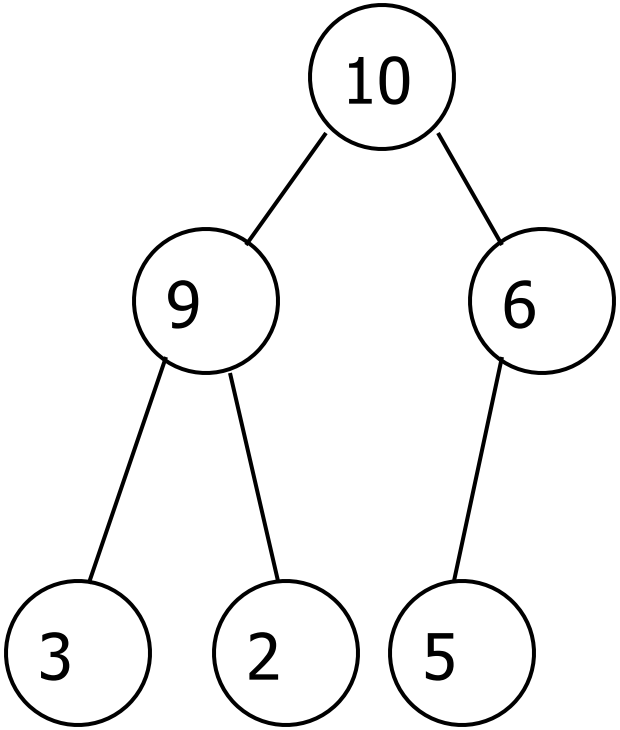
(Binary) Heaps

- Similar to a Binary Search Tree (BST)
- Heap is sorted in a weaker sense than BST
- Tree is a complete tree
  - Bottom level may not be filled but fills from left to right



(Binary) Heaps

- Binary trees are easily stored in an array
  - Array with index started at 0
    - Parent = (i-1)/2
    - Left Child = 2i + 1
    - Right Child = 2i + 2
  - Array with index started at 1
    - Parent = i/2
    - Left Child = 2i
    - Right Child = 2i + 1

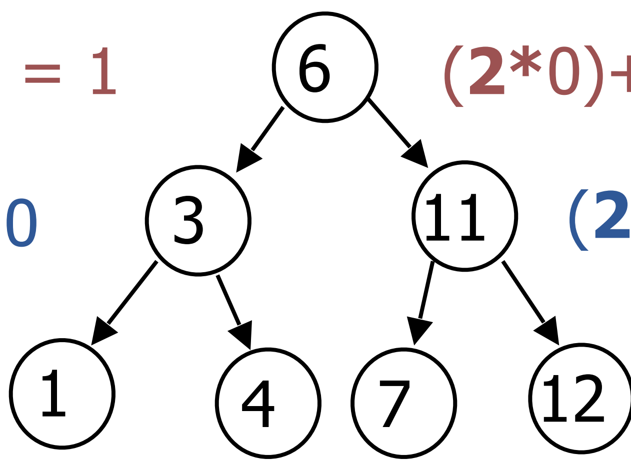
Binary Tree – Array

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
6	3	11	1	4	7	12	

$(2*\mathbf{0})+1 = 1$

$(\mathbf{2*}0)+2 = 2$

Find children



$(\mathbf{1}-1)/2 = 0$

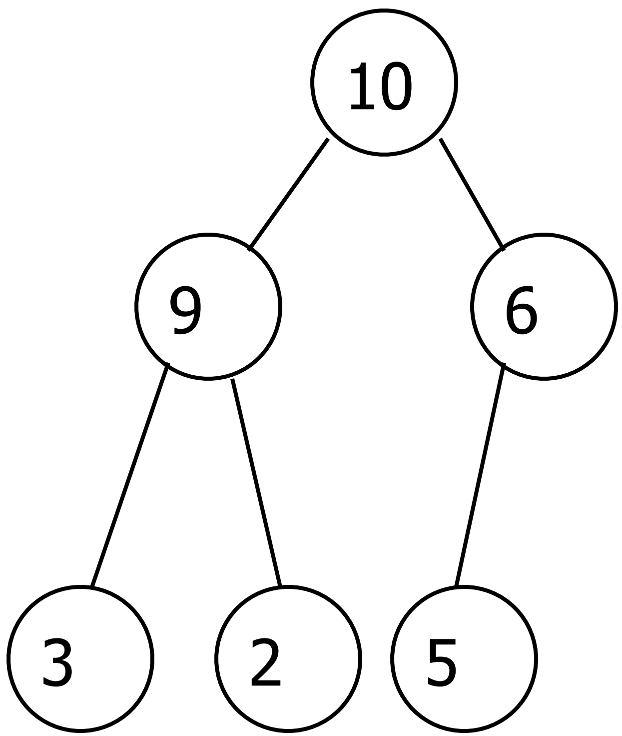
$(\mathbf{2}-1)/2 = 0$

Find parents

Heap Order Properties

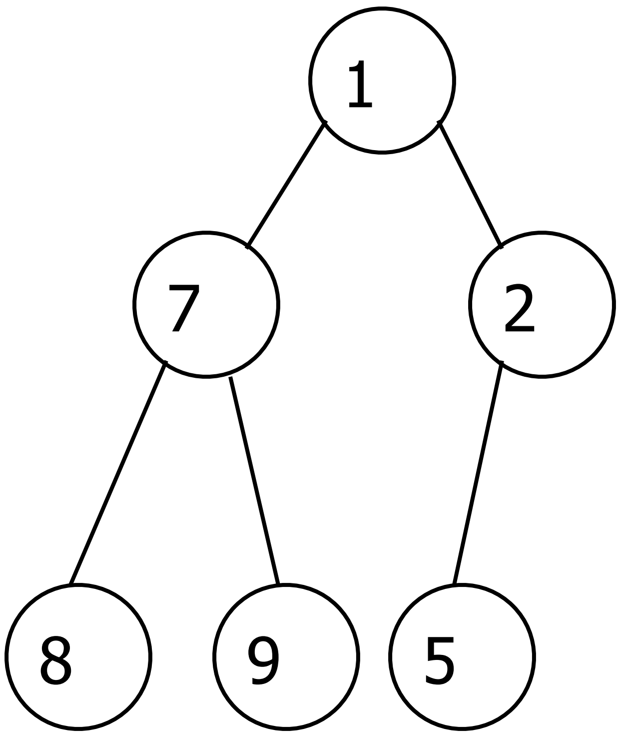
- Min Heap
  - For each node X, X.key > (X.parent).key
    - Smallest number is at the root
- Max Heap
  - For each node X, X.key < (X.parent).key
    - Largest number is at the root

Max Heap



10
9
6
3
2
5

Min Heap



1
7
2
8
9
5

Heap - Insert

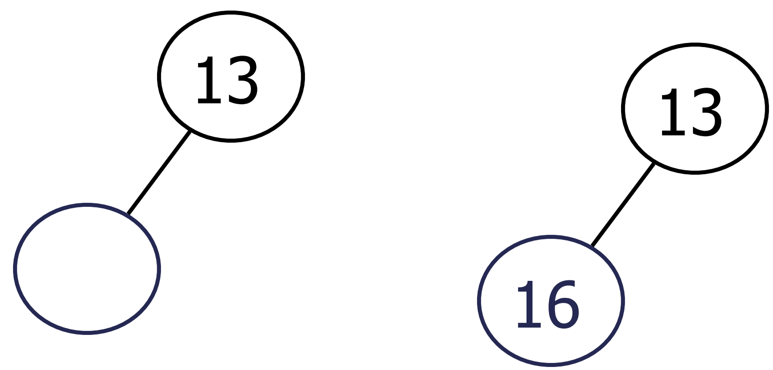
- Create a hole (empty node) in the next available complete tree location
  - Remember to fill bottom layer from left to right
- If the item can be inserted into the hole without violation of the heap property, insert item
- Otherwise, copy the hole's parent item into the hole then trickle up

Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

Insert 13

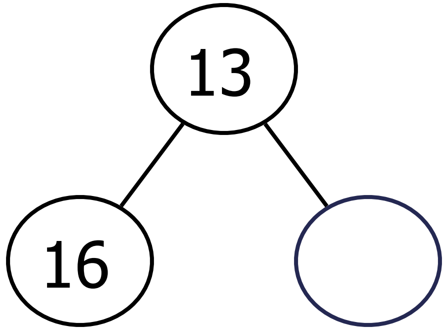
Insert 16

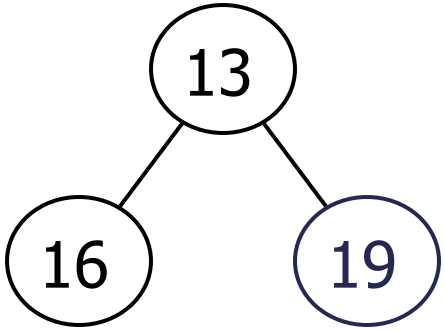


Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

Insert 19

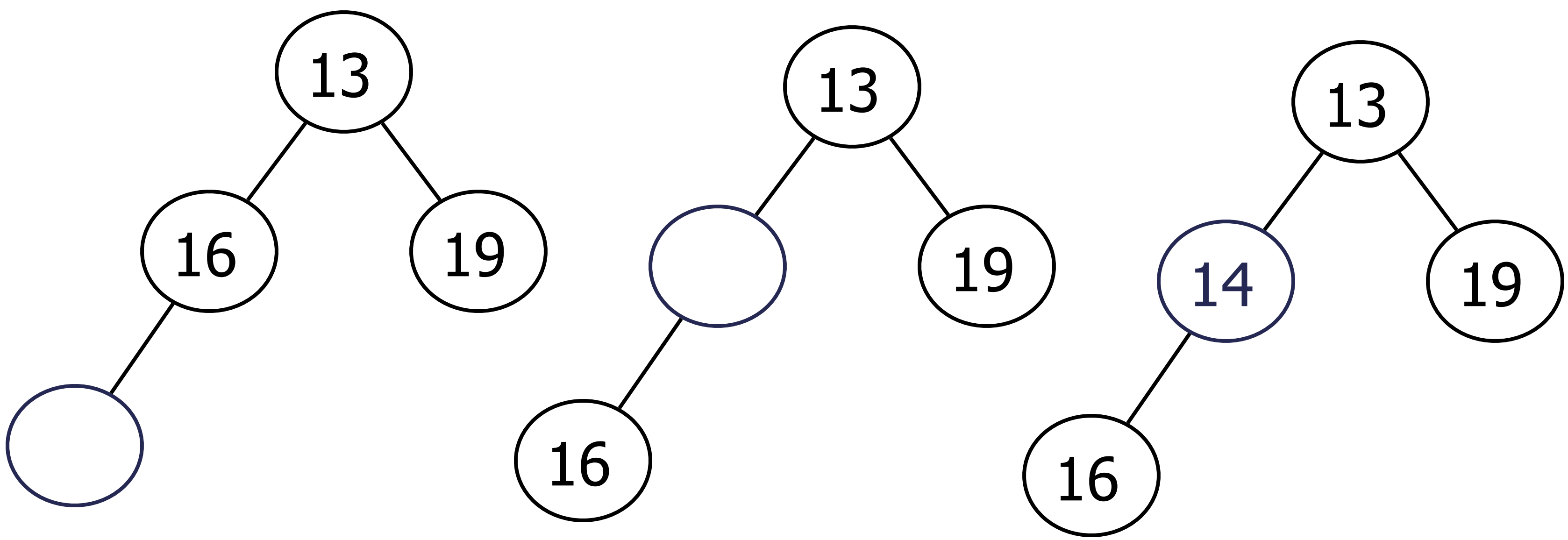




Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

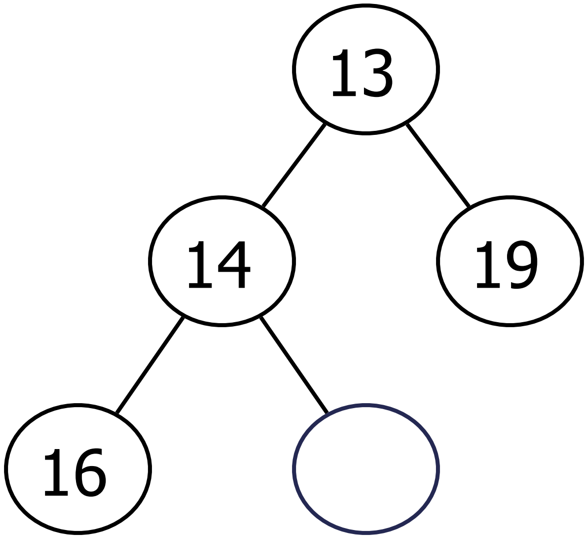
Insert 14

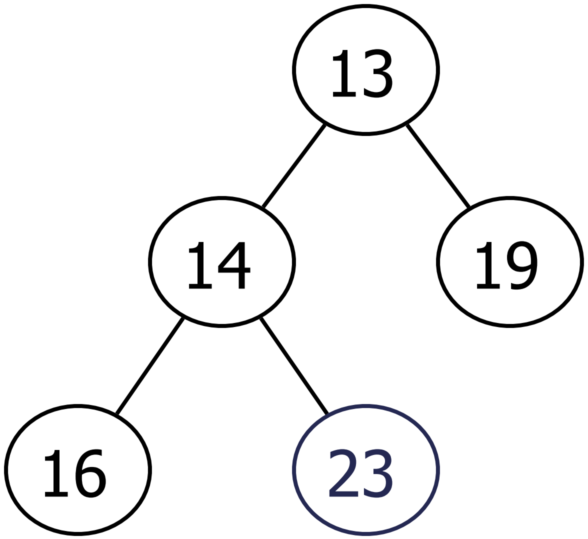


Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

Insert 23

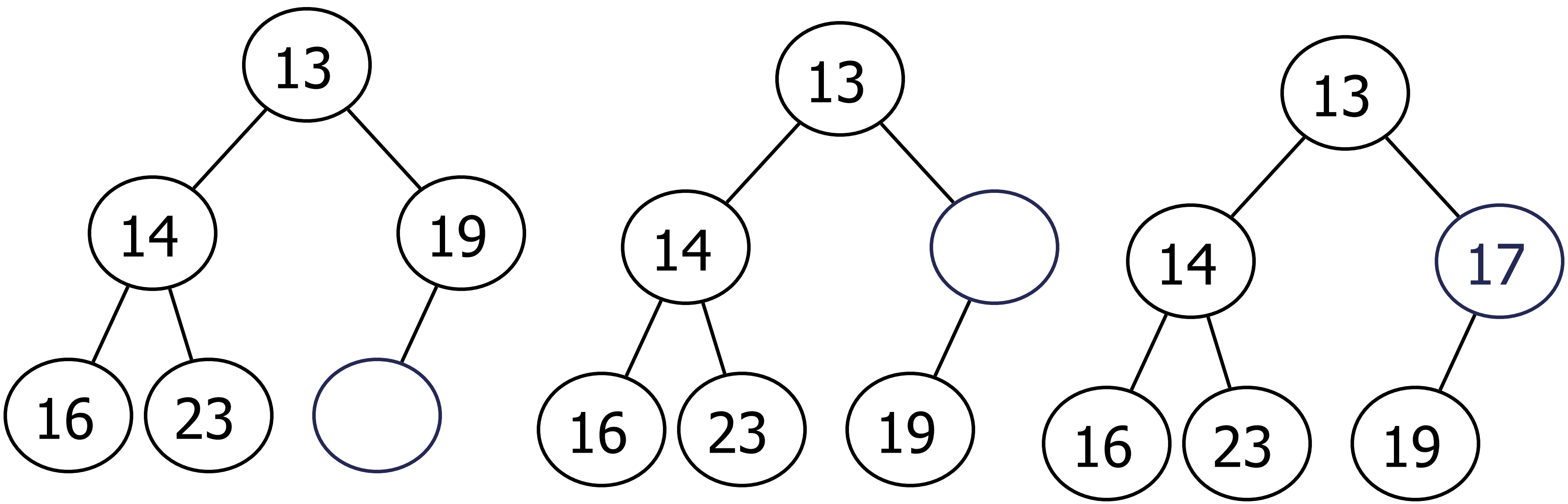




Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

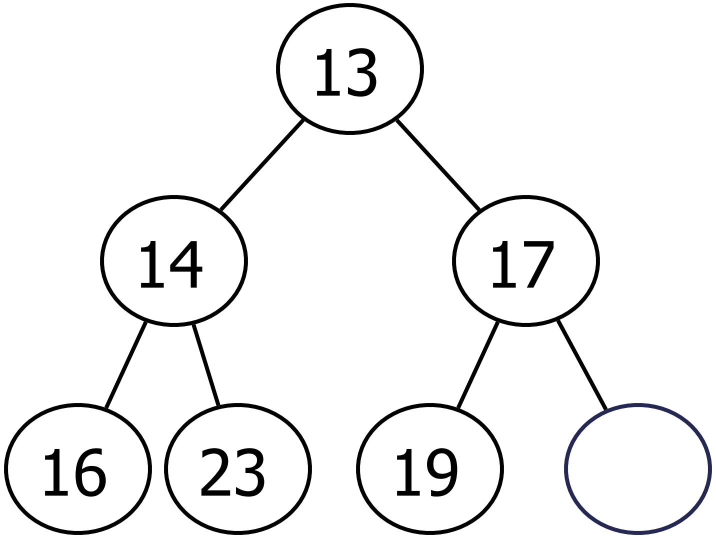
Insert 17

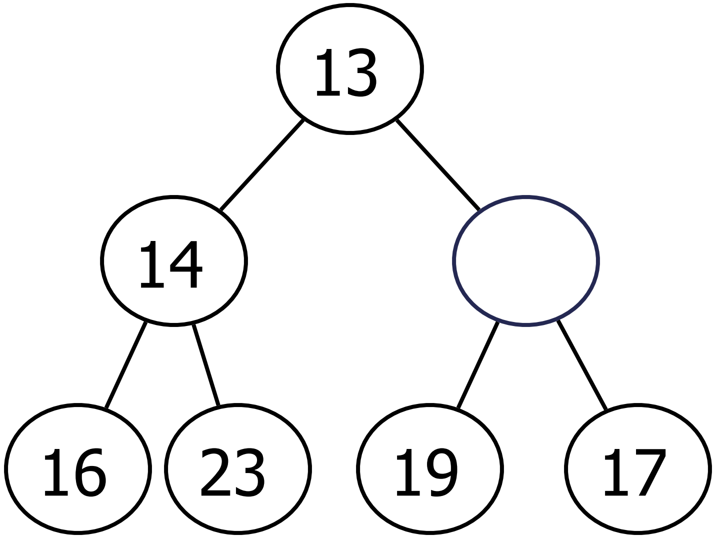


Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

Insert 6

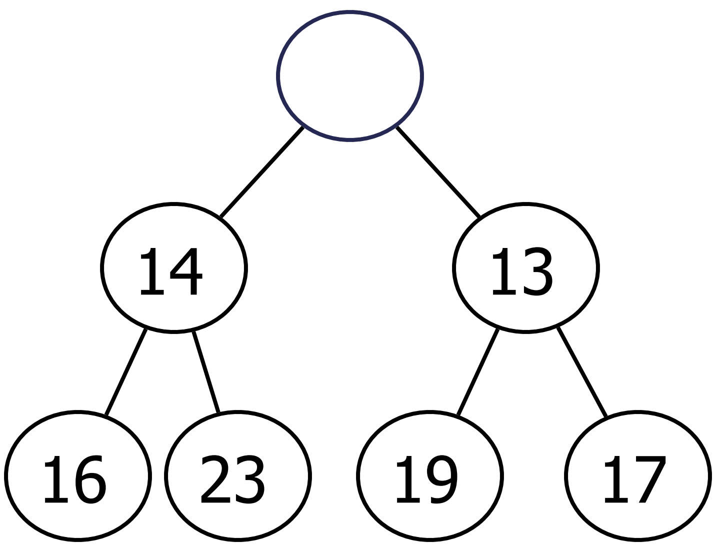


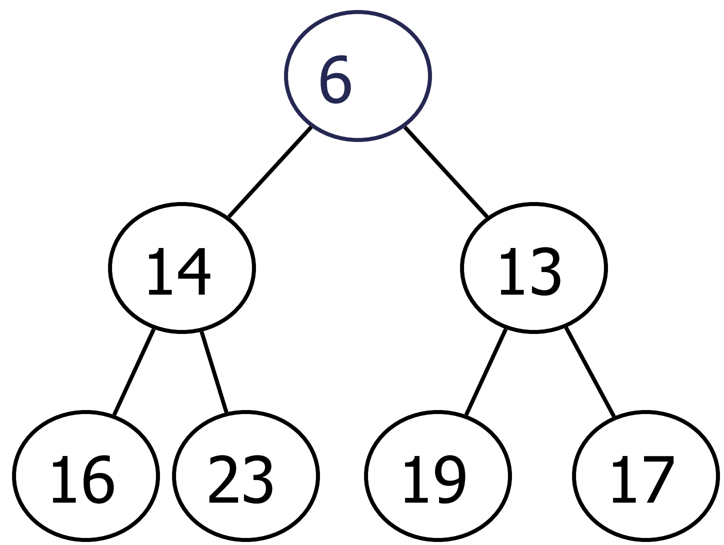


Min Heap - Insert

13, 16, 19, 14, 23, 17, 6

Insert 6

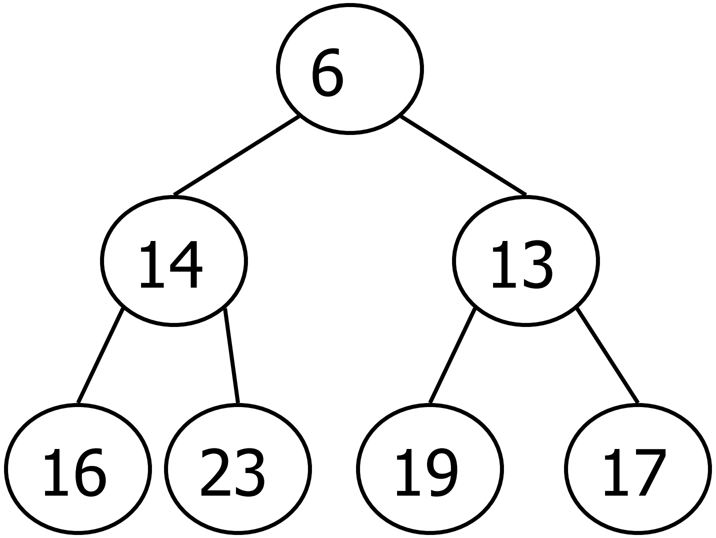




Min Heap – Extract Min

6, 14, 13, 16, 23, 19, 17

6,14,13,16 23,19,17
13,14,17,16,23,19
14,16,17,19,23
16,19,17,23
17,19,23
19,23
23

