Welcome everyone!

- You can find these slides on GitHub here: https://github.com/WomenWhoCode/WWCodePython

- Some housekeeping rules:
  - Everyone will be muted throughout the webinar, but there will be opportunities for participation!
  - Please share your thoughts on the chat and/or ask questions in the Q&A.
  - The entire team is here today. Please reach out to us with any technical questions!

WELCOME

WOMEN WHO CODE

Women Who Code Python

Intro to Data Structures with Python: Ace the Technical Interview

Session #7: Heaps

WOMEN WHO CODE®

MEET YOUR TEAM

Rishika Singh
Track Lead

Jasmeen Rajpal
Evangelist

WOMEN WHO CODE

OUR MISSION

Inspiring women to excel in technology careers.

WOMEN WHO CODE

OUR VISION

A world where women are representative as technical executives, founders, VCs, board members and software engineers.

WOMEN WHO CODE

OUR TARGET

Engineers with two or more years of experience looking for support and resources to strengthen their influence and levelup in their careers.

WOMEN WHO CODE

CODE OF CONDUCT

WWCode is an inclusive community, dedicated to providing an empowering experience for everyone who participates in or supports our community, regardless of gender, gender identity and expression, sexual orientation, ability, physical appearance, body size, race, ethnicity, age, religion, socioeconomic status, caste, creed, political affiliation, or preferred programming language(s).

Our events are intended to inspire women to excel in technology careers, and anyone who is there for this purpose is welcome. We do not tolerate harassment of members in any form. Our **Code of Conduct** applies to all WWCode events and online communities.

Read the full version and access our incident report form at **womenwhocode.com/codeofconduct**

WOMEN WHO CODE

230,000
Members

70 networks in 20 countries
Members in 97+ countries
10K+ events
$1025 daily Conference tickets
$2M Scholarships
Access to jobs + resources
Infinite connections

WOMEN WHO CODE

OUR MOVEMENT

As the world changes, we can be a connecting force that creates a sense of belonging while the world is being asked to isolate.

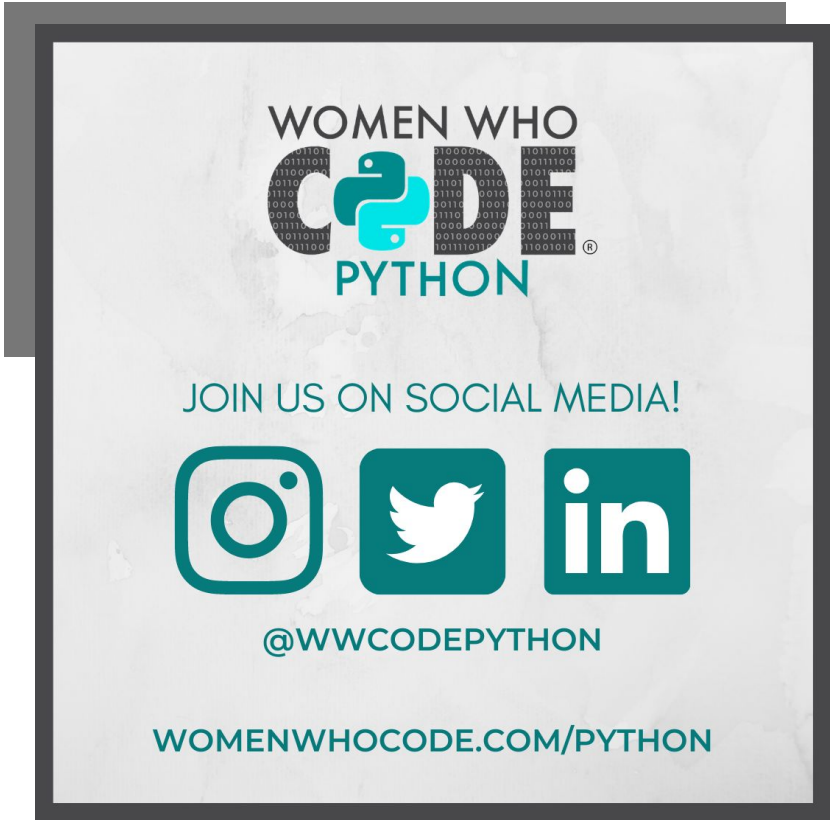
WOMEN WHO CODE

Upcoming Events

FRI 11 JUN	**CONNECT REIMAGINE Virtual Conference** *Featured* Online | 8:00 AM – 1:00 PM PDT (UTC-0700)	Register
THU 24 JUN	**✨Intro to Data Structures with Python: Ace the Technical Interview (Session #8: Graphs)✨** *Featured* Online | Python | 5:00 PM – 6:30 PM PDT (UTC-0700)	Register
TUE 29 JUN	**✨Ask Me Anything with Jane Street Software Engineers✨** *Featured* Online | Python | 8:00 AM – 9:00 AM PDT (UTC-0700)	Register
THU 01 JUL	**✨Intro to Data Structures with Python: Ace the Technical Interview (Session #9: Maps & Hash Tables)✨** *Featured* Online | Python | 5:00 PM – 6:30 PM PDT (UTC-0700)	Register

WOMEN WHO CODE PYTHON

Stay Connected



Nayeon Shin

Sophomore CS & New Media Art Major | WWCode Python Track General Volunteer

Today's Agenda

1. Queues & Trees Review
2. Priority Queue
3. Heap
    a. Binary Heap
        i. Structure Property
        ii. Order Property
        iii. Heap Operations
        iv. Implementation
4. Q&A
5. Live Coding
6. Resources

Queues & Trees Review

Queues

- On the contrary to stacks, a queue is open at both ends.
  - One end (Rear): Insertion (Enqueue)
  - The other (Front): Deletion (Dequeue)
- First-In-First-Out (FIFO)

Trees

- Two definitions:
  1. Nodes and edges

     A tree consists of a set of nodes and a set of edges that connect pairs of nodes.

