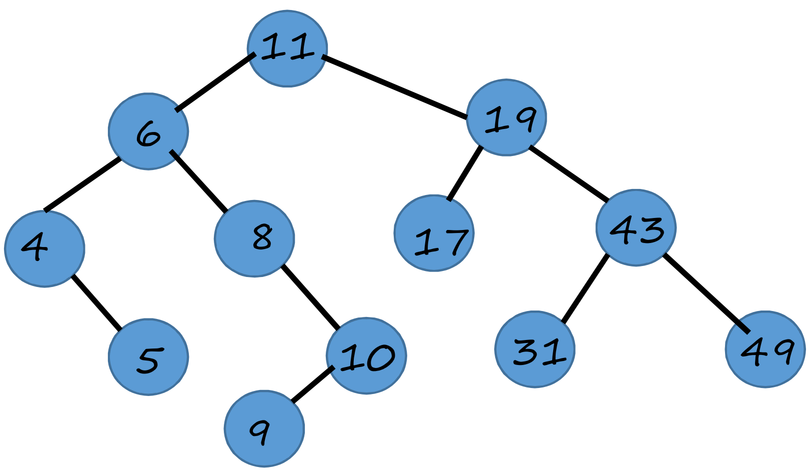
Augmented BSTs

Definitions, terminologies, notations

- Given a BST, along with the key, one or more of the following information is stored at each node:
  - Level(x): number of nodes on the shortest path from the root to x.
  - Depth(x): ~~number of edges on the shortest path~~ distance between the root and x.
  - Height(x): distance between x and its farthest descendant.
  - Size(x): number of nodes on the subtree rooted at x.



Nodes at level 2: 6, 19
Nodes at depth 2: 4, 8, 17, 43
Nodes of height 2: 6, 8, 19
Nodes of size 2: 4, 10, 17

Recursive definitions:

- Level: Level[Root]=1,
  Level[x]=Level[Parent[x]]+1
- Depth: Depth[Root]=0,
  Depth[x]=Depth[Parent[x]]+1

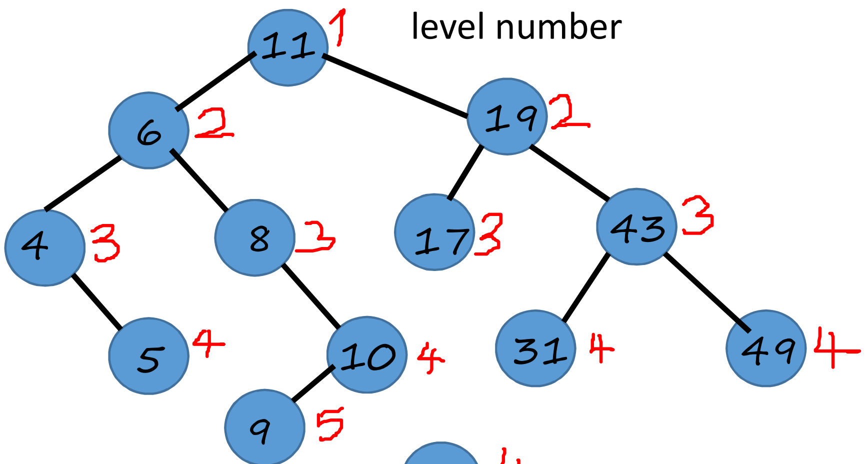
Require parent pointers

- Height: Height[Sentinel]=−1,
  Height[x]=max{Height[LeftChild[x]],
  Height[RightChild[x]]} + 1
- Size: Size[Sentinel]=0,
  Size[x]=Size[LeftChild[x]] +
  Size[RightChild[x]] + 1