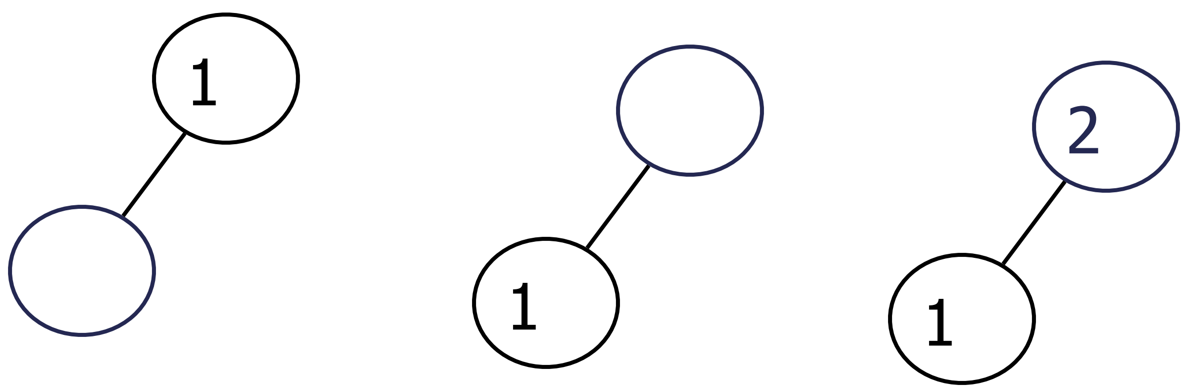


Max Heap - Insert

1, 2, 3, 4, 5, 6

Insert 1

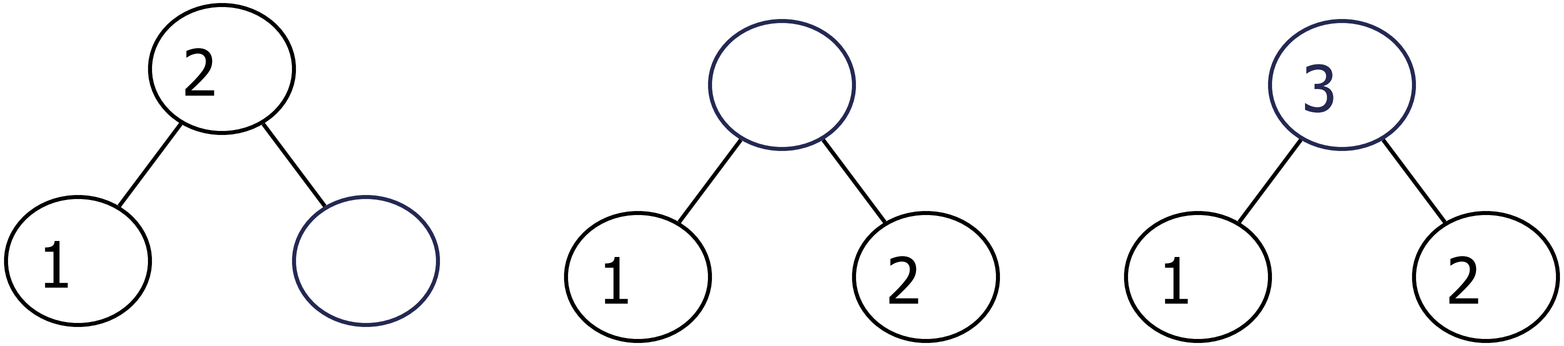
Insert 2



Max Heap - Insert

1, 2, 3, 4, 5, 6

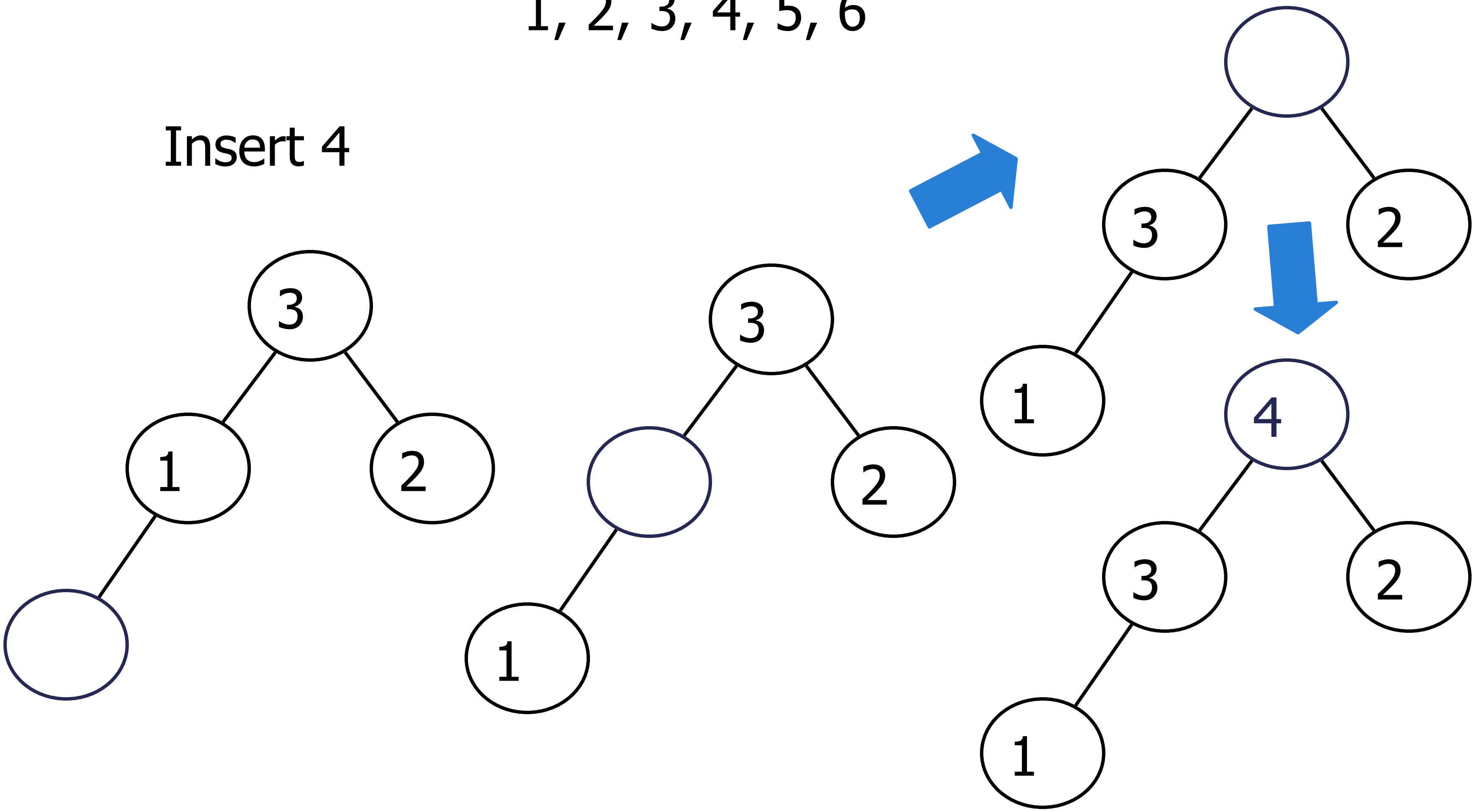
Insert 3



Max Heap - Insert

1, 2, 3, 4, 5, 6

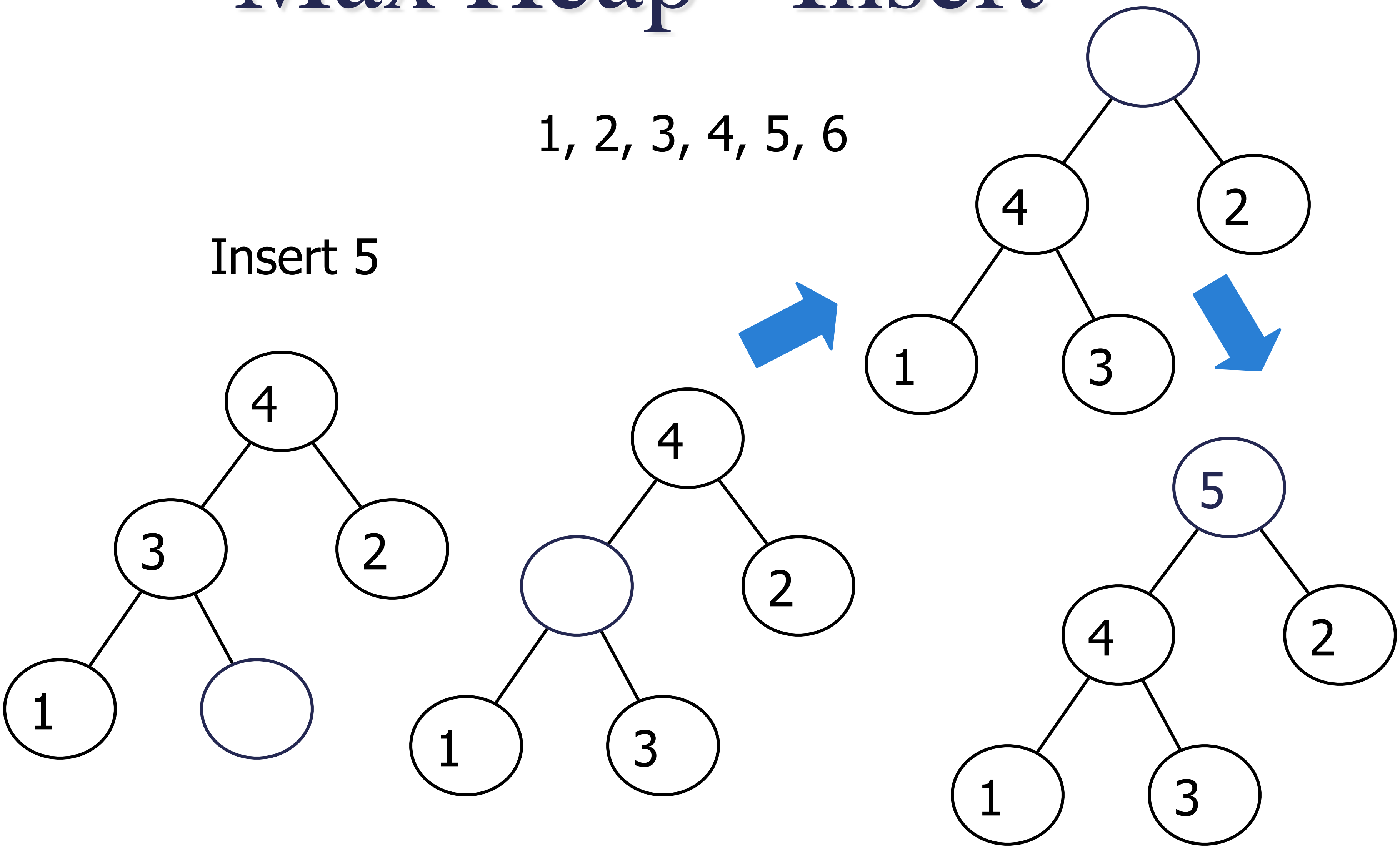
Insert 4



Max Heap - Insert

1, 2, 3, 4, 5, 6

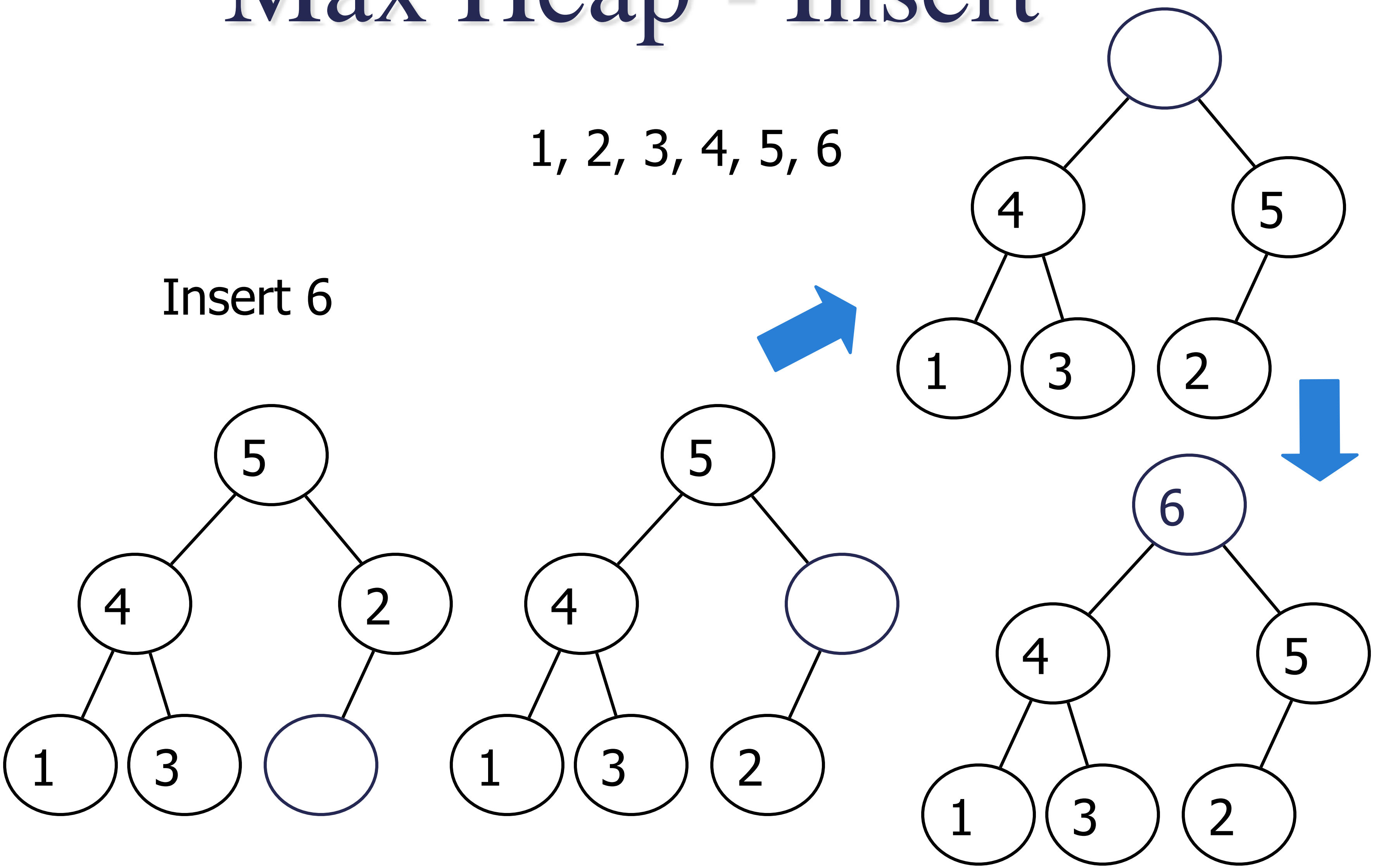
Insert 5



Max Heap - Insert

1, 2, 3, 4, 5, 6

Insert 6



Min Heap Insert

Running time = O(logn)

```
void Min-Heap-Insert ( itemtype item )
        if  heap full
            throw heap full exception
        // Trickle up
        x = numNodes
        while (x > 0 && arr[(x-1)/2] > item)
            arr[x] = arr[(x-1)/2]
            x = (x-1)/2
        arr[x] = item
        numNodes++
```

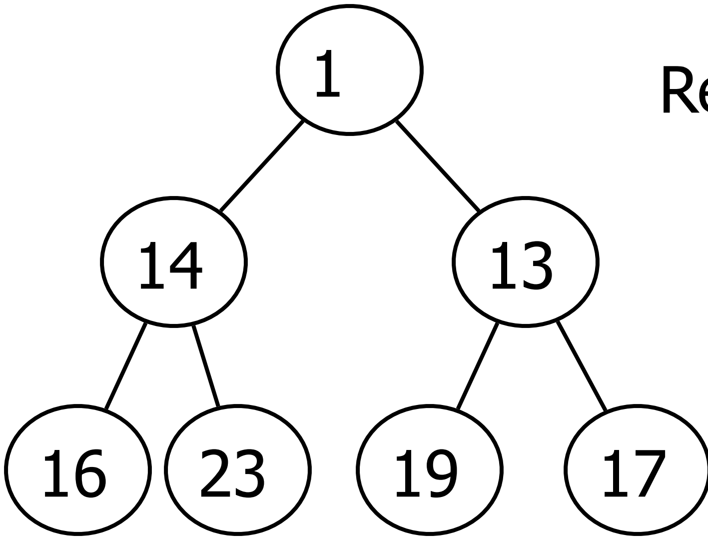
Heap Find Min/Max

- Finding the node with the highest priority is easy - it is just at the root
  - Running time = O(1)

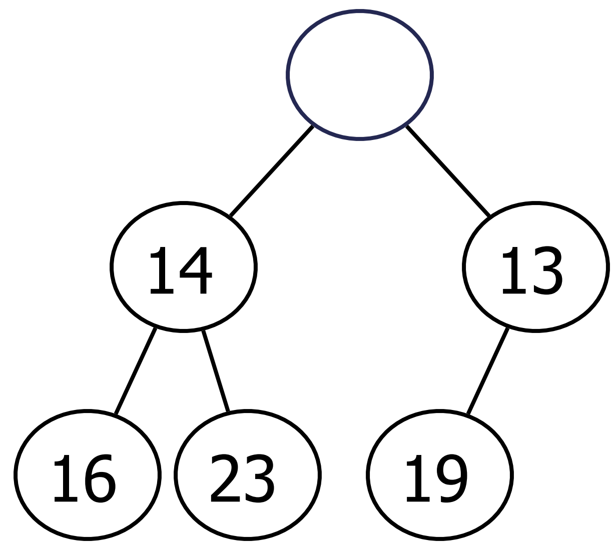
Heap – Remove (Extract Min or Max)

