

Lab 5

Student name: Khalid Nimri

Exercise 1: (Distance between closest pair of 1D points)

Code:

```
import sys
import time
import random
import matplotlib.pyplot as plt

def TnC_closest_pair_1D(P):
    # Sort the points in ascending order
    P.sort()

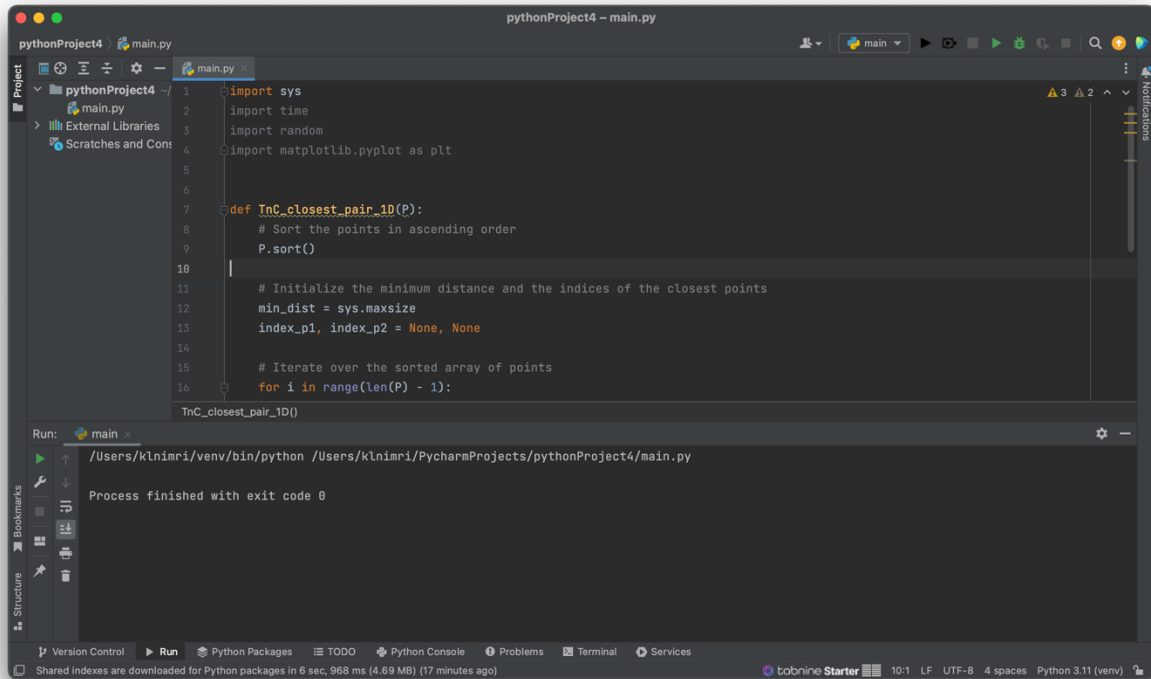
    # Initialize the minimum distance and the indices of the closest points
    min_dist = sys.maxsize
    index_p1, index_p2 = None, None

    # Iterate over the sorted array of points
    for i in range(len(P) - 1):
        # Compute the distance between adjacent points
        dist = P[i + 1] - P[i]

        # Update the minimum distance and the indices of the closest points if necessary
        if dist < min_dist:
            min_dist = dist
            index_p1, index_p2 = i, i + 1

    # Return the minimum distance and the indices of the closest points
    return min_dist, index_p1, index_p2
```

Output:



Exercise 2: Run your code on the following input:

