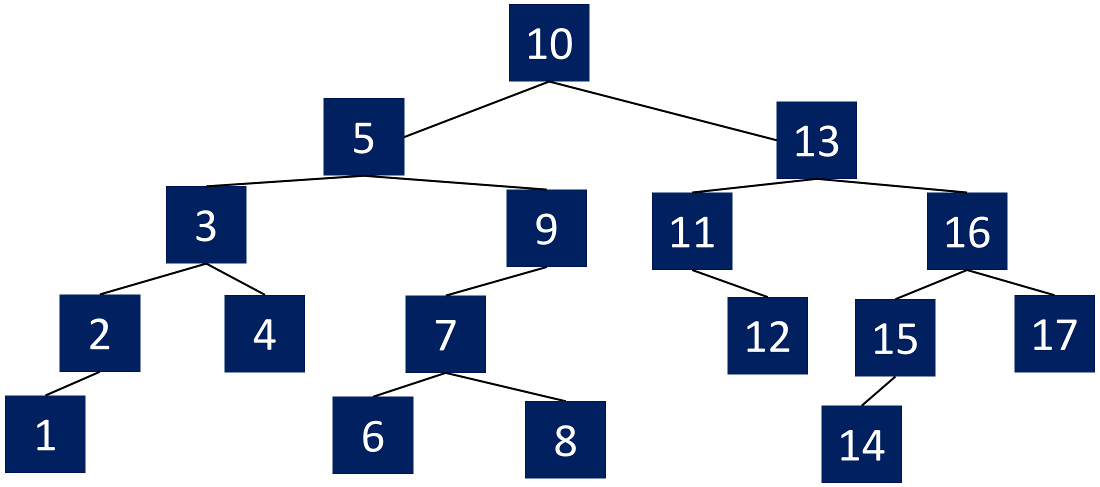
Further Binary Search Tree Revision

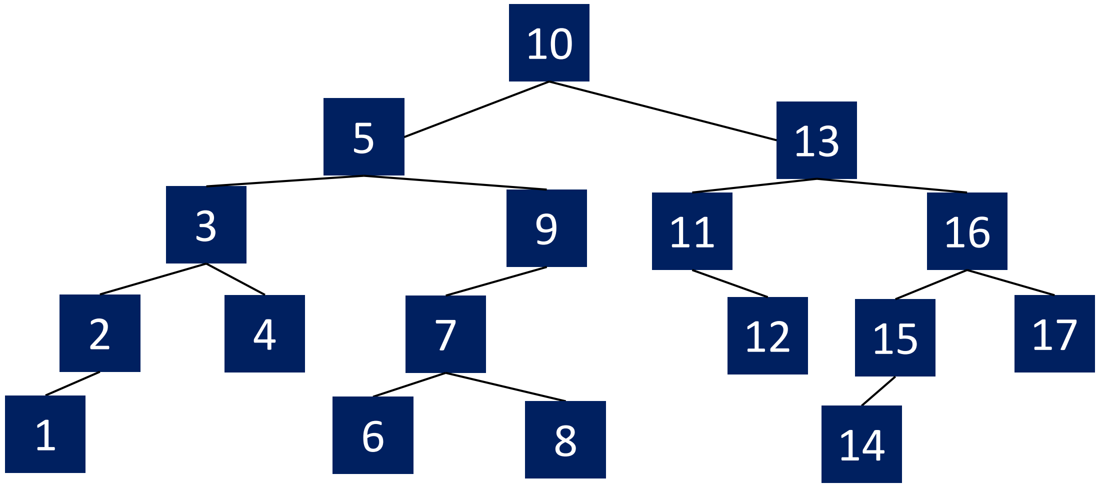
Version of March April 3, 2019

- Recall that a Binary Search Tree is designed so that
  - A node x contains a KEY K_x
  - All keys in the Left subtree hanging off of x have value $< K_x$
  - All keys in the Right subtree hanging off of x have value $> K_x$



To search for an item K in a tree rooted at node x compares K to the key K_x

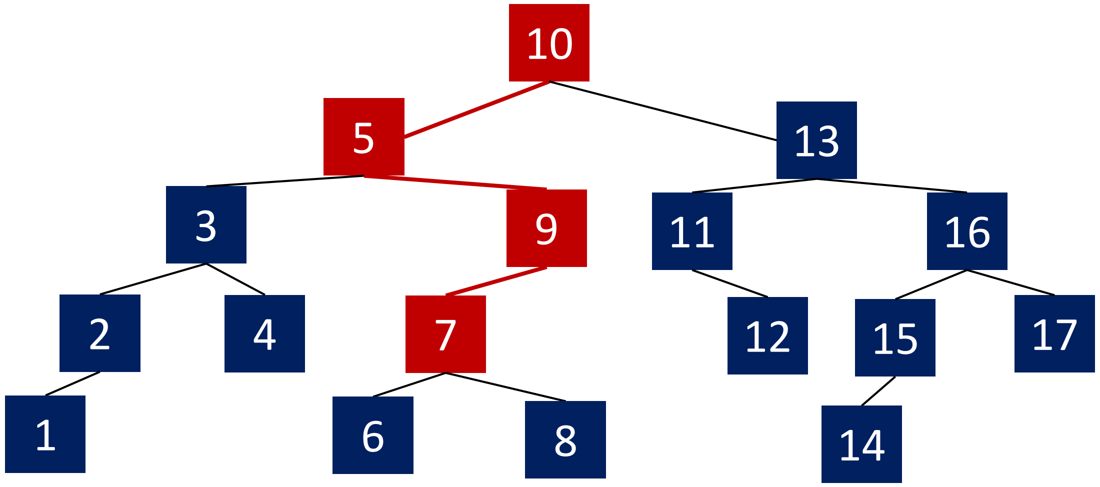
- If $K = K_x$ then stop (success)
- If $K < K_x$ search for K in the subtree rooted at the left child of x.
- If $K > K_x$ search for K in the subtree rooted at the right child of x.