- Remove highest priority item
  - Makes a hole at the root
  - Want to remain a complete tree, so attempt to place last item in the heap into the hole
    - If item can be placed in hole without violation of the heap property, then done
    - Otherwise, trickle down
    - Pick the child with the highest priority

Min Heap – Remove

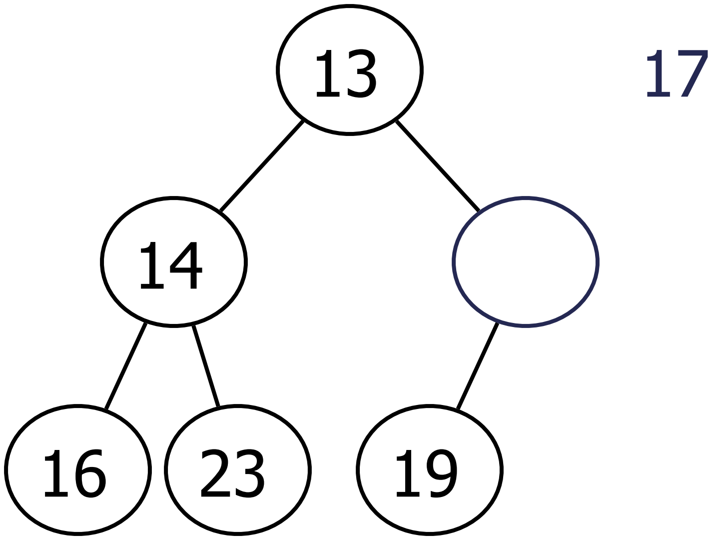


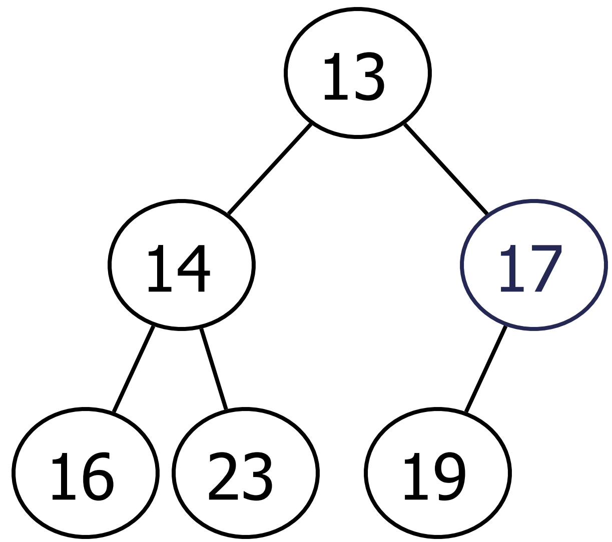
Remove 1



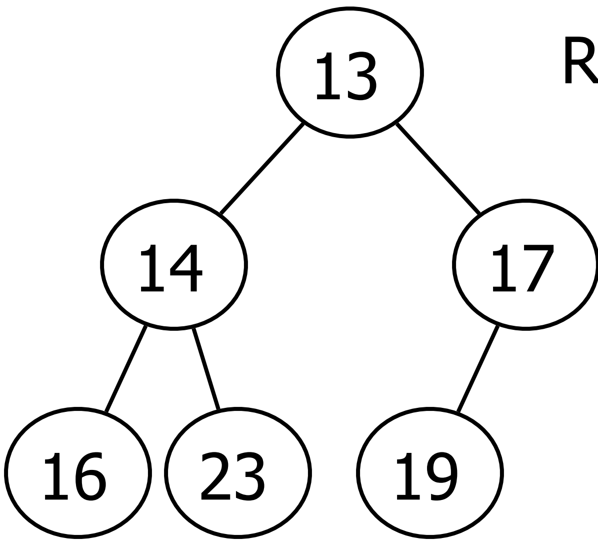
17

Min Heap - Remove

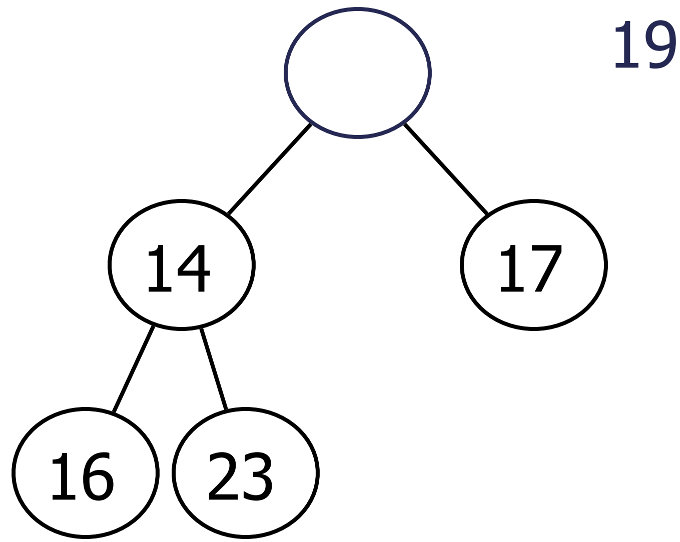




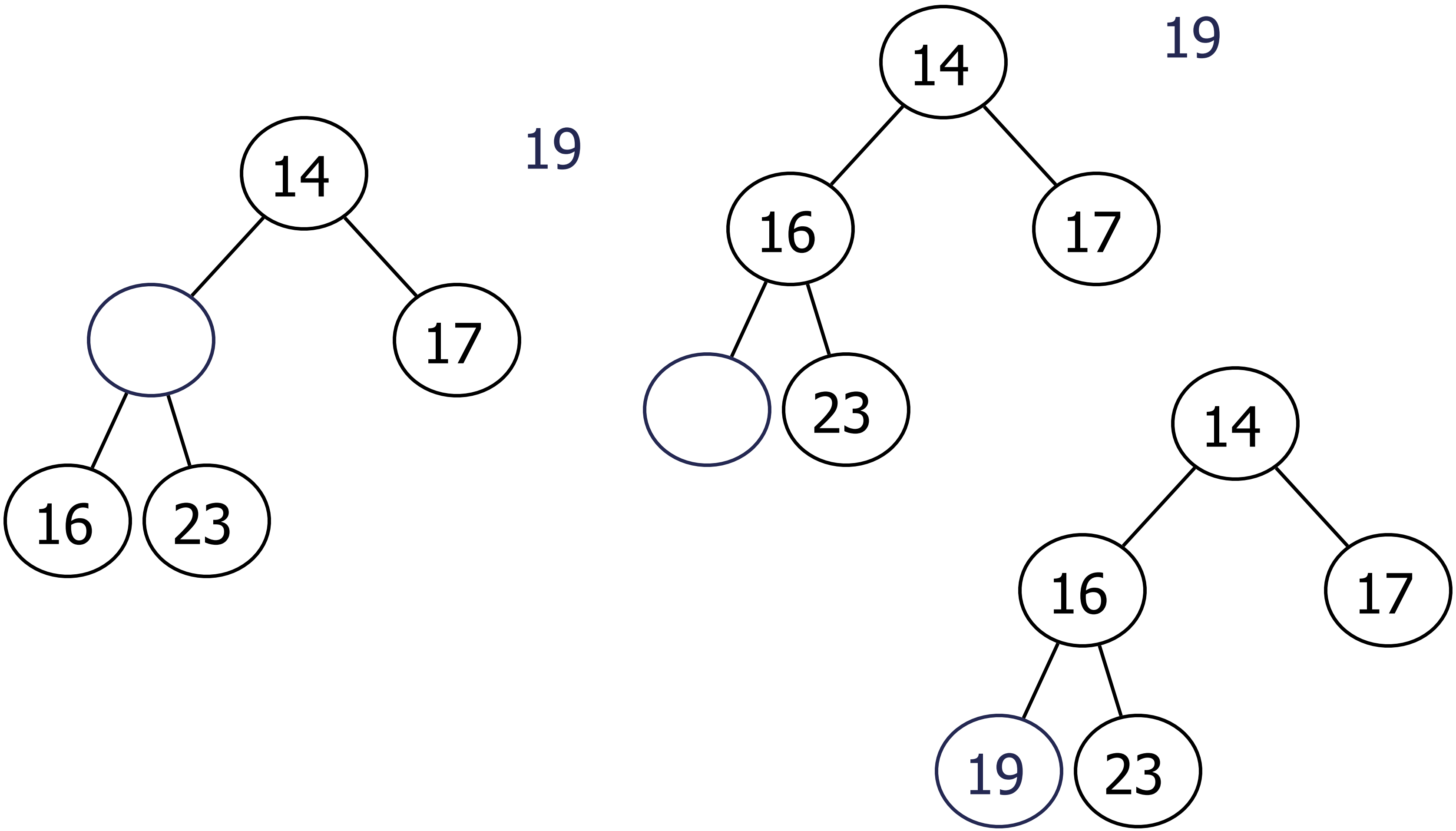
Min Heap - Remove



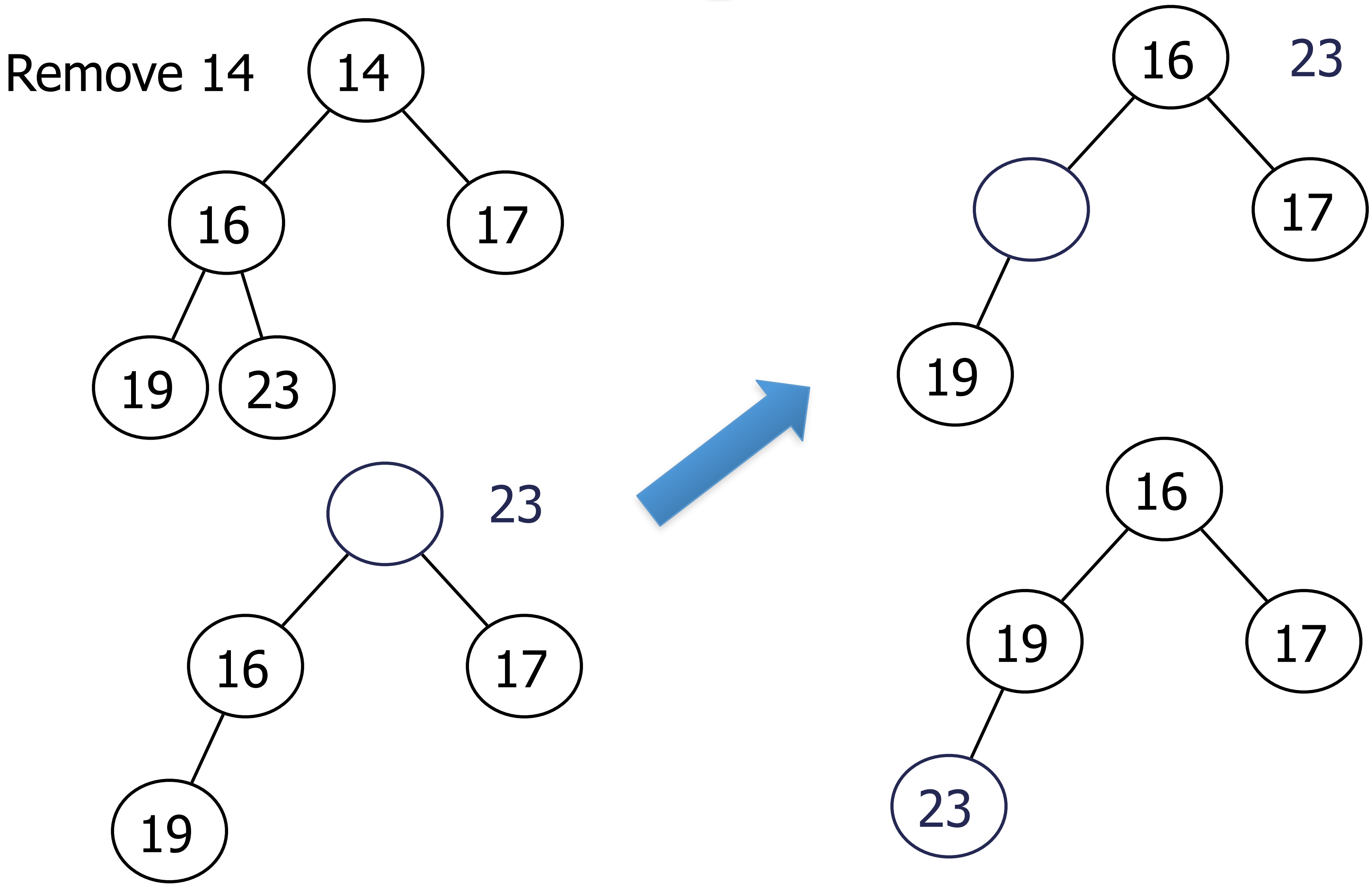
Remove 13



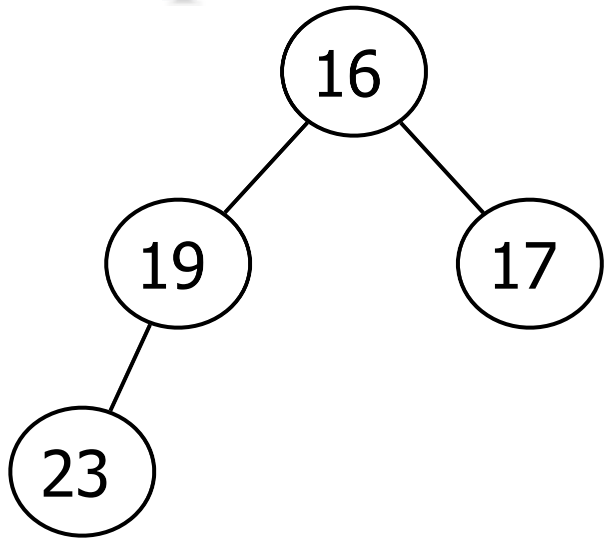
Min Heap - Remove



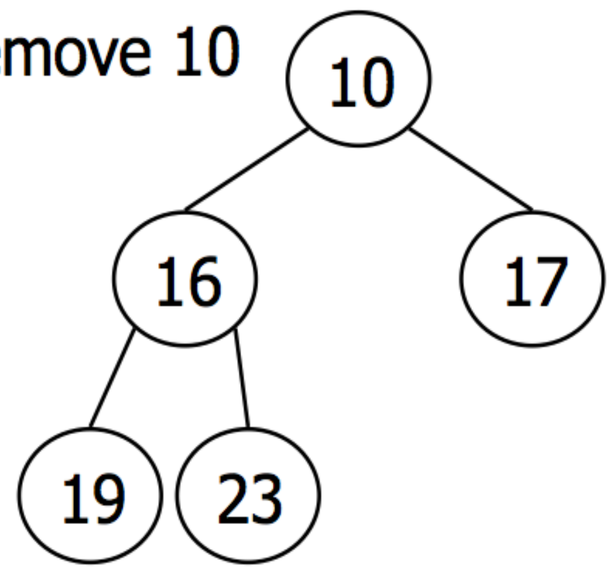
Min Heap - Remove



Exam7 Q5
Min Heap – Insert

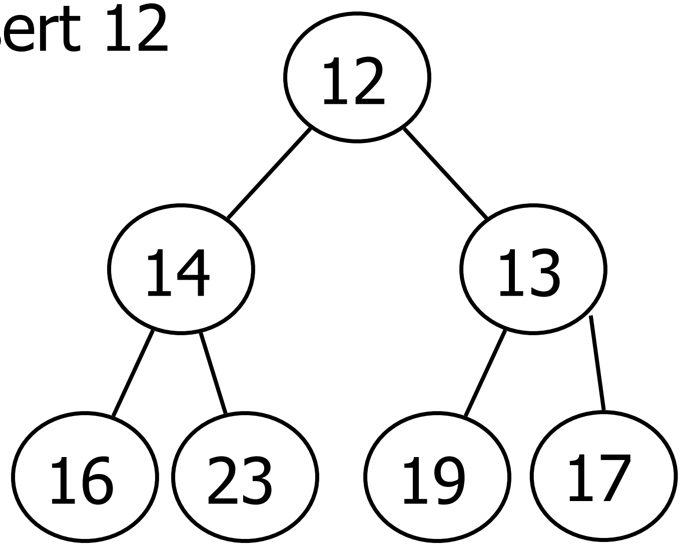


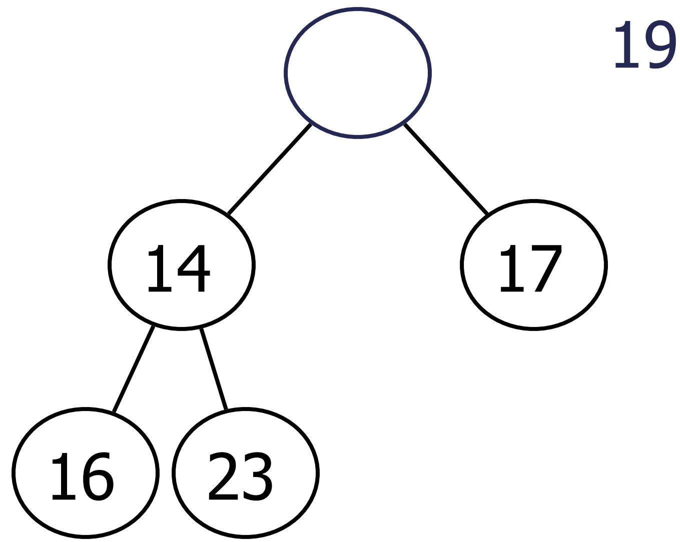
Remove 10



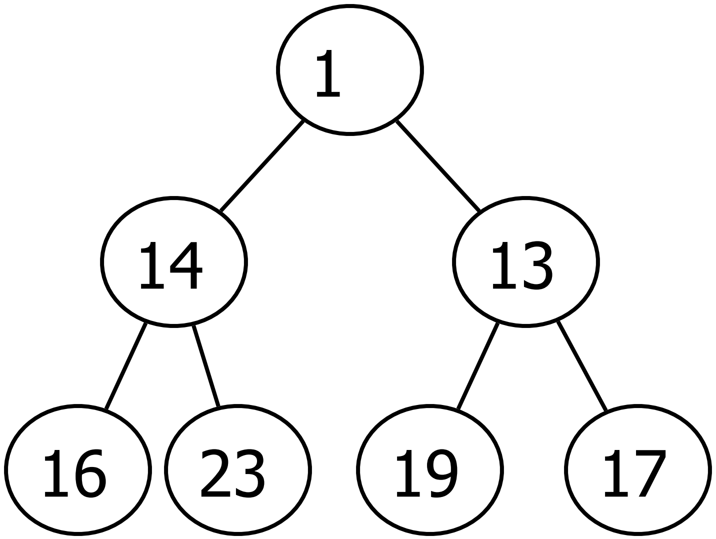
Exam7 Q5
Min Heap – Insert

Insert 12





Min Heap - Remove



1,14,13,16,23,19,17
13,14,17,16,23,19
14,16,17,19,23
16,19,17,23
17,19,23
19,23
23

