Heap Trees

PRIORITY QUEUES

- Here are marks obtained by students in a class size of 150.
- [62 76 53 45 3 27 41 22 78 17 69 38 85 …….. 12 60 37]

- How many operations are needed to get marks obtained by first, second, and third positions.
- What would be the best way to do this?
- Scheme 1: Search for the highest O(N)
- Search for the next highest O(N)
- Search for the next highest O(N)
- How to find the marks of 10^{th} highest student?
- 10 N operations

- Array with random elements

- Deletion: O(n)
- Insertion: O(1)

- Suppose we have a very large array containing 8000 integers (N) with random values,
- [87 45 3 27 41 58 22 8 17 69 38 82 7 12 60 37 …….2 76 18 46]
- and we wanted to remove 100 smallest items (k) from there one by one.
- What would be the best way to do this?
- Scheme 1: Search for the smallest and delete it O(N)
- Do this K times O(kN)
- Inserting new item O(1)

- Array with SORTED elements

- Deletion: O(1)
- Insertion: O(n)

- Suppose we have a very large array containing 8000 integers (N) with random values,
- [87 45 3 27 41 58 22 8 17 69 38 82 7 12 60 37 …….2 76 18 46]
- and we wanted to remove 100 smallest items (k) from there one by one.
- What would be the best way to do this?
- Scheme 1: Search for the smallest and delete it O(N) 8000
- Do this K times O(kN) 80000
- Inserting new item O(1)
- Scheme 2: Sort the array O(N log N) 104000
- Delete K items O(1) 100
- Insert new item in sorted array O(N) 8000
- Which is better?

Special offers to customers!

- Names of 4000 customers are stored in company database in an array. Timestamp of customer registration is stored along with each name. *Array is sorted alphabetically.*

- The company wants to reward the first 100 registrations with some special offers.

- What is the most efficient way to obtain the names of first 100 customers?

Special offers to customers!

- Names of 4000 customers are stored in company database in an array. Timestamp of customer registration is stored along with each name. *Array is sorted alphabetically.*

- The company wants to reward the first 100 registrations with some special offers.

- What is the most efficient way to obtain the names of first 100 customers?
- - 1. Linear search each time
- - 2. Sort the array and pick top 100

Computer Processor job scheduling

- A processor has list of waiting time for each of the jobs.
- It is desired to follow *'shortest job first'* policy.
- The list is stored in an array
- [28 12 3 27 18 41 15 5 22 7 11]
- How to continuously do the jobs on priority basis, by always picking up the shortest job from the array, while more items keep on getting added
- Both deletions and insertions on array

Handling wait listed passengers

- Once regular passengers have checked in, it is time for waitlisted passengers. It is not first come first served.
- The waiting list has names of passengers *with their priority points* (maybe based on how frequently they fly with the airline).

- [Harshit 10 mukul 230 Varsha 520 Jyoti 96 Pramod 360 Srinivas 30]

- The passengers are in a queue, but the airline prefers some passengers over others depending on their priority. How to do it?
- Sort the array in terms of priority points O(n log n)
- Delete the desired entry O(1)
- Insert a new passenger O(n)

Handling wait listed passengers

- Once regular passengers have checked in, it is time for waitlisted passengers. It is not first come first served.
- The waiting list has names of passengers *with their priority points* (maybe based on how frequently they fly with the airline).

- [Harshit 10 mukul 230 Varsha 520 Jyoti 96 Pramod 360 Srinivas 30]

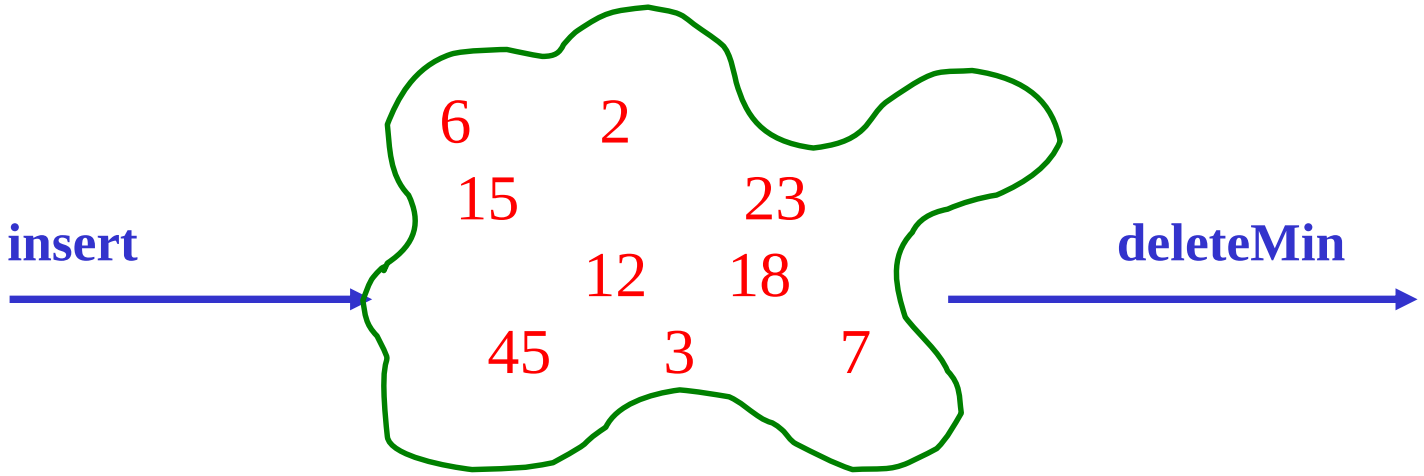
- What is the most efficient way to operate the waiting list?

- Form a Priority Queue.
- It is a new Data Structure

Priority Queues

Queues that Allow Line Jumping

- Need a new ADT
- Operations: 1. Insert a new Item,
  2. Remove the “Best” Item



Priority Queue ADT

- Given n elements, form a Priority Queue
- Insert an element in Priority Queue
- Delete the element with highest priority

Applications of Priority Queues

- Select shortest print jobs first
- Forward packets on routers in order of urgency
- Process scheduler in computer may give priority to short jobs
- Rewarding customers on basis of priority
- Select most frequent symbols for compression

ADT Implementation

- Using an unsorted array
- Maintaining a sorted array

Which Implementation is better ??

	insert	deleteMin
Unsorted list (Linked-List)	O(1)	O(n)
Sorted list (Linked-List)	O(n)	O(1)

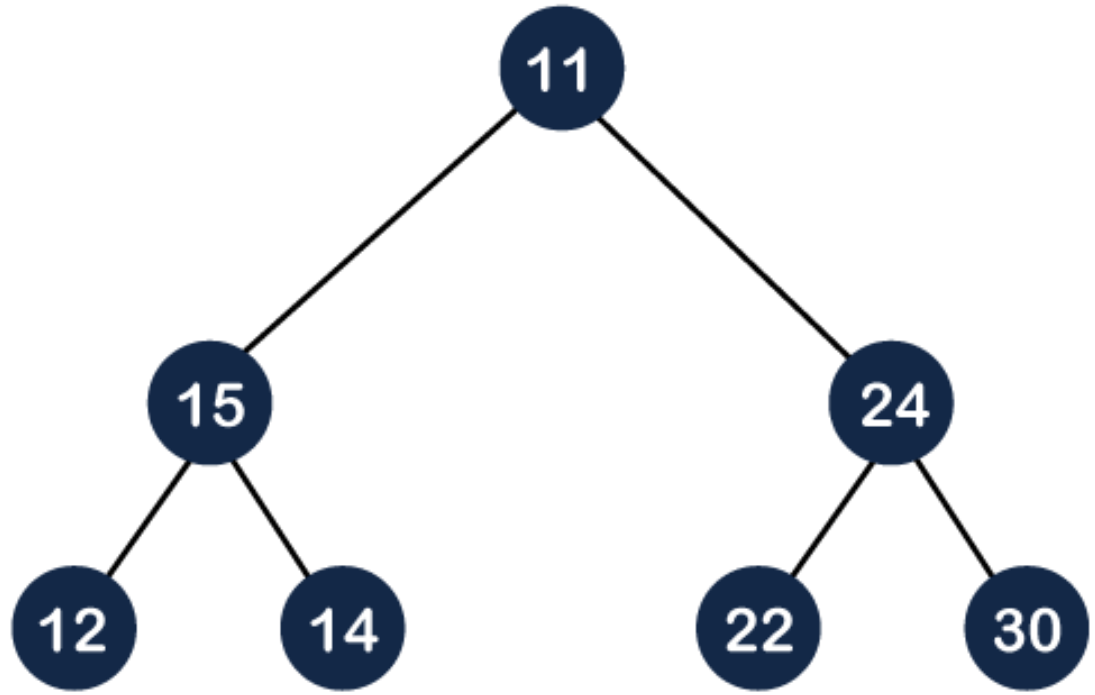
- Let us look for a solution which permits everything to be done in O(log n).

- How about using a Tree Data structure?
- Let us try using a Balanced Binary tree and get benefit of O(log n) complexity on all operations.

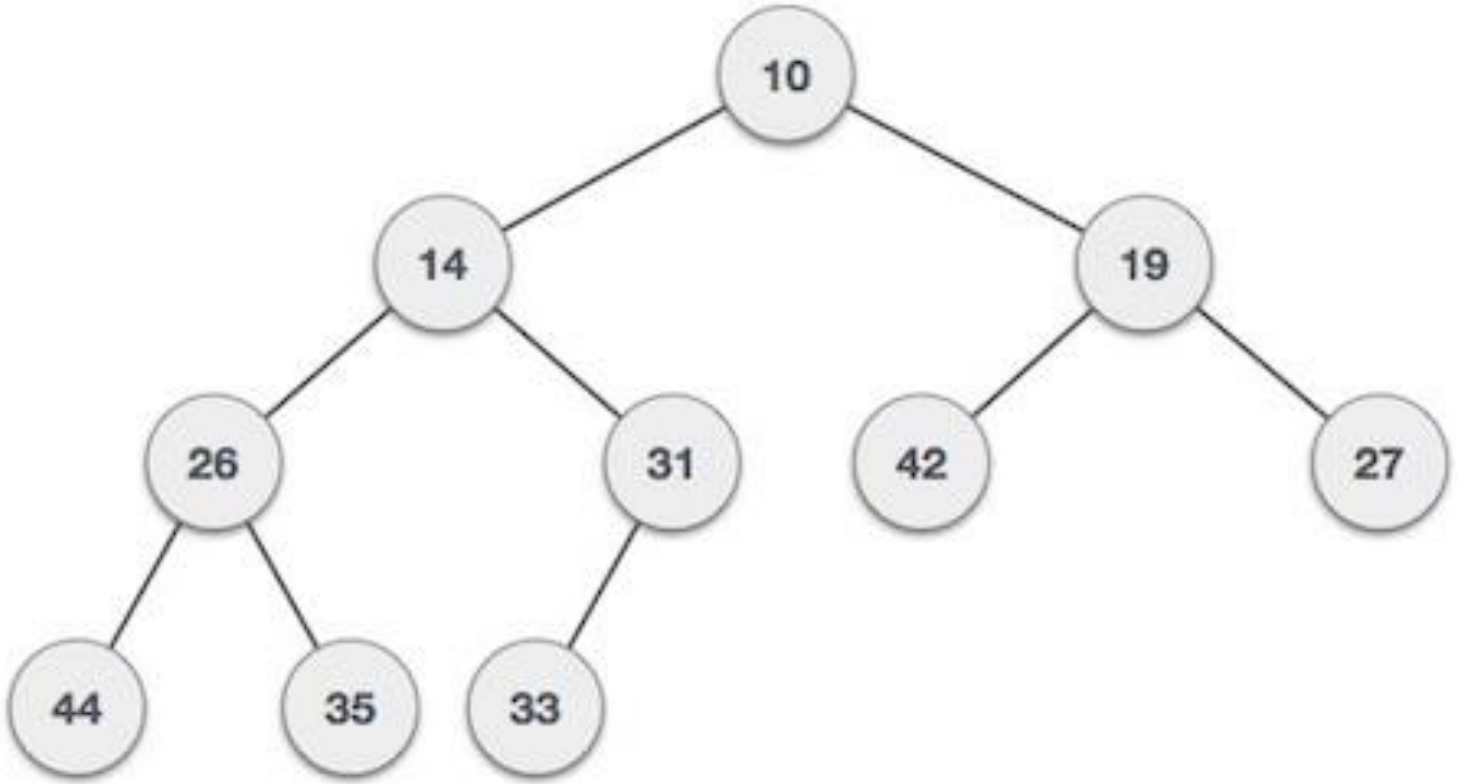
- A new concept: Binary Heap tree

Implementation using a HEAP Tree

- Priority Queue implemented using a Heap tree
- It is a complete tree, that is a completely balanced tree with no nodes missing
- The smallest value is stored at the root.
- So you can delete in O(1).
- After deletion, there will be one hole. It is filled up by next smallest value on the tree.


