Heaps
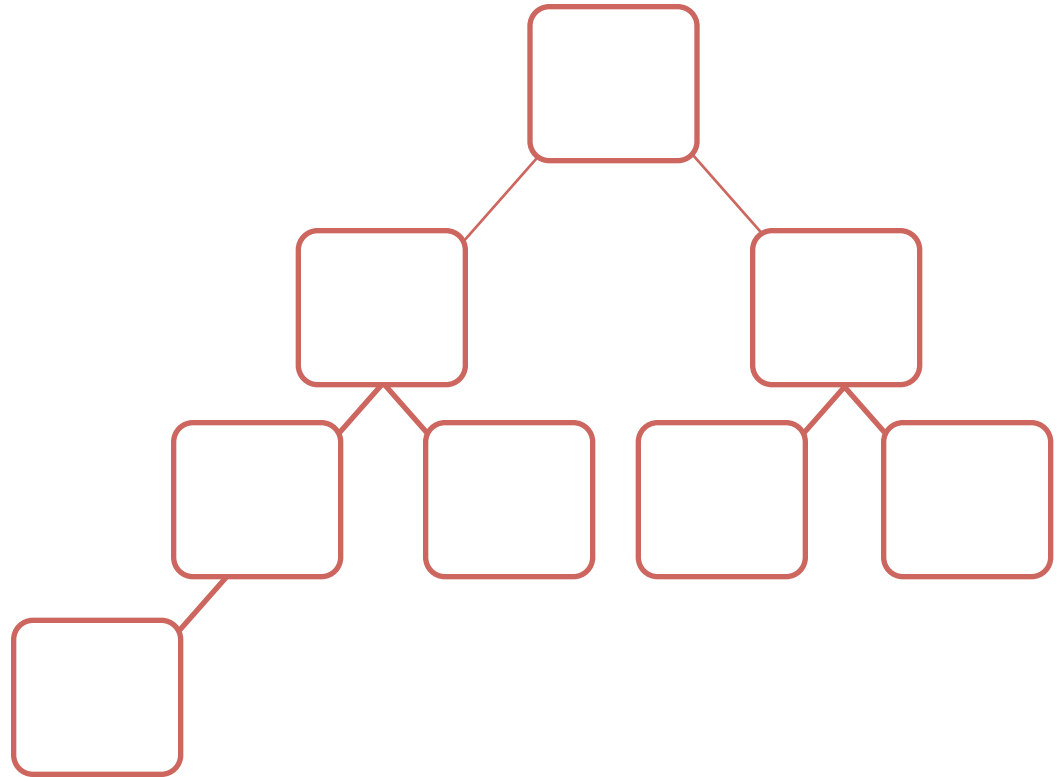
Read Ch10.1

Heaps – Complete Binary Tree
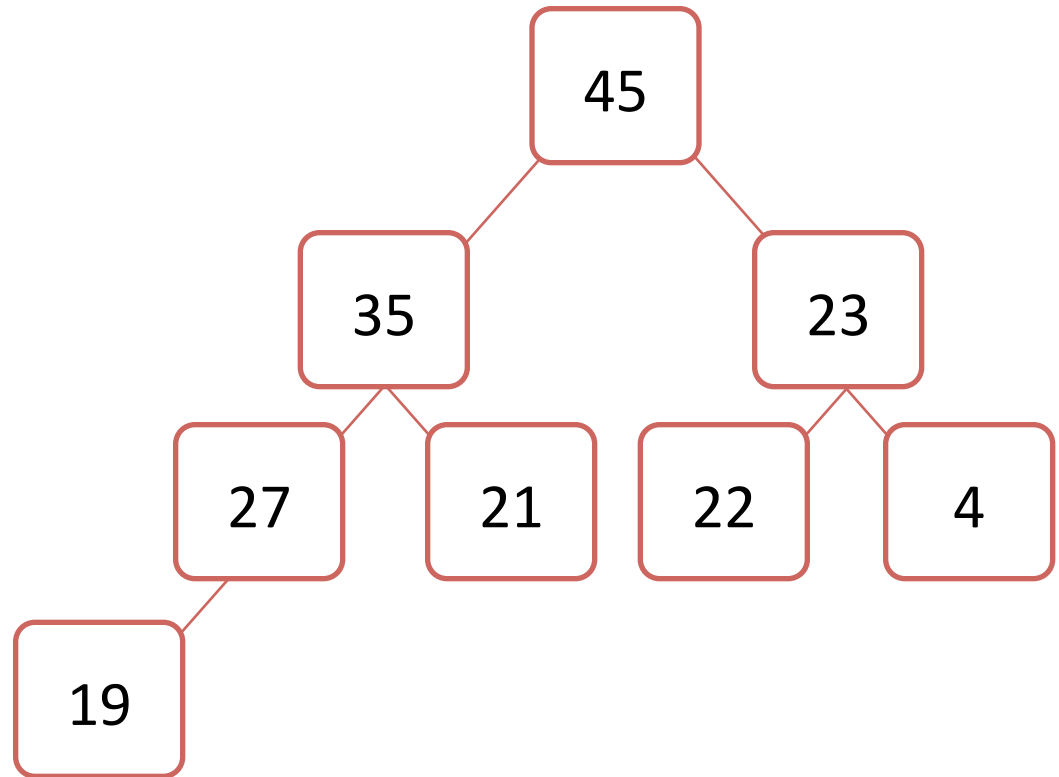
Complete binary tree.

- Level i has 2^i nodes (0≤i≤h-1)
- Level h fill this level from left to right.

