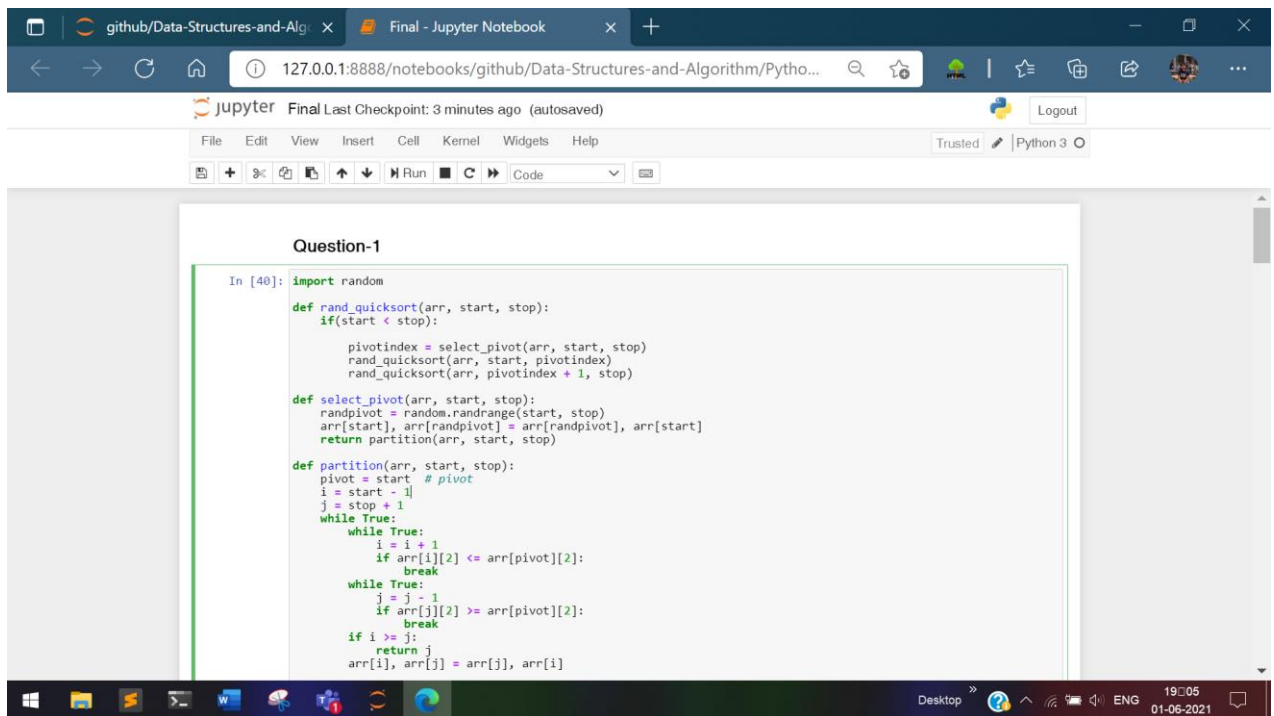


Question-1

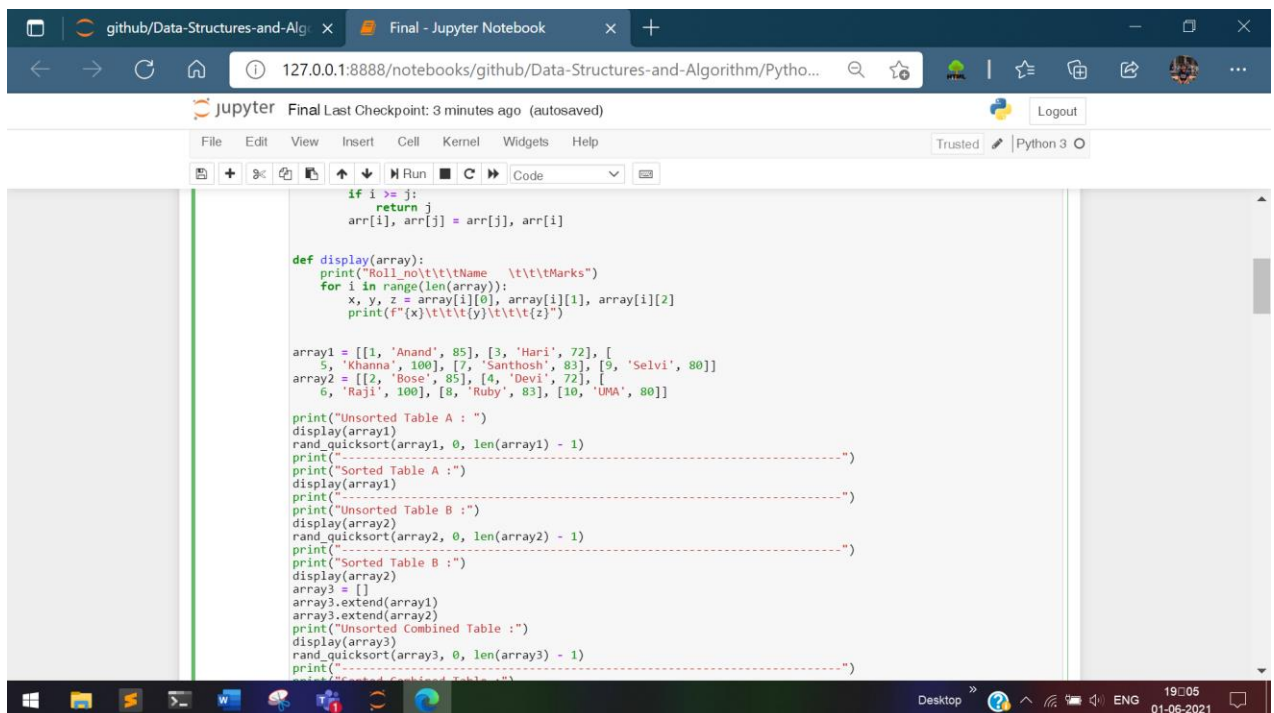
```
Question-1

In [40]: import random

def rand_quicksort(arr, start, stop):
    if(start < stop):
        pivotindex = select_pivot(arr, start, stop)
        rand_quicksort(arr, start, pivotindex)
        rand_quicksort(arr, pivotindex + 1, stop)

def select_pivot(arr, start, stop):
    randpivot = random.randrange(start, stop)
    arr[start], arr[randpivot] = arr[randpivot], arr[start]
    return partition(arr, start, stop)

def partition(arr, start, stop):
    pivot = start # pivot
    i = start - 1
    j = stop + 1
    while True:
        while True:
            i = i + 1
            if arr[i][2] <= arr[pivot][2]:
                break
        while True:
            j = j - 1
            if arr[j][2] >= arr[pivot][2]:
                break
        if i >= j:
            return j
        arr[i], arr[j] = arr[j], arr[i]
```



```
if i >= j:
    return j
arr[i], arr[j] = arr[j], arr[i]

def display(array):