11
15
24
12
14
22
30

Min Heap



Implementation using Binary Tree

- If highest priority means smallest value, it will be called a min-heap tree
- Store minimum value at root of min-heap tree
- The tree structure will help to provide O(log n) worst case for both inserting new element and deleteMin,

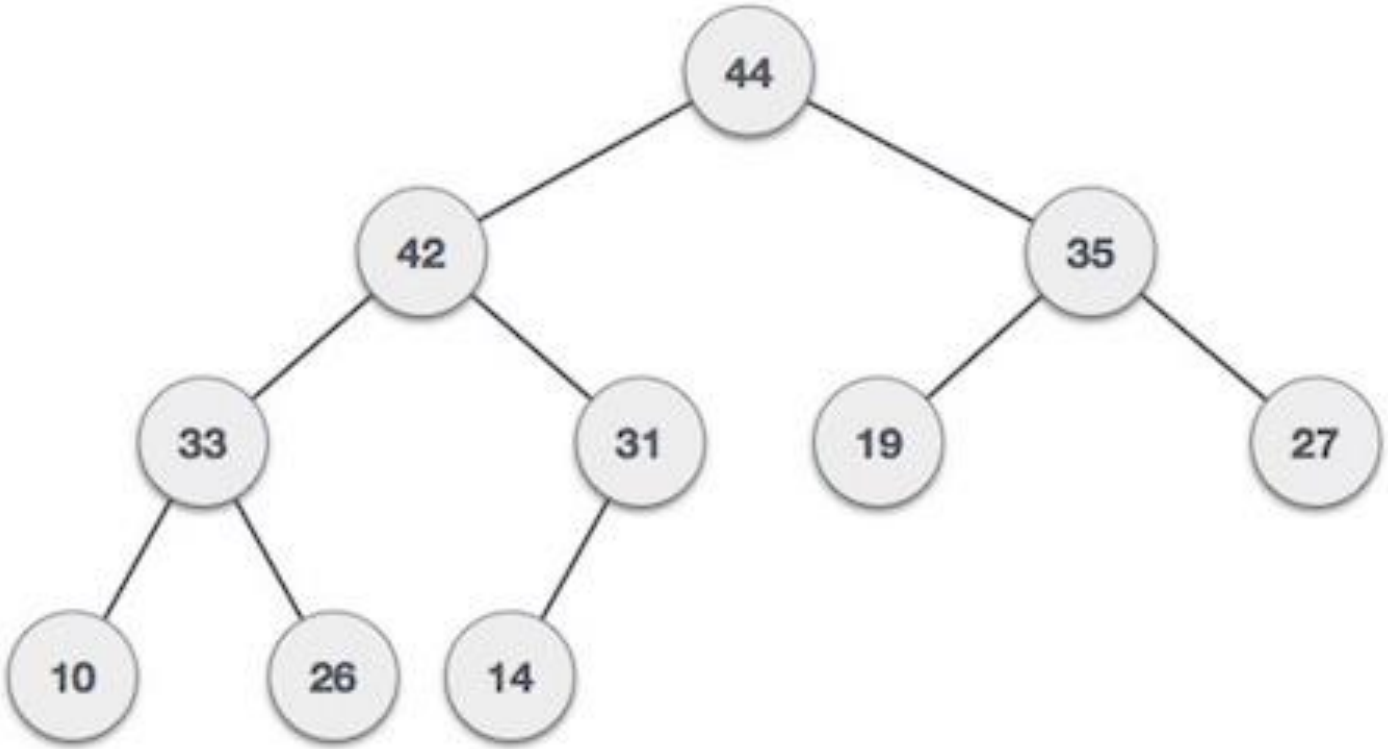
Implementation using Binary Heap Tree

- Since there are no holes on the tree, it can be represented using a simple array.

- If highest priority means largest value, create a max-heap tree

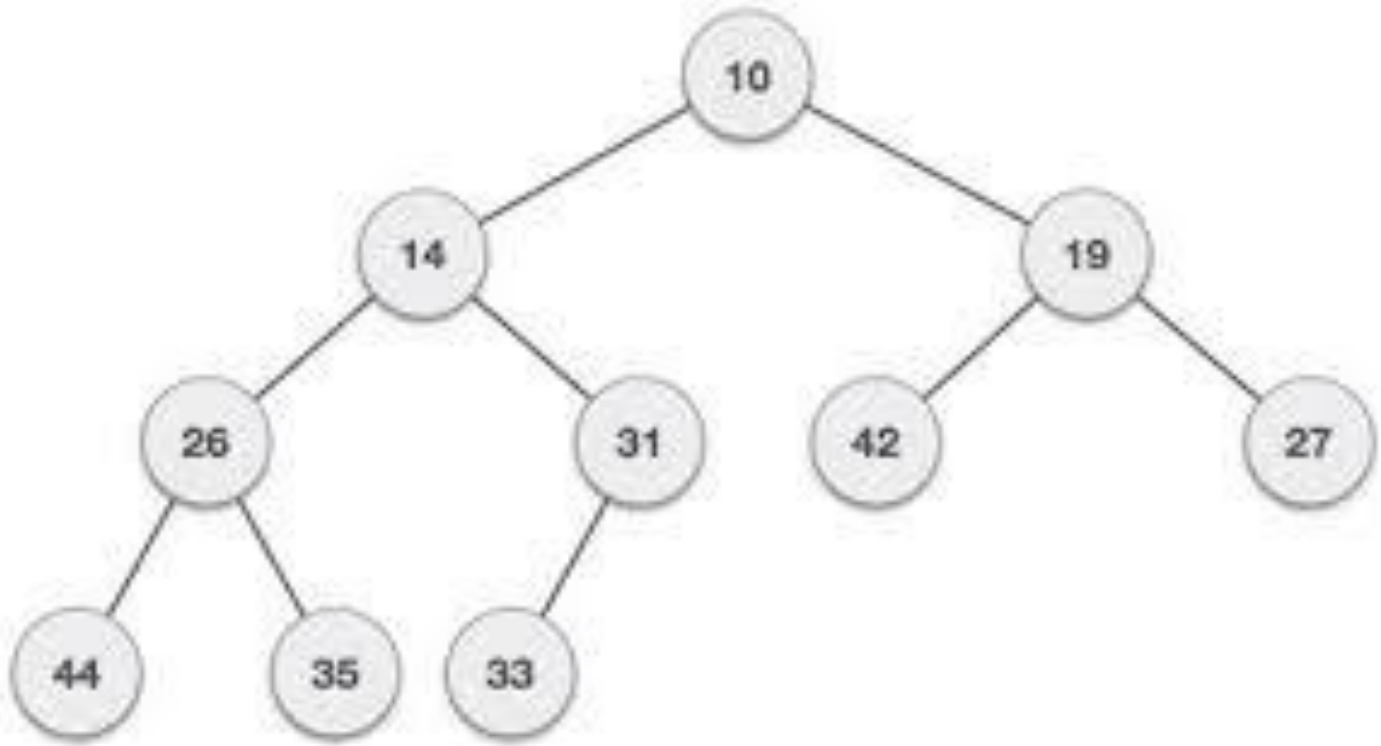
- The maximum value will be stored at root of max-heap tree

