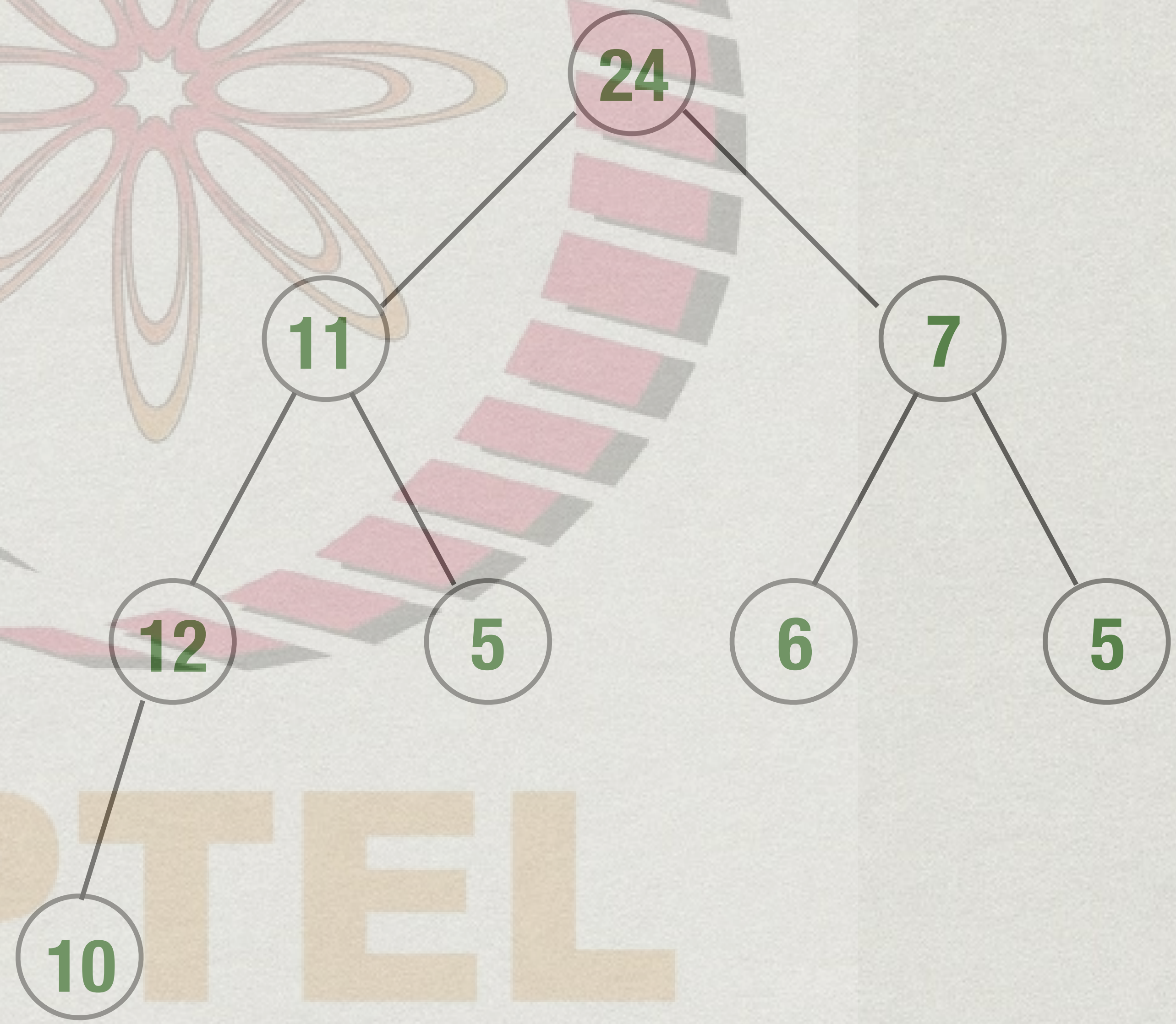
delete_max( )

- Now restore the heap property from root downwards
  - Swap with largest child
- Will follow a single path from root to leaf



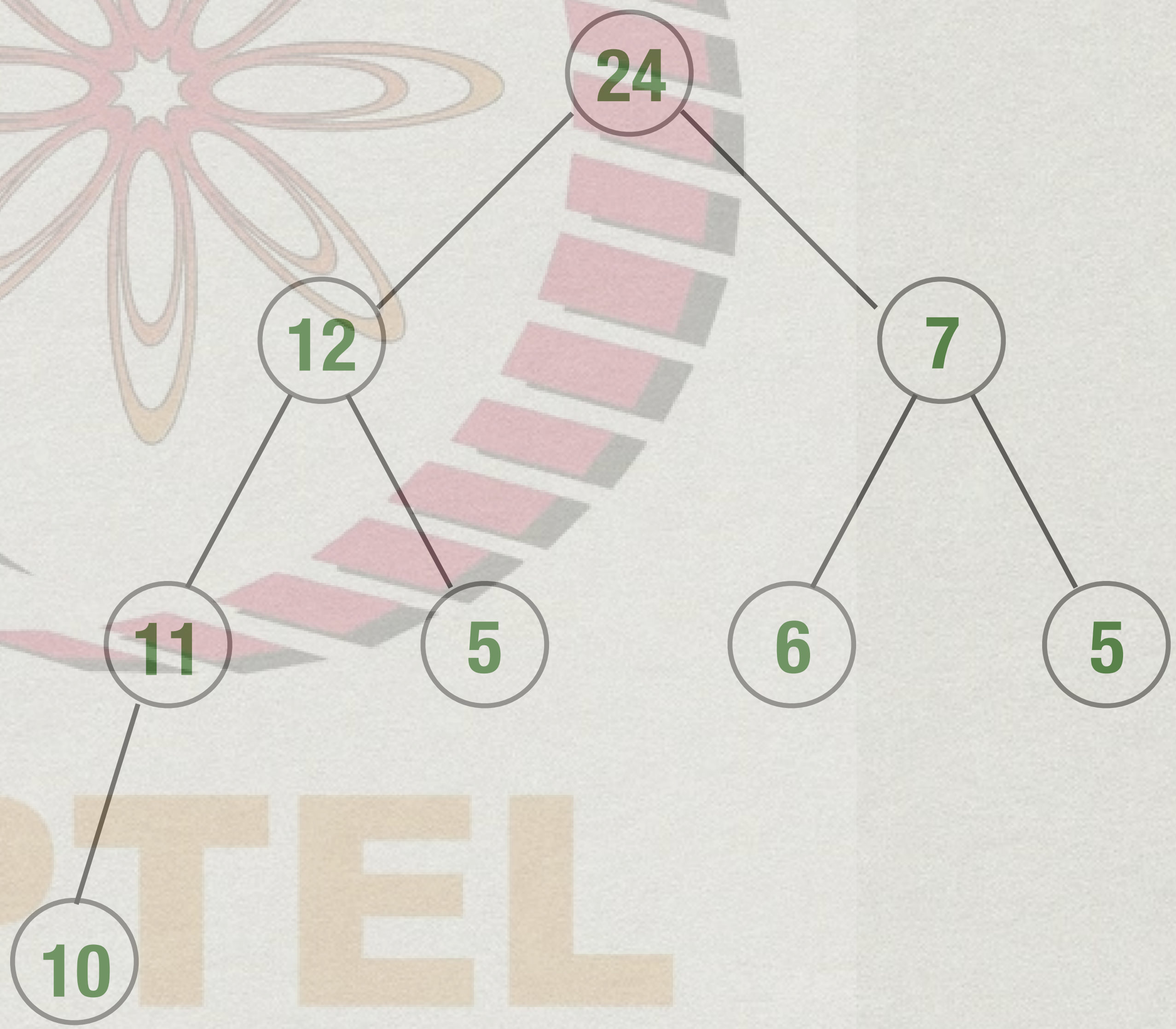
delete_max( )

