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

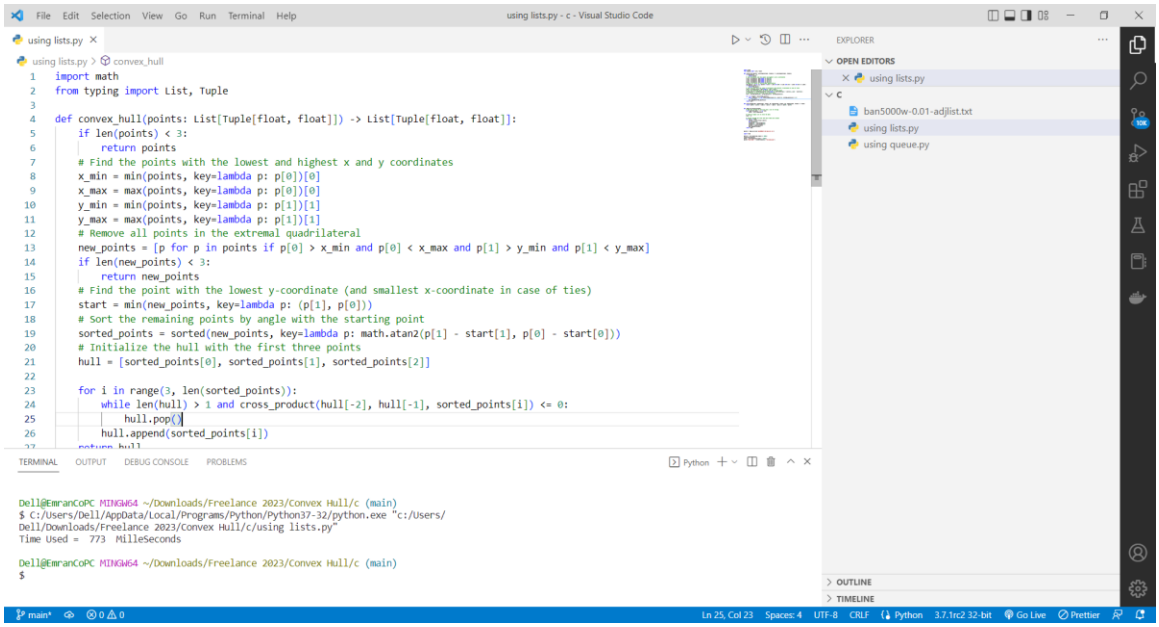
1- Using list data structure

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



2- Using queue data structure

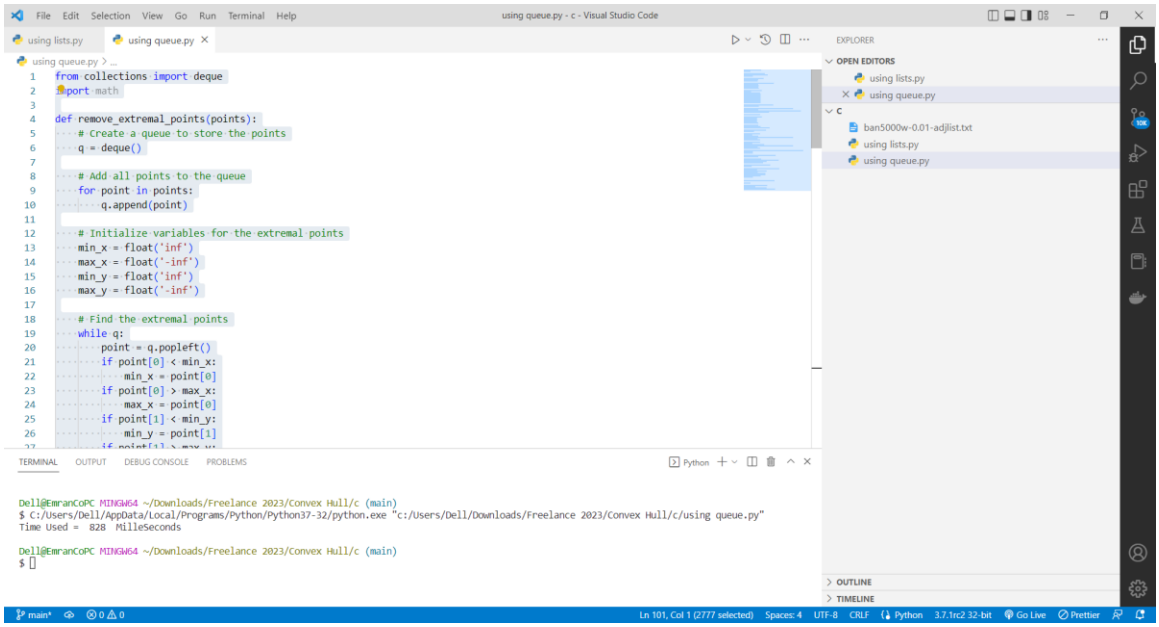
Code

```
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



3- Using three data structures (a set, a list, and a stack)

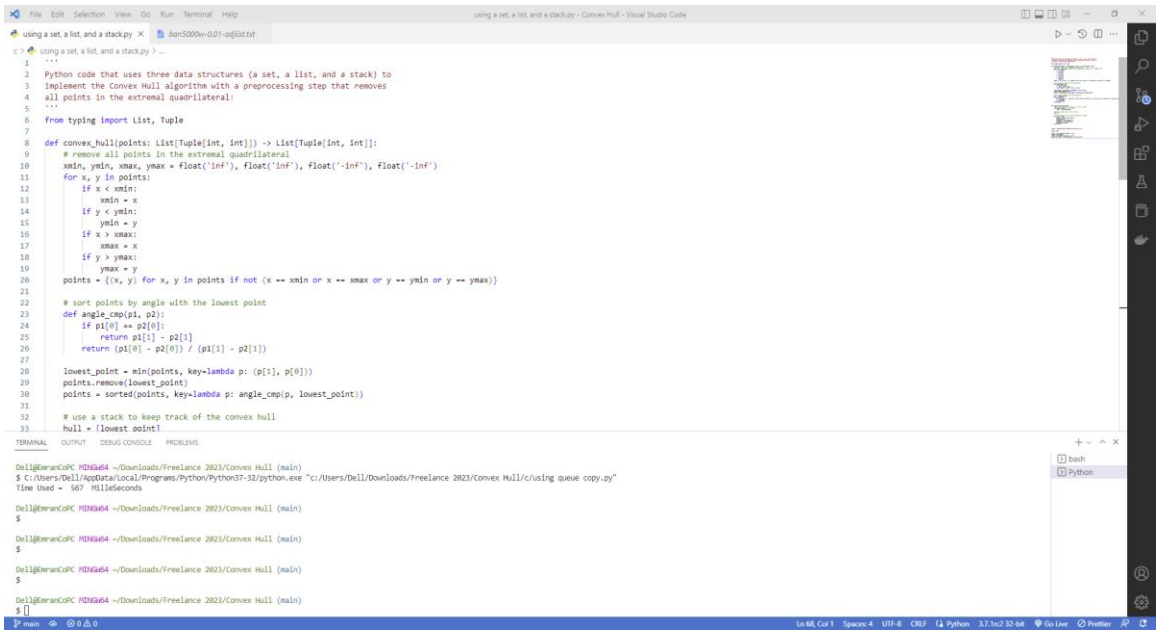
Code

```
'''
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')
```

```python
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).