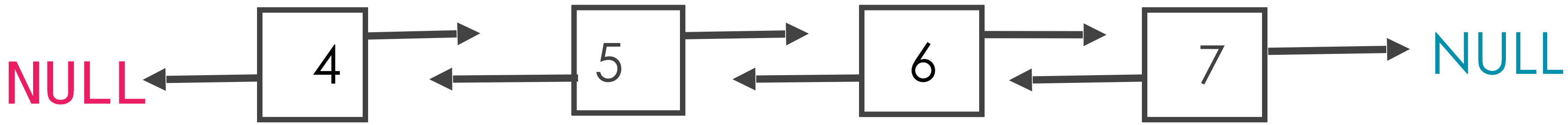
Lecture 9

Binary Search Tree

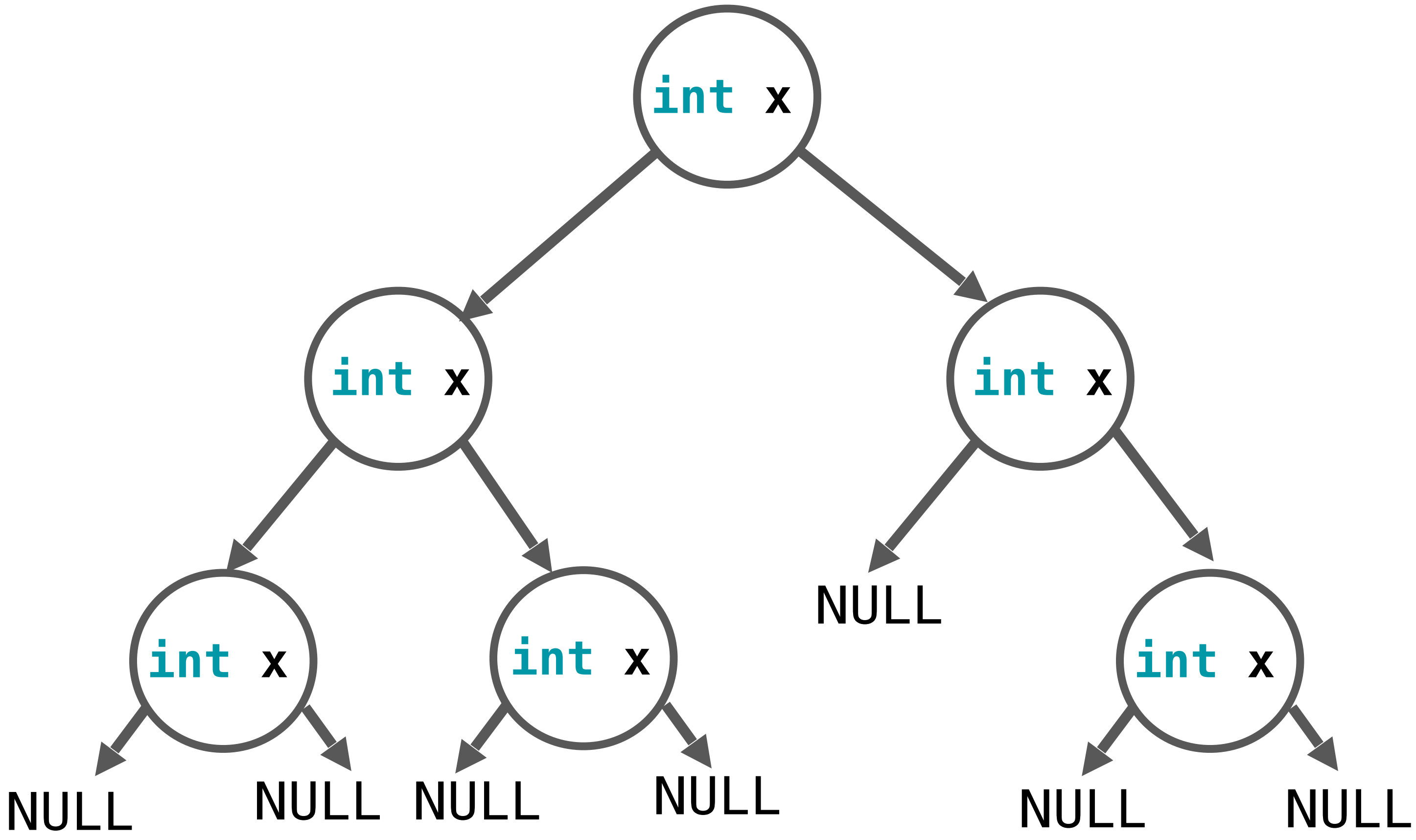
Binary Search Tree

```
struct node{
  int data;
  node* left;
  node* right;
};
```

