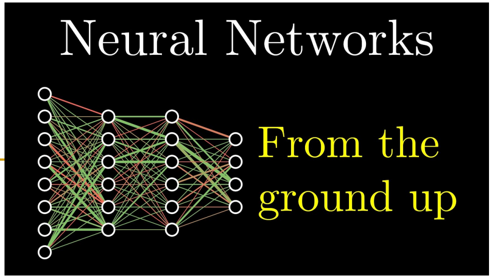
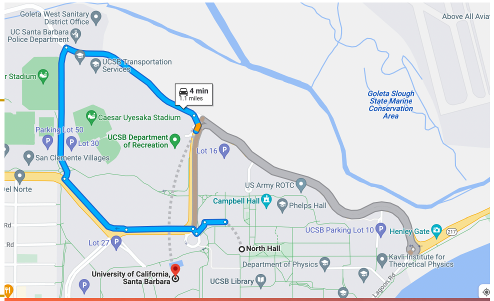
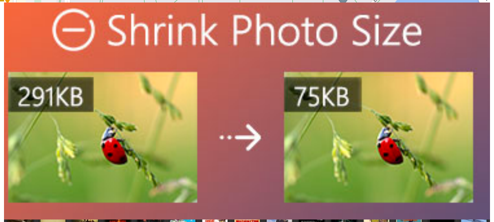
QUEUES
BREADTH-FIRST TRAVERSAL
COMPLETE BINARY TREES

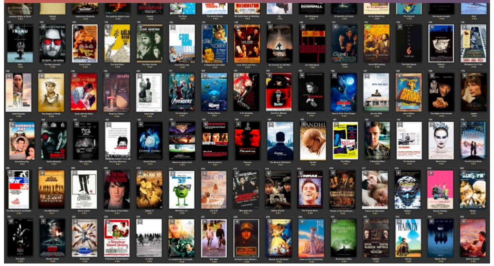
Problem Solving with Computers-II

ADT	Algorithm	Real-World Application
Stack	Depth-First Search	Machine Learning (**PA03:** Training and Prediction in Neural Nets)
Queue	Breadth-First Search	
Priority Queue (Lab 04)	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)**



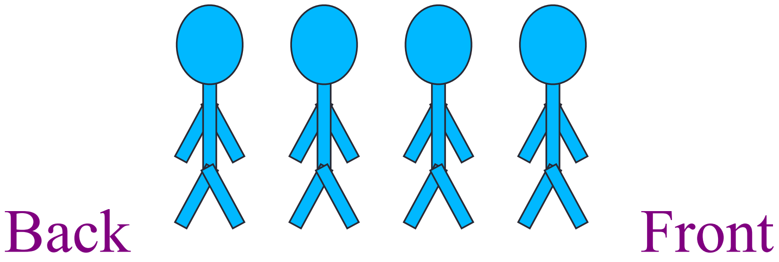