To search for an item K in a tree rooted at node x compares K to the key K_x

- If $K = K_x$ then stop (success)
- If $K < K_x$ search for K in the subtree rooted at the left child of x.
- If $K > K_x$ search for K in the subtree rooted at the right child of x.

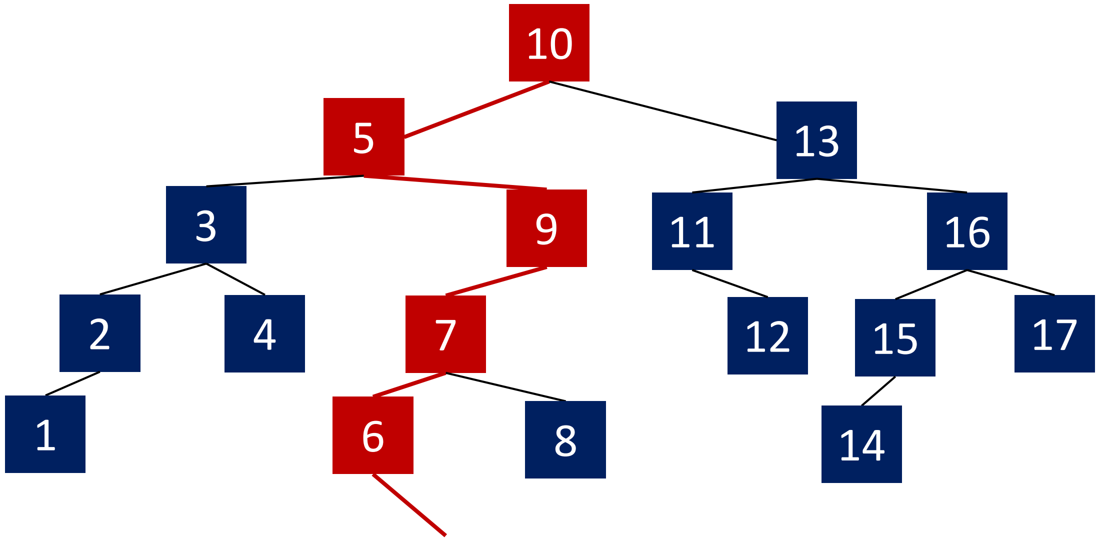
Searching for K = 7



To search for an item K in a tree rooted at node x compares K to the key K_x

- If $K = K_x$ then stop (success)
- If $K < K_x$ search for K in the subtree rooted at the left child of x.
- If $K > K_x$ search for K in the subtree rooted at the right child of x.

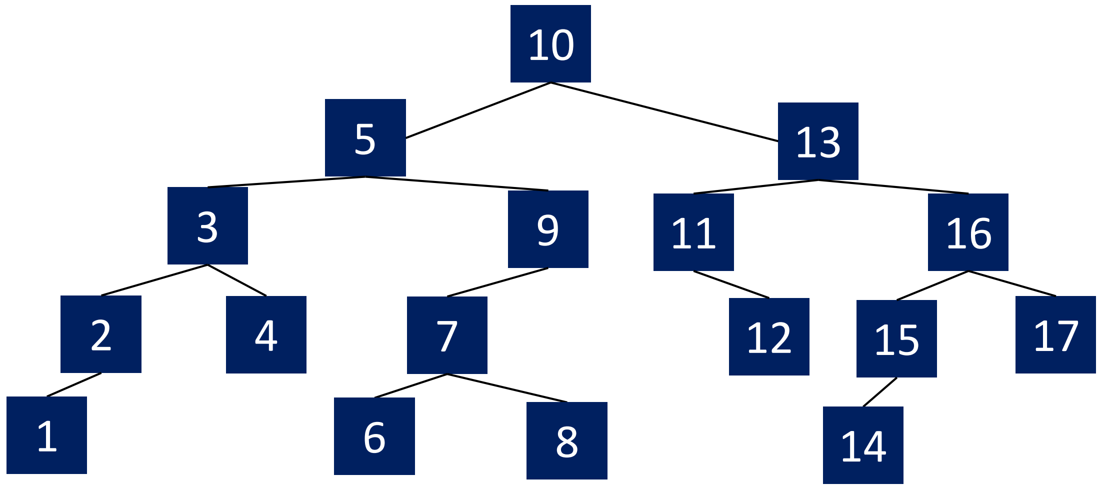
Searching for K = 6.5



Unsuccessful searches:

