***DESIGN AND ANALYSIS OF ALGORITHMS PROJECT***

**MEMBERS :-**

* SHARJEEL SAFDAR (22K4210)
* UZAIR AHMED KHAN (22K4189)
* SUHAIB SHAIKH (22K4302)

**QUESTION 2)**

import random

import math

import os

def karatsuba(x, y):

if x < 10 or y < 10:

return x \* y

num\_digits = max(len(str(x)), len(str(y)))

split = num\_digits // 2

high1, low1 = divmod(x, 10 \*\* split)

high2, low2 = divmod(y, 10 \*\* split)

z0 = karatsuba(low1, low2)

z1 = karatsuba(low1 + high1, low2 + high2)

z2 = karatsuba(high1, high2)

result = z2 \* 10 \*\* (2 \* split) + (z1 - z2 - z0) \* 10 \*\* split + z0

return result

def calc\_distance(p1, p2):

return math.sqrt((p1[0] - p2[0])\*\*2 + (p1[1] - p2[1])\*\*2)

def closest\_pair(points):

points.sort(key=lambda p: p[0])

def recursive\_find(points):

if len(points) <= 3:

return brute\_force(points)

mid = len(points) // 2

left = points[:mid]

right = points[mid:]

dist\_left, p1\_left, p2\_left = recursive\_find(left)

dist\_right, p1\_right, p2\_right = recursive\_find(right)

min\_dist = min(dist\_left, dist\_right)

closest\_pair = (p1\_left, p2\_left) if min\_dist == dist\_left else (p1\_right, p2\_right)

cross\_dist, cross\_p1, cross\_p2 = merge\_check(points, points[mid], min\_dist)

if cross\_dist < min\_dist:

return cross\_dist, cross\_p1, cross\_p2

else:

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

return recursive\_find(points)

def brute\_force(points):

min\_dist = float('inf')

closest\_pair = (None, None)

for i in range(len(points)):

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

dist = calc\_distance(points[i], points[j])

if dist < min\_dist:

min\_dist = dist

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

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

def merge\_check(points, mid\_point, min\_dist):

strip = [p for p in points if abs(p[0] - mid\_point[0]) < min\_dist]

strip.sort(key=lambda p: p[1])

min\_dist\_across = min\_dist

best\_pair = (None, None)

for i in range(len(strip)):

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

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

break

dist = calc\_distance(strip[i], strip[j])

if dist < min\_dist\_across:

min\_dist\_across = dist

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

return min\_dist\_across, best\_pair[0], best\_pair[1]

def generate\_closest\_pair\_data(file\_name, num\_points):

points = [(random.randint(0, 1000), random.randint(0, 1000)) for \_ in range(num\_points)]

if os.path.exists(file\_name):

os.remove(file\_name)

with open(file\_name, 'w') as file:

for point in points:

file.write(f"{point[0]} {point[1]}\n")

def generate\_multiplication\_data(file\_name, num\_tests):

if os.path.exists(file\_name):

os.remove(file\_name)

with open(file\_name, 'w') as file:

for \_ in range(num\_tests):

x = random.randint(10 \* (num\_tests - 1), 10 \* num\_tests - 1)

y = random.randint(10 \* (num\_tests - 1), 10 \* num\_tests - 1)

file.write(f"{x} {y}\n")

for i in range(10):

generate\_closest\_pair\_data(f"closest\_pair\_input\_{i+1}.txt", 200)

generate\_multiplication\_data(f"int\_multiplication\_input\_{i+1}.txt", 5)

**QUESTION 3)**

def apply\_closest\_pair(file\_name):

points = []

with open(file\_name, 'r') as file:

for line in file:

x, y = map(int, line.strip().split())

points.append((x, y))

dist, p1, p2 = closest\_pair(points)

print(f"Closest pair in {file\_name}: {p1}, {p2} with distance {dist}")

def apply\_karatsuba(file\_name):

with open(file\_name, 'r') as file:

for line in file:

x, y = map(int, line.strip().split())

result = karatsuba(x, y)