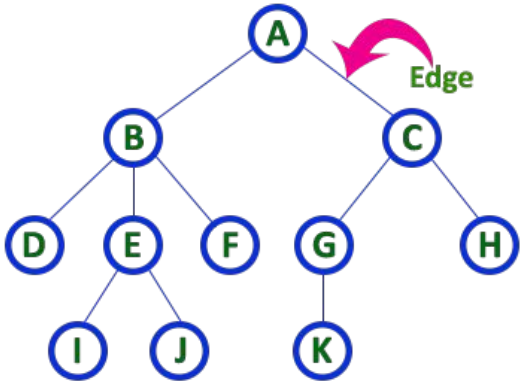
  2. Recursive

     A tree is either empty or consists of a root and zero or more subtrees, each of which is also a tree.

Trees

- Two definitions:
  1. Nodes and edges

     A tree consists of a set of nodes and a set of edges that connect pairs of nodes.



Trees

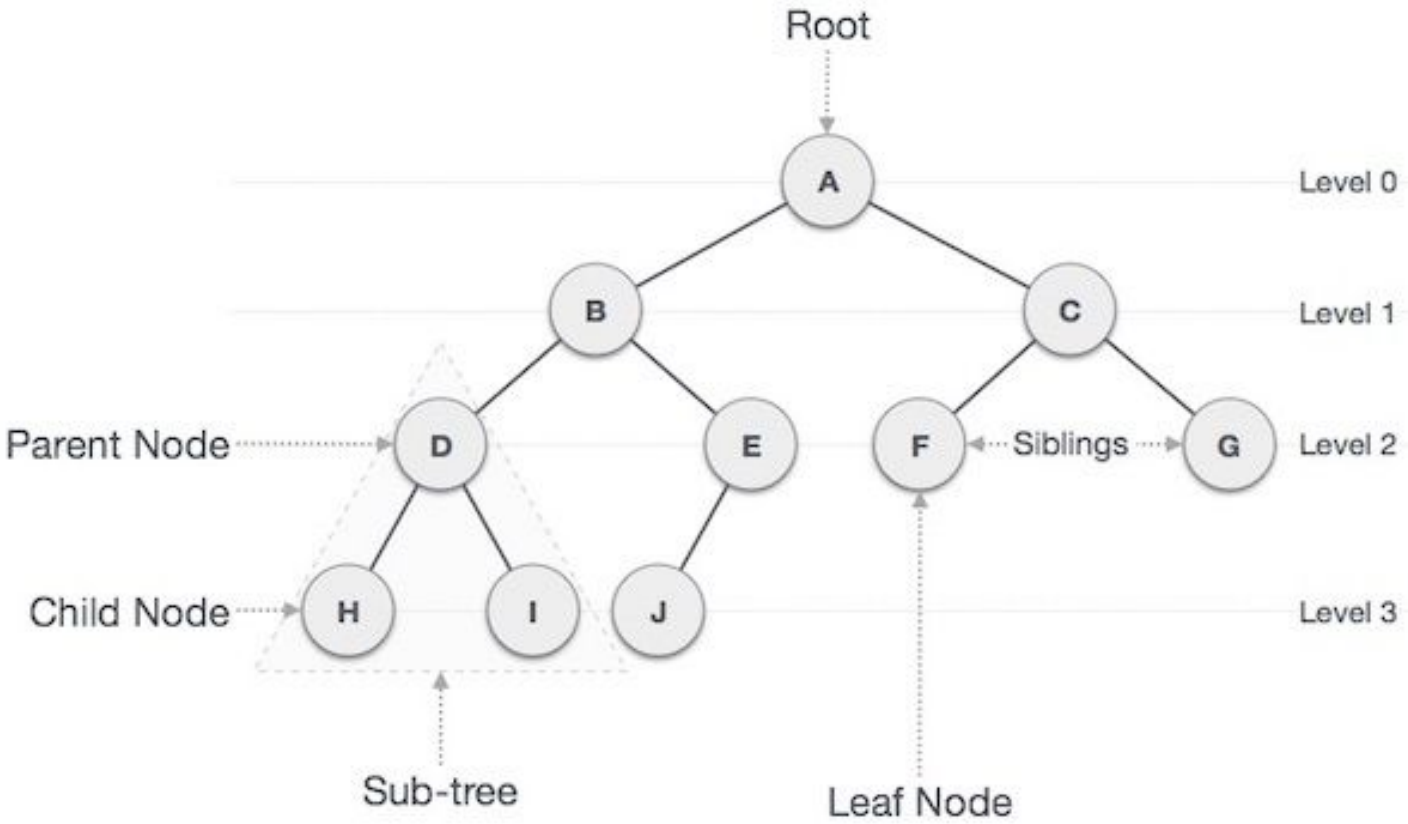
- Two definitions:
  1. Nodes and edges

     A tree consists of a set of nodes and a set of edges that connect pairs of nodes.

  2. Recursive

     A tree is either empty or consists of a root and zero or more subtrees, each of which is also a tree.

Trees


Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Child Node
Siblings
Sub-tree
Leaf Node

Priority Queue

Priority Queue

- Variation of a queue
- “Priority” queue:
  - Order of queue items is determined by their “priority”

Priority Queue

- Each entry: Priority (key) + Data value
- Data remove → Element with the highest priority
  - Priority: Any kind of comparable info

Priority Queue

- Main methods:
  - `insert(new_item)` → Insert a new item to the queue
  - `remove_max()` → Remove & return entry with largest key
    - Return `None` (Null) if empty queue
  - `get_max()` → Return entry with largest key value (leave it on queue)
  - `get_size()` → Return # of queue elements
  - `is_empty()`
- Lowest value as priority → `remove_min()` & `min()`

Implementation of a Priority Queue?

- Priority queue: Queue that has a priority order
  - Queue + Ordering?
    1. Implement PQ with unsorted lists

       → Expensive `max()` & `remove_max()`

    2. Implement PQ with sorted lists

       → Expensive `insert(k, v)`

Is there a better way? YES!