The search will discover that K is not in the tree

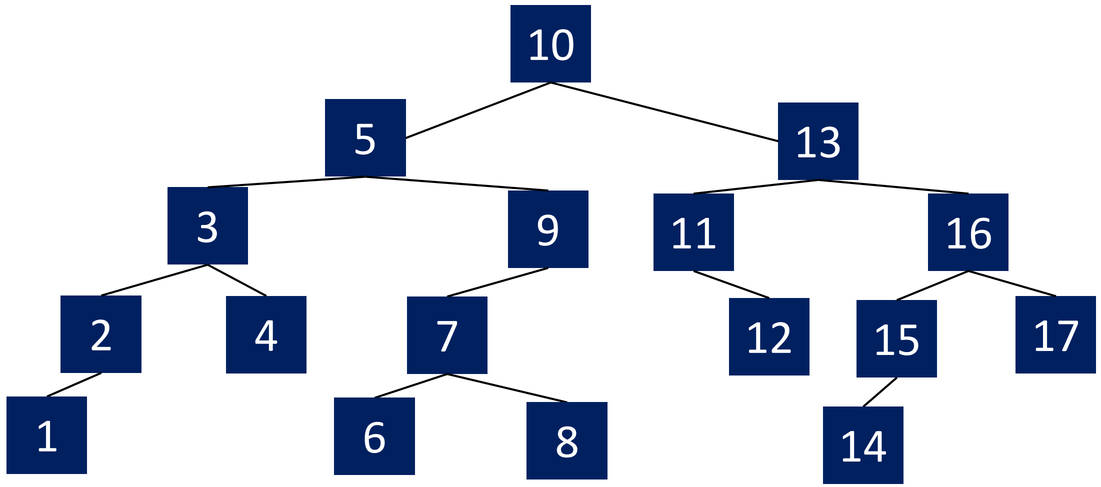
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x. x is assumed to be in S.
- **Succ(K):** Find the successor in S of x. x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

All operations can be implemented in time $O(h)$, where h is the height of the tree.

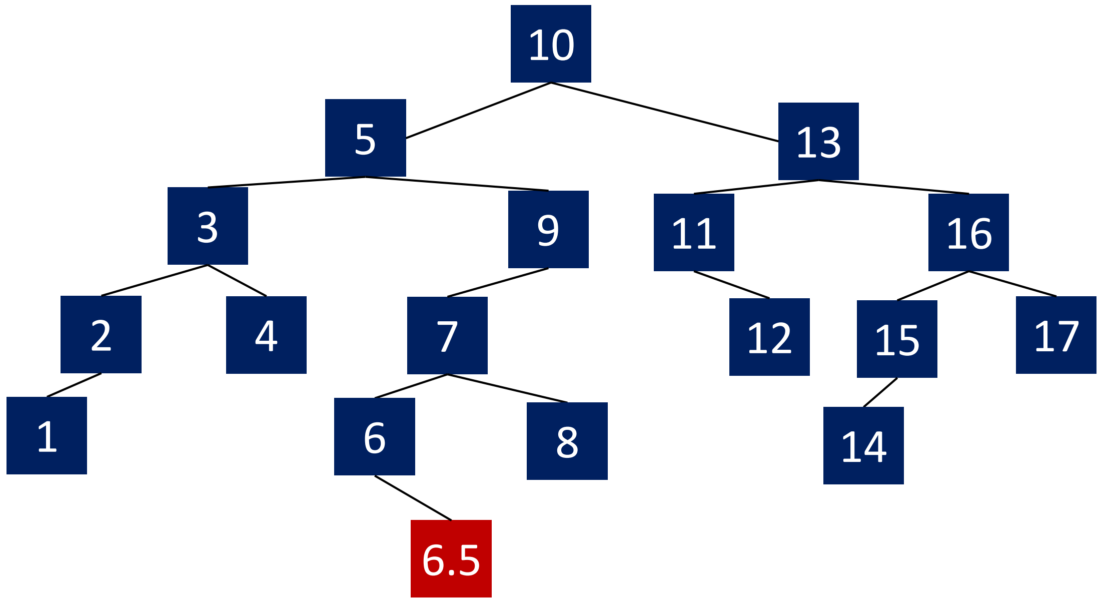
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x.
  x is assumed to be in S.
- **Succ(K):** Find the successor in S of x.
  x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