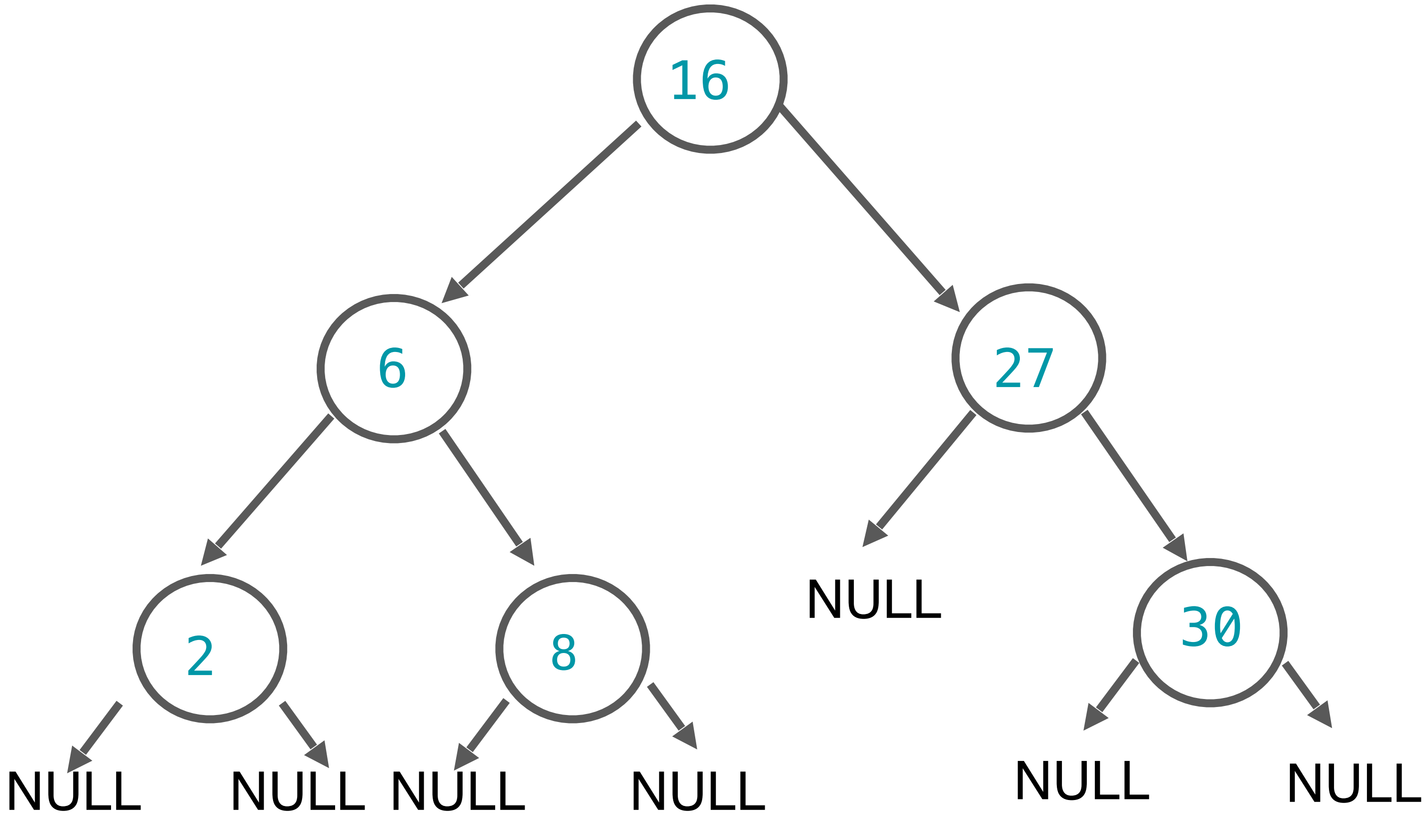
Doubly Linked List



Binary Search Tree



Binary Search Tree

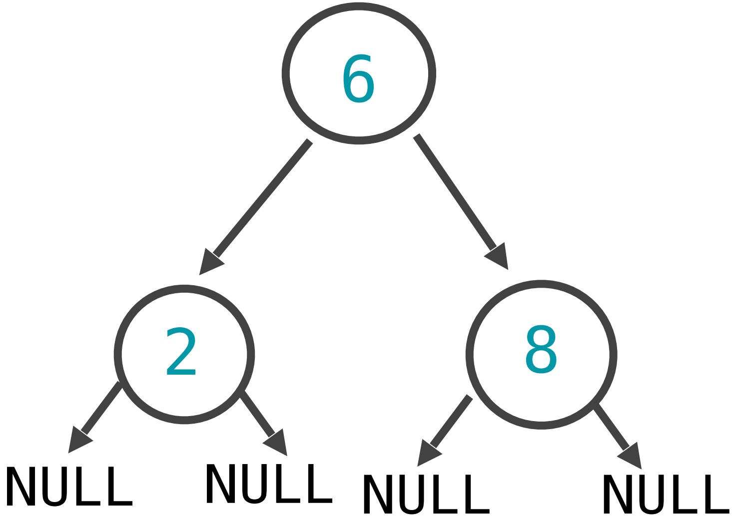


Dictionary Operations

Insert an element
Find an element
Remove an element

Insert an element

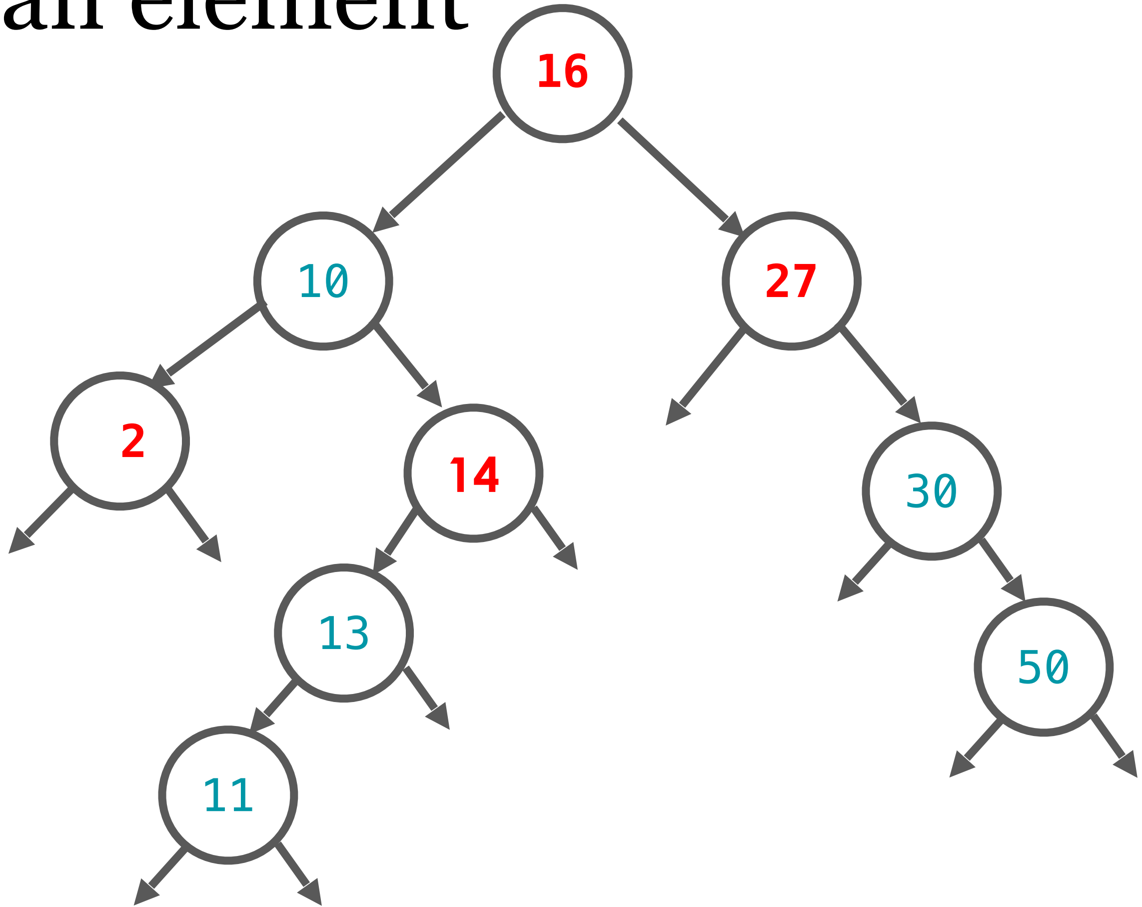
```
void insert(node *& n, int i){
        if(n == null){
          n = new node;
          n->value = i;
          n->left = n->right = null;
    }
    else if(i > n->value) insert(n->right,i);
    else if(i < n->value) insert(n->left,i);
}
```



