Priority Queue ADT:

Min-ordered PQ:

insert (x)

removeMin( )

Max-ordered PQ:

insert (x)

removeMax( )

Data structures for PQ (assume min-ordered):

Unsorted linked list

insert $\theta(1)$ time

removeMin $\theta(n)$ time

Sorted linked list

insert $\theta(n)$ time

removeMin $\theta(1)$ time

Heap = tree-based data structure for implementing a PQ

Goal: both insert and removeMin run in $O(\lg n)$ time

Binary heap = Binary tree with these two properties:

[Heap-ordering property]

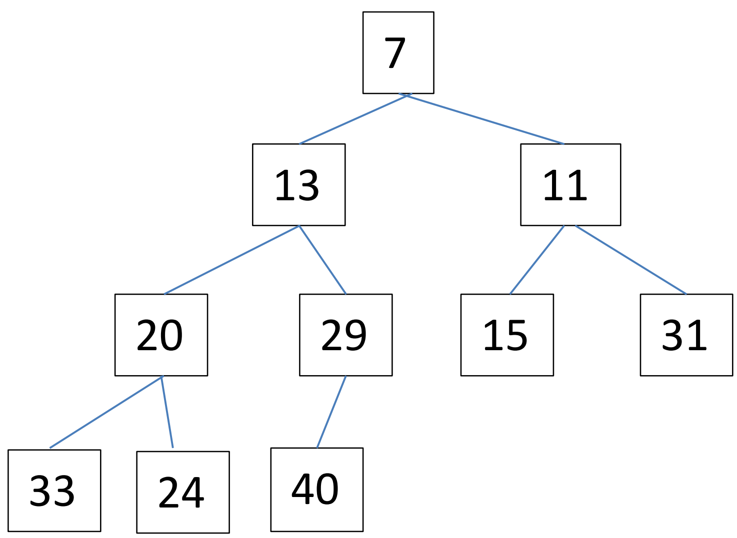
The key at each node is ≤ the keys of its children

We're still assuming a min-ordered heap

[Heap-structure property]

At most one level of the tree is only partially full

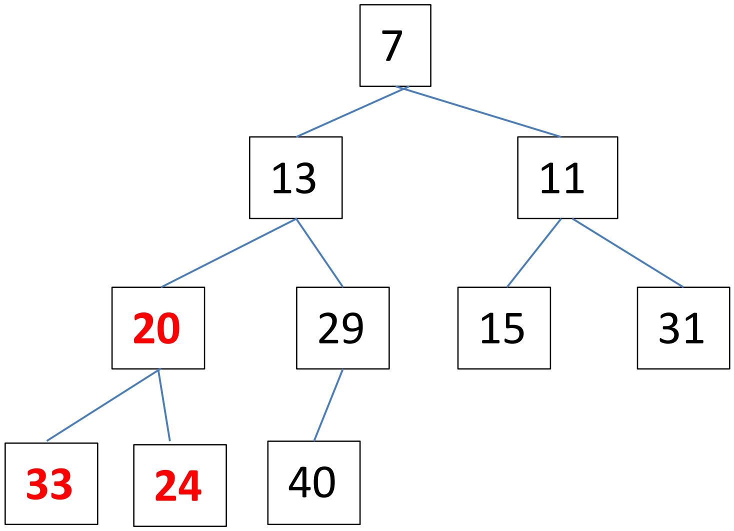
All nodes on this level are as far to the left as possible



Height of binary heap = $\theta(\lg n)$

Both insert and removeMin will run in $\theta(\text{height}) = \theta(\lg n)$ time

Array representation of binary heap: store keys in level-order



A [1…MAX]
n = 10 ≤ MAX

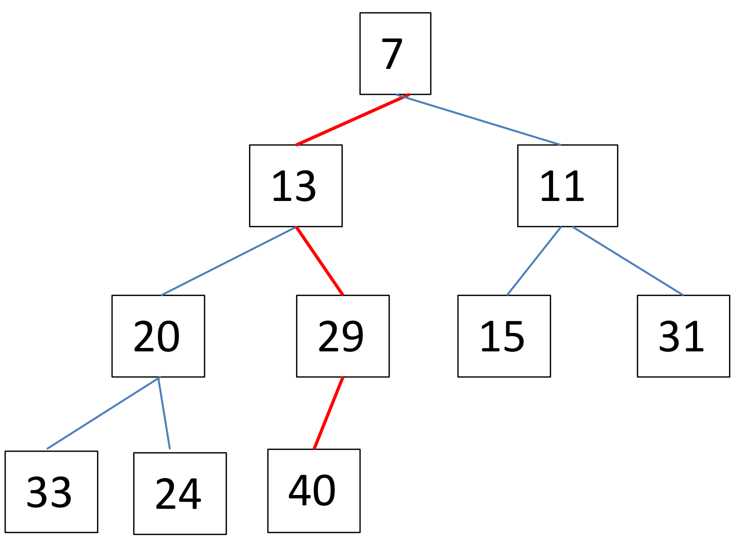
index	1	2	3	**4**	5	6	7	**8**	**9**	10	11	12	…	MAX
A	7	13	11	**20**	29	15	31	**33**	**24**	40				

parent(index) = index/2
leftChild(index) = 2*index
rightChild(index) = 2*index + 1