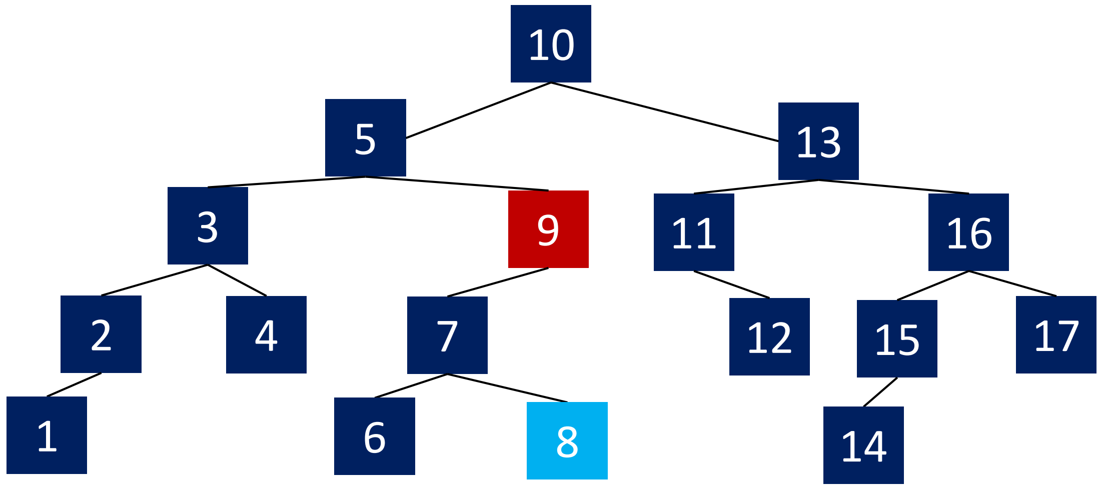
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S

- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x. x is assumed to be in S.
- **Succ(K):** Find the successor in S of x. x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.



Insert(6.5)

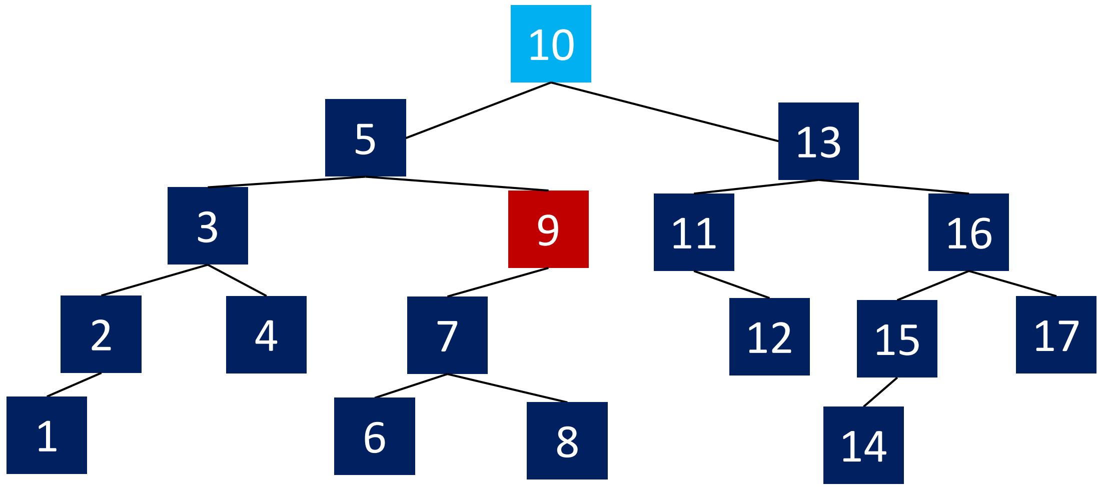
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x. x is assumed to be in S.
- **Succ(K):** Find the successor in S of x. x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

Pred(9) = 8

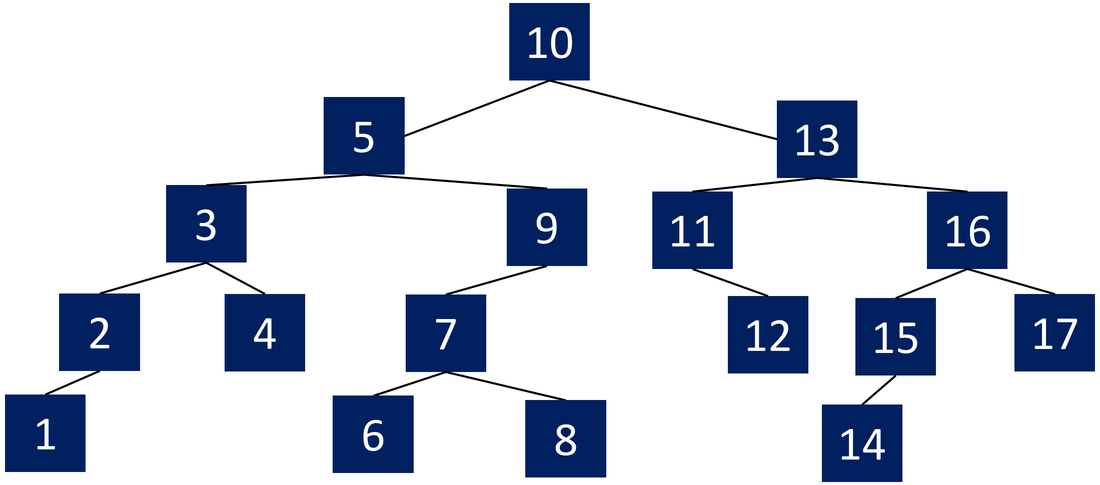
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x. x is assumed to be in S.
- **Succ(K):** Find the successor in S of x. x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