Heaps -- Heap-order property

- For every node v other than the root, the key associated with v is smaller than or equal to the key associated with v's parent.

```
                45
               /  \
             35    23
            /  \   /  \
          27   21 22   4
         /
        19
```

Adding a Node to a Heap

1. Put the new node in the next available spot.

2. Push the new node upward, swapping with its parent until the new node reaches an acceptable location.
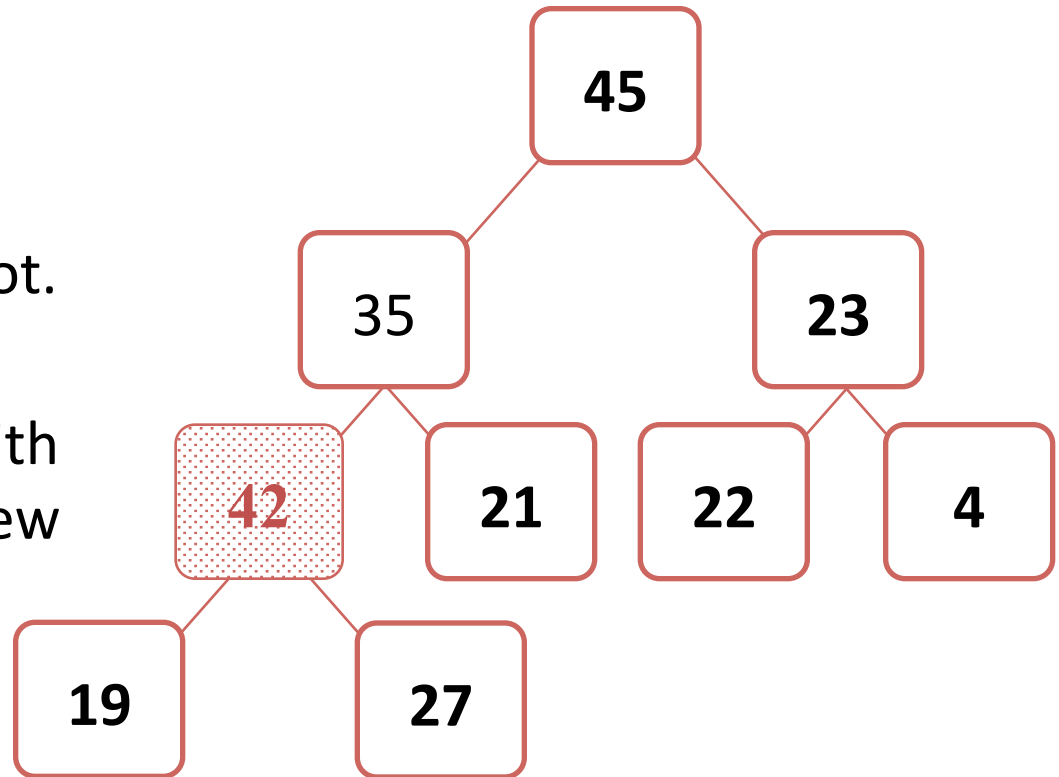
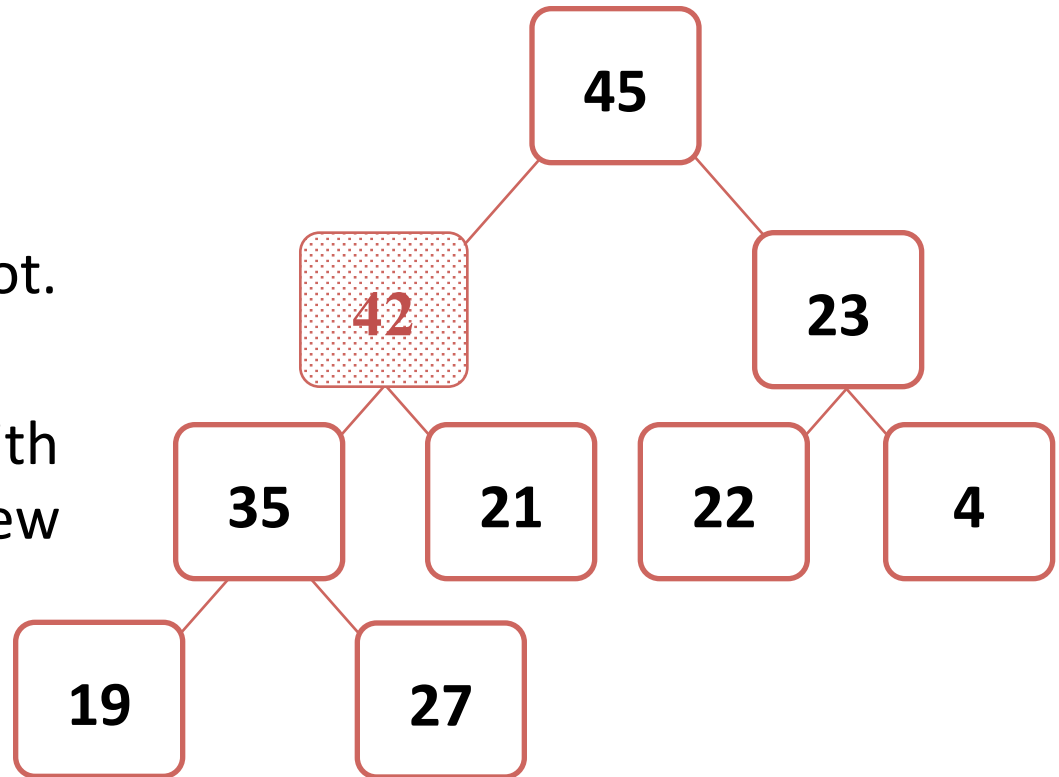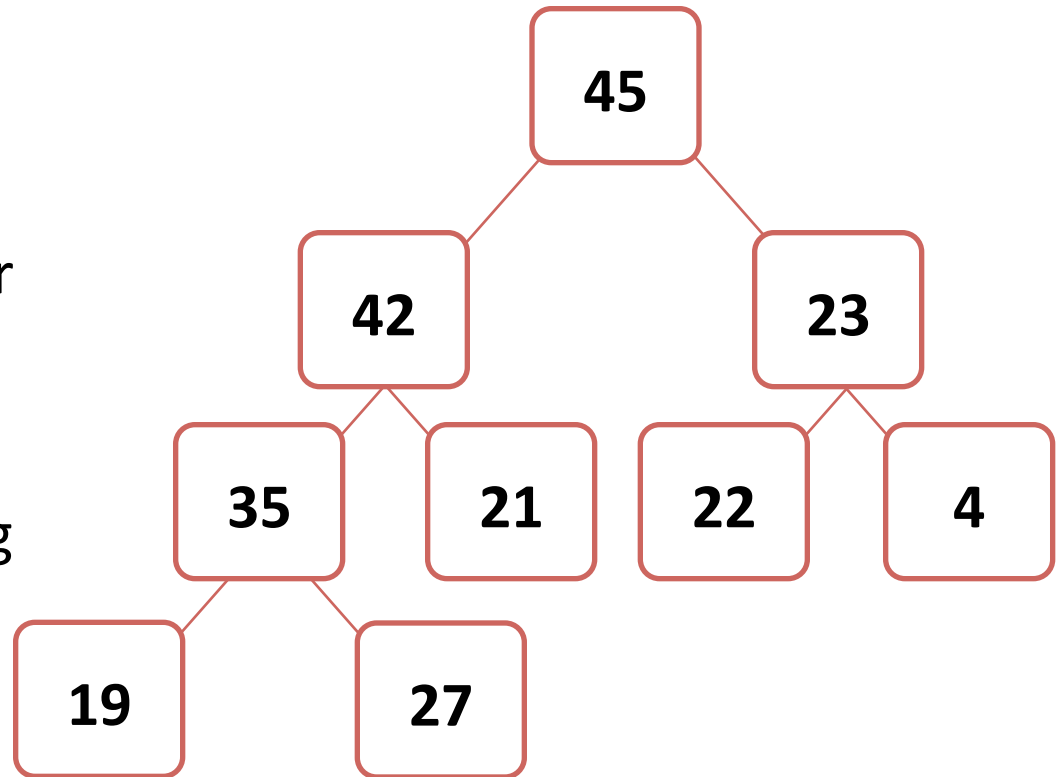# Adding a Node to a Heap

