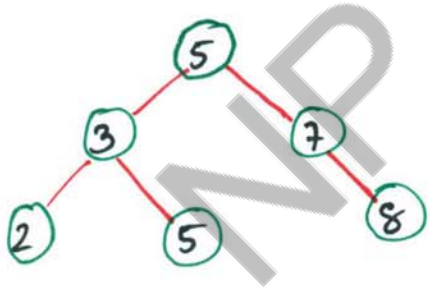
Binary Search Tree (BST) Sort

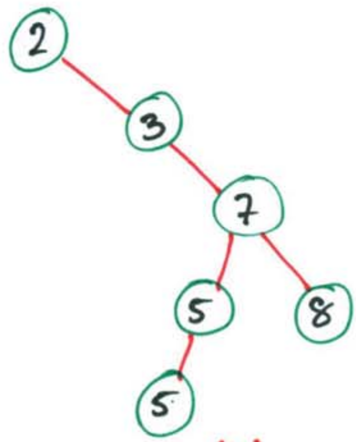
What is a binary search tree?

- A binary tree is organized in a binary tree.
- It can be represented by a linked data structure in which each node is an object.
- Each node contains a key field, satellite data, fields left, right, and p which points to its left child, right child and parent respectively.
- If child or parent is missing, the field contains NIL.
- Root is the only node whose parent field is NIL

Example:



(a)



(b)

For any node x, the keys in left subtree of x are at most key[x] and those in right subtree are at least key[x].

The worst case running time for most search-tree operations is proportional to height of tree.

(a) A BST on 6 nodes with height 2

(b) A less efficient BST on 6 nodes with height 4.

Binary search tree property:

Let x be a node in a binary search tree.

If y is a node in the left subtree of x, then

$$key[y] \le key[x]$$

If y is a node in the right subtree of x, then

$$key[x] \le key[y]$$

INORDER-TREE-WALK (root [T])

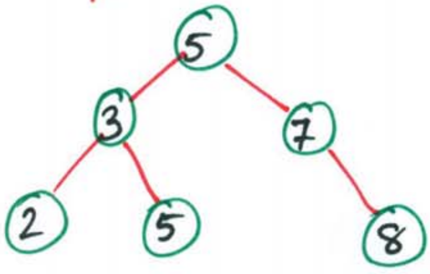
The binary search tree property allows printing all keys in a BST in sorted order by a simple algorithm, called <u>inorder tree walk</u>:

Let x be the root of BST.

```
INORDER-TREE-WALK (x)
if x ≠ NIL
   then INORDER-TREE-WALK (left[x])
        print key[x]
        INORDER-TREE-WALK (right[x])
```


Running time: $\Theta(n)$ time.

Example:



$\Rightarrow$ INORDER-TREE-WALK prints:

<u>2 3 5 5 7 8</u>

INSERTION

The operations of insertion (and deletion) cause the dynamic set represented by a binary search tree to change.

The data structure must be modified to reflect this change, but in such a way that binary search tree property holds.

```
TREE-INSERT (T, z)
    y ← NIL
    x ← root [T]
    while x ≠ NIL
          do y ← x
             if key [x] < key [x]
                then x ← left [x]
                else x ← right [x]
    p[x] ← y
    if y = NIL
       then root [T] ← z        ▸ Tree T was empty.
       else if key [x] < key [z]
               then left [y] ← z
               else right [y] ← z
```

Running Time: $O(h)$, where h = height of tree.

Binary Search Tree Sort

$T \leftarrow \phi$

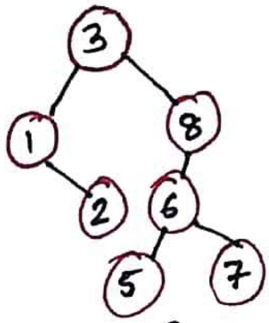
for $i=1$ to n

  do TREE-INSERT $(T, A[i])$

Perform an inorder tree walk of T

Example:

$A = [3\ 1\ 8\ 2\ 6\ 7\ 5]$



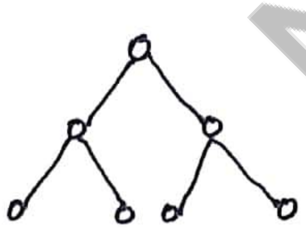
* Tree-walk time $= O(n)$

But, how long does it take to build the BST?

Time to build BST(T)

$$\text{Time} = \sum_{x \in T} \text{depth}(x)$$

Best Case

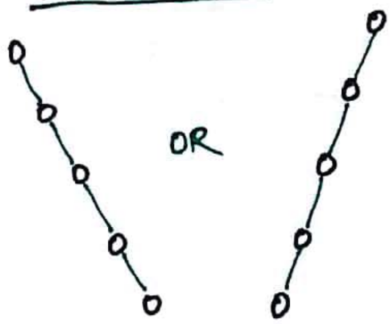


Good tree
(Balanced)

$$\text{Time} = \sum_{x \in T} \text{depth}(x)$$

$$= \Omega\ n\log n$$

Worst Case



Bad tree

$$\text{Time} = \sum_{x \in T} \text{depth}(x)$$

$$= 1+2+\cdots+n$$

$$= \frac{n(n+1)}{2}$$

$$= O(n^2)$$

Analysis of BST sort

Best case runtime = $\Omega(n \log n)$

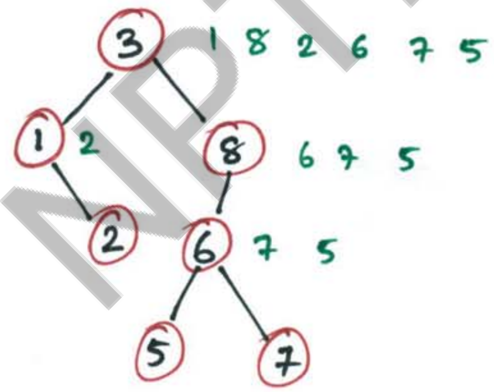
Worst case runtime = $O(n^2)$

This is same as the quicksort time complexity.

So, what is the relationship between BST sort (build BST) and quicksort?

BST sort (build BST) and Quick sort

BST sort performs the same comparisions as quick sort but in a different order.



Stable partition

The time to build the tree is asymptotically the same as the running time of quick sort.