    print("Roll_no\t\tName\t\tMarks")
    for i in range(len(array)):
        x, y, z = array[i][0], array[i][1], array[i][2]
        print(f"{x}\t\t{y}\t\t{z}")

array1 = [[1, 'Anand', 85], [3, 'Hari', 72], [5, 'Khanna', 100], [7, 'Santhosh', 83], [9, 'Selvi', 80]]
array2 = [[2, 'Bose', 85], [4, 'Devi', 72], [6, 'Raji', 100], [8, 'Ruby', 83], [10, 'UMA', 80]]

print("Unsorted Table A : ")
display(array1)
rand_quicksort(array1, 0, len(array1) - 1)
print("-----")
print("Sorted Table A : ")
display(array1)
print("-----")
print("Unsorted Table B : ")
display(array2)
rand_quicksort(array2, 0, len(array2) - 1)
print("-----")
print("Sorted Table B : ")
display(array2)
array3 = []
array3.extend(array1)
array3.extend(array2)
print("Unsorted Combined Table : ")
display(array3)
rand_quicksort(array3, 0, len(array3) - 1)
print("-----")
print("Sorted Combined Table : ")
display(array3)
```

github/Data-Structures-and-Alg... Final - Jupyter Notebook

127.0.0.1:8888/notebooks/github/Data-Structures-and-Algorithm/Pytho...