Max Heap



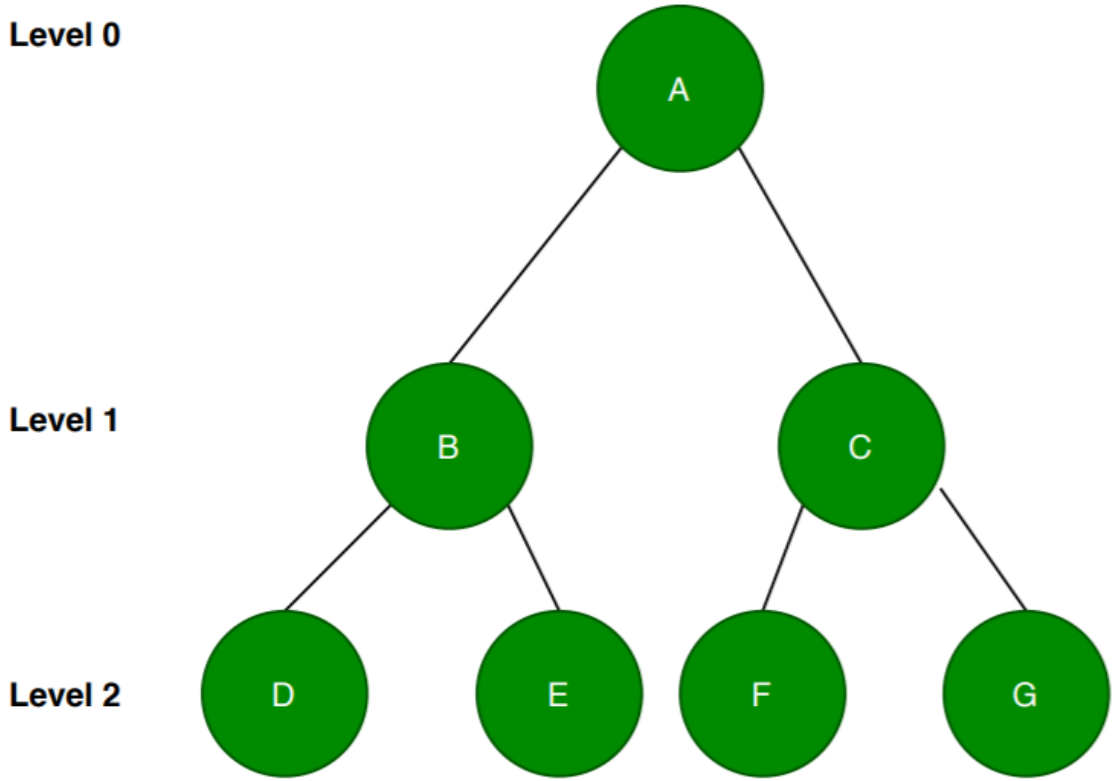
Binary Heap has 2 Properties

1. Structure Property
2. Ordering Property


10
14
19
26
31
42
27
44
35
33

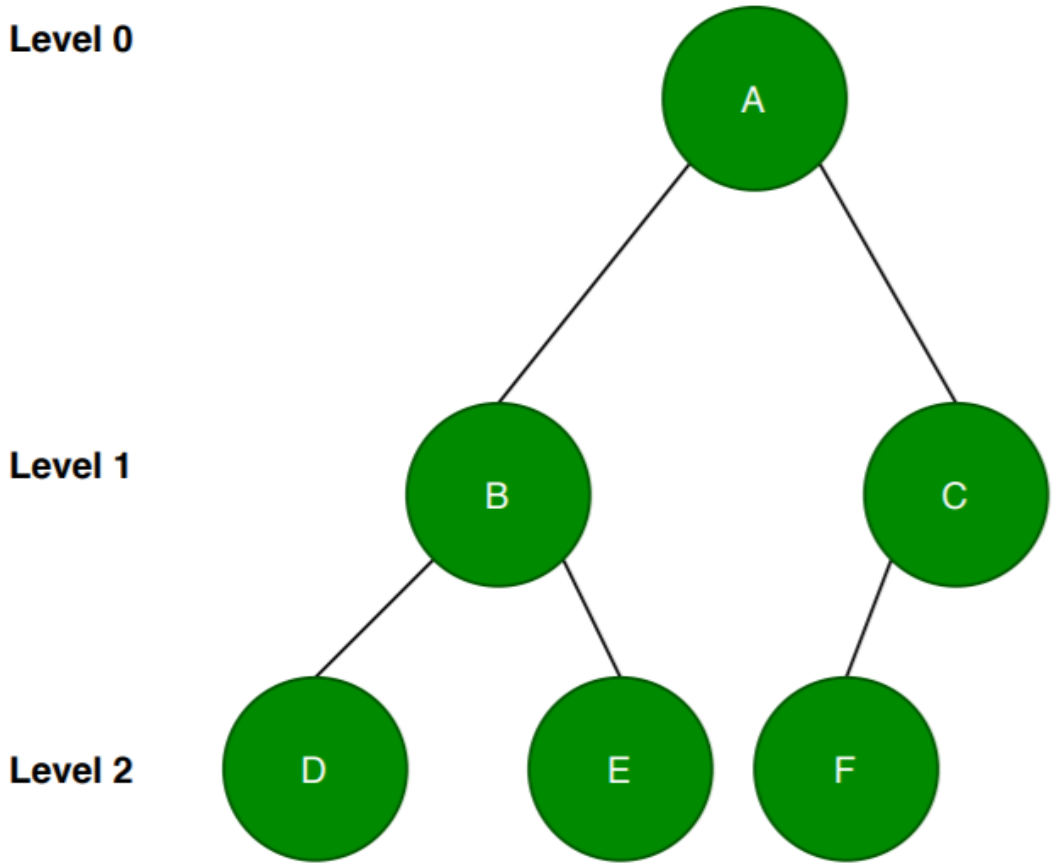
1. Structure Property

- The heap tree has to be a Complete Binary Tree

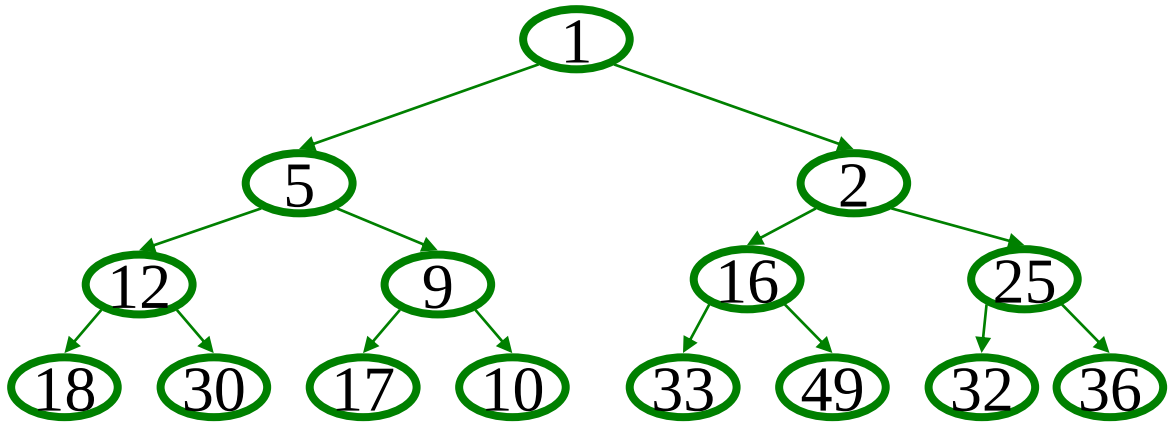


1. Structure Property

- This is also a Complete Binary Tree



A binary heap is a ***complete*** binary tree.



height h

$2^{h+1} - 1$ nodes

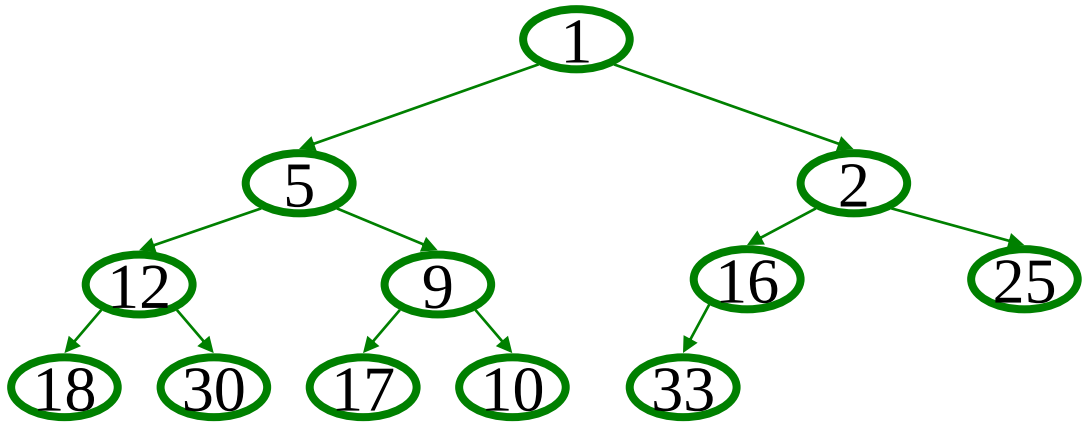
$2^{h} - 1$ non-leaves

2^{h} leaves

Note: Half the elements of Heap tree are in last level

Heap tree

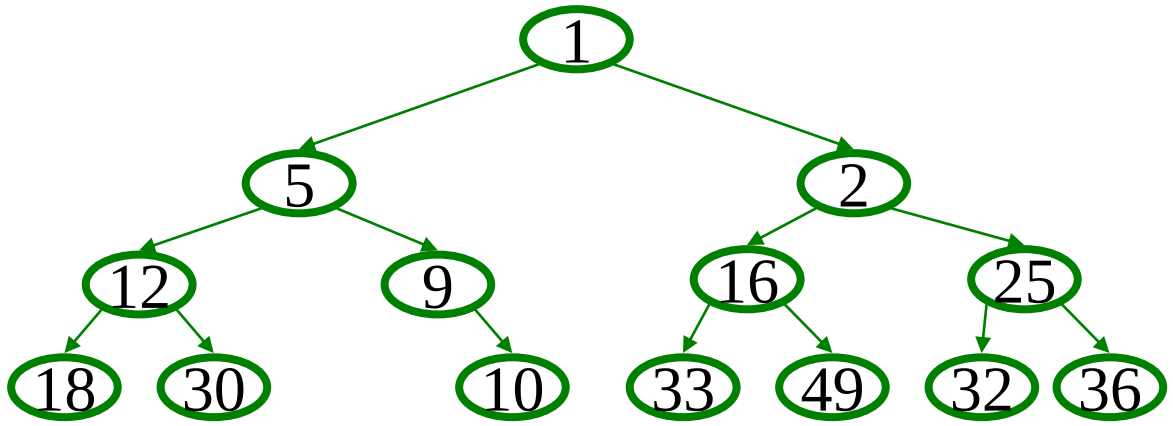
This is also a ***complete*** binary tree.



Not a heap tree

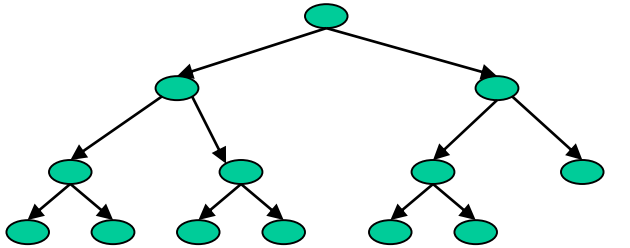
This is not a binary heap

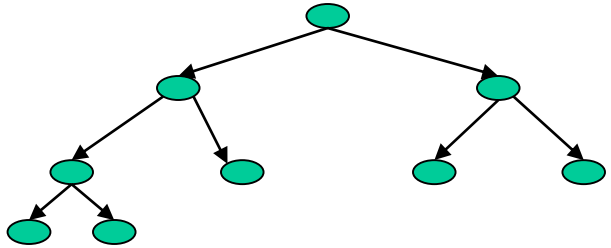
As it is not a ***complete*** binary tree.



Complete binary tree

Binary tree that is completely filled, with the possible exception of the bottom level, which is filled left to right.





