Binary Search Trees (BST)

Last lecture binary search Elements in an array (sorted)

$O(\log n)$

Insertion/deletion of elements in a sorted array

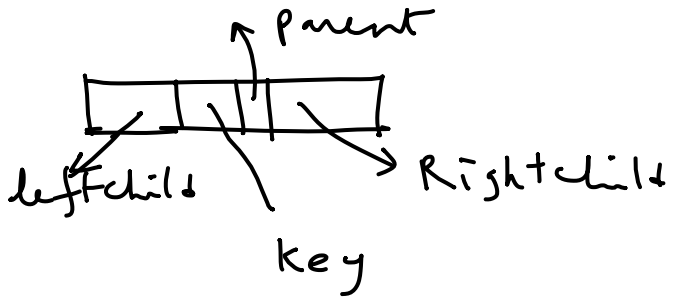
requires $O(n)$ time

[because elements of array may need to be shifted one position to their left or right]

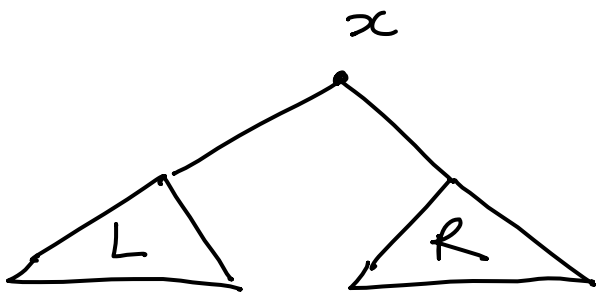
BST

binary trees

data is stored at each node



For any node in BST

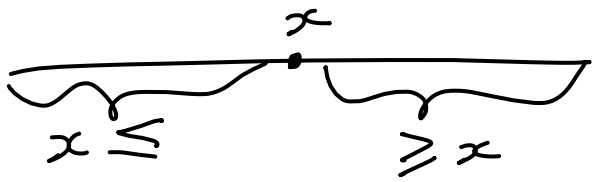


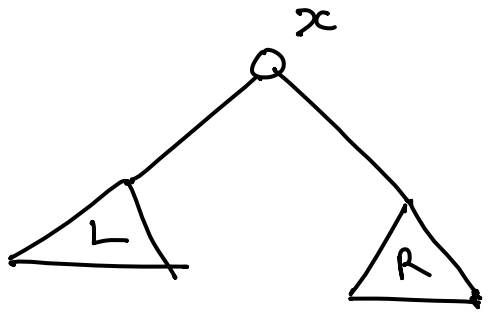
For any y in L

$y.key \leq x.key$

For any $z \in R$

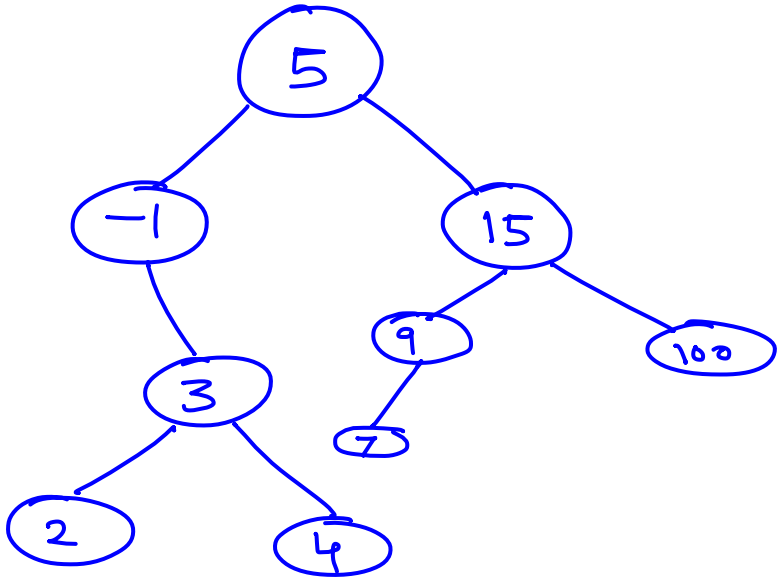
$z.key \geq x.key$





$L \leq x \leq R$

Example



-1, 2, 3, 4, 5, 7, 9, 15, 100

Inorder traversal

Searching an element in a BST

Input x / node in a tree

k / From the domain of keys

Output: If there is some node with key k in tree rooted at x then y (s.t. $y.key = k$)

Otherwise (not node in key k in tree rooted at x)

output is 'nil'

```
Search (x, k)
If x == nil then return x
If x.key == k then return x
If k < x.key then search(x.left, k)
             else search(x.right, k)
```

Recursive procedure

Time complexity: $O(h)$

h is height of x

Proof using substitution method

Correctness: By induction on number of nodes in the tree rooted at x

Exercise

Search-iter(x, k)

Correctness (T is input tree)

While $x \neq nil$ and $x.key \neq k$ do ← $k \in T$ iff $k \in x$
  If $k < x.key$ then $x = x.left$
  else $x = x.right$

return x

Time Complexity: No. of iterations of while loop $\leq$ height of x

$O(h)$

In BST, Finding minimum, finding maximum or Computing successor/predecessor can also be done efficiently.

