AA_LAB_06_Assignment

CE_054

Aim :- Implementation of Line Segment intersect Algorithm and Closest pair among all point with brute-force approach.

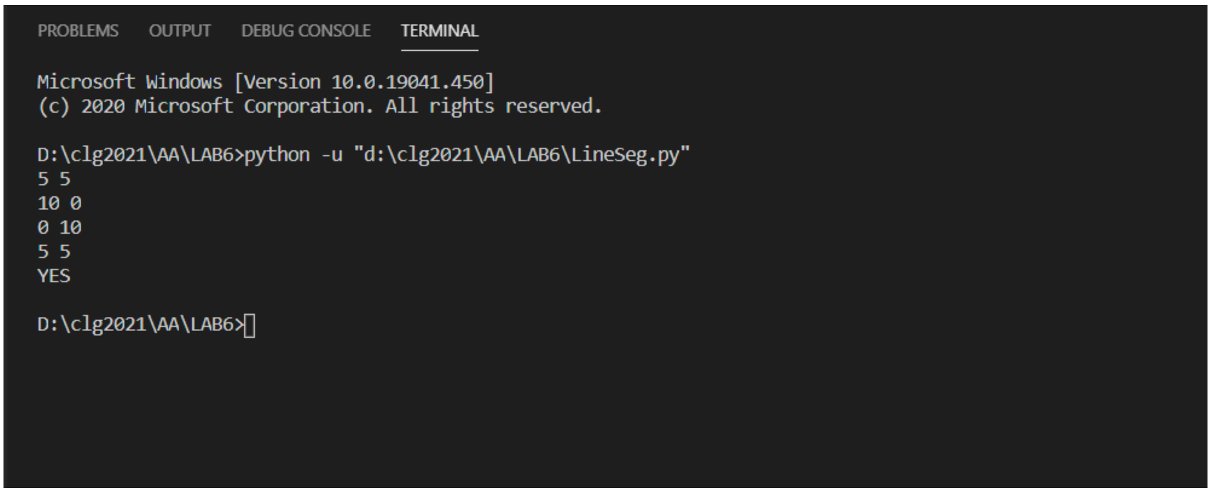
1. Write a Program to check whether two line segments intersect or not.

   Code :-

```python
      # -*- coding: utf-8 -*-
"""
Created on Fri Aug 14 16:05:58 2020

@author: DHRUV
"""


class store_points:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def onsegment(p, q, r):
    if ( (min(p.x, q.x) <= r.x <= max(p.x, q.x)) and (min(p.y, q.y) <= r.y <=
max(p.y, q.y)) ):
        return True
    return False

def find_direction(p, q, r):
    Rx, Ry = (r.x - p.x), (r.y - p.y)
    Qx, Qy = (q.x - p.x), (q.y - p.y)
    dist = (Rx * Qy) - (Qx * Ry)
    if (dist == 0):
        return 0
    elif (dist > 0):
        return 1
    else:
        return -1

def intersection(p1, p2, p3, p4):
    d1 = find_direction(p3, p4, p1)
    d2 = find_direction(p3, p4, p2)
    d3 = find_direction(p1, p2, p3)
    d4 = find_direction(p1, p2, p4)

    if (d1 * d2 < 0 and d3 * d4 < 0):
        return True
    elif (d1 == 0 and onsegment(p3, p4, p1)):
        return True
    elif (d2 == 0 and onsegment(p3, p4, p2)):
        return True
    elif (d3 == 0 and onsegment(p1, p2, p3)):
        return True
    elif (d4 == 0 and onsegment(p1, p2, p4)):
        return True
    else:
        return False

if __name__ == "__main__":
    a, b = map(int, input().split())
    p1 = store_points(a, b)
    a, b = map(int, input().split())
    p2 = store_points(a, b)
    a, b = map(int, input().split())
    p3 = store_points(a, b)
    a, b = map(int, input().split())
    p4 = store_points(a, b)

    if (intersection(p1, p2, p3, p4)):
        print("YES")
    else:
        print("NO")
```

