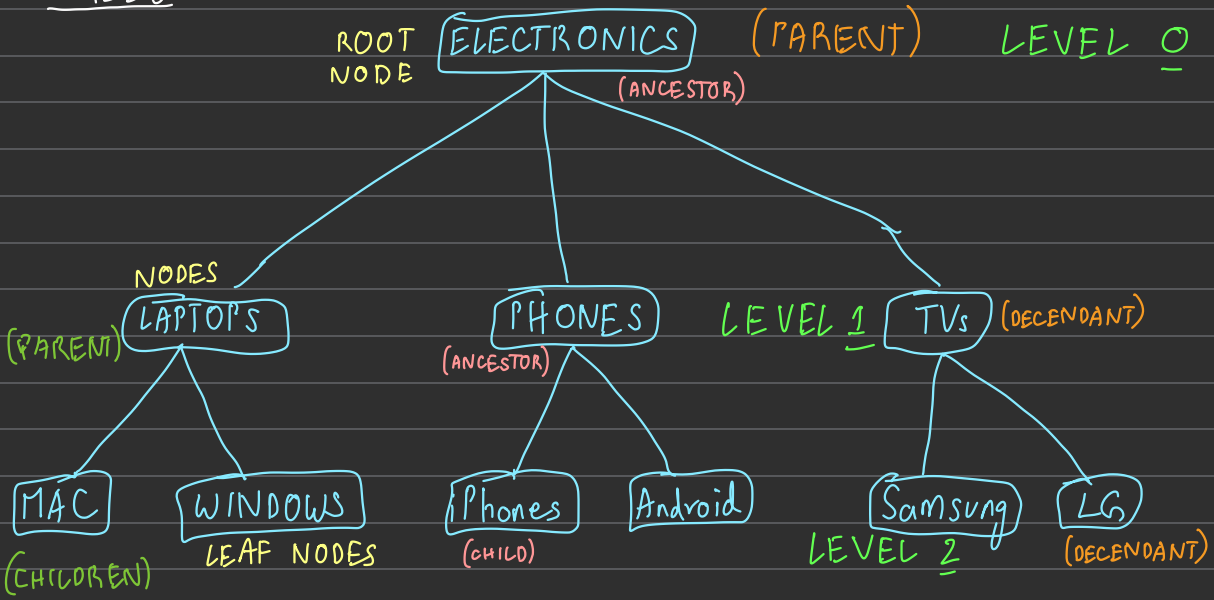
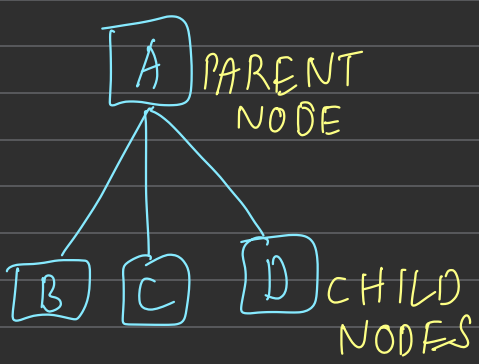
Trees

General Trees

TREES:





TREE IS A RECURSIVE DATA STRUCTURE

Programming Assignment-

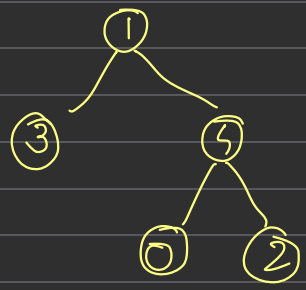
(1)

5

4 -1 4 1 1

0	1	2	3	4
4	-1	4	1	1

1	3	4	0	2
-1	1	1	4	4



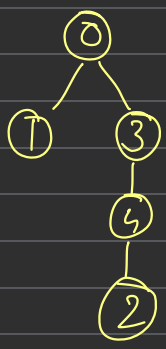
(2)