* Now restore the heap property from root downwards
    * Swap with largest child
* Will follow a single path from root to leaf



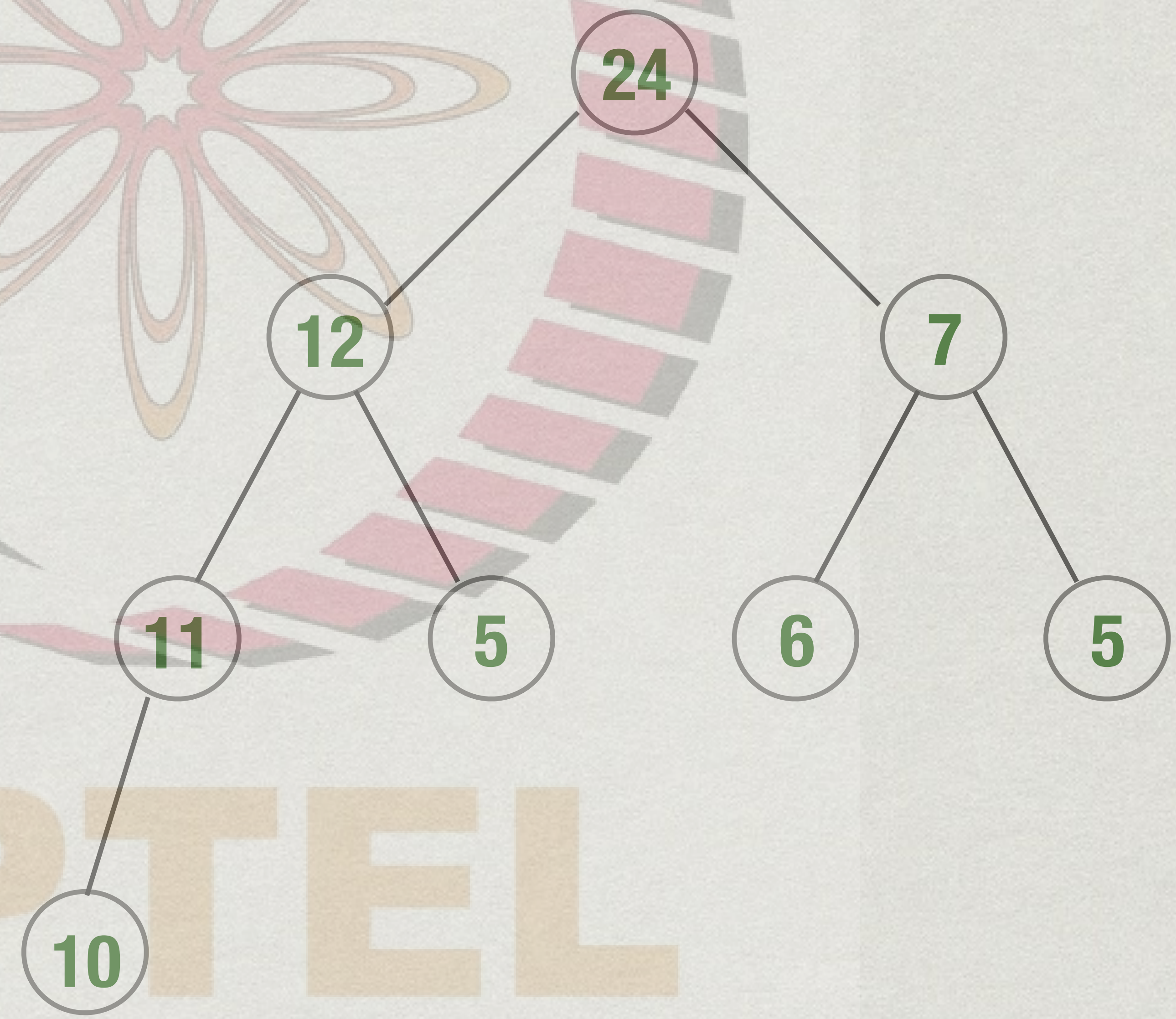
delete_max( )

- Now restore the heap property from root downwards
  - Swap with largest child
- Will follow a single path from root to leaf