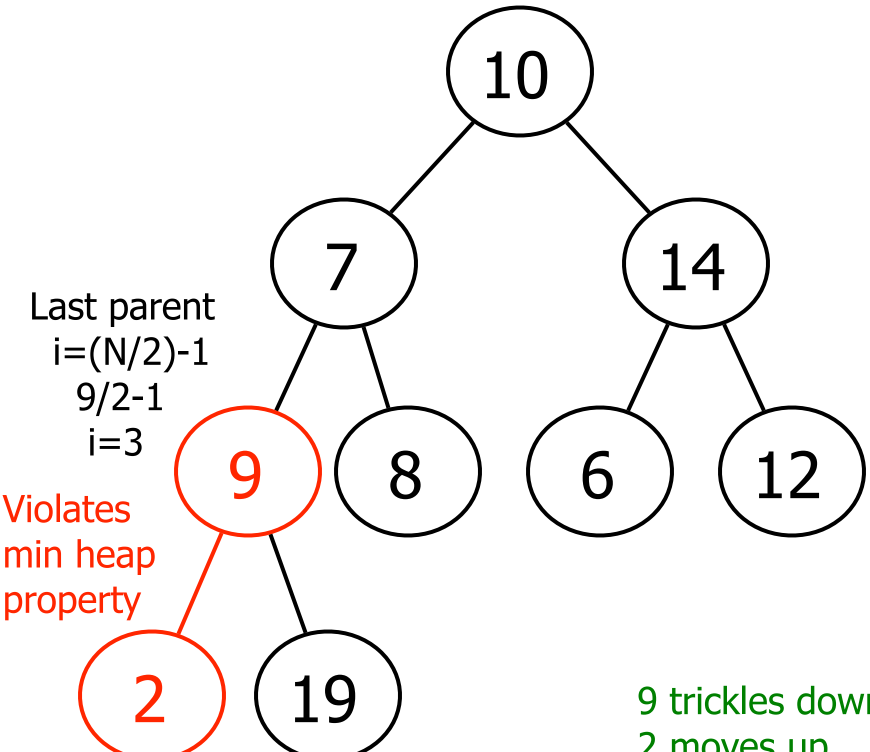
Building a Heap

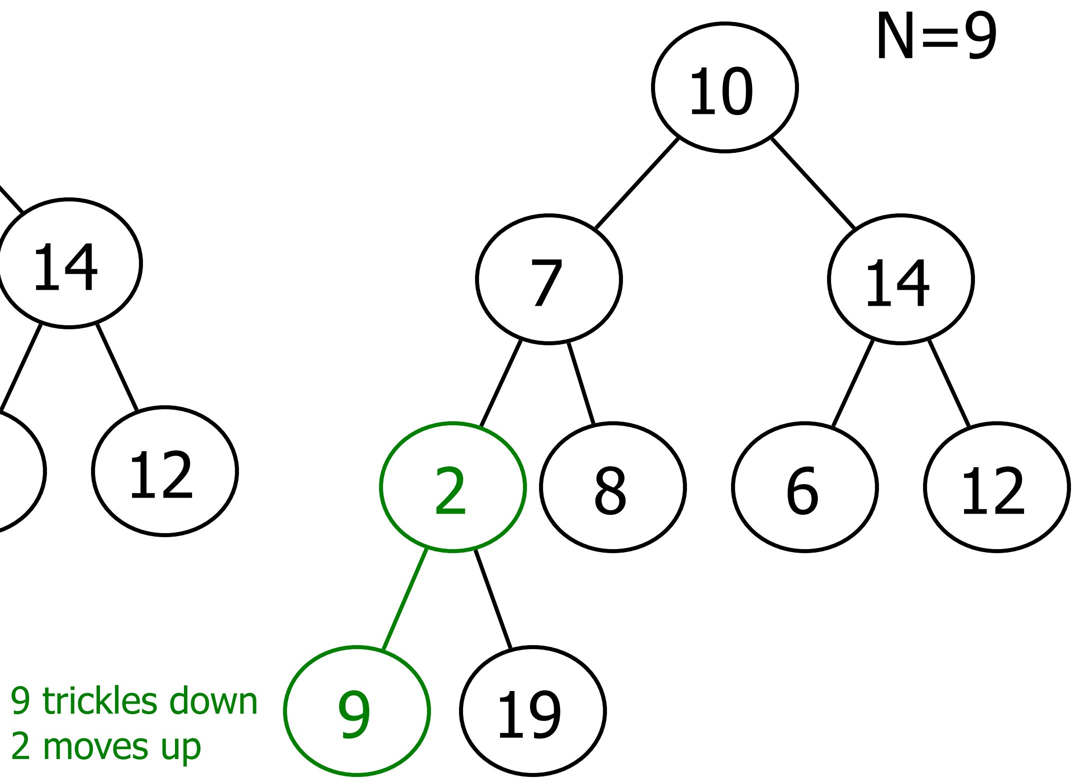
- Naïve method
  - Perform N insert operations - O(nlogn)
- Better solution
  - Given an array of unordered items:

```
for (int i=(N/2)-1; i >= 0; i--)
        Trickle down
```

Building a Min Heap

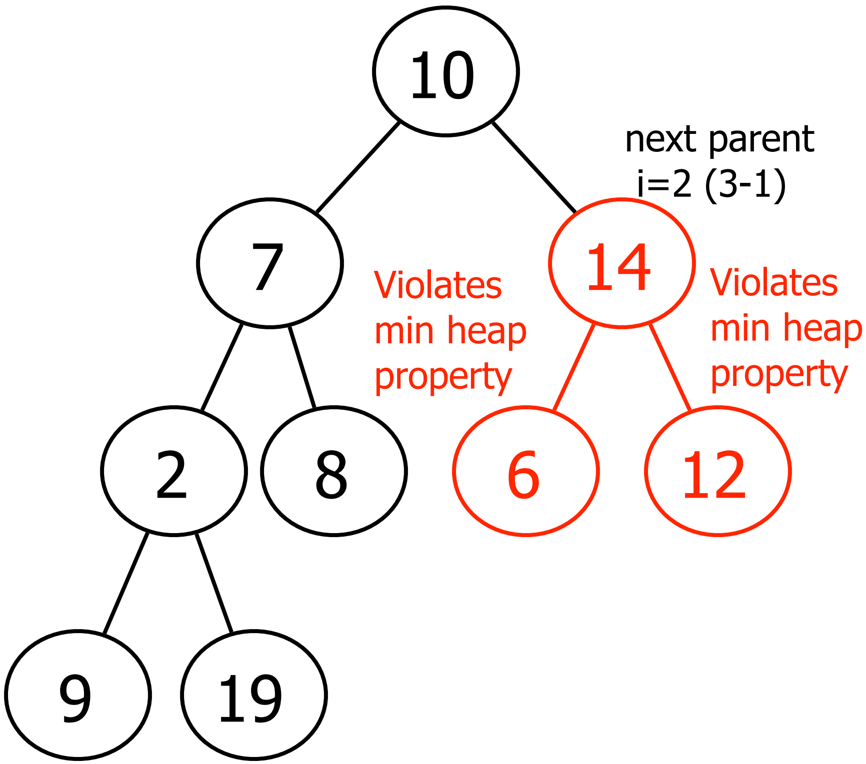
i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
10	7	14	9	8	6	12	2	19



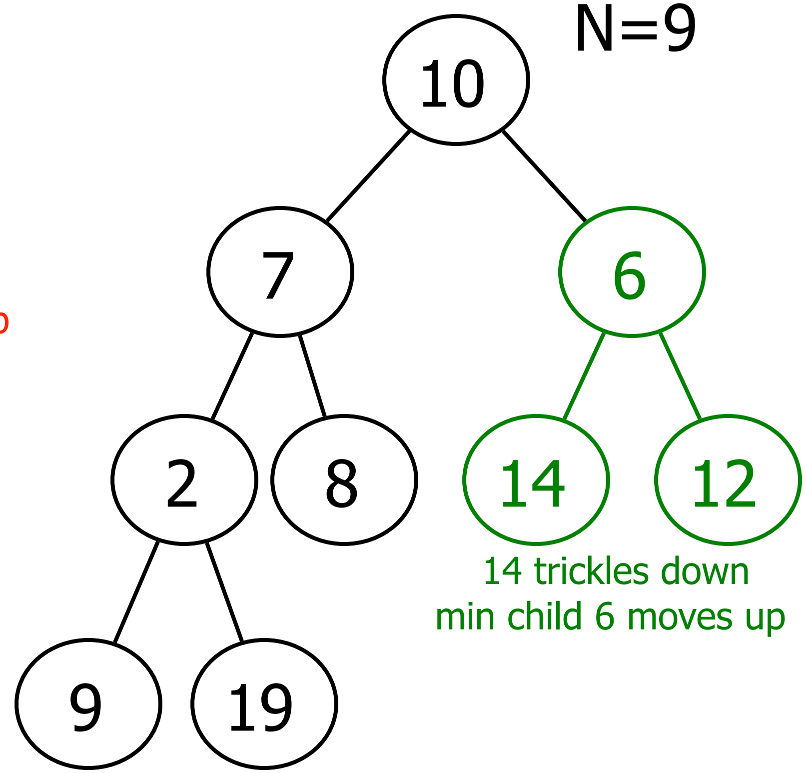


Building a Min Heap

i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
10	7	14	2	8	6	12	9	19



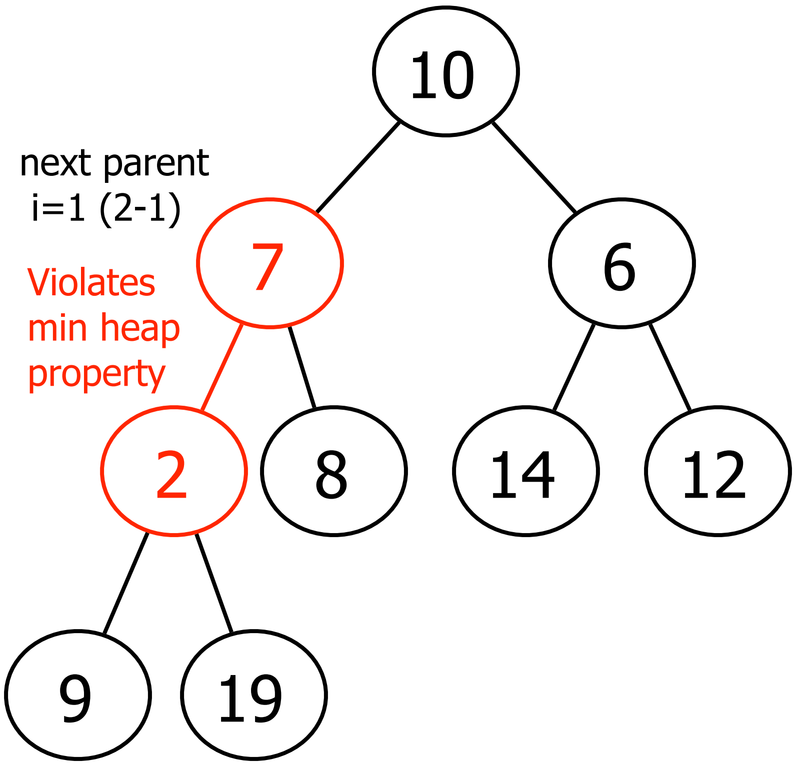
N=9

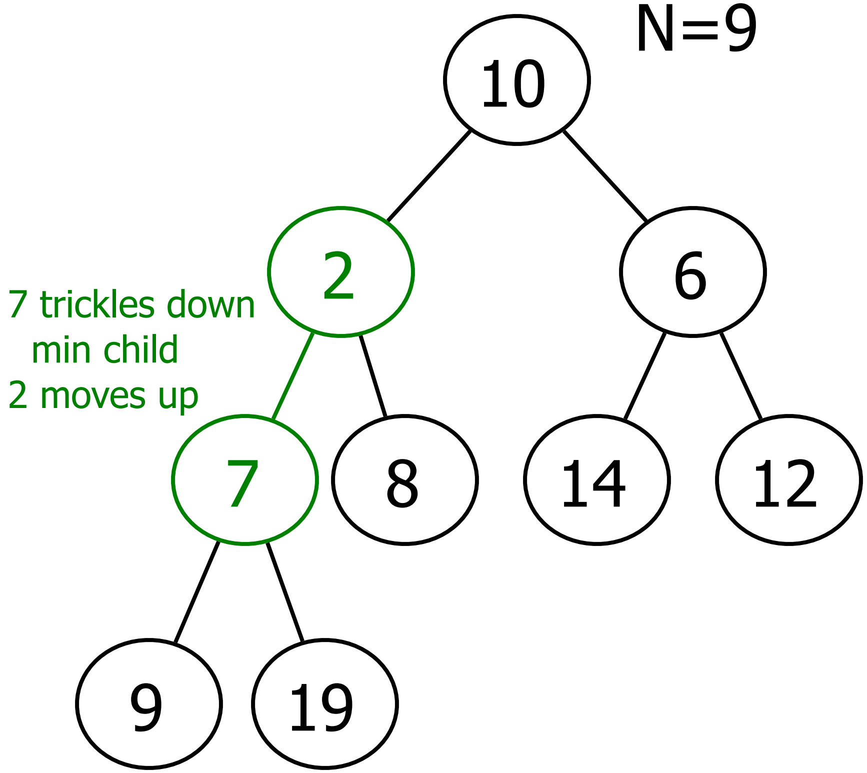


Building a Min Heap

i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
10	7	6	2	8	14	12	9	19

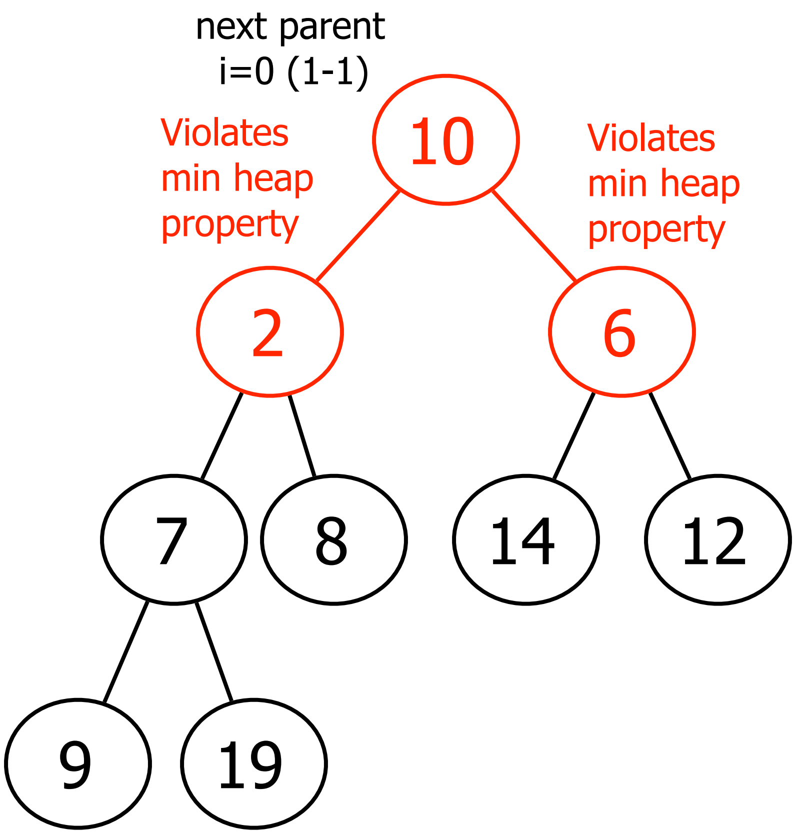
N=9

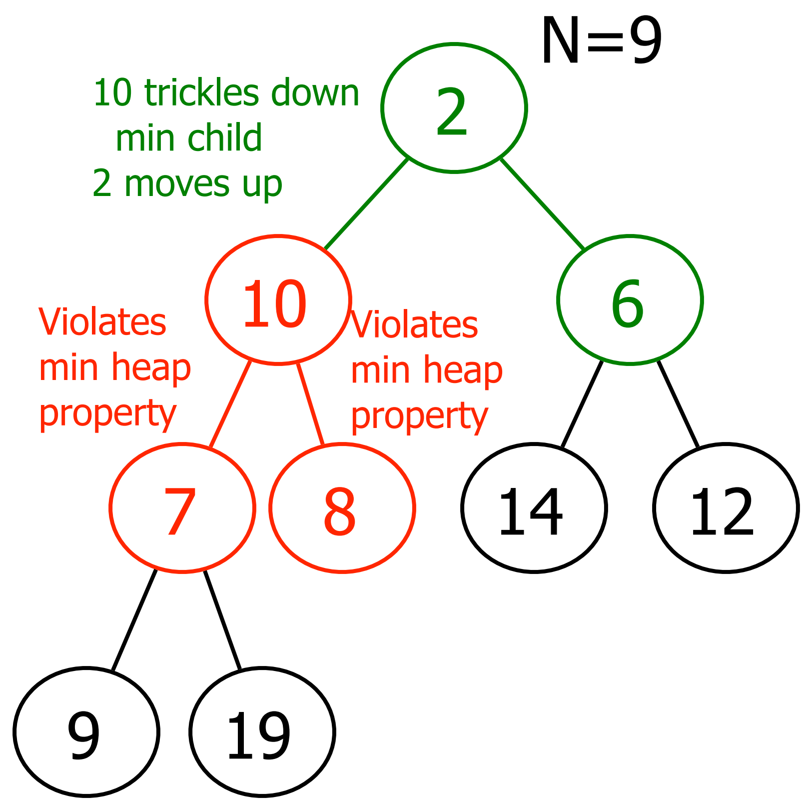




Building a Min Heap

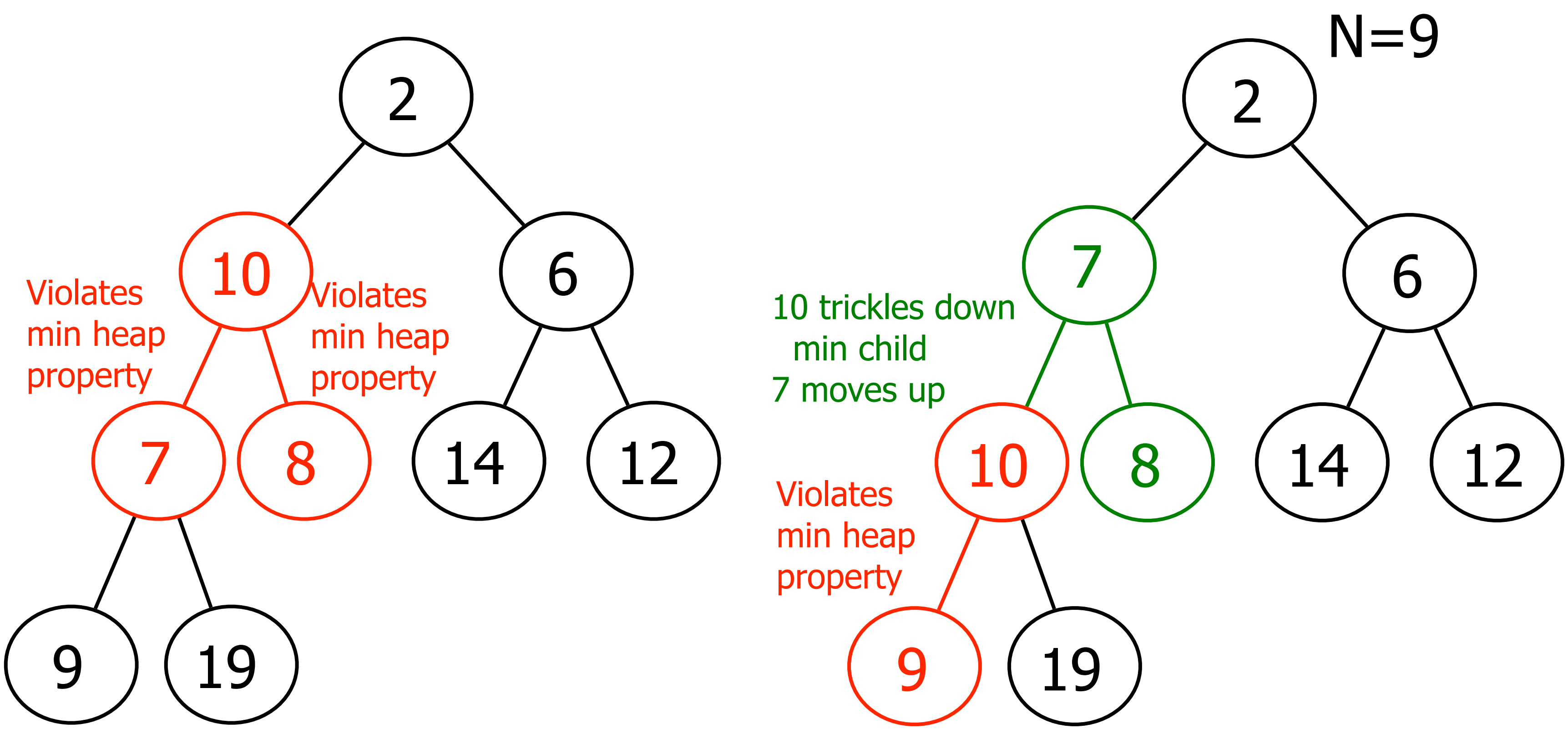
i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
10	2	6	7	8	14	12	9	19





