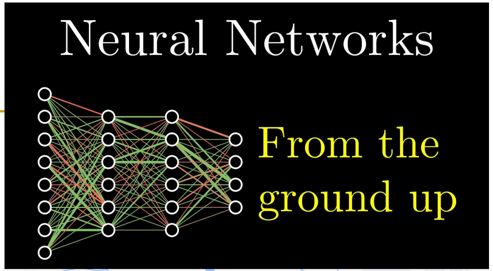
QUEUES & BREADTH-FIRST TRAVERSAL COMPLETE BINARY TREES

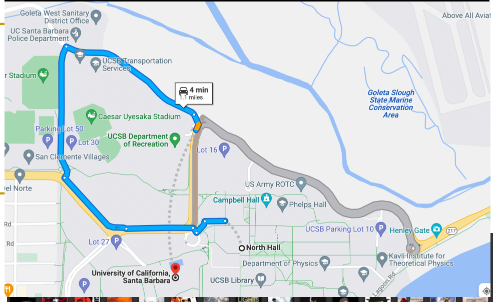
Problem Solving with Computers-II

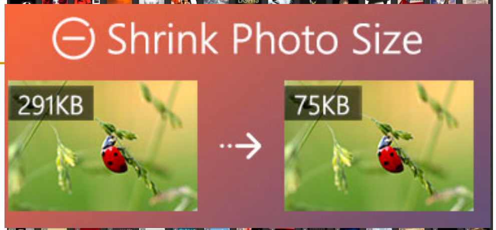
Link to handout: https://bit.ly/CS24-Queue

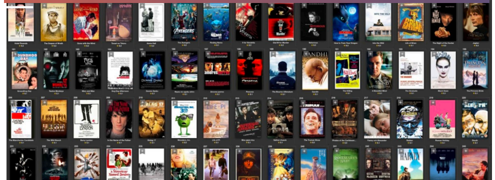
From Data Structures to Real-World Applications

Data Structure	Algorithm	Real-World Application
Queue	Breadth-First Search (BFS)	🤖 Machine Learning (**PA03:** Prediction in NNs)
Queue	Round-Robin Scheduling	🖥️ Operating Systems (Task scheduling)
Priority Queue	Dijkstra’s Algorithm	📍 GPS Navigation (Shortest path)
Priority Queue	Huffman Coding	🗜️ Data Compression (ZIP, JPEG, MP3)
Your choice!	You design!	Querying a movie dataset **(PA02)**



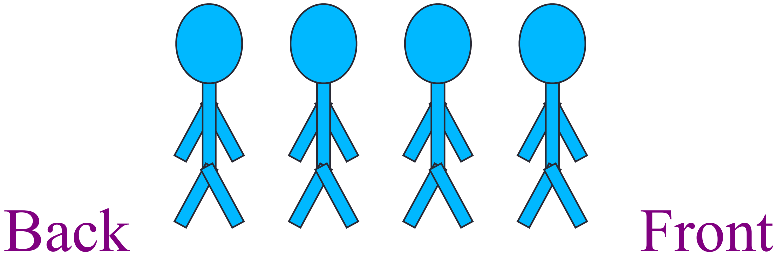






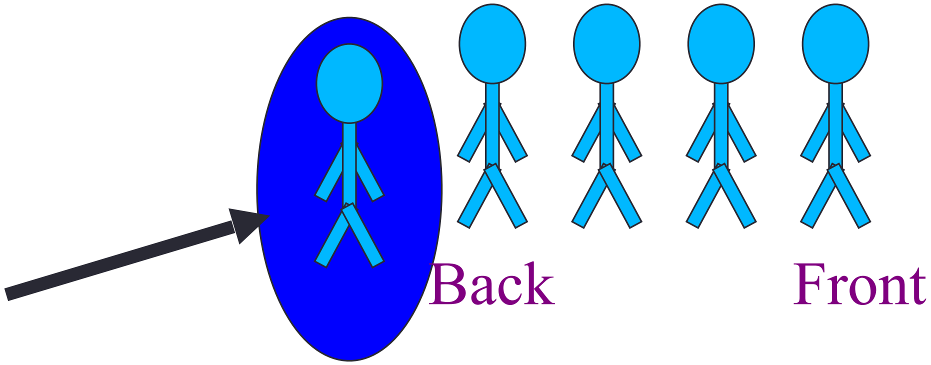
Queue: First come First Serve

- A queue is like a queue of people waiting to be serviced
- The queue has a **front** and a **back**.