5

-1 0 4 0 3

0	1	2	3	4
-1	0	4	0	3

0	1	3	4	2
-1	0	0	3	4



(Rough Work)

0	1	2	3	4
[4	-1	4	1	1]

0 1 2 3 4 root_index = -1

3 4 0 2

Code to build tree -

```
class Tree:
    def __init__(self, data):
        self.data = data
        self.children = []
        self.parent = None

    def get_lvl(self):
        count = 0
        while self.parent:
            count += 1
            self = self.parent
        return count

    def print(self):
        print(" "*self.getlvl() + str(self.data))
        for i in self.children:
            i.print(self)
        return

def createTree(n, seq):
    nodes = [None]*n
    for i in range(0, n):
        nodes[i] = Tree(i)
    for i, parent_index in enumerate(seq):
        if parent_index == -1:
            root_index = i
        else:
            nodes[parent_index].children.append(nodes[i])
    nodes[root_index].print()
    return

if __name__ == "__main__":
    createTree()
```

Problem Description

Task. You are given a description of a rooted tree. Your task is to compute and output its height. Recall that the height of a (rooted) tree is the maximum depth of a node, or the maximum distance from a leaf to the root. You are given an arbitrary tree, not necessarily a binary tree.

Input Format. The first line contains the number of nodes n. The second line contains n integer numbers from -1 to $n-1$ — parents of nodes. If the i-th one of them ($0 \le i \le n-1$) is -1, node i is the root, otherwise it's 0-based index of the parent of i-th node. It is guaranteed that there is exactly one root. It is guaranteed that the input represents a tree.

Constraints. $1 \le n \le 10^5$.

Output Format. Output the height of the tree.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	6	3	1.5	2	5	5	3

Memory Limit. 512MB.

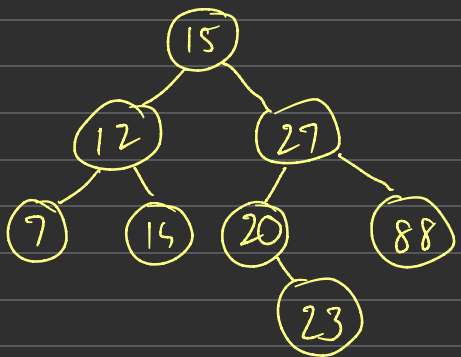
Binary Trees

BINARY TREES :

* Trees with at most 2 nodes

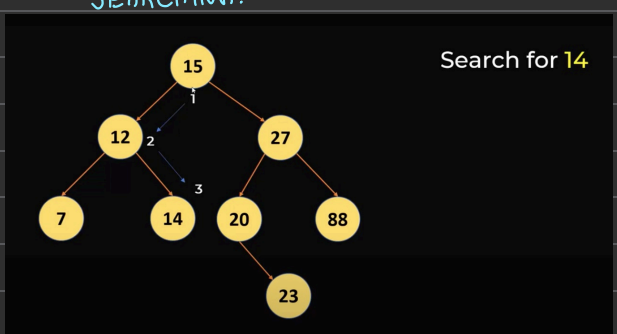
BINARY SEARCH TREE -

* Binary Tree with a special condition



- left < right
- unique elements

SEARCHING :



Search complexity

Every iteration we reduce search space by 1/2

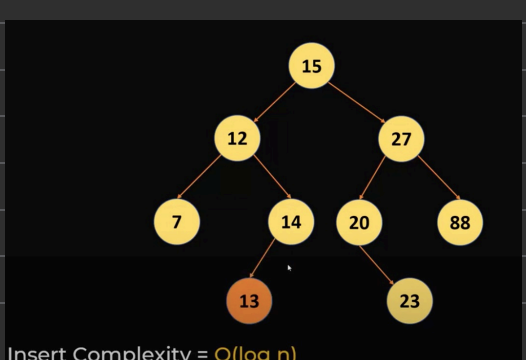
n = 8 $\quad 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