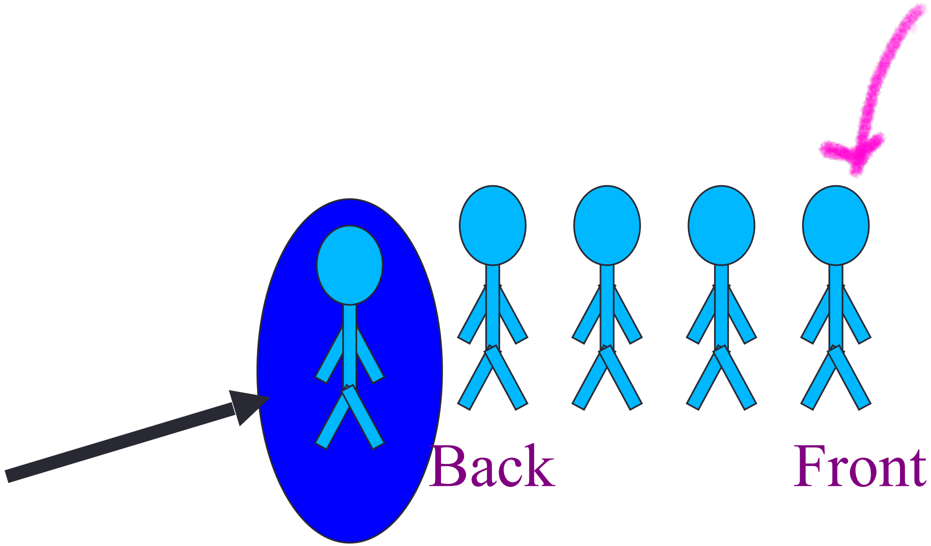
Queue: First come First Serve

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



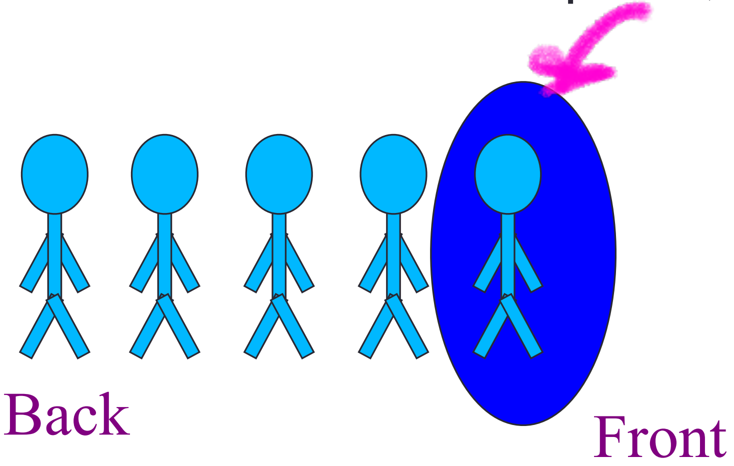
Queue Operations: push, pop, front, back

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



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

- 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 2, 3, 4, 5
q.front();
q.back();
q.pop();
```

Algorithms: **Breadth First Search** **Task Scheduling**

\ /

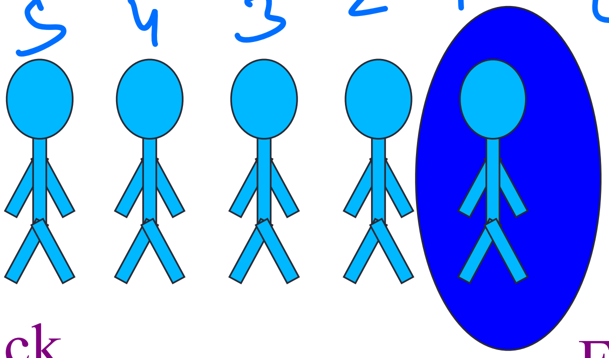