Heap

Binary Heap

- Two main properties
  1. Complete binary tree (Structure property)
  2. Ordering property

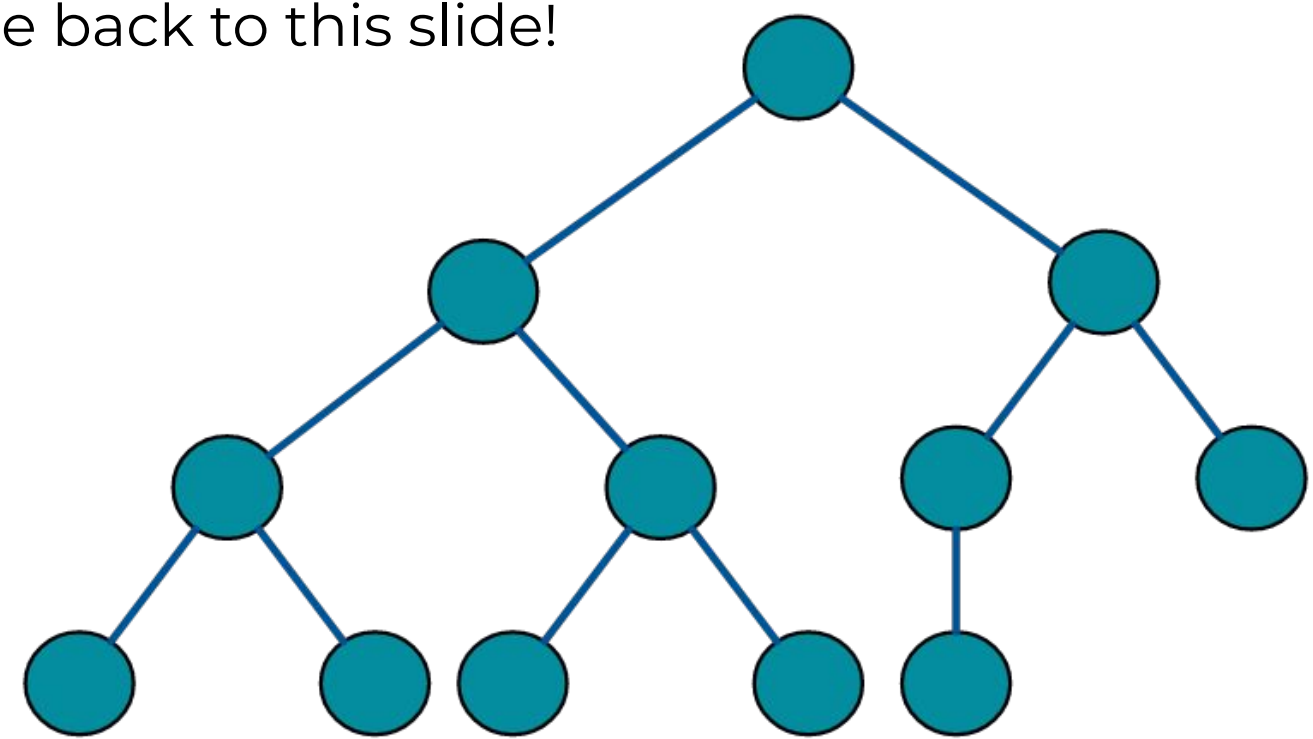
Binary Heap - Structure Property

- Two main properties
  1. Complete binary tree (Structure property)
     - For efficiency, take advantage of binary tree's logarithmic performance
     - For logarithmic performance, keep our tree balanced
       - ➢ Balanced binary tree:

         Roughly the same # of nodes in the left & right

Binary Heap - Structure Property

Will come back to this slide!



Binary Heap - Structure Property

- Two main properties
  1. Complete binary tree
     - Binary tree
     - Completely filled on each level except possibly lowest level
     - Lowest level: Filled from left to right
     - Any empty spots: At rightmost side of lowest level

Go back to previous slide!

Binary Heap - Order Property

- Two main properties

2. Order property
   - Min-heap or max-heap
   - Consider max-heap:

     For every position ***p*** other than the root,

     ***p* <= key of *p*'s parent**

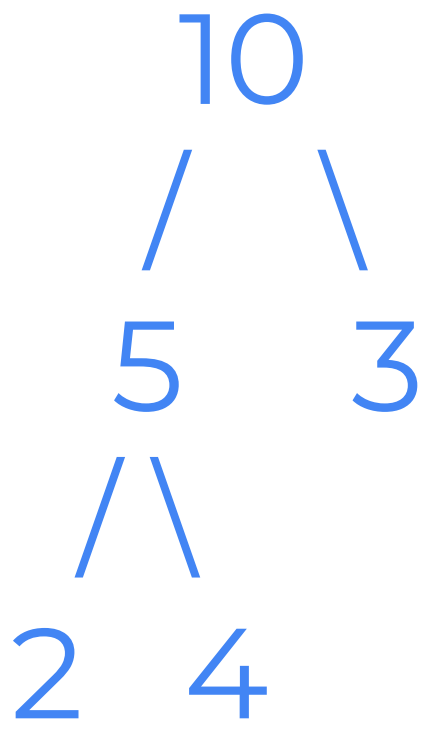
1) No relative ordering between siblings
2) Largest element: At the root
3) Along any path from root to leaf, non-increasing order

Binary Heap - Heap Operations

- Adding a new node in a max heap
  1. Add a new node at leaf
  2. while (node > parent) and (node is not root):

