Wieloboki Voronoi

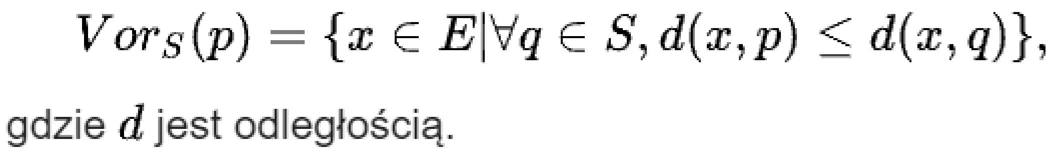
Algorytmy geometryczne 2023/24

Król Mateusz

Definicja

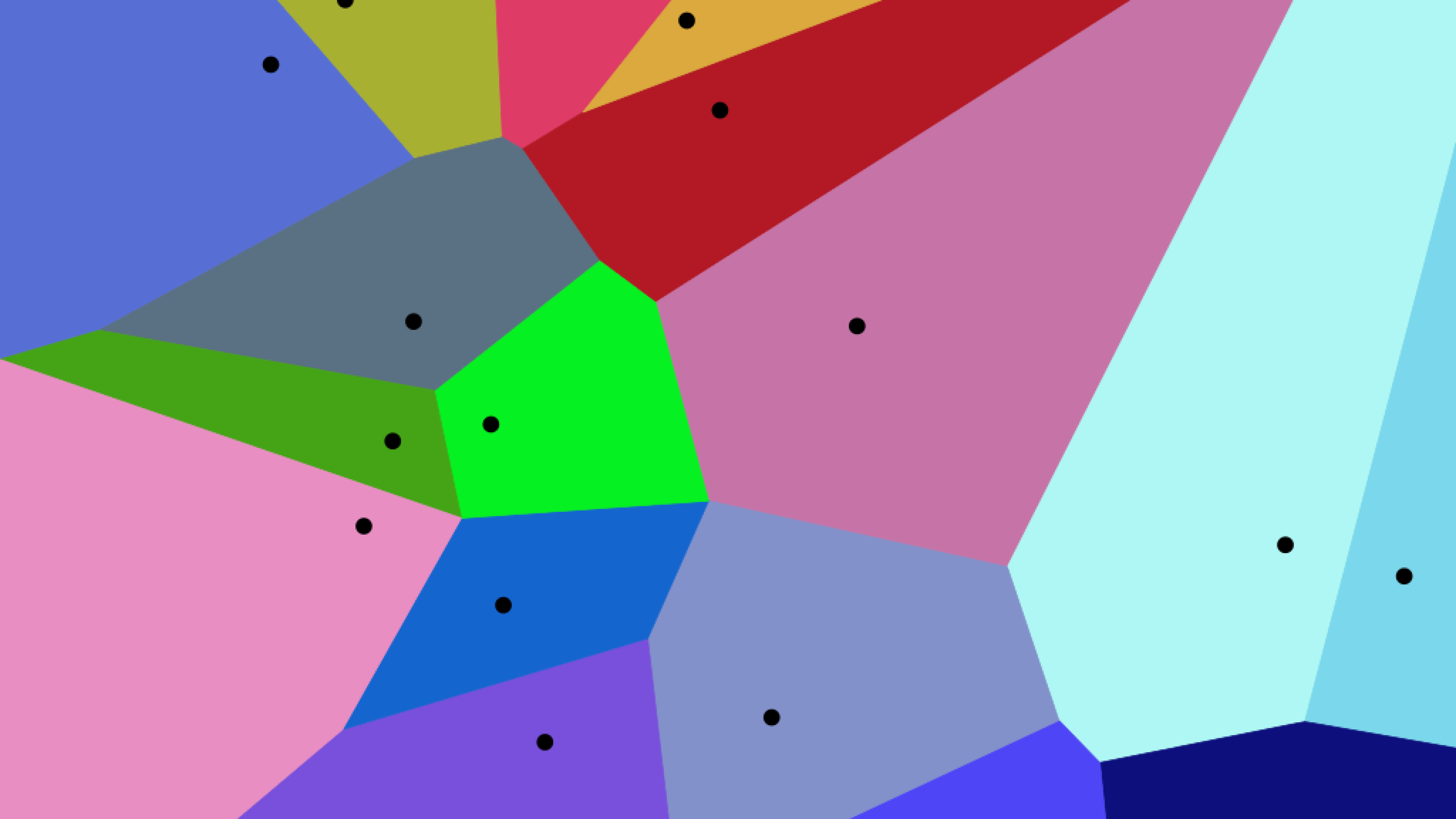
S – zbiór punktów na płaszczyźnie
E – rozważana przestrzeń