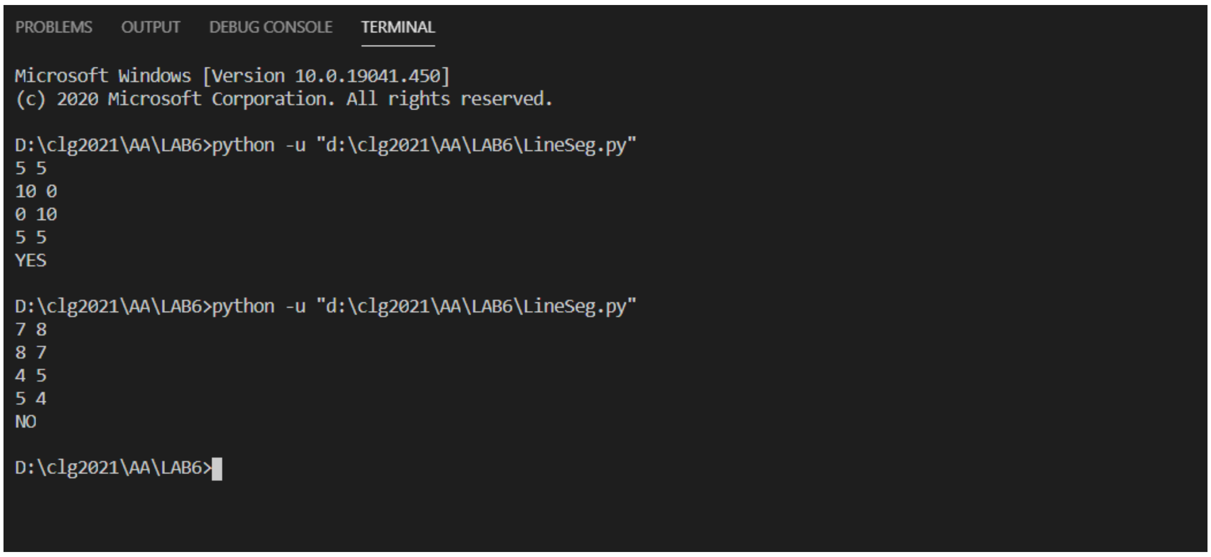
Output :-

1)



```
Microsoft Windows [Version 10.0.19041.450]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\clg2021\AA\LAB6>python -u "d:\clg2021\AA\LAB6\LineSeg.py"
5 5
10 0
0 10
5 5
YES

D:\clg2021\AA\LAB6>
```

2)



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Microsoft Windows [Version 10.0.19041.450]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\clg2021\AA\LAB6>python -u "d:\clg2021\AA\LAB6\LineSeg.py"
5 5
10 0
0 10
5 5
YES

D:\clg2021\AA\LAB6>python -u "d:\clg2021\AA\LAB6\LineSeg.py"
7 8
8 7
4 5
5 4
NO

D:\clg2021\AA\LAB6>
```

2. Write a Program to find the closest pair of points using Brute Force approach.

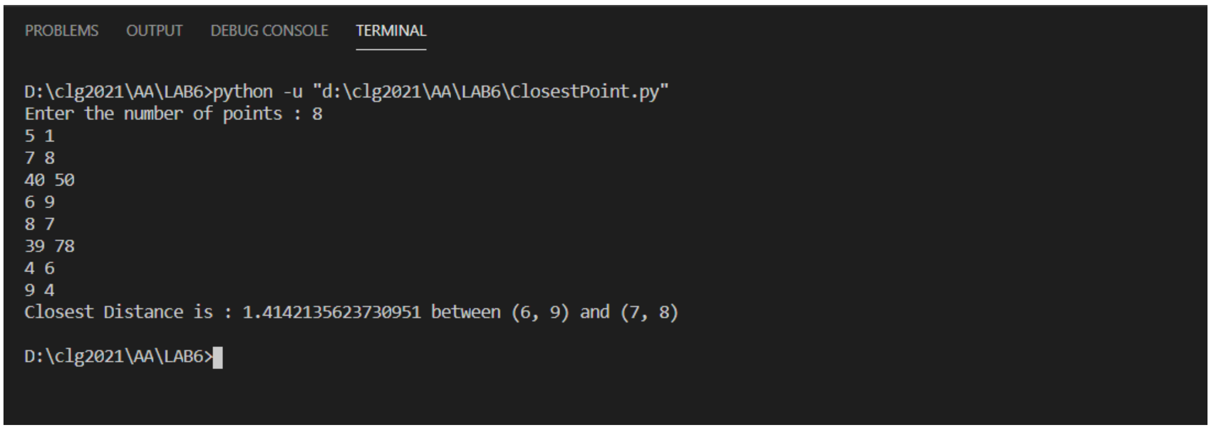
   Code :-