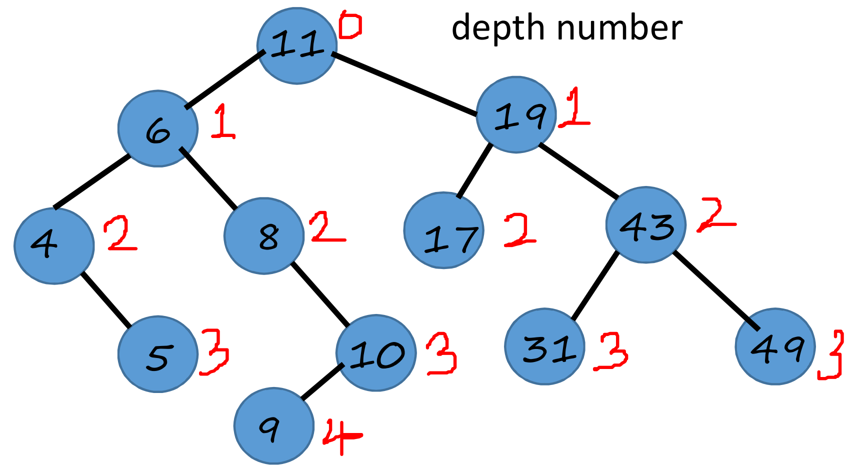
Example

- A BST with augmented information:

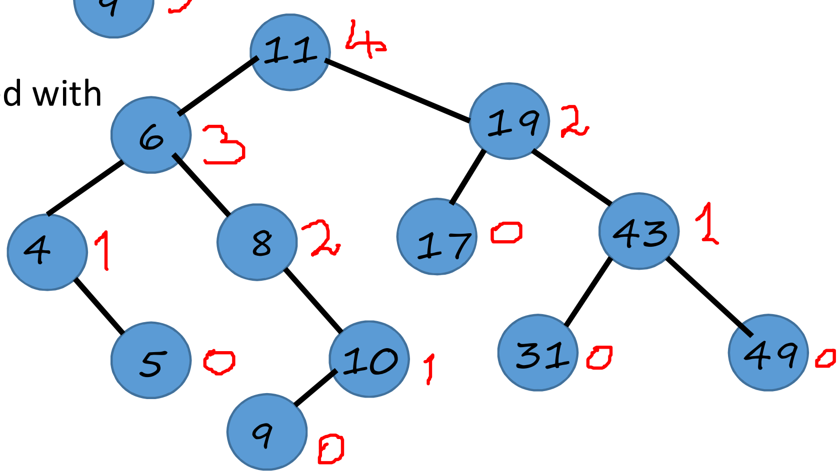
Augmented with level number



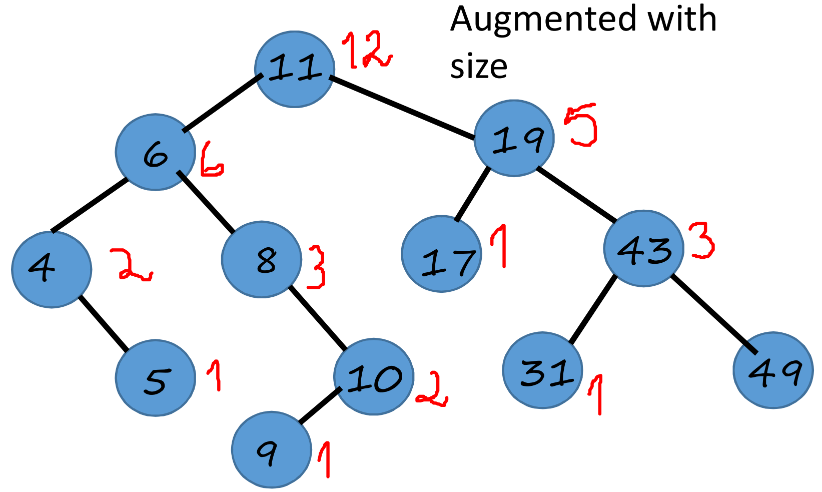
Augmented with depth number



Augmented with height

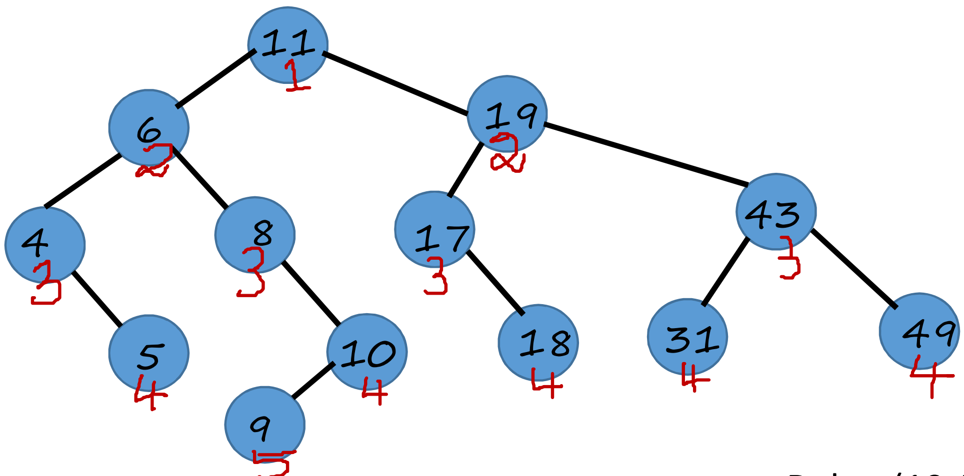


Augmented with size

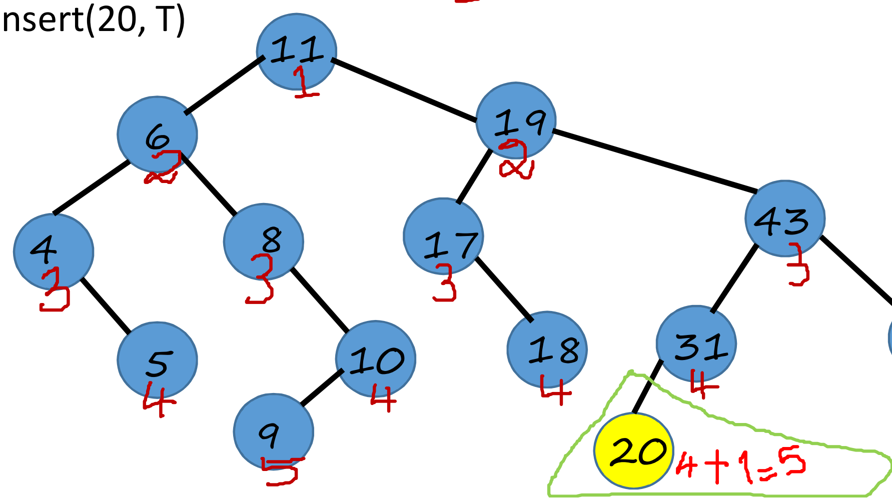


Information update for BST operations

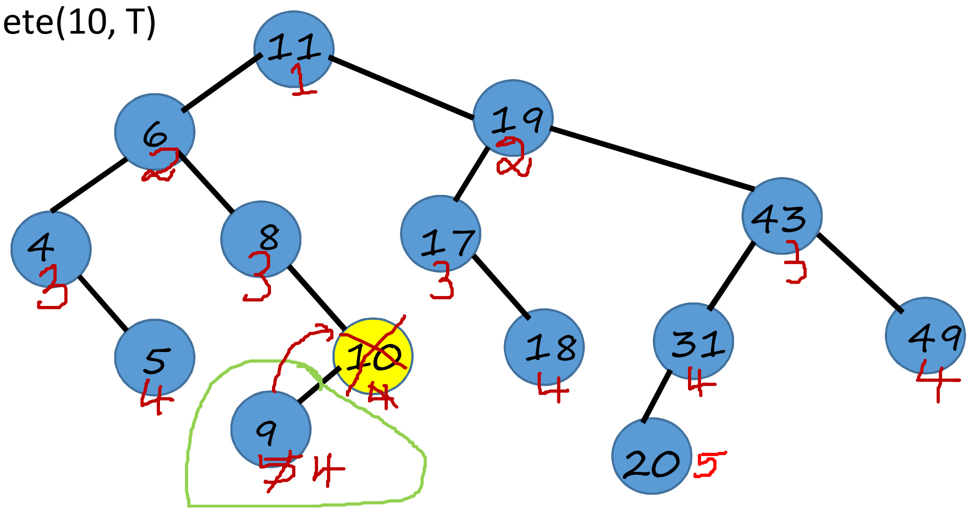
- What is the impact of insertion/deletion on level?



