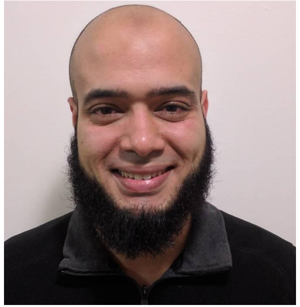
Data Structures

Binary tree using array representation

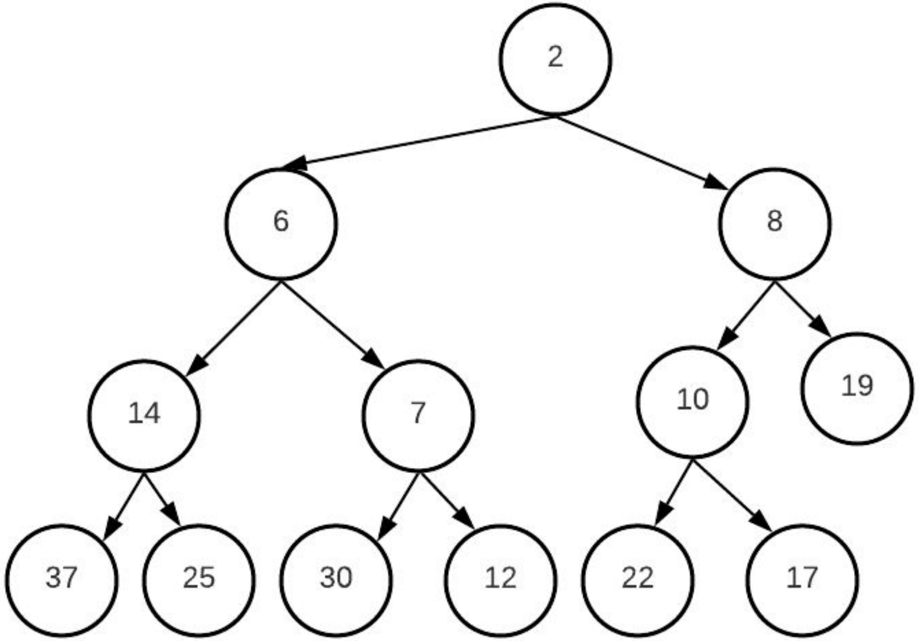
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



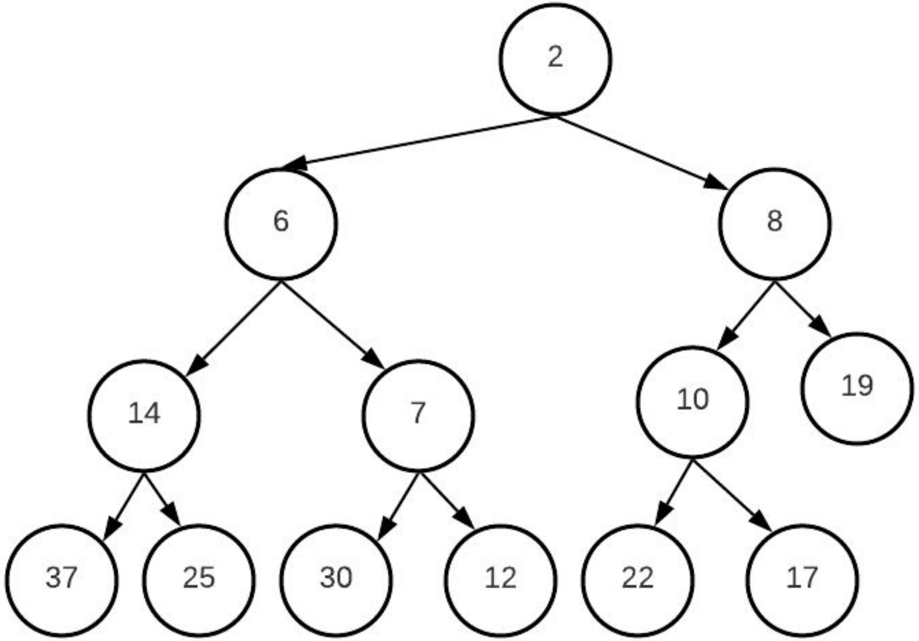
Utilizing the properties

- As clarified before, if a structure has additional special characteristics, it can offer multiple potential advantages
- What is special about perfect and complete trees?
    - Many top levels are complete



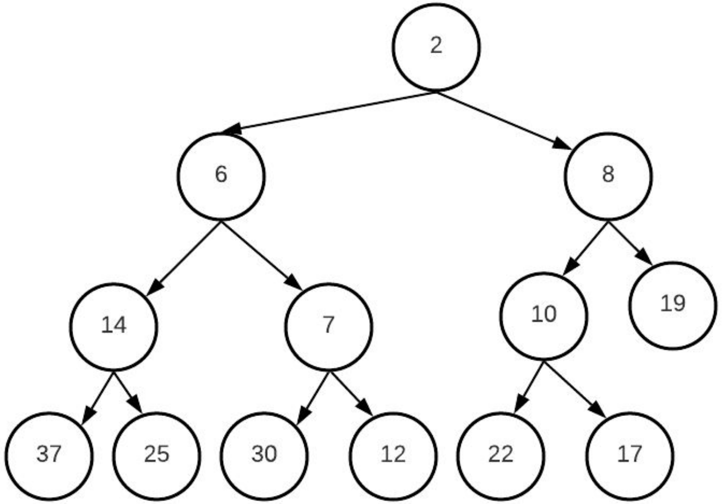
Utilizing the properties

- Nodes per level are 2^i
    - Level 0 = 1
    - Level 1 = 2
    - Level 2 = 4
    - Level 3 = 8
    - Level 4 = 16
    - Level 5 = 32
    - ….
- If we have n levels, the total number of nodes is 2^n -1
- We can simply put **tree level order traversal** in an array



