# Search Operation

A - array

search$(A, x)$ :- check if $x$ is in $A$

$O(n)$

$A =$

| | | | | |
|---|---|---|---|---|
| 2 | 5 | 1 | 8 | 3 |

search$(A, 4)$ :- 4 is not in $A$ | Linear Search

A - sorted array

search$(A, x)$

search$(A, 4)$

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 8 |

4 does not appear here

4 is not in $A$.

| | |
|---|---|
| 5 | 8 |

Viraj

Why Binary Search is $O(\log n)$

$$T(n) \leq T(n/2) + O(1)$$

$$O(\log n)$$

## Insert Operation

Insert $(A, x)$ :- insert $x$ in $A$ if it is not already in $A$.

### Array A

Time required for insertion =

Insert $(A, x)$
    Search $(A, x)$
        if $x$ is not in $A$
            insert at the end of $A$.

$O(n)$

### Sorted Array

Aditi : $O(\log n)$
    Not correct

$A=$

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 8 |

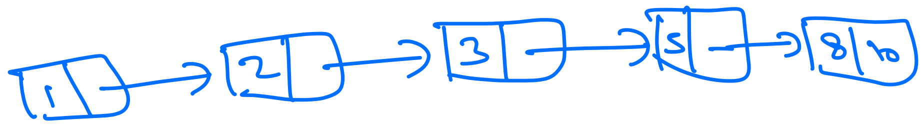
Insert $(A, 4)$

In $O(\log n)$ time we can find the position where $x$ need to be inserted.

Then, we might need to shift elements to left or right.

$O(n)$

## Sorted Linked List




Insertion is $O(1)$ | Bad!

Search is $O(n)$

## Binary Search Tree (BST)

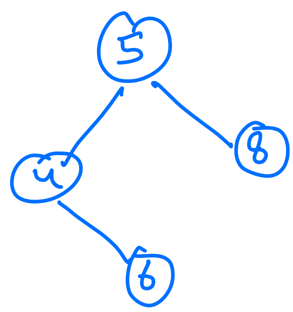
(1) BA is a binary tree :- every node has atmost two children.

(2) if $y \leq x$, then $y$ is in the left subtree rooted at $x$.

(3) if $y > x$, then $y$ is in the right subtree rooted $x$.

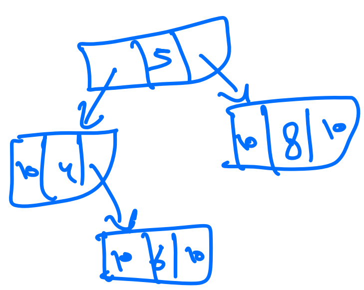
Examples :-

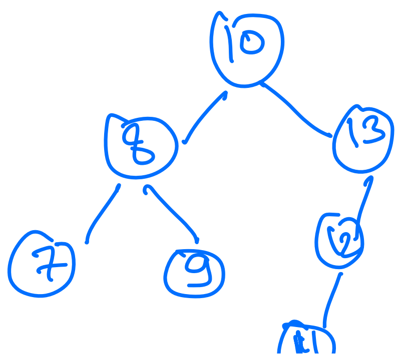
T




Is it a BST?

NO

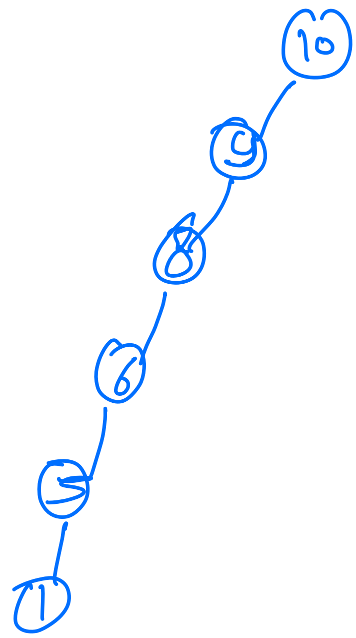


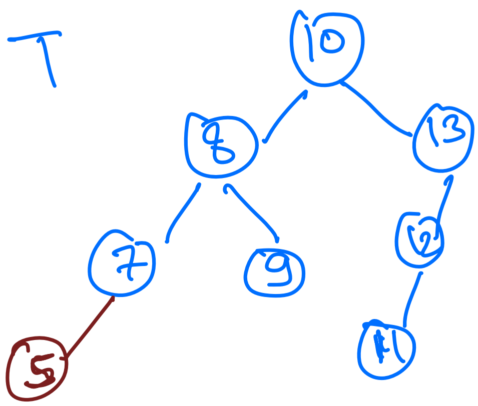

Yes

search $(T, x)$ : $O(h)$
where $h$ is the height of the tree

Running Time in worst case is $O(n)$.



T



search $(T, 5)$

If we can contro the height of the tree, then we are good!