Insert(20, T)



Delete(10, T)



Information updates for BST operations

- Update cost for level number for Insert(x, T):

  Note: x will be inserted as a leaf node.

  Level[x] = number of nodes on the insertion path = 1+Level[Parent[x]].

  Level[y] is not altered due to Insert(x, T) for all y (already) in T.

  Hence, cost of update to level numbers is O(1)

- Update cost for level number for Delete(x, T):

  Node x is replaced by its in-order predecessor (or successor), say x‘.

  Note: In this case x‘ will not have a right subtree (or left subtree).

  This causes the left subtree (or right subtree) at x‘ to move up by one level and the node x‘ moves up by one or more levels. So, x‘ and its descendants undergo a change in their level number.

  Hence, cost of update to level numbers is O(size(x‘)) = O(n)

Information updates for BST operations

- Update cost for depth number for Insert(x, T):

  Similar to level number for Insert(x, T).

  cost of update to depth numbers is $O(1)$

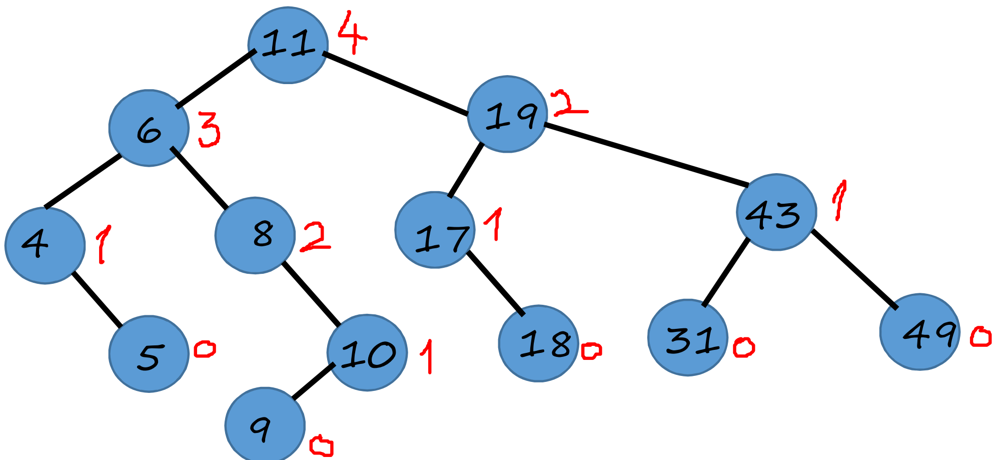
- Update cost for depth number for Delete(x, T):

  Similar to level number for Delete(x, T).

