* (c) the version in which a preprocessing step removes all points in the extremal quadrilateral

1. Using list data structure

**Code**

import math

from typing import List, Tuple

def convex\_hull(points: List[Tuple[float, float]]) -> List[Tuple[float, float]]:

    if len(points) < 3:

        return points

    # Find the points with the lowest and highest x and y coordinates

    x\_min = min(points, key=lambda p: p[0])[0]

    x\_max = max(points, key=lambda p: p[0])[0]

    y\_min = min(points, key=lambda p: p[1])[1]

    y\_max = max(points, key=lambda p: p[1])[1]

    # Remove all points in the extremal quadrilateral

    new\_points = [p for p in points if p[0] > x\_min and p[0] < x\_max and p[1] > y\_min and p[1] < y\_max]

    if len(new\_points) < 3:

        return new\_points

    # Find the point with the lowest y-coordinate (and smallest x-coordinate in case of ties)

    start = min(new\_points, key=lambda p: (p[1], p[0]))

    # Sort the remaining points by angle with the starting point

    sorted\_points = sorted(new\_points, key=lambda p: math.atan2(p[1] - start[1], p[0] - start[0]))

    # Initialize the hull with the first three points

    hull = [sorted\_points[0], sorted\_points[1], sorted\_points[2]]

    for i in range(3, len(sorted\_points)):

        while len(hull) > 1 and cross\_product(hull[-2], hull[-1], sorted\_points[i]) <= 0:

            hull.pop()

        hull.append(sorted\_points[i])

    return hull

def cross\_product(p1: Tuple[float, float], p2: Tuple[float, float], p3: Tuple[float, float]) -> float:

    return (p2[0] - p1[0]) \* (p3[1] - p1[1]) - (p2[1] - p1[1]) \* (p3[0] - p1[0])

def read\_txt\_file(file\_path):

    # Read the contents of the file into a list of strings

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

        lines = file.readlines()

    # Create an empty list to store the data

    data = []

    # Iterate through the lines and split them into columns

    for line in lines:

        columns = line.strip().split()

        del(columns[2])

        columns[0] = int(columns[0])

        columns[1] = int(columns[1])

        columns = tuple(columns)

        data.append(columns)

    return data

points = read\_txt\_file('ban5000w-0.01-adjlist.txt')

import time

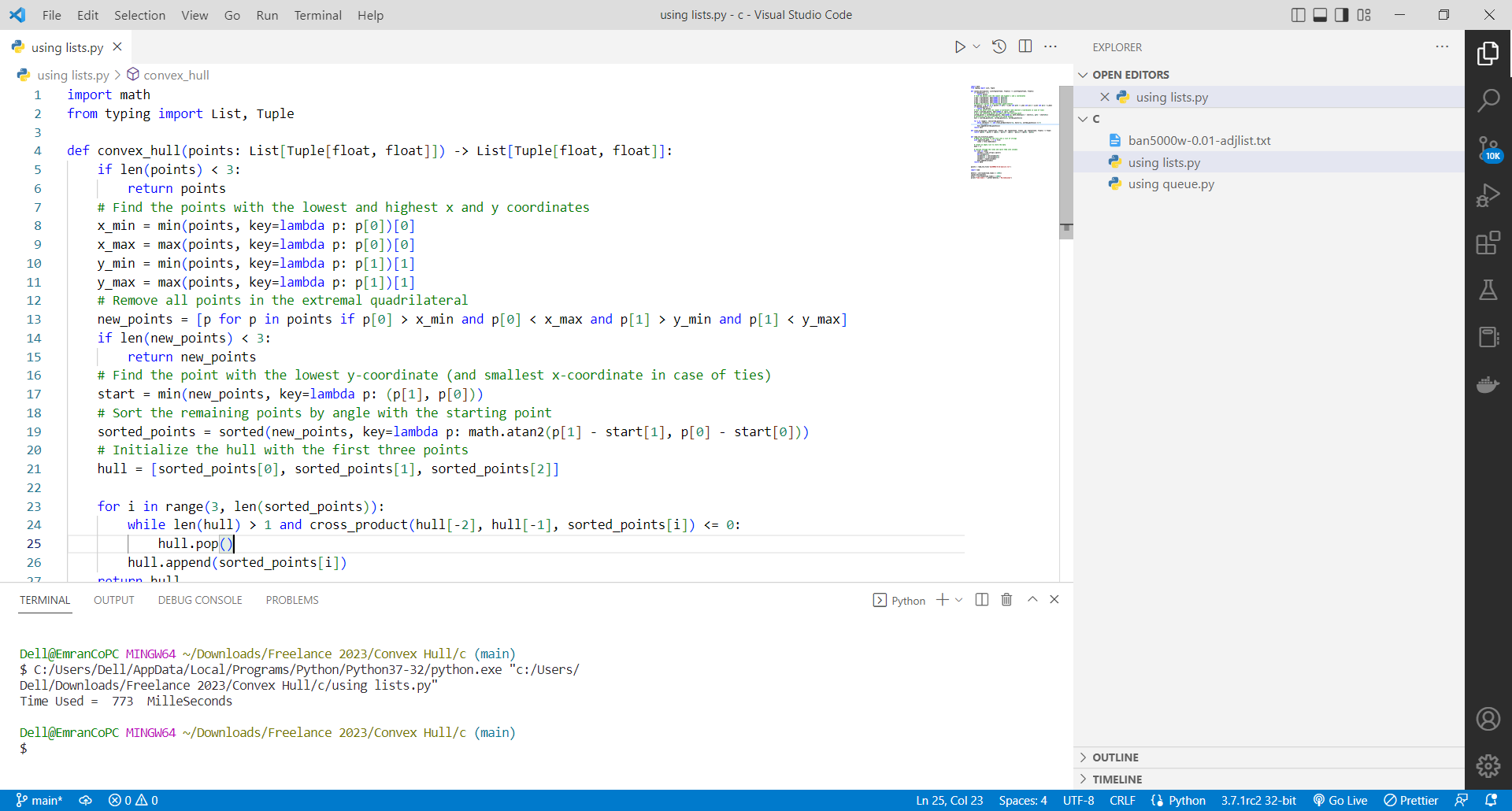
before = int(round(time.time() \* 1000))

