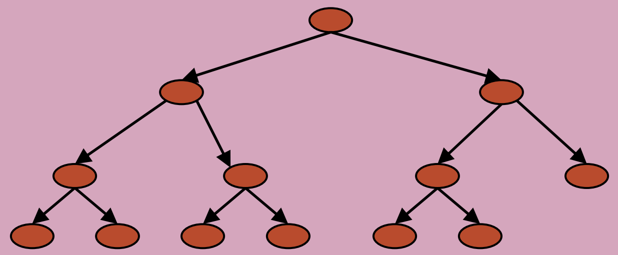
Priority Queues – Binary Heaps

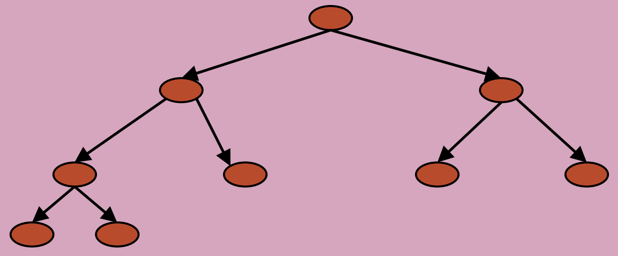
Heap <u>**Structure**</u> Property

- A binary heap is a <u>***complete***</u> binary tree.

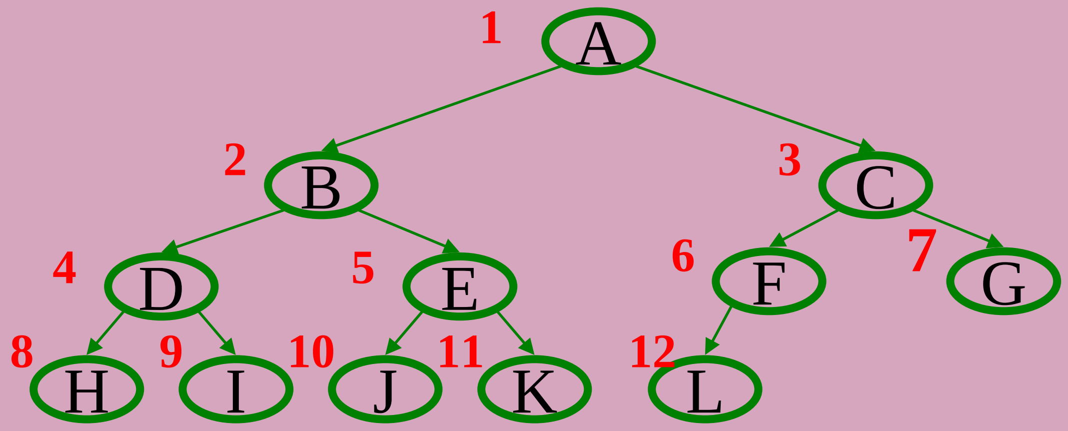
<u>**Complete binary tree**</u> – binary tree that is completely filled, with the possible exception of the bottom level, which is filled left to right.

Examples:





Representing Complete Binary Trees in an Array



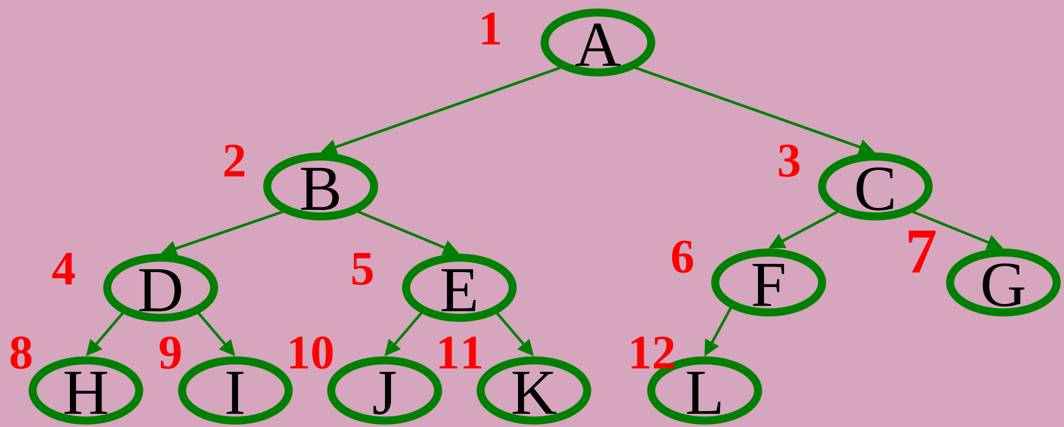
From node **i**:

left child:
right child:
parent:

implicit (array) implementation:

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Why this approach to storage?



implicit (array) implementation:

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

From node **i**:

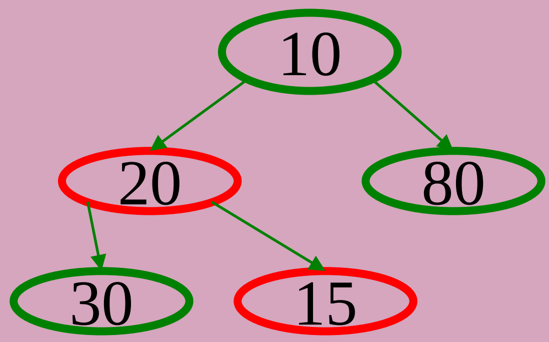
left child:

right child:

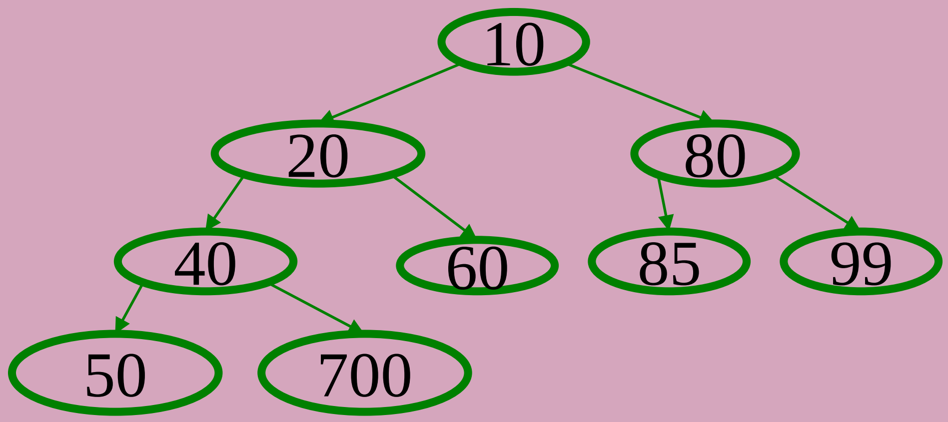
parent:

Heap **<u>Order</u>** Property

<u>Heap order property</u>: For every non-root node X, the value in the parent of X is less than (or equal to) the value in X.