Find an element

```
bool find(node* &n, int i){
      if(n==NULL) return false;
      else if(n->value==i) return true;
      else if(i > n->value) return find(n->right,i);
      else if(i < n->value) return find(n->left,i);
}
```

Remove an element

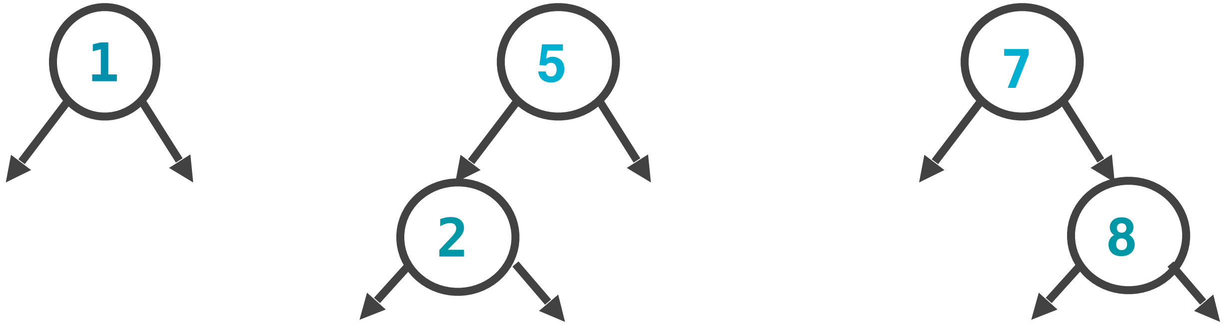


Remove an element

```
void removeElement(node *& n, int i){
  if(n==NULL) {return;}
  else if(i>n->value)removeElement(n->right,i);
  else if(i<n->value)removeElement(n->left,i);
  . . .
```

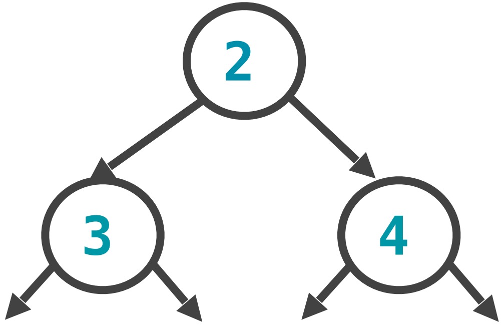
Remove an element

```
...
    else {
        if(n->left==NULL && n->right==NULL) n=NULL;
        else if(n->right==NULL) n=n->left;
        else if(n->left==NULL) n=n->right;
        ...
```