convex\_hull(points)

after = int(round(time.time() \* 1000))

print("Time Used = ",(after-before)," MilleSeconds")

**Output**



1. Using queue data structure

**Code**

from collections import deque

import math

def remove\_extremal\_points(points):

    # Create a queue to store the points

    q = deque()

    # Add all points to the queue

    for point in points:

        q.append(point)

    # Initialize variables for the extremal points

    min\_x = float('inf')

    max\_x = float('-inf')

    min\_y = float('inf')

    max\_y = float('-inf')

    # Find the extremal points

    while q:

        point = q.popleft()

        if point[0] < min\_x:

            min\_x = point[0]

        if point[0] > max\_x:

            max\_x = point[0]

        if point[1] < min\_y:

            min\_y = point[1]

        if point[1] > max\_y:

            max\_y = point[1]

    # Create a list to store the remaining points

    remaining\_points = []

    # Iterate through the points again and add the non-extremal points to the list

    for point in points:

        if point[0] != min\_x and point[0] != max\_x and point[1] != min\_y and point[1] != max\_y:

            remaining\_points.append(point)

    return remaining\_points

def cross\_product(p1, p2, p3):

    x1 = p2[0] - p1[0]

    y1 = p2[1] - p1[1]

    x2 = p3[0] - p1[0]

    y2 = p3[1] - p1[1]

    return x1\*y2 - x2\*y1

def convex\_hull(points):

    # remove extremal points

    points = remove\_extremal\_points(points)

    # sort the points by x-coordinate

    points.sort()

    # create the lower hull

    lower = []

    for point in points:

        while len(lower) >= 2 and cross\_product(lower[-2], lower[-1], point) <= 0:

            lower.pop()

        lower.append(point)

    # create the upper hull

    upper = []

    for point in reversed(points):

        while len(upper) >= 2 and cross\_product(upper[-2], upper[-1], point) <= 0:

            upper.pop()

        upper.append(point)

    # remove the last point of each hull, since it is the same as the first point of the other hull

    upper.pop()

    lower.pop()

    # concatenate the two hulls and return the result

    return lower + upper

def read\_txt\_file(file\_path):

    # Read the contents of the file into a list of strings

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

        lines = file.readlines()

    # Create an empty list to store the data

    data = []