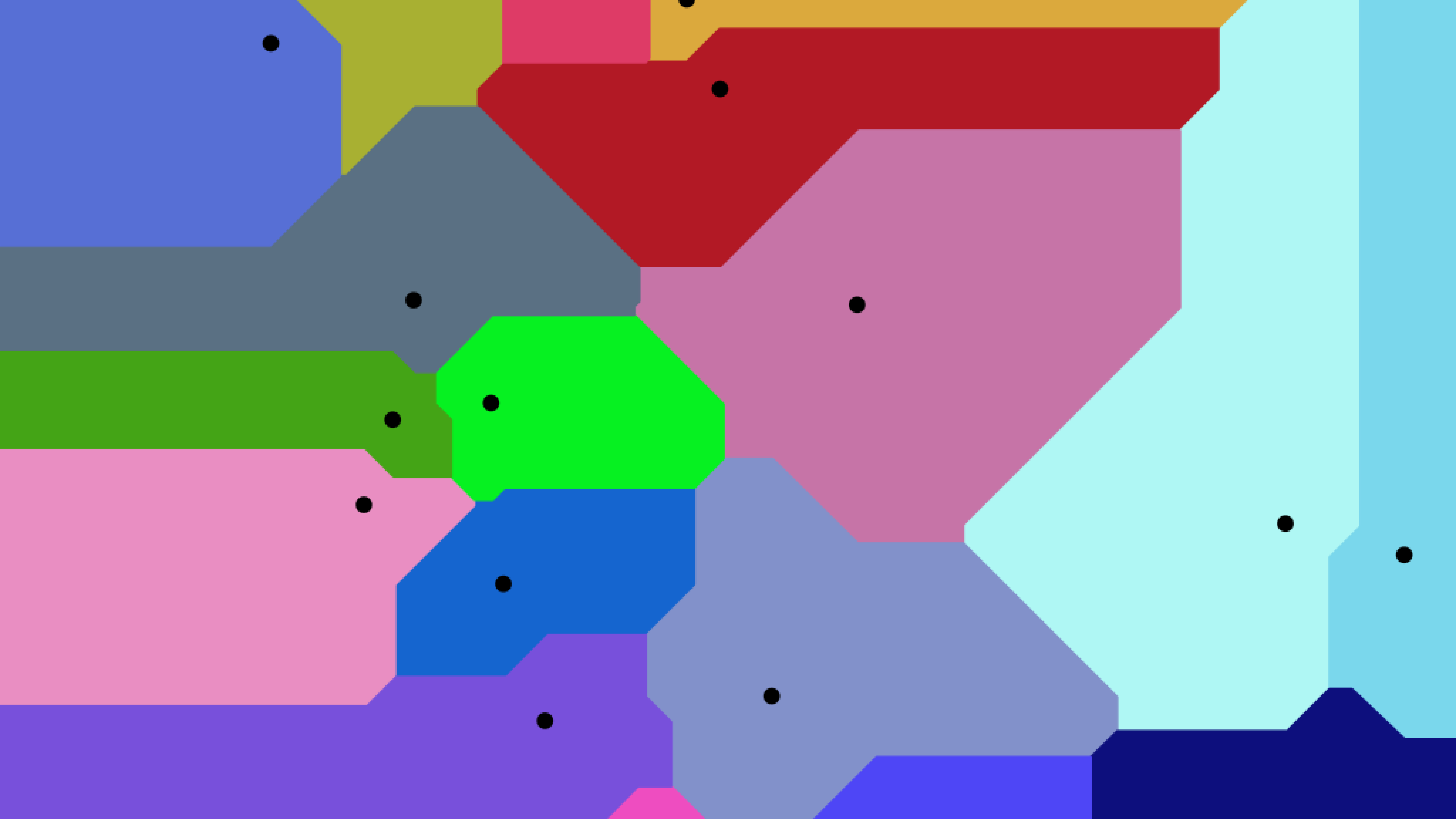
- Komórką Woronoja punktu **p** nazywamy:



$$Vor_S(p) = \{x \in E | \forall q \in S, d(x,p) \leq d(x,q)\},$$

gdzie d jest odległością.

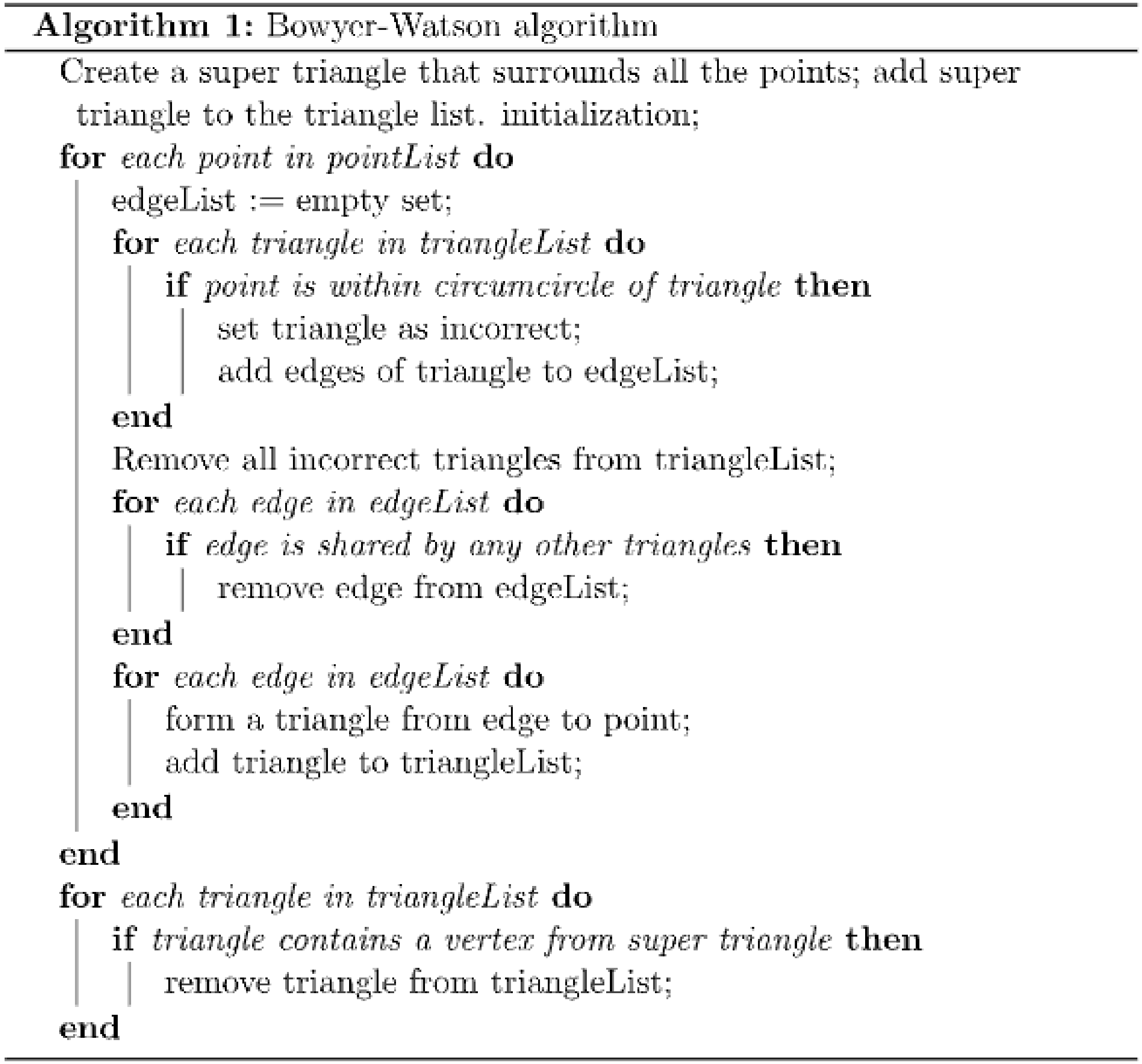




Algorytm Bowyera-Watsona

- algorytm iteracyjny
- algorytm wyznaczający trójkąty składające się na triangulację Delaunay’a zadanej chmury punktów
- złożoność zależna od implementacji procedury przeszukiwania trójkątów – przedstawiony alg. $O(n^2)$

