1. Explain the differences between linear and non-linear data structures!

- Linear data structures have their elements arranged linearly right next to one another, while non-linear data structures have elements attached to one another hierarchically from top to bottom.
- Linear data structures only have one level, while non-linear ones are involved.
- Linear is easier to implement than non-linear.
- Data elements can't be traversed in a single run on non-linear, but possible with linear.
- Non-linear utilizes memory usage far better than linear.

2. Describe the following terminology in a tree: base root, key, edge, siblings, parent, child, and leaf!

- Base root: The node at the top of the tree and only one can exist in a tree.
- Key: The value of a node to be searched for in a tree operation.
- Edge: The connection between one node to another in a tree
- Siblings: Nodes on the same level with the same parent
- Parent: Any node except the root node has one edge upward to a node (Above a node 1 level)
- Child: A node below a given node connected by its edge downward (Below a node 1 level)
- Leaf: A node which have no child node.

3. Explain the following types of binary trees: full, complete, and perfect!

- Full Binary Tree: A binary tree that with each of its nodes having 2 children except for the leaf nodes.
- Complete Binary Tree: A binary tree where all the tree levels are filled entirely with nodes, except the lowest level of the tree. Also, in the last or the lowest level of this binary tree, every node should possibly reside on the left side.
- Perfect Binary Tree: A binary tree is said to be ‘perfect’ if all the internal nodes have strictly two children, and every leaf node is at the same level within a tree.

4. What makes a tree balanced?

A binary tree is said to be ‘balanced’ if the tree height is O(logN), where ‘N’ is the number of nodes. In a balanced binary tree, the height of the left and the right subtrees of each node should vary by at most one.

5. Explain the four properties of a binary tree!

- The maximum number of nodes at level ‘l’ will be 2^{l-1} . Here level is the number of nodes on path from root to the node, including the root itself. We are considering the level of root is 1.
- Maximum number of nodes present in binary tree of height h is $2^{h}-1$. Here height is the max number of nodes on root to leaf path. Here we are considering height of a tree with one node is 1.
- In a binary tree with n nodes, minimum possible height or minimum number of levels are $\log_2(n+1)$. If we consider that the height of leaf node is considered as 0, then the formula will be $\log_2(n+1)-1$.
- A binary tree with ‘L’ leaves has at least $\log 2L+1$ number of levels.

6. Explain the intuition of implementing a binary tree using an array!

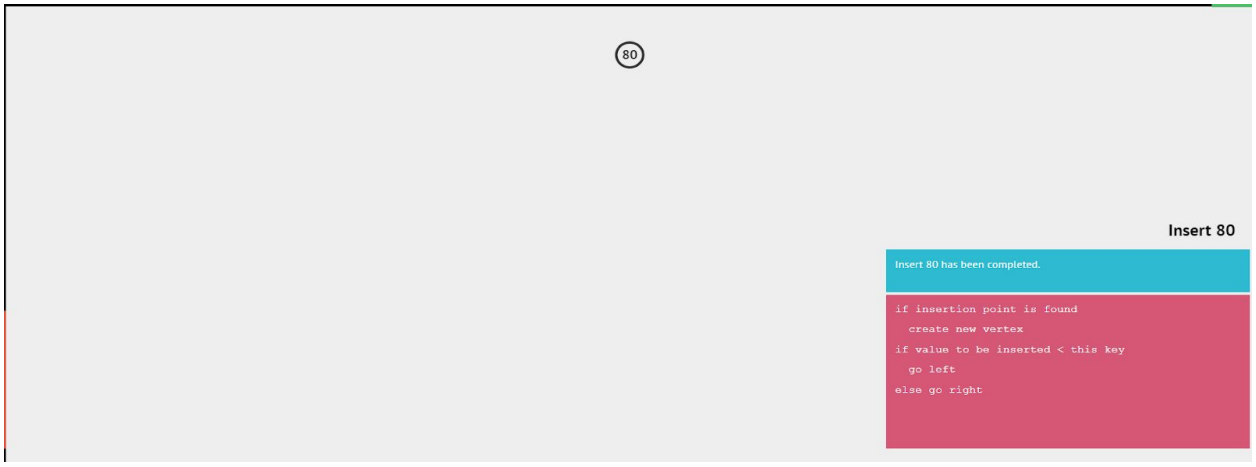
- The base root will be at index 0 of the array. The left child of a root/node can be found using the formula 2(parent node)+1, while the right child can be found using the formula 2(parent node)+2. If we want to search for the parent of a node, we can use the formula (current index -1)/2.