ADT: **Queue**

|

Data structure: **Linked list or vector**

push(6) → push front (linked list)

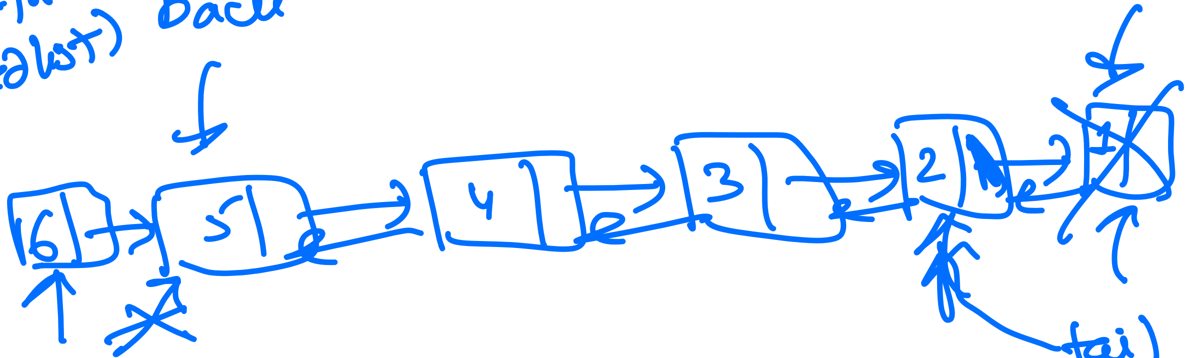
5 4 3 2 1



Back Front

back

front



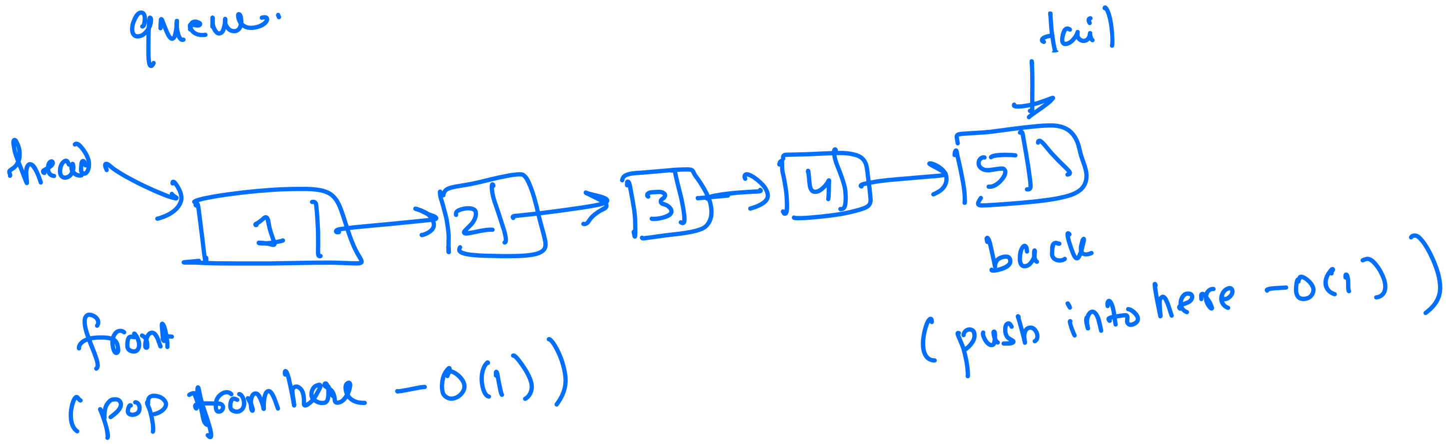
head

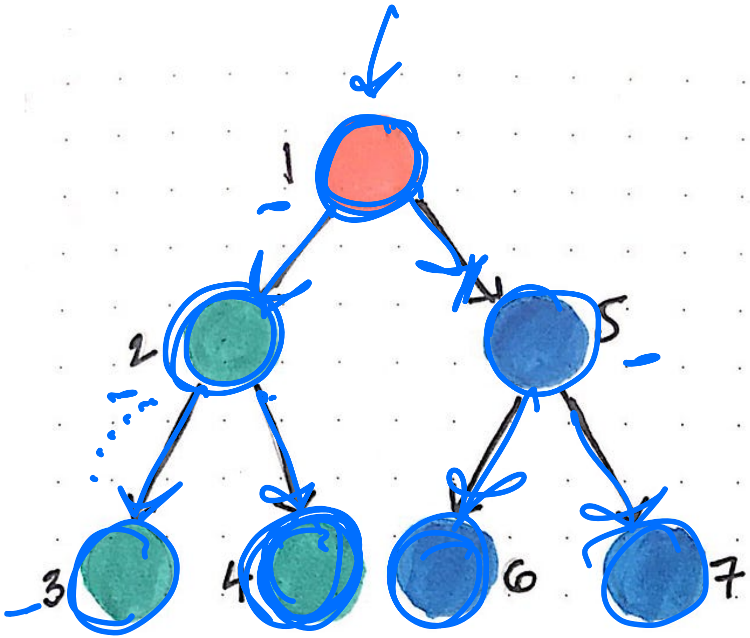
tail

O(1)

O(1)

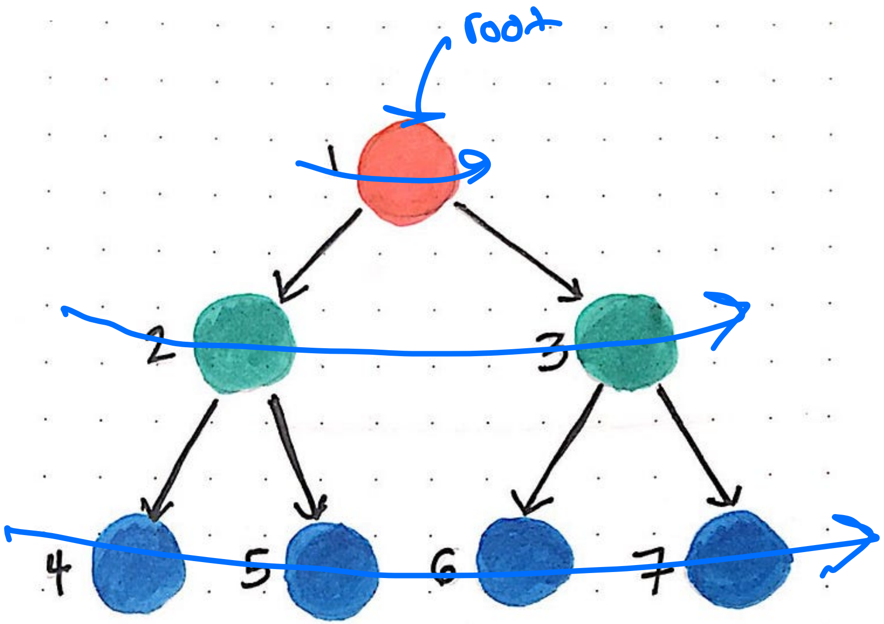
Another possible implementation for a queue is to have a singly-linked list with head of list representing the front of the queue and tail of list representing the back of the queue.