Complete tree ⇐⇒ Level order traversal

- Interestingly: we can convert a tree level-order traversal to the original binary tree
  - Consume: 1, 2, 4, 8, 16, etc
- But what is really important:
  - Given the index of node, what is the parent index?
  - Given a node index, what are the indices of the child nodes?
  - Try to find simple formulas



0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
2	**6**	**8**	**14**	**7**	**10**	**19**	**37**	**25**	**30**	**12**	**22**	**17**		

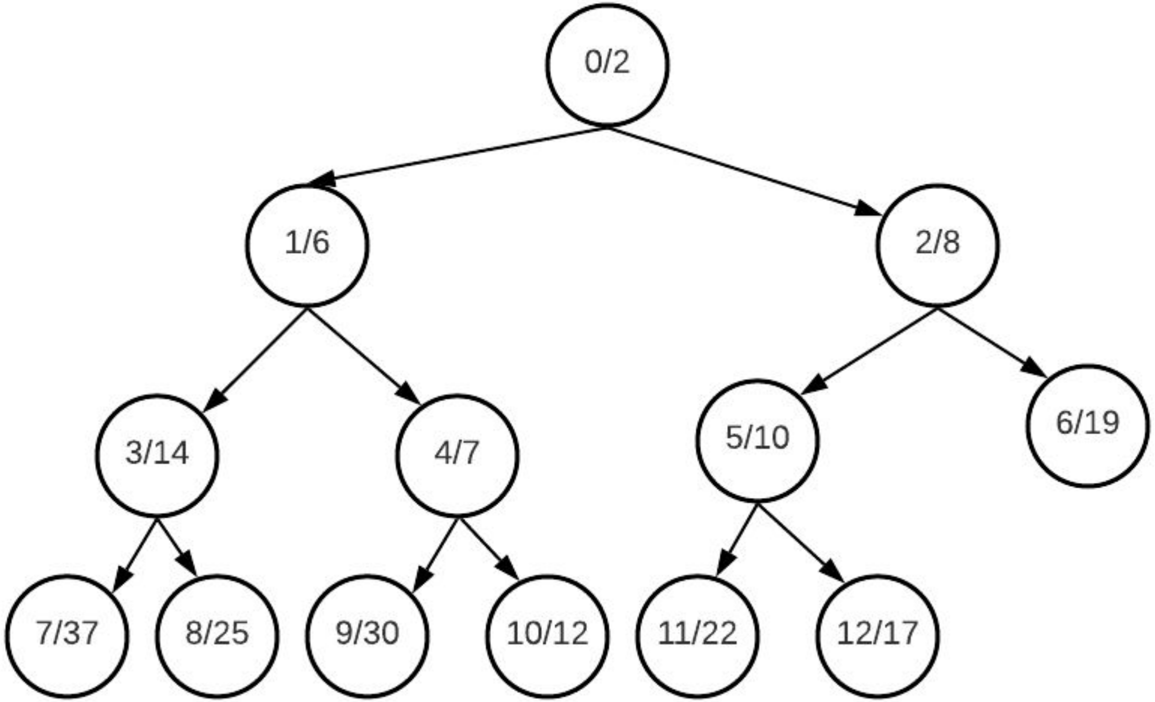
Indices relations

Index	Value	Parent index	Child 1 index	Child 2 index
0	2	NA	1	2
1	6	0	3	4
2	8	0	5	6
3	14	1	7	8
4	7	1	9	10
5	10	2	11	12
6	19	2	13	14
7	37	3	15	16
8	25	3	17	18

What are the formulas?

- Given a node i, the children are
  - 2*i + 1
  - 2*i + 2
- Given a node i, its parent is:
  - (i-1) // 2
    - Floor

Indices relations



What are the formulas?

- Given a node i, the children are:
  - 2*i + 1
  - 2*i + 2
- Given a node i, its parent is:
  - (i-1) // 2
    - Floor

So far

- We can represent a complete tree in an array (level order traversal)
- We can trivially move from a node to its parent in the array
- We can move easily from a node to any/either of its children in the array
- Why is this significant?   For the HEAP!

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
2	**6**	**8**	**14**	**7**	**10**	**19**	**37**	**25**	**30**	**12**	**22**	**17**		

Heap representation

- We can represent a heap using an array
- *Great! But, can't we just use a normal pointer-based tree?*
- Yes, but now we can find the next **available** node position trivially!
  - Below represents a tree of 13 nodes!
  - The next available node in the tree is simply index #13   ⇒ O(1)
- In the next lecture, we'll show how that's useful for the heap!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	6	8	14	7	10	19	37	25	30	12	22	17		

Array representation

- Direct implementation
- Sometimes, the current tree doesn't have a child. If our node is the root node, it will have no parent node
  - Use -1 as indicator

```
class MinHeap:
    def __init__(self):
        self.array = []
        self.size = 0   # Actual number of elements

    def _left(self, node):
        p = 2 * node + 1
        if p >= self.size:
            return -1
        return p

    def _right(self, node):
        p = 2 * node + 2
        return -1 if p >= self.size else p

    def _parent(self, node):
        return -1 if node == 0 else (node - 1) // 2
```

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”