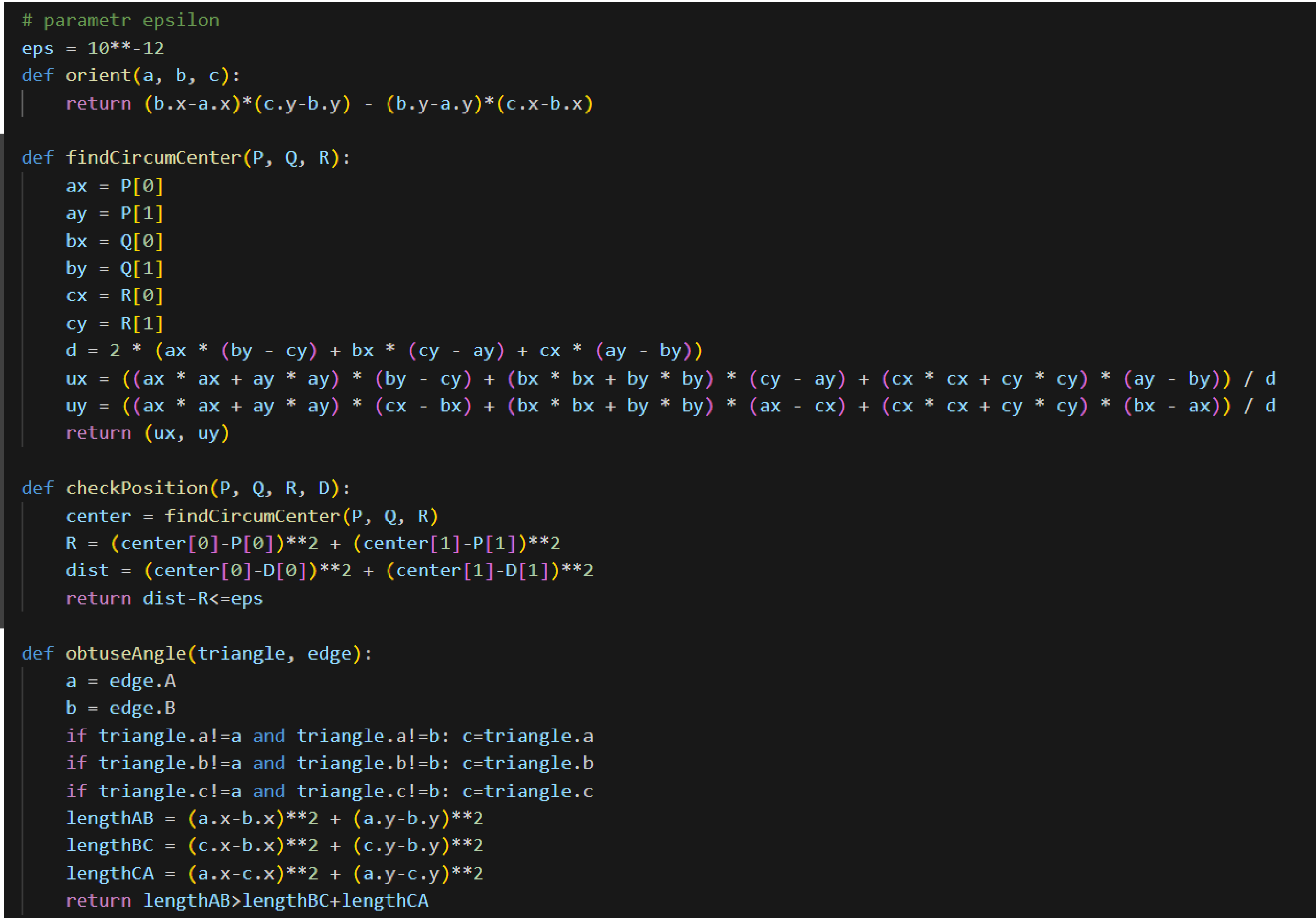
Kroki algorytmu:



Algorithm 1: Bowyer-Watson algorithm

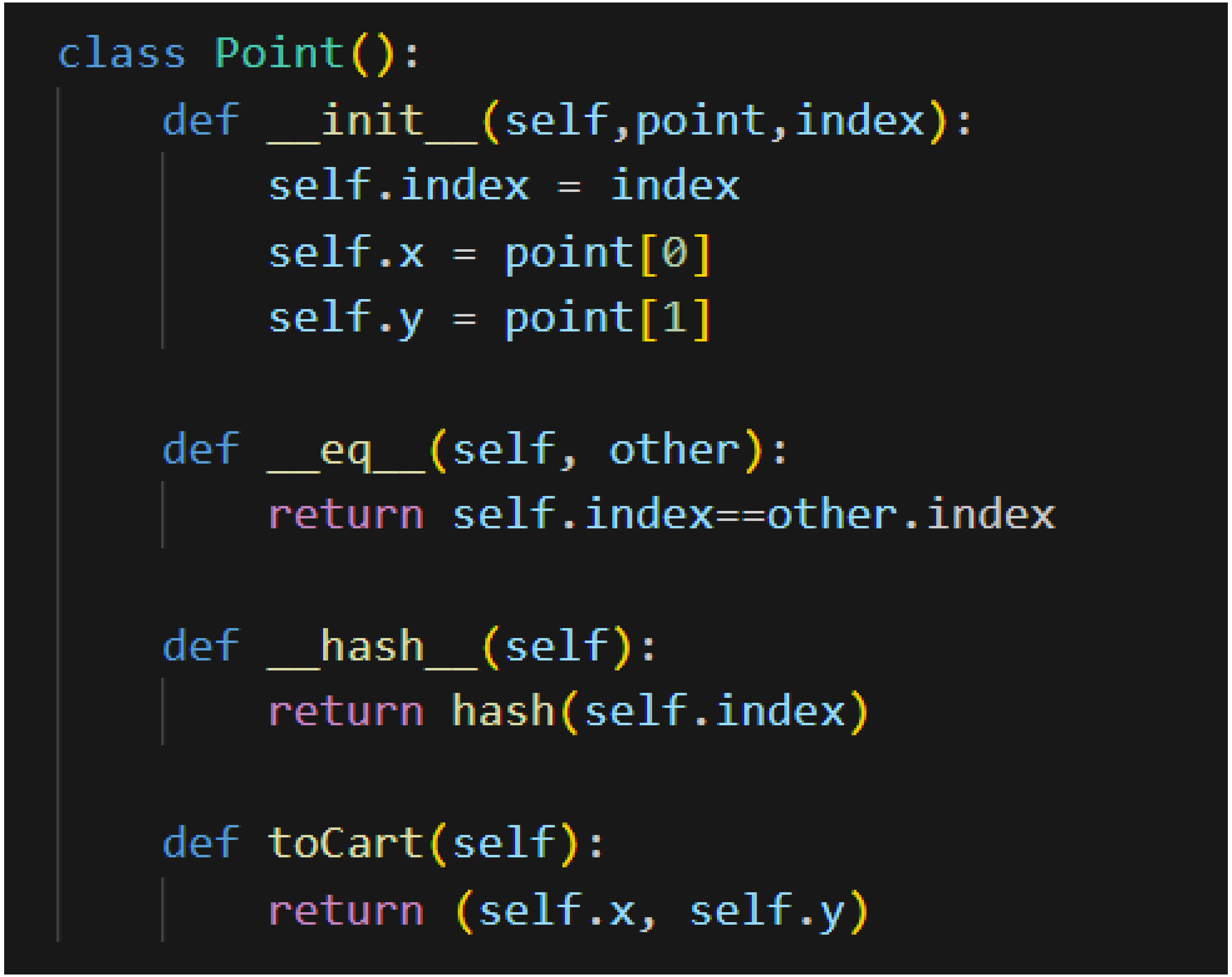
```
Create a super triangle that surrounds all the points; add super
 triangle to the triangle list. initialization;
for each point in pointList do
    edgeList := empty set;
    for each triangle in triangleList do
        if point is within circumcircle of triangle then
            set triangle as incorrect;
            add edges of triangle to edgeList;
    end
    Remove all incorrect triangles from triangleList;
    for each edge in edgeList do
        if edge is shared by any other triangles then
            remove edge from edgeList;
    end
    for each edge in edgeList do
        form a triangle from edge to point;
        add triangle to triangleList;
    end
end
for each triangle in triangleList do
    if triangle contains a vertex from super triangle then
        remove triangle from triangleList;
end
```

Funkcje pomocnicze

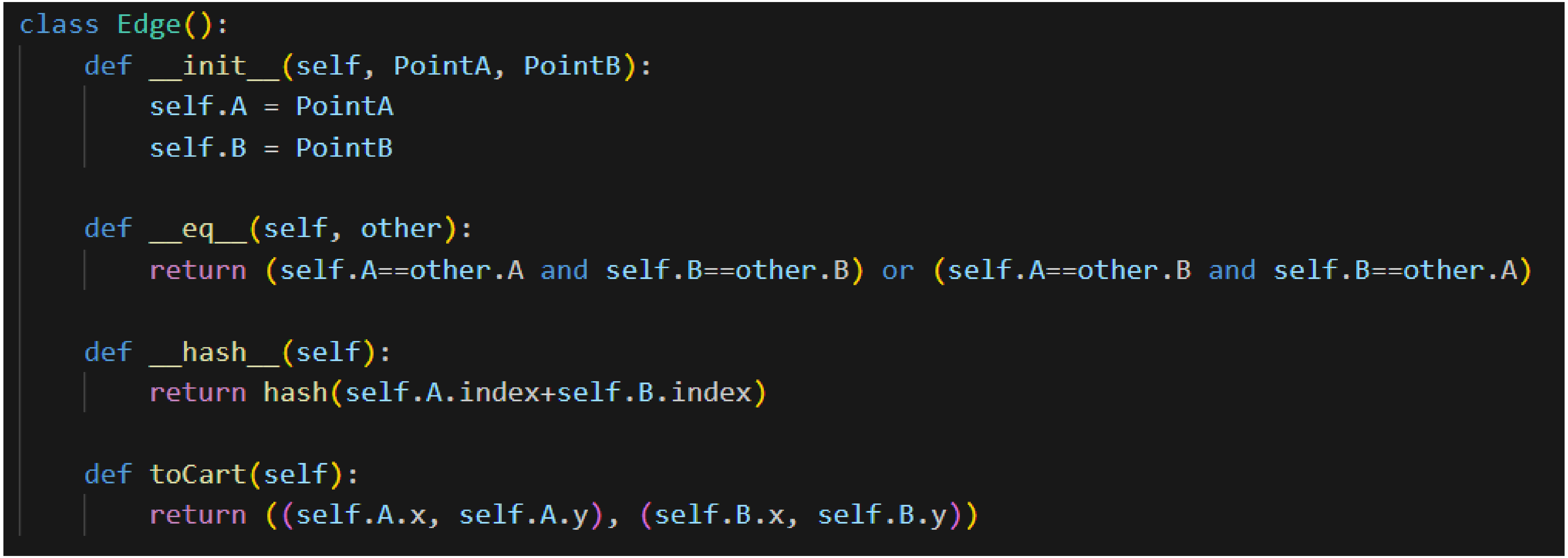


```python
# parametr epsilon
eps = 10**-12
def orient(a, b, c):
    return (b.x-a.x)*(c.y-b.y) - (b.y-a.y)*(c.x-b.x)

def findCircumCenter(P, Q, R):
    ax = P[0]
    ay = P[1]
    bx = Q[0]
    by = Q[1]
    cx = R[0]
    cy = R[1]
    d = 2 * (ax * (by - cy) + bx * (cy - ay) + cx * (ay - by))
    ux = ((ax * ax + ay * ay) * (by - cy) + (bx * bx + by * by) * (cy - ay) + (cx * cx + cy * cy) * (ay - by)) / d
    uy = ((ax * ax + ay * ay) * (cx - bx) + (bx * bx + by * by) * (ax - cx) + (cx * cx + cy * cy) * (bx - ax)) / d
    return (ux, uy)

def checkPosition(P, Q, R, D):
    center = findCircumCenter(P, Q, R)
    R = (center[0]-P[0])**2 + (center[1]-P[1])**2
    dist = (center[0]-D[0])**2 + (center[1]-D[1])**2
    return dist-R<=eps

def obtuseAngle(triangle, edge):
    a = edge.A
    b = edge.B
    if triangle.a!=a and triangle.a!=b: c=triangle.a
    if triangle.b!=a and triangle.b!=b: c=triangle.b
    if triangle.c!=a and triangle.c!=b: c=triangle.c
    lengthAB = (a.x-b.x)**2 + (a.y-b.y)**2
    lengthBC = (c.x-b.x)**2 + (c.y-b.y)**2
    lengthCA = (a.x-c.x)**2 + (a.y-c.y)**2
    return lengthAB>lengthBC+lengthCA
```