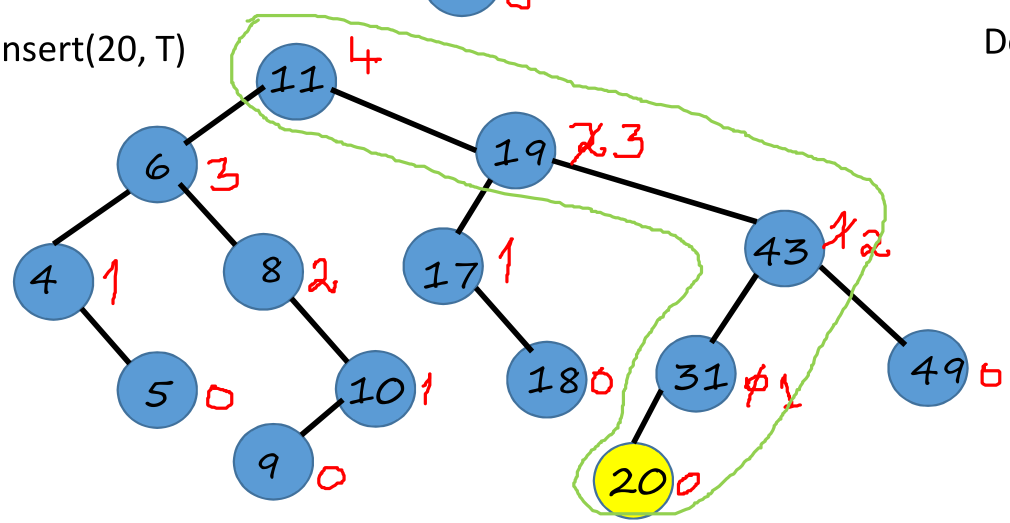
  cost of update to depth numbers is $O(n)$

Information update for BST operations

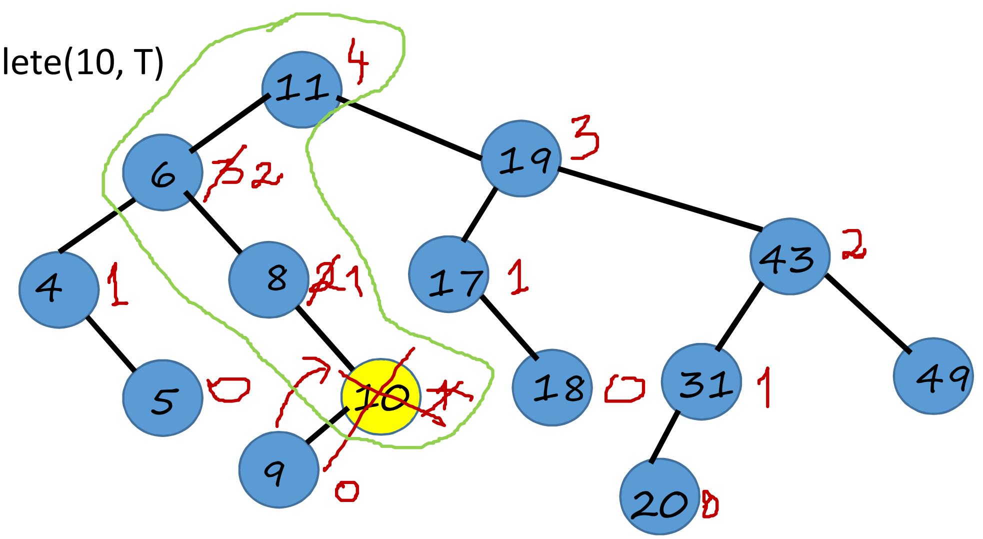
- What is the impact of insertion/deletion on height?



Insert(20, T)



Delete(10, T)



Information updates for BST operations

- Update cost for height number for Insert(x, T):

  Note: x will be inserted as a leaf node.

  Height[x] = 0.

  Height[y] may be altered, if y is on the insertion path. Also,

Height[y] = max{Height[y], 1+Height of subtree at with insertion takes place}

cost of update to height numbers is the length of the insertion path = O(h)

- Update cost for level number for Delete(x, T):

  Node x is replaced by its in-order predecessor (or successor), say x'.