     Swap node with parent

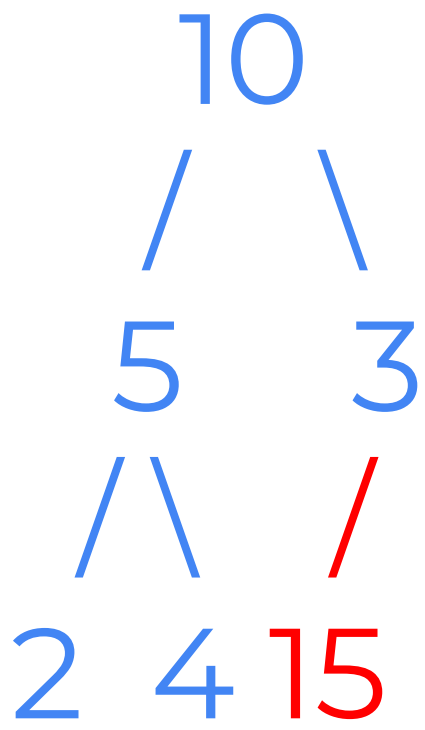
Binary Heap - Heap Operations: Add



New element to be inserted: 15

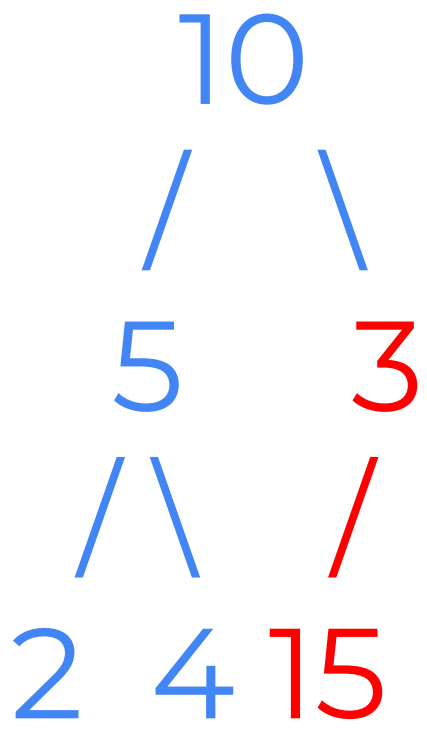
Step 1: Insert the new element at the end

Binary Heap - Heap Operations: Add



Step 1: Insert the new element at the end

Binary Heap - Heap Operations: Add



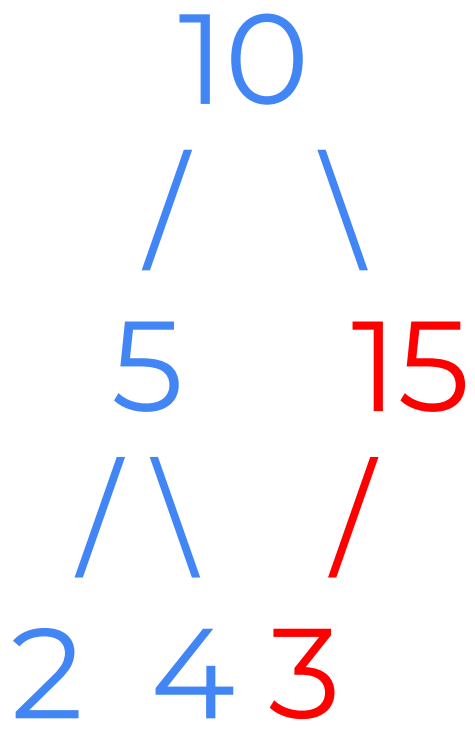
Step 2: "Heapify" the new element from the bottom.

- Heapify:
- while (node > parent) and

  (node is not root):

  Swap node with parent

Binary Heap - Heap Operations: Add

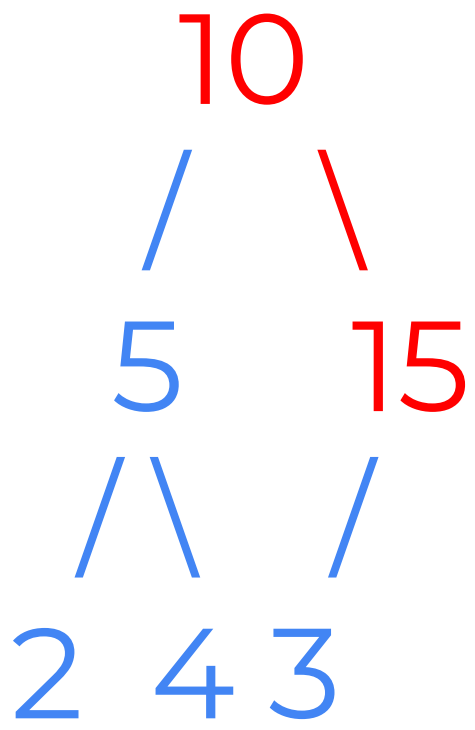


Heapify:

- while (node > parent) and (node is not root):
  Swap node with parent

15 > 3 → Swap those two!

Binary Heap - Heap Operations: Add

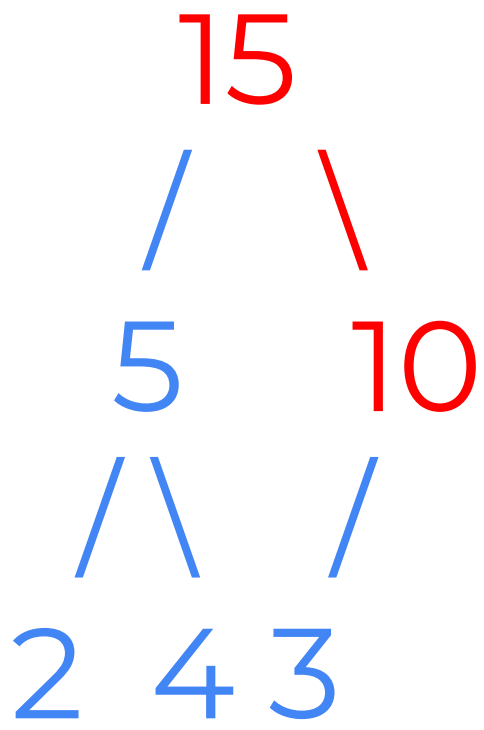


Heapify:

- while (node > parent) and (node is not root):
  Swap node with parent

15 > 10 → Swap again!