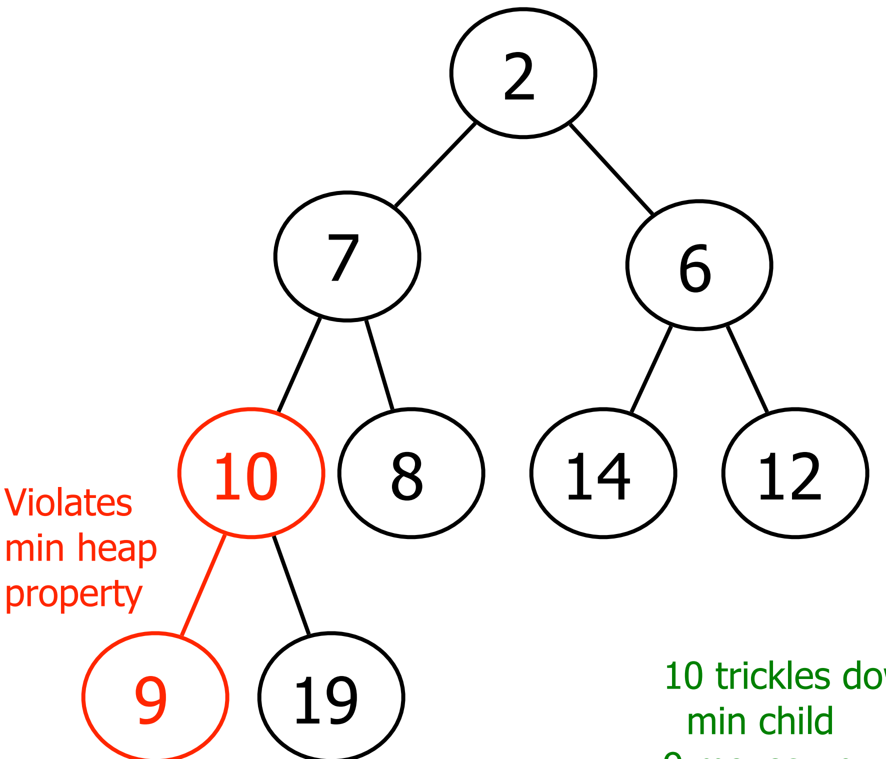
Building a Min Heap

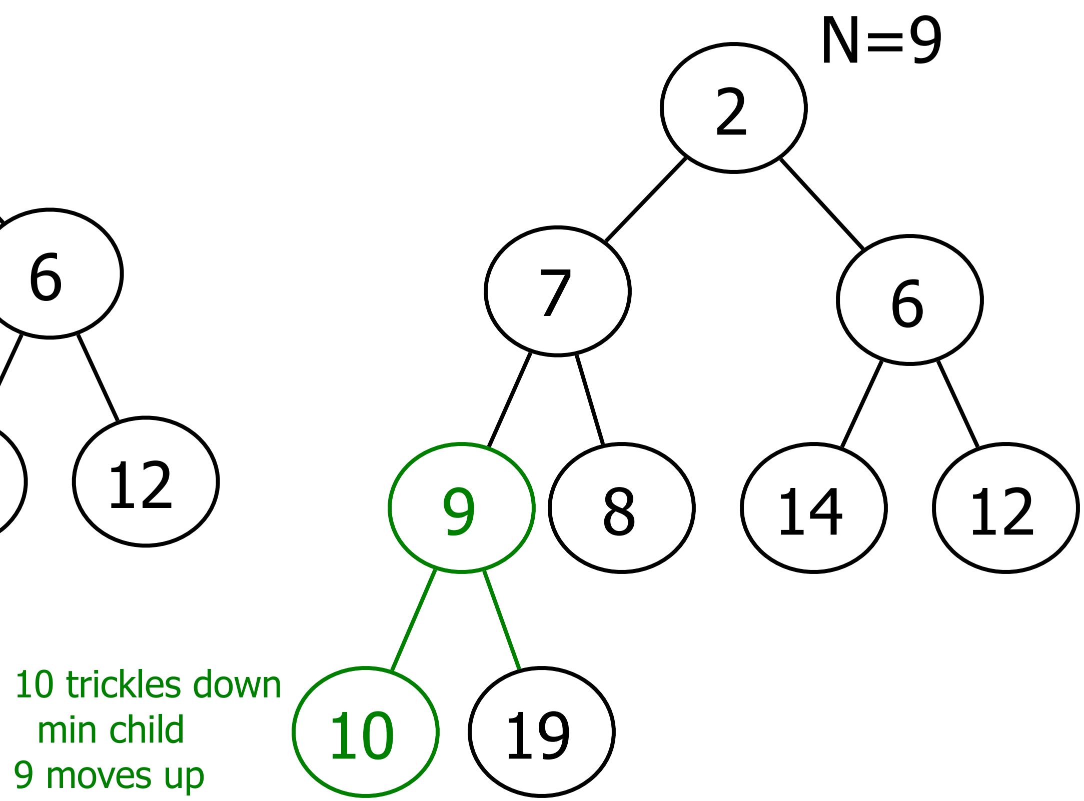
i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
2	10	6	7	8	14	12	9	19



Building a Min Heap

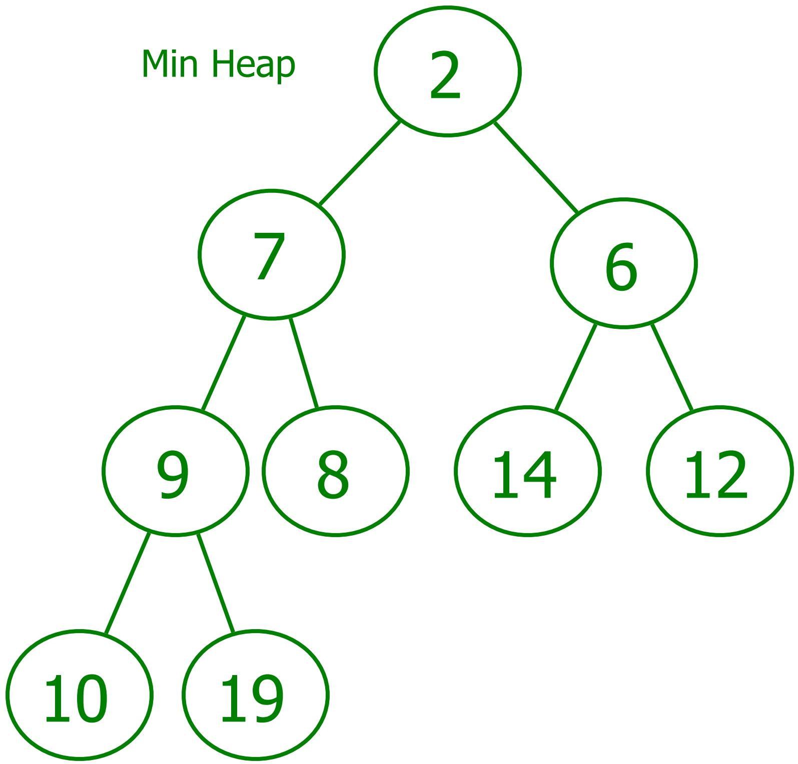
i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
2	7	6	10	8	14	12	9	19





