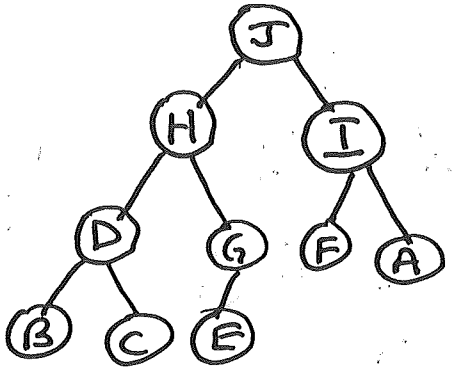
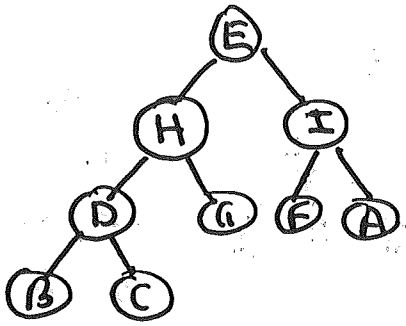
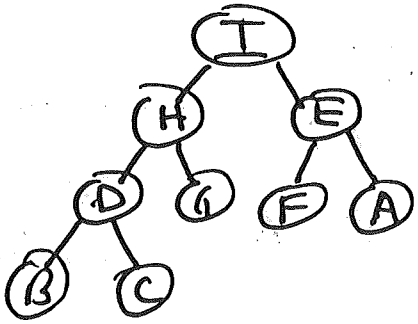
Remove root element and "Re-Heap Down"



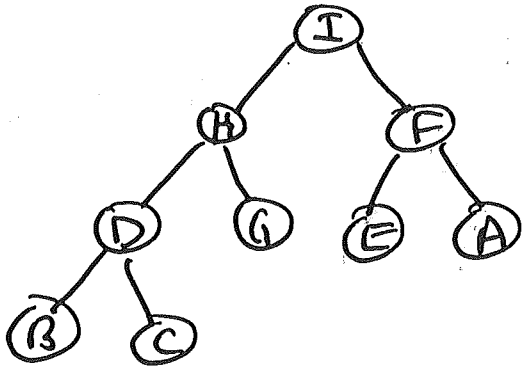
Remove J and move E to J's former position.



Shape property preserved. But order property destroyed. So, begin process of "re-heaping" by interchanging E and I.

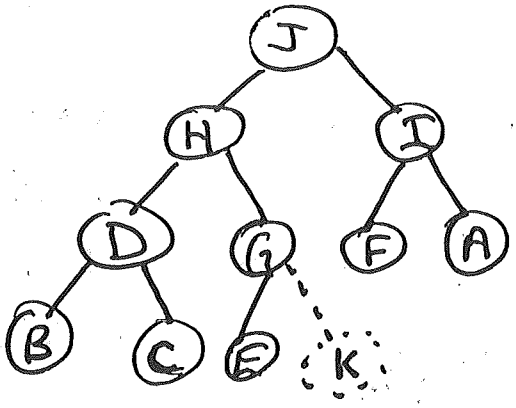


Still not a heap, so have to continue by interchanging E and F.

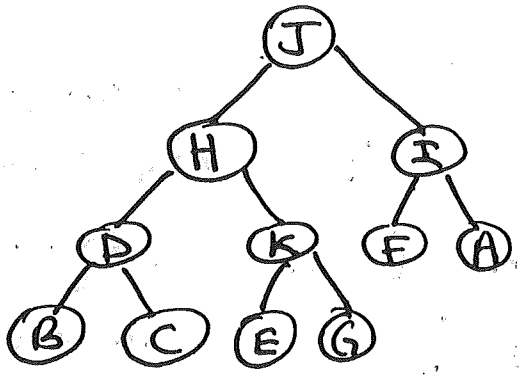


Now have a heap.