3 iterations

$\log_2 8 = 3$

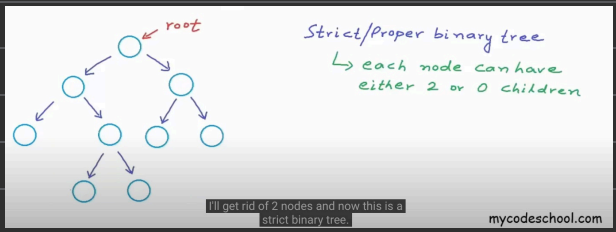
Search Complexity = O(log n)

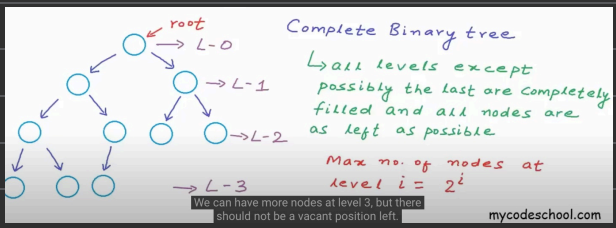
INSERTING :



https://www.youtube.com/watch?v=H5Jubkly_p8

BINARY TREE TYPES :

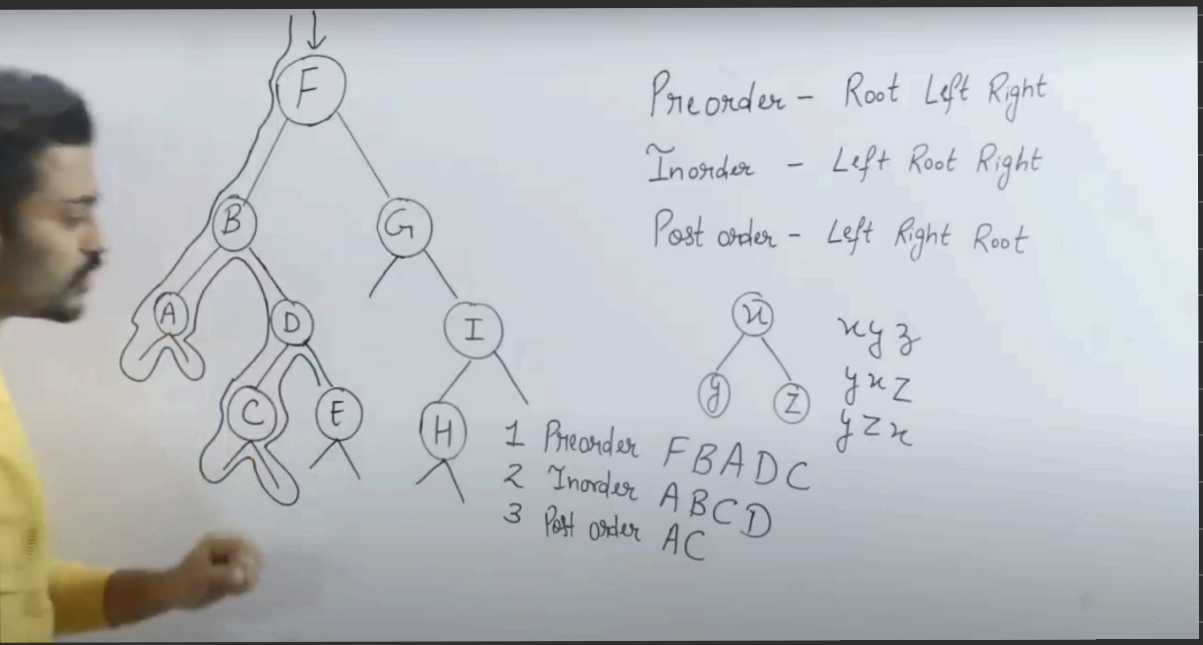




Walking a Tree

Often we want to visit the nodes of a tree in a particular order.
For example, print the nodes of the tree.