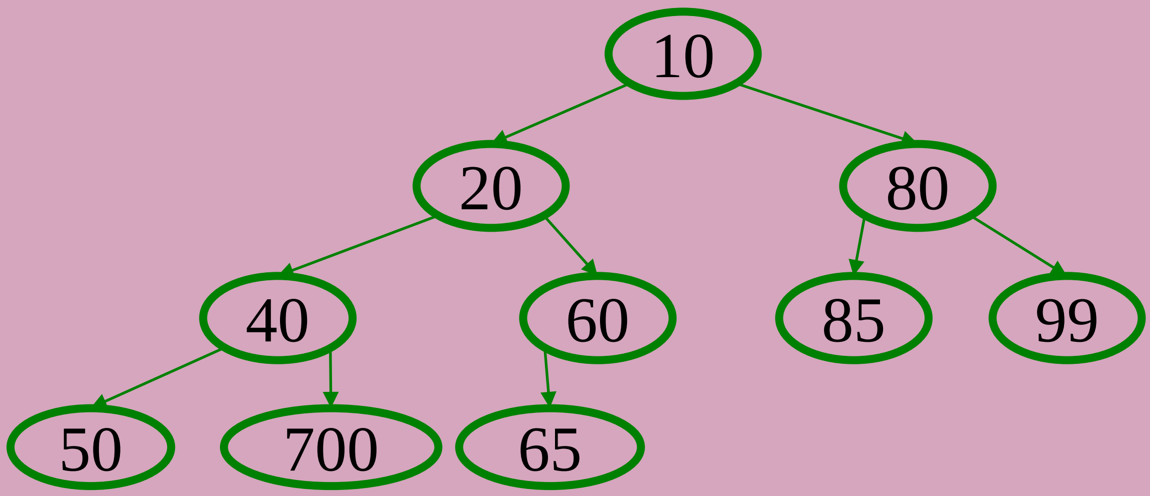


not a heap



Heap Operations

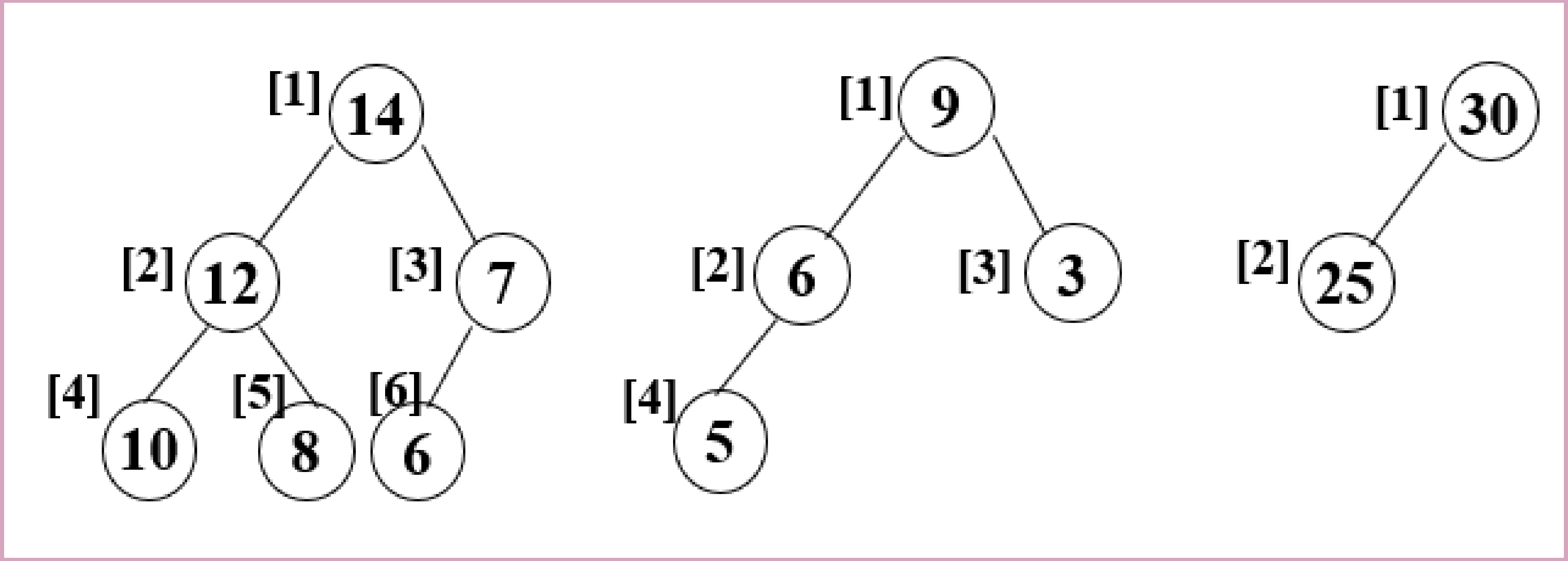
- findMin:
- insert(val): percolate up.
- deleteMin: percolate down.



Heap

- A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children. A *max heap* is a complete binary tree that is also a max tree.
- A *min tree* is a tree in which the key value in each node is no larger than the key values in its children. A *min heap* is a complete binary tree that is also a min tree.