Depth-first search

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

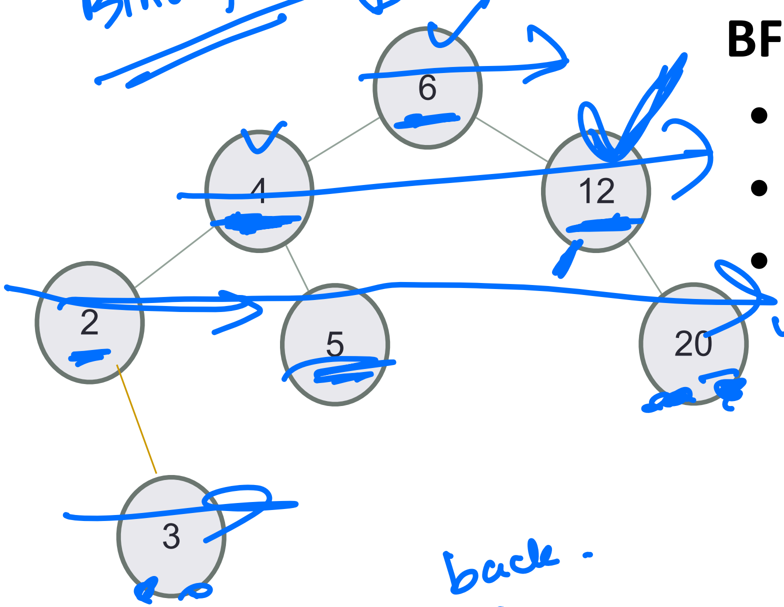


Breadth-first search

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

Breadth-first traversal/search

Binary Tree

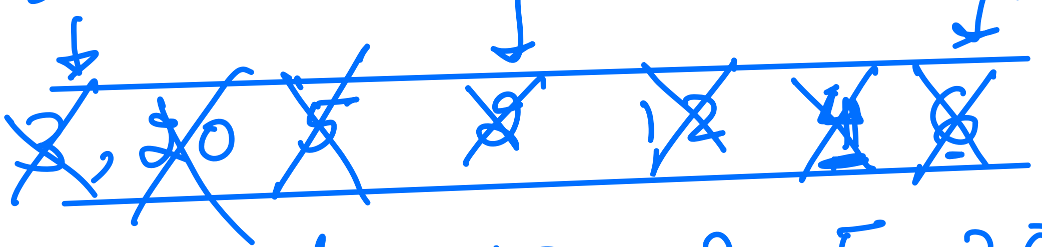


BFS Algo:

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

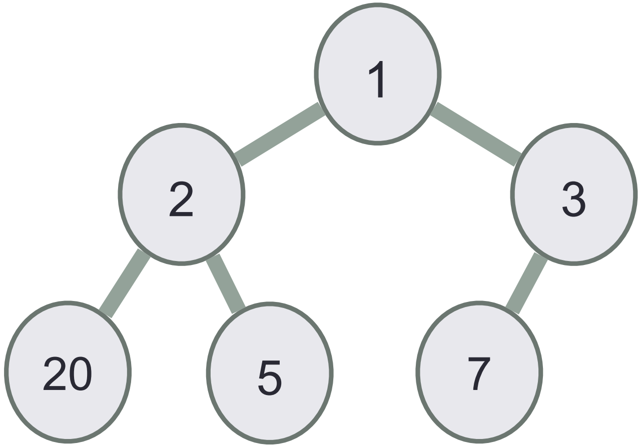
back

front



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**
  - Push all the children of the node into the queue.
  - Pop the front node from the queue

Activity 1:

1. **Write the Breadth First Traversal in your handout**
2. **Trace it on the given tree, showing how the queue evolve.**
3. **Show the output as a vector of key values?**

Connect vector with Google maps!