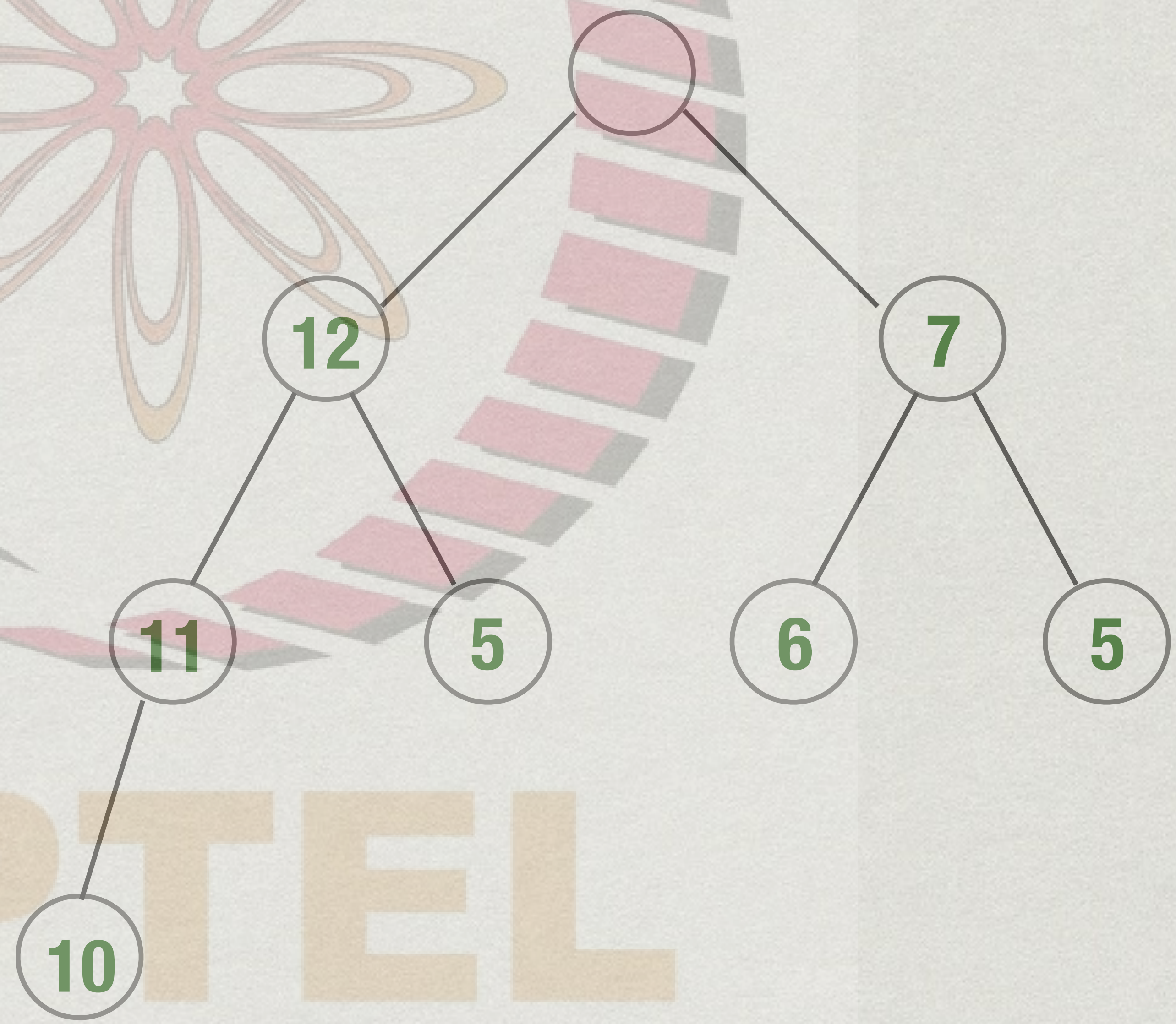
delete_max( )

- Will follow a single path from root to leaf
- Cost proportional to height of tree
- O(log N)



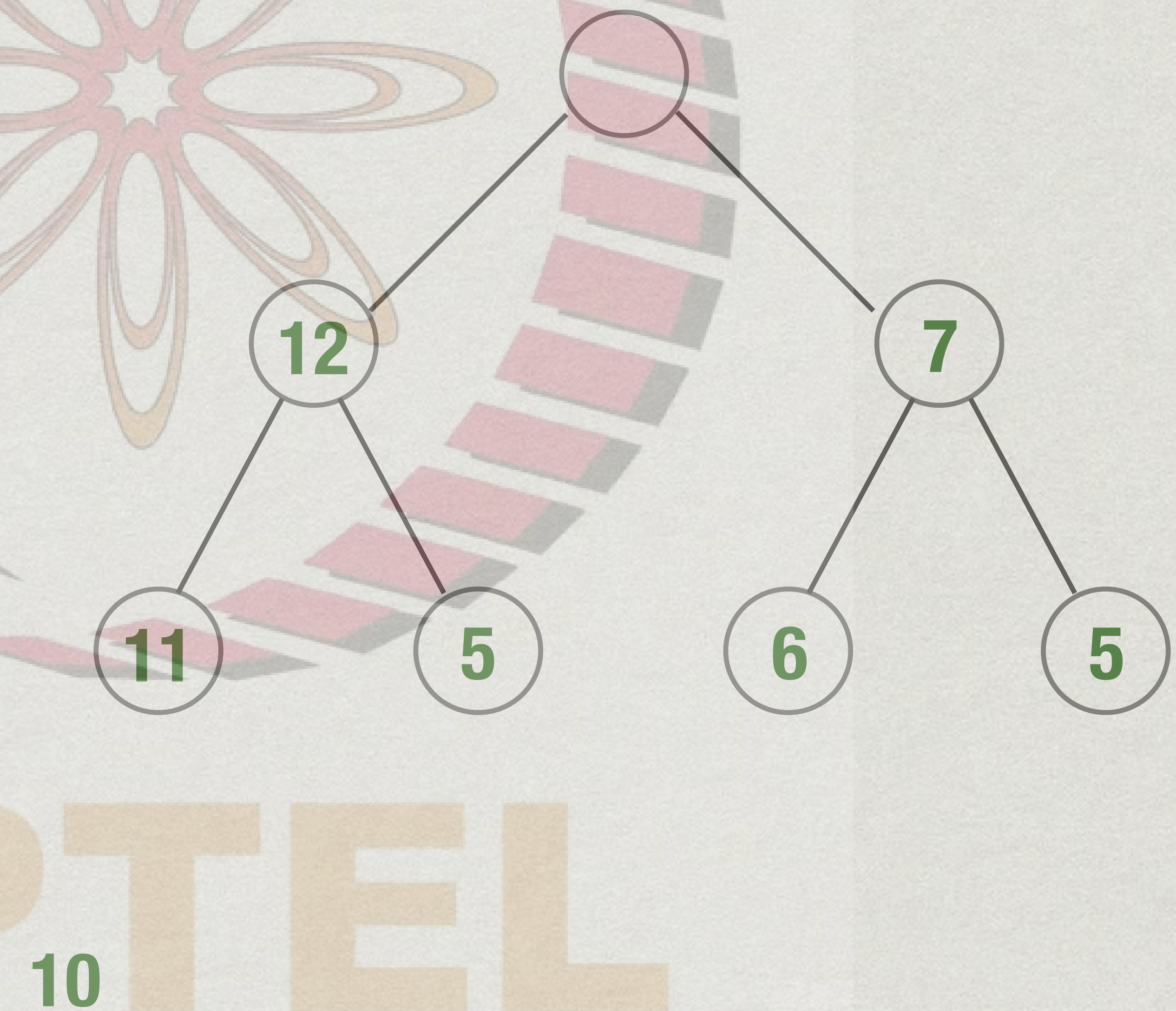
delete_max( )

