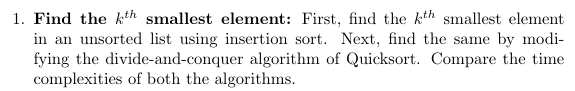
**ASSIGNMENT – 5**

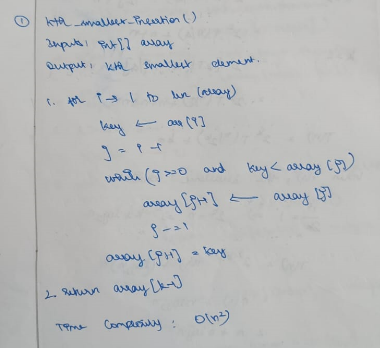
**AIM:**

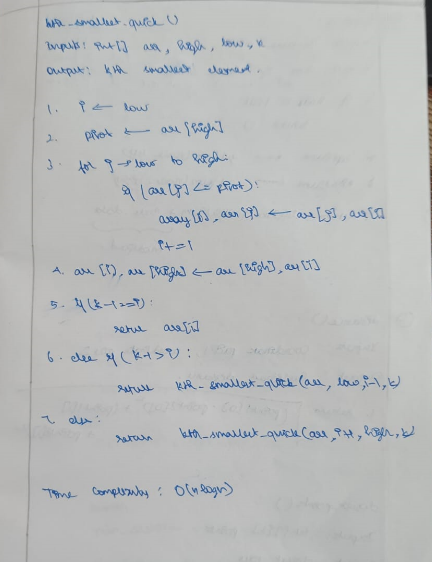
To solve and implement the given problems using Divide and Conquer Strategies.

**Qn1:**

****

**Psuedo Code:**

****

****

**Source Code:**

def findKInsertion(arr, k):

    if k<1 or k>len(arr):

        return "Invalid k"

    for i in range(1, len(arr)):

        key = arr[i]

        j = i-1

        while j>=0 and key<arr[j]:

            arr[j+1] = arr[j]

            j -= 1

        arr[j+1] = key

    return arr[k-1]

def partition(arr, low, high):

    pivot = arr[low]

    left = low + 1

    right = high

    while left <= right:

        while left <= right and arr[left] <= pivot:

            left = left + 1

        while arr[right] >= pivot and right >= left:

            right = right - 1

        if left <= right:

            arr[left], arr[right] = arr[right], arr[left]

    arr[low], arr[right] = arr[right], arr[low]

    return right

def findKQuick(arr, k, low, high):

    if k<1 or k>len(arr):

        return "Invalid k"

    if low < high:

        pivotIndex = partition(arr, low, high)

        findKQuick(arr, k, low, pivotIndex-1)

        findKQuick(arr, k-pivotIndex+low, pivotIndex+1, high)

        return arr[k-1]

l = [4,6,4,3,7,8,6,7,3,4,1]

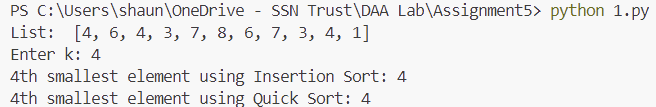
print("List: ", l)

k = int(input("Enter k: "))

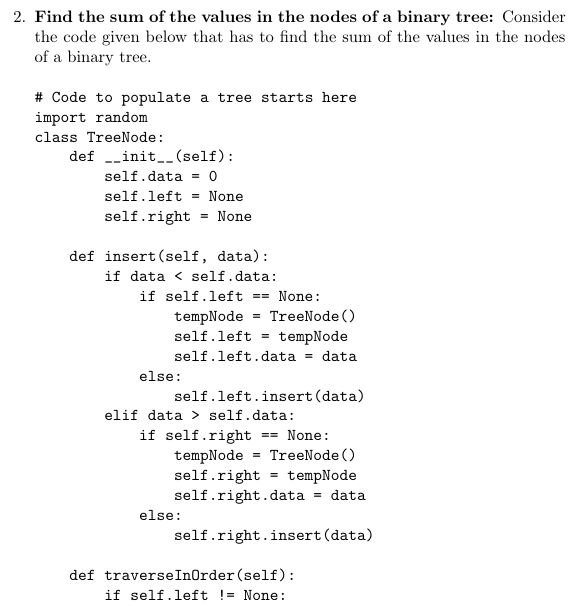
print(f"{k}th smallest element using Insertion Sort: {findKInsertion(l, k)}")