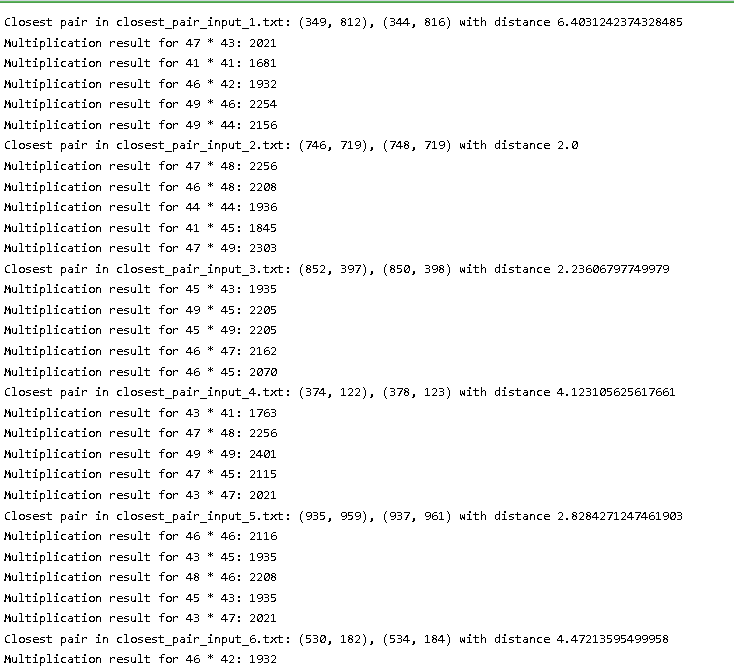
print(f"Multiplication result for {x} \* {y}: {result}")

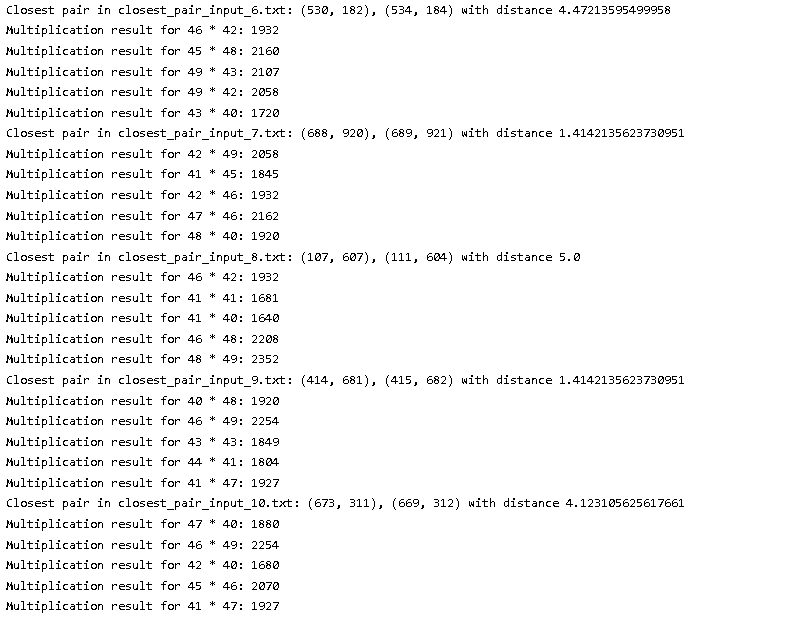
for i in range(10):

apply\_closest\_pair(f"closest\_pair\_input\_{i+1}.txt")

apply\_karatsuba(f"int\_multiplication\_input\_{i+1}.txt")