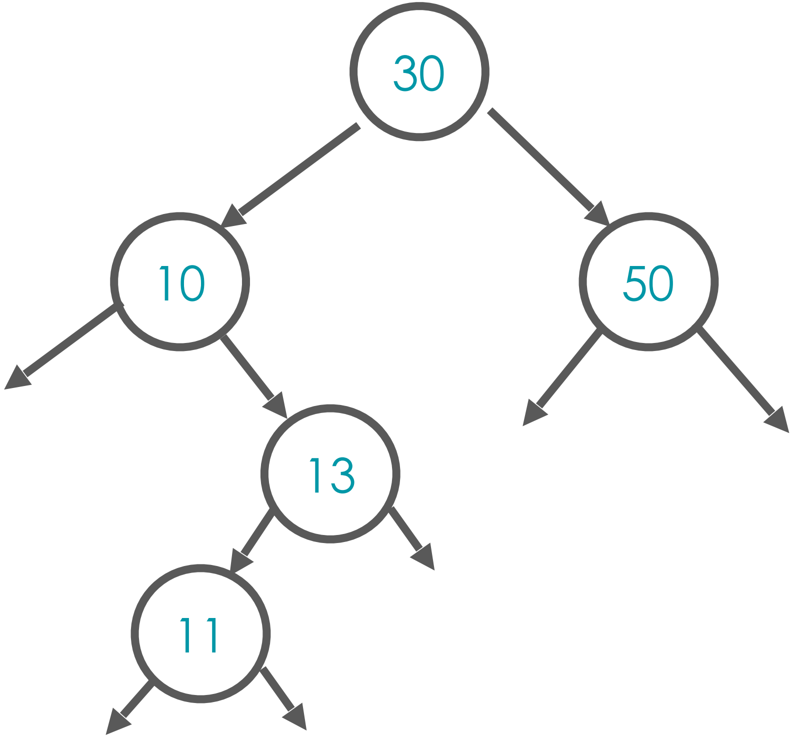


Remove an element

```
. . .
      else{
            int minVal = getmin(n->right);
            n->val = minVal;
            removeElement(n->right,minVal);
          }
      }
}
```



Remove an element



Example of trees

a. Banyan tree.

b. Binary expression tree.

c. Unix directory structure

d. Family tree.

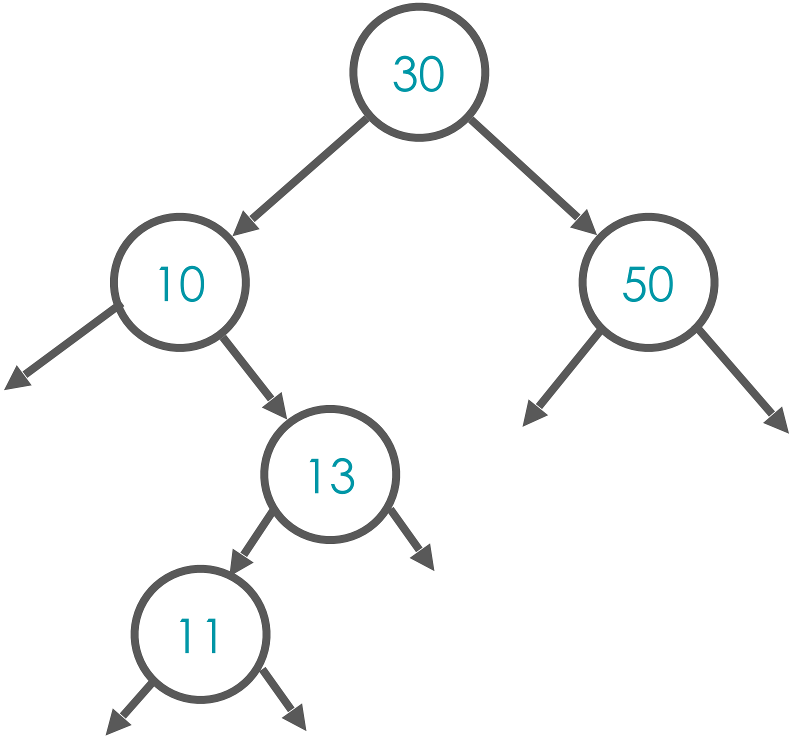
e. Binary Search Tree.

Definition of binary search tree

Binary search tree consists of number of nodes.
Each node contains a value and zero, one or two children.

All the values in a nodes left subtree is smaller than the nodes value and all the values in a nodes right subtree is greater than the nodes value.

