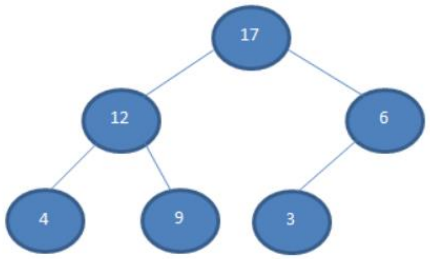
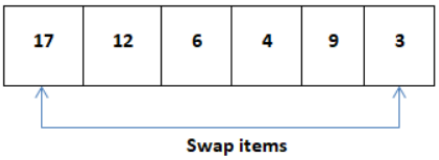
NOTES

Heap Sort

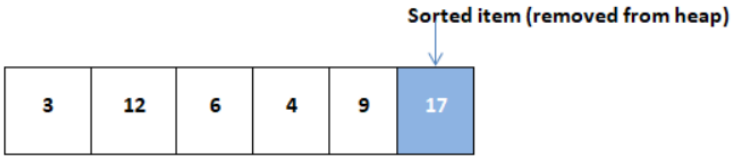
Say we have an array with elements 4 17 3 12 9 6



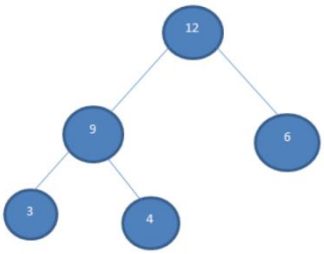
Once the heap is constructed, we represent it in an Array form as shown below. So it will be like 17 12 6 4 9 3. Now we compare the 1st node (root) with the last node and then swap them. We swap 17 and 3 so that 17 at the last position and 3 in the first position. The resultant array will be 3 12 6 4 9 3. Look at the diagram here-



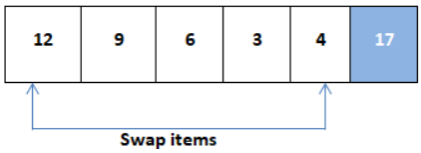
Now we remove the node 17 from the heap and put it in the sorted array as shown in the shaded portion below.



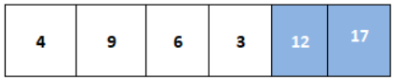
Now we again construct a heap for the array elements. This time the heap size is reduced by 1 as we have deleted one element (17) from the heap.



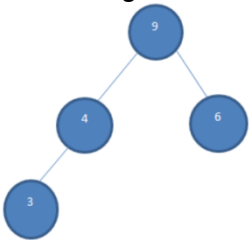
From the above heap the generated array will be 12 9 6 3 4 17. In the next step, we will repeat the same steps means to swap the root element with the last element.



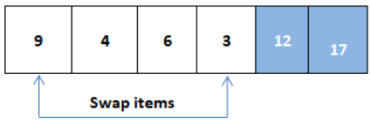
After swapping, we delete the element 12 from the heap and shift it to the sorted array.



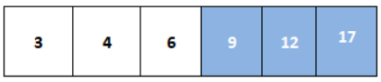
Once again we construct a max heap for the remaining elements as shown below.



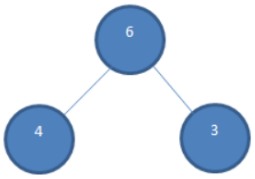
From the above heap the generated array will be 9 4 6 3 12 17. Now we swap the root and the last element i.e. 9 and 3.



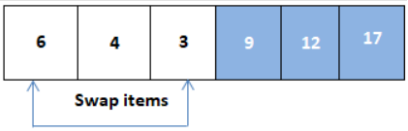
After swapping, element 9 is deleted from the heap and put in a sorted array.



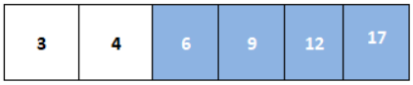
construct the heap from unsorted array elements it will be like



We swap 6 and 3.

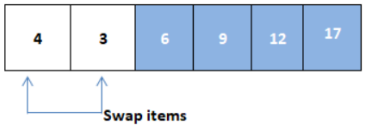


Delete the element 6 from the heap and add it to the sorted array.

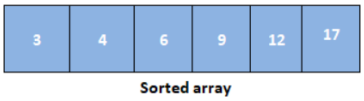


Now we construct a heap of the remaining elements.

Swap both with each other.



After swapping 4 and 3, we delete element 4 from the heap and add it to the sorted array. Now we have only one node remaining in the heap. We delete it from the heap and add it to the sorted array.