1. Put the new node in the next available spot.

2. Push the new node upward, swapping with its parent until the new node reaches an acceptable location.

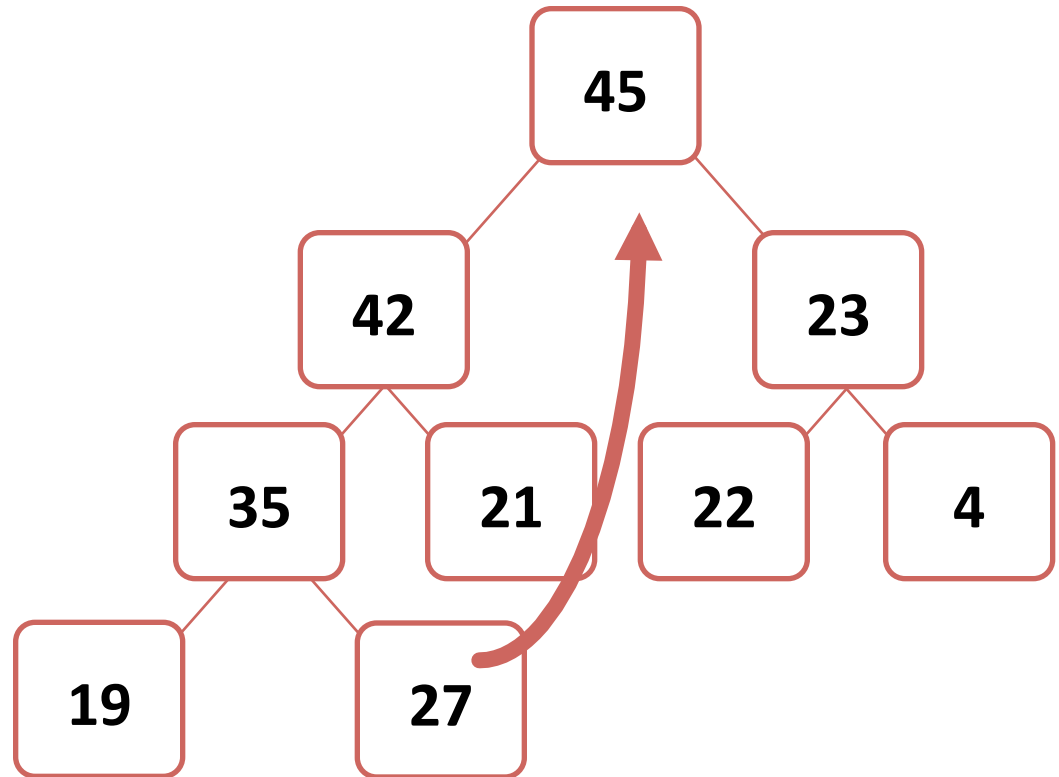# Adding a Node to a Heap

✔ The parent has a key that is >= new node, or

✔ The node reaches the root.