Binary Heap - Heap Operations: Add

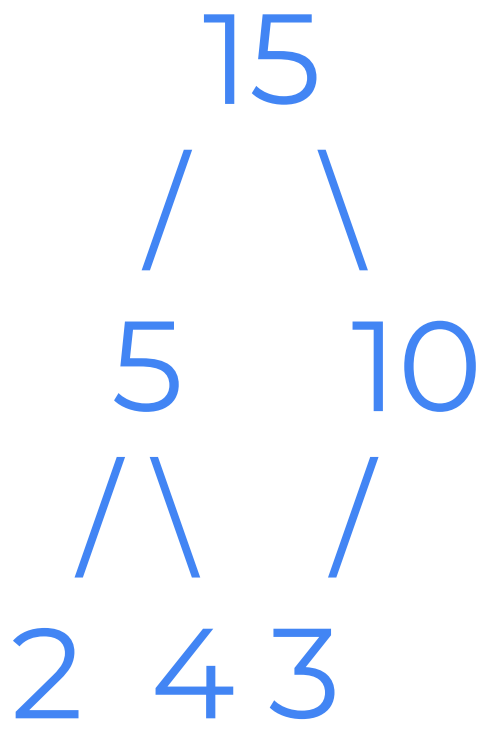


Heapify:

- while (node > parent) and (node is not root):
  Swap node with parent

15 is root; done.

Binary Heap - Heap Operations: Add



← FINAL HEAP

Heapify:

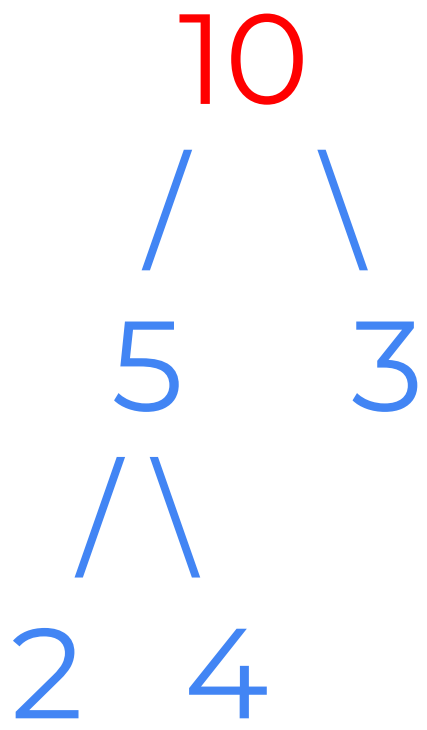
- while (node > parent) and

  (node is not root):

  Swap node with parent

Notice that the root is the largest.

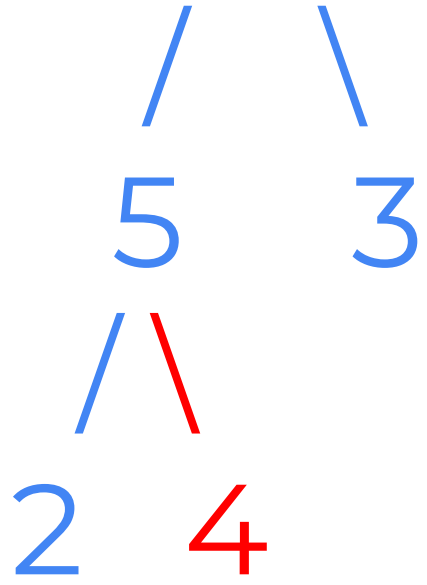
Binary Heap - Heap Operations: Remove



Element to be deleted: 10

Step 1: Remove the root

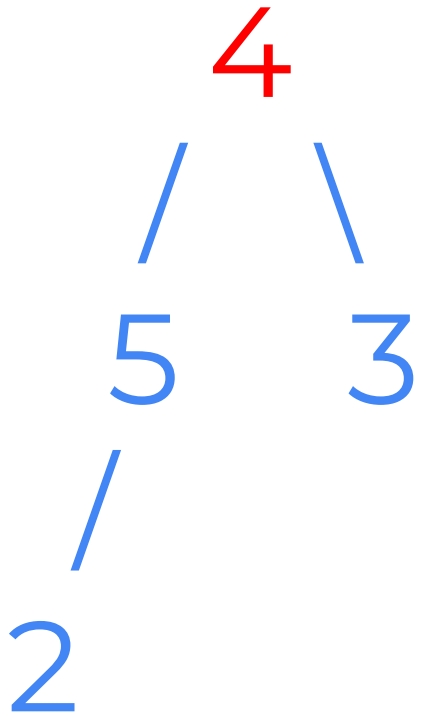
Binary Heap - Heap Operations: Remove



Element to be deleted: 10

Step 2: Make rightmost leaf the root

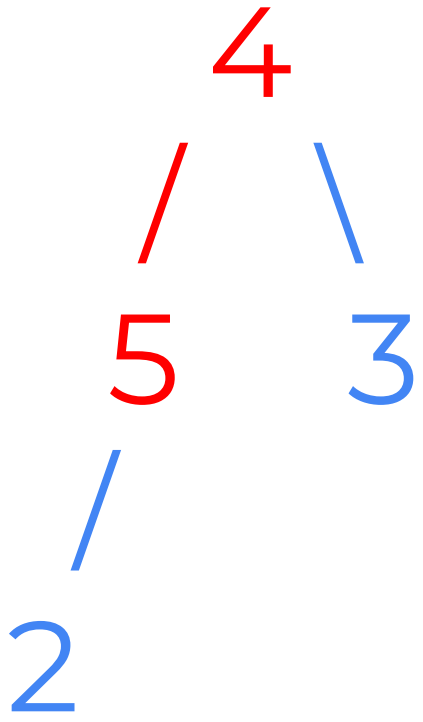
Binary Heap - Heap Operations: Remove



Element to be deleted: 10

Step 2: Make rightmost leaf the root

Binary Heap - Heap Operations: Remove



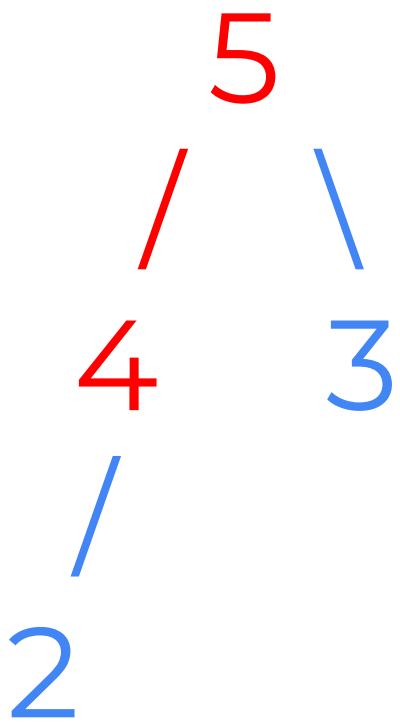
Element to be deleted: 10

Heapify:

- while (node > parent) and (node is not root):
  Swap node with parent

Step 3: Heapify the root

Binary Heap - Heap Operations: Remove



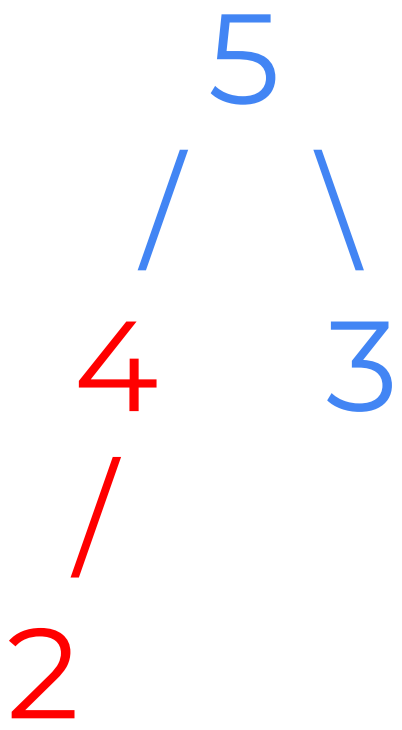
Element to be deleted: 10

Heapify:

- while (node > parent) and
  (node is not root):
  Swap node with parent

Step 3: Heapify the root

Binary Heap - Heap Operations: Remove



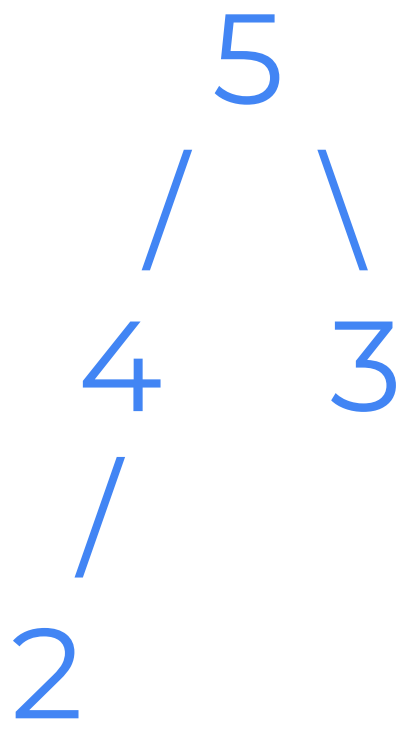
Element to be deleted: 10

Heapify:

- while (node > parent) and (node is not root):
  Swap node with parent

Step 3: Heapify the root

Binary Heap - Heap Operations: Remove



Element that has been deleted: 10

Done.

Binary Heap - Implementation

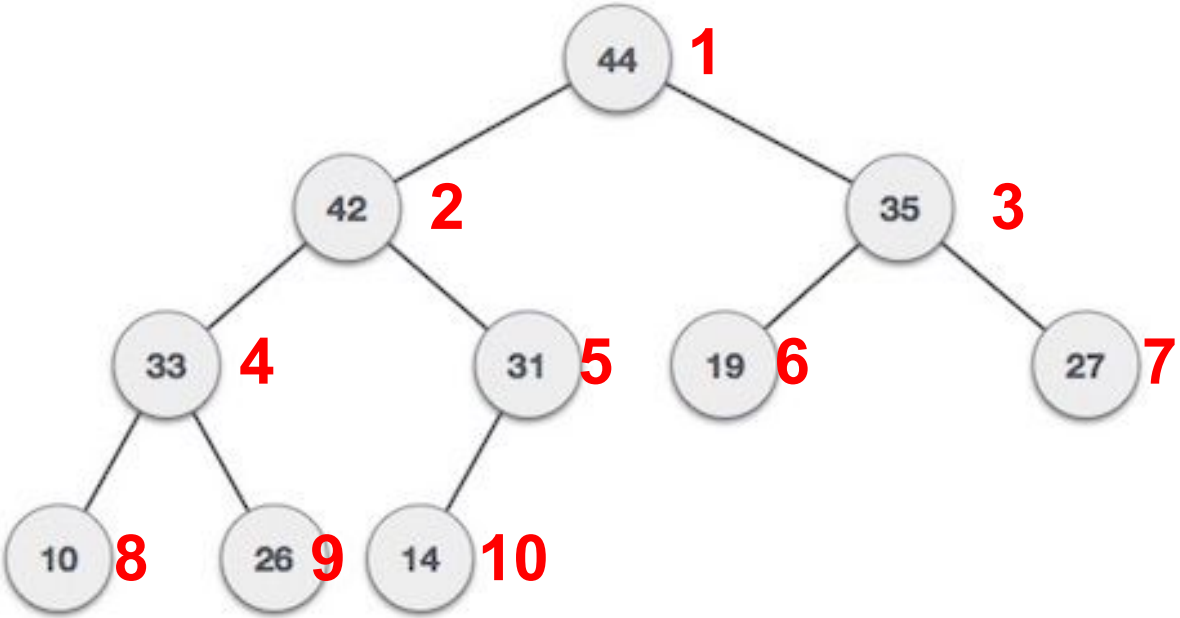
- It is yet another ADT (Abstract Data Type)!
- How should we implement it?
  1. Linked structure is possible

```
class HeapNode:
    self.value ....
    self.left ....
    self.right ....
```