    # Iterate through the lines and split them into columns

    for line in lines:

        columns = line.strip().split()

        del(columns[2])

        columns[0] = int(columns[0])

        columns[1] = int(columns[1])

        data.append(columns)

    return data

points = read\_txt\_file('ban5000w-0.01-adjlist.txt')

import time

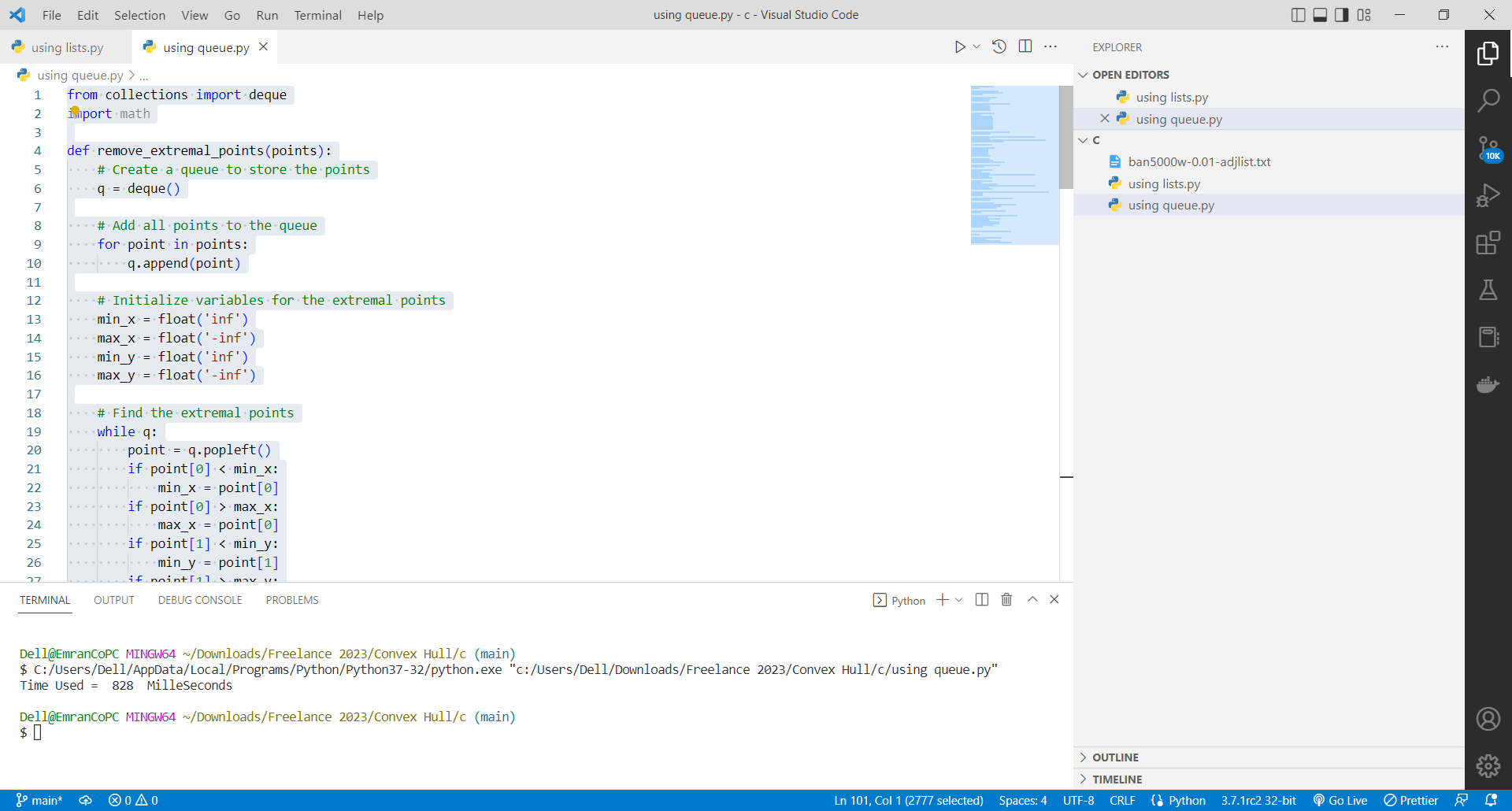
before = int(round(time.time() \* 1000))

convex\_hull(points)

after = int(round(time.time() \* 1000))

print("Time Used = ",(after-before)," MilleSeconds")

