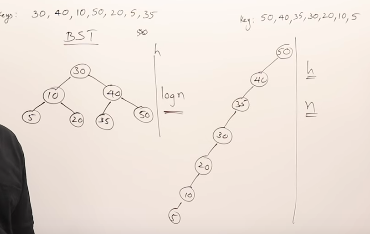
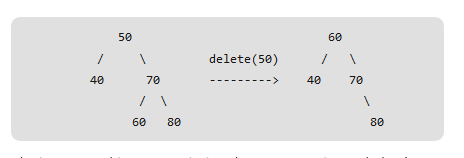
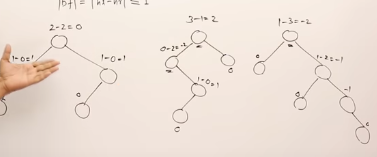
**Binary Search Tree**

* Keys such that elements to left hand side are smaller than the key element.
* Elements in right subtree are greater.
* ***Search*** (let say 30)
  + We start at root (40)
  + We see 40 is smaller so move left reach (20)
  + Now from 20 we see 30 greater so move right
  + Now got 30 in 3 comparisons.
  + If reach place where no children than search fails
  + TIME TAKEN IN SEARCH DEPENDS ON HEIGHT
* (insertion)
  + Create BST by inserting one element at a time. First element root, then move left right accordingly.
* BST does not have checks or fixes for height so can have this
  + - thats why worse case O(N)
* **OPERATIONS TIME COMPLEXITY**:
  + INSERT, DELETE, SEARCH
    - Best: O(1)
    - AVG: O(LOG(N))
    - WORST: O(H) h=height
* DELETING
  + 3 cases
    - Deleting leaf node
      * Simply remove from tree (set to null)
    - deleting node with one child
      * remove node and replace with child
    - deleting node with 2 child
      * Lets say the node to be deleted is N
      * We don’t delete N, instead we choose get
        + Minimum of right subtree or
        + Maximum of left, swap the data and than delete

**AVL TREE**

* Want minimum height tree so that search can be improved.
* Utilizes rotations to help balance tree.
* Height balanced BST, we balance by
  + Balance factor= height of left subtree – height of right subtree
  + Calculate balance factor for each node and IT HAS TO HAVE answer of
    - -1, 0 , 1
    - IF NOT THESE THAN INBALANCED
  + Initially insert like bst, than PERFORMS ROTATIONS to balance