## $\boxed{\text{Red - Black Trees}}$

1) It is a binary search tree
2) Every node is colored either red (R) or Black (B).
3) root is colored B

4) no red node can have red children

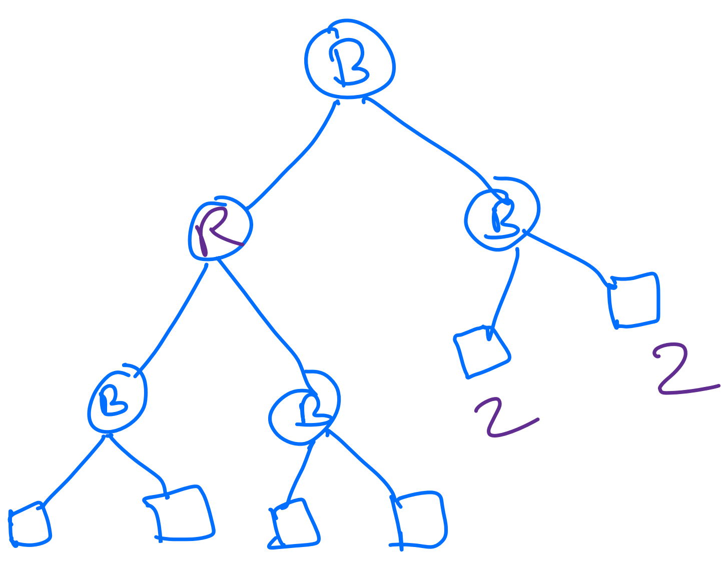
5) if a node does not have left or right child, add an external node. External nodes are not colored.

6) for all the external nodes, the path from external node to root contains the same number of black nodes.

Black Depth of external node :- number of black node in the path from external node to the root.

Black depth of all the external node is same.

Black height of tree := black depth of external node.




(1) ✓

(2) ✓

(3) ✓

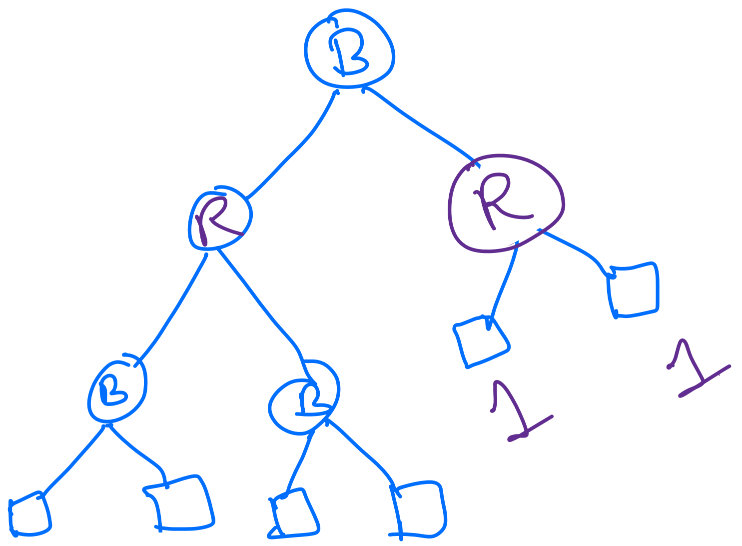
(4) ✓

(5) ✓

(6) ✓

2 2 2 2

This is a Red Black Tree




Not a Red Black Tree

2 2 2 2

**Lemma:** The height of Red Black tree is $O(\log n)$

**Intuition:-** black height $= bh$

$$n \geq 2^{bh} \Rightarrow bh \leq \log n$$

$$h \leq 2bh \Rightarrow h \leq 2\log n$$

**Formal Proof:**

**Claim:-** The subtree rooted at any node $x$ contains at least $2^{bh(x)} - 1$ internal nodes.

**Proof** ⇒ Prove it by induction on h, where h is height.

**Base Case** :- $h = 0$

$$2^{bh(x)} - 1 = 0$$

**Induction ~~Step~~ hypothesis** :- The statement is true for $h \leq j$

**IS** :- $h = j+1$

$x \rightarrow j+1$

$l_x \rightarrow$ left child of $x$

$r \rightarrow$ right child of $x$

$bh(l)$ is either $bh(x)$ or $bh(x) - 1$

# nodes in the subtree rooted at x.

$$\geq 2^{bh(x)-1} - 1 + 2^{bh(x)-1} - 1 + 1$$

$$= 2^{bh(x)} - 1$$

$$n \geq 2^{bh(x)} - 1 \quad ; \quad bh(x) \leq \log(n+1)$$

Due to property 4 of Red Black Tree

$$h \leq 2\, bh(x)$$

$$\leq 2(\log(n+1))$$

Since height of Red Black Tree is $O(\log n)$, Search, Min, Max are $O(\log n)$