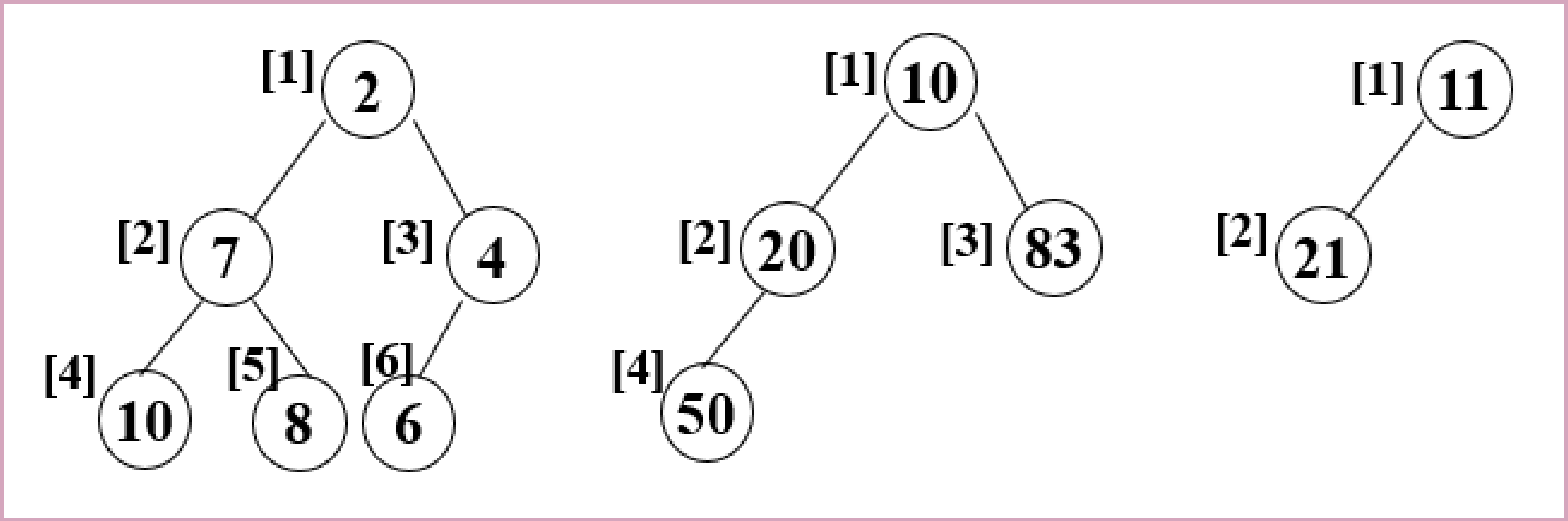
Sample max heaps



Property:

The root of max heap (min heap) contains the largest (smallest).

Sample min heaps



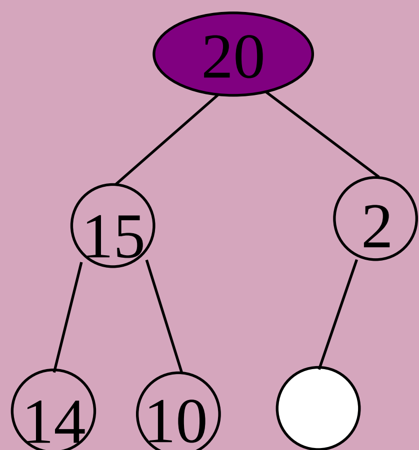
Data Structures

- unordered linked list
- unordered array
- sorted linked list
- sorted array
- heap

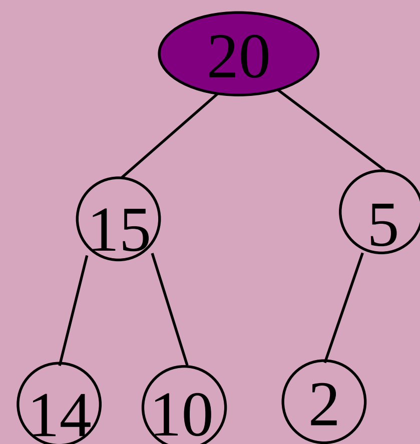
Priority queue representations

Representation	Insertion	Deletion
Unordered array	$\Theta(1)$	$\Theta(n)$
Unordered linked list	$\Theta(1)$	$\Theta(n)$
Sorted array	$O(n)$	$\Theta(1)$
Sorted linked list	$O(n)$	$\Theta(1)$
Max heap	$O(\log_2 n)$	$O(\log_2 n)$