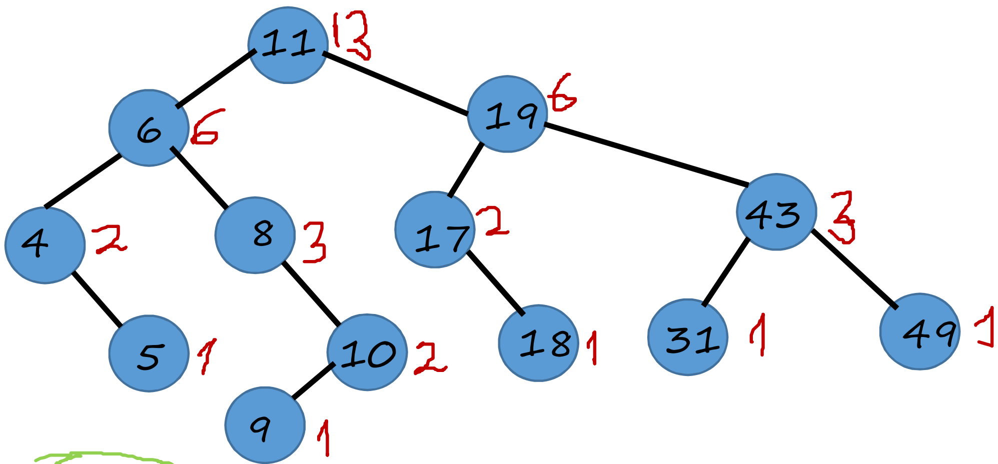
  The height of the nodes that lie on the path from the root to x may get altered. [If the root is deleted, then the nodes on the path from the root to x' may get altered.

  Height[y] = max{Height[y], Height of the subtree at which deletion happened-1}

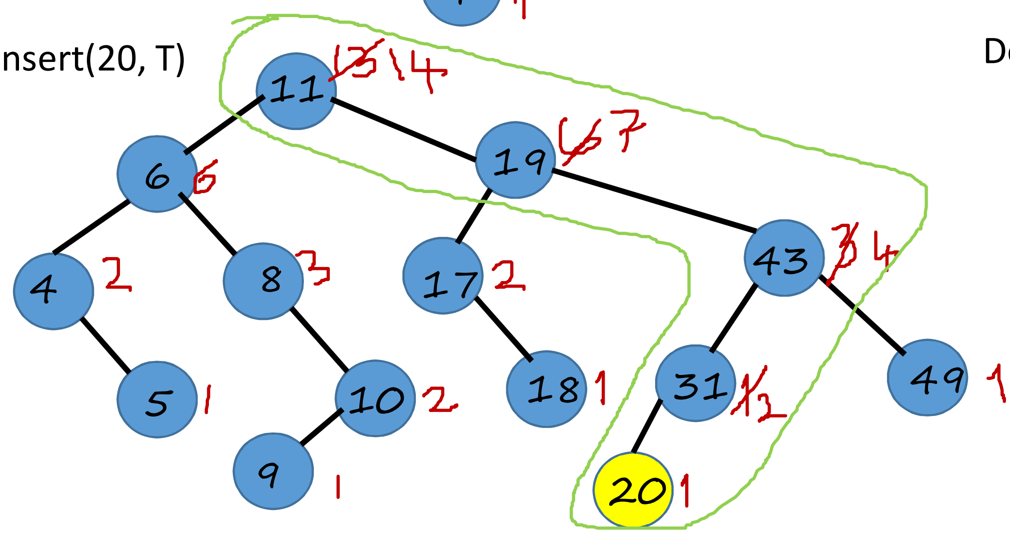
  cost of update to height numbers is O(h)