- Will follow a single path from root to leaf
- Cost proportional to height of tree
- O(log N)



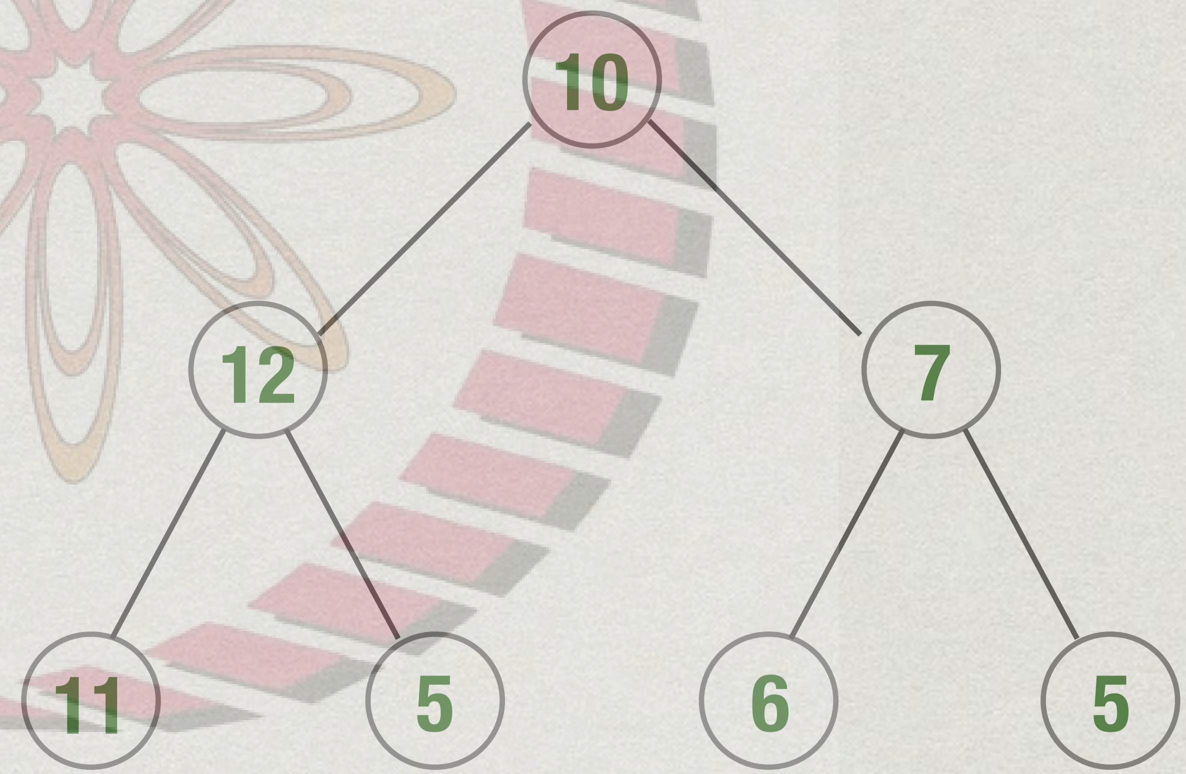
delete_max( )

- Will follow a single path from root to leaf
- Cost proportional to height of tree
- O(log N)



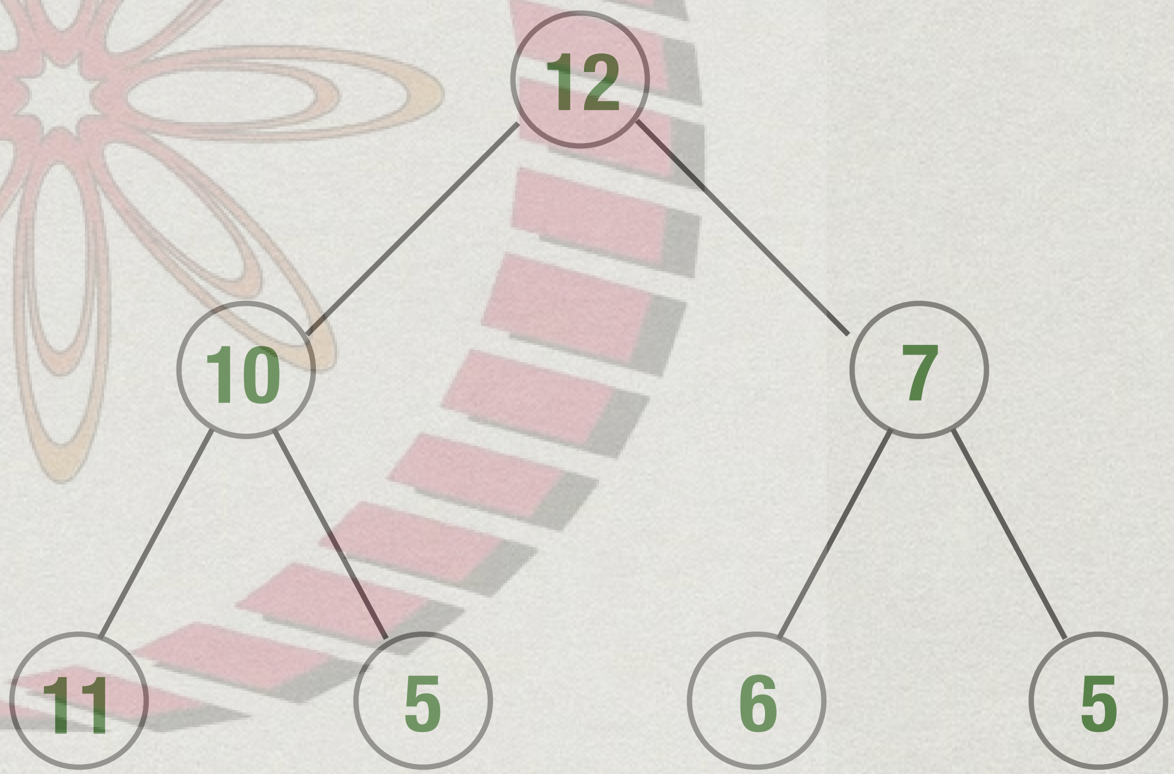
delete_max( )