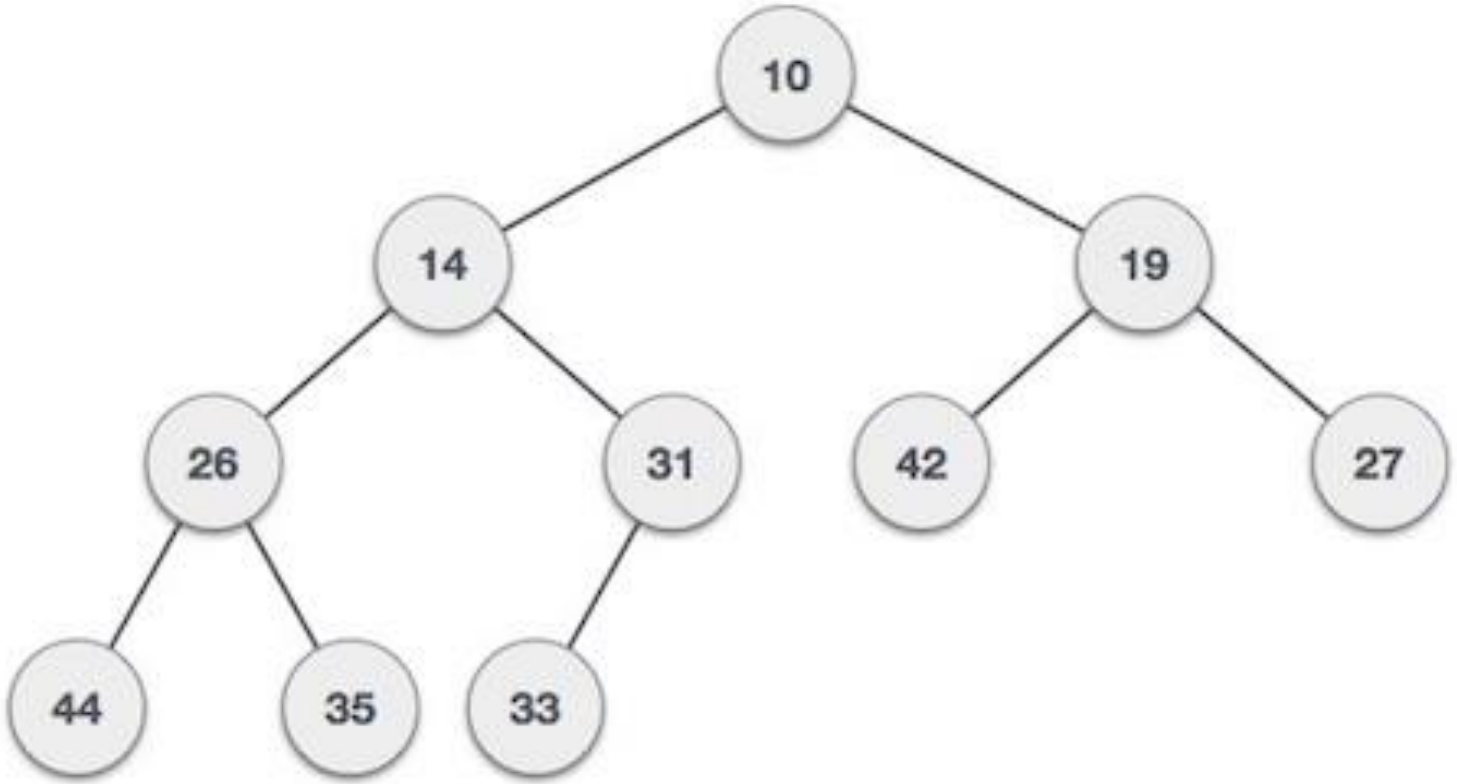
2. Ordering property

- If it is a Min Heap tree

- Each node is smaller than its children

- Smallest element will be located at the root of the heap tree

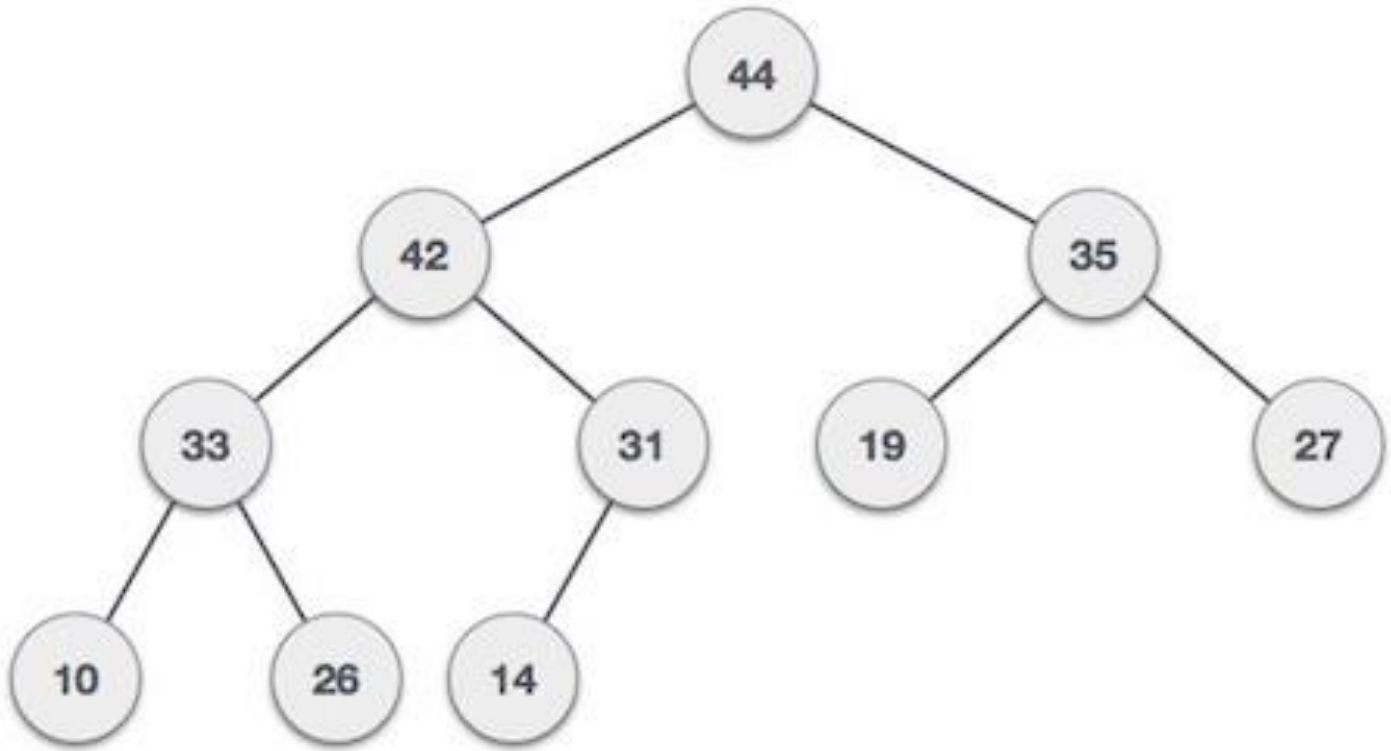
Min Heap



2. Ordering property MAX-Heap

- Each node is greater than its children
- Largest element will be located at the root of the heap tree

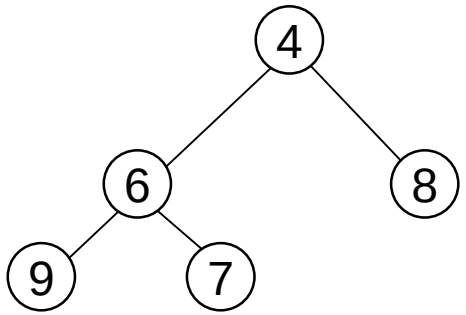
Max Heap



The Heap Data Structure

- *Def:* A **heap** is a nearly complete binary tree with the following two properties:
  - **Structural property:** all levels are full, except possibly the last one, which is filled from left to right
  - **Order (heap) property:** for any node x

    Parent(x) <= x



Heap

From the heap property, it follows that:
"The root is the smallest element of the heap!"

A heap is a binary tree that is filled in order

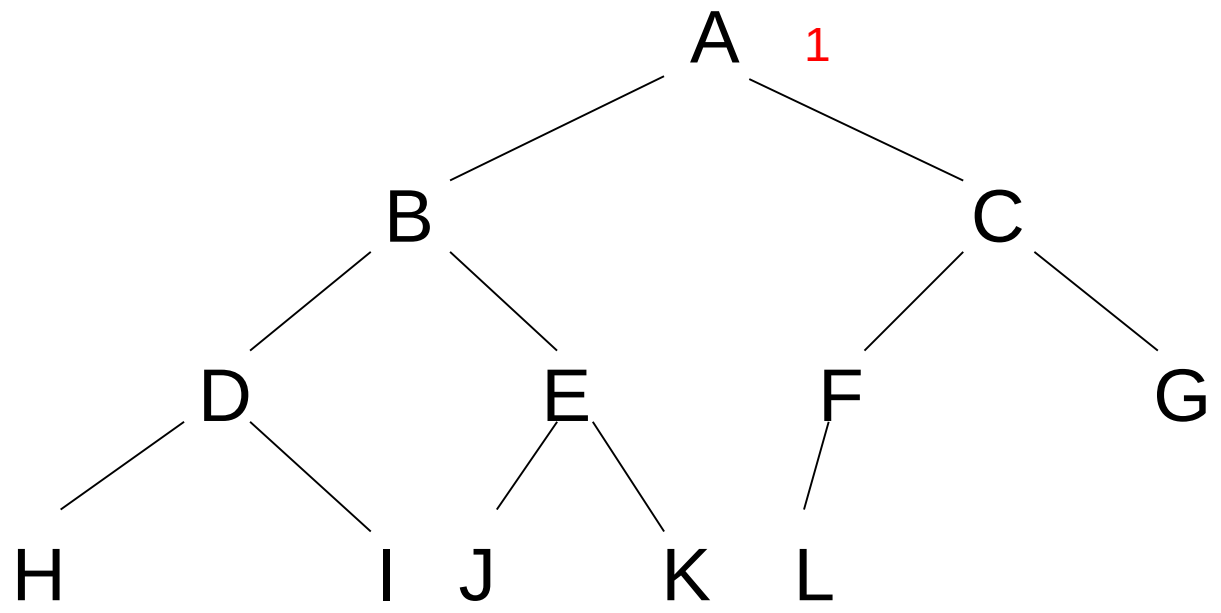
Implementing Binary Heap with an Array

1. It is difficult to store an ordinary binary tree using an array, as there could be many missing child nodes (holes)

2. However, there are no holes in the HEAP TREE, and so it can be stored compactly using an array structure

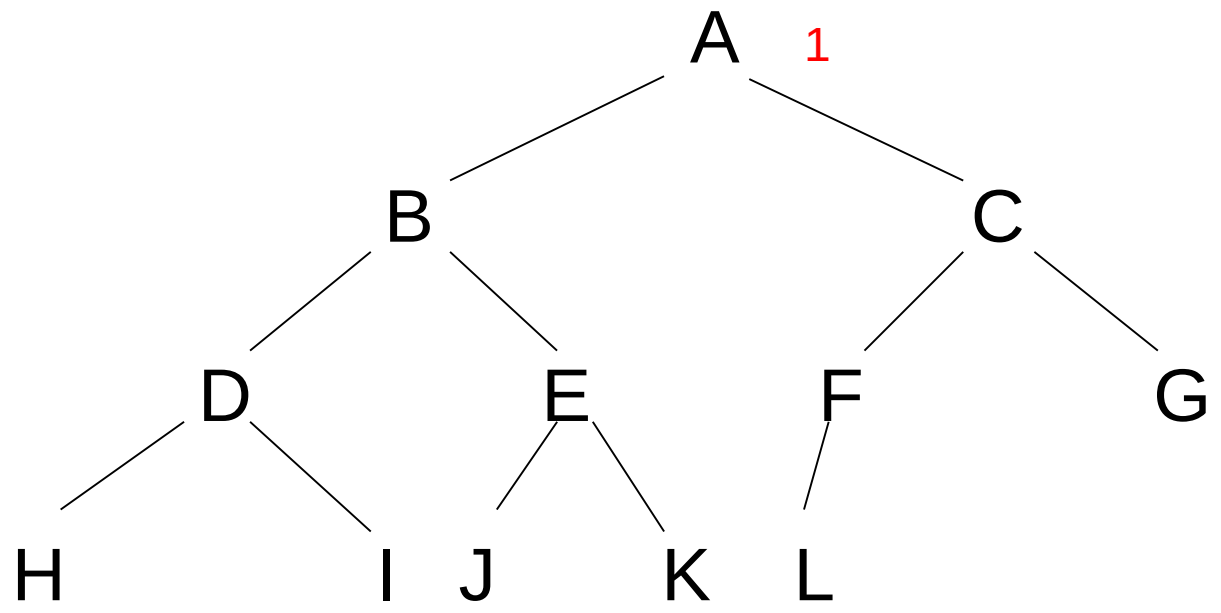
3. The first element (root) can be stored in array position 1, followed by other elements , stored level by level

Representing Heap tree as an Array



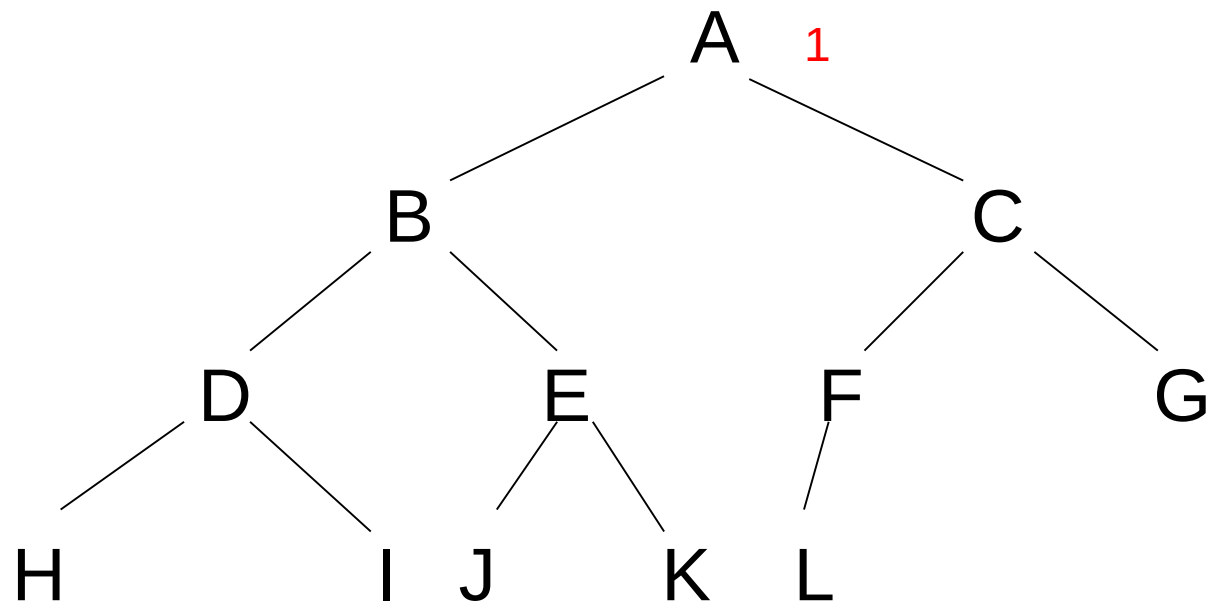
	A	B	C	D	E	F	G	H	I	J	K	L	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Representing Heap tree as an Array