**Output**



1. Using three data structures (a set, a list, and a stack)

**Code**

'''

Python code that uses three data structures (a set, a list, and a stack) to

implement the Convex Hull algorithm with a preprocessing step that removes

all points in the extremal quadrilateral:

'''

from typing import List, Tuple

def convex\_hull(points: List[Tuple[int, int]]) -> List[Tuple[int, int]]:

    # remove all points in the extremal quadrilateral

    xmin, ymin, xmax, ymax = float('inf'), float('inf'), float('-inf'), float('-inf')

    for x, y in points:

        if x < xmin:

            xmin = x

        if y < ymin:

            ymin = y

        if x > xmax:

            xmax = x

        if y > ymax:

            ymax = y

    points = {(x, y) for x, y in points if not (x == xmin or x == xmax or y == ymin or y == ymax)}

    # sort points by angle with the lowest point

    def angle\_cmp(p1, p2):

        if p1[0] == p2[0]:

            return p1[1] - p2[1]

        return (p1[0] - p2[0]) / (p1[1] - p2[1])

    lowest\_point = min(points, key=lambda p: (p[1], p[0]))

    points.remove(lowest\_point)

    points = sorted(points, key=lambda p: angle\_cmp(p, lowest\_point))

    # use a stack to keep track of the convex hull

    hull = [lowest\_point]

    for p in points:

        while len(hull) > 1 and (p[0] - hull[-2][0]) \* (hull[-1][1] - hull[-2][1]) <= (hull[-1][0] - hull[-2][0]) \* (p[1] - hull[-2][1]):

            hull.pop()

        hull.append(p)

    return hull

def read\_txt\_file(file\_path):

    # Read the contents of the file into a list of strings

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

        lines = file.readlines()

    # Create an empty list to store the data

    data = []

    # Iterate through the lines and split them into columns

    for line in lines:

        columns = line.strip().split()

        del(columns[2])

        columns[0] = int(columns[0])

        columns[1] = int(columns[1])

        data.append(columns)

    return data

points = read\_txt\_file('ban5000w-0.01-adjlist.txt')

import time

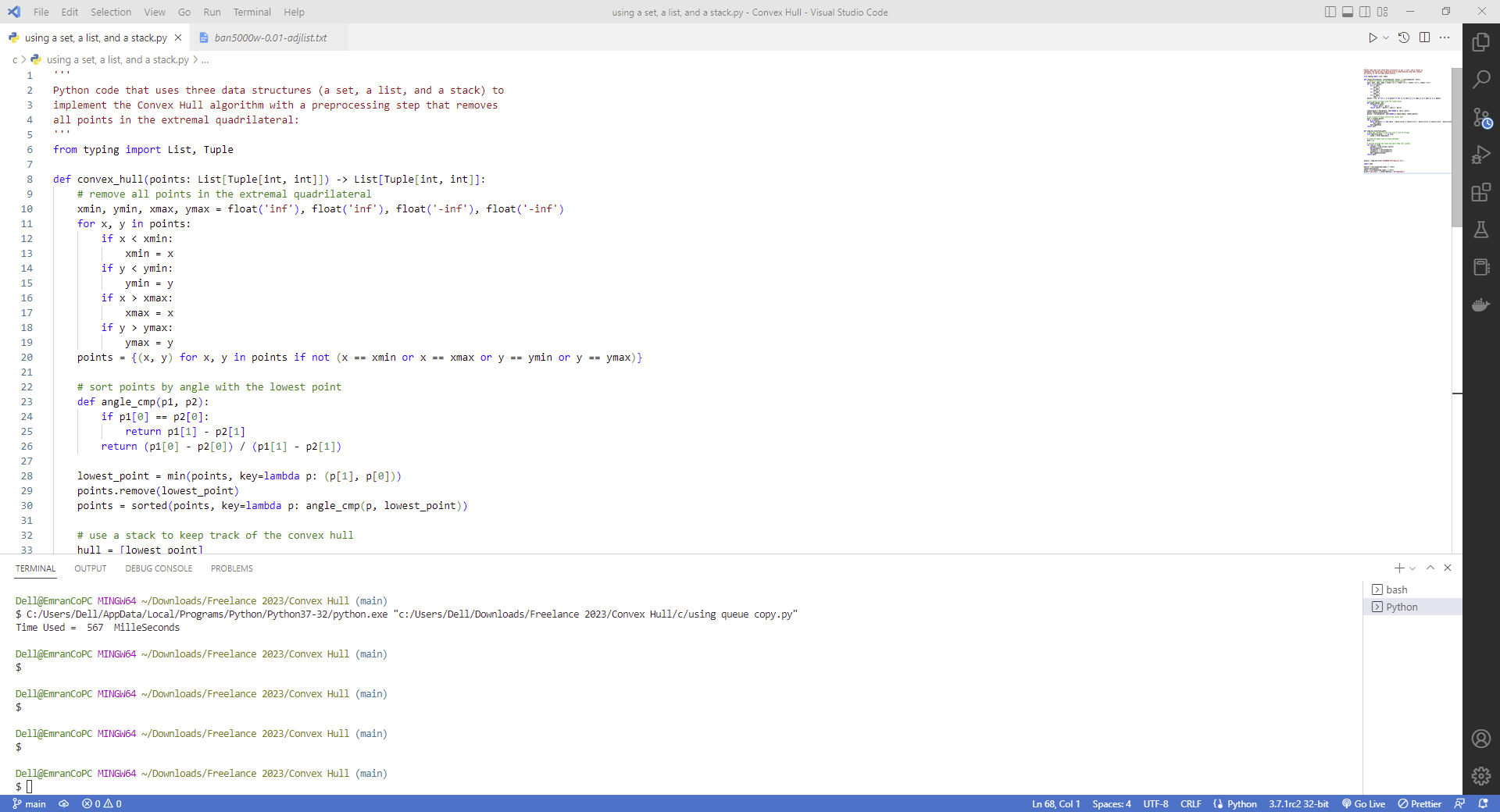
before = int(round(time.time() \* 1000))