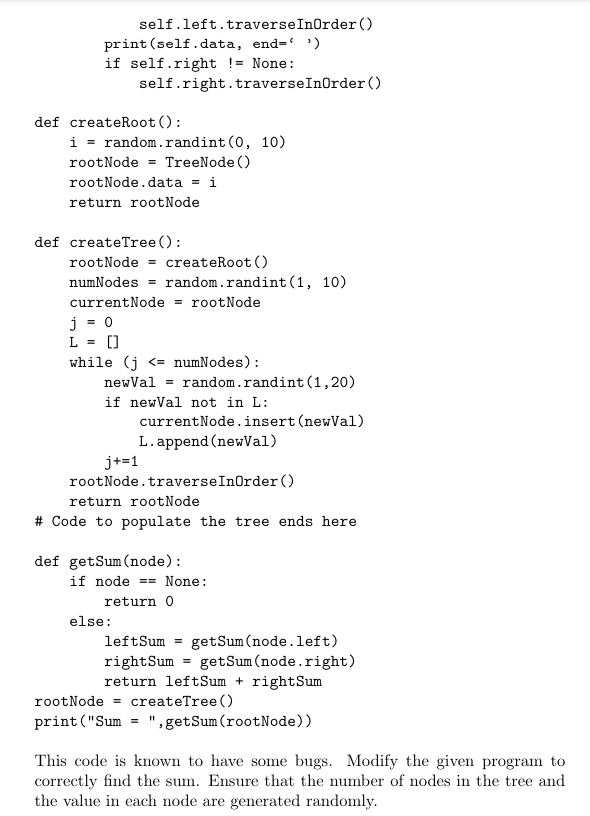
print(f"{k}th smallest element using Quick Sort: {findKQuick(l, k, 0, len(l)-1)}")

**Output:**

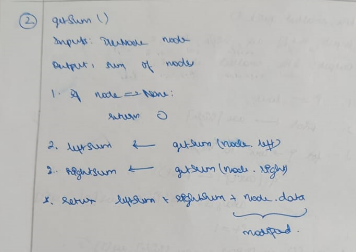
****

**Qn2:**

****

****

**Psuedo Code:**

****

**Source Code:**

import random

class TreeNode:

    def \_\_init\_\_(self):

        self.data = 0

        self.left = None

        self.right = None

    def insert(self, data):

        if data < self.data:

            if self.left == None:

                tempNode = TreeNode()

                self.left = tempNode

                self.left.data = data

            else:

                self.left.insert(data)

        elif data > self.data:

            if self.right == None:

                tempNode = TreeNode()

                self.right = tempNode

                self.right.data = data

            else:

                self.right.insert(data)

    def traverseInOrder(self):

        if self.left != None:

            self.left.traverseInOrder()

        print(self.data, end=" ")

        if self.right != None:

            self.right.traverseInOrder()

def createRoot():

    i = random.randint(0, 10)

    rootNode = TreeNode()

    rootNode.data = i

    return rootNode

def createTree():

    rootNode = createRoot()

    numNodes = random.randint(1, 10)

    currentNode = rootNode

    j = 0

    L = []

    while j <= numNodes:

        newVal = random.randint(1,20)

        if newVal not in L:

            currentNode.insert(newVal)

            L.append(newVal)

            j+=1

    rootNode.traverseInOrder()

    return rootNode

def getSum(node):

    if node == None:

        return 0

    else:

        leftSum = getSum(node.left)

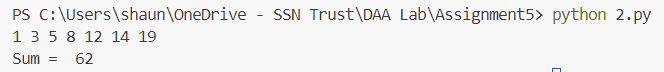
        rightSum = getSum(node.right)

        return leftSum + rightSum + node.data

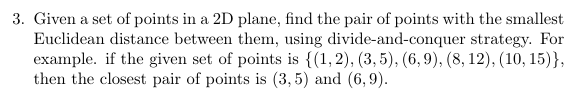
rootNode = createTree()

print("\nSum = ", getSum(rootNode))