	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

Locating children and Parent



For node: **i**

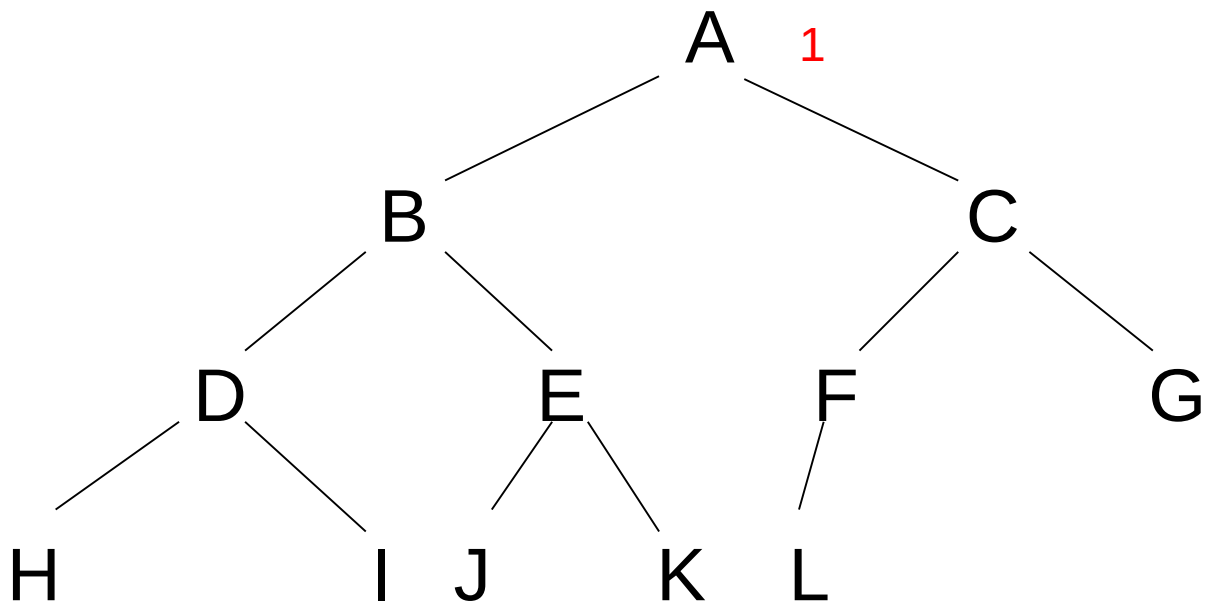
left child: **2i**

right child: **2i+1**

parent: **i/2**

	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

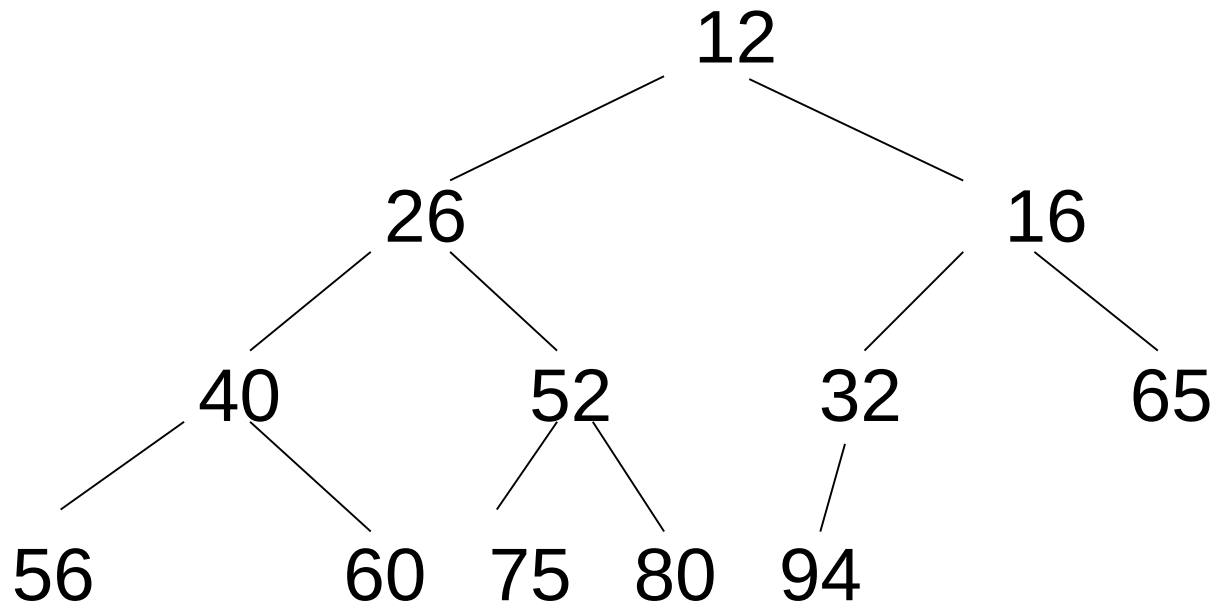
Locating children and Parent



Parent of K:
11/2 = 5
left child of E
= 2i = 10
right child of C
= 2i+1 = 7

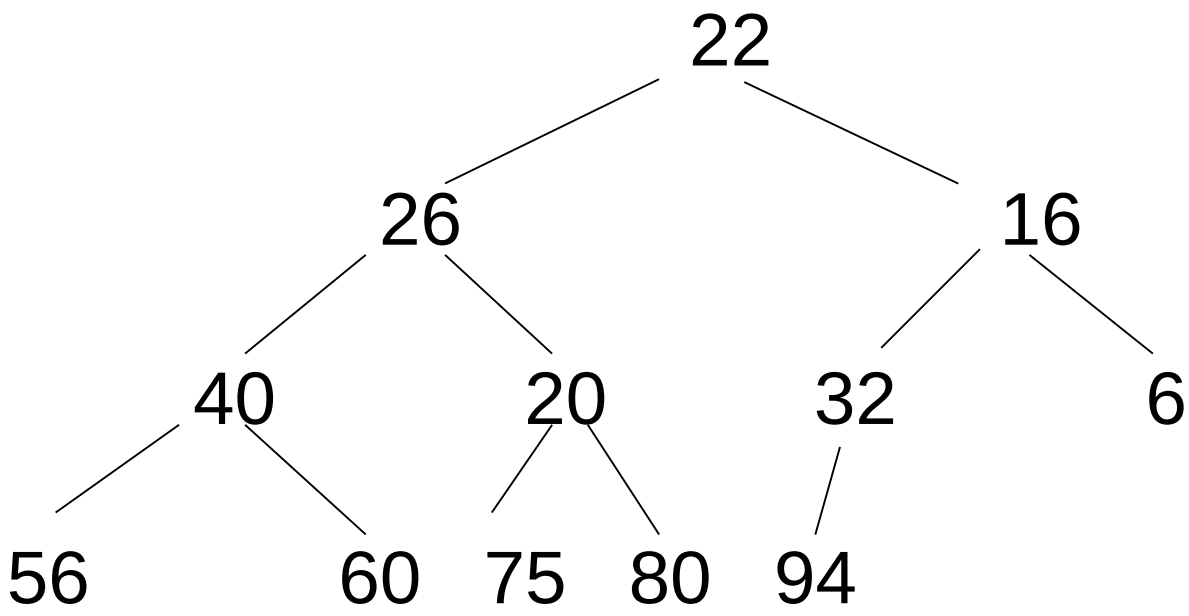
	A	B	C	D	E	F	G	H	I	J	K	L	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Satisfies Heap property (minHeap)



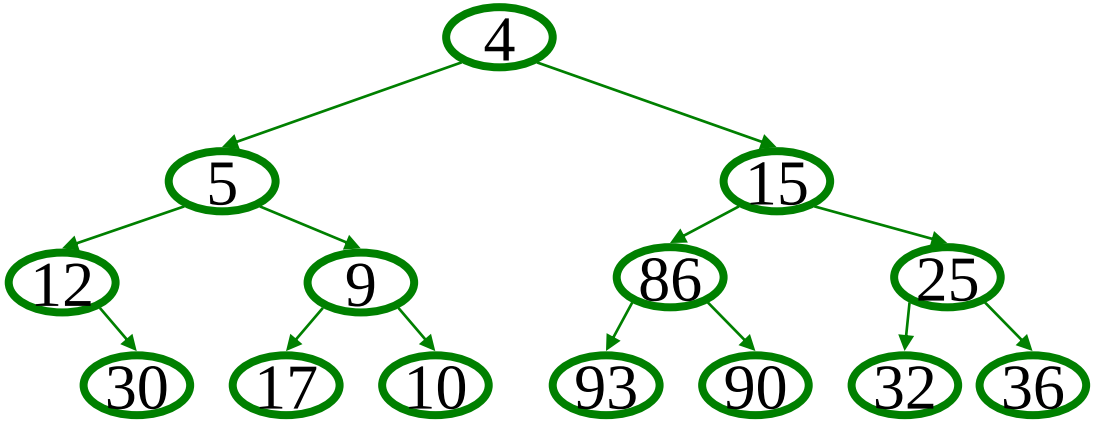
Node value less than values of its children

Does not satisfy Heap property



Is this a Min Heap?

1. Check if it is a complete tree
2. check if parent value smaller than siblings



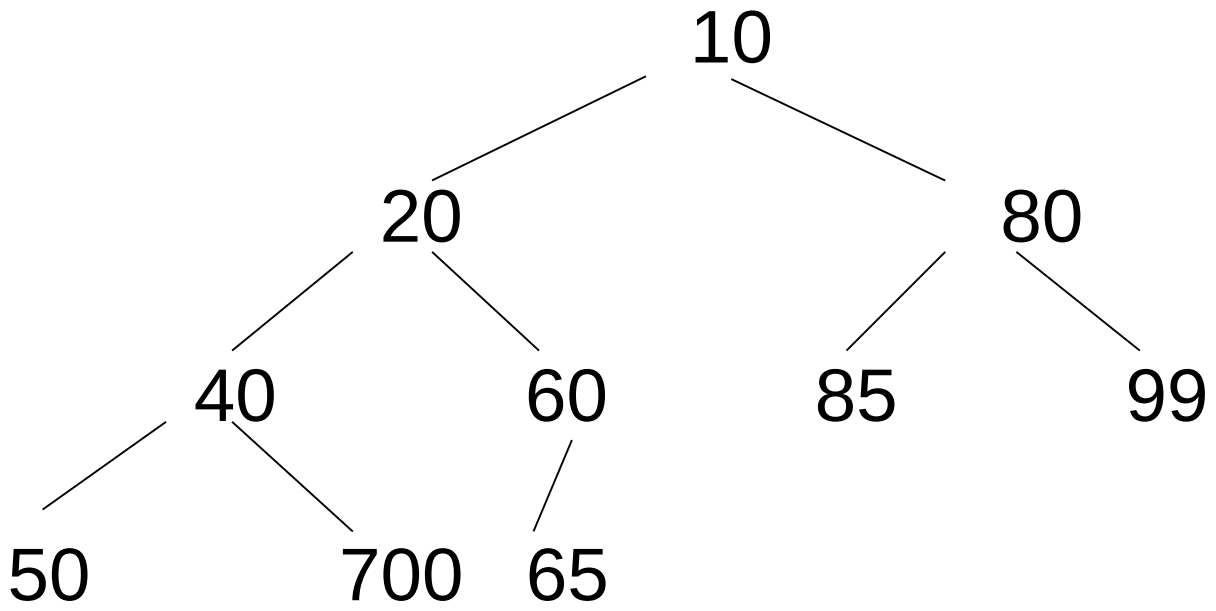
minHeap Tree ADT

- 1. Given n elements, create a *min* Heap
- 2. Insert a new element in Heap tree
- 3. Delete smallest element

- We start with insertion and deletion
- Then we shall take up creation

INSERTION IN HEAP TREE

insert 15



	10	**20**	**80**	**40**	**60**	**85**	**99**	**50**	**700**	**65**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Heap – Insert

1. The tree is full, (No holes) so there will not be any place for it on the tree.

2. Put *val* at “next” available position, that is at the end of the array

3. It may violate the heap property

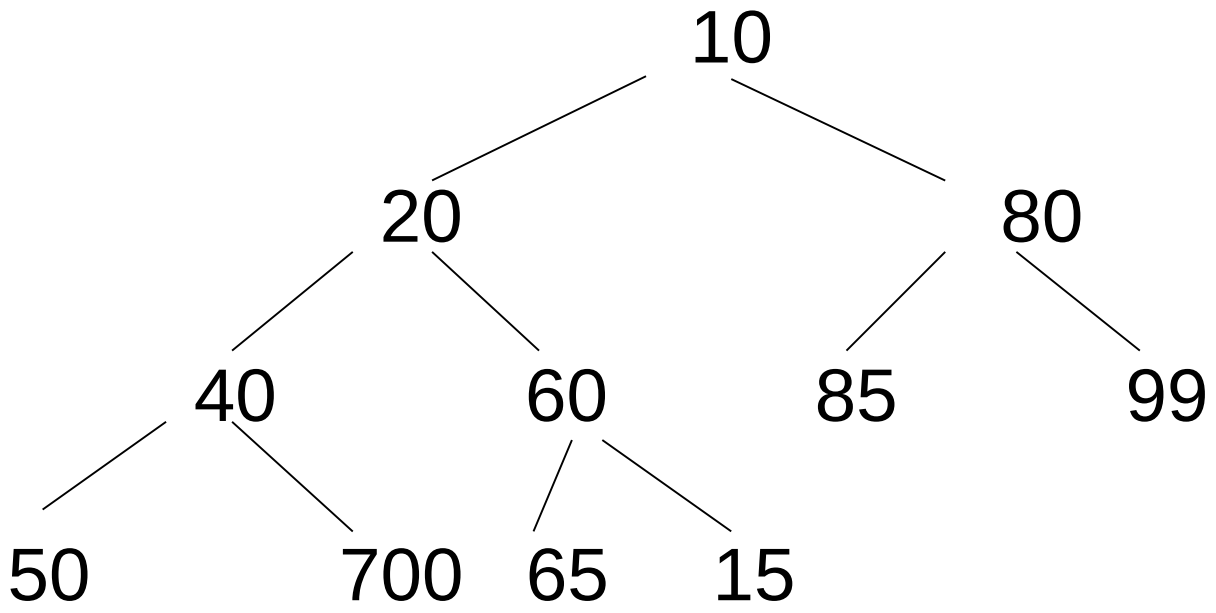
4. Percolate the node up to its correct level

There is no holes in the heap tree to insert 15.

So insert it after the last element
(in position 11).