7. Explain the differences between inorder successor and inorder predecessor!

- Inorder successor refers to the node that lies in front the root node. Inorder predecessor refers to the node that lies behind the root node.
- Successor is positioned on the leftmost element on the right subtree, while predecessor is situated on rightmost element on the left subtree.

8. Draw the following binary search tree step by step (14 pictures):

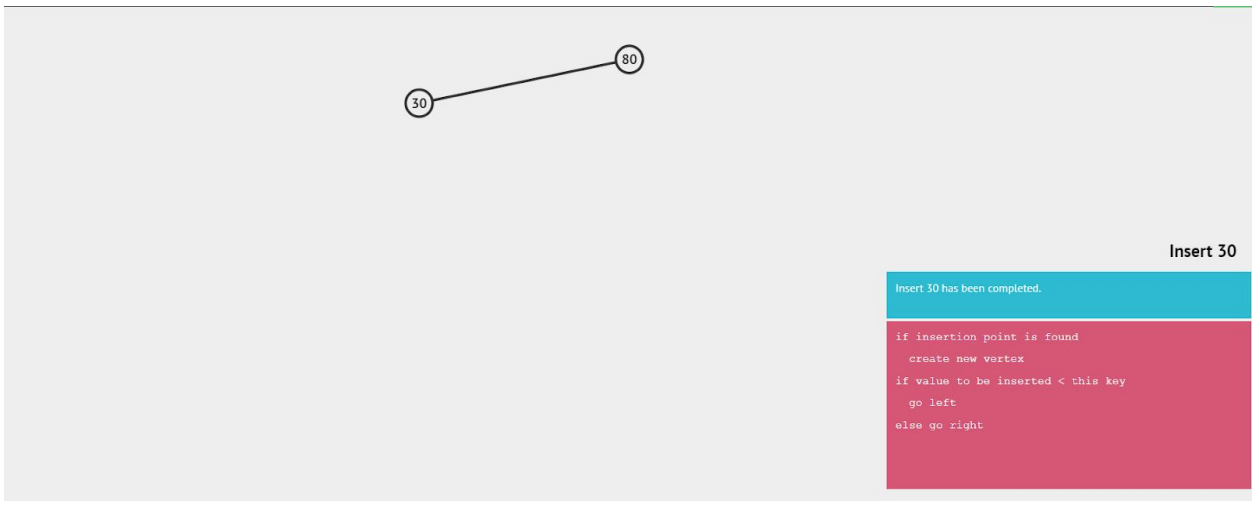
- Insert 80, 30, 60, 50, 75



Insert 80

Insert 80 has been completed.

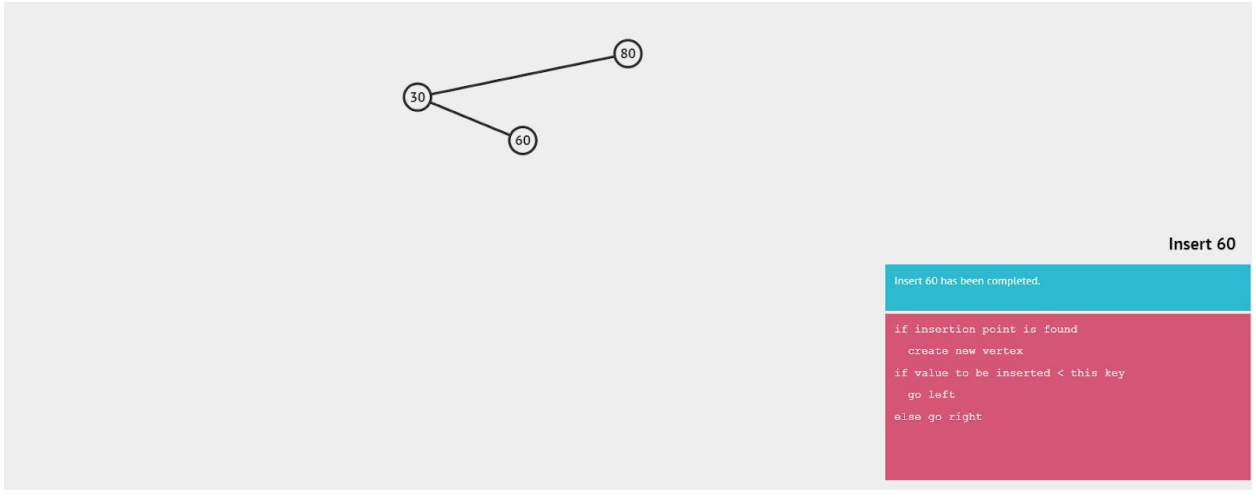
```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```



Insert 30

Insert 30 has been completed.

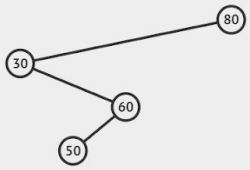
```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```