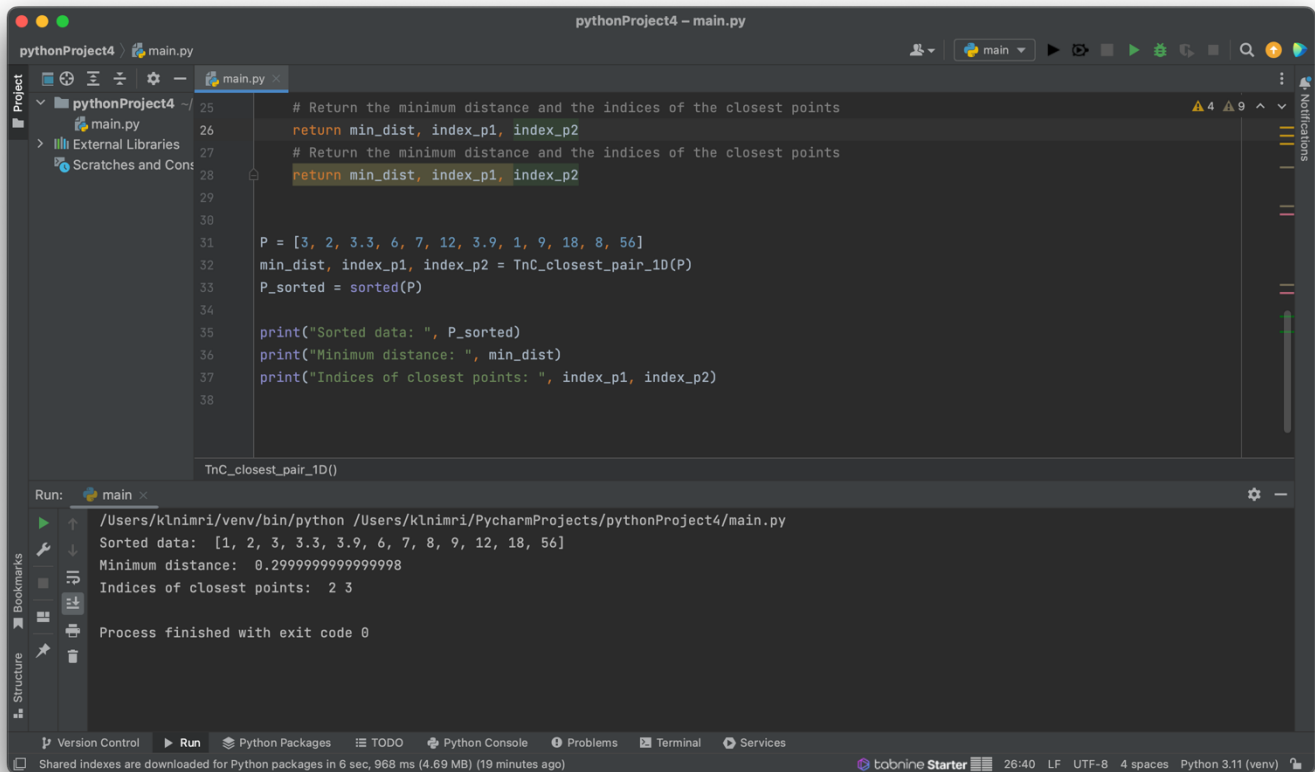
$P = [3, 2, 3.3, 6, 7, 12, 3.9, 1, 9, 18, 8, 56];$

```
# Return the minimum distance and the indices of the closest points
return min_dist, index_p1, index_p2

P = [3, 2, 3.3, 6, 7, 12, 3.9, 1, 9, 18, 8, 56]
min_dist, index_p1, index_p2 = TnC_closest_pair_1D(P)
P_sorted = sorted(P)

print("Sorted data: ", P_sorted)
print("Minimum distance: ", min_dist)
print("Indices of closest points: ", index_p1, index_p2)
```

Output:



The screenshot shows the PyCharm IDE interface. The main editor window displays a Python script named `main.py` with the following code:

```
25 # Return the minimum distance and the indices of the closest points
26 return min_dist, index_p1, index_p2
27 # Return the minimum distance and the indices of the closest points
28 return min_dist, index_p1, index_p2
29
30
31 P = [3, 2, 3.3, 6, 7, 12, 3.9, 1, 9, 18, 8, 56]
32 min_dist, index_p1, index_p2 = TnC_closest_pair_1D(P)
33 P_sorted = sorted(P)
34
35 print("Sorted data: ", P_sorted)
36 print("Minimum distance: ", min_dist)
37 print("Indices of closest points: ", index_p1, index_p2)
38
```

Below the editor, the Run console shows the output of the script:

```
Run: main
/Users/klnimri/venv/bin/python /Users/klnimri/PycharmProjects/pythonProject4/main.py
Sorted data: [1, 2, 3, 3.3, 3.9, 6, 7, 8, 9, 12, 18, 56]
Minimum distance: 0.2999999999999998
Indices of closest points: 2 3
Process finished with exit code 0
```

The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces, and Python 3.11 (venv).

