Date: 18.03.24 Reg No: 3122225001127
ExNo: 5 Name: Shaun Allan H

ASSIGNMENT – 5

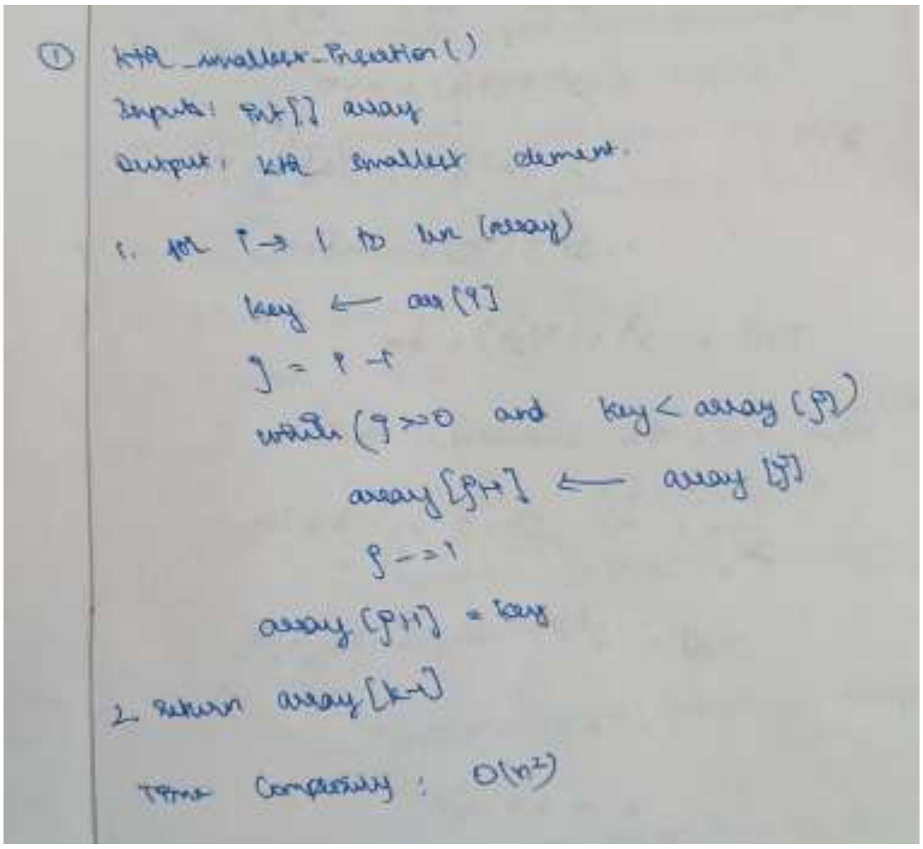
AIM:

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

Qn1:

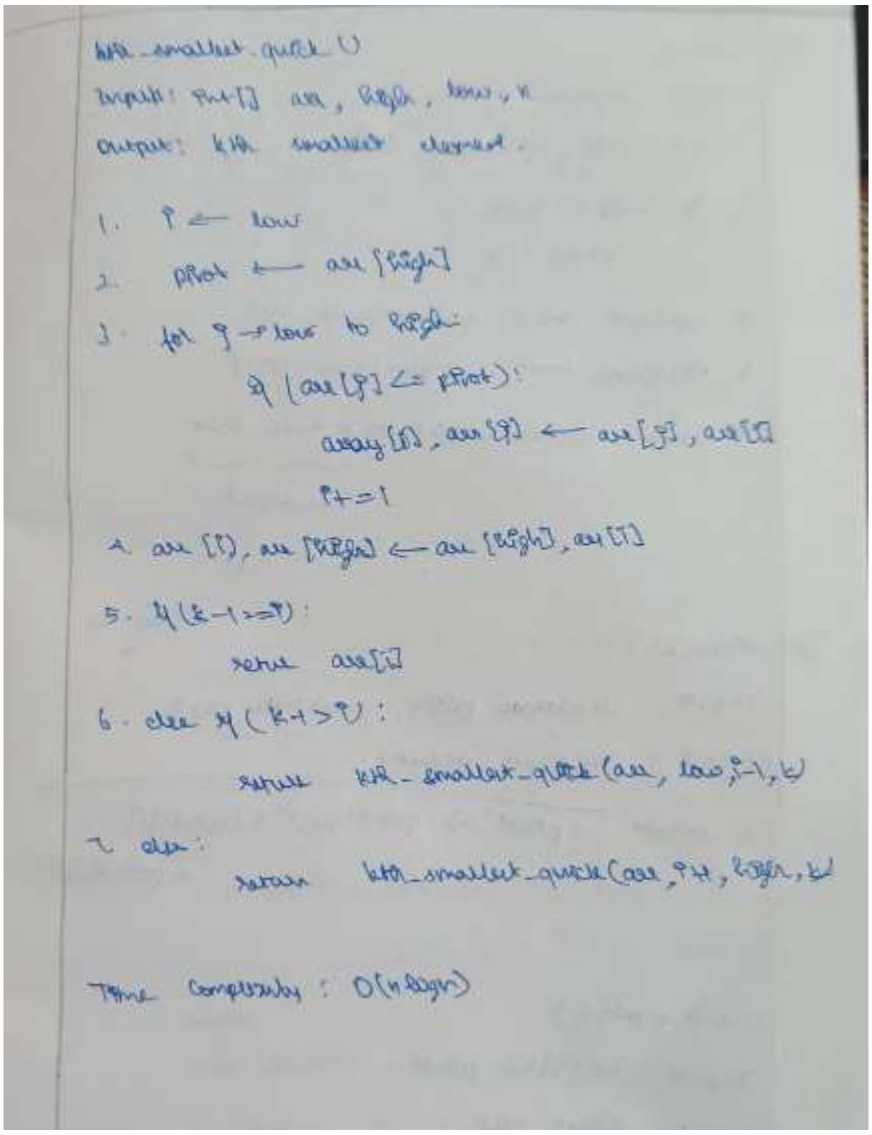
1. **Find the k^{th} smallest element:** First, find the k^{th} smallest element in an unsorted list using insertion sort. Next, find the same by modifying the divide-and-conquer algorithm of Quicksort. Compare the time complexities of both the algorithms.

Psuedo Code:



```
① kth_smallest_Insertion()
Input: int[] array
Output: kth smallest element.

1. for i -> 1 to len(array)
        key <- arr[i]
        j = i - 1
        while (j >= 0 and key < array[j])
            array[j+1] <- array[j]
            j -= 1
        array[j+1] = key
2. return array[k-1]

Time Complexity : O(n^2)
```

Date: 18.03.24 Reg No: 3122225001127
ExNo: 5 Name: Shaun Allan H

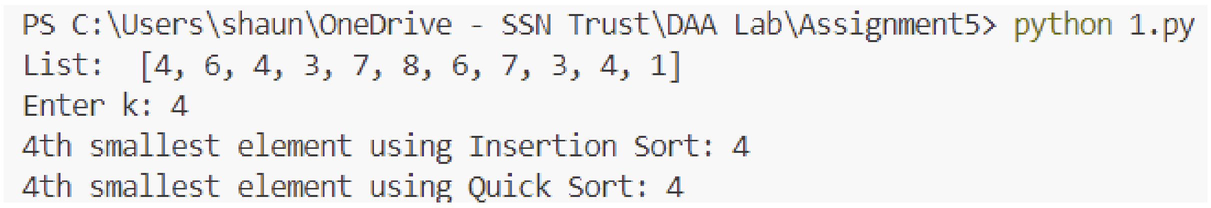


```
kth_smallest_quick()
Input: int[] arr, right, low, k
Output: kth smallest element.

1.  i ← low
2.  pivot ← arr[right]
3.  for j → low to right:
        if (arr[j] <= pivot):
            array[i], arr[j] ← arr[j], arr[i]
            i+=1
4.  arr[i], arr[right] ← arr[right], arr[i]
5.  if (k-1==i):
        return arr[i]
6.  else if (k-1>i):
        return kth_smallest_quick(arr, low, i-1, k)
7.  else:
        return kth_smallest_quick(arr, i+1, right, k)

Time Complexity: O(n logn)
```