Information update for BST operations

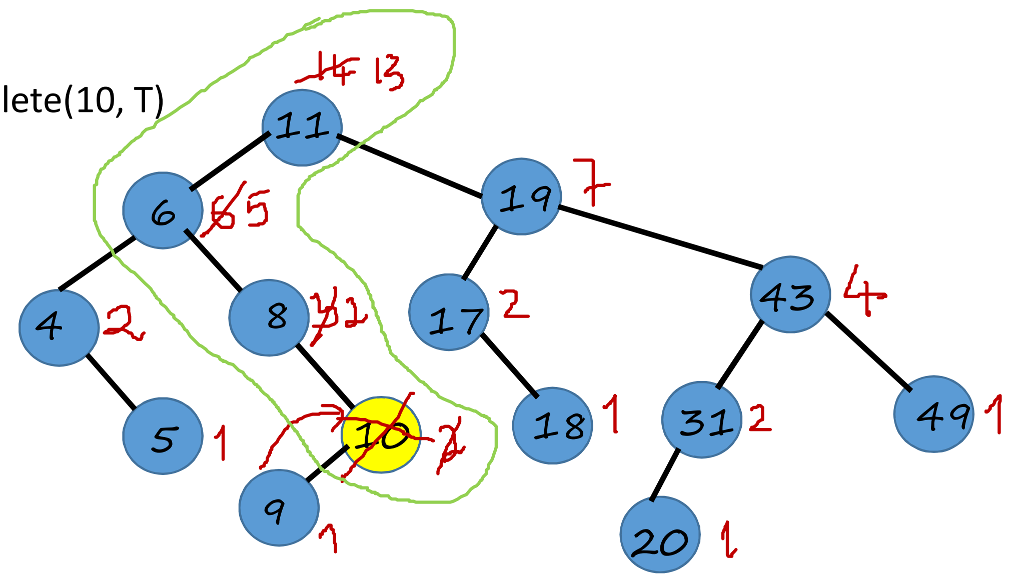
- What is the impact of insertion/deletion on size?



Insert(20, T)



Delete(10, T)



Information updates for BST operations

- Update cost for size number for Insert(x, T):

  Note: x will be inserted as a leaf node.

  Size[x] = 1.

  Size[y] may be altered, if y is on the insertion path. Also, for such y,

  $$\text{Size}[y] = 1+\text{Size}[y]$$

  cost of update to size is the length of the insertion path = O(h)

- Update cost for level number for Delete(x, T):

  Node x is replaced by its in-order predecessor (or successor), say x'.

  The size of the nodes that lie on the path from the root to x may get altered. [If the root is deleted, then the nodes on the path from the root to x' may get altered. Also, for such nodes,

  $$\text{Size}[y] = \text{Size}[y]-1$$

  cost of update to size is O(h)

Information updates for BST operations

Operation Property maintained	Insert	Delete
BST	$O(h)$	$O(h)$
Level	$O(1) + O(h) = O(h)$	$O(n) + O(h) = O(n)$
Depth	$O(1) + O(h) = O(h)$	$O(n) + O(h) = O(n)$
Height	$O(h) + O(h) = O(h)$	$O(h) + O(h) = O(h)$
Size	$O(h) + O(h) = O(h)$	$O(h) + O(h) = O(h)$

Height and Size can be augmented without incurring any additional (asymptotic) cost.