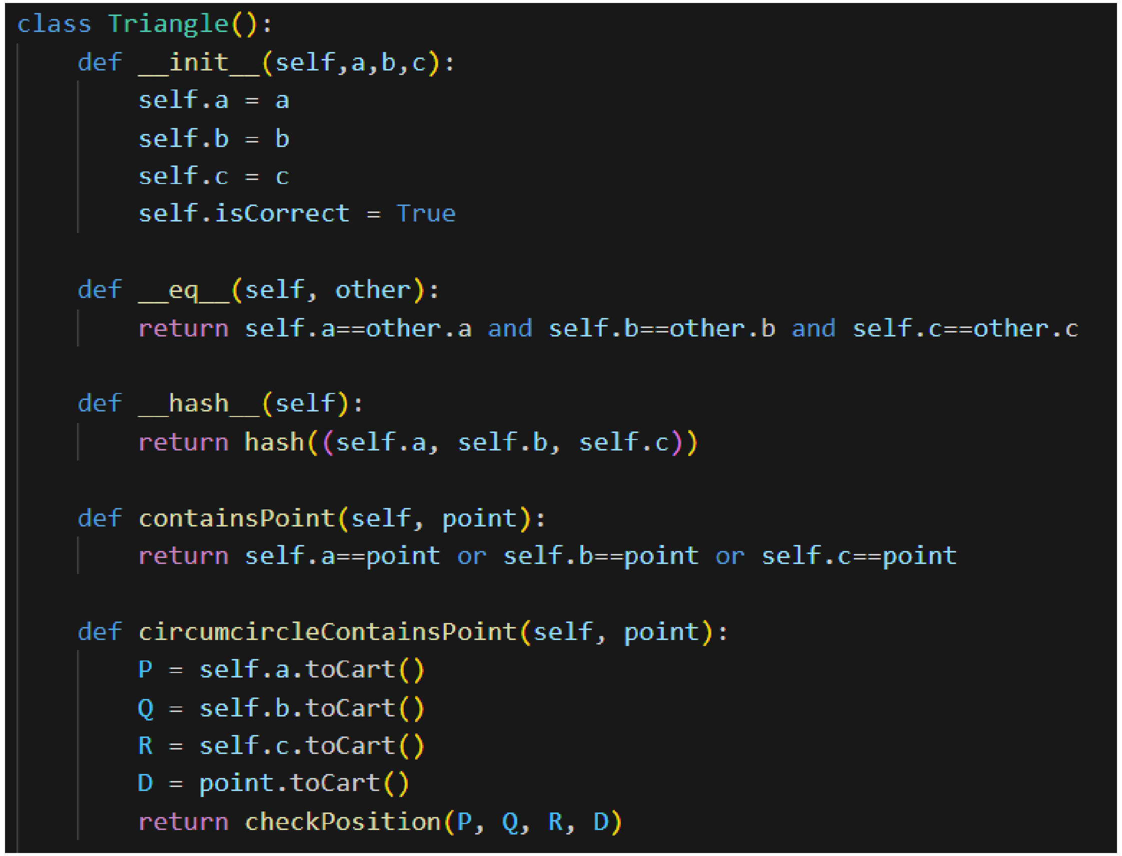
Klasy



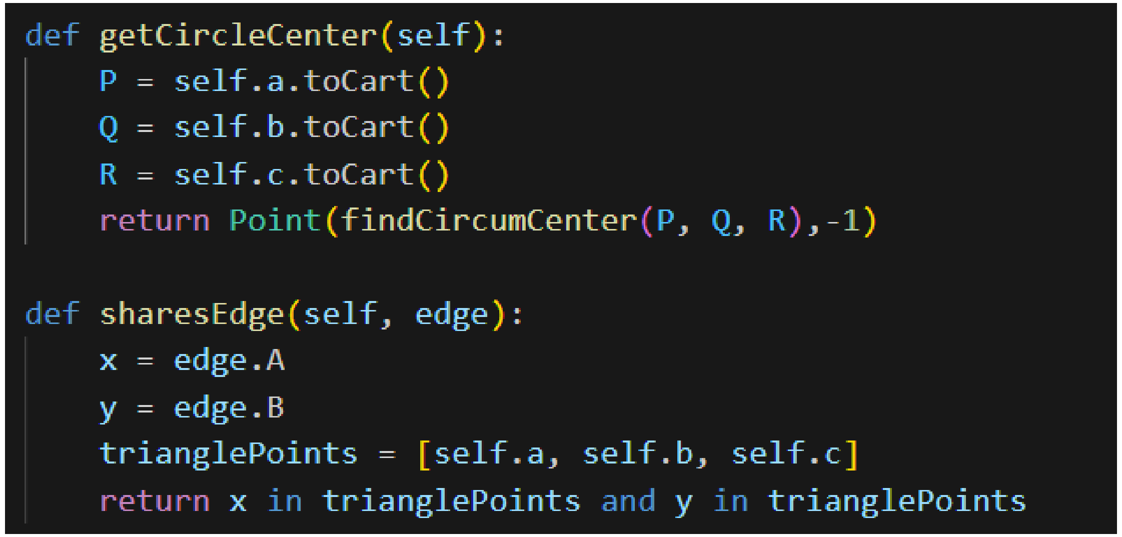
```python
class Point():
    def __init__(self,point,index):
        self.index = index
        self.x = point[0]
        self.y = point[1]

    def __eq__(self, other):
        return self.index==other.index

    def __hash__(self):
        return hash(self.index)

    def toCart(self):
        return (self.x, self.y)
```



```python
class Edge():
    def __init__(self, PointA, PointB):
        self.A = PointA
        self.B = PointB

    def __eq__(self, other):
        return (self.A==other.A and self.B==other.B) or (self.A==other.B and self.B==other.A)

    def __hash__(self):
        return hash(self.A.index+self.B.index)