Linked binary tree representation of binary heap:

```
class Node {
	ElementType key;
	Node parent, left, right;
}
class BinaryHeap {
	Node root;
	int n;
}
```

How to find the location of the n^{th} node in $\theta(\lg n)$ time?
Use binary representation of n
Example: $n = 10 = 1010_2$



1: q = root
0: q = q.left
1: q = q.right
0: q = q.left

Insert operation:

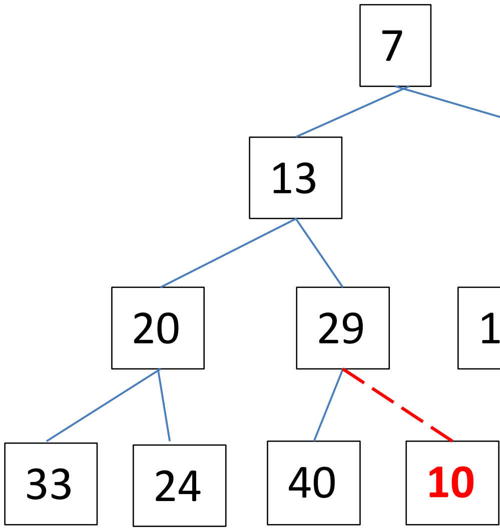
```
insert (x) {
	n += 1;
	add new leaf node q with key x;
	while (q.parent != null  &&  q.key < q.parent.key) {
		swap (q.key, q.parent.key);
		q = q.parent;
	}
}
```

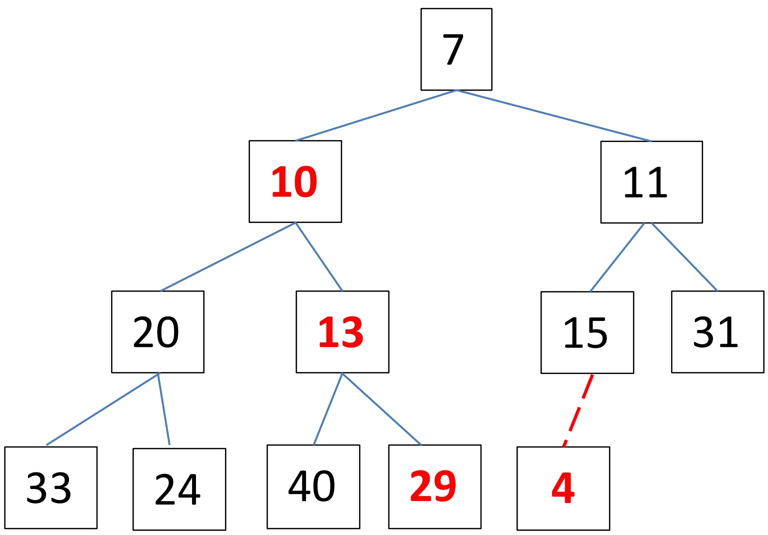
Running time = θ(height) = θ(lg n)

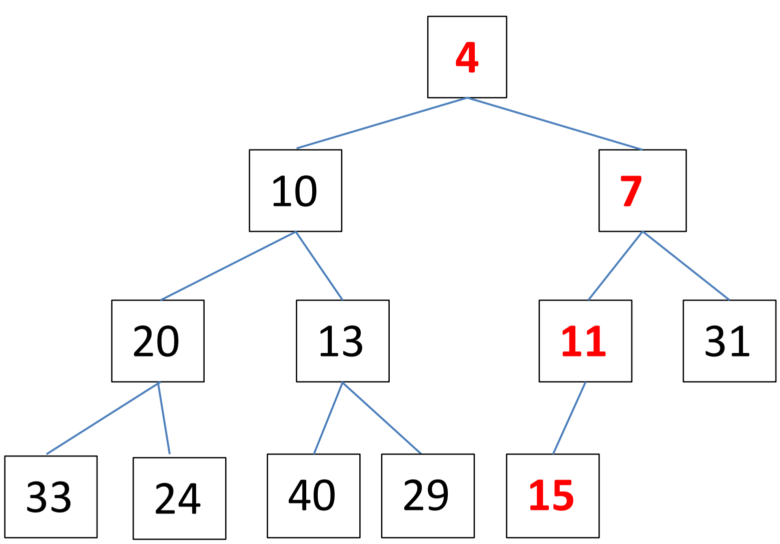
Using the array representation of binary heap:

```
insert (x) {
	if (n==MAX) throw exception;
	n += 1;
	A[n] = x;
	q = n;
	while (q>1  &&  A[q] < A[q/2]) {
		swap (A[q], A[q/2]);
		q /= 2;
	}
}
```

Example: Insert 10, 4







RemoveMin operation:

```
removeMin( ) {
    if (n==0) throw exception;
    x = root.key;
    find rightmost leaf node q on deepest level;
    root.key = q.key;
    remove node q from tree;
    n -= 1;
    p = root;
    while ((p.left != null  &&  p.left.key < p.key)
            || (p.right != null  &&  p.right.key < p.key)) {
        c = p.left;
        if (p.right != null  &&  p.right.key < p.left.key)
            c = p.right;
        swap (p.key, c.key);
        p = c;
    }
    return x;
}
```

Running time = θ(height) = θ(lg n)

RemoveMin can also be adapted to use the array representation of binary heap

Example: removeMin