- Will follow a single path from root to leaf
- Cost proportional to height of tree
- O(log N)



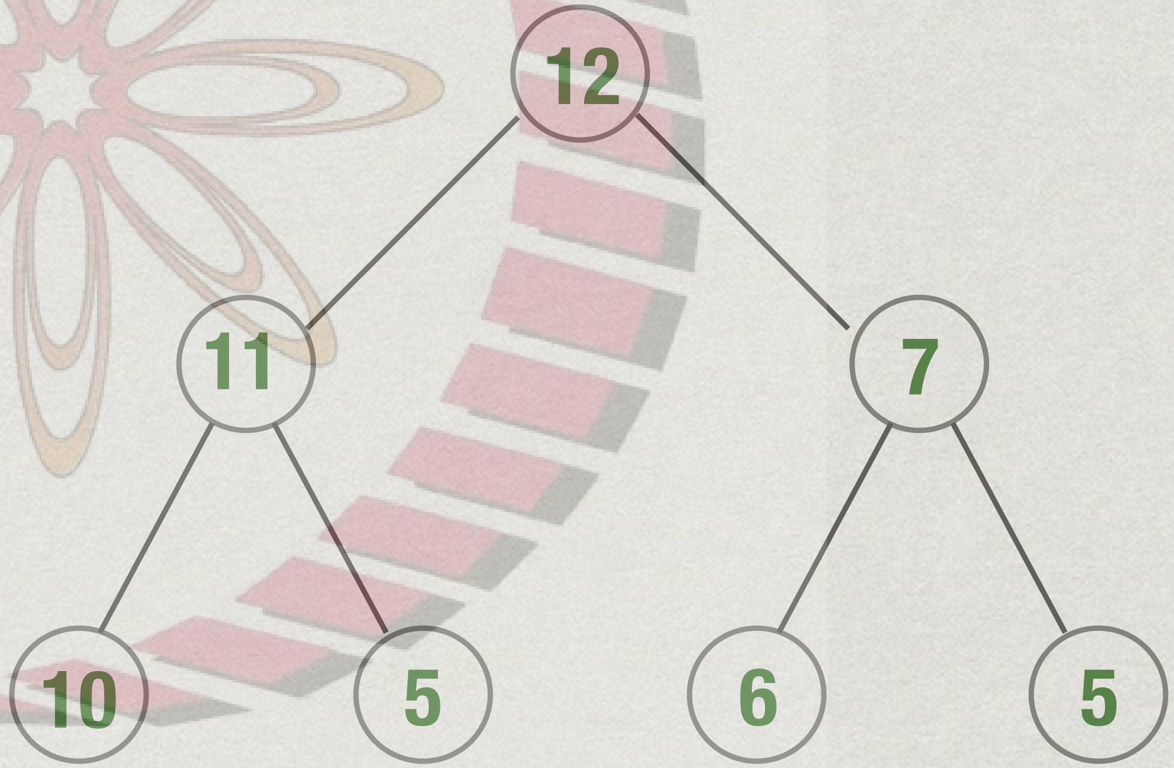
delete_max( )

- Will follow a single path from root to leaf
- Cost proportional to height of tree
- O(log N)



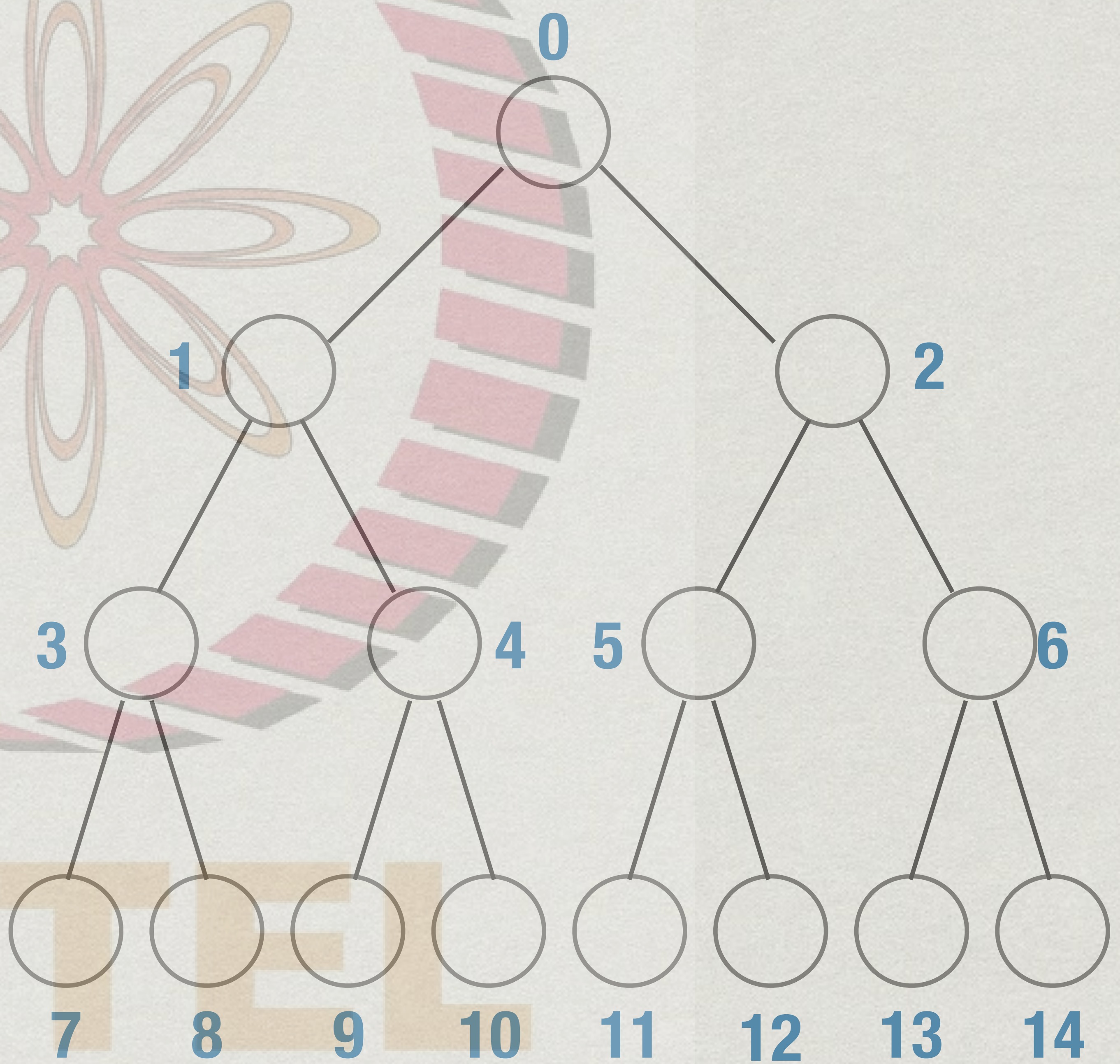
delete_max( )