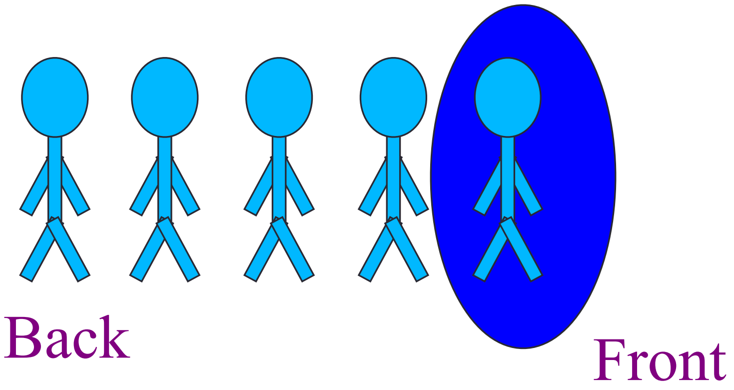
Queue Operations: push, pop, front, back

New people must enter the queue at the back. The C++ queue class calls this a **push** operation.



Queue Operations: push, pop, front, back

- To check the item in the front of the queue, use **front()**
- To check the item at the back of the queue, use **back()**
- When an item is taken from the queue, it always comes from the front.
- To delete an element from the front of the queue, use **pop()**



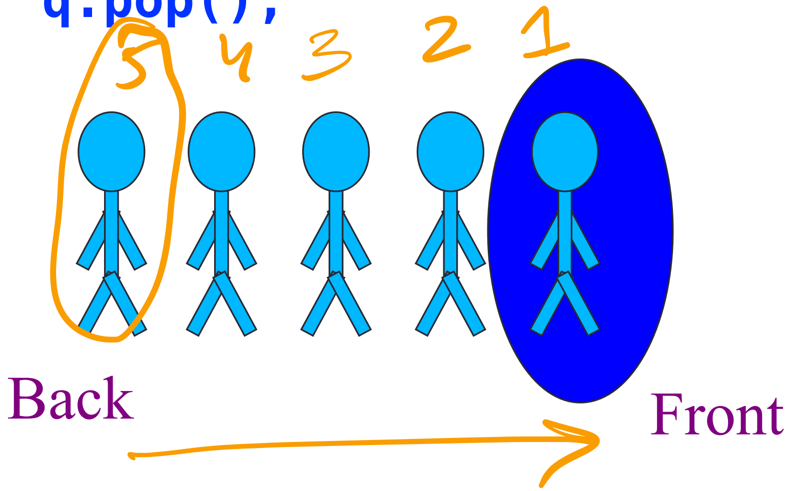
Queue Operations: empty(), push, pop, front, back: **O(1)**

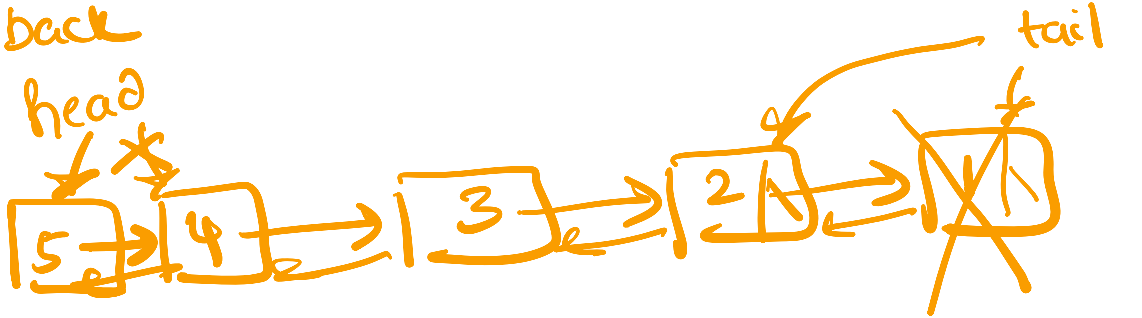
```
std::queue<int> q;
q.empty(); //true
q.push(1);          = push-front
// push 2, 3, 4, 5
q.front();
q.back();
q.pop();
```