Source Code:

```python
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:

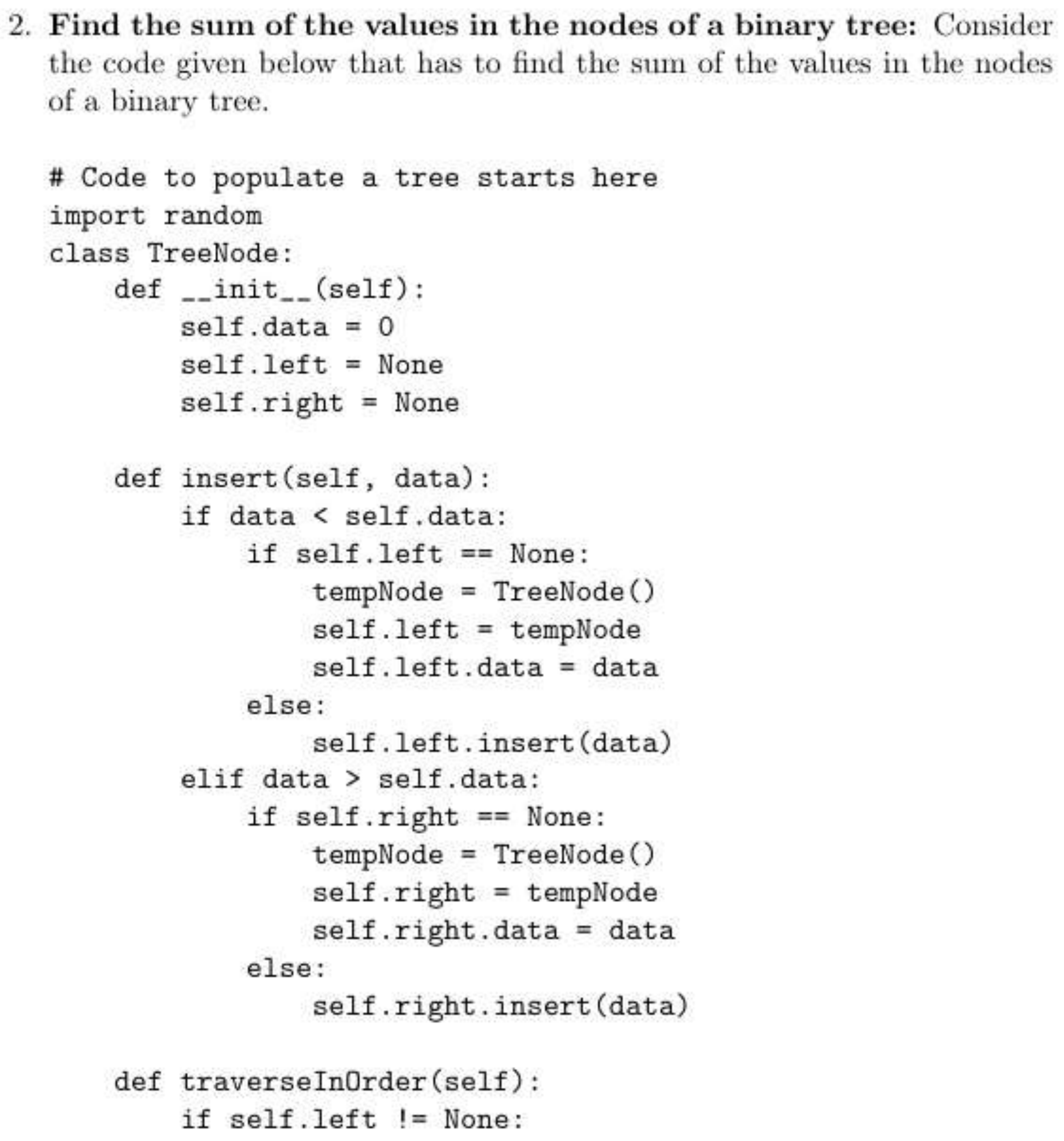


