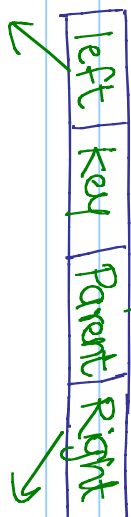


# Binary Search Trees

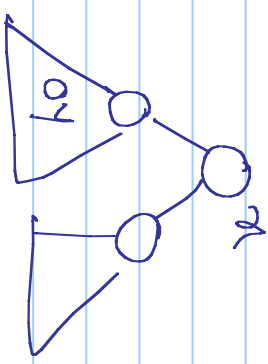
linked data structure of nodes

node  $n$

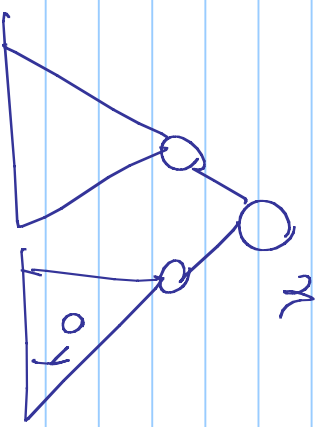


The invariant needed to be satisfied  
 $n \rightarrow$  node in BST

If  $y$ : node in left-subtree of  $n \rightarrow \text{key}(y) \leq \text{key}(n)$

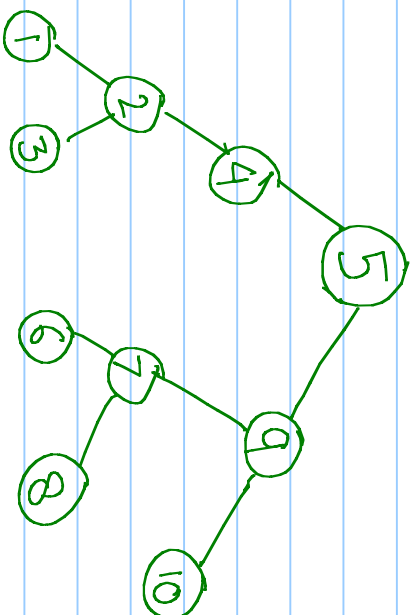


If  $y$ : node in right-subtree of  $n \rightarrow \text{key}(y) \geq \text{key}(n)$

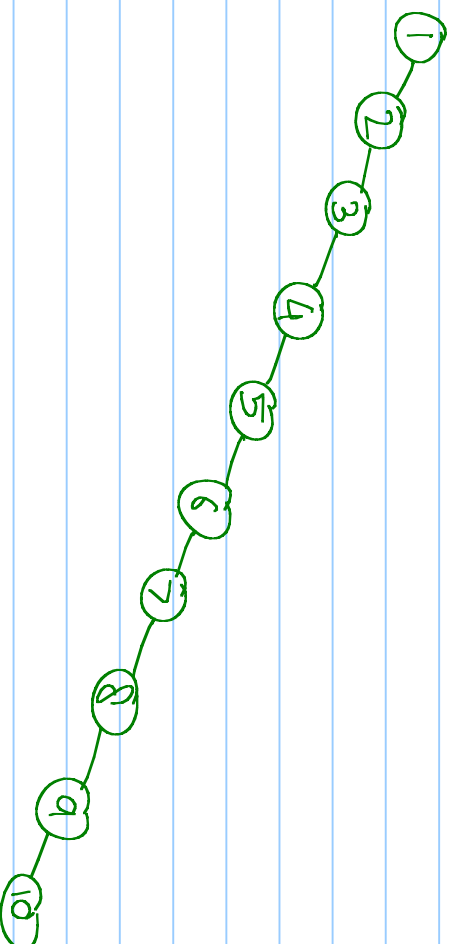


# Examples

5 9 4 7 8 2 10 1 6 3



1 2 3 4 5 6 7 8 9 10



10 1 9 2 8 3 7 4 6 5  
5 4 3 2 1 6 7 8 9 10

## Inorder walk

```
INORDER-TREE-WALK( $n$ )  
  If  $x \neq \text{NIL}$   
    Then INORDER-TREE-WALK(left[ $n$ ])  
        Print key[ $n$ ]  
        INORDER-TREE-WALK(right[ $n$ ])
```