Jupyter Final Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
display(array3)
rand_quicksort(array3, 0, len(array3) - 1)
print("-----")
print("Sorted Combined Table :")
display(array3)
```

Unsorted Table A :

Roll_no	Name	Marks
1	Anand	85
3	Hari	72
7	Khanna	100
9	Santhosh	80

Sorted Table A :

Roll_no	Name	Marks
5	Khanna	100
1	Anand	85
7	Santhosh	80
3	Hari	72

Unsorted Table B :

Roll_no	Name	Marks
2	Bose	85
4	Devi	72
6	Raji	100
8	Ruby	83
10	UMA	80

Sorted Table B :

Roll_no	Name	Marks
6	Raji	100
2	Bose	85
8	Ruby	83
10	UMA	80
4	Devi	72

github/Data-Structures-and-Alg... Final - Jupyter Notebook

127.0.0.1:8888/notebooks/github/Data-Structures-and-Algorithm/Pytho...

Jupyter Final Last Checkpoint: 4 minutes ago (autosaved)

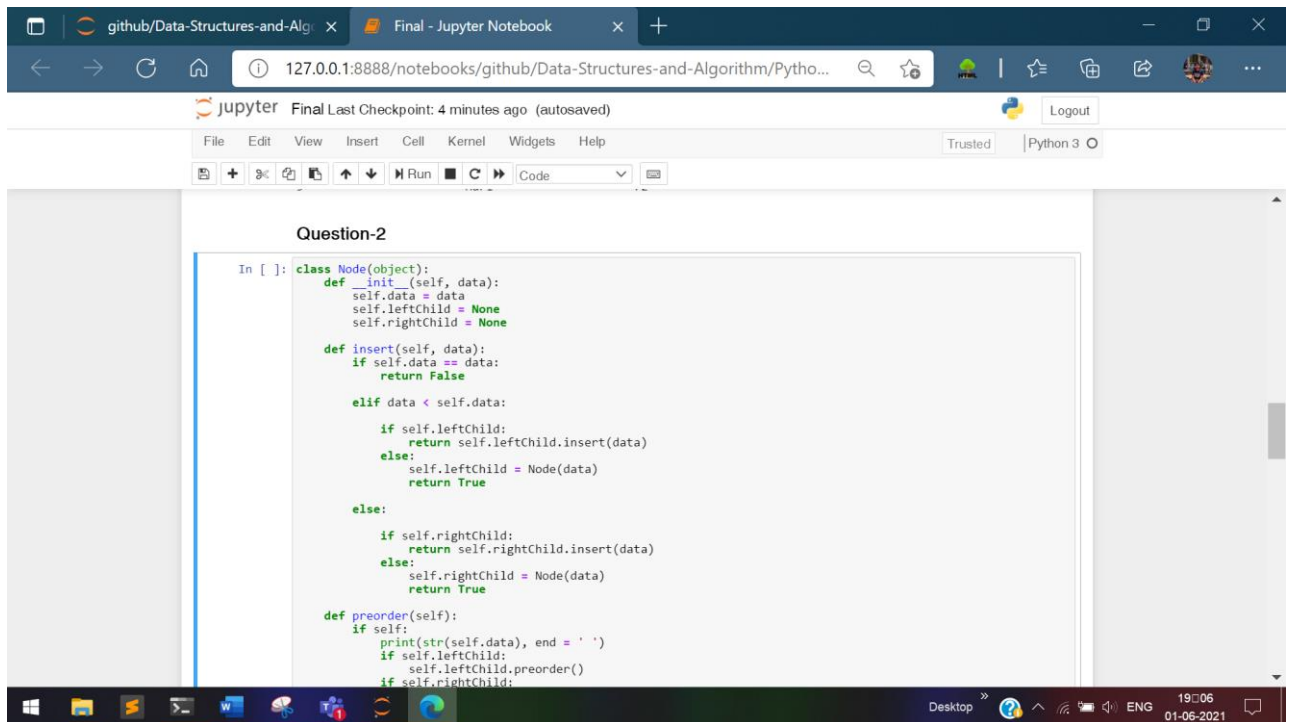
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
8 Ruby 83
10 UMA 80
4 Devi 72
Unsorted Combined Table :
Roll_no Name Marks
5 Khanna 100
1 Anand 85
7 Santhosh 80
9 Selvi 80
3 Hari 72
6 Raji 100
2 Bose 85
8 Ruby 83
10 UMA 80
4 Devi 72
Sorted Combined Table :
Roll_no Name Marks
6 Raji 100
5 Khanna 100
2 Bose 85
1 Anand 85
7 Santhosh 80
8 Ruby 83
10 Selvi 80
9 Devi 72
3 Hari 72
```

