**Exercise - 4**

**Closest Pair Problem**

**Aim:** To write a python code to implement Closest Pair Problem

**Algorithm:**

1. Find the middle point in the sorted array, we can take *P[n/2]* as middle point.
2. Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].
3. Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be d.
4. From the above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from the left half and the other is from the right half. Consider the vertical line passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.
5. Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging.
6. Find the smallest distance in strip[]. This is tricky. From the first look, it seems to be a O(n^2) step, but it is actually O(n). It can be proved geometrically that for every point in the strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). See [this](http://people.csail.mit.edu/indyk/6.838-old/handouts/lec17.pdf)for more analysis.
7. Finally return the minimum of d and distance calculated in the above step (step 6)

**Source Code:**

import math

class Point:

    def \_\_init\_\_(self, x, y):

        self.x = x

        self.y = y

def compareX(a, b):

    p1 = a

    p2 = b

    return (p1.x - p2.x)

def compareY(a, b):

    p1 = a

    p2 = b

    return (p1.y - p2.y)

def dist(p1, p2):

    return math.sqrt((p1.x - p2.x)\*(p1.x - p2.x) + (p1.y - p2.y)\*(p1.y - p2.y))

def bruteForce(P, n):

    min\_dist = float("inf")

    for i in range(n):

        for j in range(i+1, n):

            if dist(P[i], P[j]) < min\_dist:

                min\_dist = dist(P[i], P[j])

    return min\_dist

def min(x, y):

    return x if x < y else y

def stripClosest(strip, size, d):

    min\_dist = d

    strip = sorted(strip, key=lambda point: point.y)

    for i in range(size):

        for j in range(i+1, size):

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

                break

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

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

    return min\_dist

def closestUtil(P, n):

    if n <= 3:

        return bruteForce(P, n)

    mid = n//2

    midPoint = P[mid]

    dl = closestUtil(P, mid)

    dr = closestUtil(P[mid:], n - mid)

    d = min(dl, dr)

    strip = []

    for i in range(n):

        if abs(P[i].x - midPoint.x) < d:

            strip.append(P[i])

    return min(d, stripClosest(strip, len(strip), d))

def closest(P, n):

    P = sorted(P, key=lambda point: point.x)

    return closestUtil(P, n)

if \_\_name\_\_ == "\_\_main\_\_":

    n=int(input("Enter the no.of points: "))

    P=[]

    for i in range(n):

        x=int(input("Enter x point: "))

        y=int(input("Enter y point: "))

        print()

        P.append(Point(x,y))

    print("The smallest distance is:", closest(P, n))

**Sample Input and Output:**

A screenshot of a computer program

Description automatically generated with medium confidence

**Result:**

Thus, the closest pair problem has been successfully implemented using Python code and the output is verified.