Succ(9) = 10

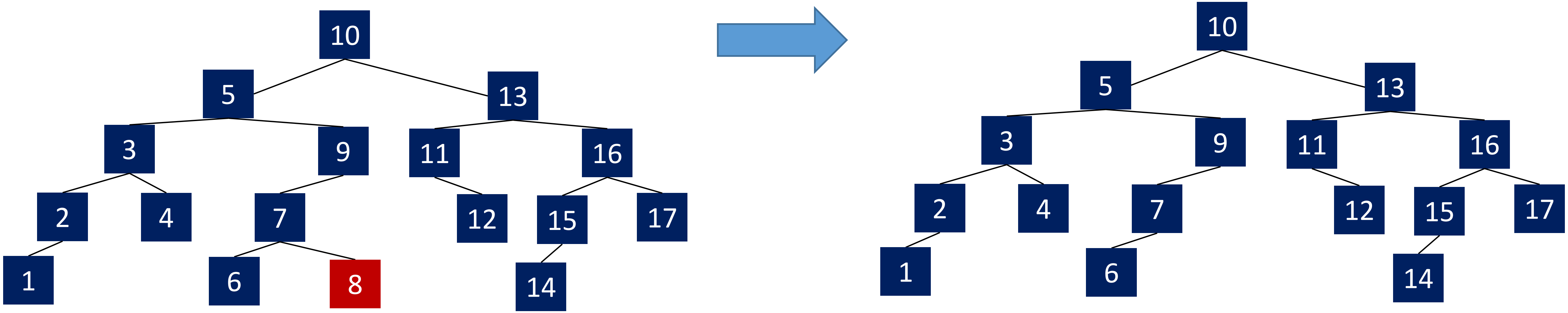
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x. x is assumed to be in S.
- **Succ(K):** Find the successor in S of x. x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

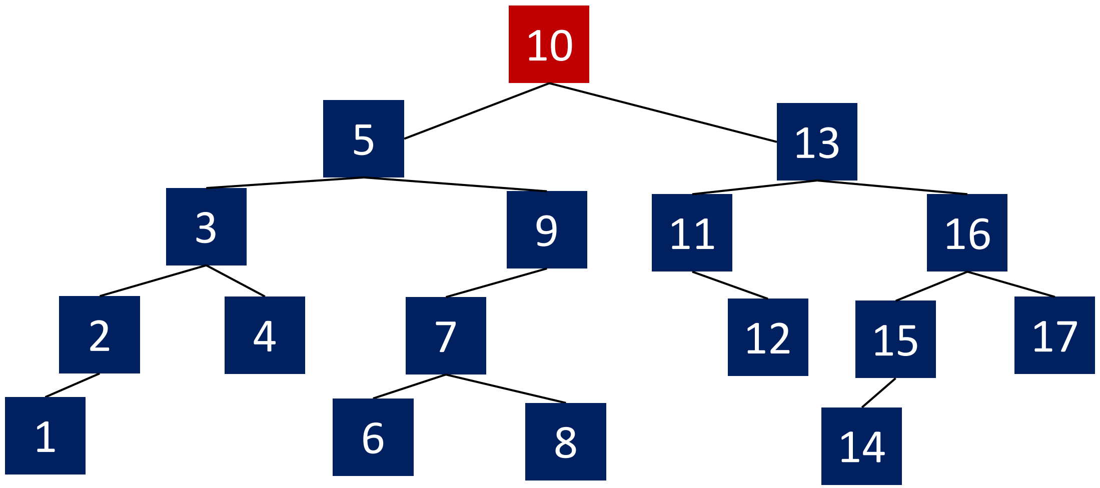
Delete(8), Delete (10)

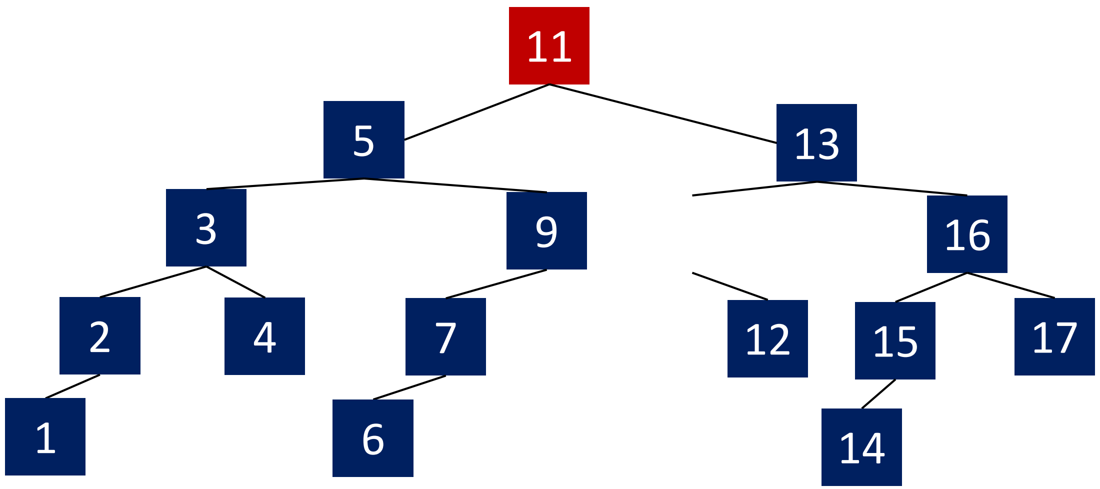
Delete(8)