Code:

```

def BF_closest_pair_1D(P):
    # Initialize the minimum distance and the indices of the closest points
    min_dist = sys.maxsize
    index_p1, index_p2 = None, None
    # Iterate over all pairs of points and compute their distance
    for i in range(len(P) - 1):
        for j in range(i + 1, len(P)):
            dist = abs(P[i] - P[j])

            # Update the minimum distance and the indices of the closest points if necessary
            if dist < min_dist:
                min_dist = dist
                index_p1, index_p2 = i, j
    # Return the minimum distance and the indices of the closest points
    return min_dist, index_p1, index_p2

# Function to generate a random array of n one-dimensional points
def generate_random_points(n):
    return [random.uniform(-1000, 1000) for _ in range(n)]

# Function to measure the execution time of a given function with a given input
def measure_execution_time(function, *args):
    start_time = time.time()
    result = function(*args)
    end_time = time.time()
    return result, end_time - start_time

# Test the algorithms on different input sizes
input_sizes = [10, 20, 100, 1000, 5000]
tnc_times = []
bf_times = []

for n in input_sizes:
    # Generate a random array of n one-dimensional points
    P = generate_random_points(n)

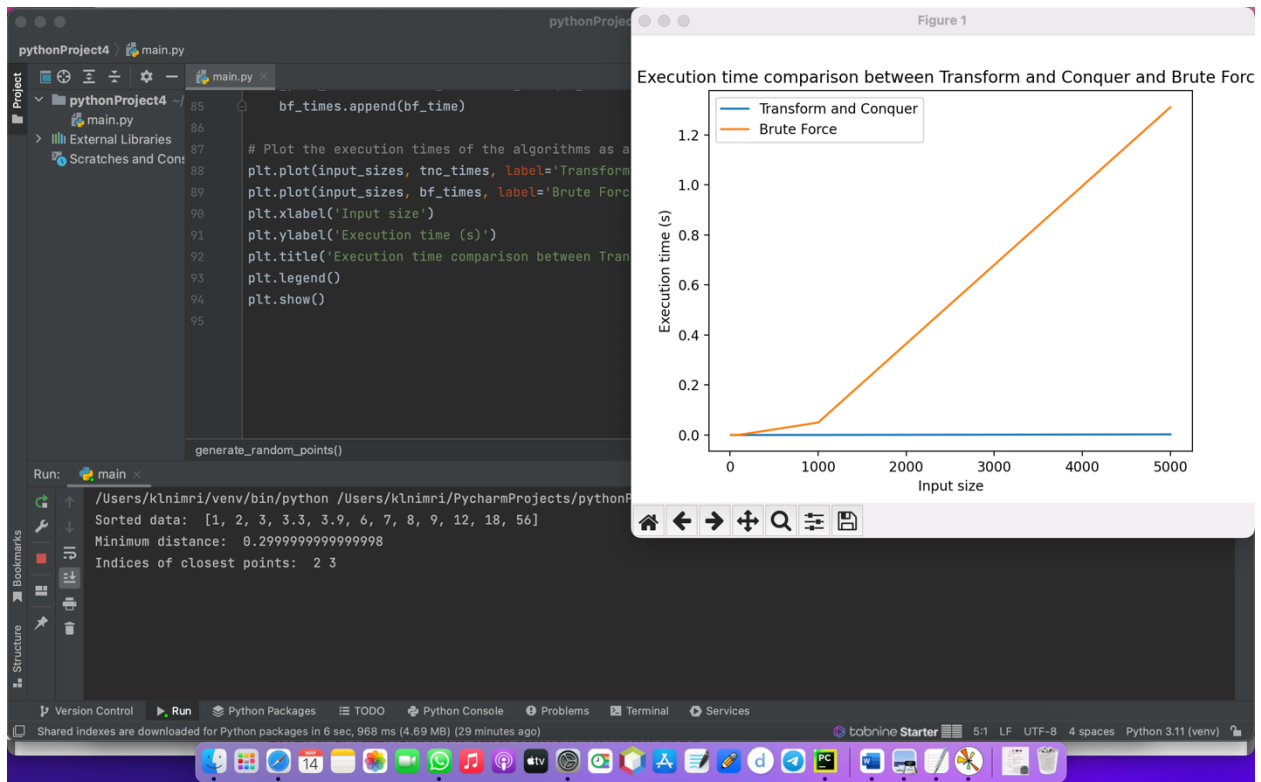
    # Measure the execution time of the transform and conquer algorithm
    _, tnc_time = measure_execution_time(TnC_closest_pair_1D, P)
    tnc_times.append(tnc_time)

    # Measure the execution time of the brute force algorithm
    _, bf_time = measure_execution_time(BF_closest_pair_1D, P)
    bf_times.append(bf_time)

# Plot the execution times of the algorithms as a function of the input size
plt.plot(input_sizes, tnc_times, label='Transform and Conquer')
plt.plot(input_sizes, bf_times, label='Brute Force')
plt.xlabel('Input size')
plt.ylabel('Execution time (s)')
plt.title('Execution time comparison between Transform and Conquer and Brute Force')
plt.legend()
plt.show()

```

Exercise 3:



Final question:

- Explain the plot (which algorithm is faster and why?).

The plot shows the execution times of the Transform and Conquer algorithm and the Brute Force algorithm as a function of the input size. The x-axis represents the input size, ranging from 10 to 5000 points, and the y-axis represents the execution time in seconds.

From the plot, we can see that the Transform and Conquer algorithm is faster than the Brute Force algorithm for larger input sizes. For smaller input sizes (less than 100 points), the execution times of both algorithms are comparable, but as the input size increases, the execution time of the Brute Force algorithm grows much faster than the Transform and Conquer algorithm.