- Will follow a single path from root to leaf
- Cost proportional to height of tree
- O(log N)



Impementing using arrays

- Number the nodes left to right, level by level
- Represent as an array H[0..N-1]
- Children of H[i] are at H[2i+1], H[2i+2]
- Parent of H[j] is at H[floor((j-1)/2)] for j > 0



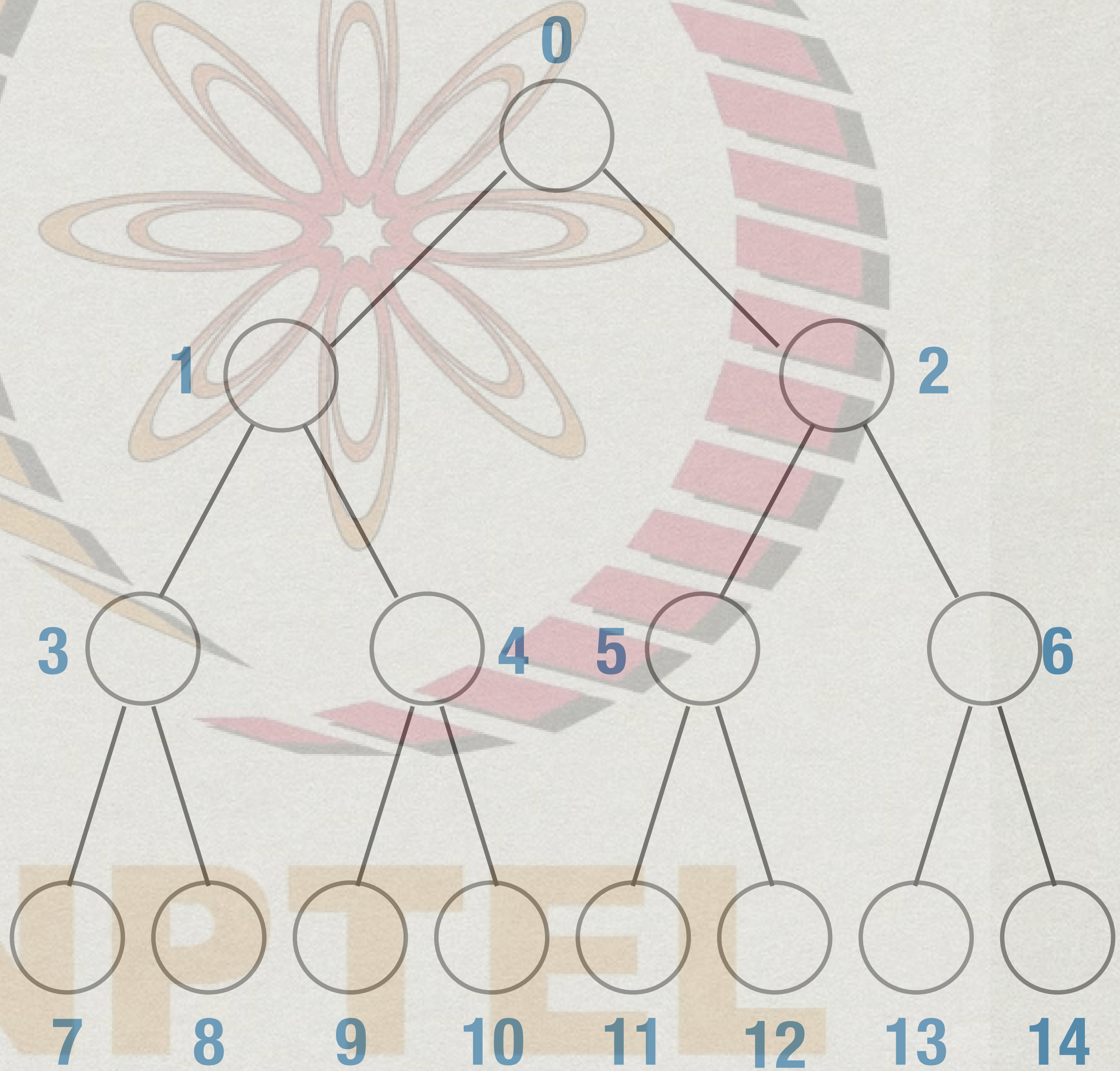
Building a heap, heapify( )

- Given a list of values $[x_1,x_2,...,x_N]$, build a heap
- Naive strategy
  - Start with an empty heap
  - Insert each x_j
  - Overall O(N log N)

Better heapify( )