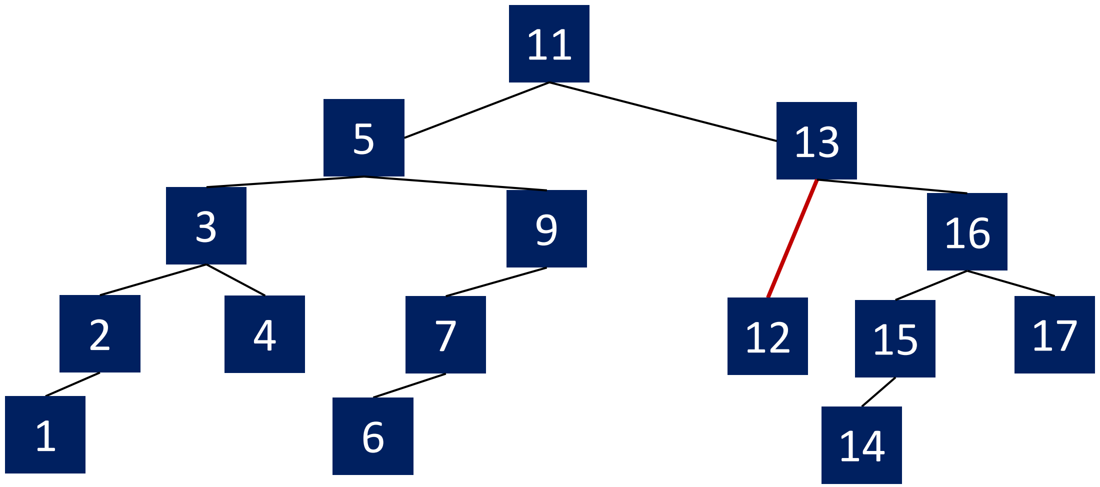


Delete(10)

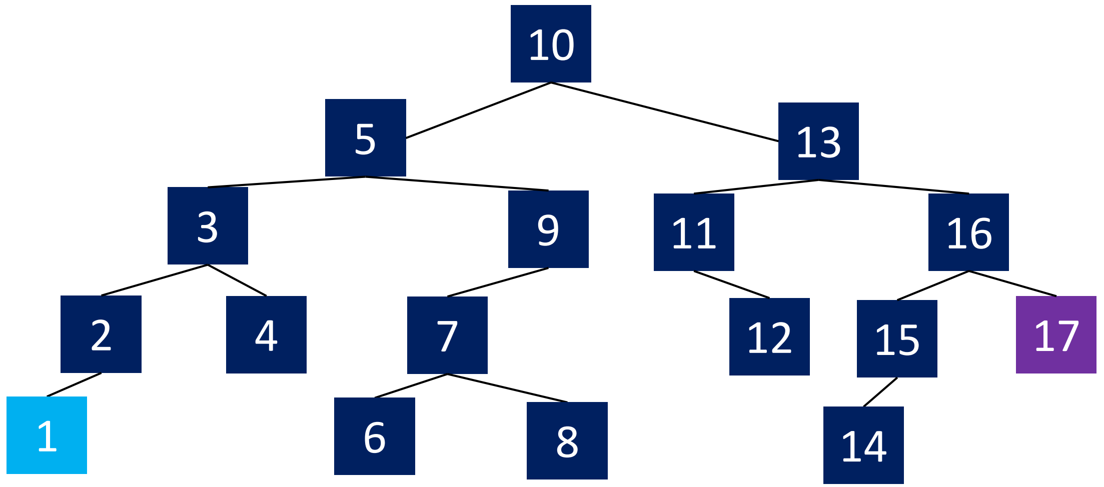
Replace (10) with Succ(10)