Applications of augmented BSTs

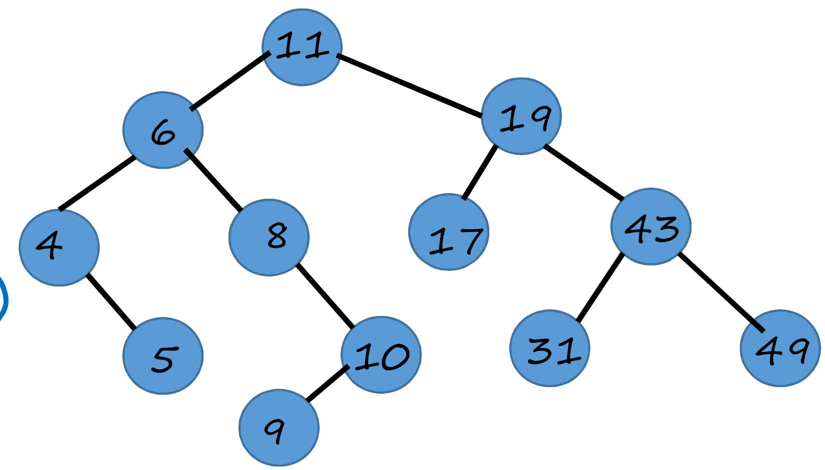
- The k^{th} order statistic of a set is the element with the k^{th} smallest value; i.e., the k^{th} element in the sorted set.
- Special Cases:
  - 1^{st} order statistic – the least element
  - n^{th} order statistic – the largest element
  - $n/2^{th}$ order statistic – the median [ceil(n/2) or floor(n/2) is opted when n is odd.]
- Rank of an element k of a set is the number of elements < k; i.e., the position of k in the sorted set.
- Given a sorted set, using an array implementation, the k^{th} order statistic (for $1 \le k \le n$) can be computed in $O(1)$ time and rank of an element can be computed in $O(n)$ time by using linear search or in $O(\log n)$ time by using binary search.
- Given an unsorted set, using a BST implementation, it can be sorted in $O(n)$ time.
- Given an unsorted set, using a size augmented BST implementation, k^{th} order statistics and ranks can be computed in $O(h)$ time.

Order Statistics and Rank using (size augmented) BSTs

- **Select(k, r)**: returns the k^{th} order statistic of a set represented by a BST rooted at r

SELECT(k, r)

1. $i \leftarrow$ Size[LeftChild[r]]+1
2. If k=i then Return(r)
3. Elseif k<i then Return(SELECT(k, LeftChild[r]))
4. Else Return(SELECT(k−i, RightChild[r]))



The procedure requires traversal (along a path) from r to the k^{th} order statistic; so run-time = $O(h)$

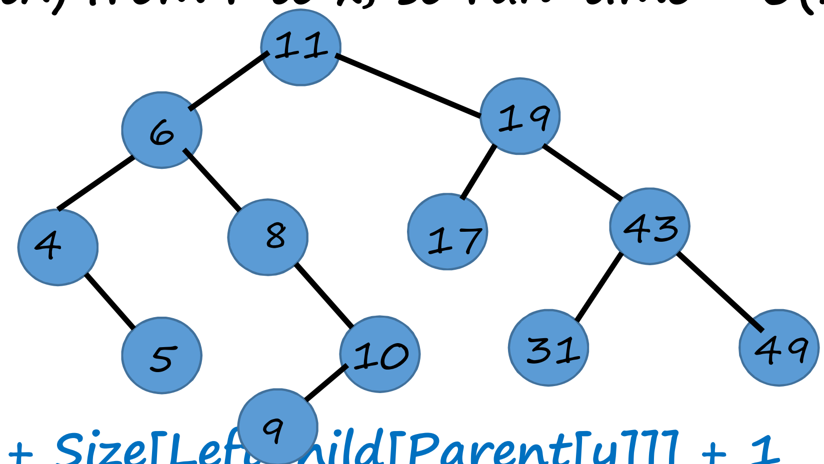
Order Statistics and Rank using (size augmented) BSTs

- Rank(x, r): returns the rank of element x of a set represented by a BST rooted at r

RANK(x, r)

1. If x=r then Rank(x,r) ← Size[LeftChild[x]]+1 //Return(Size[LeftChild[x]]+1)
2. Elseif x<r then Rank(x,r) ← RANK(x, LeftChild[r])
3. Else Rank(x,r) ← RANK(x, RightChild[r]) + $\text{RANK}(r, r)_{=\text{Size[LeftChild[r]]+1}}$

The procedure requires traversal (along a path) from r to x; so run-time = O(h)



RANK(x,r)

1. i ← Size[LeftChild[r]]+1
2. y ← x
3. While y ≠ x
4. if y = RightChild[Parent[y]] then i ← i + Size[LeftChild[Parent[y]]] + 1
5. y ← Parent[y]
6. Return(i)