Question-2

```
In [ ]: class Node(object):
def __init__(self, data):
self.data = data
self.leftChild = None
```

Question-2

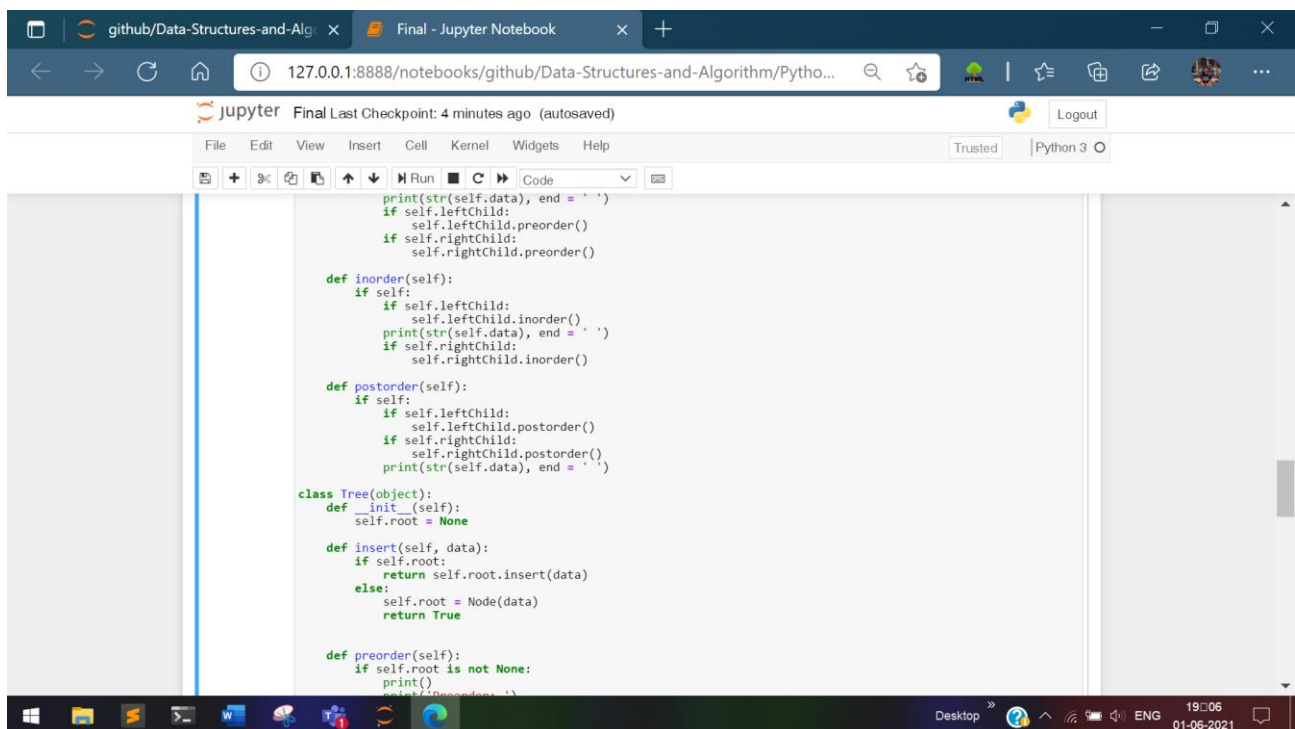


The image shows a Jupyter Notebook interface with a browser window at the top. The notebook is titled "Final - Jupyter Notebook" and is running on a server at 127.0.0.1:8888. The notebook content is titled "Question-2" and contains a Python code cell. The code defines a `Node` class with methods `__init__`, `insert`, and `preorder`. The `insert` method checks if the data is already in the tree and then inserts it as a left or right child. The `preorder` method prints the data and recursively calls itself on the left and right children.

```
In [ ]: class Node(object):
    def __init__(self, data):
        self.data = data
        self.leftChild = None
        self.rightChild = None

    def insert(self, data):
        if self.data == data:
            return False
        elif data < self.data:
            if self.leftChild:
                return self.leftChild.insert(data)
            else:
                self.leftChild = Node(data)
                return True
        else:
            if self.rightChild:
                return self.rightChild.insert(data)
            else:
                self.rightChild = Node(data)
                return True

    def preorder(self):
        if self:
            print(str(self.data), end = ' ')
            if self.leftChild:
                self.leftChild.preorder()
            if self.rightChild:
                self.rightChild.preorder()
```



The image shows a Jupyter Notebook interface with a browser window at the top. The notebook is titled "Final - Jupyter Notebook" and is running on a server at 127.0.0.1:8888. The notebook content is titled "Question-2" and contains a Python code cell. The code defines a `Tree` class with methods `__init__`, `insert`, `preorder`, `inorder`, and `postorder`. The `insert` method adds a new node to the tree. The `preorder`, `inorder`, and `postorder` methods traverse the tree and print the data in the respective order.