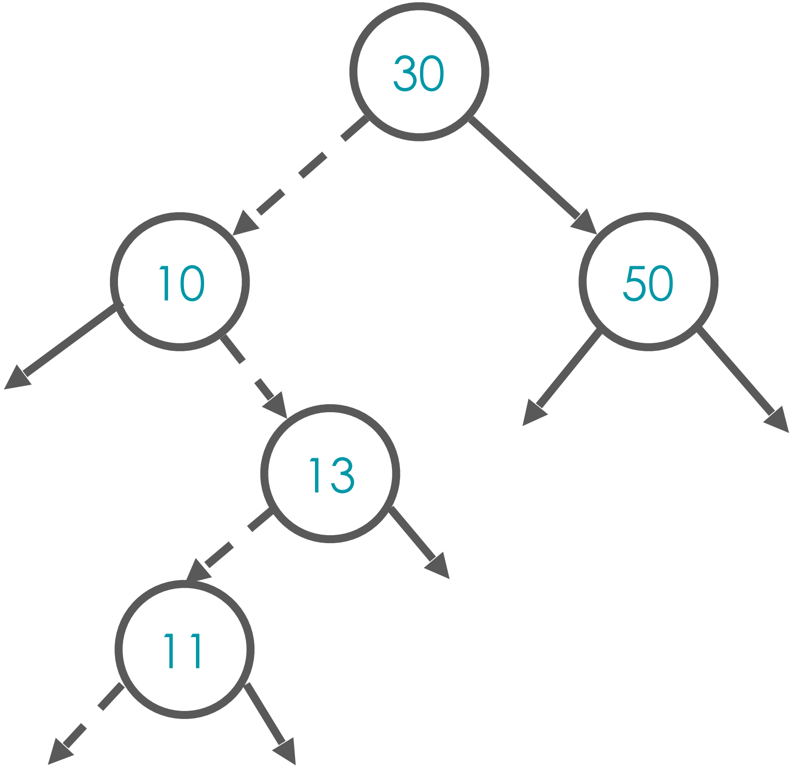
Examples of BST




```
struct node{
   Type data;
   node* left;//smaller values
   node* right;//larger values
};
```

Depth of a of binary search tree node

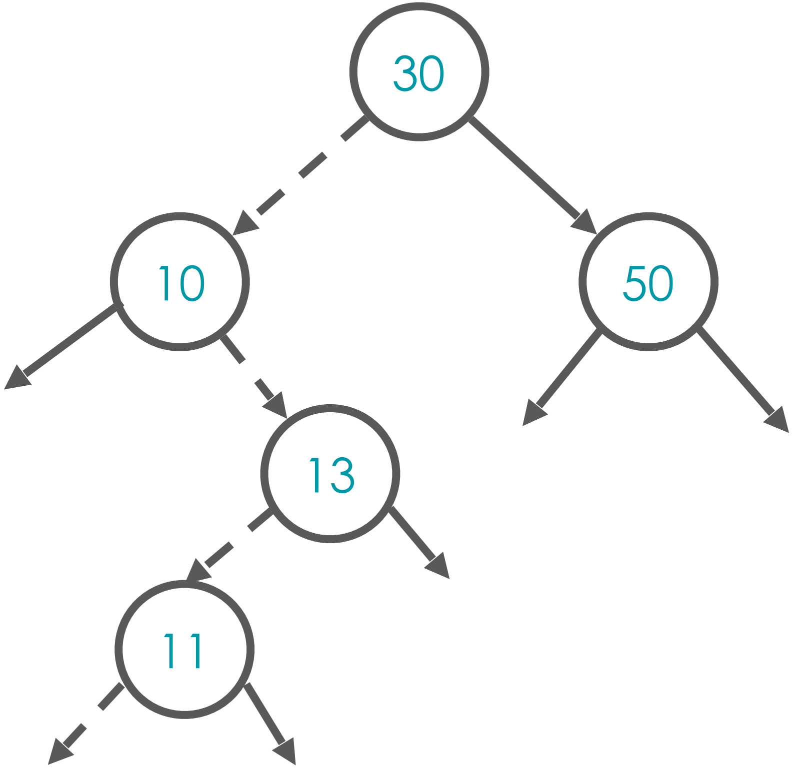
Depth of a node is the number of edges from the root to that given node.
Depth of node containing 13 is 2; node containing 5 is 1; node containing 11 is 3.