Applications: Machine Learning, Operating System   Image compression, Google maps

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

ADT: Queue   Priority Queue

Data structure: Linked list or vector   Heap

Complete Binary Tree

Vector

The priority_queue **abstract data type (ADT)** is implemented as a **heap**

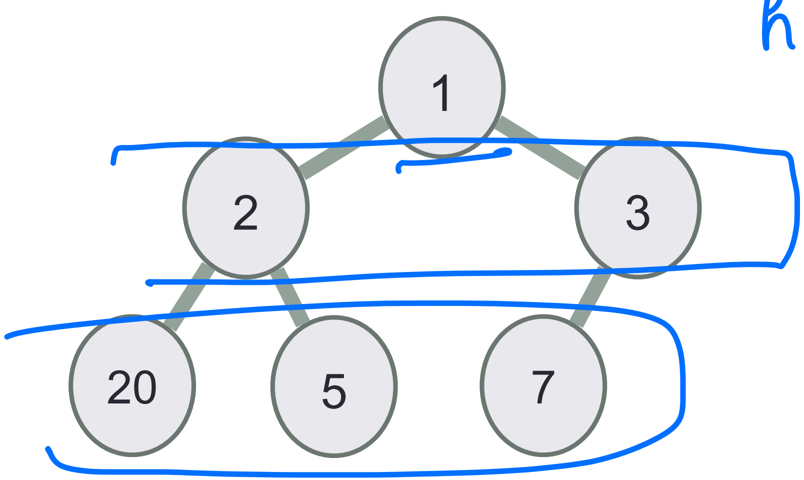
Heap is a complete binary tree with some properties (next lecture)

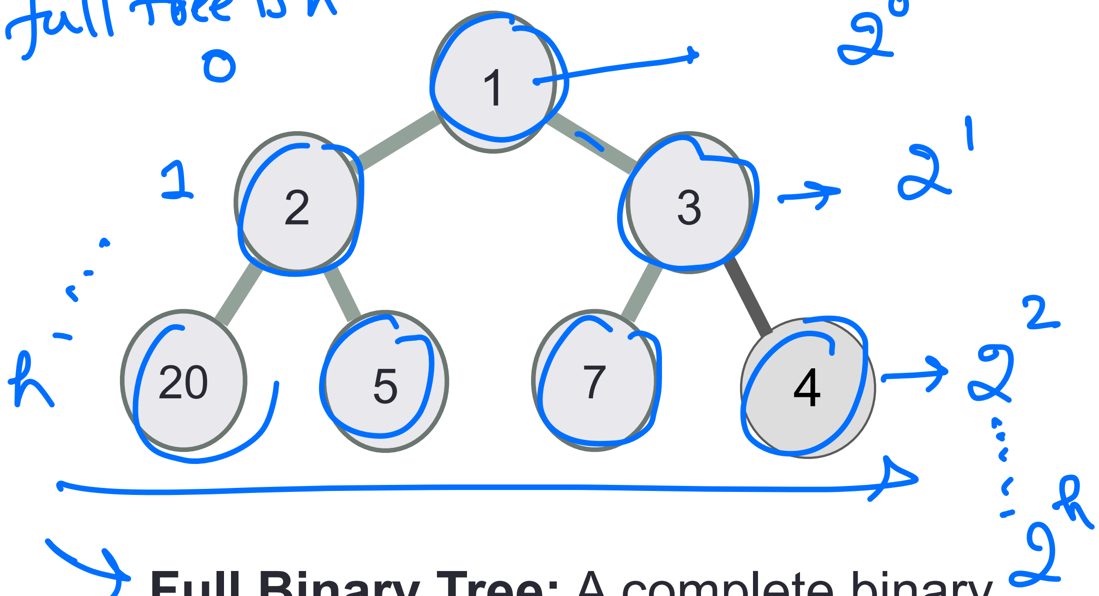
Complete Binary Tree (today!)