```
print(str(self.data), end = ' ')
if self.leftChild:
    self.leftChild.preorder()
if self.rightChild:
    self.rightChild.preorder()

def inorder(self):
    if self:
        if self.leftChild:
            self.leftChild.inorder()
        print(str(self.data), end = ' ')
        if self.rightChild:
            self.rightChild.inorder()

def postorder(self):
    if self:
        if self.leftChild:
            self.leftChild.postorder()
        if self.rightChild:
            self.rightChild.postorder()
        print(str(self.data), end = ' ')

class Tree(object):
    def __init__(self):
        self.root = None

    def insert(self, data):
        if self.root:
            return self.root.insert(data)
        else:
            self.root = Node(data)
            return True

    def preorder(self):
        if self.root is not None:
            print()
            self.root.preorder()
```

```
github/Data-Structures-and-Alg x Final - Jupyter Notebook x +
127.0.0.1:8888/notebooks/github/Data-Structures-and-Algorithm/Pytho...
Jupyter Final Last Checkpoint: 4 minutes ago (autosaved) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
Code
def preorder(self):
    if self.root is not None:
        print()
        print('Preorder: ')
        self.root.preorder()

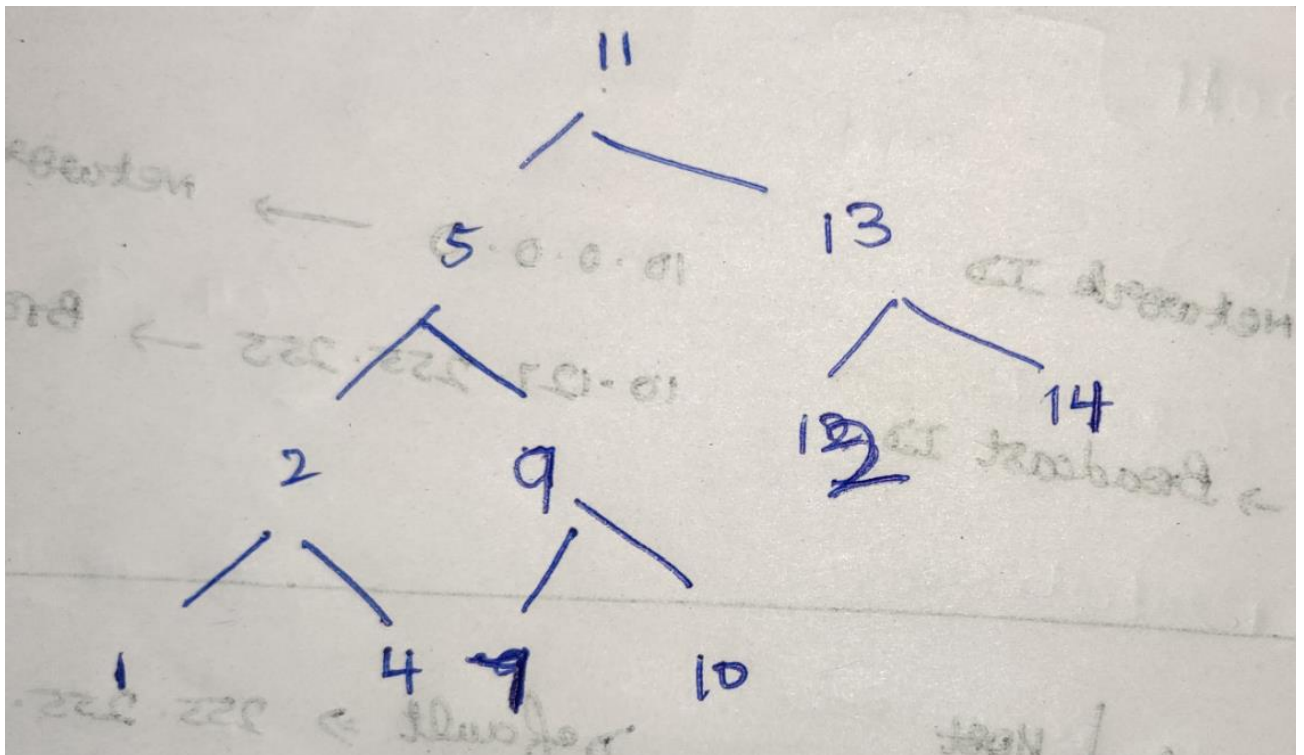
def inorder(self):
    print()
    if self.root is not None:
        print('Inorder: ')
        self.root.inorder()

def postorder(self):
    print()
    if self.root is not None:
        print('Postorder: ')
        self.root.postorder()

if __name__ == '__main__':
    tree = Tree()
    tree.insert(11)
    tree.insert(5)
    tree.insert(13)
    tree.insert(7)
    tree.insert(10)
    tree.insert(14)
    tree.insert(2)
    tree.insert(4)
    tree.insert(1)
    tree.insert(9)
    tree.insert(12)

    tree.preorder()
    tree.inorder()
    tree.postorder()
```