```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment5> python 1.py
List:  [4, 6, 4, 3, 7, 8, 6, 7, 3, 4, 1]
Enter k: 4
4th smallest element using Insertion Sort: 4
4th smallest element using Quick Sort: 4
```

Date: 18.03.24
ExNo: 5

Reg No: 3122225001127
Name: Shaun Allan H

Qn2:



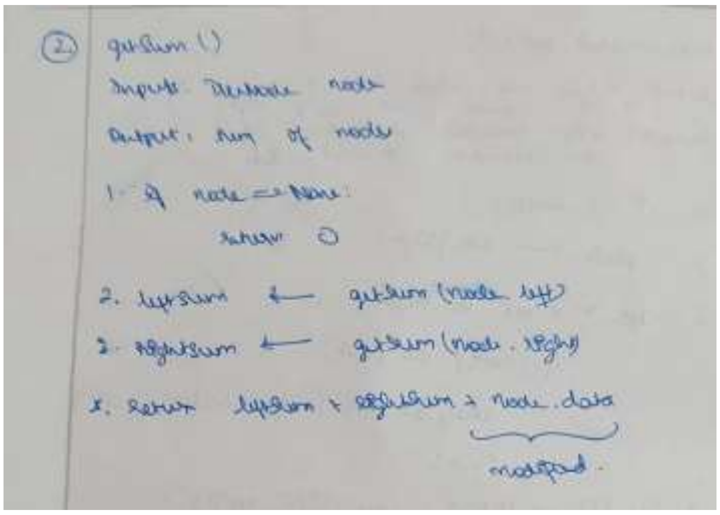
2. **Find the sum of the values in the nodes of a binary tree:** Consider the code given below that has to find the sum of the values in the nodes of a binary tree.

```
# Code to populate a tree starts here
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
# Code to populate the tree ends here

def getSum(node):
    if node == None:
        return 0
    else:
        leftSum = getSum(node.left)
        rightSum = getSum(node.right)
        return leftSum + rightSum
rootNode = createTree()
print("Sum = ",getSum(rootNode))
```

This code is known to have some bugs. Modify the given program to correctly find the sum. Ensure that the number of nodes in the tree and the value in each node are generated randomly.

Date: 18.03.24 Reg No: 3122225001127
ExNo: 5 Name: Shaun Allan H

Psuedo Code:

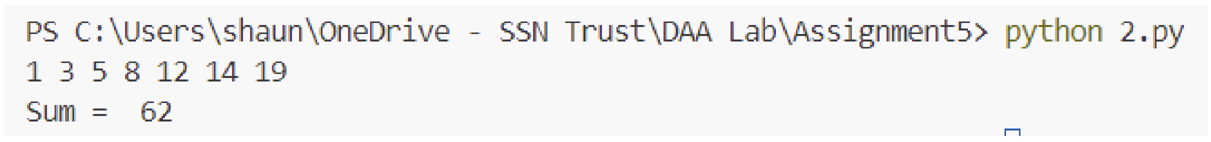


```
(2) getSum ()
Input: TreeNode node
Output: sum of nodes
1. If node == None:
        return 0
2. leftSum ← getSum (node.left)
3. rightSum ← getSum (node.right)
4. Return leftSum + rightSum + node.data
                          modified.
```

Source Code:

```python
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:



```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment5> python 2.py
1 3 5 8 12 14 19
Sum =  62
```