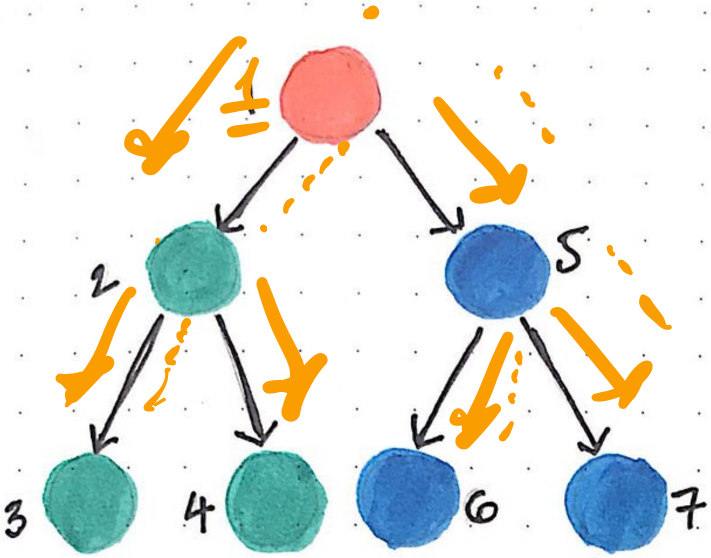
Algorithms: Breadth First Search   Task Scheduling