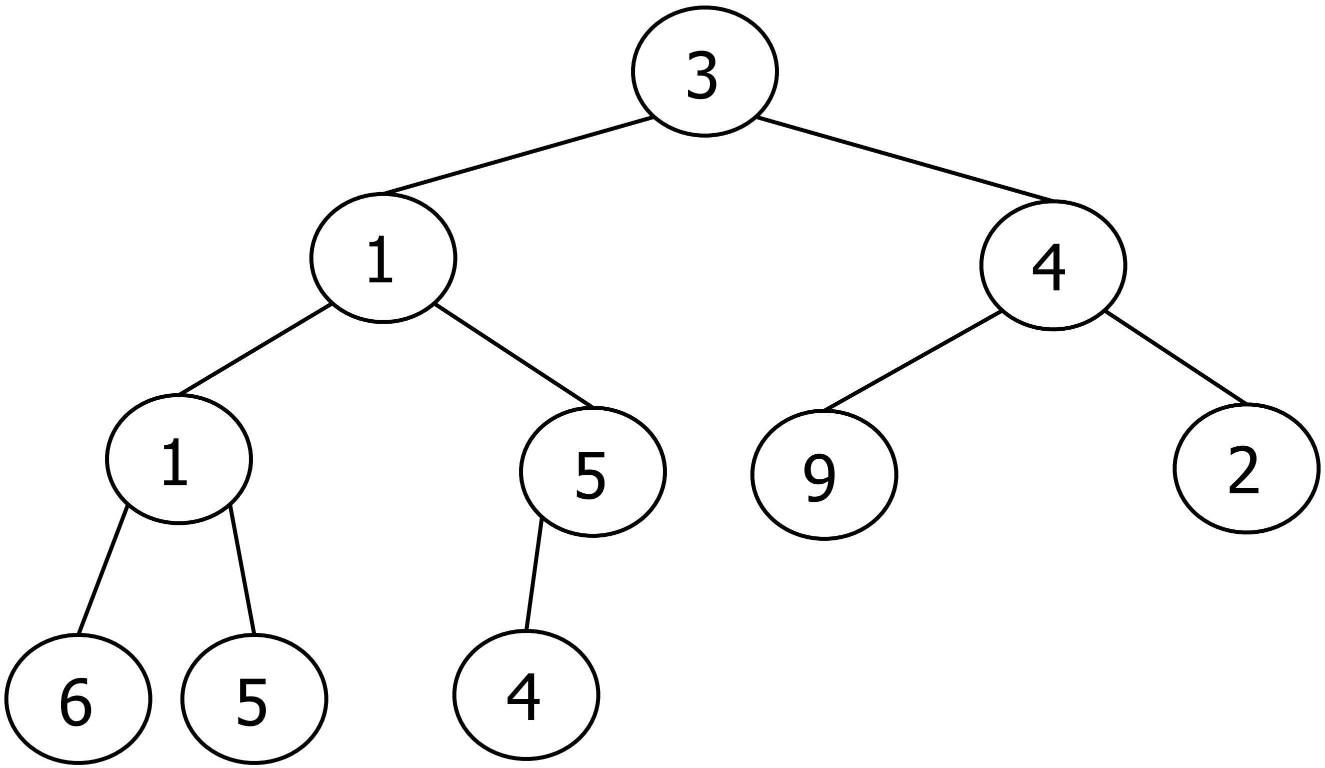
Building a Min Heap

i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
2	7	6	9	8	14	12	10	19



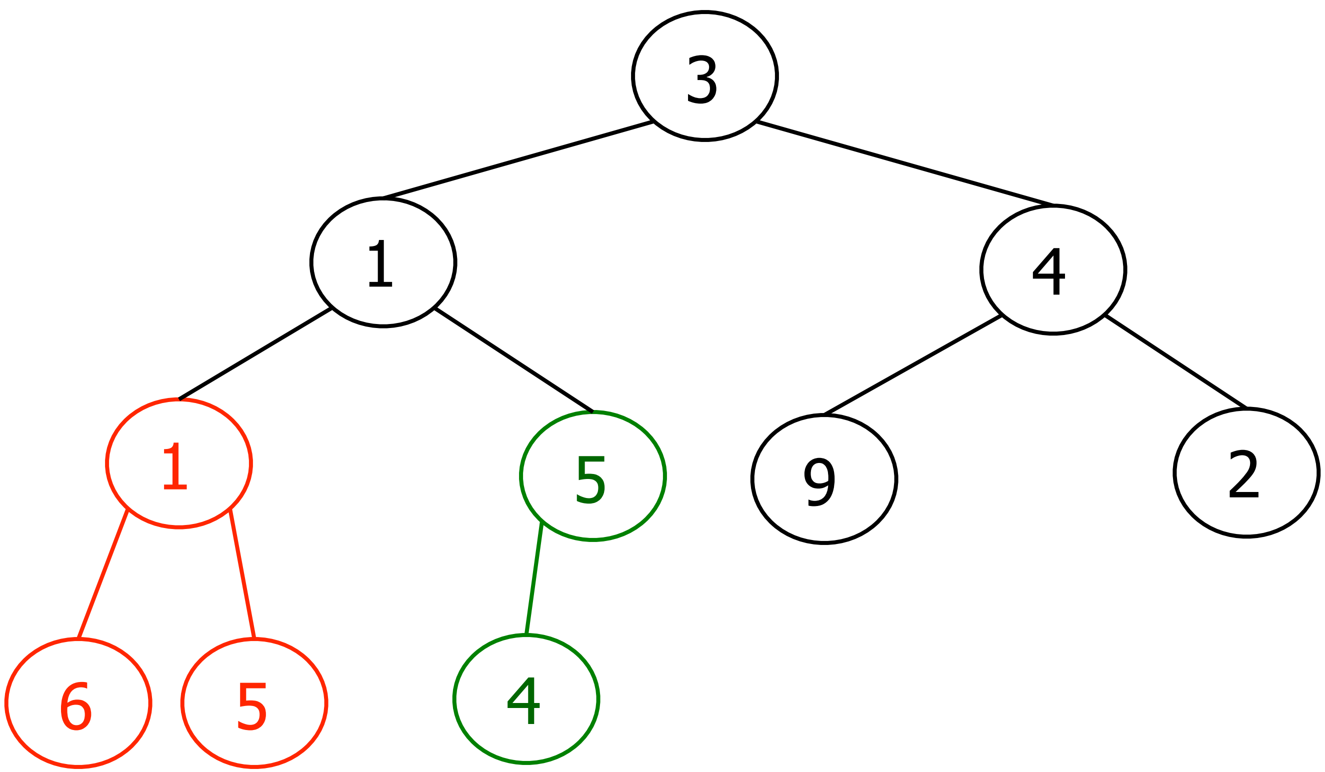
ICE - Max Heap Build

3	1	4	1	5	9	2	6	5	4



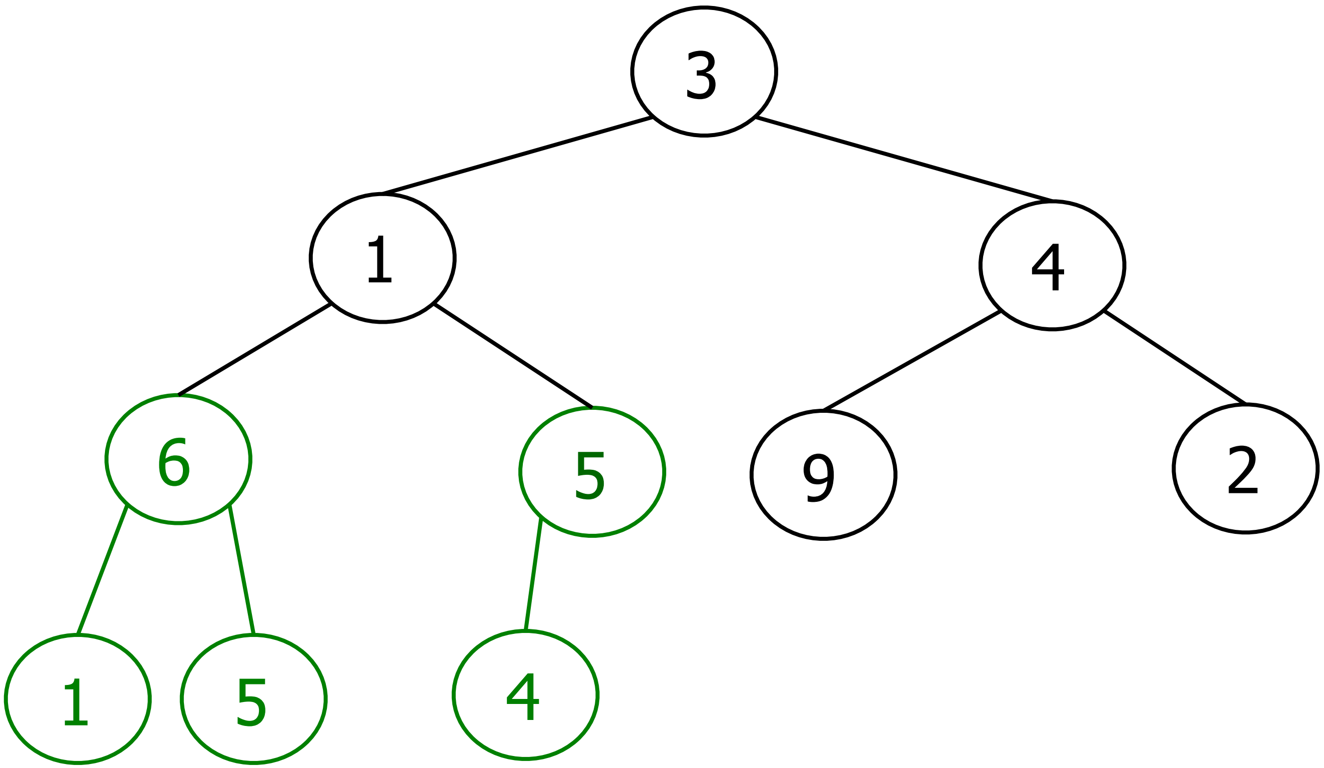
ICE - Max Heap Build

3	1	4	1	5	9	2	6	5	4



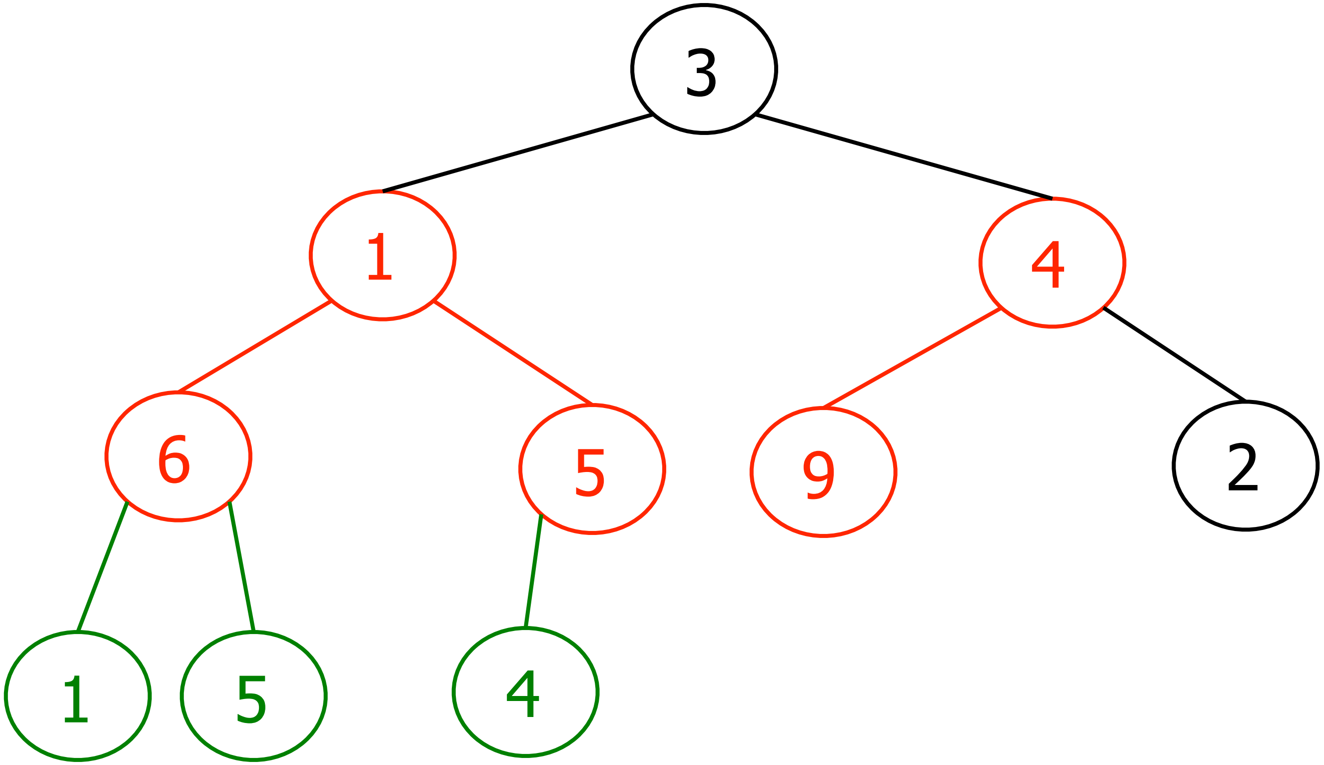
ICE - Max Heap Build

3	1	4	6	5	9	2	1	5	4



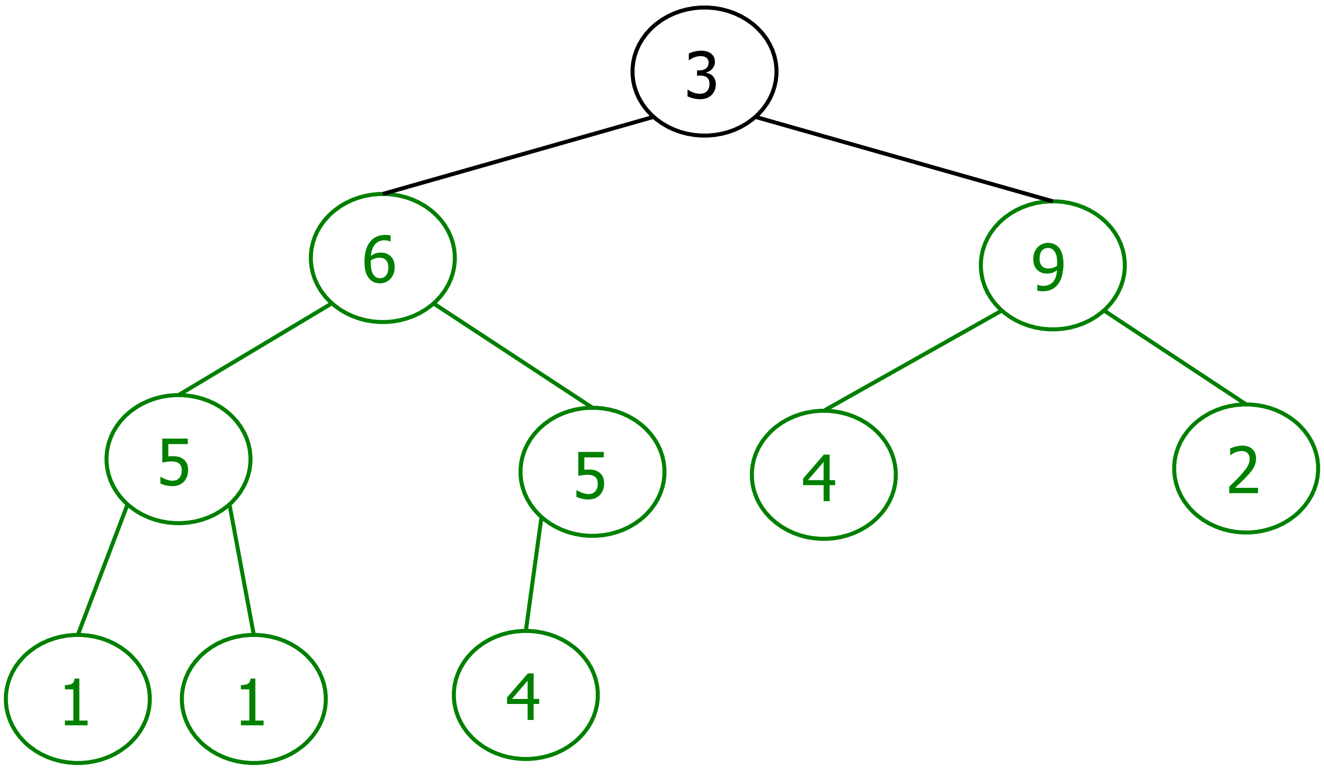
ICE - Max Heap Build

3	1	4	6	5	9	2	1	5	4



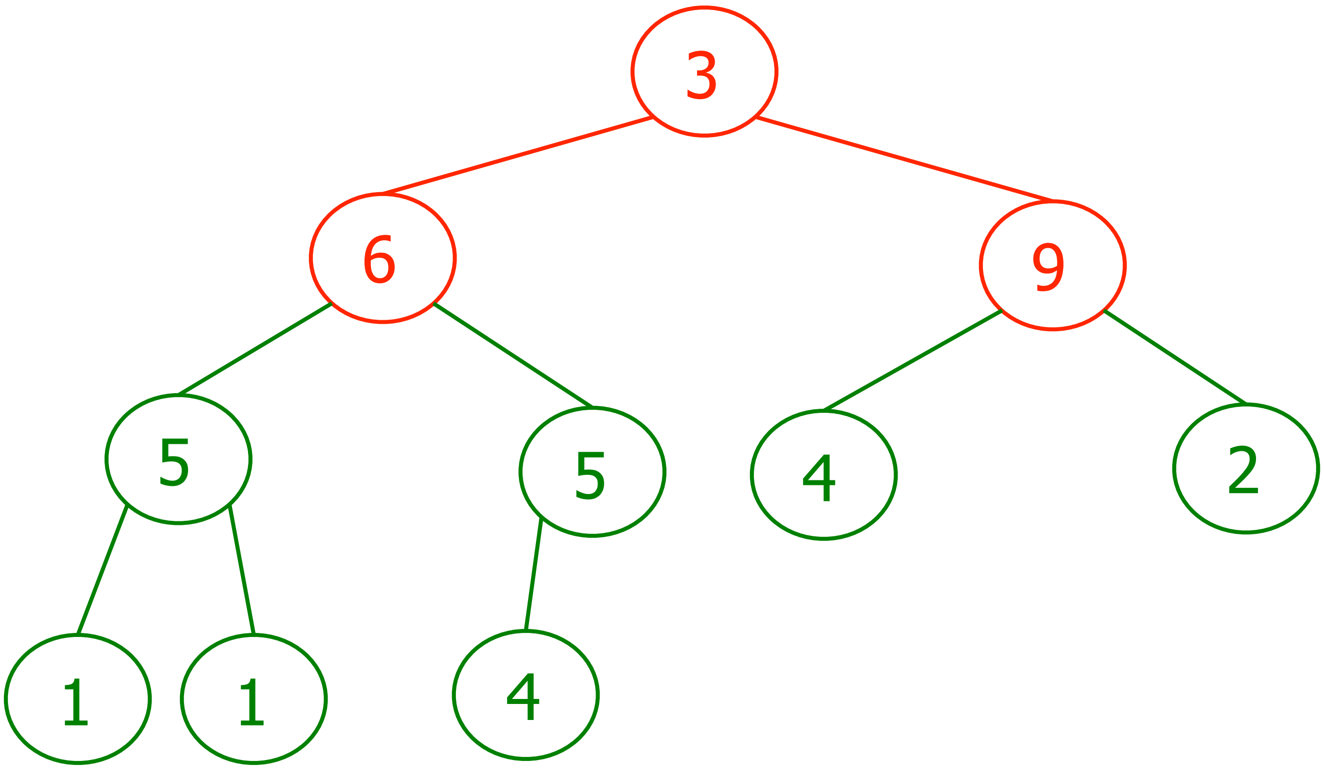
ICE - Max Heap Build

3	6	9	5	5	4	2	1	1	4



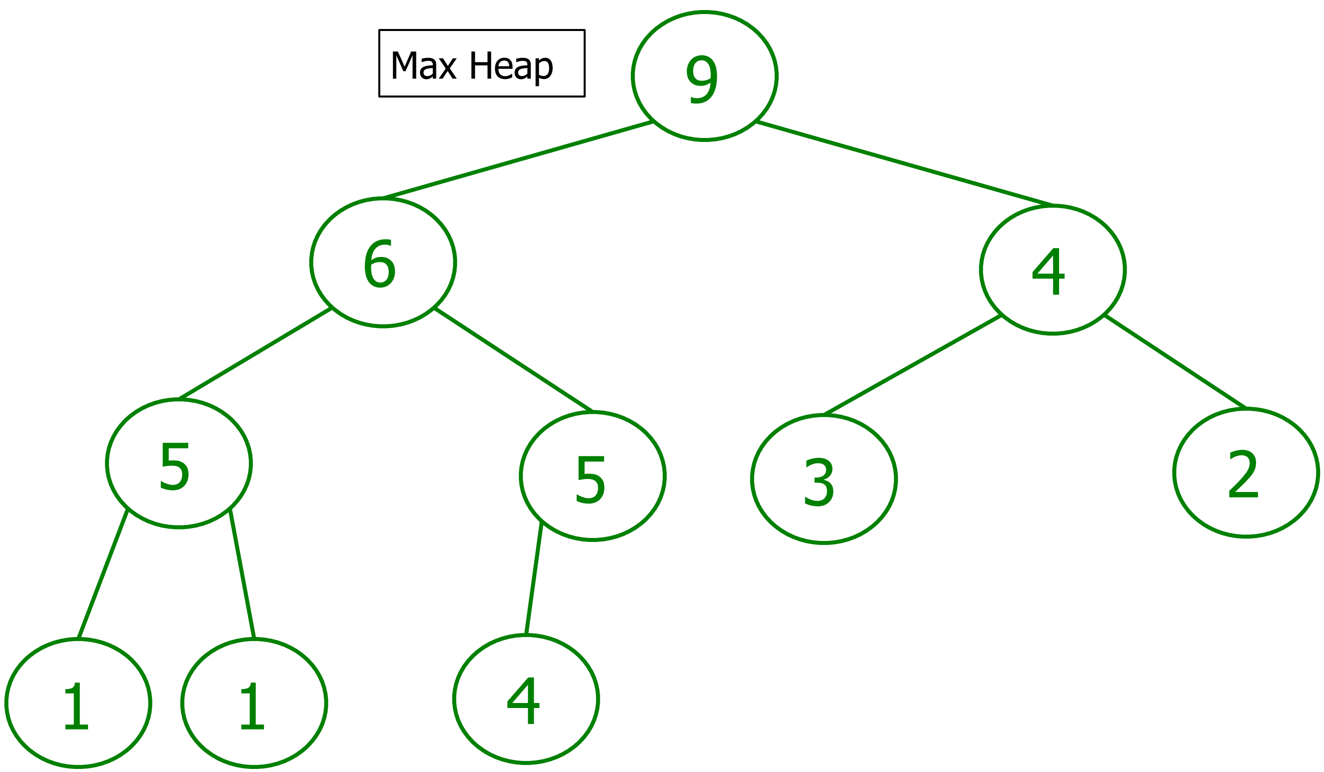
ICE - Max Heap Build

3	6	9	5	5	4	2	1	1	4



ICE - Max Heap Build

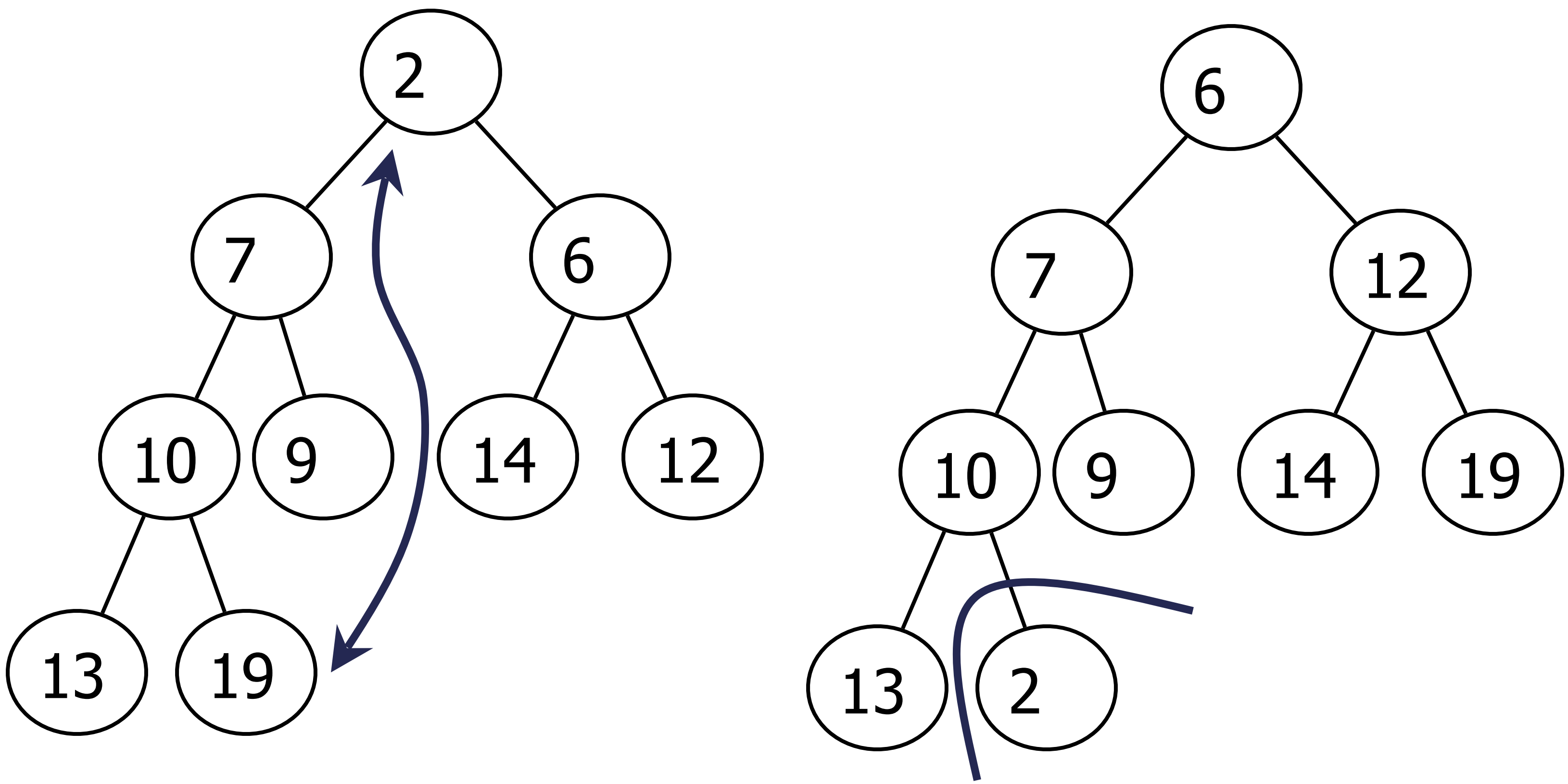
9	6	4	5	5	3	2	1	1	4



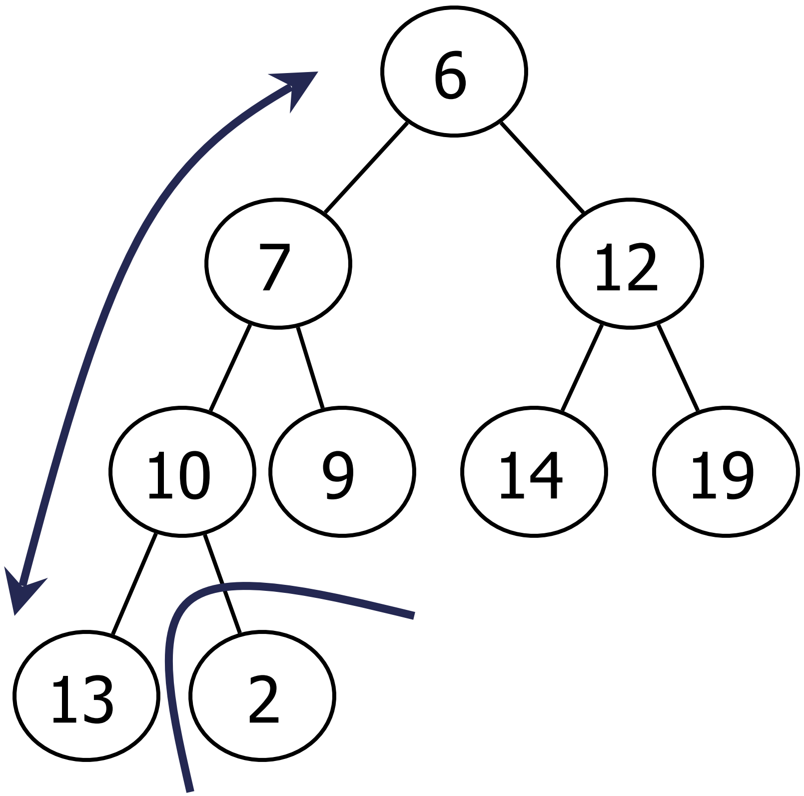
Heapsort

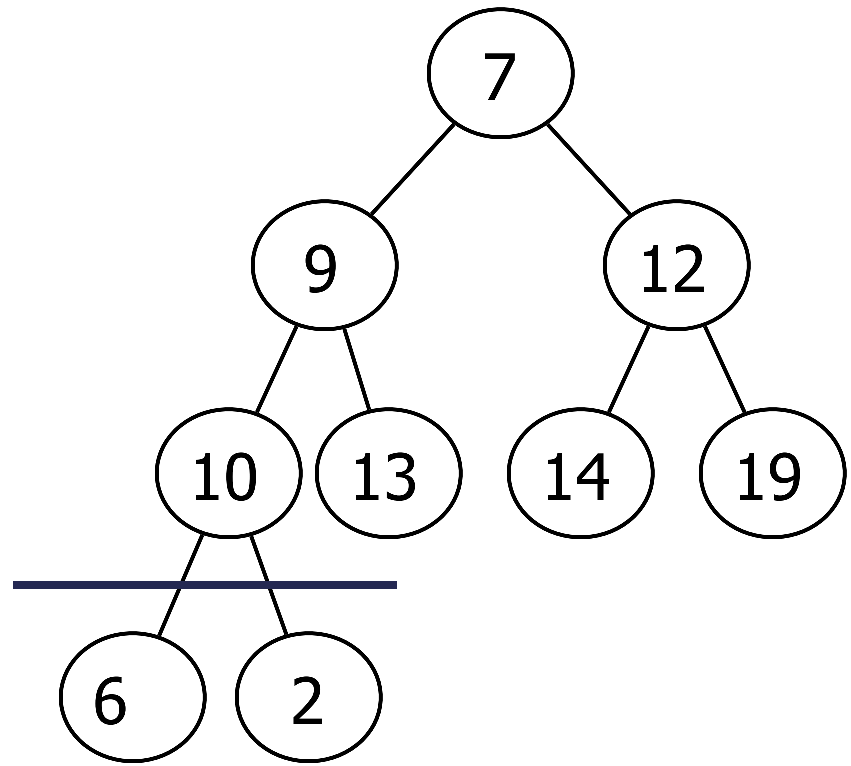
- Build heap for the opposite of what you want
  - Max heap for ascending order
  - Min heap for descending order
- Take root and place in last array position, then think of array as 1 smaller
- Trickle down from root to rebuild heap