Insert 60

Insert 60 has been completed.

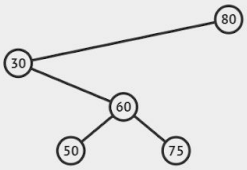
```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```



Insert 50

Insert 50 has been completed.

```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```

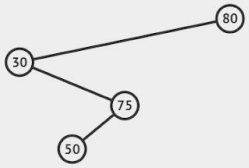


Insert 75

Insert 75 has been completed.

```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```

- Delete 60, 30, 75



Remove 60

Removal of 60 completed

```
search for v
if v is a leaf
  delete leaf v
else if v has 1 child
  bypass v
else replace v with successor
```

Remove 30

Removal of 30 completed

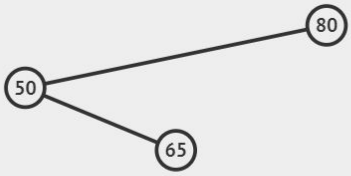
```
search for v
if v is a leaf
  delete leaf v
else if v has 1 child
  bypass v
else replace v with successor
```

Remove 75

Removal of 75 completed

```
search for v
if v is a leaf
  delete leaf v
else if v has 1 child
  bypass v
else replace v with successor
```

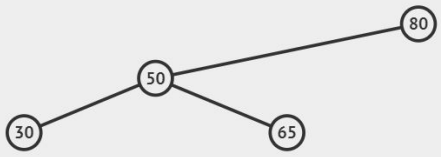
- Insert 65, 30, 35



Insert 65

Insert 65 has been completed.

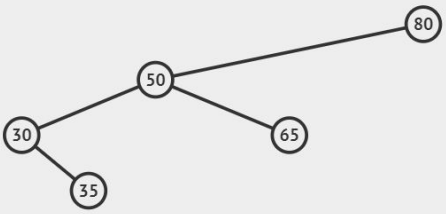
```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```



Insert 30

Insert 30 has been completed.

```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```

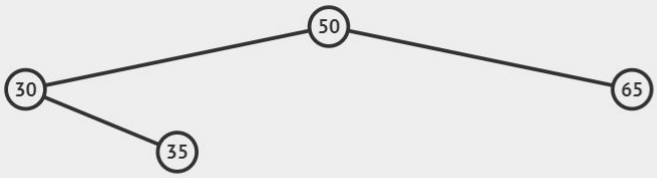


Insert 35

Insert 35 has been completed.

```
if insertion point is found
  create new vertex
if value to be inserted < this key
  go left
else go right
```

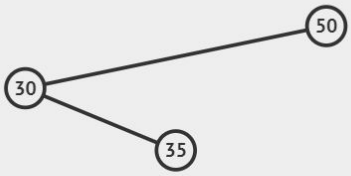
- Delete 80, 65, 35



Remove 80

Removal of 80 completed

```
search for v
if v is a leaf
  delete leaf v
else if v has 1 child
  bypass v
else replace v with successor
```



Remove 65

Removal of 65 completed

```
search for v
if v is a leaf
  delete leaf v
else if v has 1 child
  bypass v
else replace v with successor
```

Remove 35

Removal of 35 completed

```
search for v
if v is a leaf
  delete leaf v
else if v has 1 child
  bypass v
else replace v with successor
```