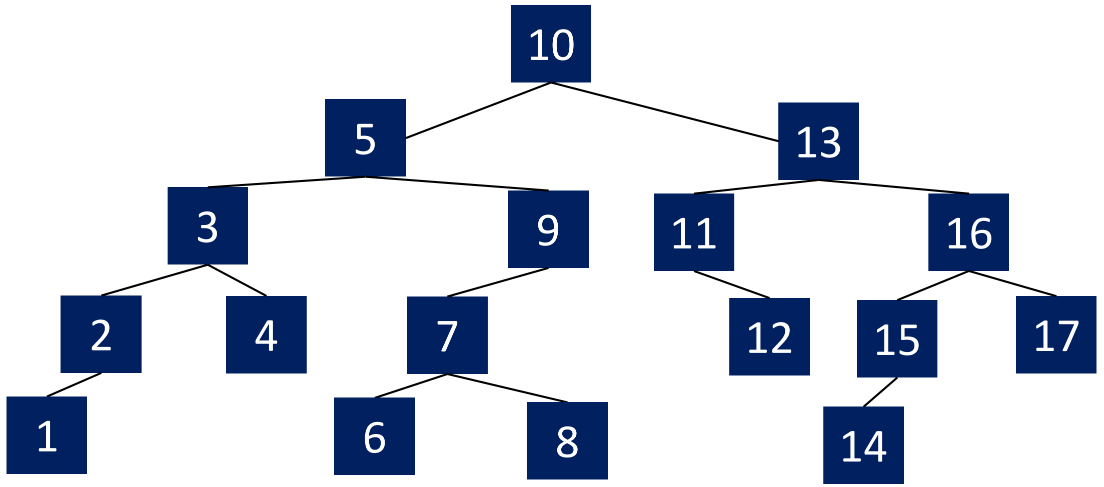
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x.
  x is assumed to be in S.
- **Succ(K):** Find the successor in S of x.
  x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

MIN is leftmost item in tree.
MAX is rightmost item in tree.

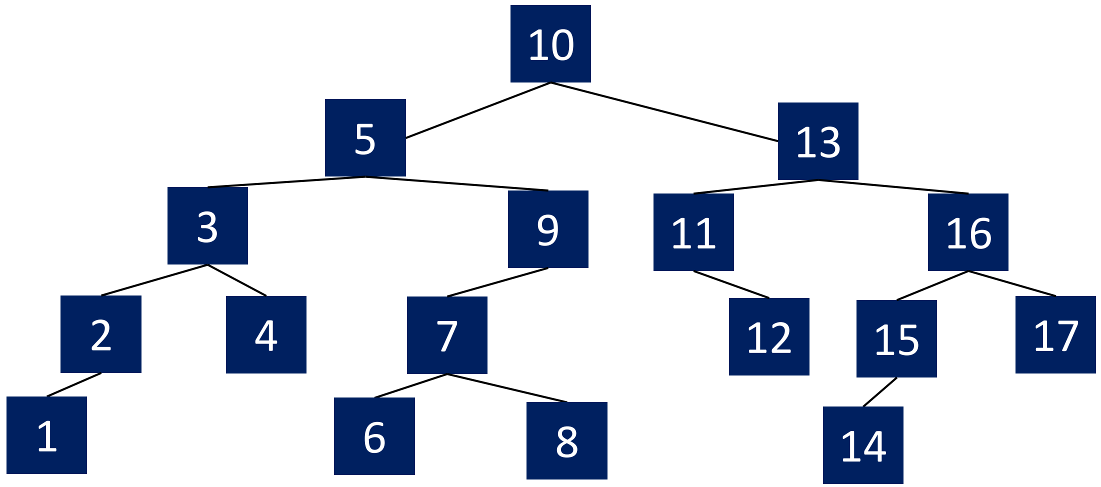
Binary Trees Support the following Dictionary Operations on a dynamic ordered set S



- **Search(K):** Find if $x \in S$
- **Insert(K):** Add x to S
- **Delete(K):** Delete x from S
- **Pred(K):** Find the predecessor in S of x. x is assumed to be in S.
- **Succ(K):** Find the successor in S of x. x is assumed to be in S.
- **MIN & MAX:** Finding the minimum and maximum items in S.

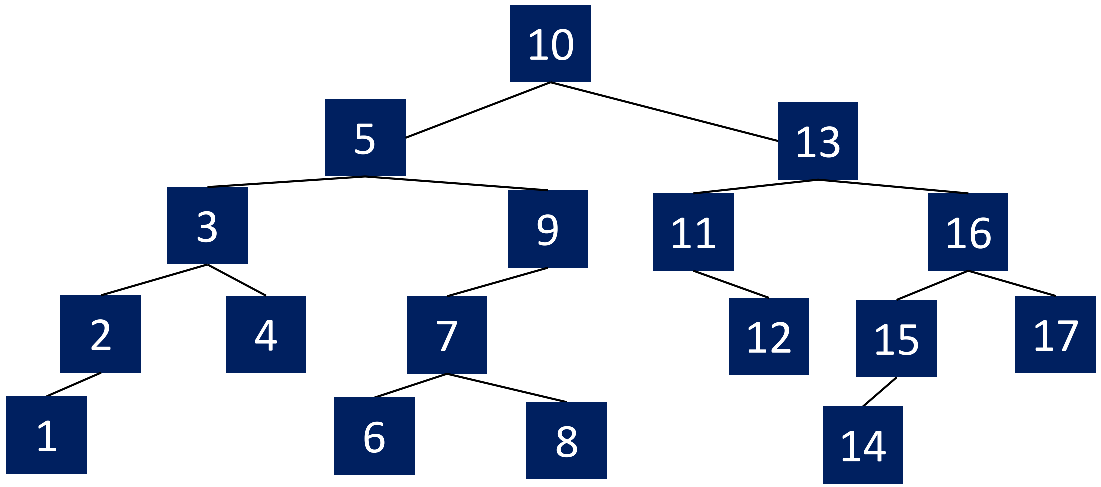
All operations can be implemented in time $O(h)$, where h is the height of the tree.