Heapsort - Sort in descending order

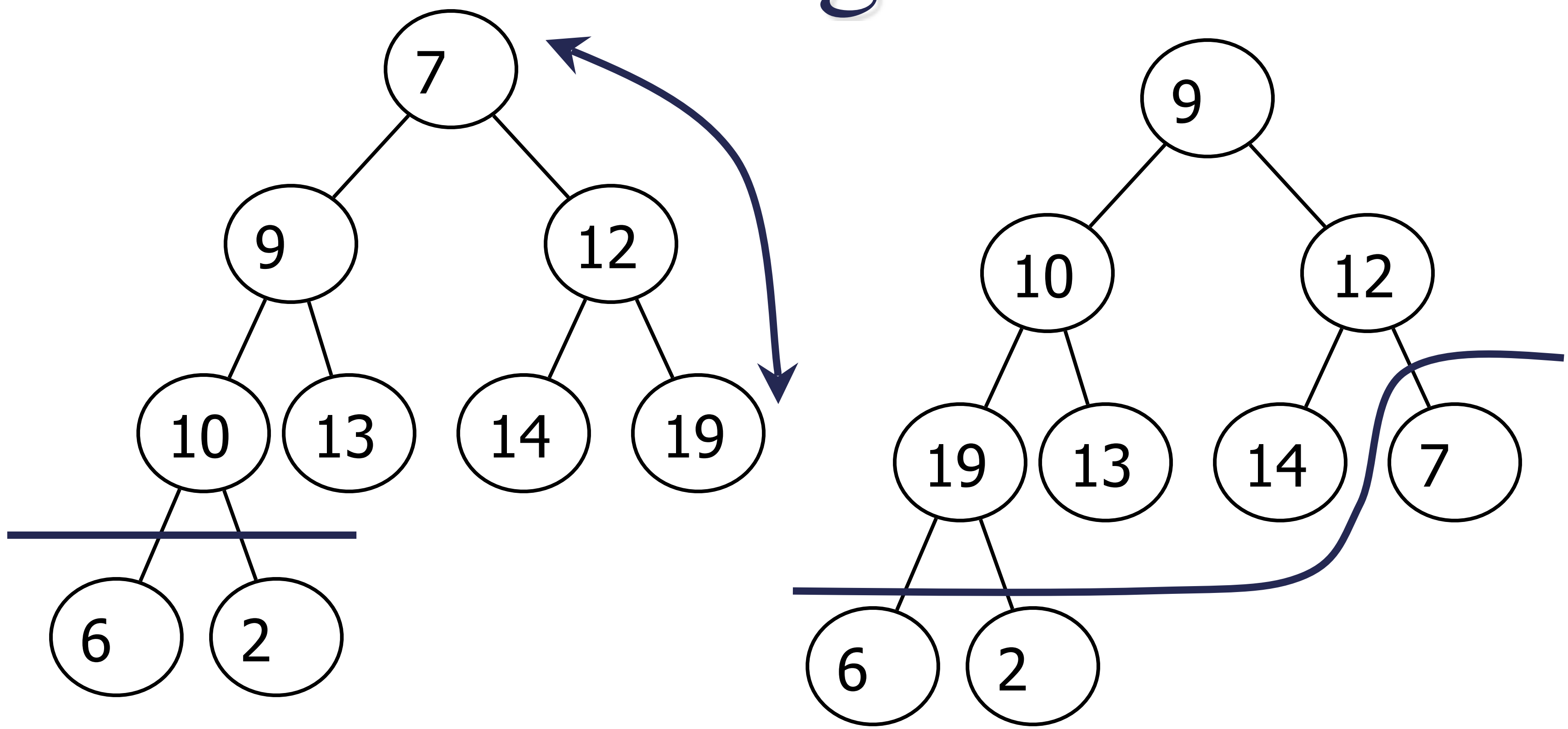


Heapsort - Sort in descending order

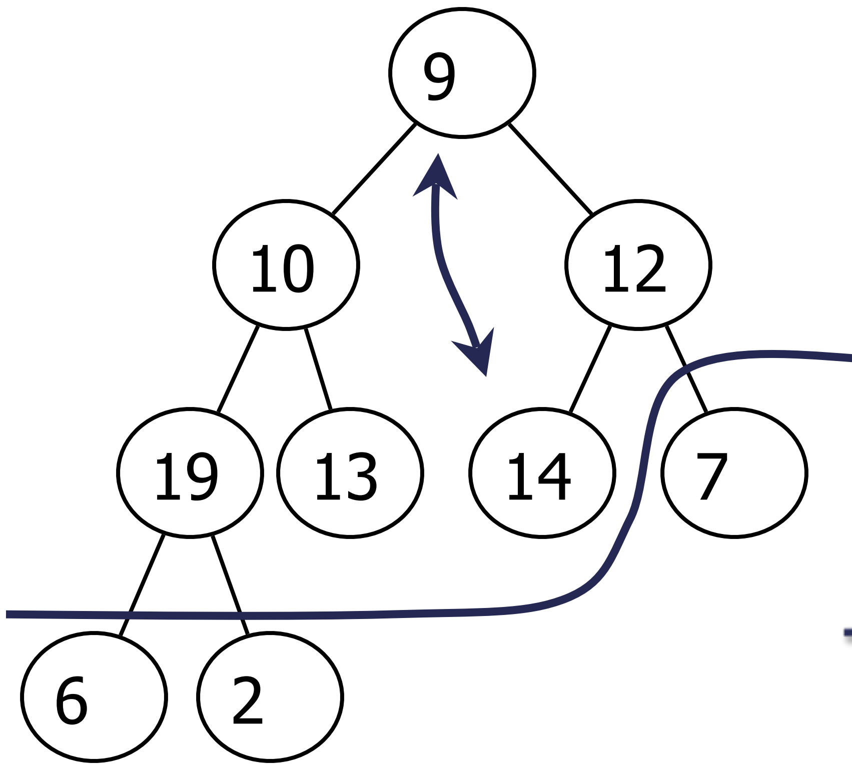


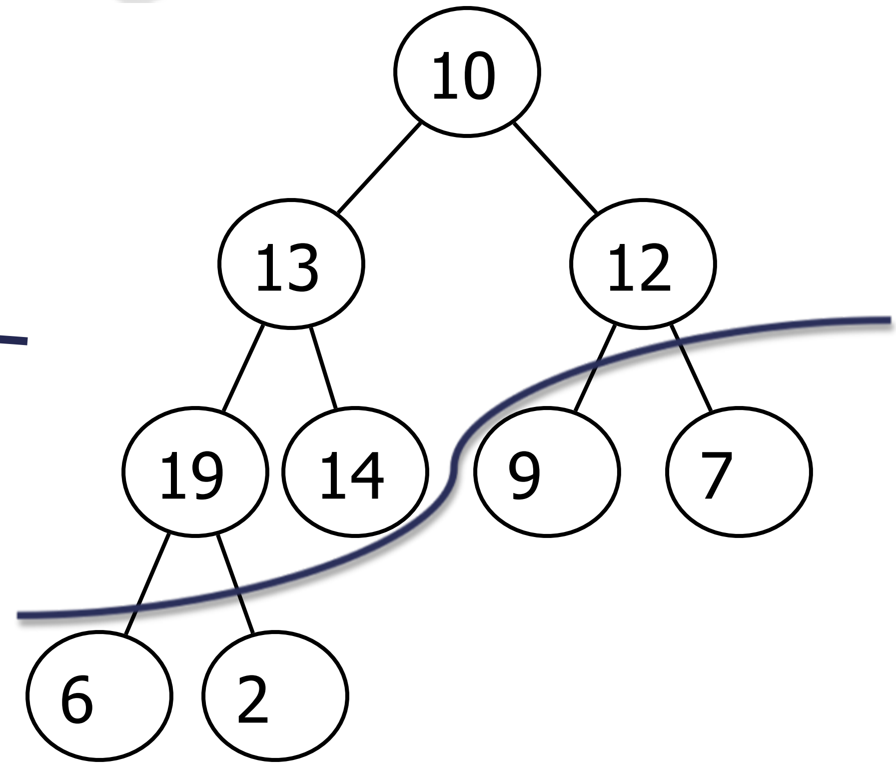


Heapsort - Sort in descending order

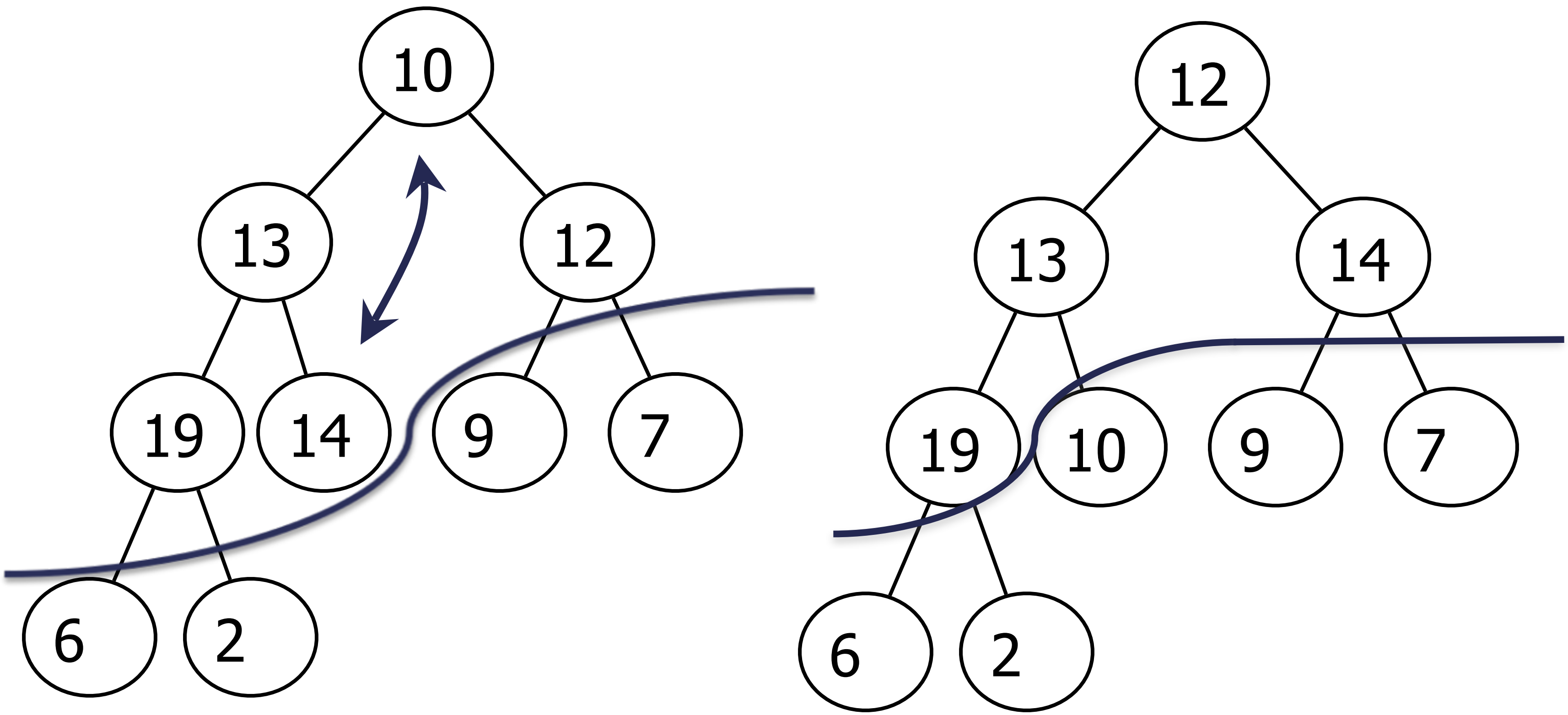


Heapsort - Sort in descending order

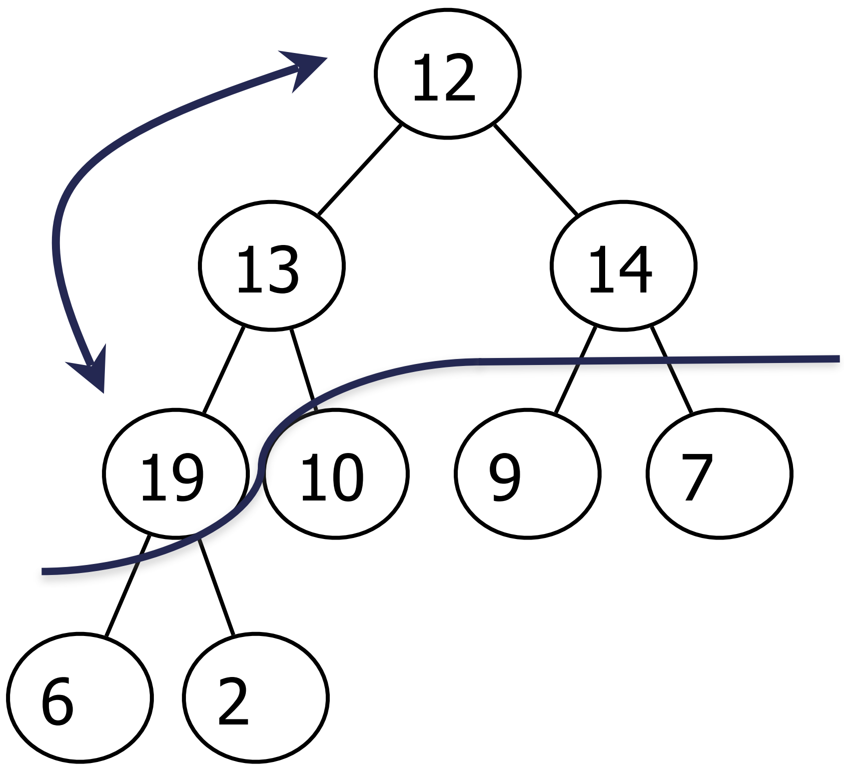


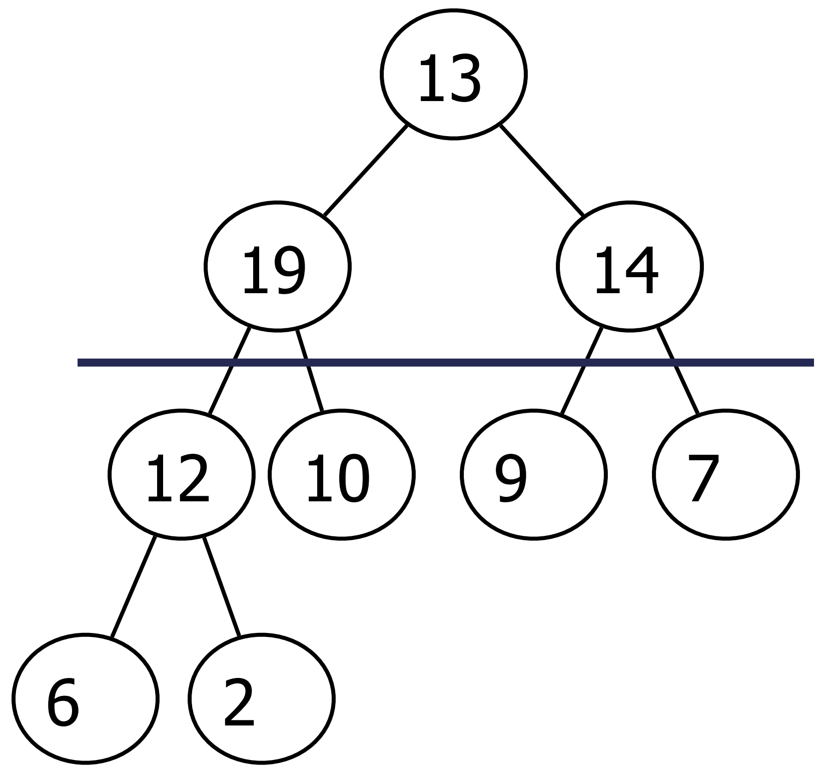


Heapsort - Sort in descending order

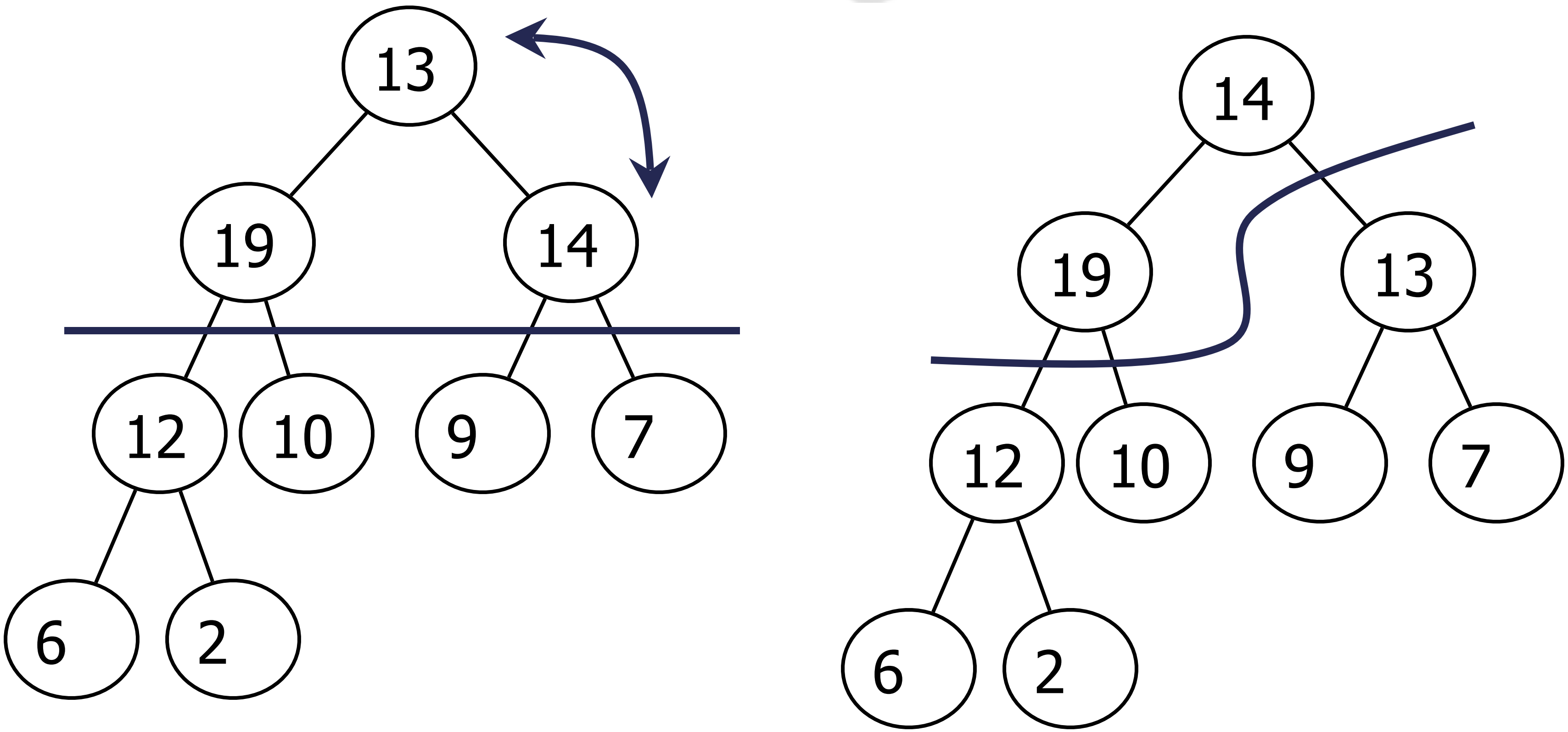


Heapsort - Sort in descending order

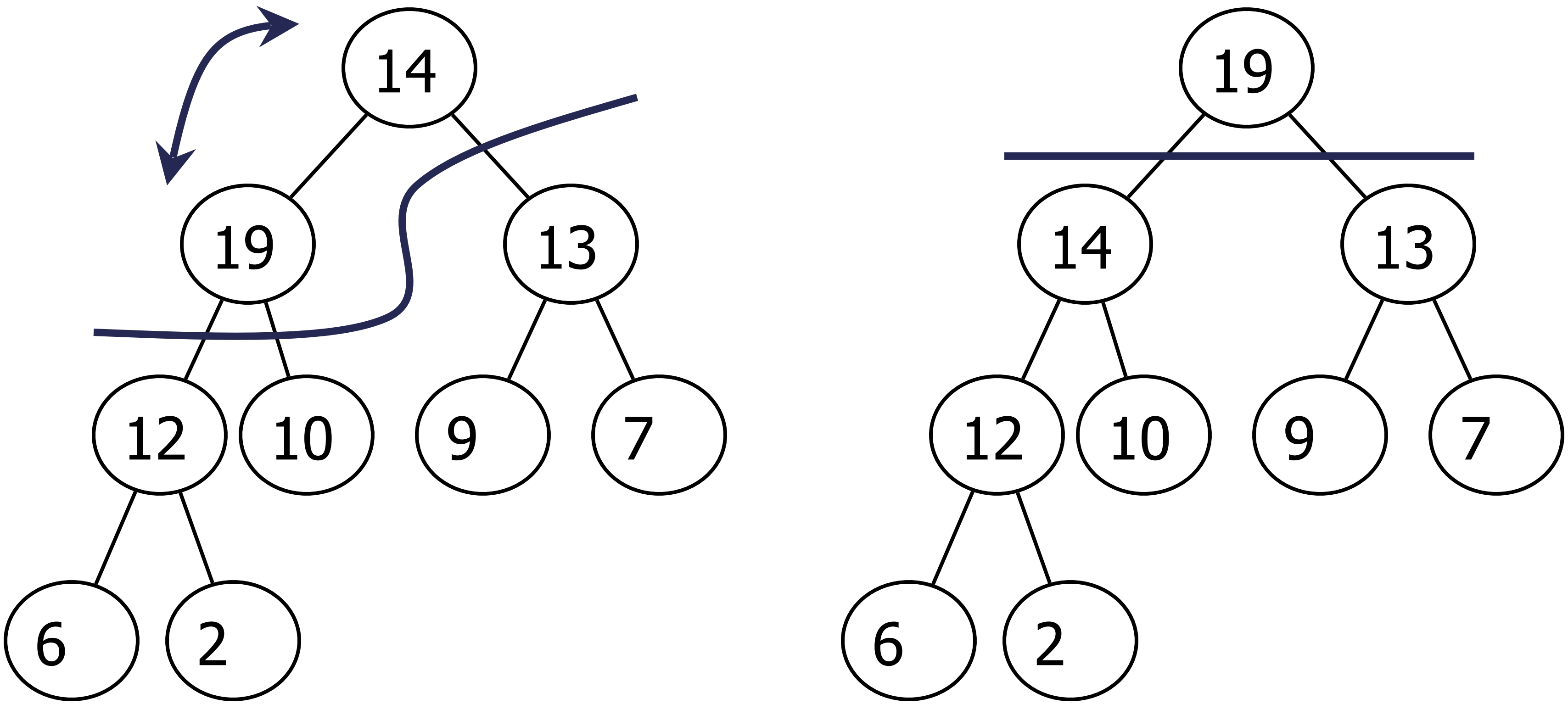




Heapsort - Sort in descending order



Heapsort - Sort in descending order



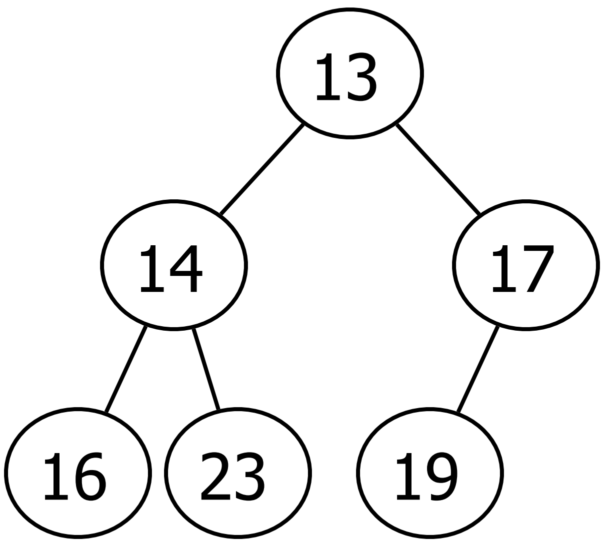
Heapsort Run Time

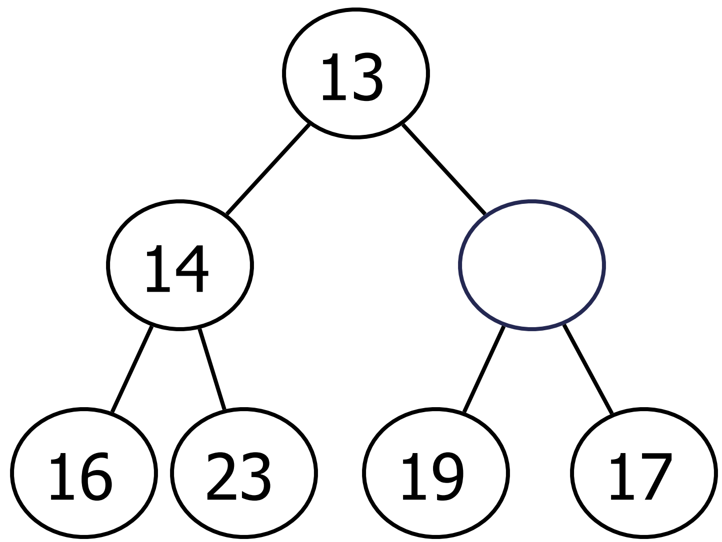
- Build step - O(n)
- Sorting step - O(nlogn)
- Heapsort - O(n) + O(nlogn) = O(nlogn)

Exam 7 Min Heap - Insert

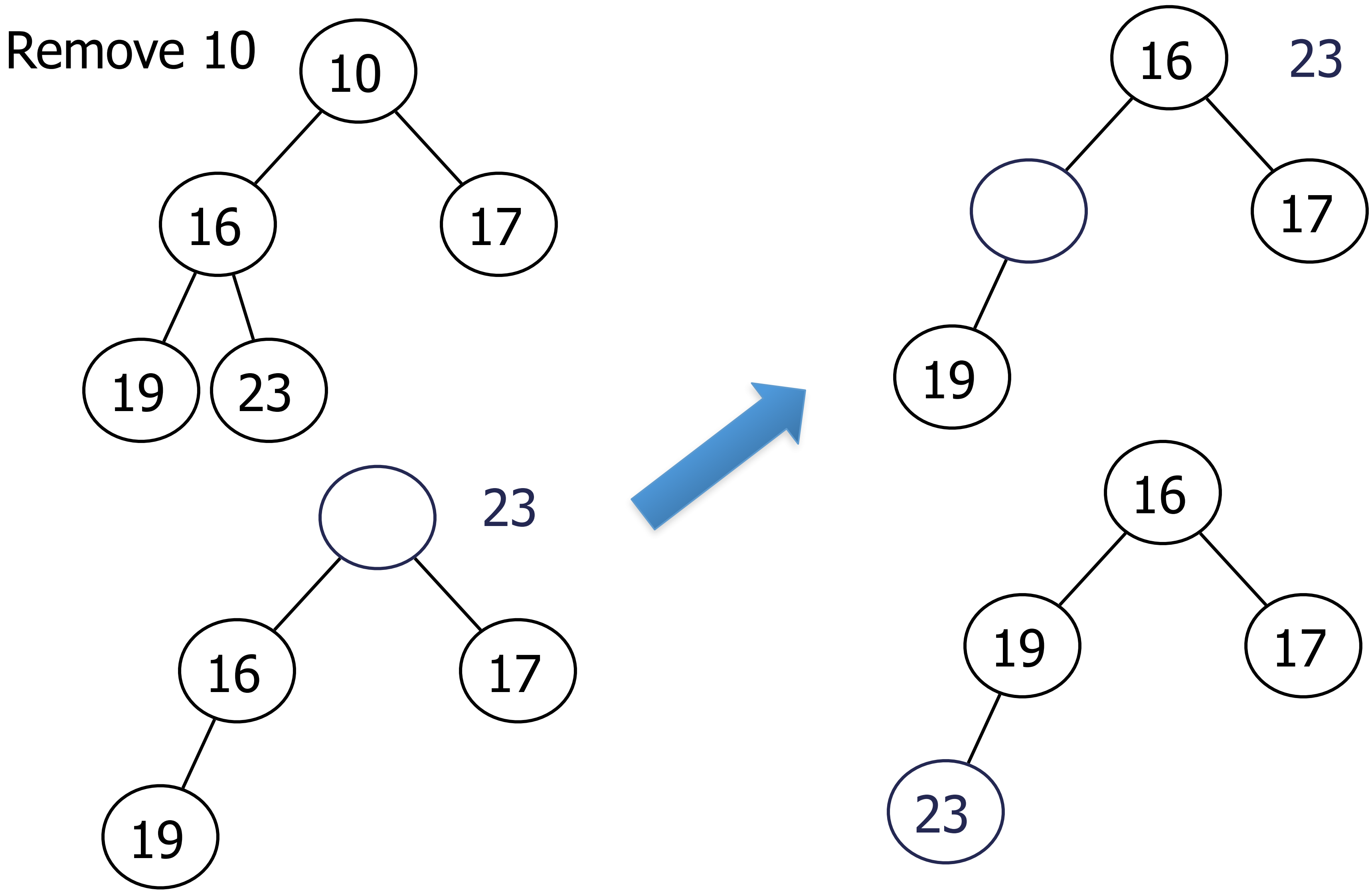
13, 16, 19, 14, 23, 17, 12

Insert 12



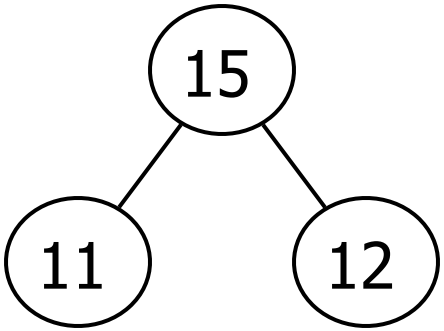