$O(n)$   $n$ : number of nodes in the tree

## TREE-SEARCH

```
TREE-SEARCH( $n, k$ )  
  If  $n = \text{NIL}$  or key[ $n$ ] =  $k$   
    return  $n$   
  If  $k < \text{key}[n]$   
    return TREE-SEARCH(left[ $n$ ],  $k$ )  
  else  
    return TREE-SEARCH(right[ $n$ ],  $k$ )
```

$O(h)$ ,  $h$ : height of the tree

```
ITERATIVE TREE-SEARCH  
  while  $n \neq \text{NIL}$  and  $k \neq \text{key}[n]$   
    if  $k < \text{key}[n]$   
       $n = \text{left}[n]$   
    else  
       $n = \text{right}[n]$   
  return  $n$ 
```

## TREE - MINIMUM

```
TREE-MINIMUM(x)
while left(x) ≠ NIL
  x = left(x)
return x
```

## TREE - MAXIMUM

```
TREE-MAXIMUM(x)
while right(x) ≠ NIL
  x = right(x)
return x
```

## TREE - SUCCESSOR

TREE-SUCCESSOR(x)

```
if right(x) ≠ NIL
  return TREE-MINIMUM(right(x))
y = p(x)
while y ≠ NIL and x = right(y)
  x = y
  y = p(y)
return y
```

# TREE-INSERT

TREE-INSERT (T, z)

y = NIL

r = root(T)

while r ≠ NIL

y = r

if key[z] < key[r]

r = left[r]

else

r = right[r]

if z = y

if y = NIL

root[T] = z

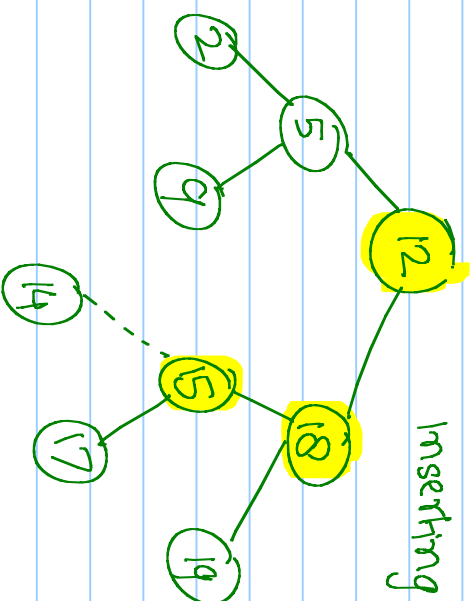
else if key[z] < key[y]

left[y] = z

else

right[y] = z

} Find the correct location to insert z



# TREE - DELETE

TREE-DELETE (T, z)

IF left[z] = NIL or right[z] = NIL

y = z

else

y = TREE-SUCCESSOR(z)

IF left[y] ≠ NIL

x = left[y]

else

x = right[y]

IF x ≠ NIL

P[x] = P[y]

∧ P[y] = NIL

root [T] = x

else if y = left [P[y]]

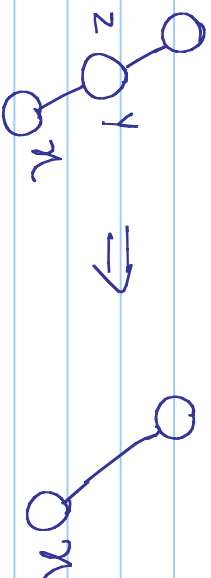
left [P[y]] = x

else right [P[y]] = x

∧ y ≠ z key [z] = key [y]

return y

a) If z has no children, delete it



b) if z has 1 child, splice out z

c) if z has both children, find its successor, splice it and copy key value