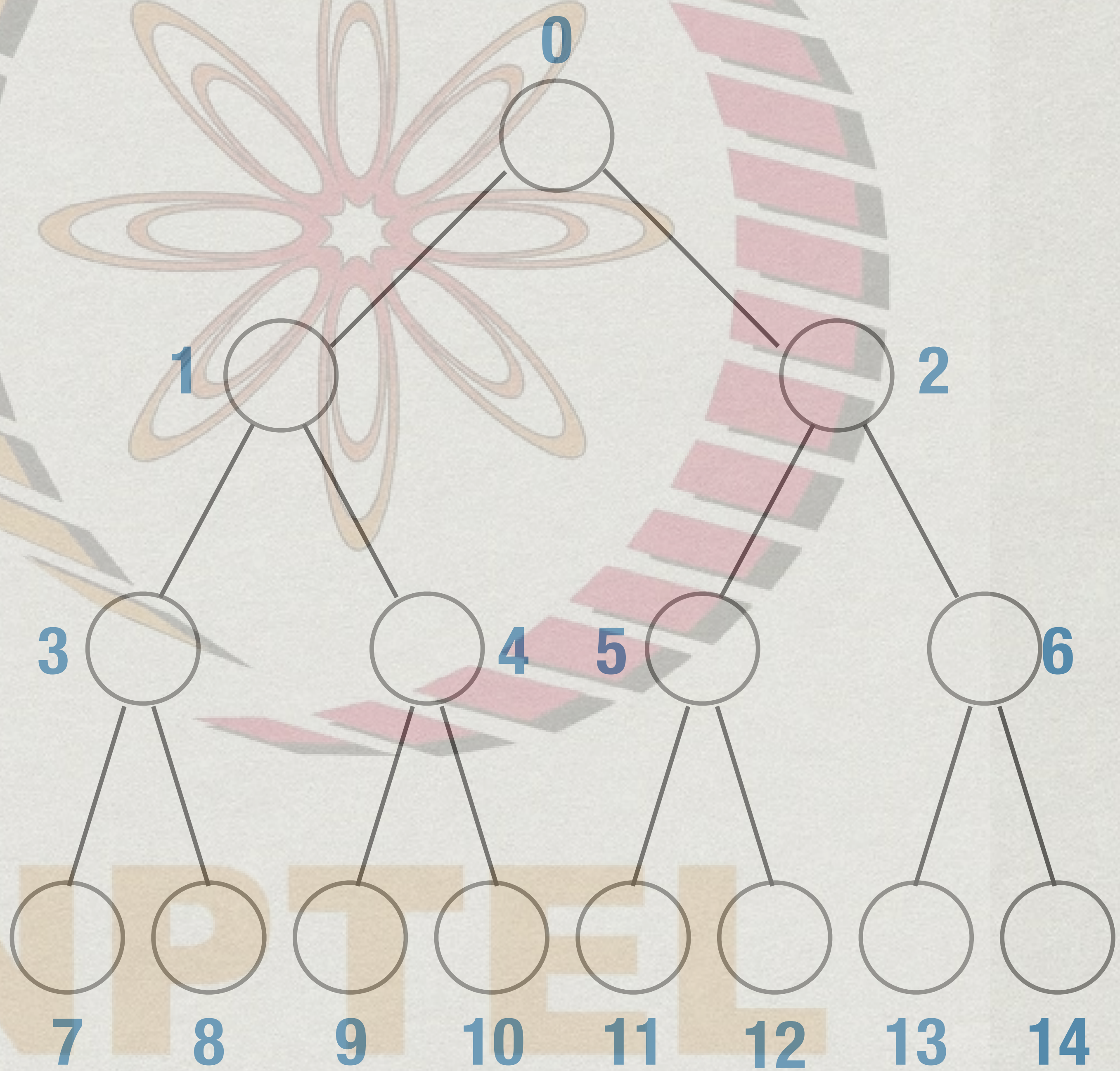
- Set up the array as $[x_1,x_2,\ldots,x_N]$
    - Leaf nodes trivially satisfy heap property
    - Second half of array is already a valid heap
- Assume leaf nodes are at level k
    - For each node at level k-1, k-2, … , 0, fix heap property
    - As we go up, the number of steps per node goes up by 1, but the number of nodes per level is halved
    - Cost turns out to be O(N) overall

Better heapify( )



N/2 nodes already satisfy heap property

Better heapify( )