    def toCart(self):
        return ((self.A.x, self.A.y), (self.B.x, self.B.y))
```



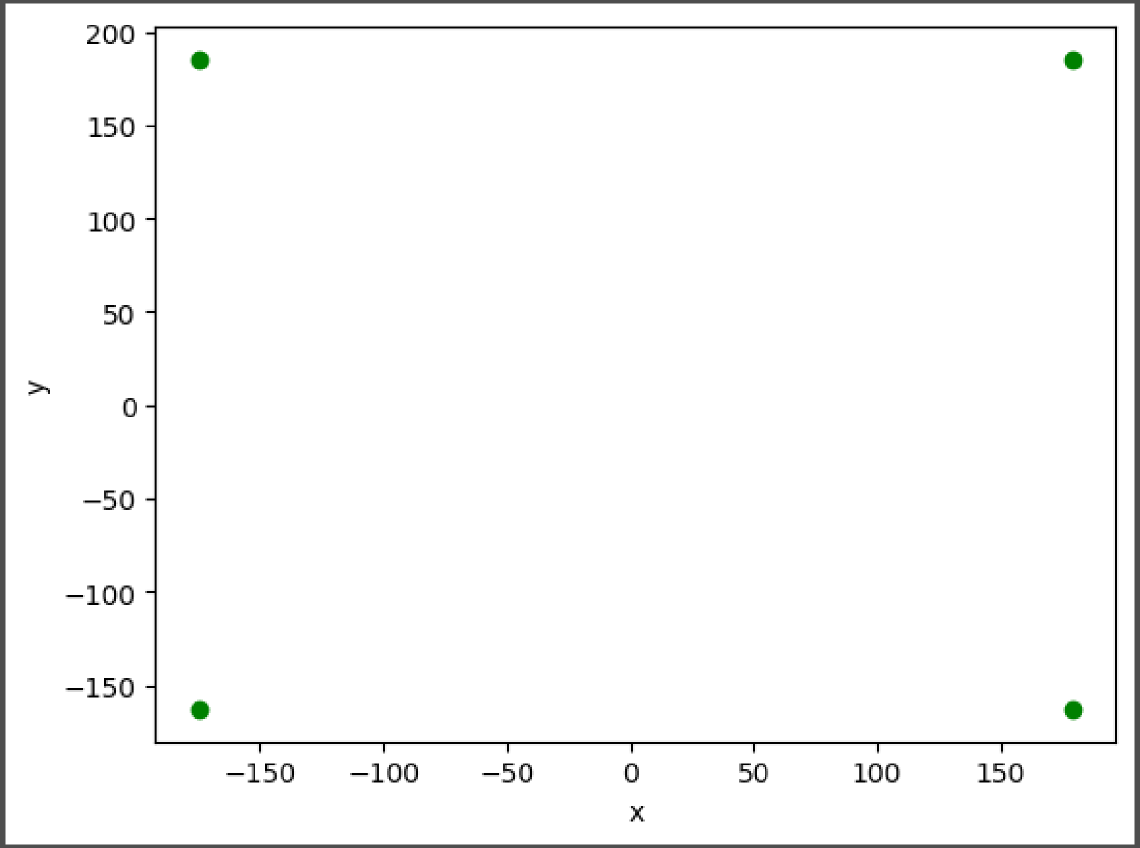
```
class Triangle():
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
        self.isCorrect = True

    def __eq__(self, other):
        return self.a==other.a and self.b==other.b and self.c==other.c

    def __hash__(self):
        return hash((self.a, self.b, self.c))

    def containsPoint(self, point):
        return self.a==point or self.b==point or self.c==point

    def circumcircleContainsPoint(self, point):
        P = self.a.toCart()
        Q = self.b.toCart()
        R = self.c.toCart()
        D = point.toCart()
        return checkPosition(P, Q, R, D)
```