```python
      # -*- coding: utf-8 -*-
"""
Created on Fri Sep 4 16:36:45 2020

@author: DHRUV
"""

import math as m

class points():
    def __init__(self, x, y):
        self.x = x
        self.y = y

def distance(p1, p2):
    return m.sqrt(((p1.x - p2.x) ** 2) + ((p1.y - p2.y) ** 2))

def find_distance(p, n):
    min_ = float('inf')
    for i in range(n):
        for j in range(i + 1, n):
            if (distance(p[i], p[j]) < min_):
                min_ = distance(p[i], p[j])
                point1, point2 = (p[i].x, p[i].y), (p[j].x, p[j].y)
    return min_, point1, point2
```

```python
def closest_point(p, n):
    p.sort(key = lambda point : point.x)
    return find_distance(p, n)

P = []
point = int(input("Enter the number of points : "))
for _ in range(point):
    a, b = map(int, input().split())
    P.append(points(a, b))

closest_distance, point1, point2 = closest_point(P, len(P))
print(f"Closest Distance is : {closest_distance} between {point1} and {point2}
")
```

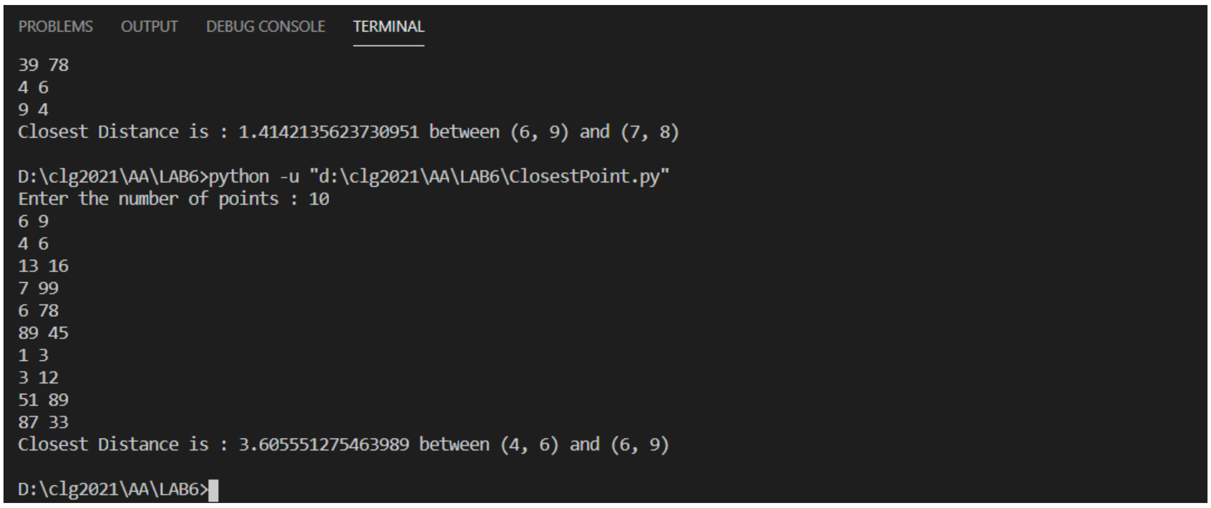
Output :-

1)



```
D:\clg2021\AA\LAB6>python -u "d:\clg2021\AA\LAB6\ClosestPoint.py"
Enter the number of points : 8
5 1
7 8
40 50
6 9
8 7
39 78
4 6
9 4
Closest Distance is : 1.4142135623730951 between (6, 9) and (7, 8)

D:\clg2021\AA\LAB6>
```

2)



```
39 78
4 6
9 4
Closest Distance is : 1.4142135623730951 between (6, 9) and (7, 8)

D:\clg2021\AA\LAB6>python -u "d:\clg2021\AA\LAB6\ClosestPoint.py"
Enter the number of points : 10
6 9
4 6
13 16
7 99
6 78
89 45
1 3
3 12
51 89
87 33
Closest Distance is : 3.605551275463989 between (4, 6) and (6, 9)

D:\clg2021\AA\LAB6>
```

Theory regarding this two algorithm:-

- Simple algorithms examine each pair of segments. However, if a large number of possibly intersecting segments are to be checked, this becomes increasingly inefficient since most pairs of segments are not close to one another in a typical input sequence. The most common, and more efficient, way to solve this problem for a high number of segments is to use a sweep line Algorithm. where we imagine a line sliding across the line segments and we track which line segments it intersects at each point in time using a dynamic data structure based on binary search trees.

- We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. The Brute force solution is O(n^2), compute the distance between each pair and return the smallest.