Python implementation of the Jarvis march algorithm for finding the convex hull of a nonconvex polygon:

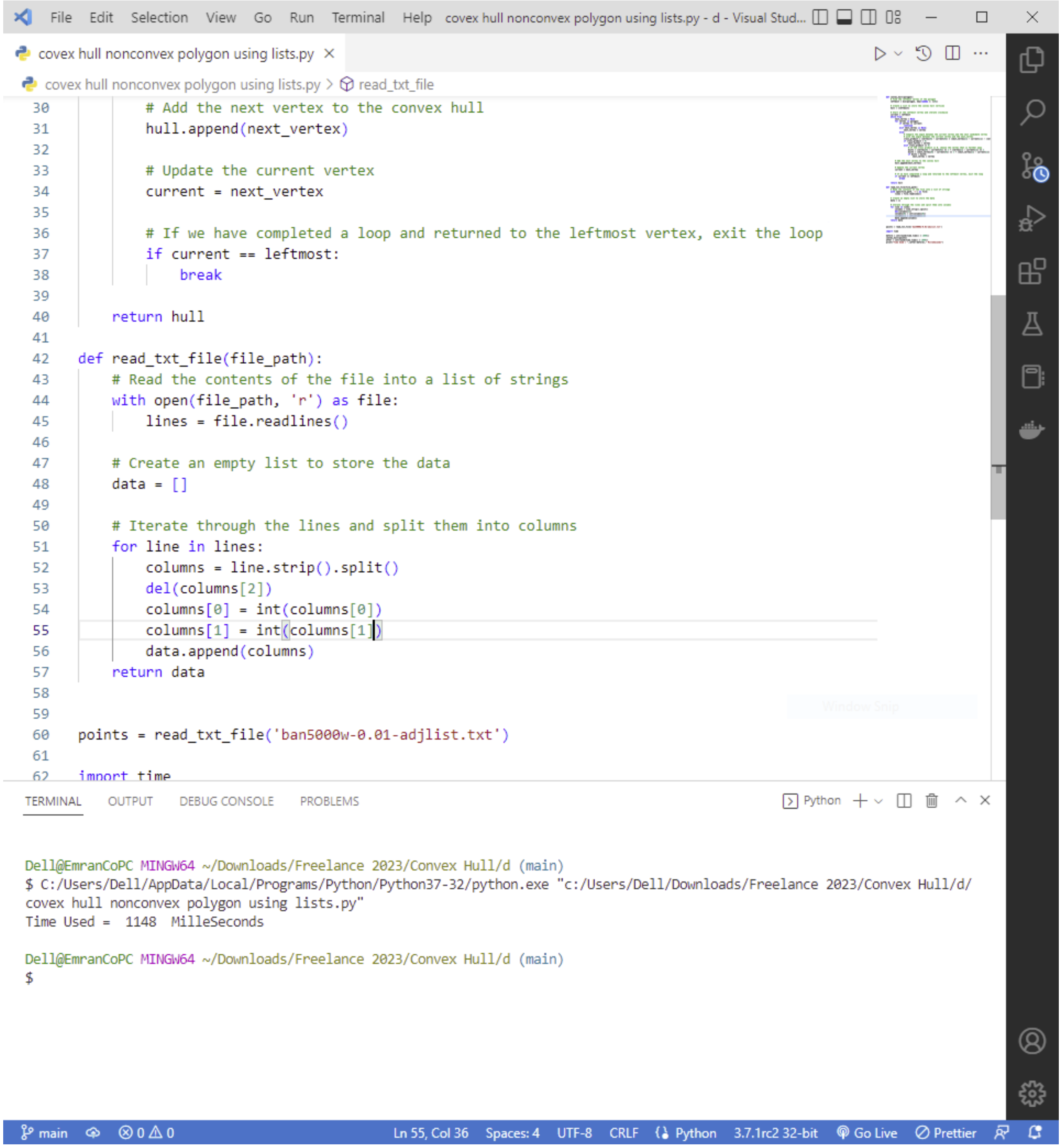
- (d) Python implementation of the Jarvis march algorithm for finding the convex hull of a nonconvex polygon