Exam 7 Min Heap - Remove

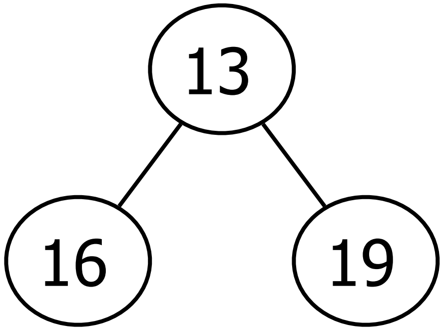


Heap Order Property

- Every root or sub-root has a higher priority than all of its children nodes.
- **Min Heap –**
  - For each node X, X.key > (X.parent).key
  - Smallest number is at the root
- **Max Heap –**
  - For each node X, X.key < (X.parent).key
  - Largest number is at the root



Max-heap

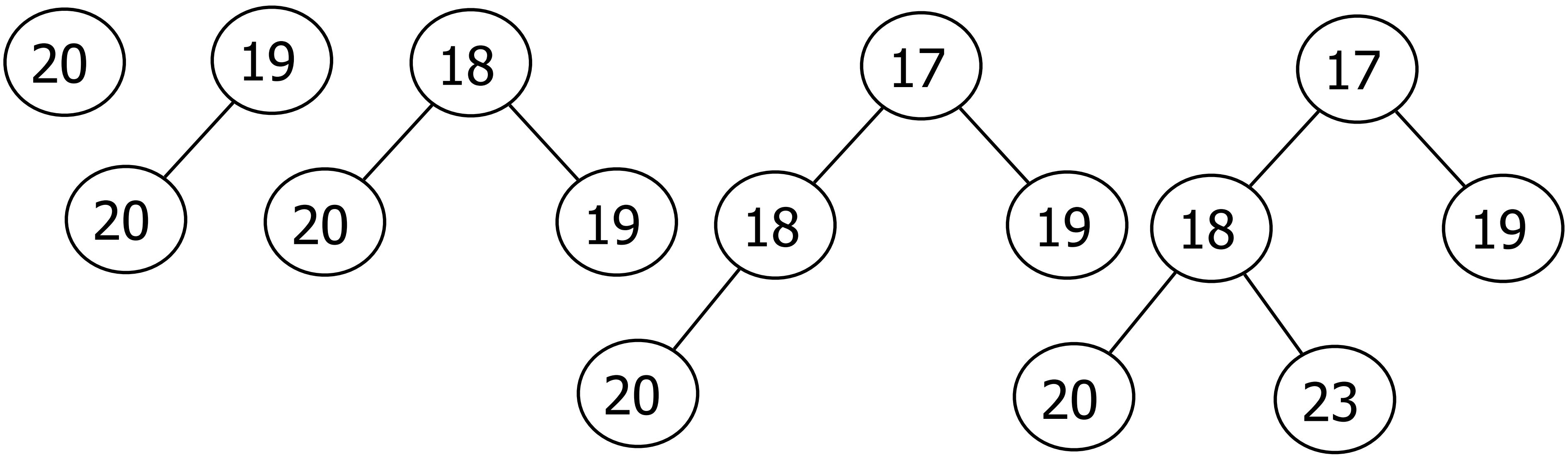


Min-heap

Exam 5 Q2:
Summer 17

Min-Heap Insert

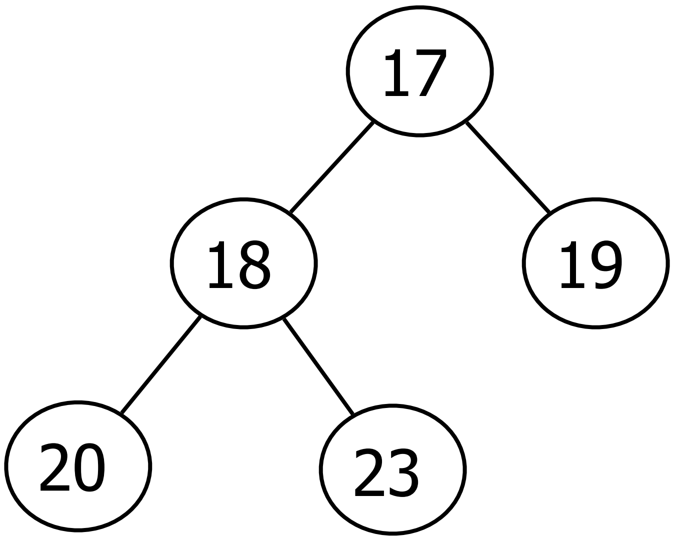
20, 19, 18, 17, 23, 4

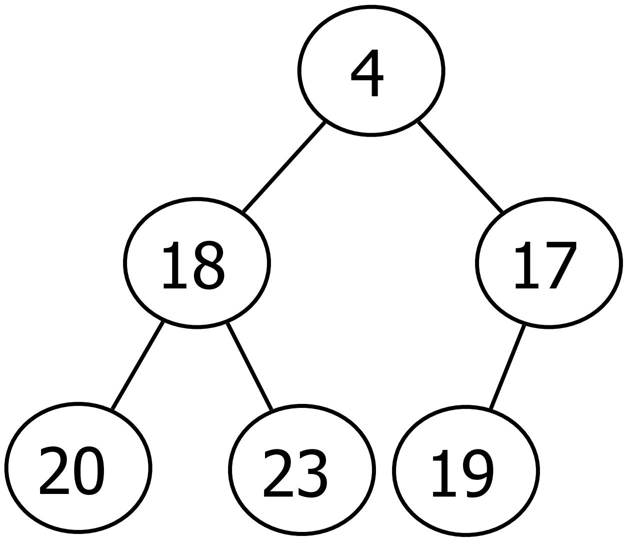


Exam 5 Q3:
Summer 17

Min-Heap Insert

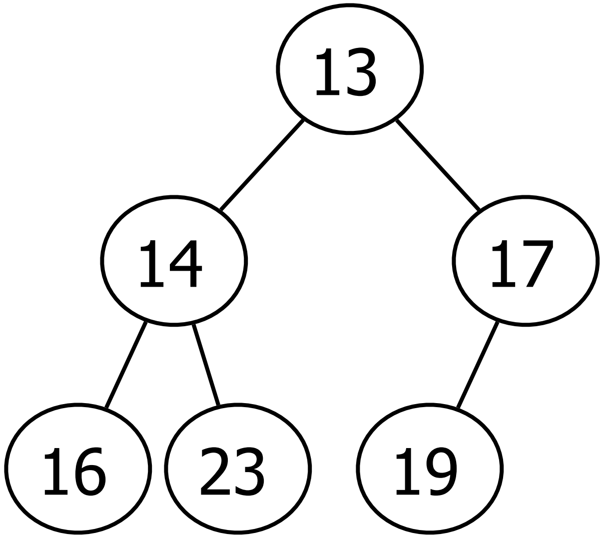
20, 19, 18, 17, 23, 4

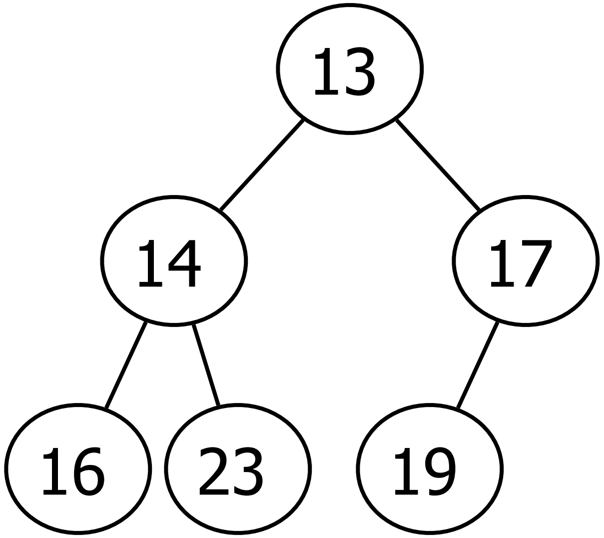




Exam 5 Q3:
Summer 17

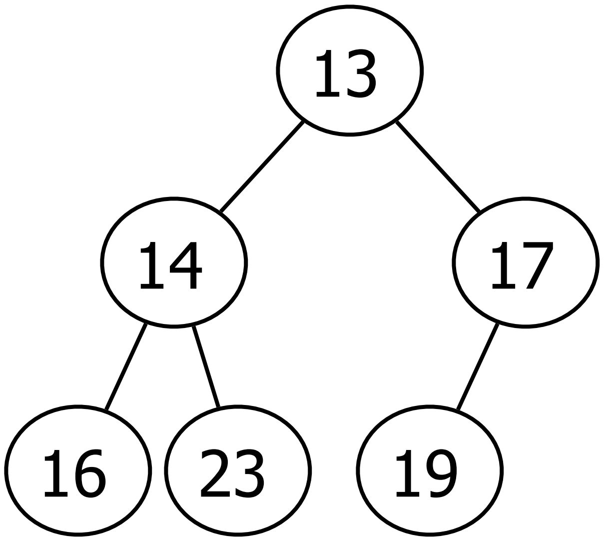
Binary Heap Insert





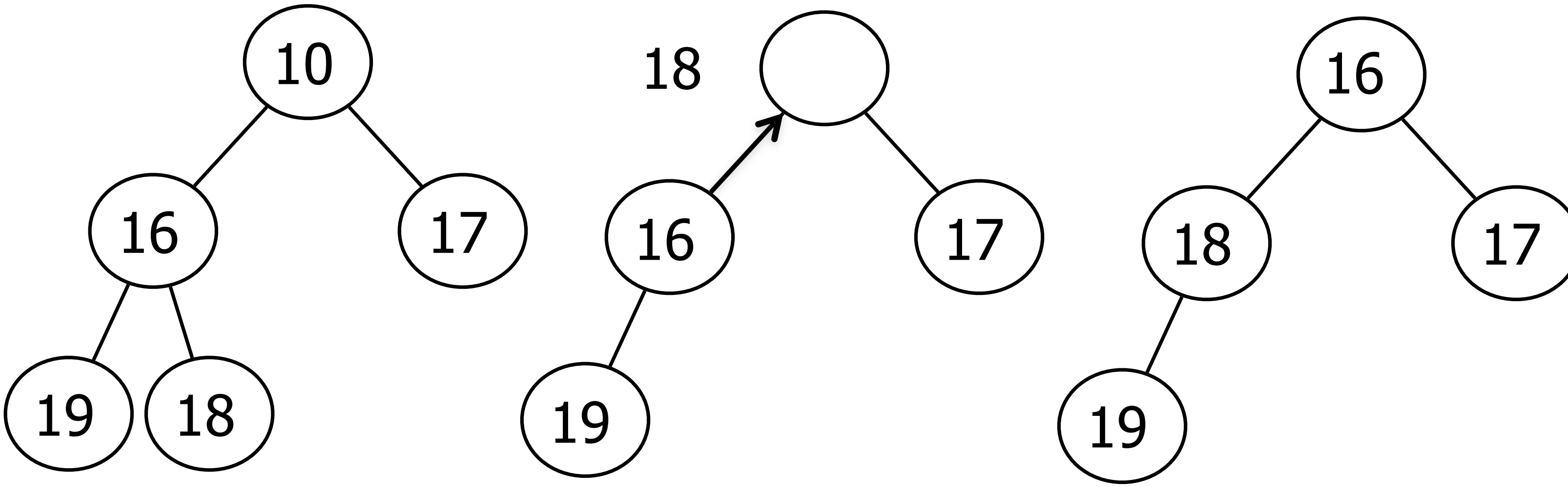
Exam 5 Q3:
Summer 17

Binary Heap Insert



Exam 5 Q3:
Summer 17

Binary Heap Remove



Exam 5 Q8:
Summer 17