ADT: Queue

Data structure: Linked list or vector

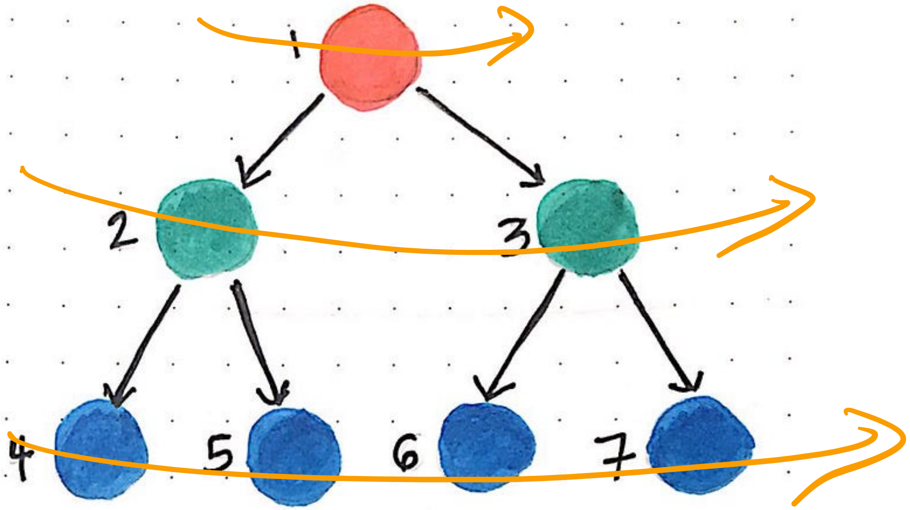






Depth-first search

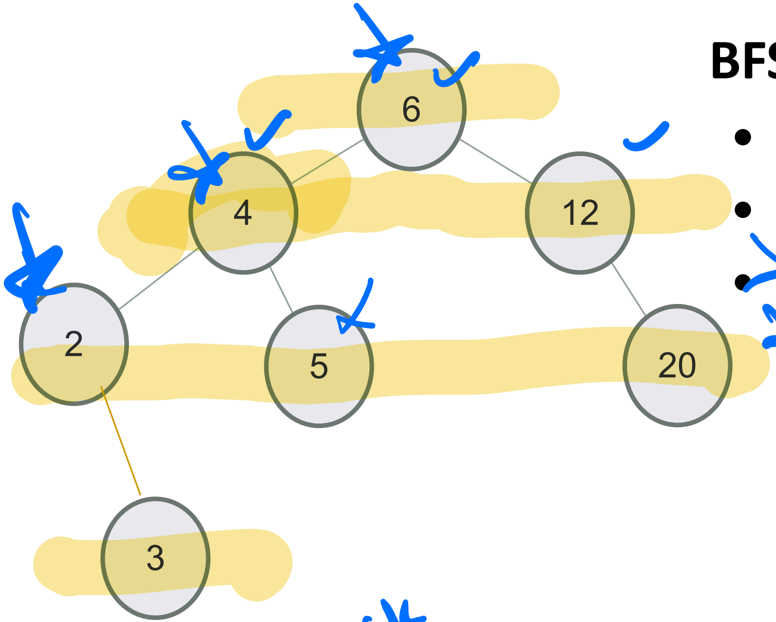
- Traverse through left subtree(s) first, then traverse through the right subtree(s).



Breadth-first search

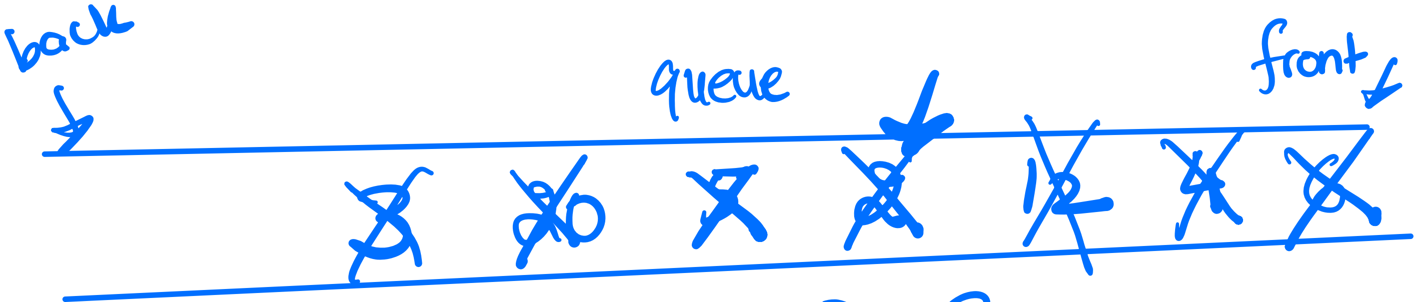
- Traverse through one level of children nodes, then traverse through the level of grandchildren nodes (and so on...).

Breadth-first traversal/search



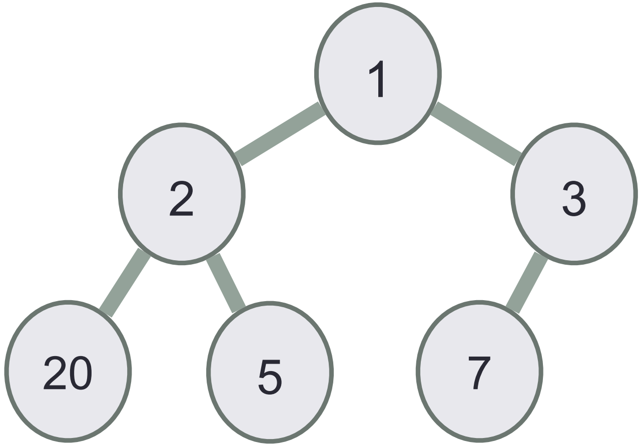
BFS Algo:

- Create an empty **queue**.
- Insert the **root** into the **queue**.
- While queue is not empty,
  - Print the key in the front of the queue
  - Insert all the children of the node into the queue.
  - Pop the front node from the queue



Output: 6, 4, 12, 2, 5, 20, 3

Breadth-first traversal



BFS Algo (store output in a vector: result):

- Create an empty **queue**.
- Create an empty **vector called result**.
- Insert the **root** into the **queue**.
- While queue is not empty,
  - **Append the key in the front of the queue to result**
  - Insert all the children of the node into the queue.
  - Pop the front node from the queue

Activity 1:

1. Trace BFS for the given tree, show how the queue evolves

2. What is the resulting vector?

Connecting: vector and Google maps!