1- Using list data structure

Code

```
def convex_hull(polygon):
    # Find the leftmost vertex of the polygon
    leftmost = min(polygon, key=lambda v: v[0])

    # Create a list to store the convex hull vertices
    hull = [leftmost]

    # Start at the leftmost vertex and iterate clockwise
    current = leftmost
    while True:
        next_vertex = None
        for vertex in polygon:
            if vertex == current:
                continue
            elif next_vertex is None:
                next_vertex = vertex
            else:
                # Compare the angle between the current vertex and the next candidate vertex
                # with the angle between the current vertex and the next_vertex
                cross_product = (vertex[0] - current[0]) * (next_vertex[1] - current[1]) - (vertex[1] - current[1]) * (next_vertex[0] - current[0])
                if cross_product > 0:
                    next_vertex = vertex
                elif cross_product == 0:
                    # If the cross product is 0, choose the vertex that is farther away
                    dist1 = (vertex[0] - current[0]) ** 2 + (vertex[1] - current[1]) ** 2
                    dist2 = (next_vertex[0] - current[0]) ** 2 + (next_vertex[1] - current[1]) ** 2
                    if dist1 > dist2:
                        next_vertex = vertex

        # Add the next vertex to the convex hull
        hull.append(next_vertex)

        # Update the current vertex
        current = next_vertex

        # If we have completed a loop and returned to the leftmost vertex,
exit the loop
        if current == leftmost:
            break

    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


File
Edit
Selection
View
Go
Run
Terminal
Help
covex hull nonconvex polygon using lists.py - d - Visual Stud...
covex hull nonconvex polygon using lists.py
covex hull nonconvex polygon using lists.py > read_txt_file
TERMINAL
OUTPUT
DEBUG CONSOLE
PROBLEMS
Python
main
0 0
Ln 55, Col 36
Spaces: 4
UTF-8
CRLF
Python
3.7.1rc2 32-bit
Go Live
Prettier

```python
        # Add the next vertex to the convex hull
        hull.append(next_vertex)

        # Update the current vertex
        current = next_vertex

        # If we have completed a loop and returned to the leftmost vertex, exit the loop
        if current == leftmost:
            break