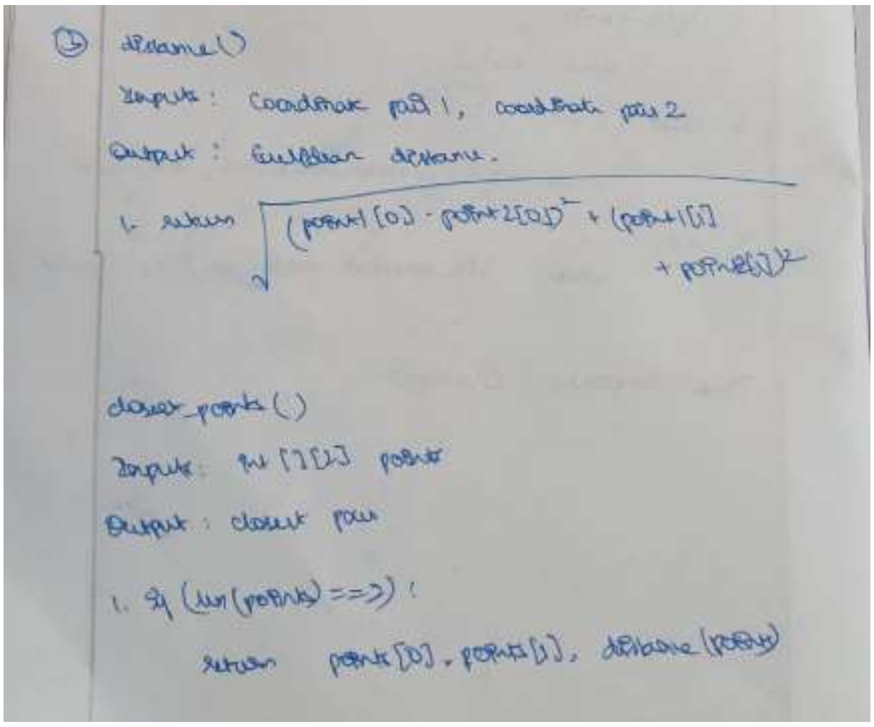
Date: 18.03.24 Reg No: 3122225001127

ExNo: 5 Name: Shaun Allan H

Qn3:

3. Given a set of points in a 2D plane, find the pair of points with the smallest Euclidean distance between them, using divide-and-conquer strategy. For example. if the given set of points is $\{(1,2),(3,5),(6,9),(8,12),(10,15)\}$, then the closest pair of points is $(3,5)$ and $(6,9)$.