(PA03)

Applications: Machine Learning, Operationing Systems, Image compression, Google maps

Algorithms: BFS | Task Scheduling | Huffman Coding | Dijkstra's Shortest Path

ADT: Queue | Priority Queue

Datastructure: Linked list or vector | Binary Heap

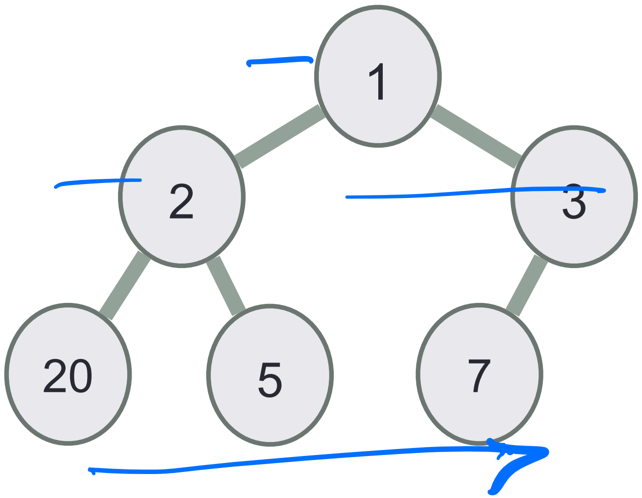
Complete Binary Tree

Vector

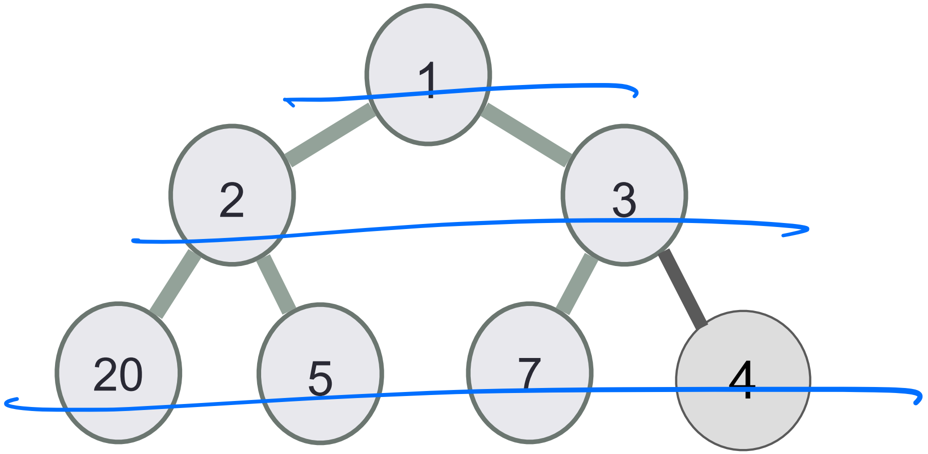
The priority_queue abstract data type (ADT) is implemented as a complete binary tree.

Complete binary tree is effciently represented as a vector, by indexing keys in BFS order.

Structure behind a priority queue



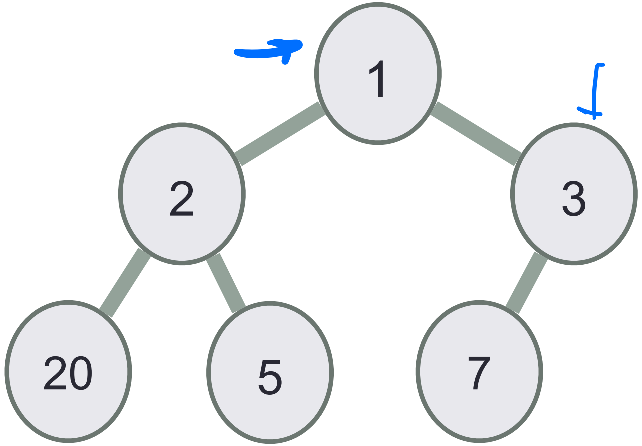
Complete Binary Tree:
Every level is completely filled (except possibly the last level), and all nodes on the last level are as far left as possible