↗ The process of pushing the new node upward is called **reheapification upward**.

```
                45
              /    \
            42      23
           /  \    /  \
         35    21 22    4
        /  \
      19    27
```
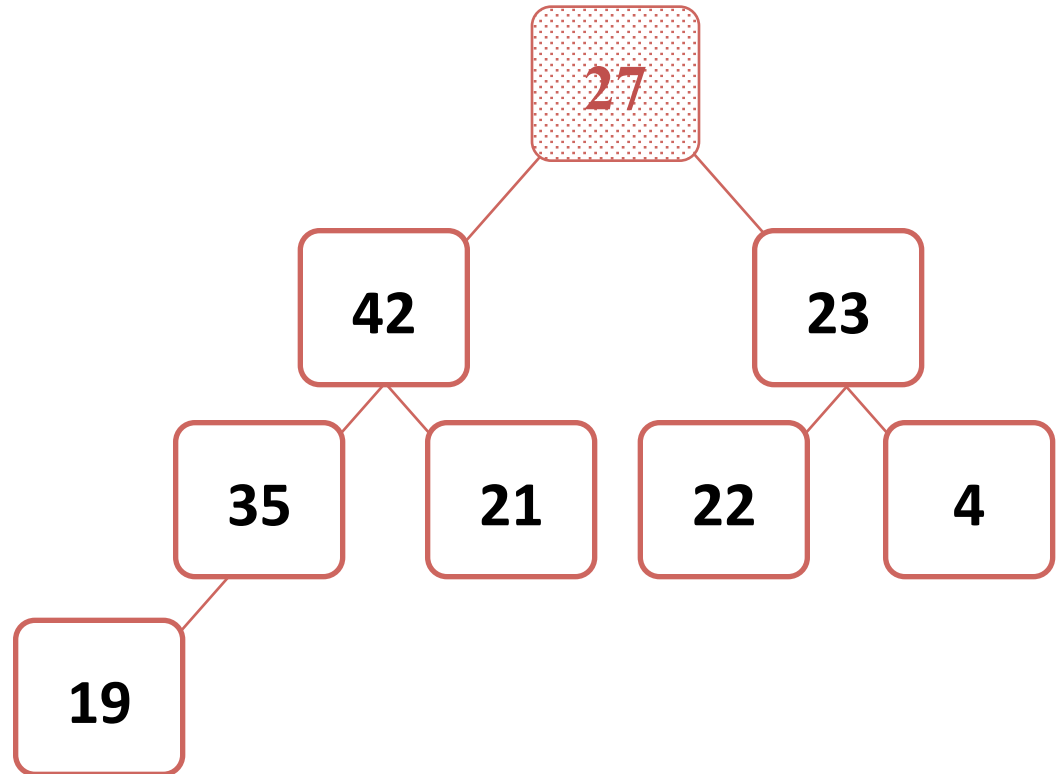
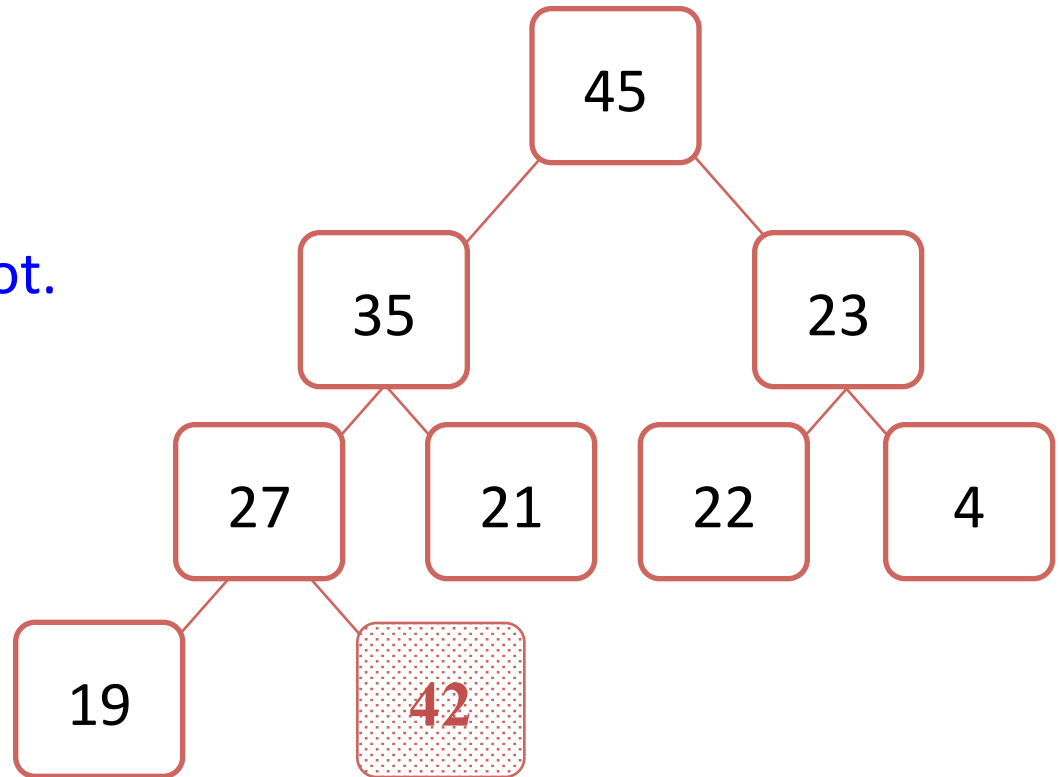# Removing the Top of a Heap

- Move the last node onto the root.

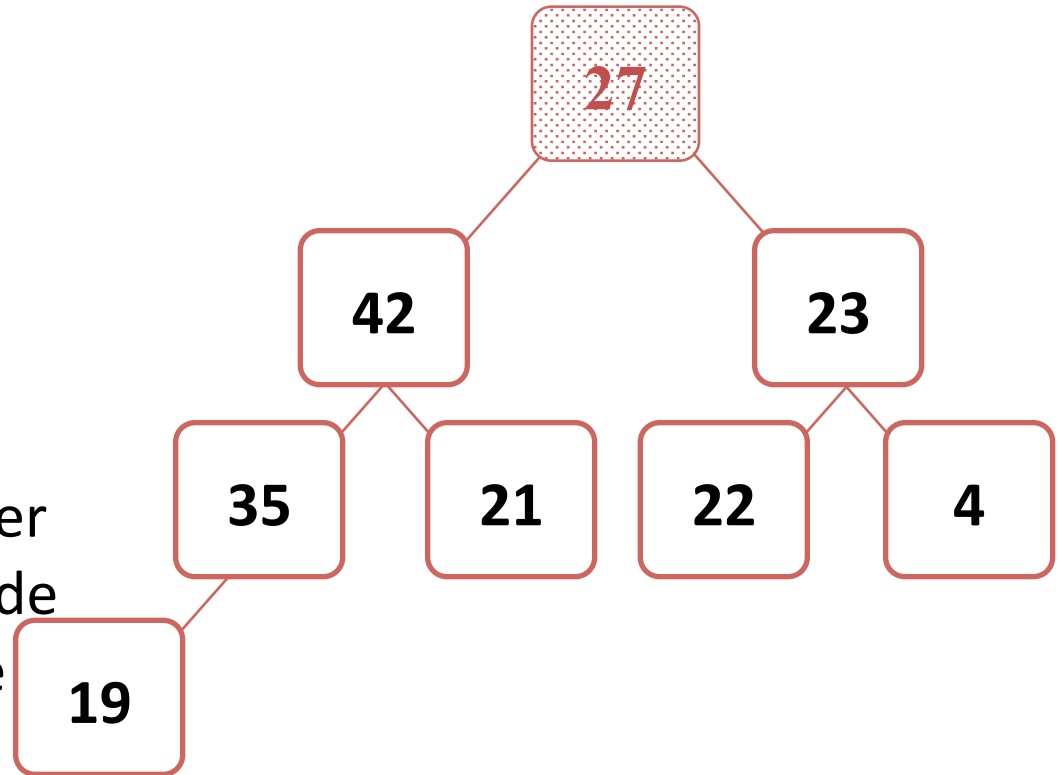# Removing the Top of a Heap

- Move the last node onto the root.

Adding a Node to a Heap

1. Put the new node in
   the next available spot.

45

35   23

27   21   22   4

19   **42**

Removing the Top of a Heap

- Move the last node onto the root.
- Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.

27

42

23

35

21

22

4

19

Removing the Top of a Heap

- Move the last node onto the root.

- Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.