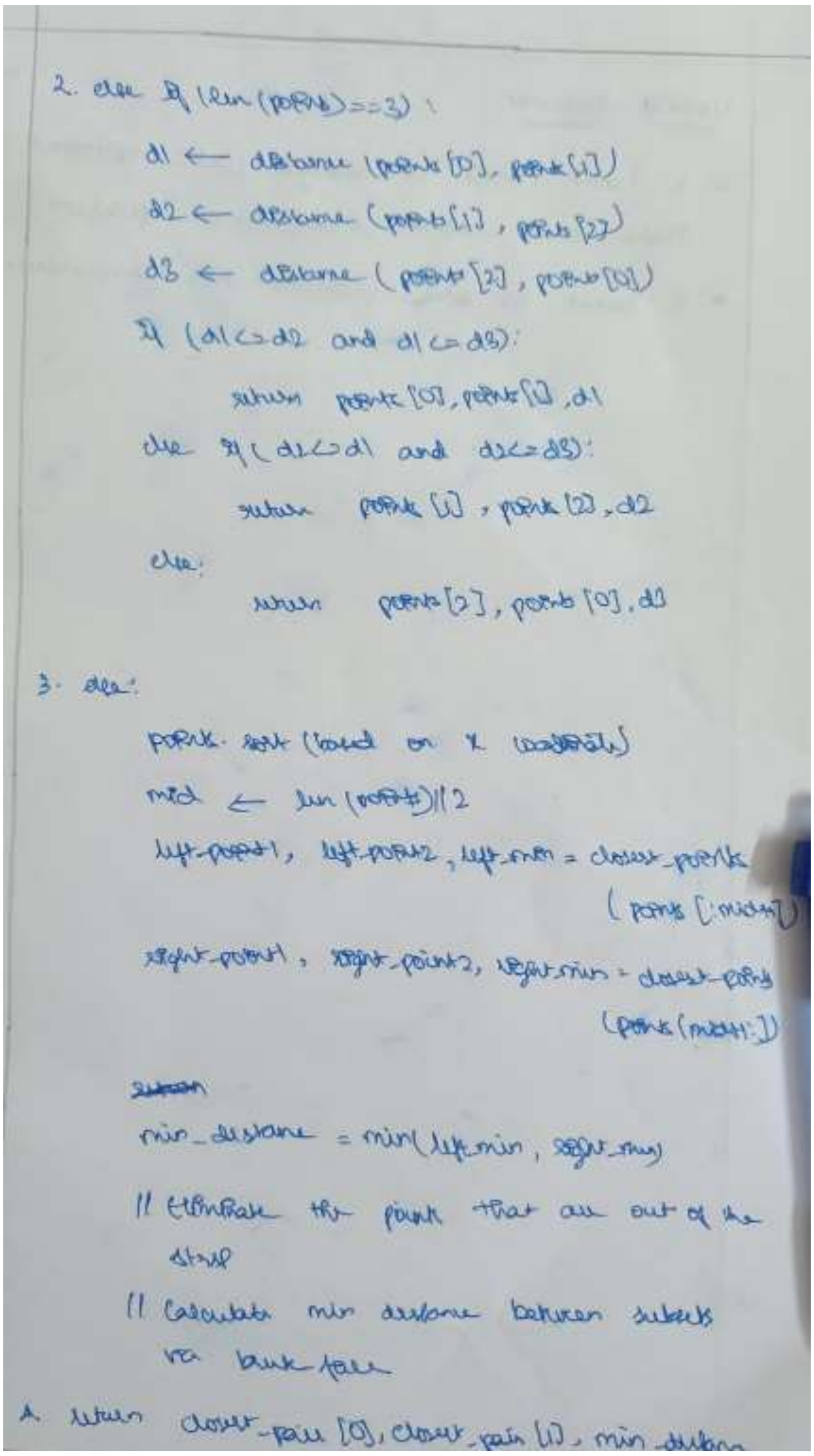
Psuedo Code:



```
3 distance()
Input: Coordinate pair 1, coordinate pair 2
Output: Euclidean distance.
1. return $\sqrt{(point1[0] - point2[0])^2 + (point1[1] + point2[1])^2}$

closest_points()
Input: int [ ][2] points
Output: closest pair
1. if (len(points) == 2) :
       return points[0], points[1], distance(points)
```

Date: 18.03.24
ExNo: 5

Reg No: 3122225001127
Name: Shaun Allan H



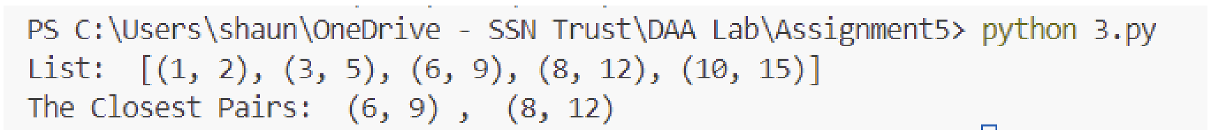
```
2. else if (len(points)==3):
       d1 ← distance (points[0], points[1])
       d2 ← distance (points[1], points[2])
       d3 ← distance (points[2], points[0])
       if (d1<=d2 and d1<=d3):
            return  points[0], points[1], d1
       else if (d2<=d1 and d2<=d3):
            return  points[1], points[2], d2
       else:
            return  points[2], points[0], d3

3. else:
       points.sort (based on x coordinate)
       mid ← len(points)//2
       left_point1, left_point2, left_min = closest_points
                                          (points[:mid])
       right_point1, right_point2, right_min = closest_points
                                          (points[mid:])

      ~~return~~
      min_distance = min(left_min, right_min)
      // Eliminate the points that are out of the
         strip
      // Calculate min distance between subsets
         via brute force
4. return closest_pair[0], closest_pair[1], min_distance
```

Source Code:

```python
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])
```

```python
        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)
```

<u>Output:</u>



```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment5> python 3.py
List:  [(1, 2), (3, 5), (6, 9), (8, 12), (10, 15)]
The Closest Pairs:  (6, 9) ,  (8, 12)
```

<u>Learning Outcomes:</u>

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