Thus the above shown is the sorted array that we have obtained as a result of the heap sort.

```
def heapify(arr, n, i):
  largest = i
  l = 2 * i + 1
  r = 2 * i + 2
  if l < n and arr[i] < arr[l]:
    largest = l
  if r < n and arr[largest] < arr[r]:
    largest = r
  if largest != i:
    arr[i],arr[largest] = arr[largest],arr[i] # swap
    # root.
    heapify(arr, n, largest)

def heapSort(arr):
  n = len(arr)
  for i in range(n, -1, -1):
    heapify(arr, n, i)
  for i in range(n-1, 0, -1):
    arr[i], arr[0] = arr[0], arr[i] # swap
    heapify(arr, i, 0)
```

```
arr = [2,5,3,8,6,5,4,7]
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
   print (arr[i],end=" ")
```

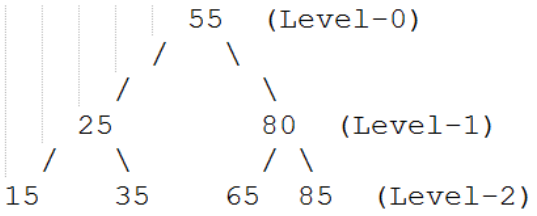
Binary Search Tree

1. Each node can have maximum two children
2. Children on the LHS must be less than the root element
3. Children on the RHS must be more than the root element
4. BST (Binary Search Tree) cannot have duplicate values

Insert in BST

If element is less than the root element then move to left subtree
If element is more than the root element then move to right subtree
if root is null then insert the element

Say we have elements 55 25 15 80 65 35 85 then the constructed tree will be-



Traversal (Accessing)

For the above tree

1. Pre-Order (Root-Left-Right): The first element will be the root element
55 25 15 35 80 65 85

2. In-Order (Left-Root-Right): Always provide you the sorted order
15 25 35 55 65 80 85

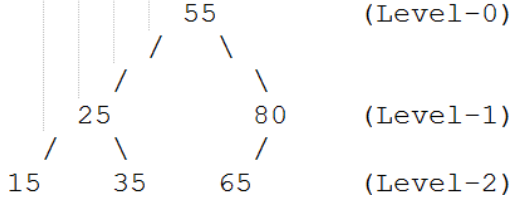
3. Post-Order (Left-Right-Root): The last element will be the root element
15 35 25 65 85 80 55

Search in binary search tree

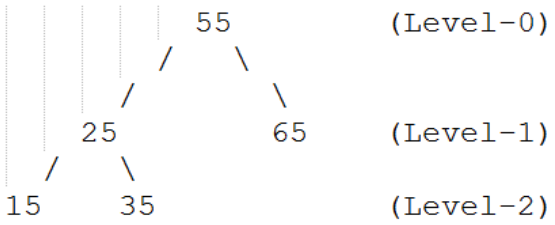
if ele < root then root = root->left
else if ele > root then root = root->right
else return root;

Deletion in BST

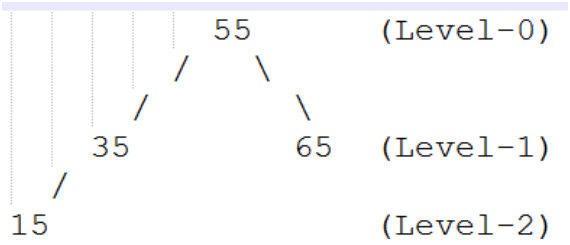
Consider the tree given above and consider following cases
Case 1: When leaf node is to be deleted i.e. 85 delete directly set right of 80 to null



Case 2/3: When node to be deleted has one child only (e.g. 80 for tree given immediately above) swap content of node to be deleted with its left or right child then delete the left or right child



Case 4: When node to be deleted has two children (e.g. 25) search for inorder successor (in this case it is 35) swap the inorder successor with node to be deleted. Then delete the inorder successor.




```
class Node:
	def __init__(self, val):
		self.left = None
		self.right = None
		self.val = val

def insert(root, val):
	if root is None:
		return Node(val)
	else:
		if root.val == val:
			return root
		elif root.val < val:
			root.right = insert(root.right, val)
		else:
			root.left = insert(root.left, val)
	return root

def inorder(root):
	if root:
		inorder(root.left)
		print(root.val,end = ' ')
		inorder(root.right)

def preorder(root):
	if root:
		print(root.val,end = ' ')
		preorder(root.left)
		preorder(root.right)

def postorder(root):
	if root:
		postorder(root.left)
		postorder(root.right)
		print(root.val,end = ' ')

def minValueNode(node):
	current = node

	# loop down to find the leftmost leaf
```

```
        while(current.left is not None):
                current = current.left

        return current

def deleteNode(root, val):
        if root is None:
                return root

        if val < root.val:
                root.left = deleteNode(root.left, val)
        elif(val > root.val):
                root.right = deleteNode(root.right, val)
        else:
                if root.left is None:
                        temp = root.right
                        root = None
                        return temp
                elif root.right is None:
                        temp = root.left
                        root = None
                        return temp

                temp = minValueNode(root.right)

                root.val = temp.val
                root.right = deleteNode(root.right, temp.val)
        return root

r = Node(50)
r = insert(r, 30)
r = insert(r, 20)
r = insert(r, 40)
r = insert(r, 70)
r = insert(r, 60)
r = insert(r, 80)
inorder(r)
print("\n===============")
preorder(r)
print("\n===============")
postorder(r)
print("\n===============")
deleteNode(r, 30)
deleteNode(r, 80)
deleteNode(r, 40)
inorder(r)
print("\n===============")
preorder(r)
print("\n===============")
postorder(r)
print("\n===============")
```