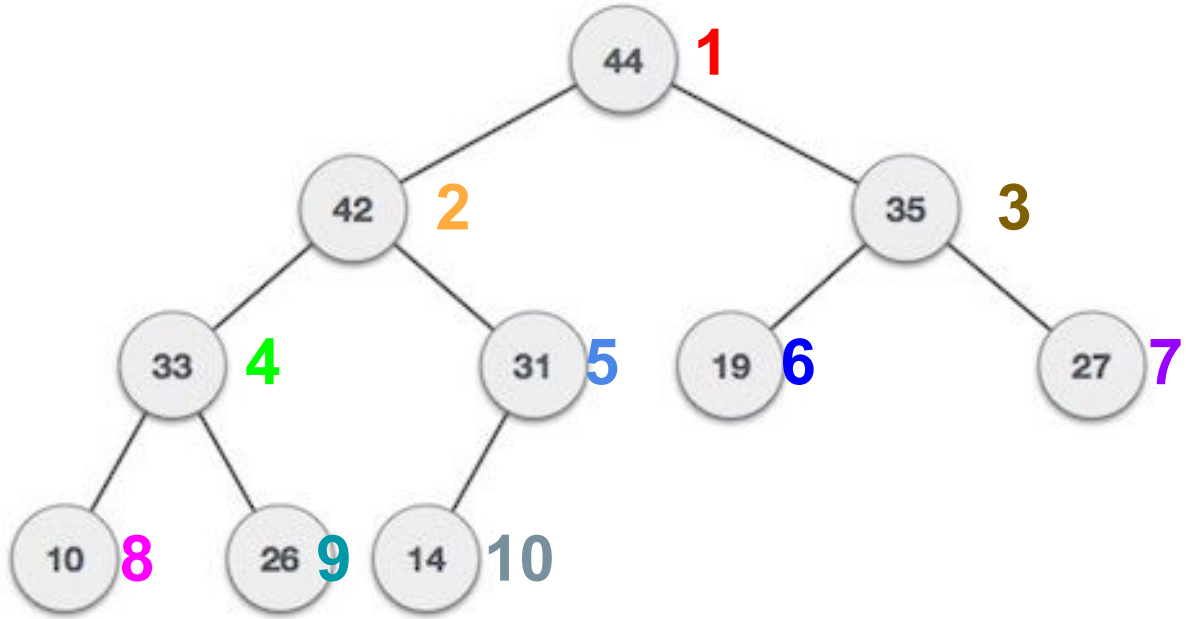
  2. **Another way?**

Binary Heap - Implementation

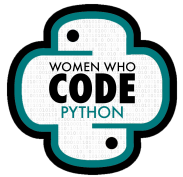
- Use **lists**!
  - Order nodes top to bottom, left to right



Binary Heap - Implementation

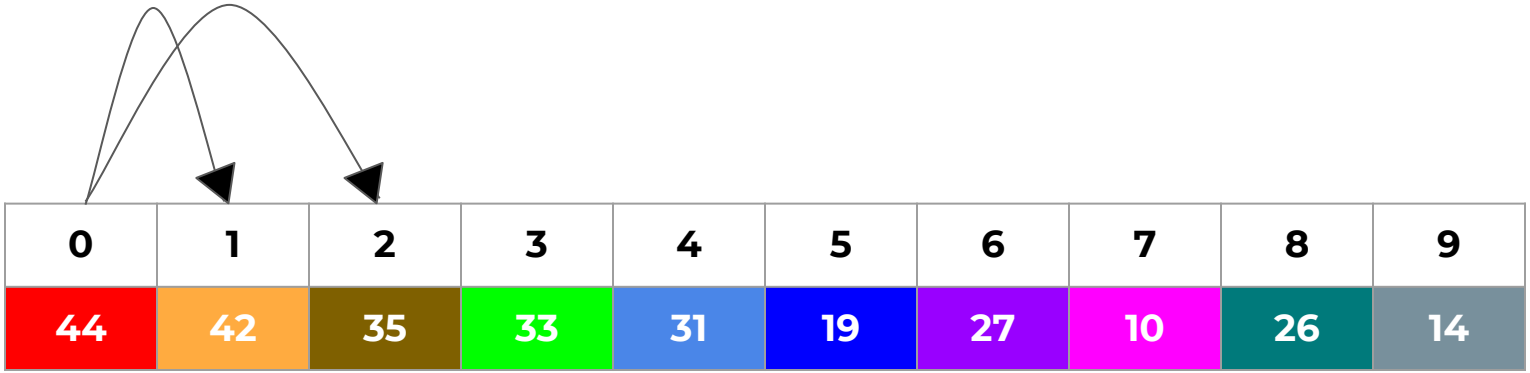


0	1	2	3	4	5	6	7	8	9
44	42	35	33	31	19	27	10	26	14



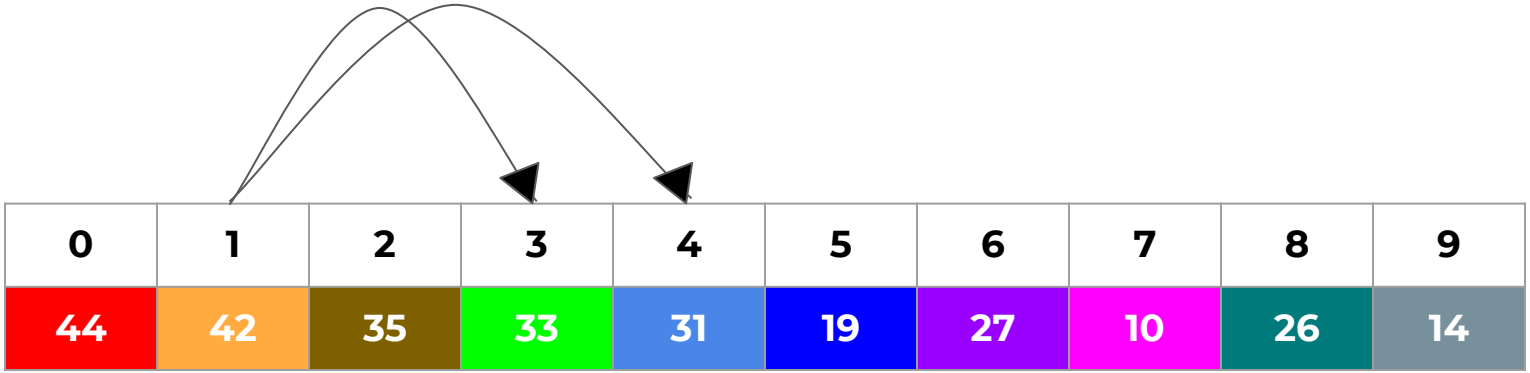
Binary Heap - Implementation

- For any node **`i`**
  - Left child is at **`(2 * i) + 1`**
  - Right child is at **`(2 * i) + 2`**
  - Parent is at **math.ceil(`i / 2`) - 1; `(i - 1) // 2`**