In BST inorder traversal produces a sorted list.

```
Min(x)    // returns the node containing minimum element in the tree rooted at x
          // if no such element then returns nil.

if x == nil then return x

if x.left == nil then return x
              else min(x.left)
```

Correctness : Exercise

Complexity $O(h)$ h is the height of x

Correctness

```
min-iter(x)
if x == nil then return x
While x.left ≠ nil do
      x = x.left
return x
```

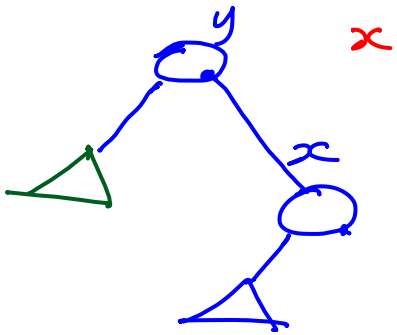
Invariant: Input.min = x.min

Time Complexity : $O(h)$

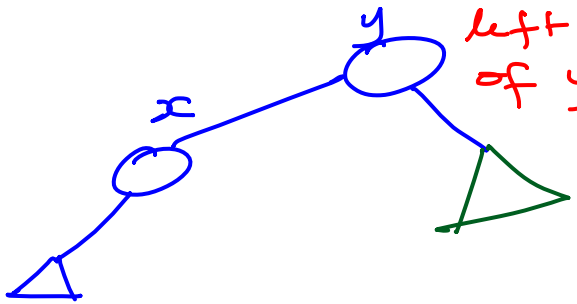
Exercise : Write pseudo-code for finding maximum element in a BST.

Succ(x) // x is a node in tree T
succ x is the node to be visited just after visiting x in inorder traversal of T.
If there is no successor of x (x is the last node) then nil

Succ(x)
If x.right ≠ nil then return min(x.right)
y = x.p
If y == nil return nil
If x = y.left then return y
else return succ(y)



x is the right child of its parent



x is the left child of y

Correctness: Exercise

Time complexity $O(h_T)$

Time complexity

$O(h_T)$

```
Succ-iter(x)
If x.right ≠ nil then return min(x.right)
y = x.p
while y ≠ nil and x = y.right do
        x = y
        y = x.p
return y
```

Correctness

think of a suitable invariant

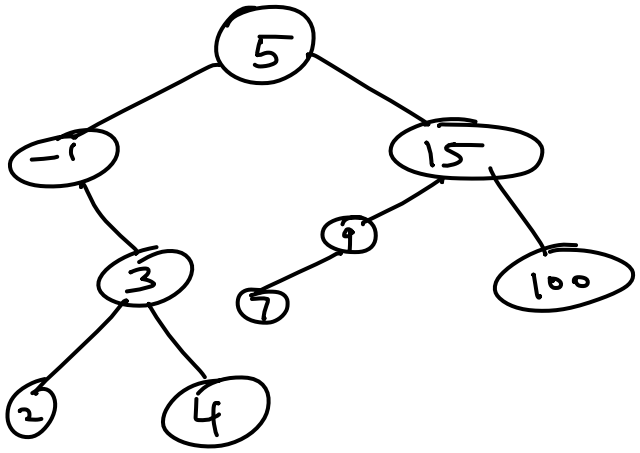
succ-iter(4)

$x = 4, \quad y = 3$

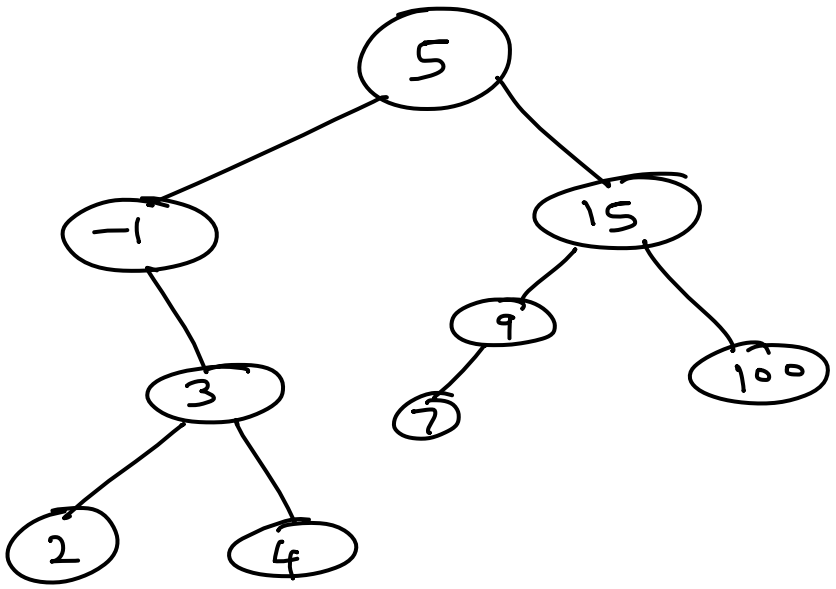
$x = 3, \quad y = -1$

$x = -1, \quad y = 5$

return 5



[Abuse of notation: nodes are being identified by their key values, for convenience.]



succ-iter(100)

$x = 100, \quad y = 15$

$x = 15, \quad y = 5$

$x = 5, \quad y = nil$

return y