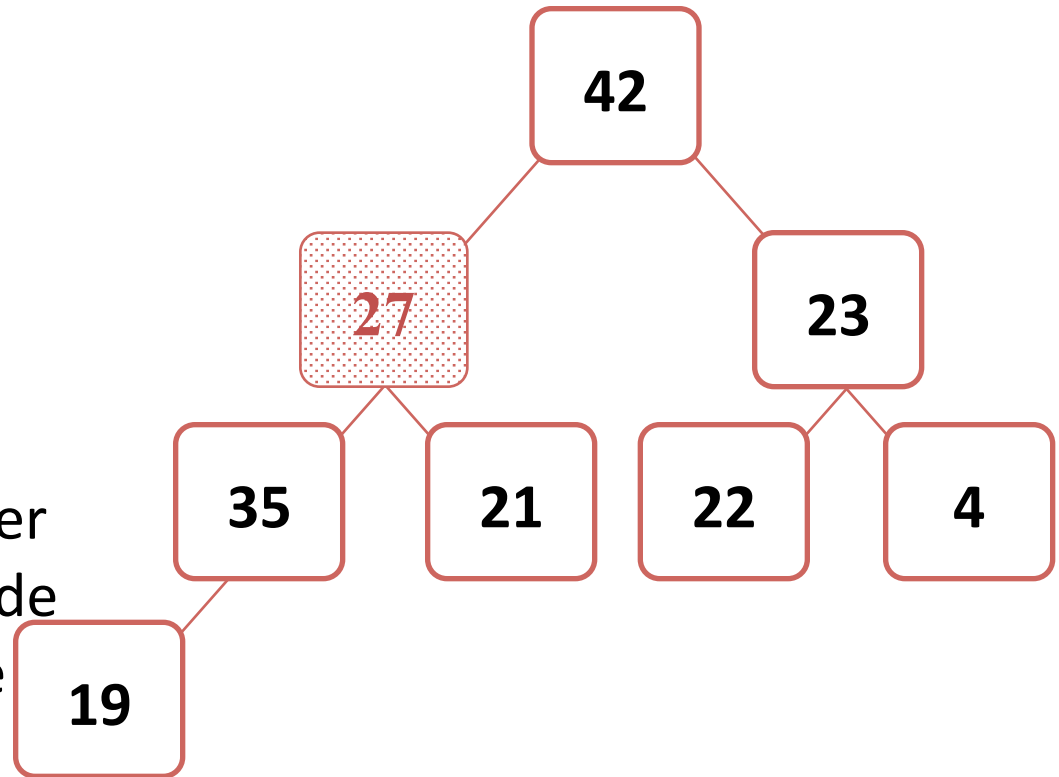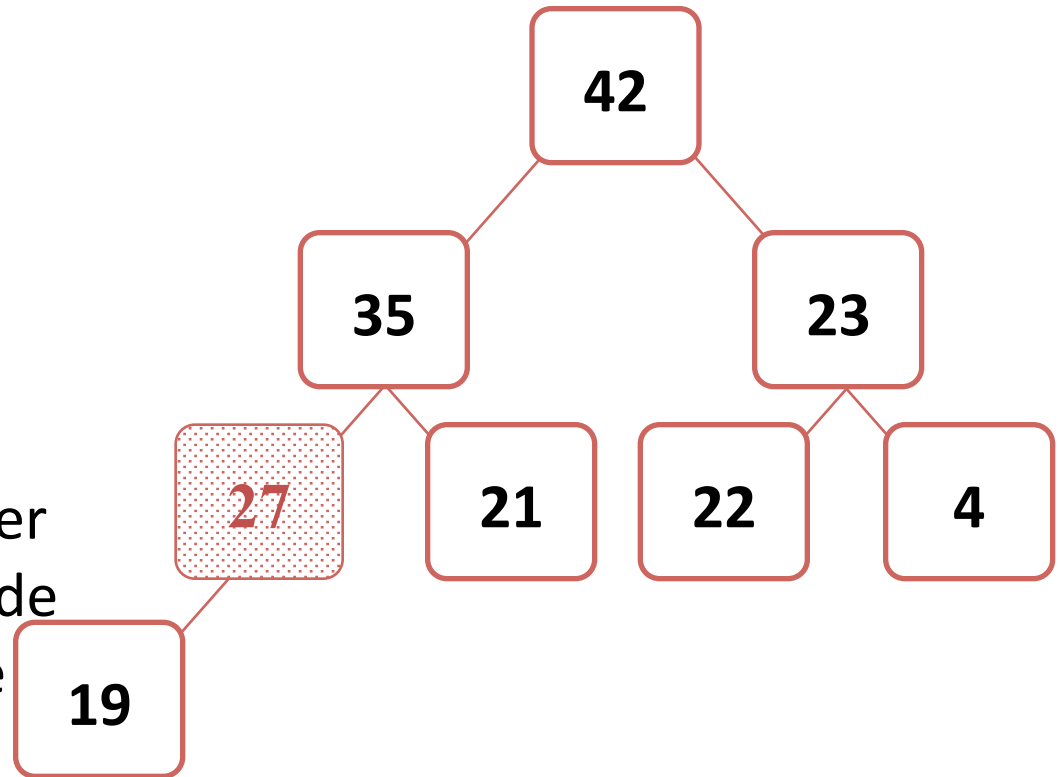# Removing the Top of a Heap

- Move the last node onto the root.

- Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.
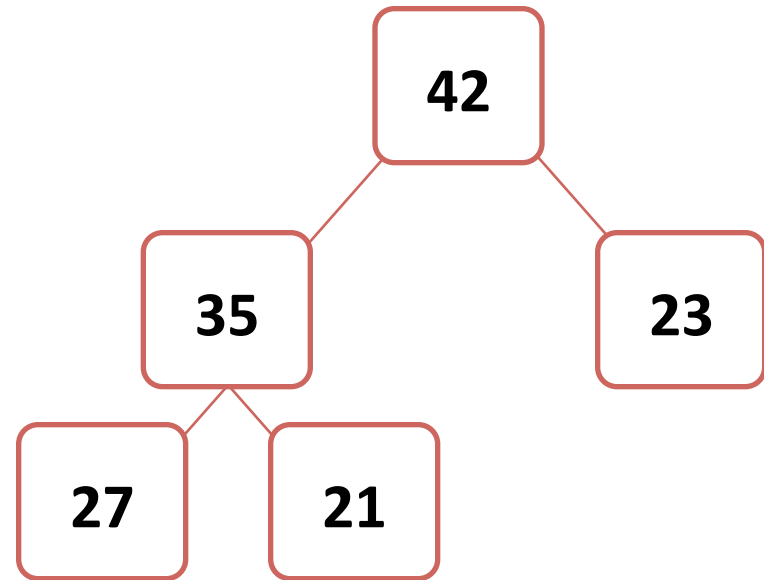
Removing the Top of a Heap

✔ The children all have keys <= the out-of-place node, or

✔ The node reaches the leaf.