There are three standard walks on Trees, starting from node *x.*



- **Inorder(x):**
  - Inorder (Left[x]); Print key[x]; Inorder(Right[x]).
- **Preorder(x):**
  - Print key[x]; Preorder (Left[x]); Preorder(Right[x]).
- **Postorder(x):**
  - Postorder (Left[x]); Postorder(Right[x]); Print key[x];

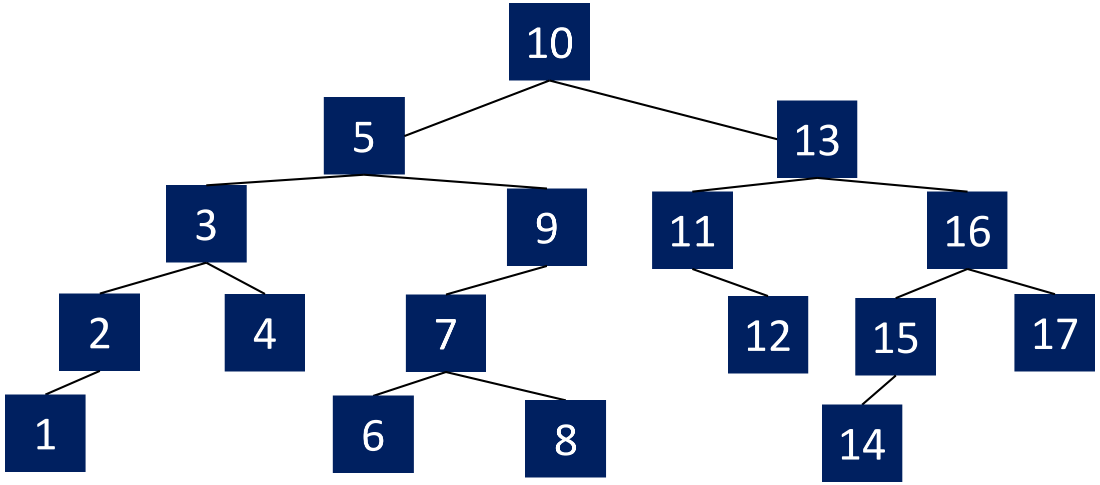
There are three standard walks on Trees, starting from node *x.*



- **Inorder(x):**
  - Inorder (Left[x]); Print key[x]; Inorder(Right[x]).

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17

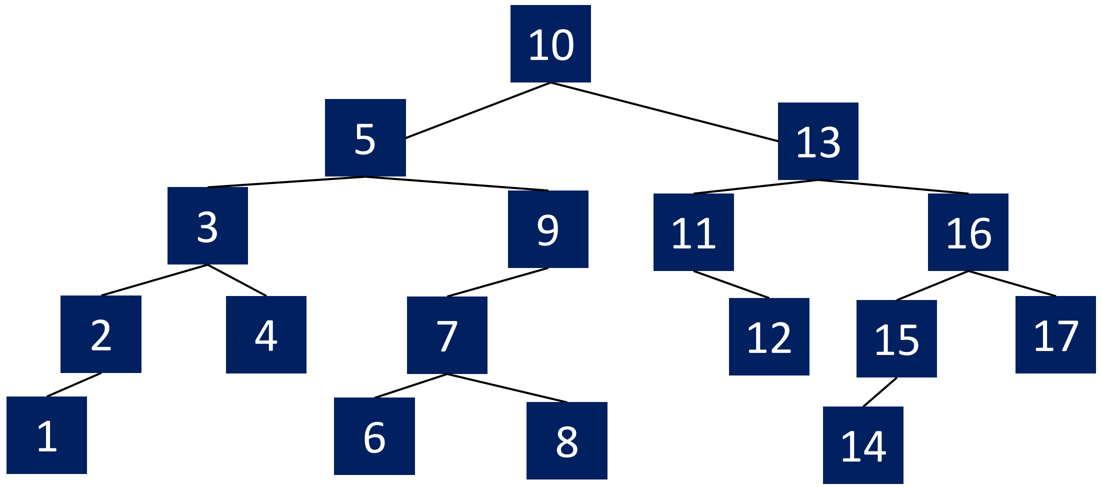
There are three standard walks on Trees, starting from node *x.*



- **Preorder(x):**
  - Print key[x]; Preorder (Left[x]); Preorder(Right[x]).

10, 5, 3, 2, 1, 4, 9, 7, 6, 8, 13, 11, 12, 16, 15, 14, 17

There are three standard walks on Trees, starting from node *x.*



- **Postorder(x):**
  - Postorder (Left[x]); Postorder(Right[x]); Print key[x];

1, 2, 4, 3, 6, 8, 7, 9, 5, 12, 11, 14, 15, 17, 16, 13, 10