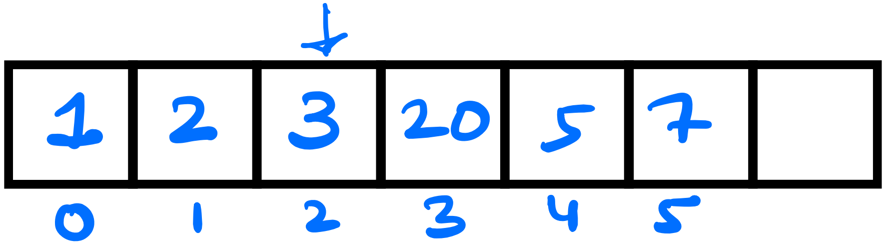


Full Binary Tree: A complete binary tree whose last level is completely filled

Complete/full binary trees are **balanced trees**!

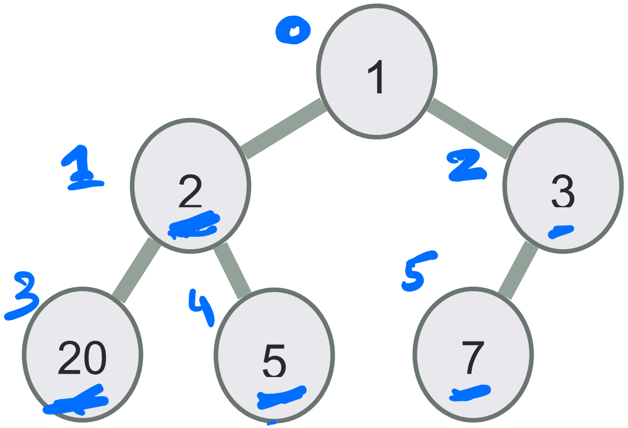
Representing a complete binary tree as a vector!





- How is the index of each key related to the index of its parent?
- How is the index of each key related to the indices of its left and right child?

Representing a complete binary tree as a vector!



index	0	1	2	3	4	5	
key	1	2	3	20	5	7	
parent	-1	0	0	1	1	2	
left child	1	3	5	-	-	-	
right child	2	4	-	-	-	-	

Root is at index 0

For a key at index i, index of its

- **parent is** $\lfloor (i-1)/2 \rfloor$
- **left child is** $2i+1$
- **right child is** $2i+2$

$\frac{5-1}{2} = 2$

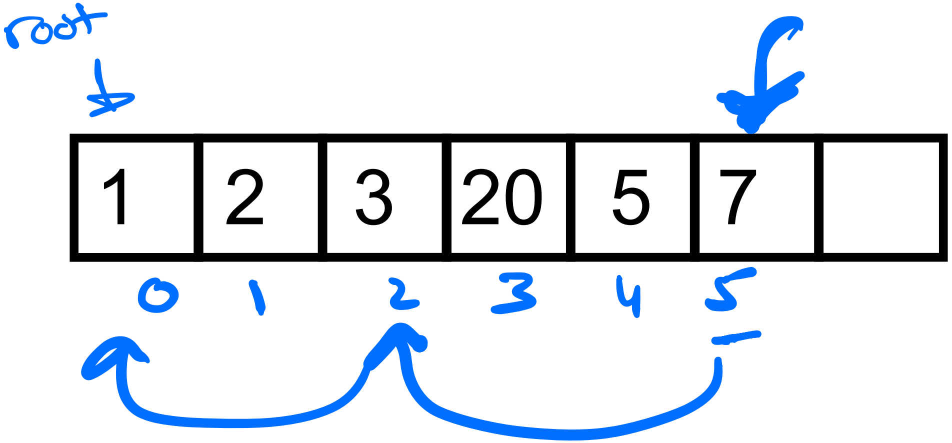
Activity 2: For a key at index i, determine the indices of its parent and children.

Traverse up the tree using the vector (only)!