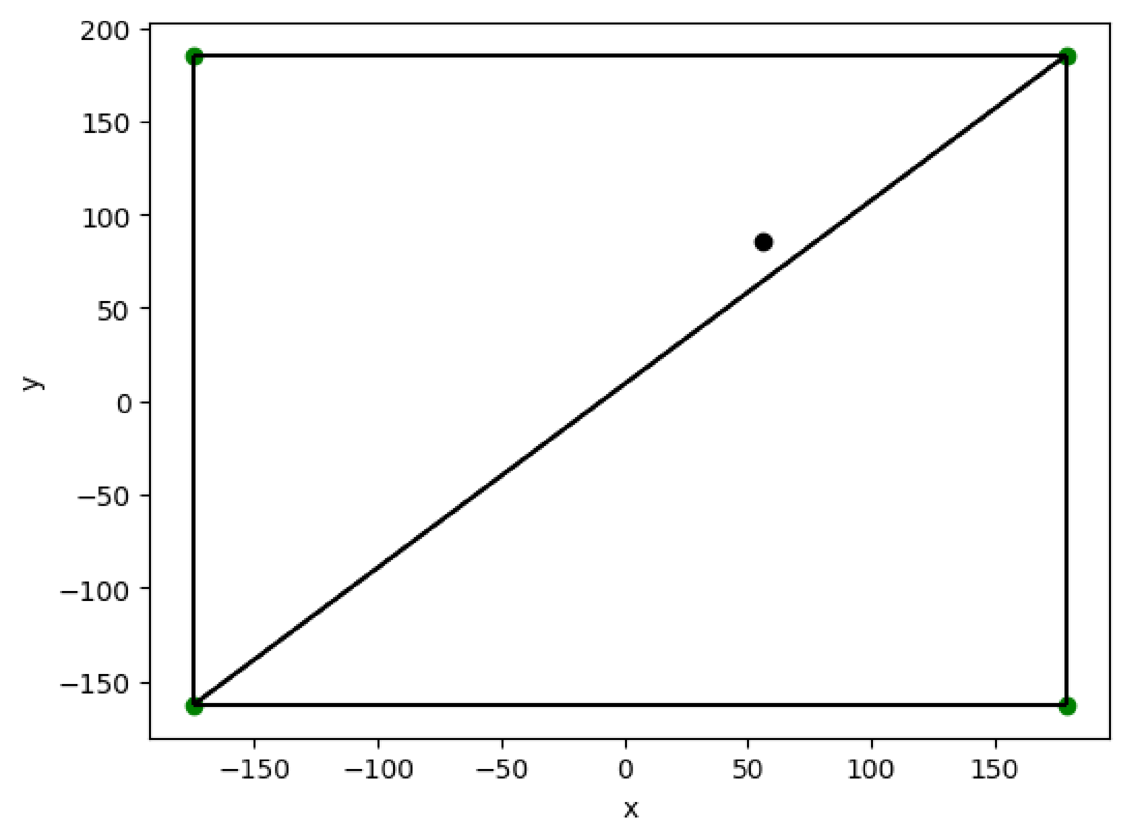


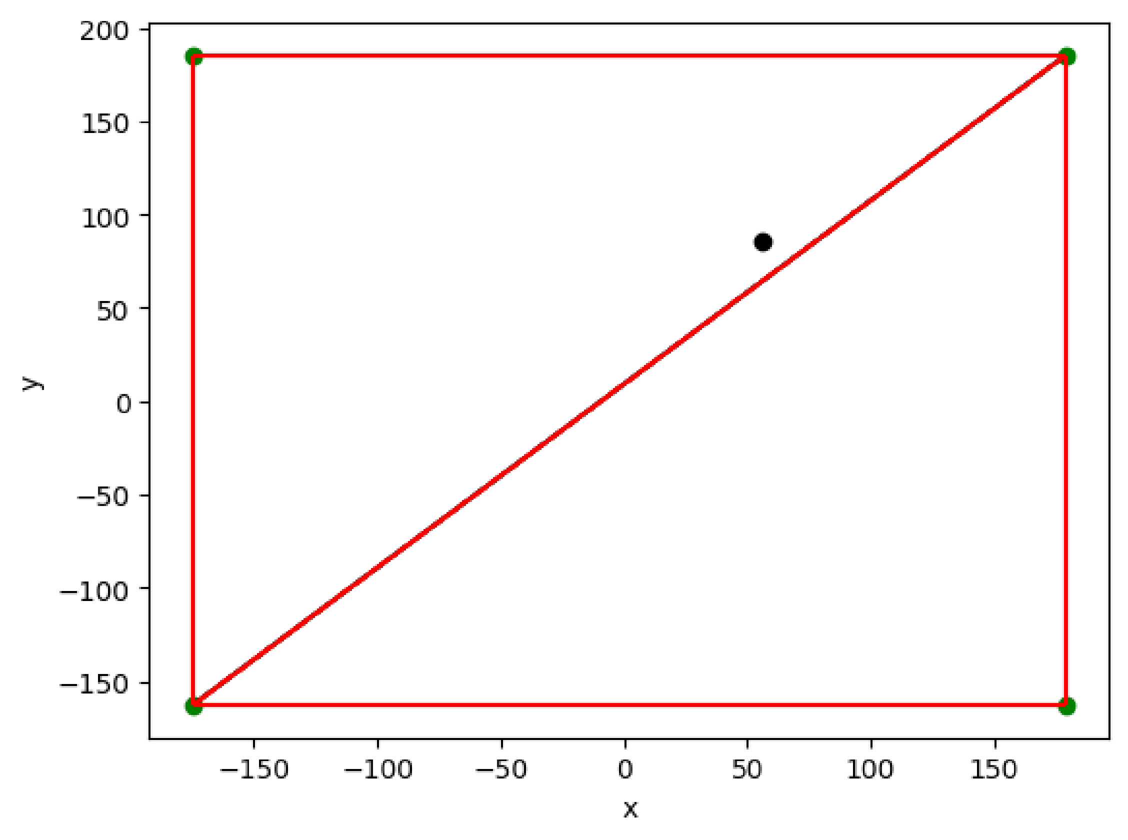
```
def getCircleCenter(self):
    P = self.a.toCart()
    Q = self.b.toCart()
    R = self.c.toCart()
    return Point(findCircumCenter(P, Q, R),-1)

def sharesEdge(self, edge):
    x = edge.A
    y = edge.B
    trianglePoints = [self.a, self.b, self.c]
    return x in trianglePoints and y in trianglePoints
```