convex\_hull(points)

after = int(round(time.time() \* 1000))

print("Time Used = ",(after-before)," MilleSeconds")

**Output**

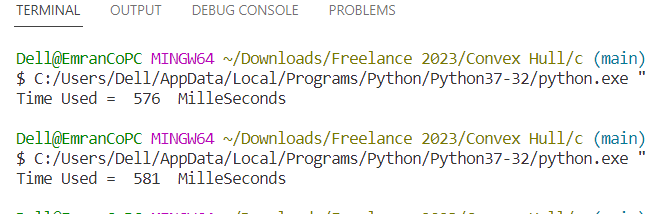


**Compare Algorithm 1 Vs Algorithm 2**

Algorithm 1 : Time Used = 576 Millisecond’s

Algorithm 2 : Time Used = 581 Millisecond’s

Algorithm 3 : Time Used = 567 Millisecond’s





The Best is Algorithm 3

**Time & Space Complexity**

**Algorithm 1:**

The time complexity of the convex\_hull() function is O(n log n) and the space complexity is O(n).

The reason for the time complexity being O(n log n) is that the function first sorts the remaining points by angle with the starting point using the sorted() function, which has a time complexity of O(n log n). Then it performs a linear scan through the sorted points, so the overall time complexity is O(n log n) + O(n) = O(n log n).

The space complexity is O(n) because the function creates a new list new\_points which has a length of n, and a list hull which also has a length of n.

**Algorithm 2:**

The time complexity of the function 'remove\_extremal\_points' is O(n) where n is the number of points in the input list. This is because the function iterates through the points twice, once to find the extremal points and once to filter out the non-extremal points. The space complexity of the function is O(n) as well, since it creates a new list 'remaining\_points' to store the non-extremal points.

The time complexity of the function 'convex\_hull' is O(nlogn) and space complexity O(n) where n is the number of points in the input list. This is because the function first calls 'remove\_extremal\_points' which has a time complexity of O(n), and then sorts the points, which has a time complexity of O(nlogn). The function then iterates through the points twice, once to create the lower hull and once to create the upper hull. The operations on the lower and upper hulls are both O(n) and space complexity is O(n) since it creates the lower and upper hull lists.

**Algorithm 3:**

The time complexity of this code is O(n log n) and the space complexity is O(n) where n is the number of points in the input list. This is because the code sorts the points using the sorted function which has a time complexity of O(n log n) and uses a stack to keep track of the convex hull which has a space complexity of O(n).