↘ The process of pushing the new node downward is called **reheapification downward**.

42

35

23

27

21

22

4

19
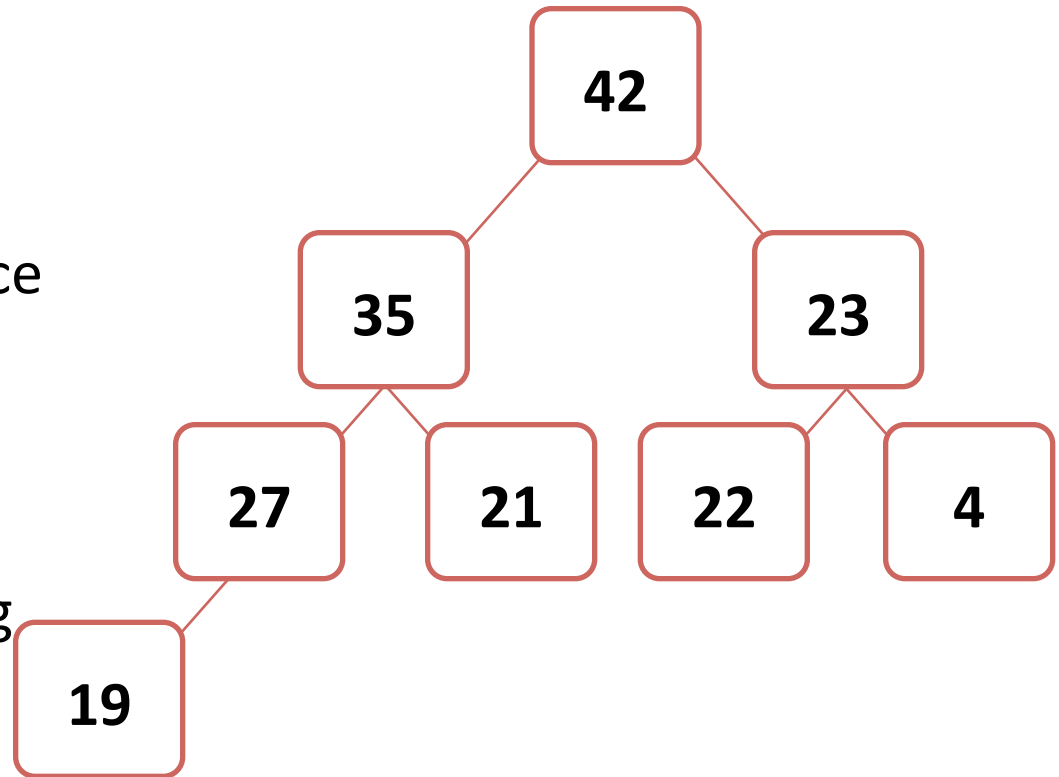
Implementing a Heap

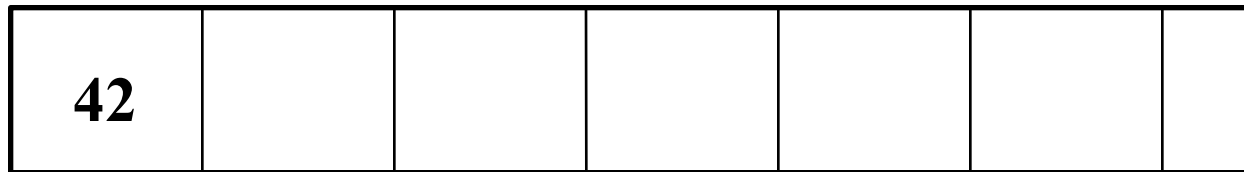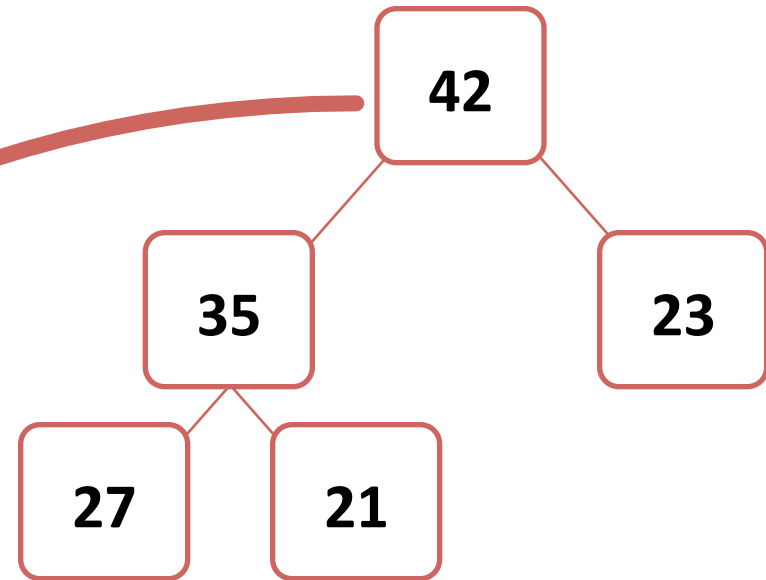- We will store the data from the nodes in a partially-filled array.

An array of data

Implementing a Heap

- Data from the root goes in the first location of the array.

42

35   23

27   21

42

An array of data

Implementing a Heap

- Data from the next row goes in the next two array locations.

42

35   23

27   21

| 42 | 35 | 23 | | | | |

An array of data

Implementing a Heap

- Data from the next row goes in the next two array locations.