Complete Binary Tree

$2^0 + 2^1 + 2^2 + \cdots 2^7 = 2^8 - 1$

height of full tree is h





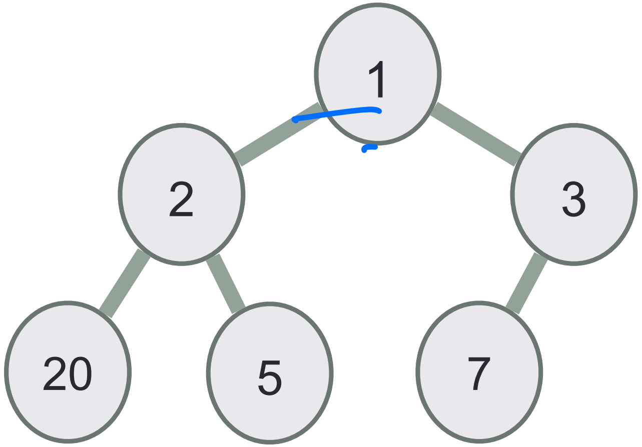
Complete Binary Tree:
Every level is completely filled (except possibly the last level), and all nodes on the last level are as far left as possible

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

$$2^0 + 2^1 + \cdots\cdots 2^h = 2^{h+1} - 1$$
$$= n$$

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

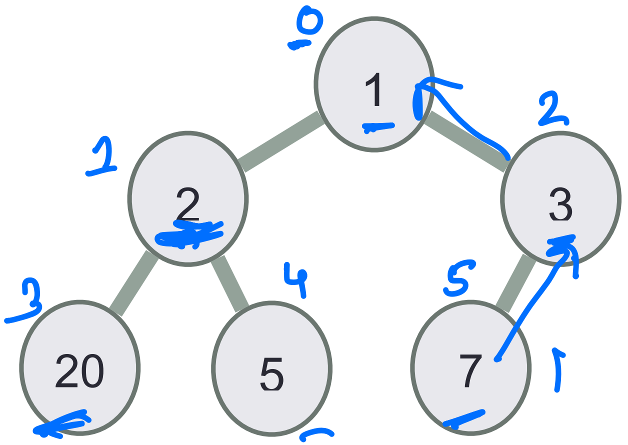
Representing a complete binary tree as a vector!



1	2	3	20	5	7	

- 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	/	/	/	/	

← indices.

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

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** $2i + 2$

$\left\lfloor \frac{1-1}{2} \right\rfloor = 0$

$2*1+1 = 3$

$2*1+2 = 4$

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** $2i+2$

1	2	3	20	5	7	
0	1	2	3	4	5	

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

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

5, 2, 0
(7) (3) (1)

Activity 4: Add a new key with value 4 to the complete binary tree represented by the following vector

O(1)

1	2	3	20	5	7	4

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

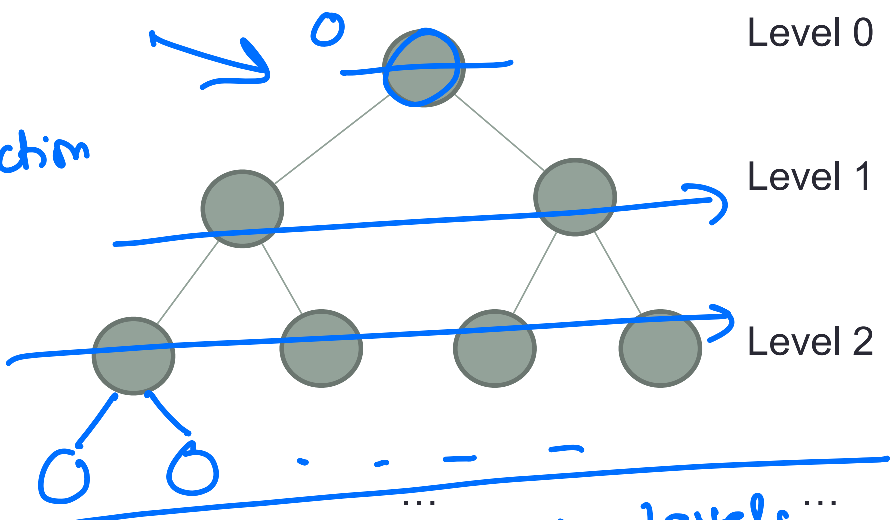
A. O(1)
B. O(log n)
C. O(n)
D. None of the above

Activity 5: Show that a complete binary tree is balanced

height = h no. of key = n

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

Approach: upper bound h as a function of n



Total no. of keys in levels 0 to $h-1$ $\leq n \leq 2^{h+1}-1$

Total no of nodes in levels 0 to $h-1$ is: $2^0 + 2^1 + 2^2 + \ldots 2^{h-1} = 2^{(h-1)+1} - 1$

$2^h - 1 \leq n$

$\Rightarrow 2^h \leq n+1$

$h \leq \log(n+1) \Rightarrow h = O(\log n)$

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>