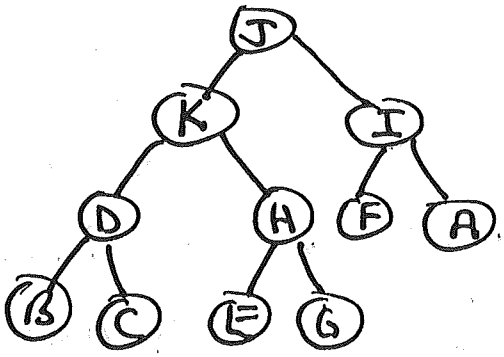
Add an element and "Re-Heap Up"



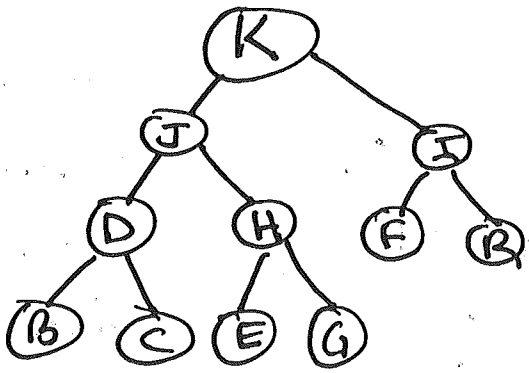
Add new element K as rightmost element of lowest (non-full) level.



Shape property preserved. But order property destroyed. So, begin process of "re-heaping" by interchanging G and K.



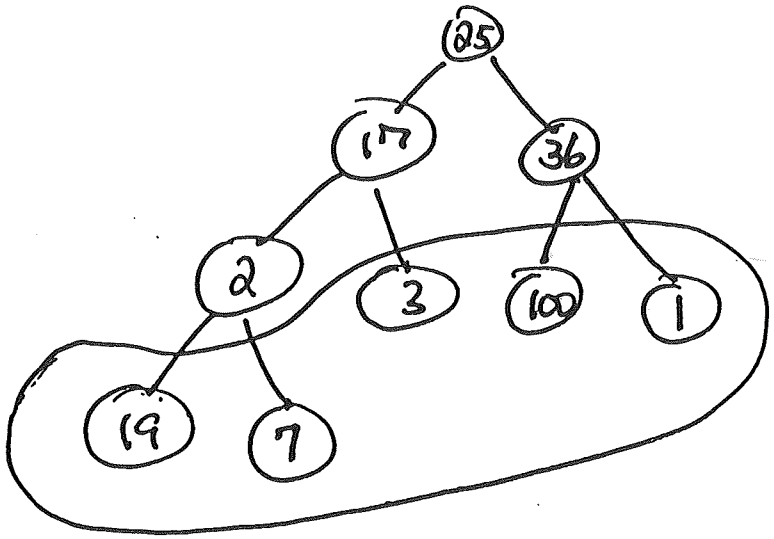
Still not a heap, so have to continue by interchanging H and K.



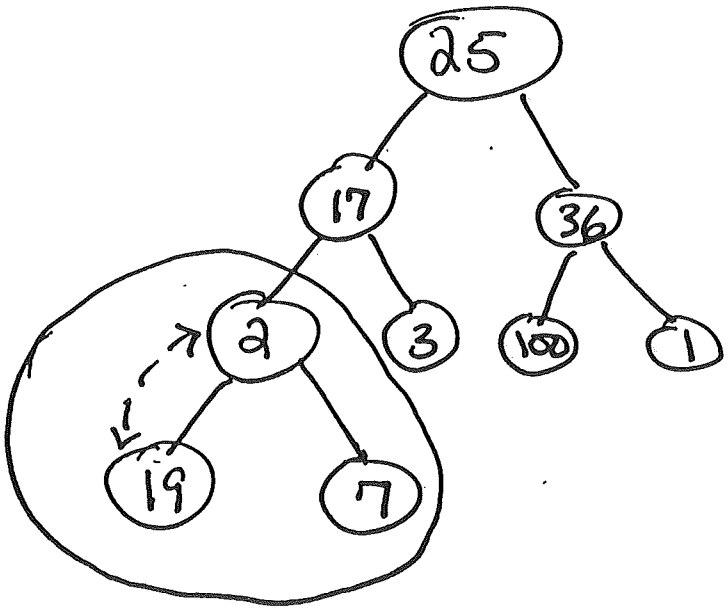
Still not a heap, so interchange J and K, after which we have a heap.