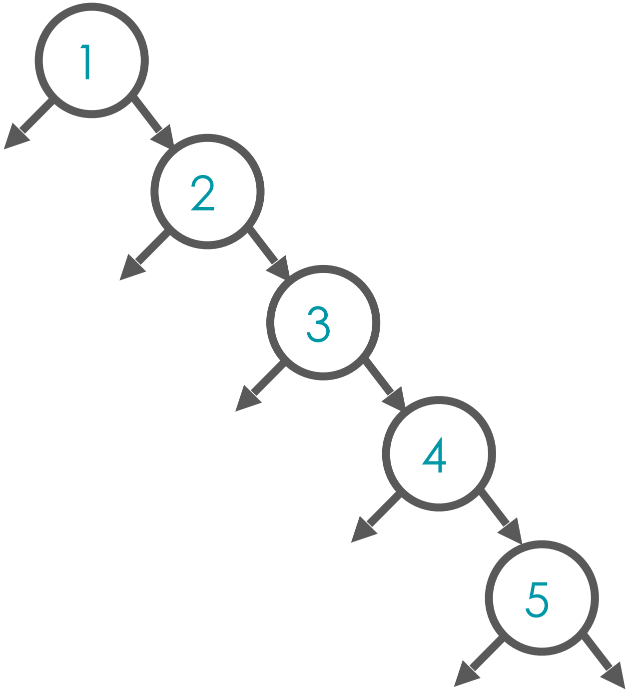
Height of binary search tree

The height of a tree is the number of nodes in the longest path from a root to a leaf . It can be described as the height of the deepest node.

Height of the tree is 3 in the figure.

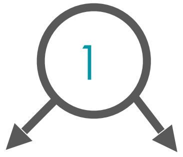


Height of binary search tree



Height of tree is 4 in the figure.

Height of binary search tree



Height of tree is 0 in the figure.

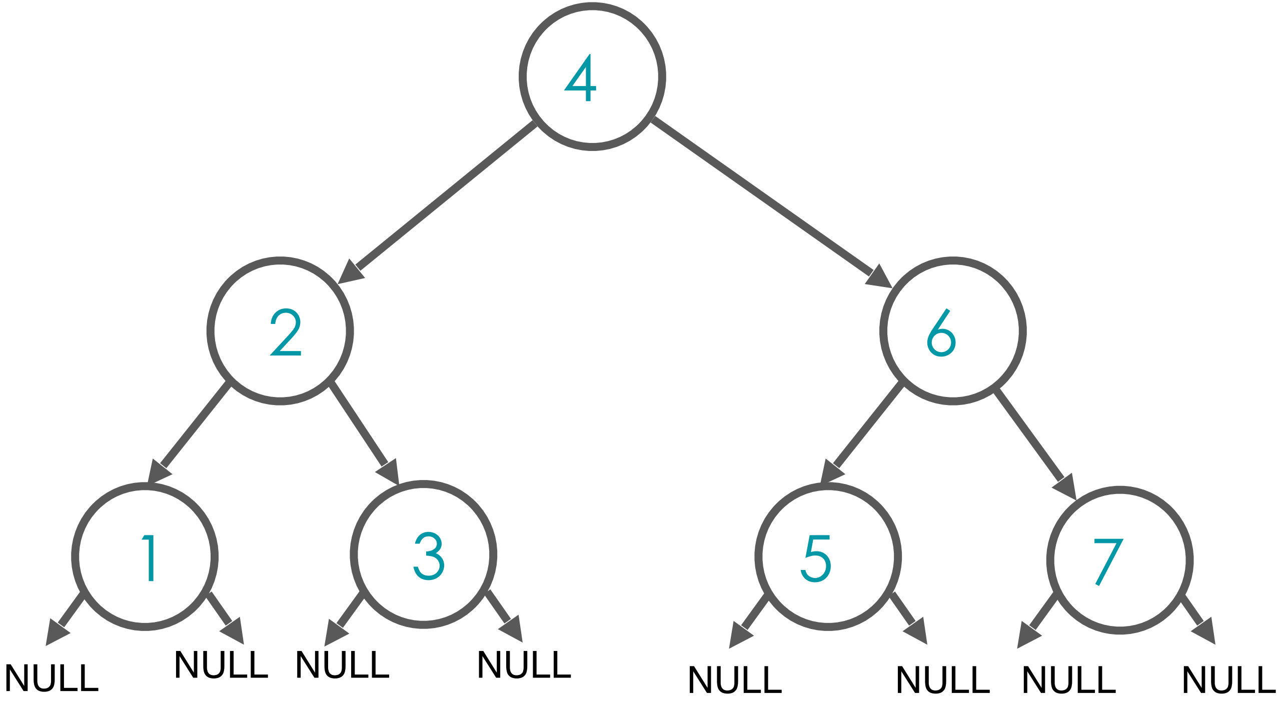
Tree heights

Maximum and minimum height of a tree with n nodes.