Creating the graph from BST



The screenshot shows a Jupyter Notebook with two code cells. The first cell defines a `Graph` class with methods `__init__`, `addEdge`, `DFSUtil`, and `DFS`. The second cell creates an instance `obj1` and adds several edges to the graph.

```
tree.preorder()
tree.inorder()
tree.postorder()

In [36]: from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self, u, v):
        self.graph[u].append(v)

    def DFSUtil(self, v, visited):
        visited.add(v)
        print(v, end=' ')

        for neighbour in self.graph[v]:
            if neighbour not in visited:
                self.DFSUtil(neighbour, visited)

    def DFS(self, v):
        list1 = []
        visited = set()
        self.DFSUtil(v, visited)

In [37]: obj1 = Graph()
obj1.addEdge(11,5)
obj1.addEdge(11,13)

obj1.addEdge(5,2)
obj1.addEdge(5,9)

obj1.addEdge(2,1)
obj1.addEdge(2,4)
```

Searching whether the give elements are there in the bst

The screenshot shows a Jupyter Notebook with three code cells. The first cell continues adding edges to the graph and performs a DFS starting from node 11. The second cell defines two lists: `bst_list1_dfs` and `given_list`. The third cell iterates through `given_list` to check if each element is present in `bst_list1_dfs`.

```
obj1.addEdge(5,2)
obj1.addEdge(5,9)

obj1.addEdge(2,1)
obj1.addEdge(2,4)

obj1.addEdge(9,7)
obj1.addEdge(9,10)

obj1.addEdge(13,12)
obj1.addEdge(13,14)

obj1.DFS(11)
11 5 2 1 4 9 7 10 13 12 14

In [38]: bst_list1_dfs = [11,5,2,1,4,9,7,10,13,12,14]
given_list = [5,6,10,13,14,16]

In [39]: for i in given_list:
        cnt = 0
        for j in bst_list1_dfs:
            if (i==j):
                print("{} is found in the bst tree".format(j))
                break

5 is found in the bst tree
10 is found in the bst tree
13 is found in the bst tree
14 is found in the bst tree

In [ ]:
```

Sir, I have attached my Source Code in the below link

<https://drive.google.com/file/d/1BJO4yLgKZoKdXMYZv9EznQY-v-X5Oauz/view?usp=sharing>