Building a Heap
(from a random, i.e. unsorted, array)

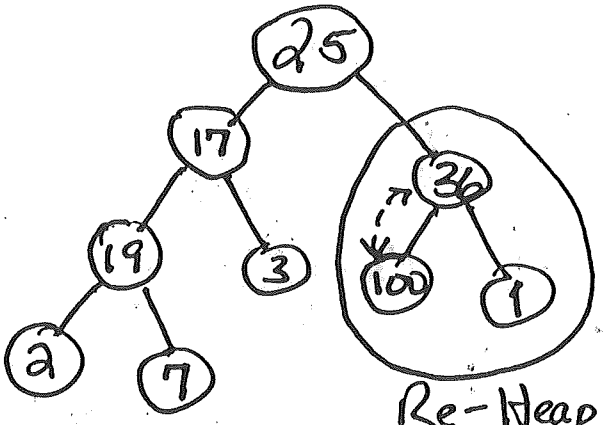
25	17	36	2	3	100	1	19	7
0	1	2	3	4	5	6	7	8



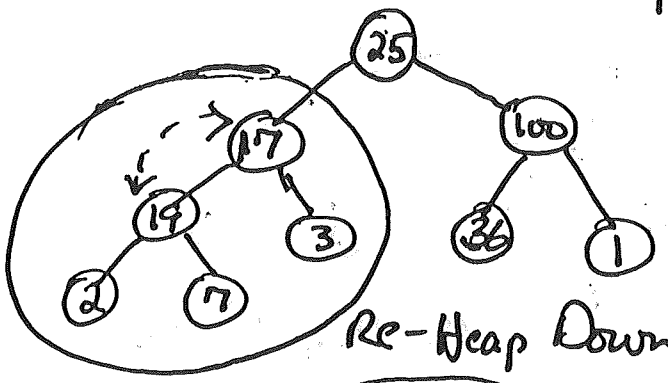
These are all heaps.



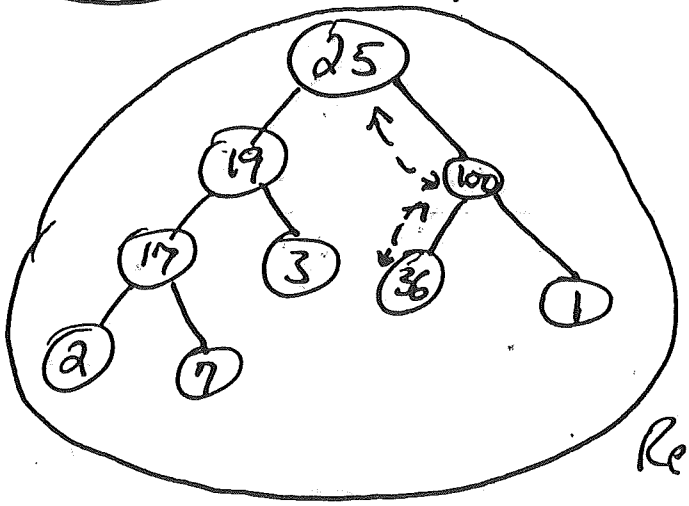
Re-Heap Down



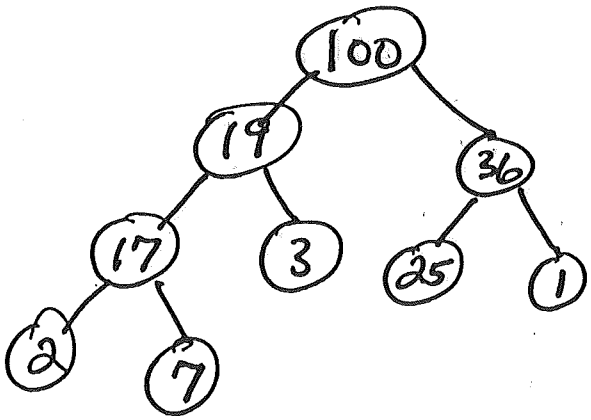
Re-Heap Down



Re-Heap Down



Re-Heap Down



We now have a heap.

Pseudocode for heap algorithms

Algorithm ReheapDown

```
if tree is not a leaf
    Set "maxChild" to index of child with larger value
    if value of tree < value of maxChild
        Swap the two values
        ReheapDown starting now at maxChild
```

Algorithm ReheapUp

```
if bottom > tree
    Set "parent" to index of parent of bottom
    if value of parent < value of bottom
        Swap the two values
        ReheapUp starting now at parent
```

Algorithm BuildHeap

```
for each index from first non-leaf back up to root
    ReheapDown starting at that index
```