    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
```

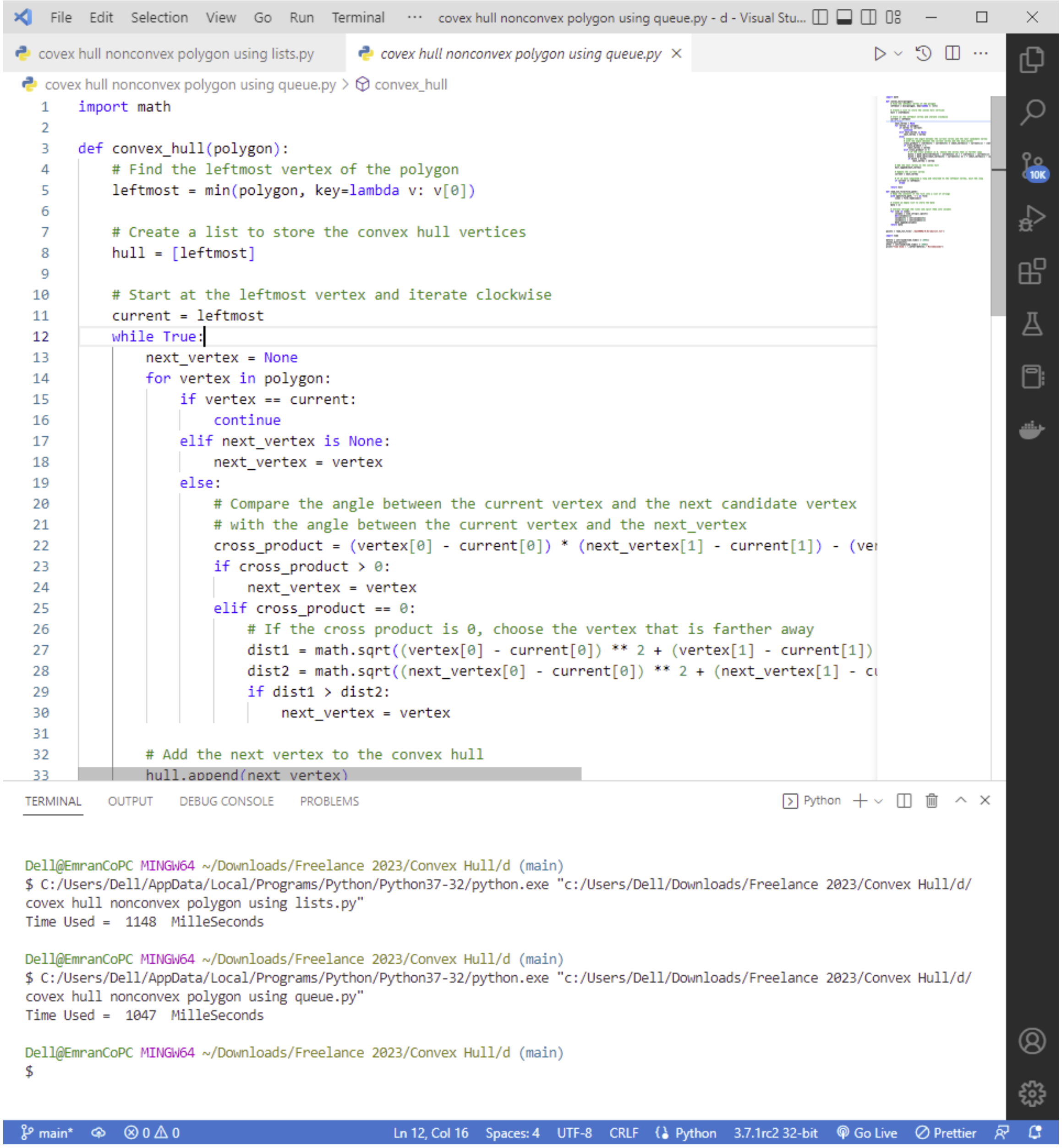
```
Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/
covex hull nonconvex polygon using lists.py"
Time Used =  1148  MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$
```

2- Using queue data structure

Code

```python
import math

def convex_hull(polygon):
    # Find the leftmost vertex of the polygon
    leftmost = min(polygon, key=lambda v: v[0])

    # Create a list to store the convex hull vertices
    hull = [leftmost]

    # Start at the leftmost vertex and iterate clockwise
    current = leftmost
    while True:
        next_vertex = None
        for vertex in polygon:
            if vertex == current:
                continue
            elif next_vertex is None:
                next_vertex = vertex
            else:
                # Compare the angle between the current vertex and the
next candidate vertex
                # with the angle between the current vertex and the
next_vertex
                cross_product = (vertex[0] - current[0]) * (next_vertex[1]
- current[1]) - (vertex[1] - current[1]) * (next_vertex[0] - current[0])
                if cross_product > 0:
                    next_vertex = vertex
                elif cross_product == 0:
                    # If the cross product is 0, choose the vertex that is
farther away
                    dist1 = math.sqrt((vertex[0] - current[0]) ** 2 +
(vertex[1] - current[1]) ** 2)
                    dist2 = math.sqrt((next_vertex[0] - current[0]) ** 2 +
(next_vertex[1] - current[1]) ** 2)
                    if dist1 > dist2:
                        next_vertex = vertex

        # Add the next vertex to the convex hull
        hull.append(next_vertex)

        # Update the current vertex
        current = next_vertex

        # If we have completed a loop and returned to the leftmost vertex,
exit the loop
        if current == leftmost:
            break
    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