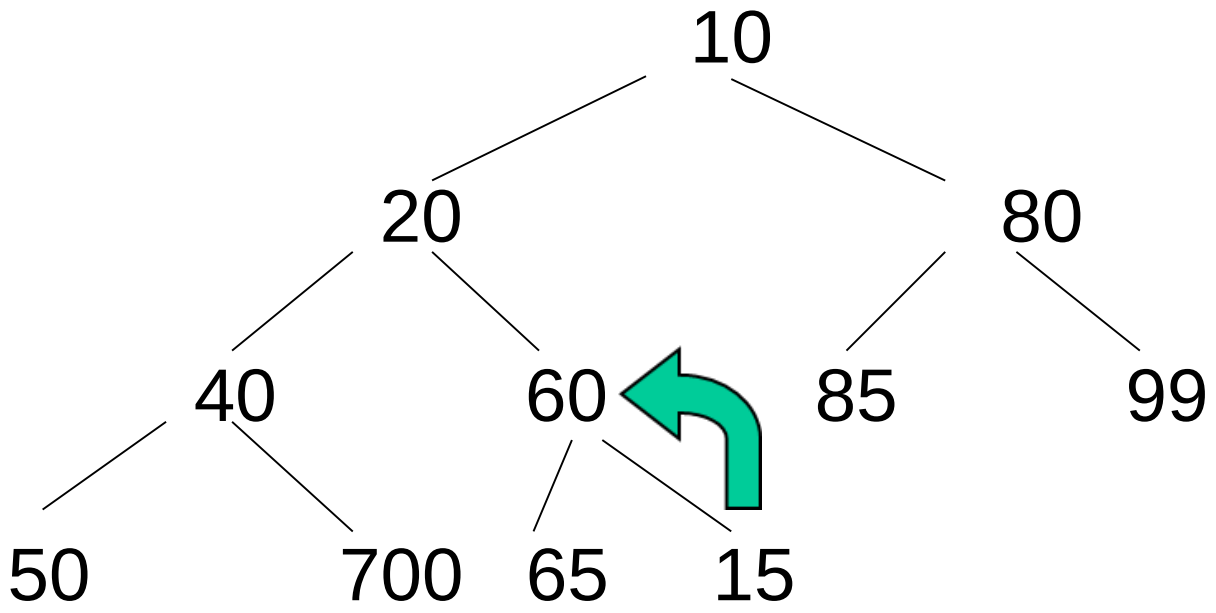
	10	20	80	40	60	85	99	50	700	65	15		
0	1	2	3	4	5	6	7	8	9	10	11	12	13

15 inserted, but it violates heap property



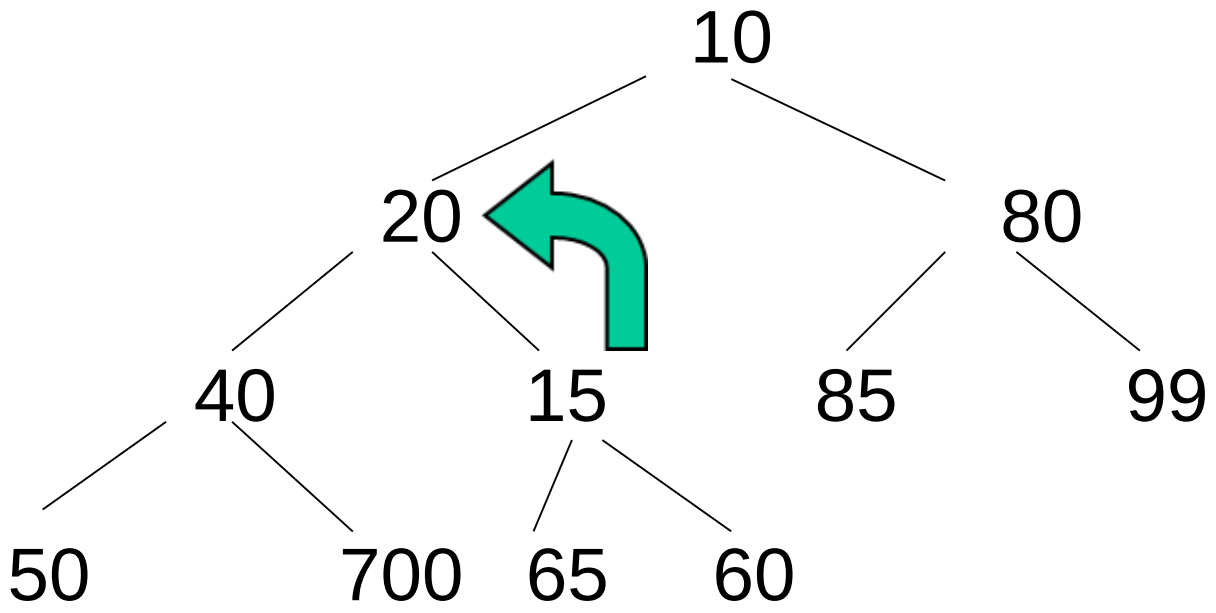
	10	20	80	40	60	85	99	50	700	65	15		
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Percolate15 up to its parent 60

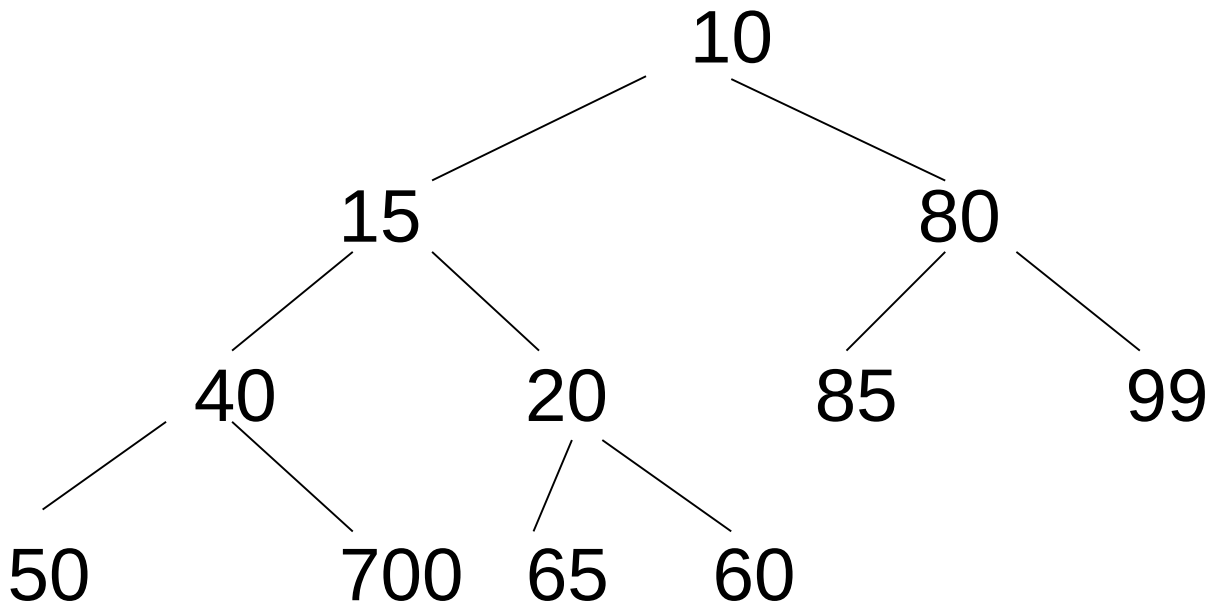


	10	**20**	**80**	**40**	**60**	**85**	**99**	**50**	**700**	**65**	**15**		
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Percolate15 up to its new parent 20 now



Percolate15 up to its new parent 20 now

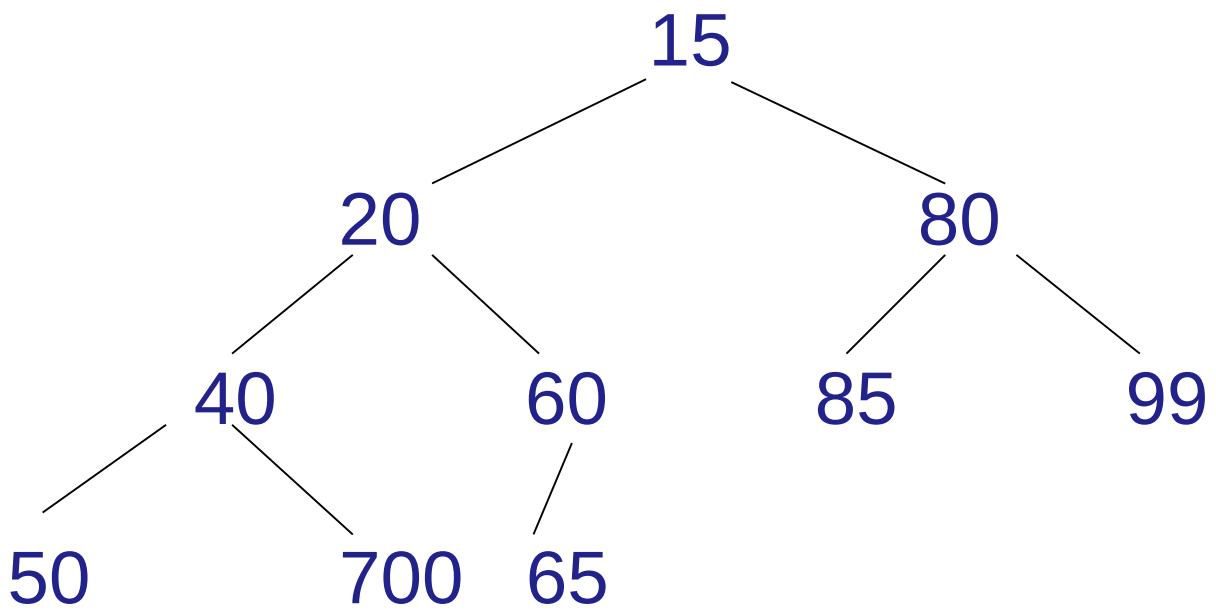


No more changes needed now, tree satisfies MinHeap property

To insert 15 on the original heap tree, only two changes were made to the Heap tree, which is even less than O(log n).

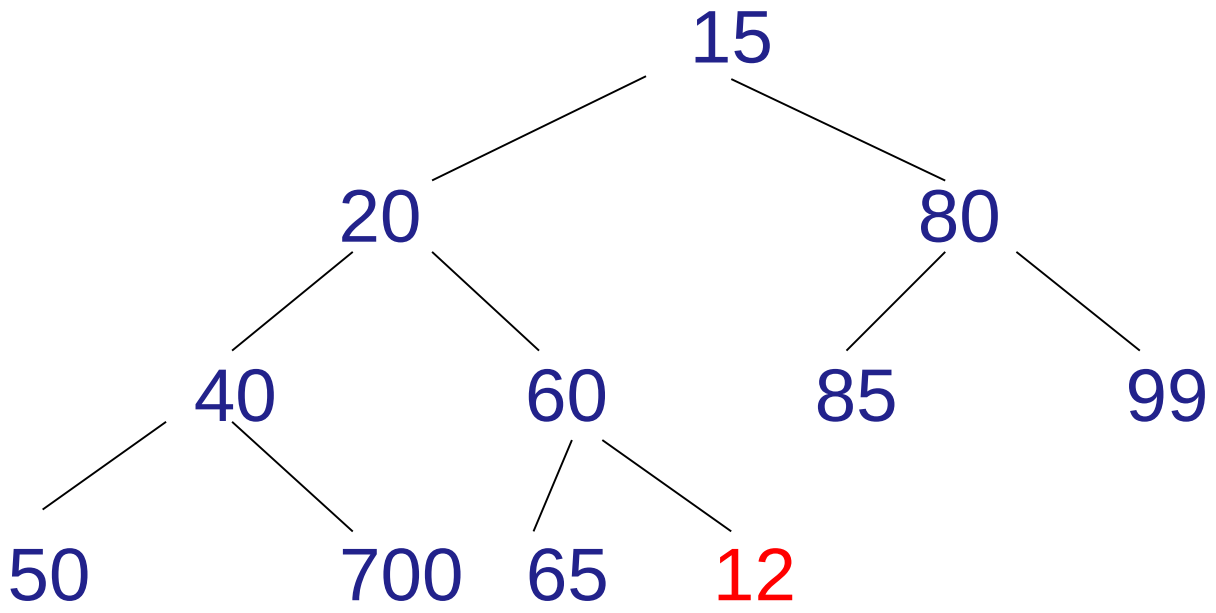
	10	15	80	40	20	85	99	50	700	65	60		
0	1	2	3	4	5	6	7	8	9	10	11	12	13