Root is at index 0

For a key at index i, index of its

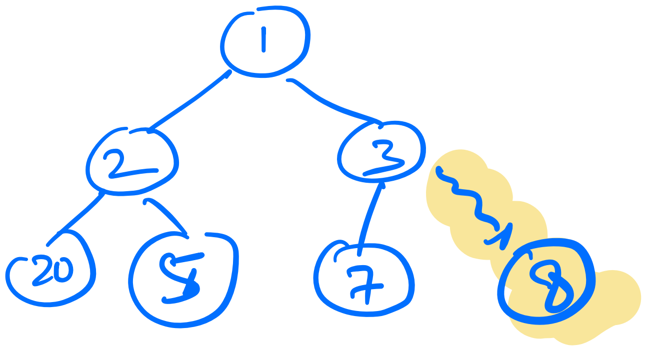
- **parent is** $\lfloor (i-1)/2 \rfloor$
- **left child is** $2i+1$
- **right child is** $2i+2$



Activity 3: Starting at the last node in the last level (7), write the indices of the keys visited on the path to the root node with key (1):

A. 5, 4, 3, 2, 1, 0
B. 5, 4, 2, 1, 0
C. 5, 3, 1
D. 5, 2, 0
E. None of the above

Add a new key with value 4 to the tree represented by this vector



push_back(key)

1	2	3	20	5	7	8

What is the complexity of adding new keys to a complete binary tree?

A. O(1)
B. O(log n)
C. O(n)
D. None of the above

Show that a complete binary tree is balanced

Defn $h(n)$: height of the tree has n nodes

To show: $h(n) = O(\log n)$

For a complete tree with n keys

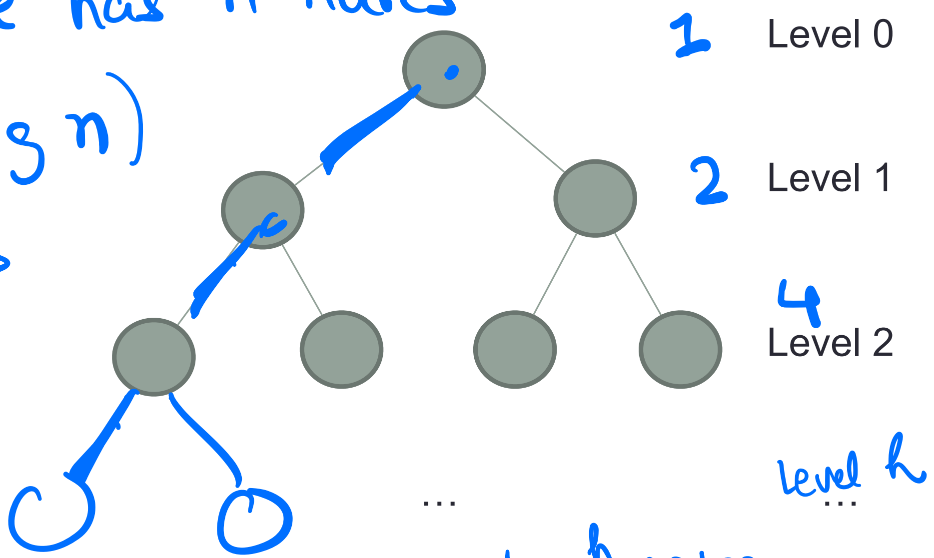
$2^{h(n)} \leq n \leq 2^{h(n)+1} - 1$

Taking log:

$h(n) \leq \log n \leq \log(n+1) \leq h(n)+1$

Since $h(n)$ is bounded by $\log n$

$h(n) = O(\log n)$



Level h may have upto 2^h nodes

Minimum number of nodes in levels 0 to $h-1 = 2^{h-1+1} - 1 = 2^h - 1$

Maximum number of nodes in levels 0 to $h = 2^{h+1} - 1$

Related Leetcode problems to attempt in problem set 3:

- Level Order Traversal of Binary Tree (medium): <https://leetcode.com/problems/binary-tree-level-order-traversal/description/?envType=problem-list-v2&envId=binary-tree>

- Binary Level Order Traversal II (medium): <https://leetcode.com/problems/binary-tree-level-order-traversal-ii/description/?envType=problem-list-v2&envId=binary-tree>