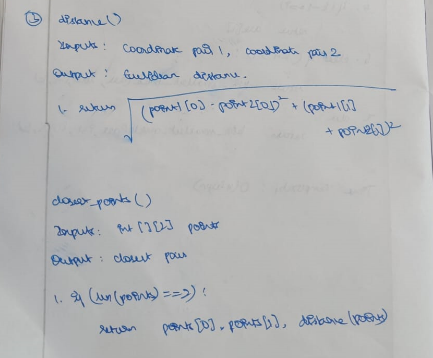
**Output:**

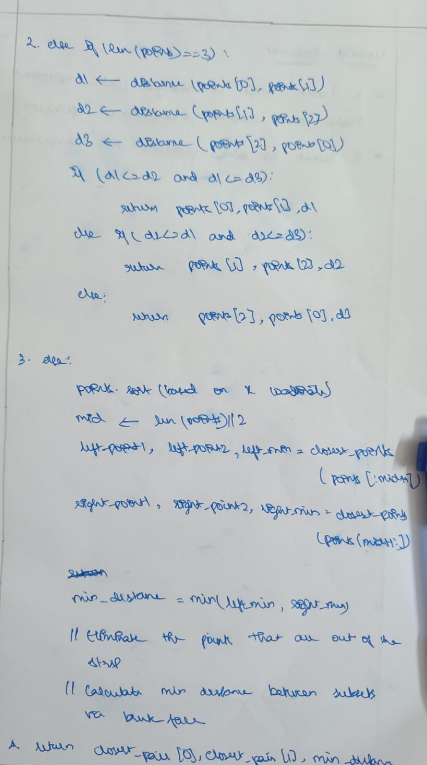
****

**Qn3:**

****

**Psuedo Code:**

****

****

**Source Code:**

def distance(p1, p2):

    return ((p1[0]-p2[0])\*\*2 + (p1[1]-p2[1])\*\*2)\*\*0.5

def stripClosest(strip, d, p1, p2):

    min\_dist = d

    closest\_pair = (p1, p2)

    strip = sorted(strip, key = lambda x: x[1])

    for i in range(len(strip)):

        for j in range(i+1, len(strip)):

            if (strip[j][1] - strip[i][1]) >= min\_dist:

                break

            if distance(strip[i], strip[j]) < min\_dist:

                min\_dist = distance(strip[i], strip[j])

                closest\_pair = (strip[i], strip[j])

    return closest\_pair[0], closest\_pair[1], min\_dist

def closest\_pairs(points):

    if len(points) == 2:

        d = distance(points[0], points[1])

        return points[0], points[1], d

    elif len(points) == 3:

        d1 = distance(points[0], points[1])

        d2 = distance(points[1], points[2])

        d3 = distance(points[2], points[0])

        if d1<=d2 and d1<=d3:

            return points[0], points[1], d1

        elif d2<=d1 and d2<=d3:

            return points[1], points[2], d2

        else:

            return points[2], points[0], d3

    else:

        points = sorted(points, key = lambda x: x[0])

        mid = len(points)//2

        midpoint = points[mid]

        lp1, lp2, dl = closest\_pairs(points[:mid])

        rp1, rp2, dr = closest\_pairs(points[mid:])

        d = 0

        closest\_pair = ()

        if dl<dr:

            d = dl

            closest\_pair = (lp1,lp2)

        else:

            d = dr

            closest\_pair = (rp1,rp2)

        strip = []

        for i in range(len(points)):

            if abs(points[i][0] - midpoint[0]) < d:

                strip.append(points[i])

        strip1, strip2, strip\_dist = stripClosest(strip, d, closest\_pair[0], closest\_pair[1])

        if d <= strip\_dist:

            return closest\_pair[0], closest\_pair[1], d

        else:

            return strip1, strip2, strip\_dist

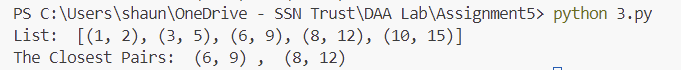
l = [(1,2),(3,5),(6,9),(8,12),(10,15)]

print("List: ", l)

p1, p2, d = closest\_pairs(l)

print("The Closest Pairs: ", p1, ", ", p2)

**Output:**

****

**Learning Outcomes:**

* I learnt to analyse and implement divided and conquer algorithms
* I learnt how to implement various sorting and searching algorithms in Python­­