42
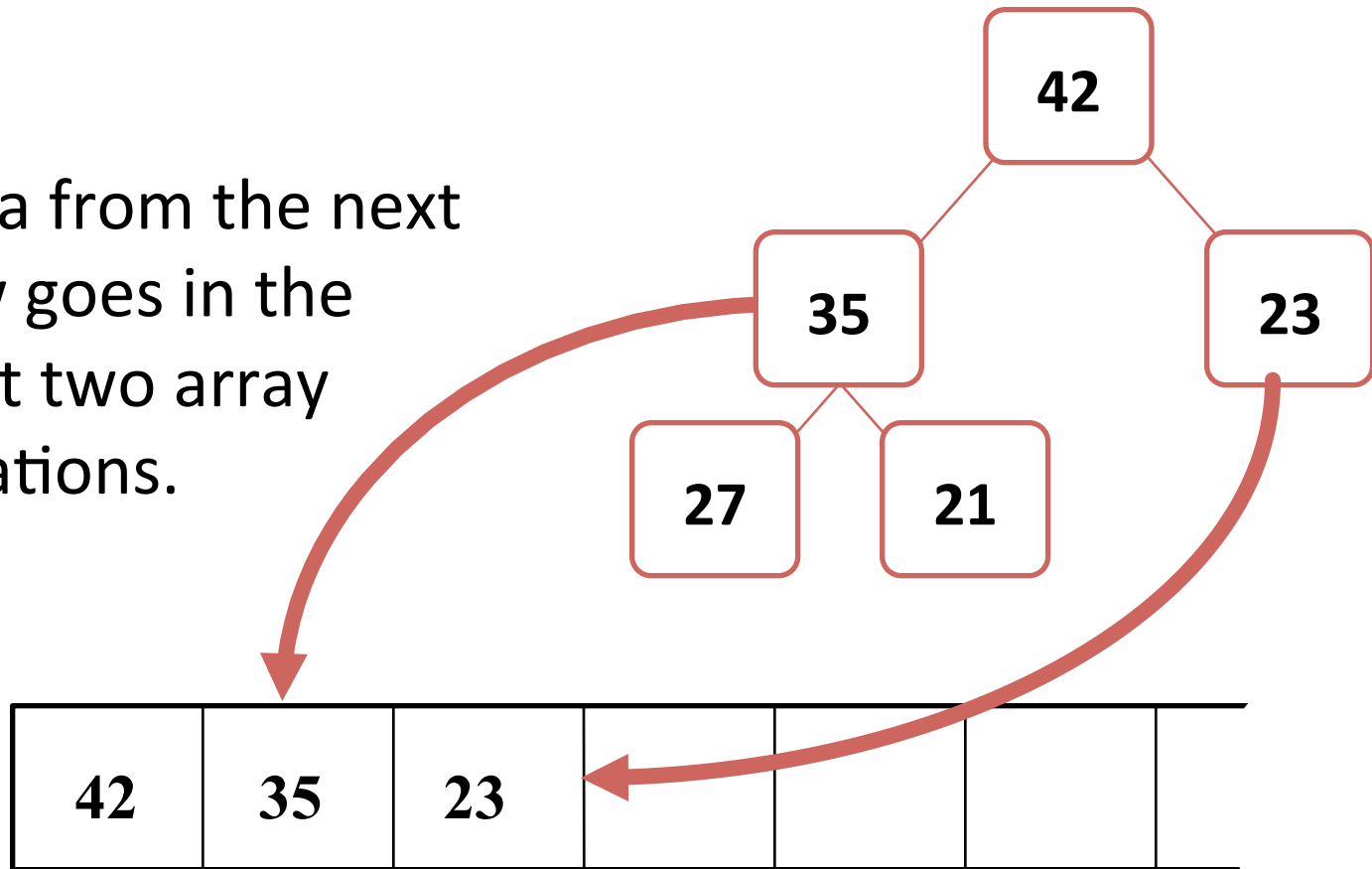
35   23
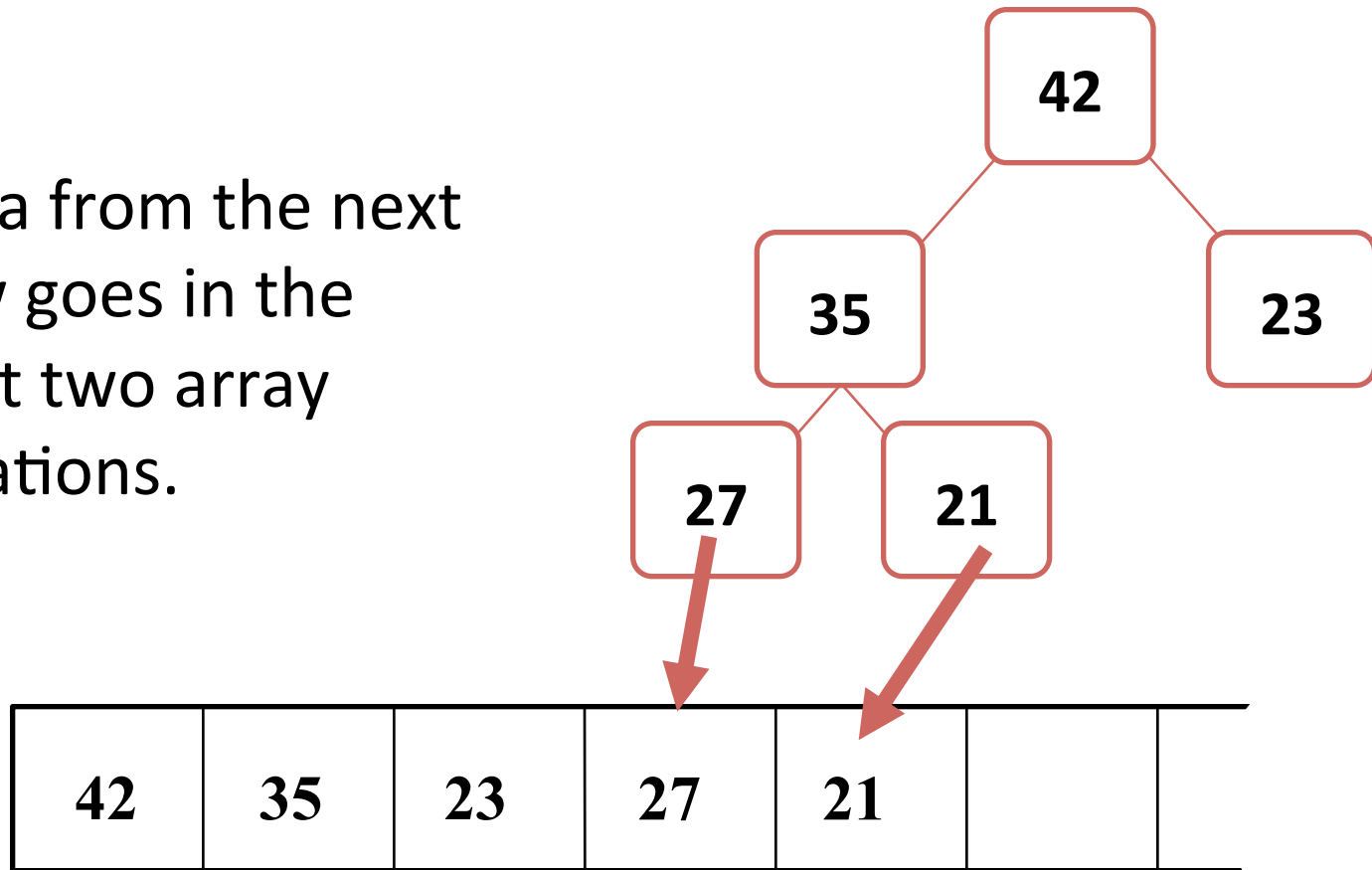
27   21

