Binary Tree - optimized for searching.

Node.
↓
* Data
* Left
* right

property:-

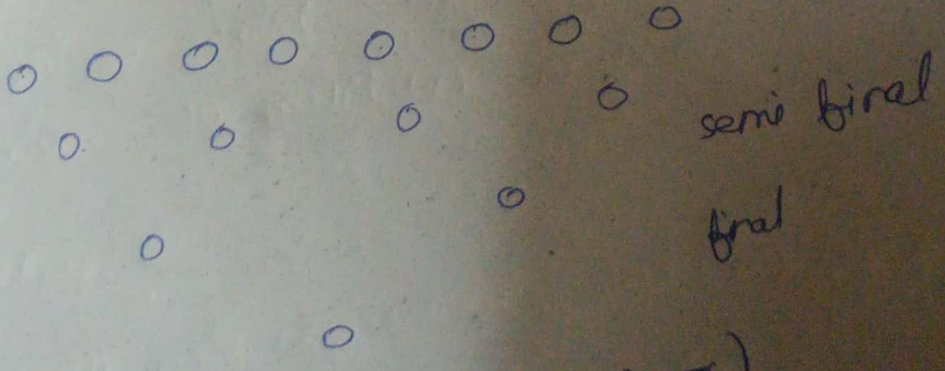
Right child < root.
left child > root.

⇒At each step, it'll cut down the search time.

E.g

Tennis Tournament.

$2 \to 4 \to 8 \to 16 \to 32$



$F(256) \to 8$

$F(128) \to 7$

$F(64) - 6$

$F(32) - 5.$

$\log(256)$

$\log(2^8)$

$8 \log 2$

$\boxed{8.}$

Logarithmic complexity.

At every step, state space is cut down by half.

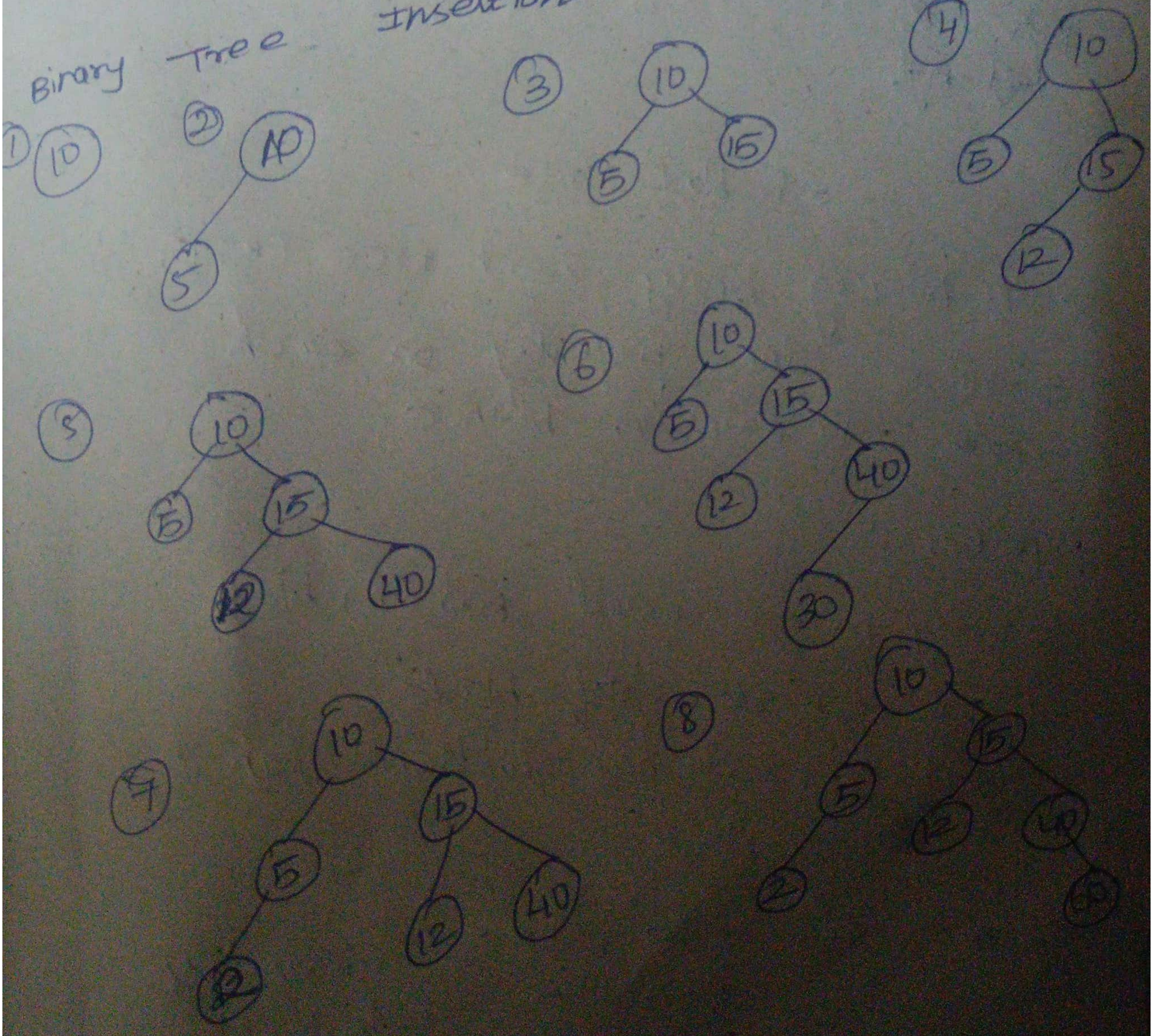
e.g. 3^2 3^3 3^4 3^5

3 → 9 → 27 → 81 → 243

order of time complexity :-

O(1) → O(logn) → O(n) → O(nlogn)

Binary Tree Insertion

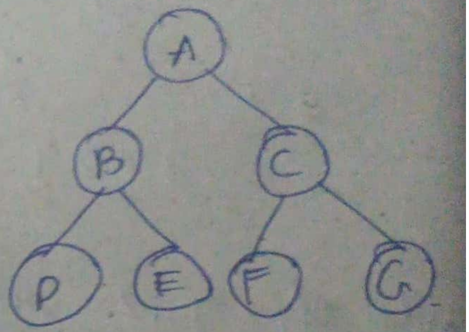


Binary Tree Traversal

1) Inorder

2) pre order.

3) post order.



Inorder E.g DBEAFCG.

Left → Root → Right

pre order
Root → left → right. e.g: ABDE CFG.

post order:-
~~Right~~ ~~→Left~~ → root. e.g: ~~[illegible]~~ DEBFGCA.
left - Right

Red black Tree.

⇒ self balancing binary search tree (BST)

⇒ utilize color property on each node to maintain balance.

properties:-

⇒ Every node is colored Red (or) black

⇒ root = black, leaf = black

⇒ If red node has childrens then childrens = black

⇒ ~~[illegible]~~ any simple path from this node to decendant leaf has the same black depth

Treenode

1) color.

2) left

3) right.

~~1~~4) parent.

5) key

Rotation:-

⇒ Used to maintain properties of red-black tree. when violated by,

* insertion

* deletion

1) Left Rotate

2) Right Rotate

3) Left Right Rotate

4) Right Left Rotate.