File
Edit
Selection
View
Go
Run
Terminal
covex hull nonconvex polygon using queue.py - d - Visual Stu...
covex hull nonconvex polygon using lists.py
covex hull nonconvex polygon using queue.py
covex hull nonconvex polygon using queue.py > convex_hull
TERMINAL
OUTPUT
DEBUG CONSOLE
PROBLEMS
Python
main*
0 0
Ln 12, Col 16
Spaces: 4
UTF-8
CRLF
Python
3.7.1rc2 32-bit
Go Live
Prettier
10K

```python
import math

def convex_hull(polygon):
    # Find the leftmost vertex of the polygon
    leftmost = min(polygon, key=lambda v: v[0])

    # Create a list to store the convex hull vertices
    hull = [leftmost]

    # Start at the leftmost vertex and iterate clockwise
    current = leftmost
    while True:
        next_vertex = None
        for vertex in polygon:
            if vertex == current:
                continue
            elif next_vertex is None:
                next_vertex = vertex
            else:
                # Compare the angle between the current vertex and the next candidate vertex
                # with the angle between the current vertex and the next_vertex
                cross_product = (vertex[0] - current[0]) * (next_vertex[1] - current[1]) - (ver
                if cross_product > 0:
                    next_vertex = vertex
                elif cross_product == 0:
                    # If the cross product is 0, choose the vertex that is farther away
                    dist1 = math.sqrt((vertex[0] - current[0]) ** 2 + (vertex[1] - current[1])
                    dist2 = math.sqrt((next_vertex[0] - current[0]) ** 2 + (next_vertex[1] - cu
                    if dist1 > dist2:
                        next_vertex = vertex

        # Add the next vertex to the convex hull
        hull.append(next_vertex)
```

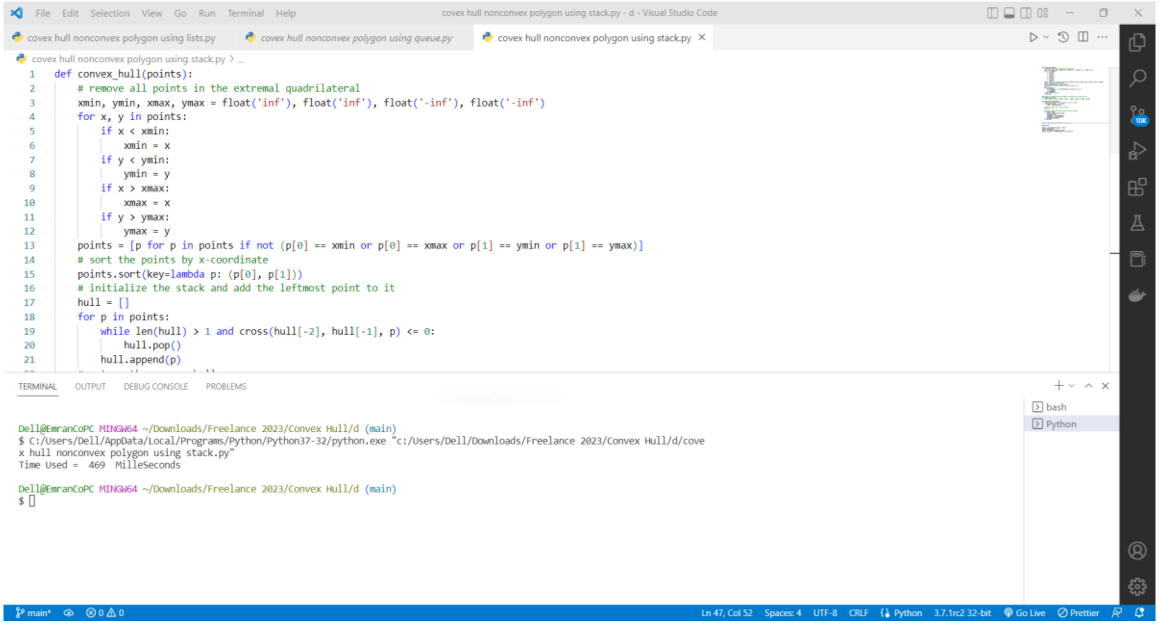
```
Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/
covex hull nonconvex polygon using lists.py"
Time Used =  1148  MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/
covex hull nonconvex polygon using queue.py"
Time Used =  1047  MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$
```

