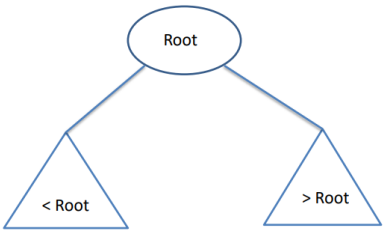
Binary Search Trees:

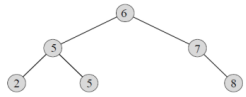
```
1    BST-Insert(T, z)
2      y := NIL
3      x := T.root
4      while x ≠ NIL do
5        y := x
6        if z.key < x.key then
7          x := x.left
8        else
9          x := x.right
10       end if
11     repeat
12     z.parent := y
13     if y = NIL then
14       T.root := z
15     else if z.key < y.key then
16       y.left := z
17     else
18       y.right := z
19     end if
```

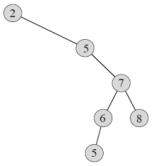
If median is placed at the root then bst would be balanced



Algorithm	Balanced Average	Skewed Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

Let x be a node in a binary search tree. If y is a node in the left subtree of x, then y.key <= x.key. If y is a node in the right subtree of x, then y.key >=x.key.





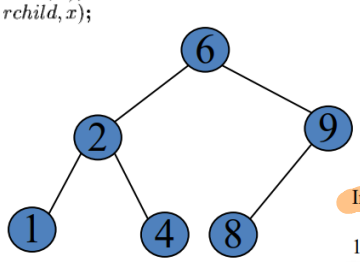
```
Algorithm Search(t, x)
{
    if (t = 0) then return 0;
    else if (x = t → data) then return t;
         else if (x < t → data) then
                  return Search(t → lchild, x);
              else return Search(t → rchild, x);
}
```

right most leaf node in right-subtree

```
BST-Maximum(x)
  while x.right ≠ NIL then
    x := x.right
  repeat
  return x
```

left most leaf node in left-subtree

```
BST-Minimum(x)
  while x.left ≠ NIL then
    x := x.left
  repeat
  return x
```

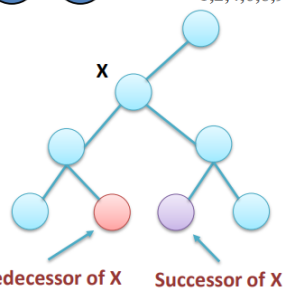


Inorder Traversal (Left, root, right)

1,2,4,6,8,9 ➔ Sorted !

```
BST-Successor(x)
  if x.right ≠ NIL then
    return BST-Minimum(x.right)
  end if
  y := x.parent
  while y ≠ NIL and x = y.right then
    x := y
    y := y.parent
  repeat
  return y
```

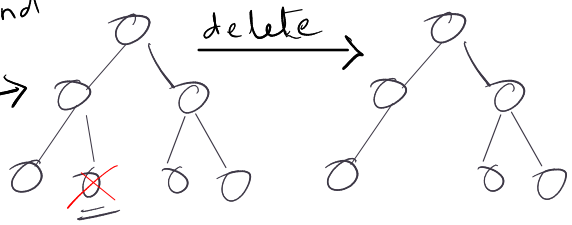
```
BST-Predecessor(x)
  if x.left ≠ NIL then
    return BST-Maximum(x.left)
  end if
  y := x.parent
  while y ≠ NIL and x = y.left then
    x := y
    y := y.parent
  repeat
  return y
```



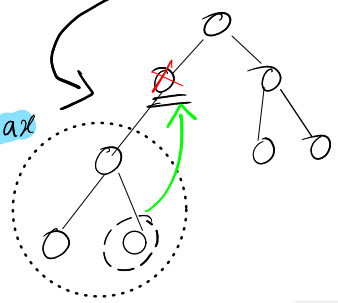
- Deletion involves three cases
  - Node has no children (leaf)
  - Node has one child
  - Node has two children

even after deletion bst property should stand

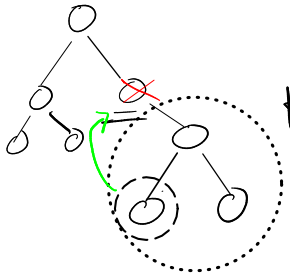
direct delete



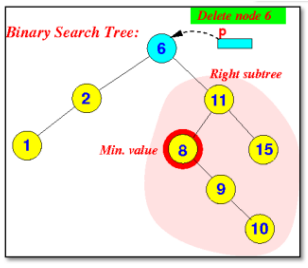
do either one of below two



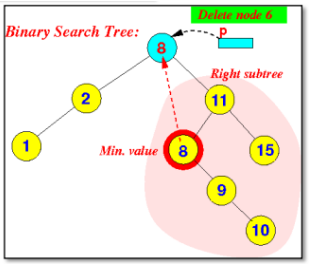
find Max in left subtree



find Min in right subtree



4




```
struct Node* Delete(struct Node *root, int data) {
    if(root == NULL) return root;
    else if(data < root->data) root->left = Delete(root->left,data);
    else if (data > root->data) root->right = Delete(root->right,data);
    else {// Wohoo... I found you, Get ready to be deleted
        if(root->left == NULL && root->right == NULL) { // Case 1:  No child
            delete root;
            root = NULL;
        }
        //Case 2: One child
        else if(root->left == NULL) {
            struct Node *temp = root;
            root = root->right;
            delete temp;
        }
        else if(root->right == NULL) {
            struct Node *temp = root;
            root = root->left;
            delete temp;
        }
        else { // case 3: 2 children
            struct Node *temp = FindMin(root->right);
            root->data = temp->data;
            root->right = Delete(root->right,temp->data);
        }
    }
    return root;
}
```

links:

https://en.wikipedia.org/wiki/Binary_search_tree

https://visualgo.net/en/bst

https://www.youtube.com/watch?v=pYT9F8_LFTM

https://www.youtube.com/watch?v=gcULXE7ViZw (deletion)