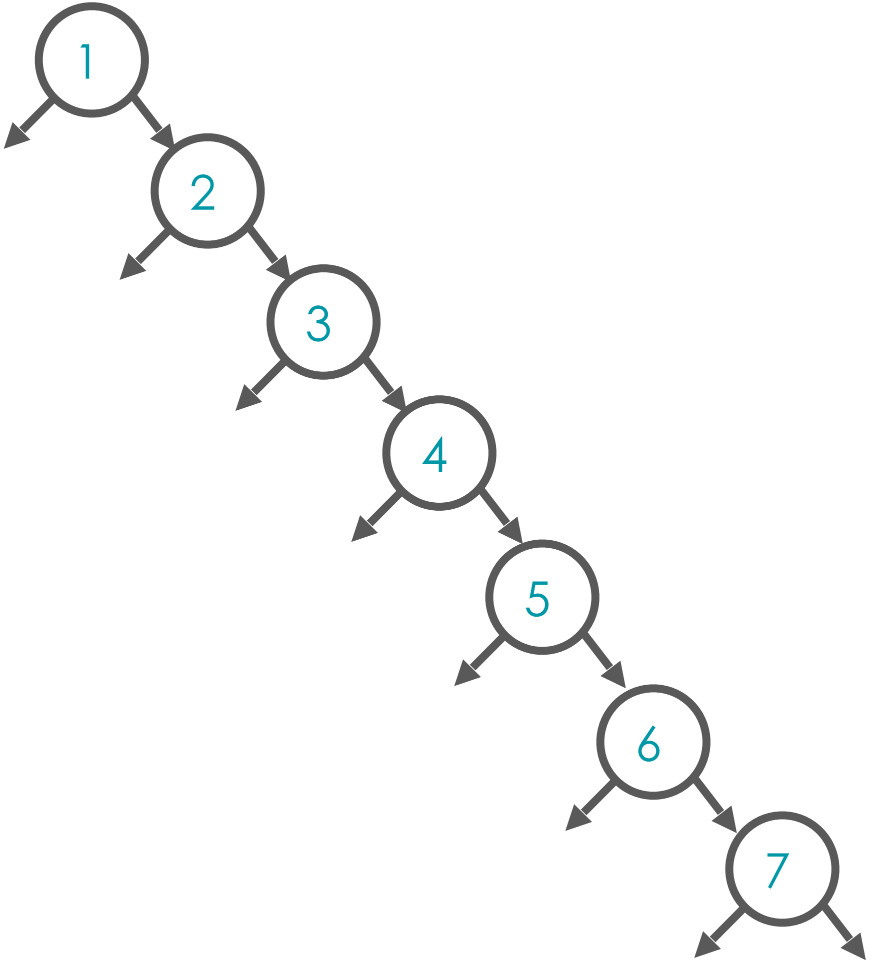
Maximum height is O(n).

Minimum height is O(log n).

Height is O(log n)



Height is O(n)



Balanced search tree

A binary search tree is called balanced if its height is O(log n), where n is the number of nodes in the tree.

Balanced binary search are highly efficient.

- find takes O(log n)
- insertion takes O(log n)
- deletion takes O(log n)

Balanced search tree algorithms

AVL trees

Red Black tree

B+ trees

Runtime analysis of sorting algorithm

a. Bubble Sort $O(n^2)$

b. Merge Sort O(nlogn)

Runtime analysis of search algorithm

a. Linear Search $O(n)$
b. Binary Search $O(\log n)$

Runtime analysis of fibonacci

a. Iterative algorithm $O(n)$
b. Recursive algorithm $O(2^n)$