3- Using stack data structure

Code

```
def convex_hull(points):
    # 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 = [p for p in points if not (p[0] == xmin or p[0] == xmax or 
p[1] == ymin or p[1] == ymax)]
    # sort the points by x-coordinate
    points.sort(key=lambda p: (p[0], p[1]))
    # initialize the stack and add the leftmost point to it
    hull = []
    for p in points:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    # return the convex hull
    return hull

# function for computing the cross product of vectors (p1, p2) and (p1, 
p3)
def cross(p1, p2, p3):
    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])
        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



File
Edit
Selection
View
Go
Run
Terminal
Help
covex hull nonconvex polygon using stack.py - d - Visual Studio Code
covex hull nonconvex polygon using lists.py
covex hull nonconvex polygon using queue.py
covex hull nonconvex polygon using stack.py
covex hull nonconvex polygon using stack.py > ...
TERMINAL
OUTPUT
DEBUG CONSOLE
PROBLEMS
bash
Python
main*
0 0
Ln 47, Col 52
Spaces: 4
UTF-8
CRLF
Python
3.7.1rc2 32-bit
Go Live
Prettier

```
def convex_hull(points):
    # 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 = [p for p in points if not (p[0] == xmin or p[0] == xmax or p[1] == ymin or p[1] == ymax)]
    # sort the points by x-coordinate
    points.sort(key=lambda p: (p[0], p[1]))
    # initialize the stack and add the leftmost point to it
    hull = []
    for p in points:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
```

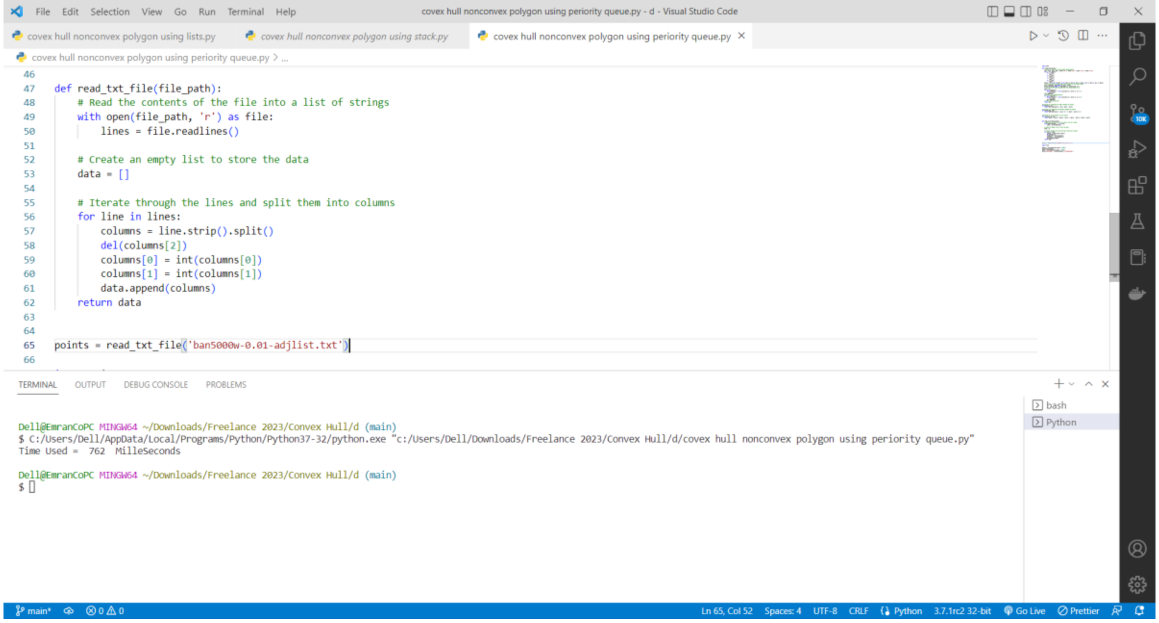
```
Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/cove
x hull nonconvex polygon using stack.py"
Time Used =  469  MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ 
```

4- Using priority queue data structure

Code