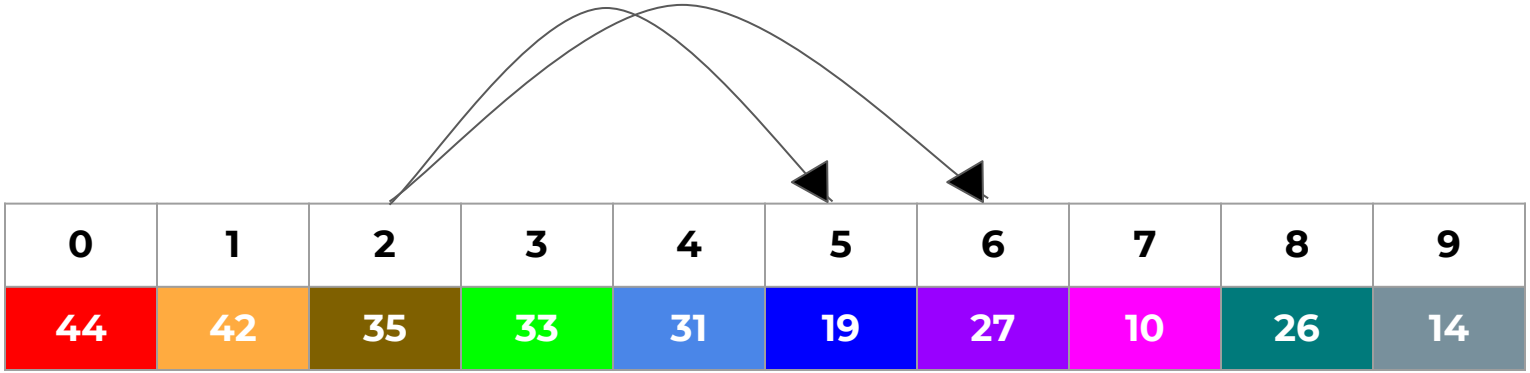
Binary Heap - Implementation

- For any node **`i`**
  - Left child is at **`(2 * i) + 1`**
  - Right child is at **`(2 * i) + 2`**
  - Parent is at **math.ceil(`i / 2`) - 1; `(i - 1) // 2`**



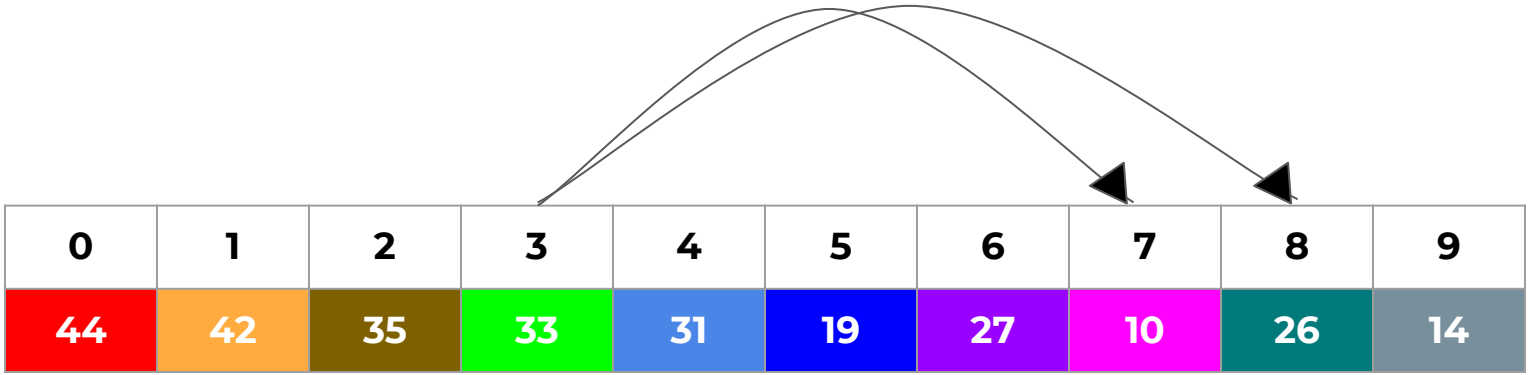
Binary Heap - Implementation

- For any node **`i`**
  - Left child is at **`(2 * i) + 1`**
  - Right child is at **`(2 * i) + 2`**
  - Parent is at **math.ceil(`i / 2`) - 1; `(i - 1) // 2`**



Binary Heap - Implementation

- For any node **`i`**
  - Left child is at **`(2 * i) + 1`**
  - Right child is at **`(2 * i) + 2`**
  - Parent is at **math.ceil(`i / 2`) - 1; `(i - 1) // 2`**



Binary Heap

- Other heaps the other way
  - Max-heap (max at the root):

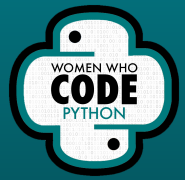
    For every position p other than the root,

    p <= key of p's parent
  - Min-heap (min at the root):

    For every position p other than the root,

    p >= key of p's parent

Q&A Time!



Time for Live Coding!

- https://drive.google.com/file/d/1DMjHnlLhb4b-KoqiGx7OaudAwRdLX5WY/view?usp=sharing

Next Session!

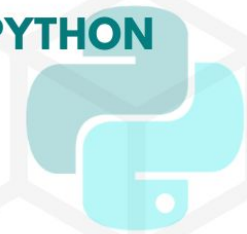
INTRO TO

DATA STRUCTURES

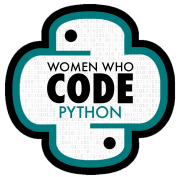
WITH PYTHON

ACE THE TECHNICAL INTERVIEW

THU. JUNE 17TH
@ 8:00PM EDT



WOMEN WHO CODE PYTHON



Questions?

Join our Slack channel:
#intro-data-structures-stdy-grp

Thank You!