- Depth-first: We completely traverse one sub-tree before exploring a sibling sub-tree
- Breadth-first: We traverse all nodes at one level before progressing to the next level



Preorder = 1st time you encounter element
Inorder = 2nd time
Postorder = 3rd time

NOTE - * complete the tree, leaf nodes pe add dummy nodes

https://www.youtube.com/watch?v=XRcC7bAtL3c

Priority Queue

Priority Queue -

It is a generalisation of a queue where each element is assigned a priority & elements come out in order of priority.

Priority Queues: Typical Use Case

Scheduling jobs

- Want to process jobs one by one in order of decreasing priority. While the current job is processed, new jobs may arrive.
- To add a job to the set of scheduled jobs, call `Insert(`*job*`)`.
- To process a job with the highest priority, get it by calling `ExtractMax()`.

(*) NOT important how elements are inserted (stored)

(*) Remove MAX priority first.

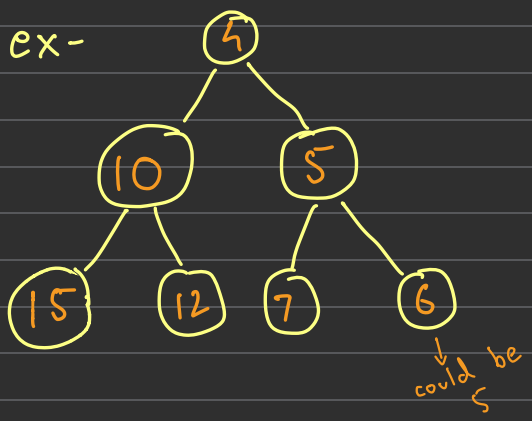
Priority can be - value of the element itself as the priority
(OR) a tuple is used → storing data & priority

Lists can be used to implement this, but HEAPQ is best way to implement.

HEAPs -

① MIN HEAP - root → SMALLEST NUMBER

ex-

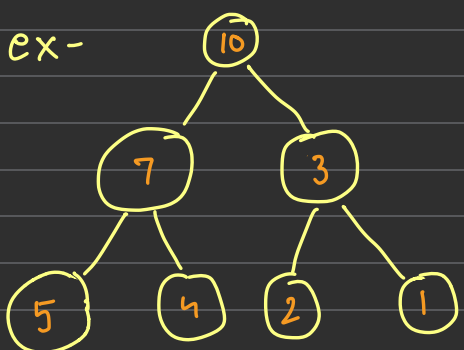


LOWER the number,
HIGHER the priority.

→ smallest element is removed first, the we "heapify" the rest of the tree to match the property again.

② MAX HEAP - root → LARGEST ELEMENT

ex-

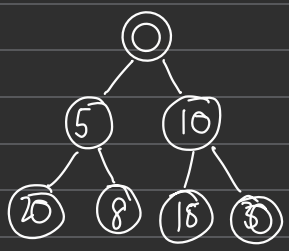


HIGHER the number,
HIGHER the priority.

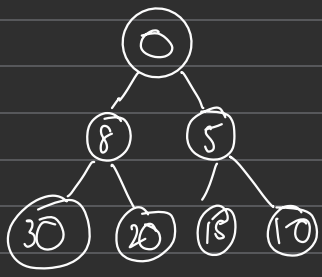
→ largest element is removed first, the we "heapify" the rest of the tree to match the property again.

Rough Work-

[0, 5, 10, 20, 8, 15, 30]



[0, 8, 5, 30, 20, 15, 10]



https://www.youtube.com/watch?v=hkyzcLkmoBY

Disjoint Sets

Definition

A disjoint-set data structure supports the following operations:

- `MakeSet`(x) creates a singleton set $\{x\}$
- `Find`(x) returns ID of the set containing x:
  - if x and y lie in the same set, then `Find`$(x) =$ `Find`(y)
  - otherwise, `Find`$(x) \neq$ `Find`(y)
- `Union`(x, y) merges two sets containing x and y