**OUTPUT:-**





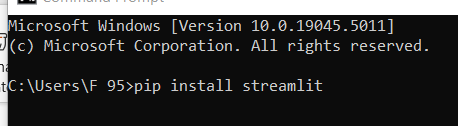
**QUESTION 3)**

GUI : Streamlit

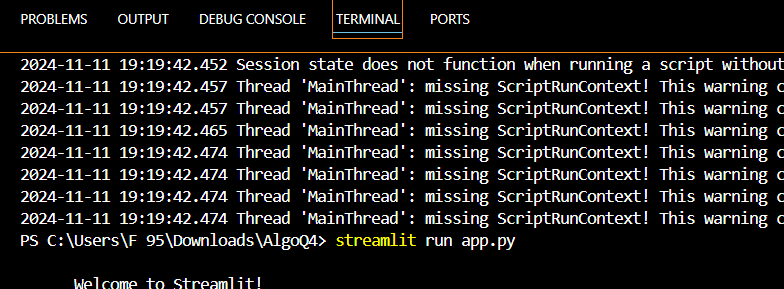
LANGUAGE : Python

IDE : Visual Studio

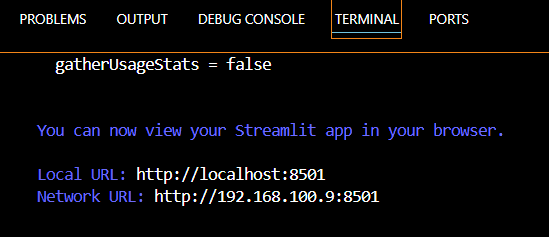
STEP 1 :-



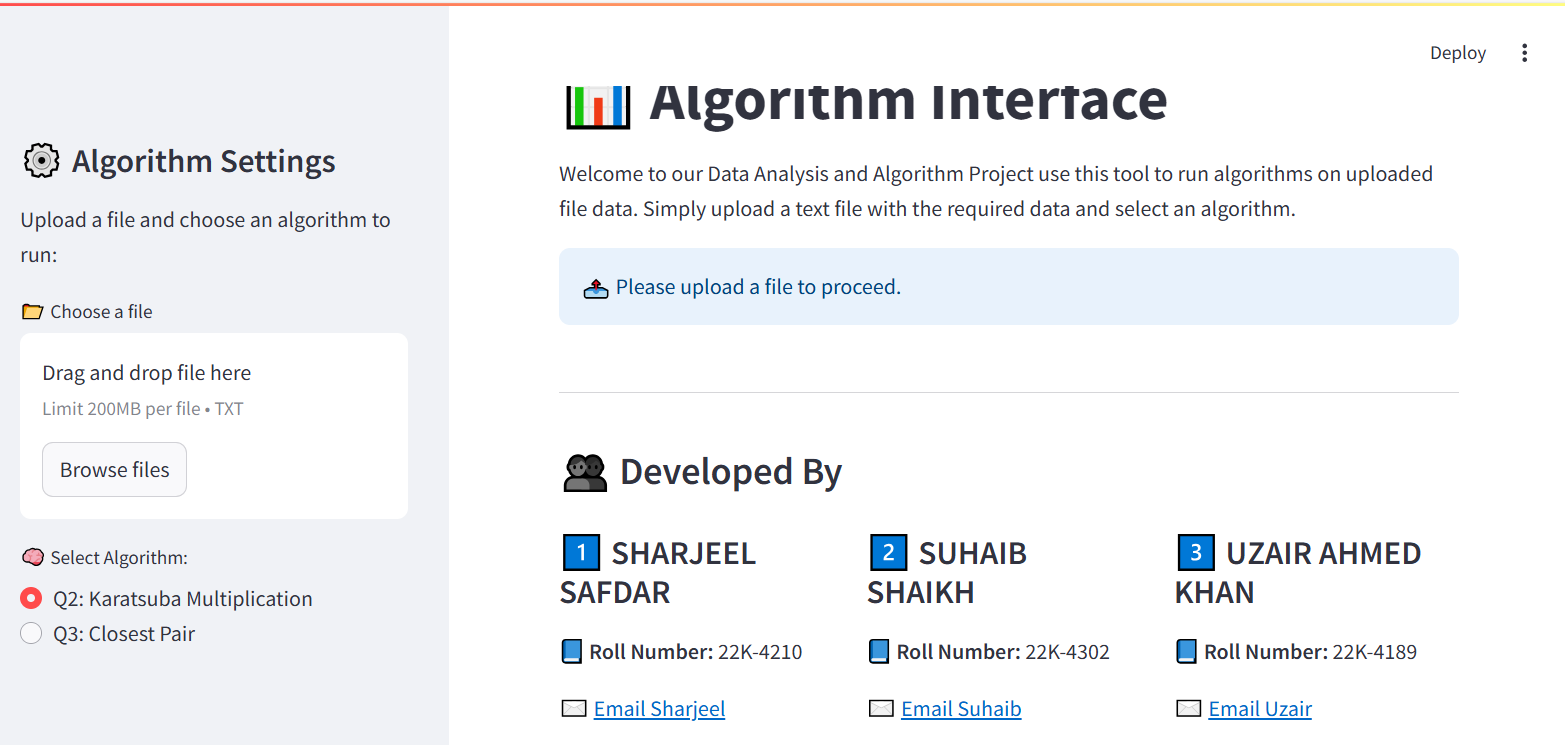
STEP 2 :-



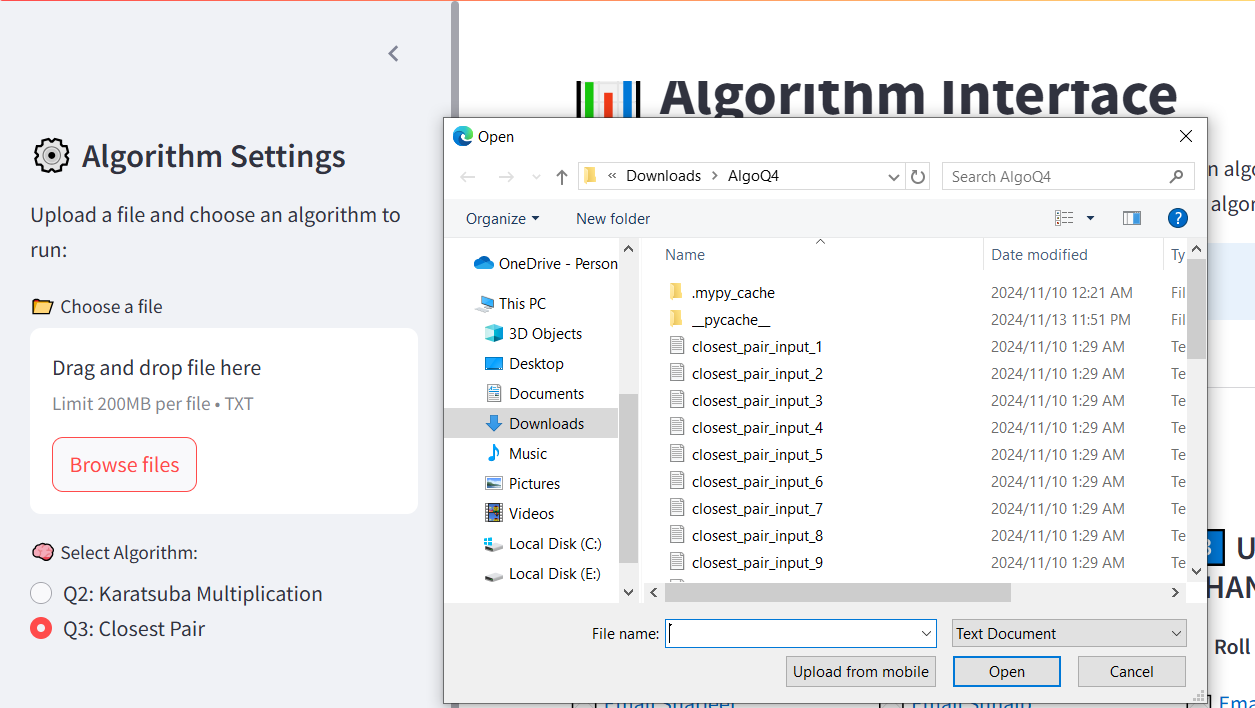