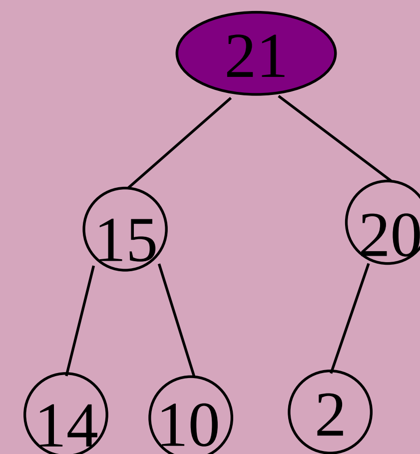
Example of Insertion to Max Heap



initial location of new node



insert 5 into heap



insert 21 into heap

Insertion into a Max Heap

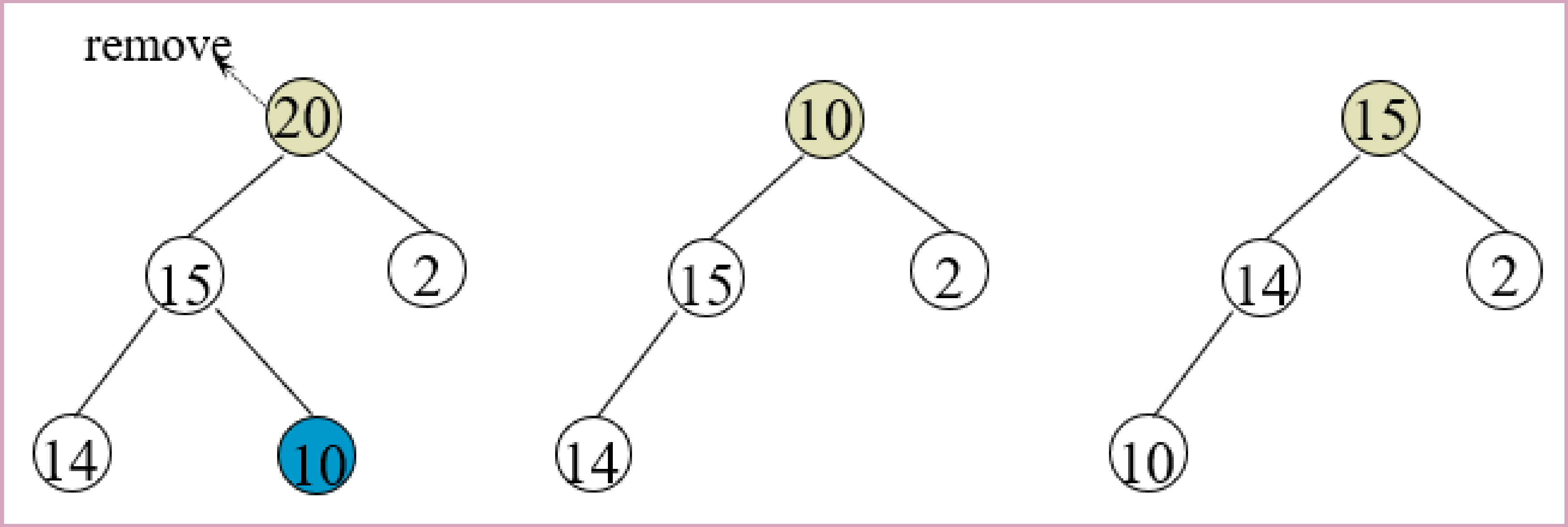
```
insert_max_heap(heap,n,x):
  n += 1
  i = n
  while ((i!=1)&&(x > heap[i/2])):
    heap[i] = heap[i/2]
    i /= 2

  heap[i]= x
```

$$2^k-1=n \implies k=\lceil \log_2(n+1) \rceil$$

$O(\log_2 n)$

Example of Deletion from Max Heap



Deletion from a Max Heap

```
deleteMax(heap,n):
   x = heap[1]
   heap[1] = heap[n]
   n -= 1
   MAX-HEAPIFY(heap,n,1)
   return x
```

```
MAX-HEAPIFY(heap,n,i):
  parent = i, child = 2 * i, temp = heap[i]
  while (child <= n):
    if (child < n)&& heap[child] < heap[child+1])):
            child++
    if (temp >= heap[child]):
         break
    heap[parent] = heap[child]
    parent = child, child *= 2

  heap[pt] = temp
```

Complexity: O(log n)

Heapsort Algorithm

- The algorithm
  - (Heap construction) Build heap for a given array (either bottom-up or top-down)
  - (Maximum deletion) Apply the root-deletion operation n-1 times to the remaining heap until heap contains just one node.
- An example: 2 9 7 6 5 8

HEAP SORT

```
Algorithm heapSort(A[1..n])
    for (i = n/2 to 1):
        MAX-HEAPIFY(A,n,i)

    for (i = n to 1):
        Swap(A, 1, i);
        MAX-HEAPIFY(A, i-1,1)
```

Analysis of Heapsort

Recall algorithm:

$\Theta(n)$ 1. Bottom-up heap construction

$\Theta(\log n)$ 2. Root deletion

Repeat 2 until heap contains just one node.

***n* – 1 times**

Total: $\Theta(n) + \Theta(n \log n) = \Theta(n \log n)$

- **Note:** this is the worst case. Average case also $\Theta(n \log n)$.