insert 12 on this tree



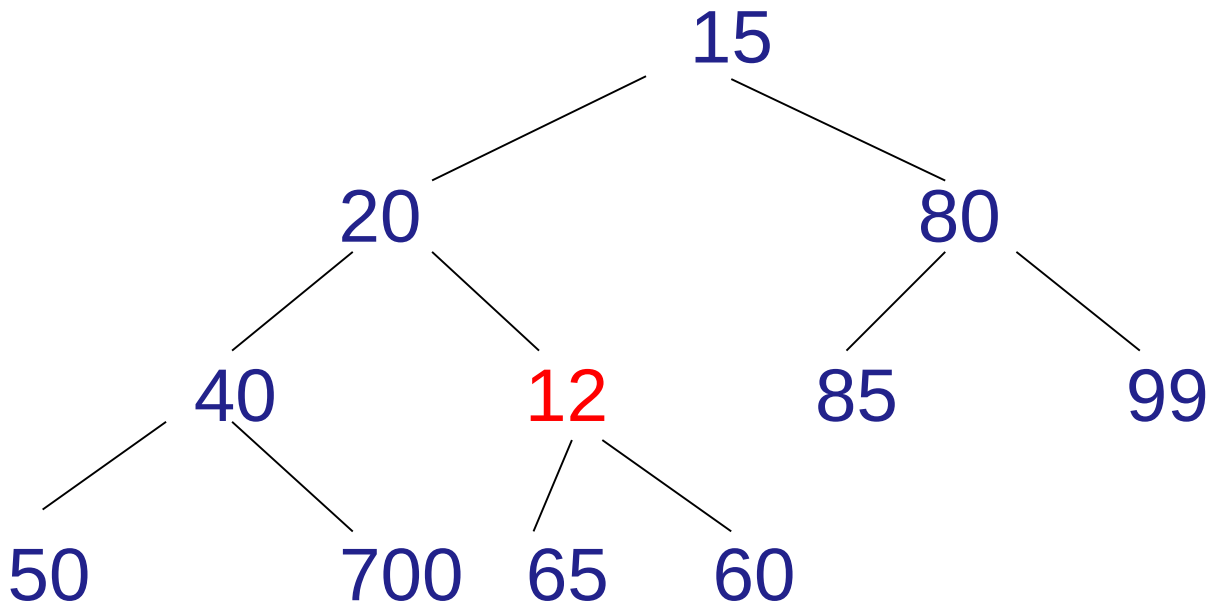
No more changes needed now, tree satisfies MinHeap property

12 inserted, violates heap property



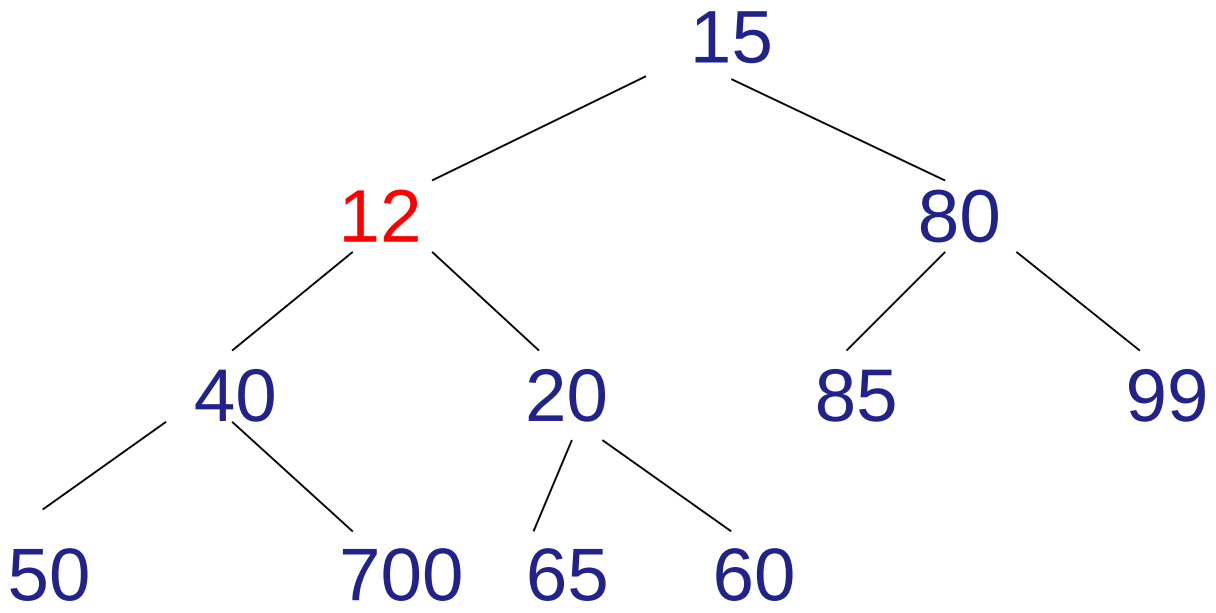
Percolate 12 up the tree

12 moved up, still violates heap property

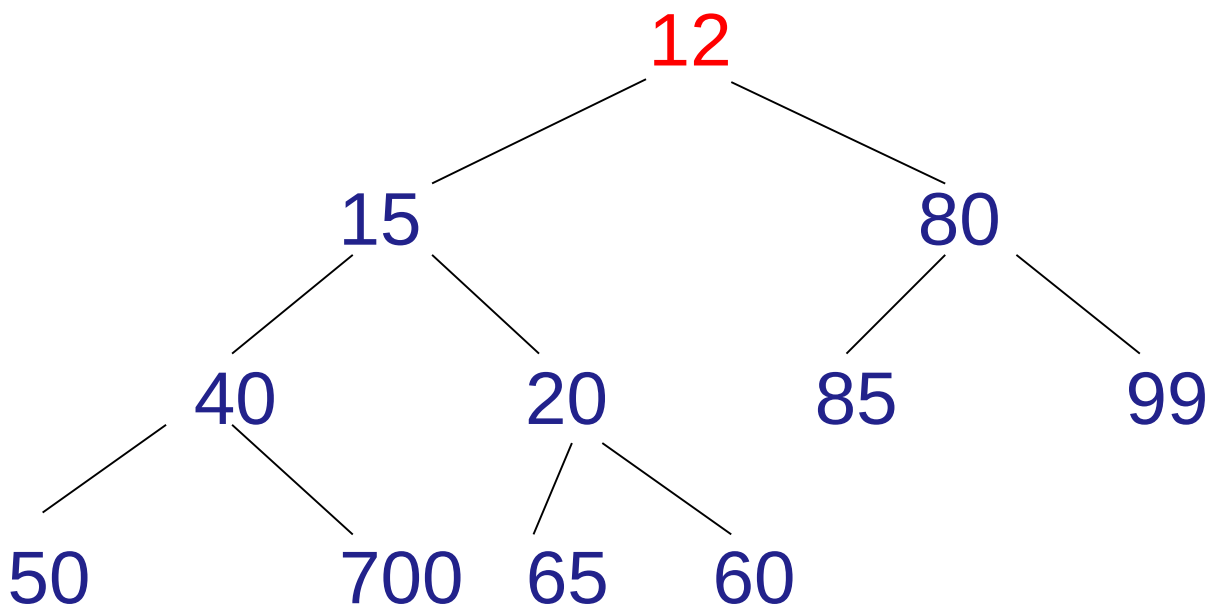


Percolate 12 up the tree

12 moved up, still violates heap property



Still there is need to Percolate 12 up the tree



Insertion complete, tree satisfies heap property.
3 changes made, complexity O(log n)

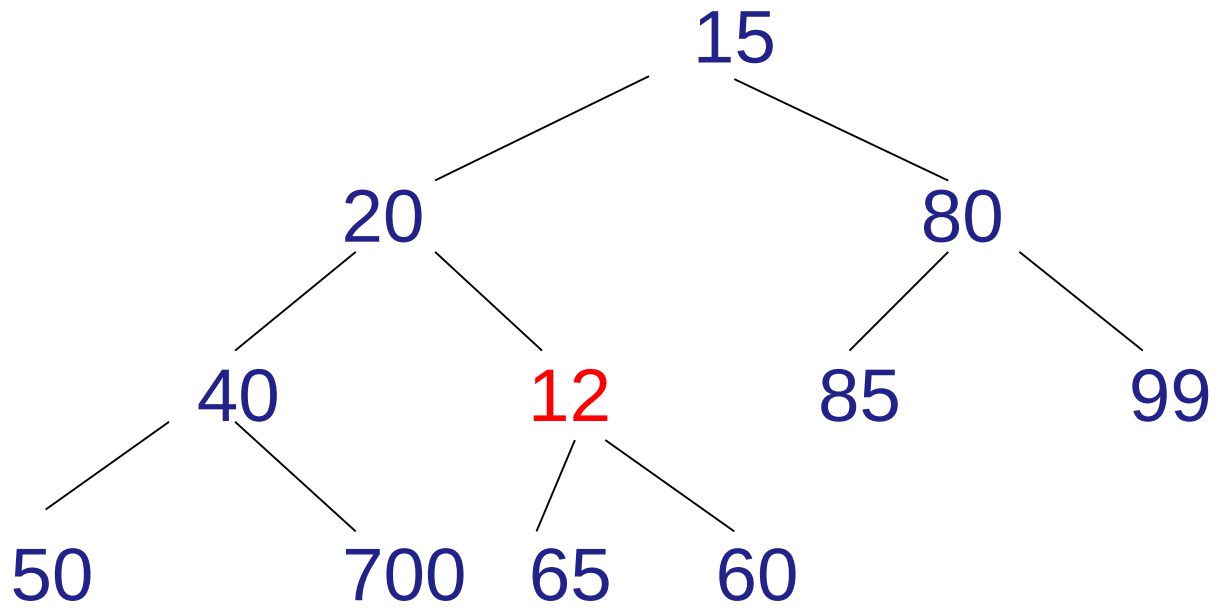
InsertHeap

Algorithm for insertion *val* on minHeap

```
Make a new hole (with val) at size + 1;
 while(hole > 1 && val < parent value of hole)
    {
        Put parent value in hole.
        move hole to its parent.
    }
 Put val in last position of hole.
```

Insert *val* on minHeap

```
hole = size + 1;
Heap[hole] = val;
while (hole > 1 &&   val < Heap[hole/2])
 {
       Heap[hole] = Heap[hole/2];
       hole = hole/2;
 }
Heap[hole] = val;
```




```
hole = size + 1;          Heap[hole] = val;

Heap[hole] = Heap[hole/2];  hole = hole/2;
```

DELETION IN HEAP TREE

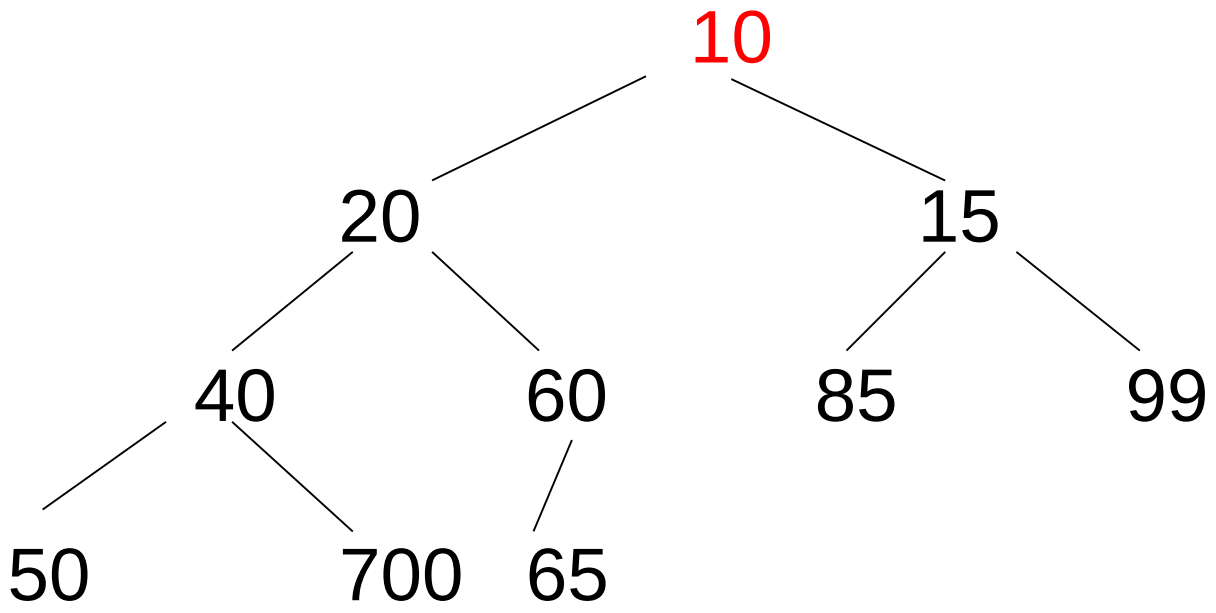
DELETE-MIN

Heap – Deletemin

Basic Idea:

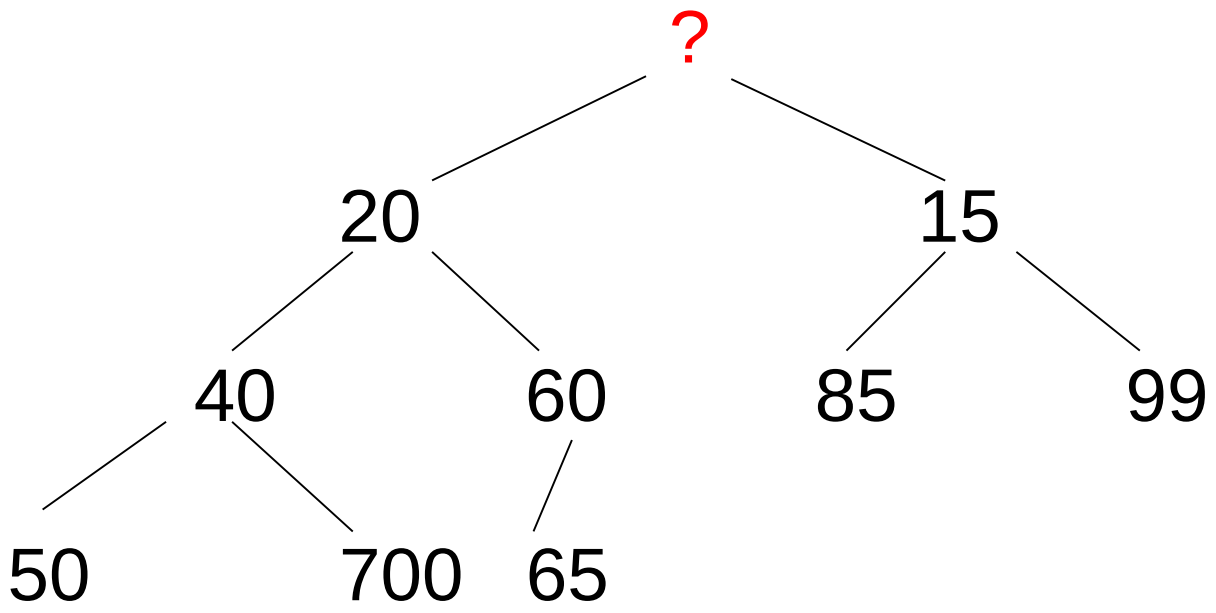
1. Remove root (that is always the min!)
2. Creates a hole
3. Put “last” leaf node at this hole
4. Compare its value with two children
5. If needed, Swap node with its smaller child
6. Repeat steps 3 & 4 until no swaps needed.

DeleteMin from this tree



	10	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**65**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Deletion creates a hole in the tree



	--	20	15	40	60	85	99	50	700	65			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

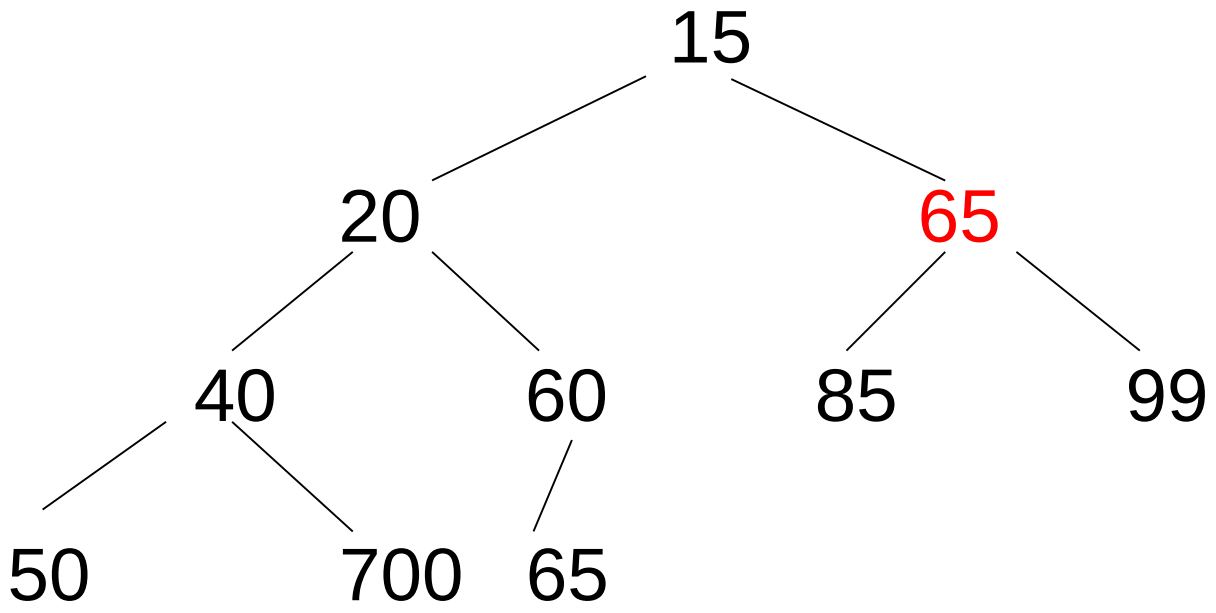
After deletion, there will be 9 elements left

Position 1 can not be left blank.

Shift last element 65 to position 1 (root)

	65	**20**	**15**	**40**	**60**	**85**	**99**	**50**	**700**	**--**			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Exchange 65 with smaller child

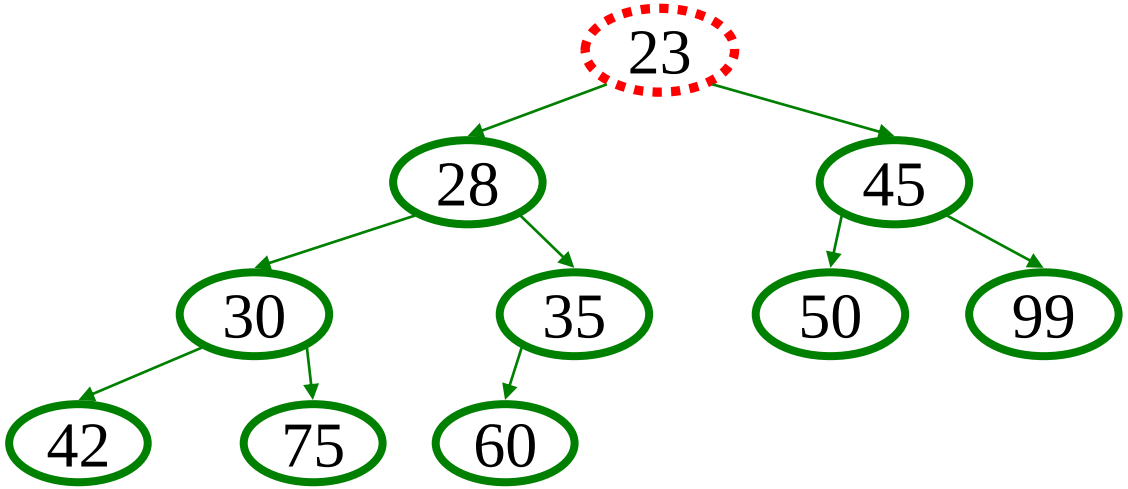


	15	**20**	**65**	**40**	**60**	**85**	**99**	**50**	**700**				
0	1	2	3	4	5	6	7	8	9	10	11	12	13

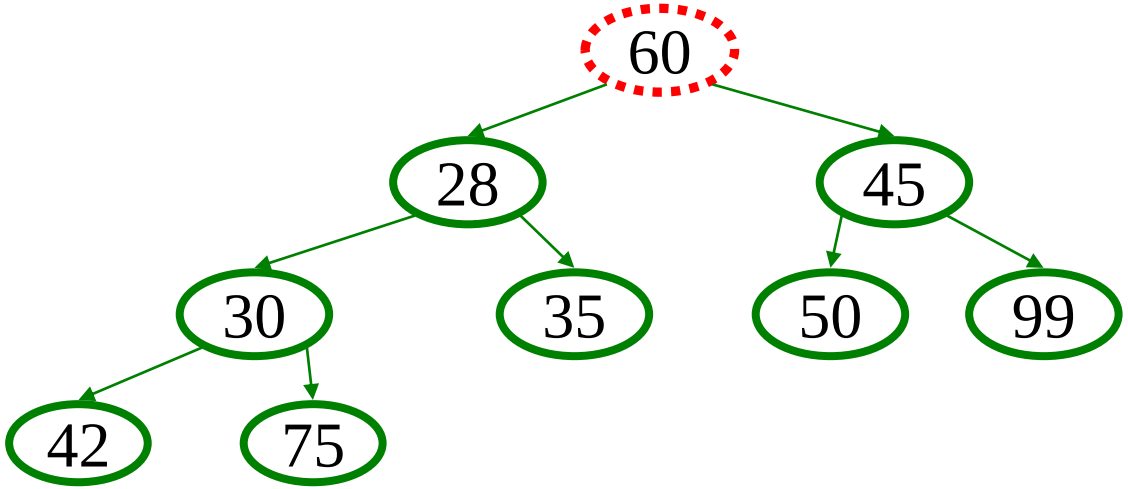
- 65 exchanged with 15.
  Heap order is maintained everywhere.
  No more adjustments needed.

  Complexity of Deletion: O(log n)

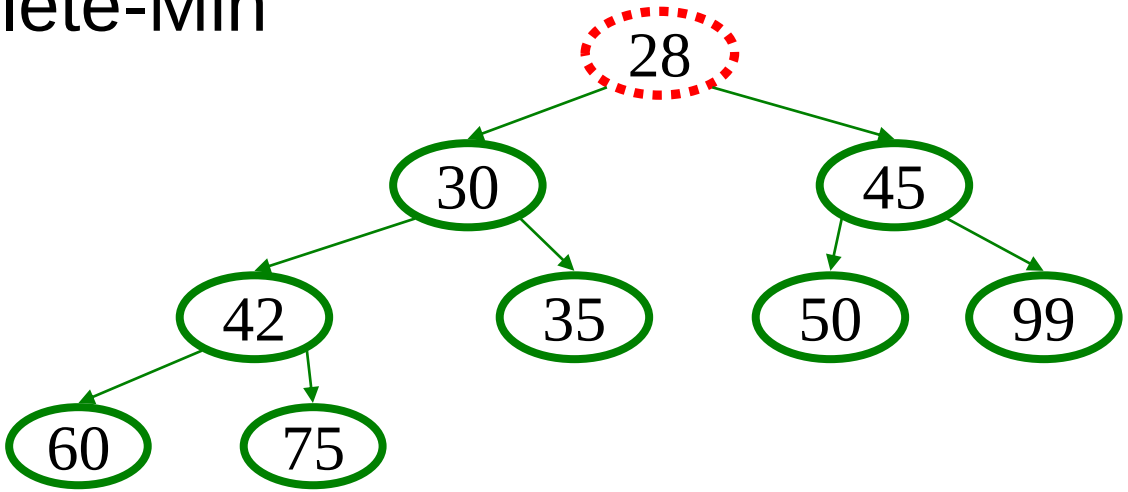
Ex 2: Delete from this heap tree, show working



Replace root with last element



Tree after delete-Min



Time for Deletion : Tree height: O(log n)

DeleteMin Code

```
First do size--
Get Heap[size+1]
Find position of this by Percolate Down

newP = percolateDown(1,Heap[size+1]);
Heap[newP] = Heap[size + 1];
```

DeleteMin

```
int percolateDown(int hole, int val) {
while (2*hole <= size) {       //locate values of children
    left = 2*hole;
    right = left + 1;
                        //find out which child will replace parent
    if (right ≤size && Heap[right]<Heap[left])
            target = right;
    else    target = left;

    if (Heap[target] < val) {
        Heap[hole] = Heap[target]; // store target in hole
        hole = target;          }
    else          break;
  }
  return hole;
}
```