STEP 3 :-



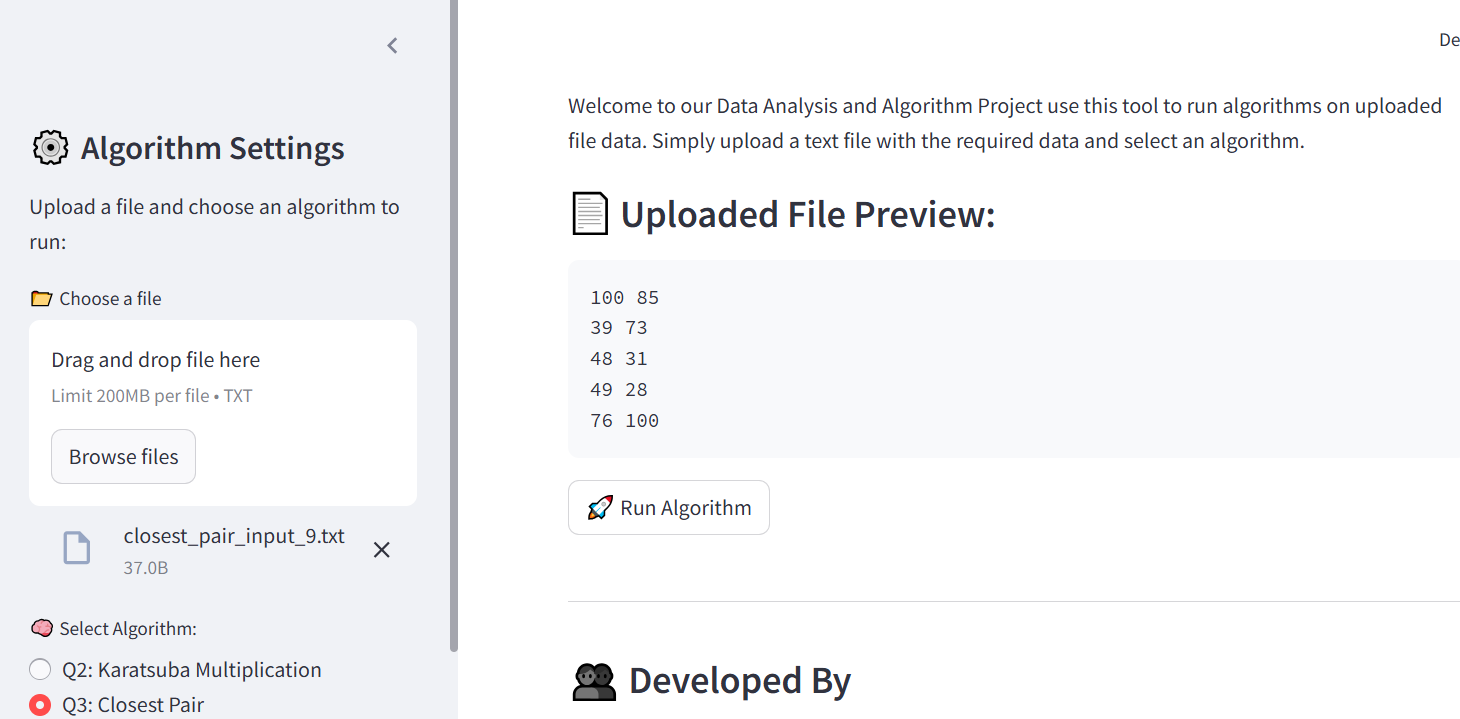
STEP 4 :-



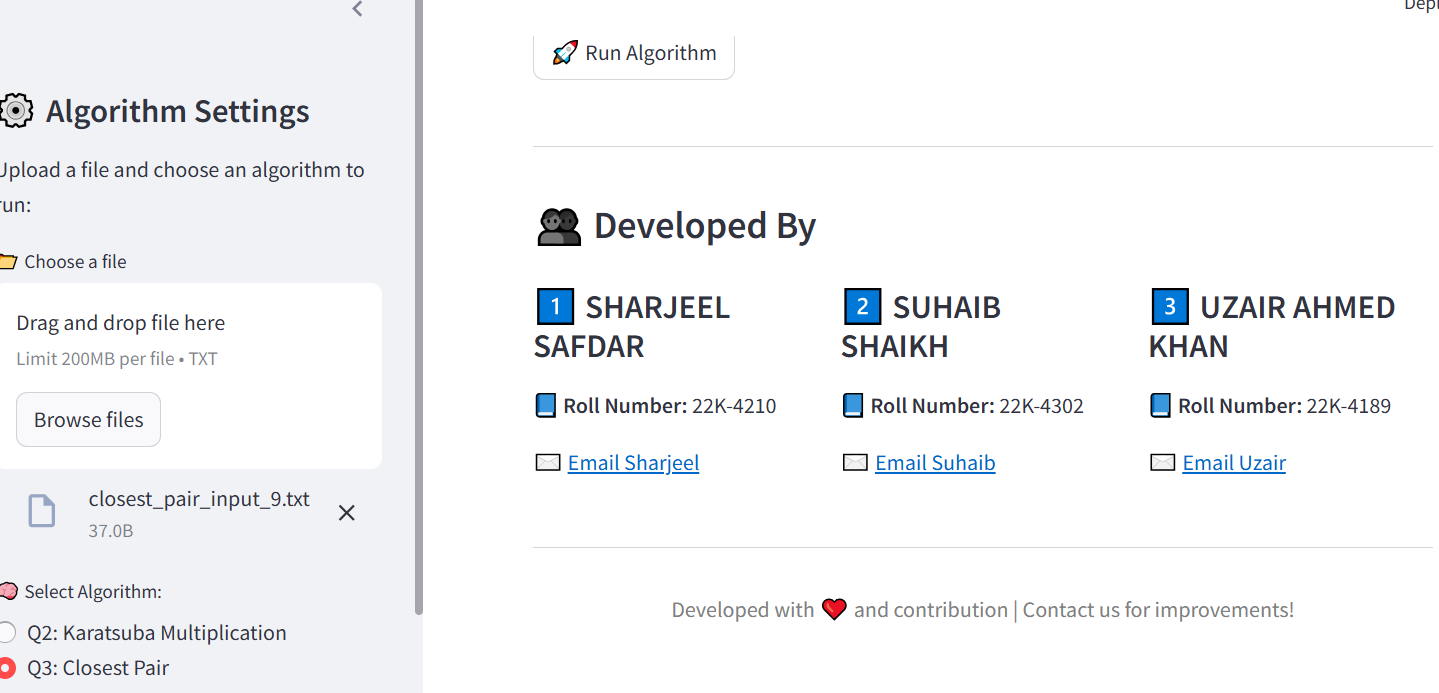
STEP 5 :-



STEP 6 :-



STEP 7 :-



STEP 8 :-