4 nodes, height 1 repair

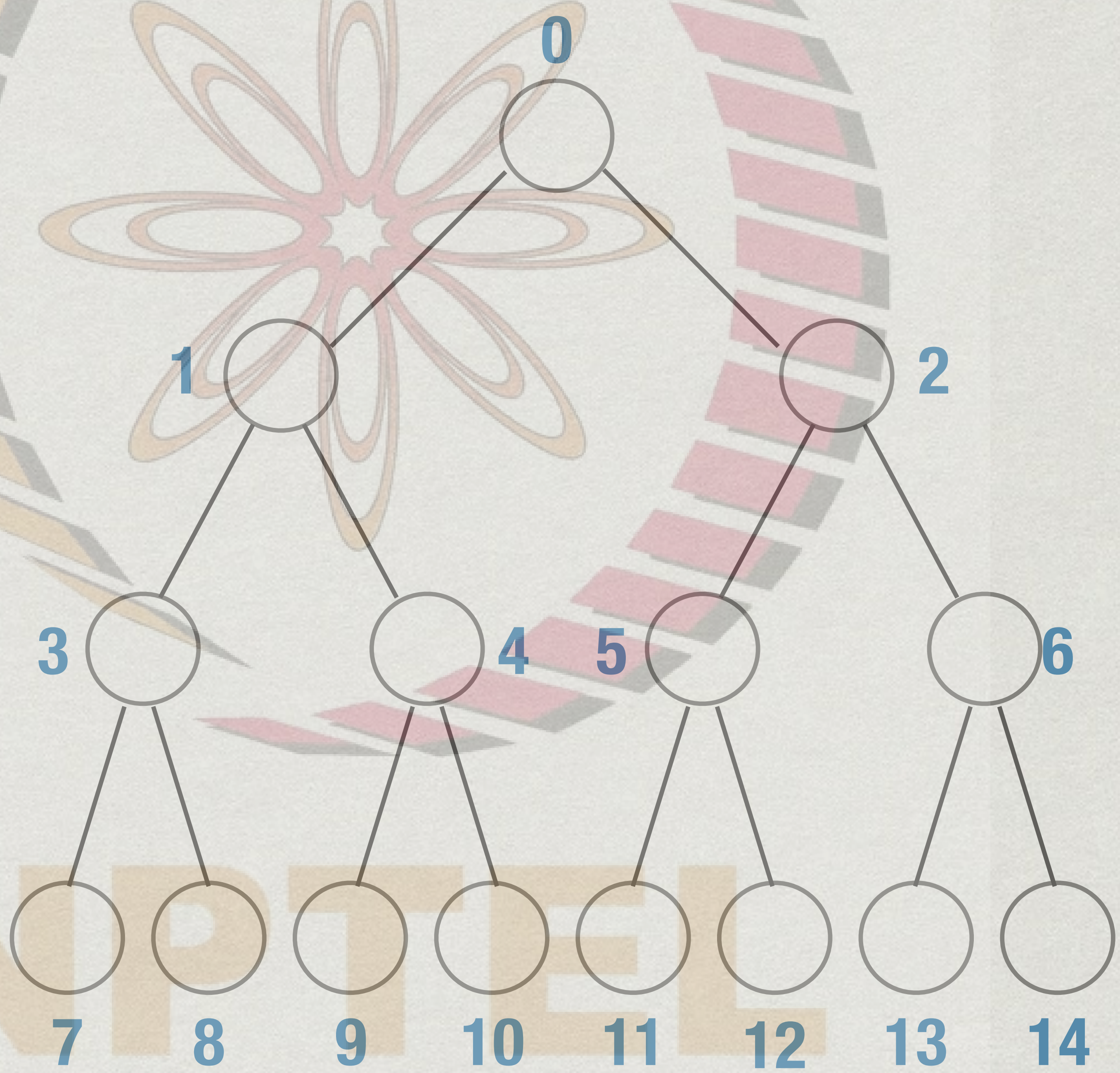
N/2 nodes already satisfy heap property

Better heapify( )

2 nodes,
height 2 repair

4 nodes,
height 1 repair

N/2 nodes
already satisfy
heap property



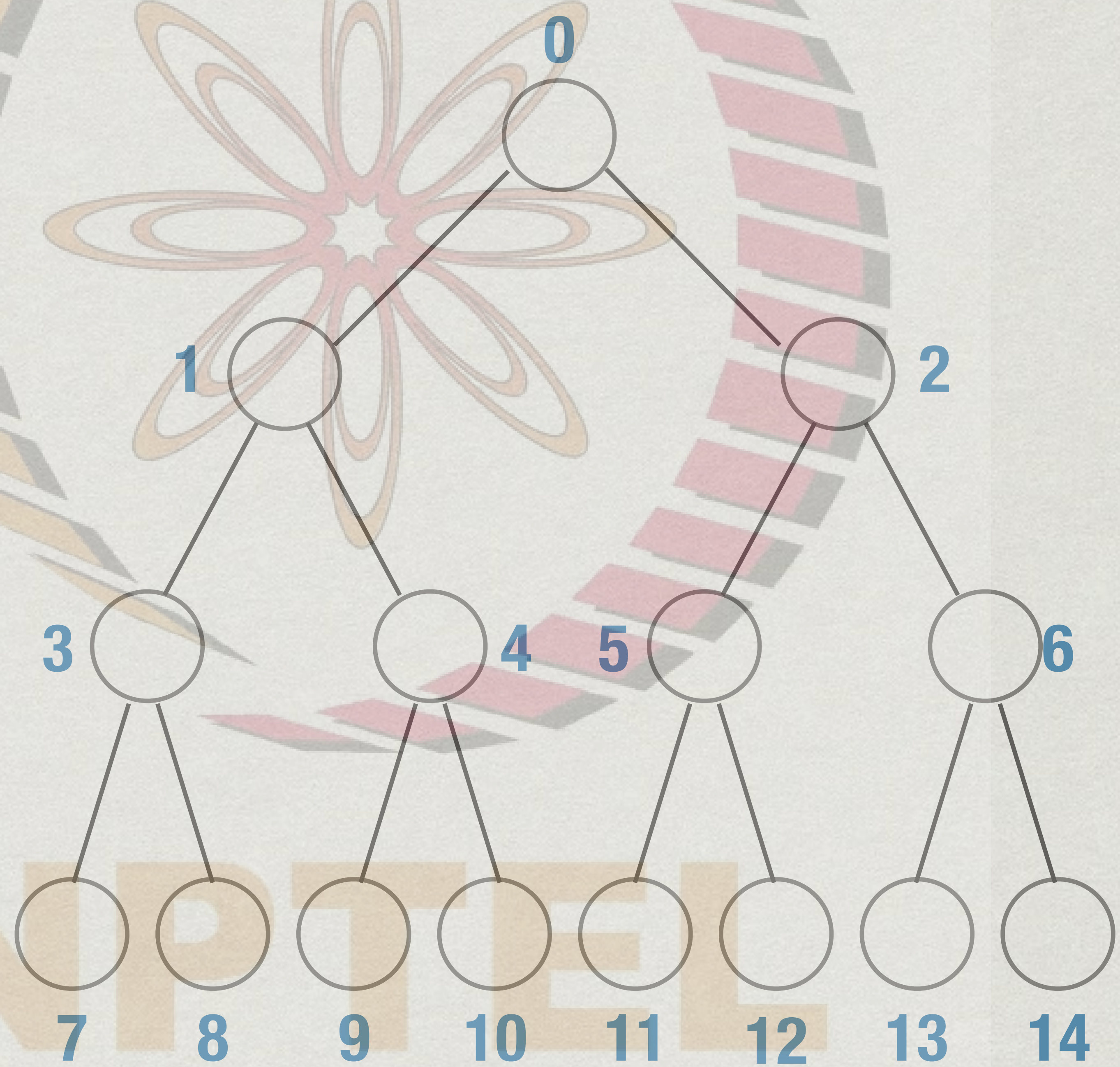
Better heapify()

1 node,
height 3 repair

2 nodes,
height 2 repair

4 nodes,
height 1 repair

N/2 nodes
already satisfy
heap property



Summary

- Heaps are a tree implementation of priority queues
  - insert( ) and delete_max( ) are both O(log N)
  - heapify( ) builds a heap in O(N)
  - Tree can be manipulated easily using an array
- Can invert the heap condition
  - Each node is smaller than its children
  - Min-heap, for insert( ), delete_min( )