```
import math

def convex_hull(points):
    # 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 = [p for p in points if not (p[0] == xmin or p[0] == xmax or
p[1] == ymin or p[1] == ymax)]
    # sort the points by polar angle with respect to the lowest point
    p0 = min(points, key=lambda p: (p[1], p[0]))
    points.sort(key=lambda p: (angle(p0, p), distance(p0, p)))
    # initialize the stack and add the first three points to it
    hull = []
    for p in points[:3]:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
    hull.append(p)
    # process the remaining points
    for p in points[3:]:
        while len(hull) > 1 and cross(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    # return the convex hull
    return hull

# function for computing the angle between two points
def angle(p1, p2):
    return math.atan2(p2[1] - p1[1], p2[0] - p1[0])

# function for computing the distance between two points
def distance(p1, p2):
    return math.sqrt((p2[1] - p1[1]) ** 2 + (p2[0] - p1[0]) ** 2)


# function for cross product of two vectors
def cross(p1, p2, p3):
    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])
        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

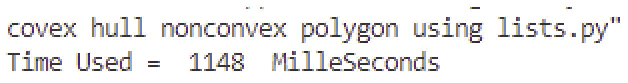

File Edit Selection View Go Run Terminal Help
covex hull nonconvex polygon using periority queue.py - d - Visual Studio Code
covex hull nonconvex polygon using lists.py
covex hull nonconvex polygon using stack.py
covex hull nonconvex polygon using periority queue.py
covex hull nonconvex polygon using periority queue.py > ...
TERMINAL OUTPUT DEBUG CONSOLE PROBLEMS
bash
Python
main* ⊗ 0 ⚠ 0
Ln 65, Col 52 Spaces: 4 UTF-8 CRLF Python 3.7.1rc2 32-bit Go Live Prettier

```python

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')

```

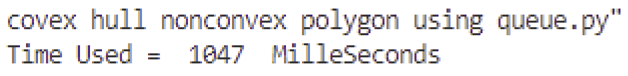
```
Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ C:/Users/Dell/AppData/Local/Programs/Python/Python37-32/python.exe "c:/Users/Dell/Downloads/Freelance 2023/Convex Hull/d/covex hull nonconvex polygon using periority queue.py"
Time Used =  762  MilleSeconds

Dell@EmranCoPC MINGW64 ~/Downloads/Freelance 2023/Convex Hull/d (main)
$ 
```

Compare Algorithm 1 Vs Algorithm 2

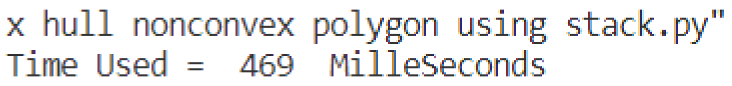
Algorithm 1 : Time Used = 1148 Millisecond's
Algorithm 2 : Time Used = 1047 Millisecond's
Algorithm 3 : Time Used = 469 Millisecond's
Algorithm 4 : Time Used = 762 Millisecond's



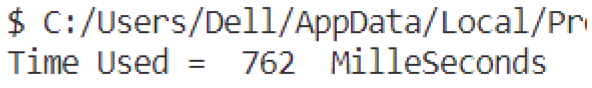
```
covex hull nonconvex polygon using lists.py"
Time Used =  1148  MilleSeconds
```



```
covex hull nonconvex polygon using queue.py"
Time Used =  1047  MilleSeconds
```



```
x hull nonconvex polygon using stack.py"
Time Used =  469  MilleSeconds
```



```
$ C:/Users/Dell/AppData/Local/Pr
Time Used =  762  MilleSeconds
```

The Best is Algorithm 3