42	35	23	27	21		

An array of data

Implementing a Heap

- Data from the next row goes in the next two array locations.
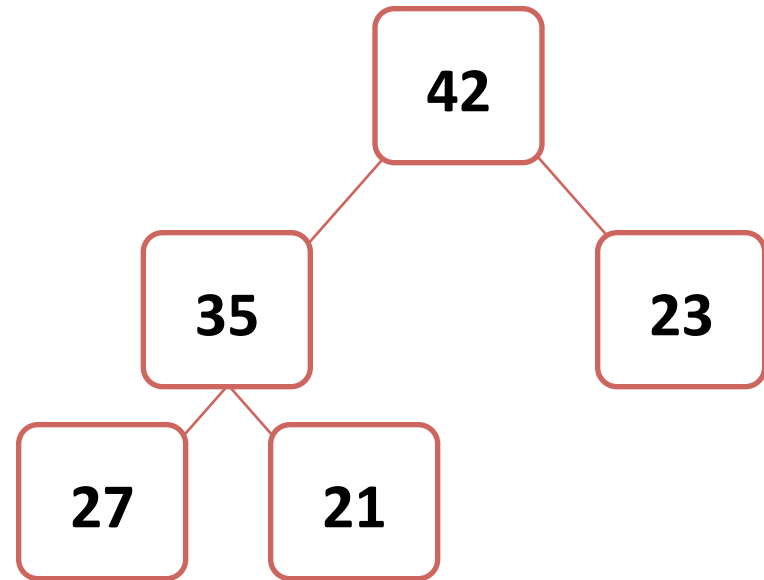
42	35	23	27	21		

An array of data

What is the next available spot?

Important Points about the Implementation

- Left child of [i] = [2i+1]
- Right child of [i] = [2i+2]

- Parent of [i]= [floor((i-1)/2)]

```
       42
      /  \
    35    23
   /  \
  27   21
```
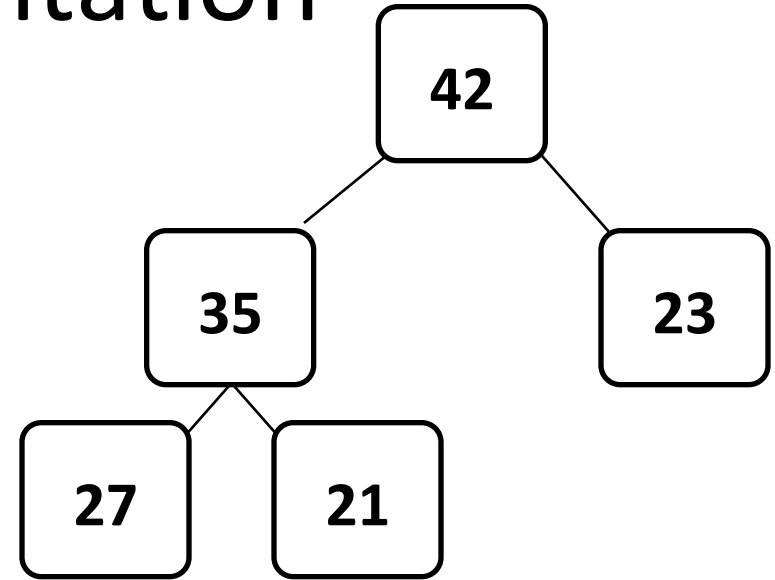
42	35	23	27	21		
[0]	[1]	[2]	[3]	[4]		

void add(int e)

- 1. add e as the last element to the array/ arrayList

- 2. Push up the element until the heap property is satisfied

  - reheapUpward(int pos)

reheapUpward(pos)

- //pos: the position of element that violates the heap
- Base case: pos<=0
- Recursive
- 1. get the position of the parent, parentPos
- 2. if array[pos]>array[parentPos]
  - 2.1 Swap array[pos] and array[parentPos]
  - 2.2 Recursively reheapUp(parentPos)

int remove()

- 1. the answer is the first element, array[0]
- 2. if array contains more than one element
  - Put the last element to array[0]
  - Remove the last element
  - reheapDownward(int pos=0)

Time complexity

- Top
- Add
- Remove

The height of a heap

- A heap storing n entries has height 1+h
  - Where h= floor (logn)
- Justification
  - For $0 \leq i \leq h-1$, each level has 2^i, the least number of nodes $(1+2+\ldots+2^{h-1})+1 = (2^h-1)+1 = 2^h$
  - The most number of nodes $(1+2+\ldots+2^{h-1})+2^h = 2^{h+1}-1$
  - Thus, $2^h \leq n \leq 2^{h+1}-1$
  - $\log(n+1)-1 \leq h \leq \log n$
  - h is an integer, thus h = floor (logn)

Heap

- Priority Queue
  - http://docs.oracle.com/javase/6/docs/api/
- What about implementing a heap using a binary tree?

Summary

- A heap is a complete binary tree, where the entry at each node is greater than or equal to the entries in its children.

- To add an entry to a heap, place the new entry at the next available spot, and perform a reheapification upward.

- To remove the biggest entry, move the last node onto the root, and perform a reheapification downward.