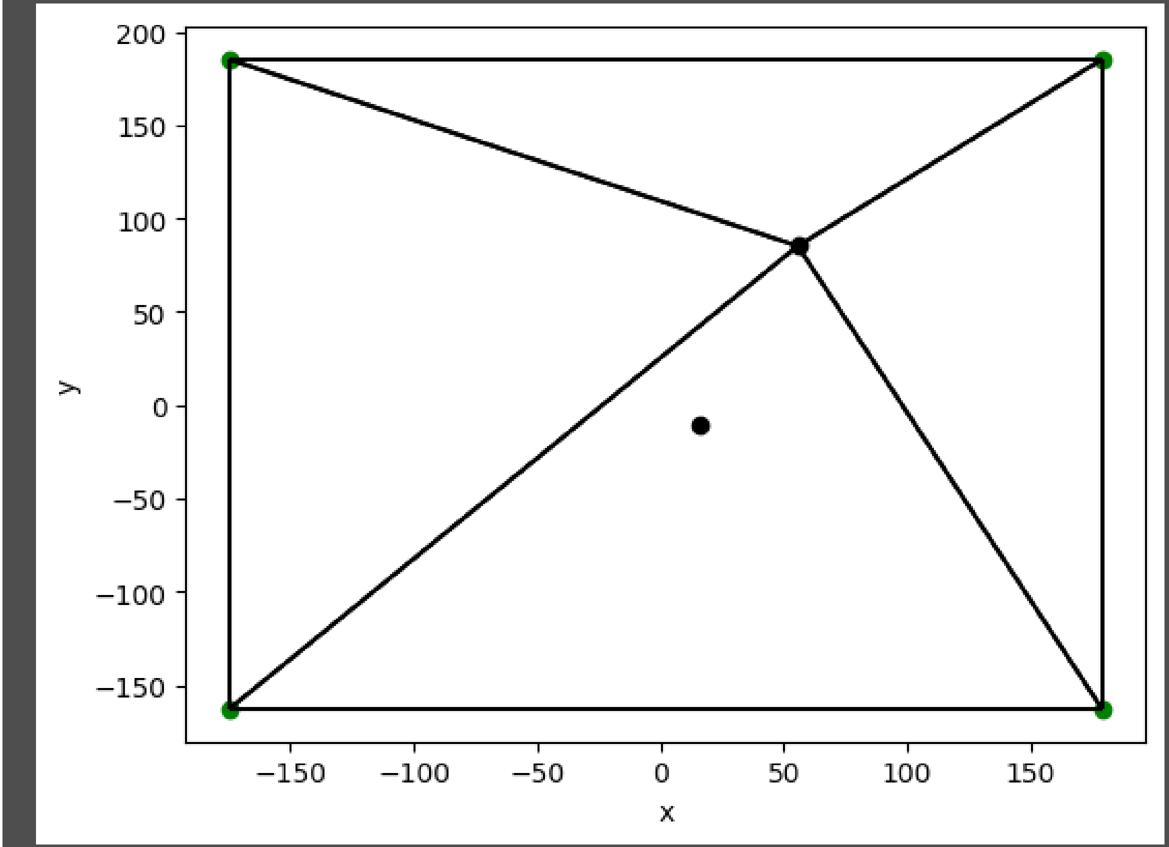
Wizualizacja działania algorytmu

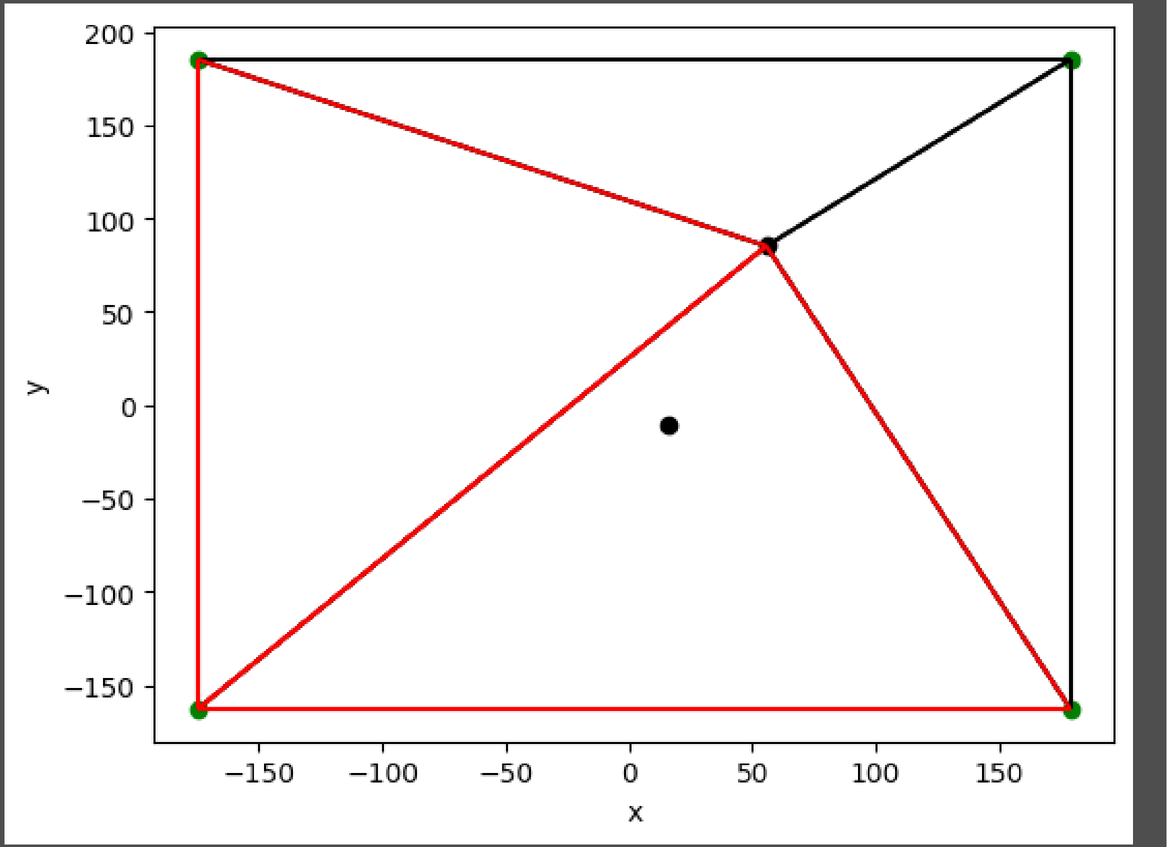
Dane wejściowe: 7 losowo wygenerowanych punktów na płaszczyźnie

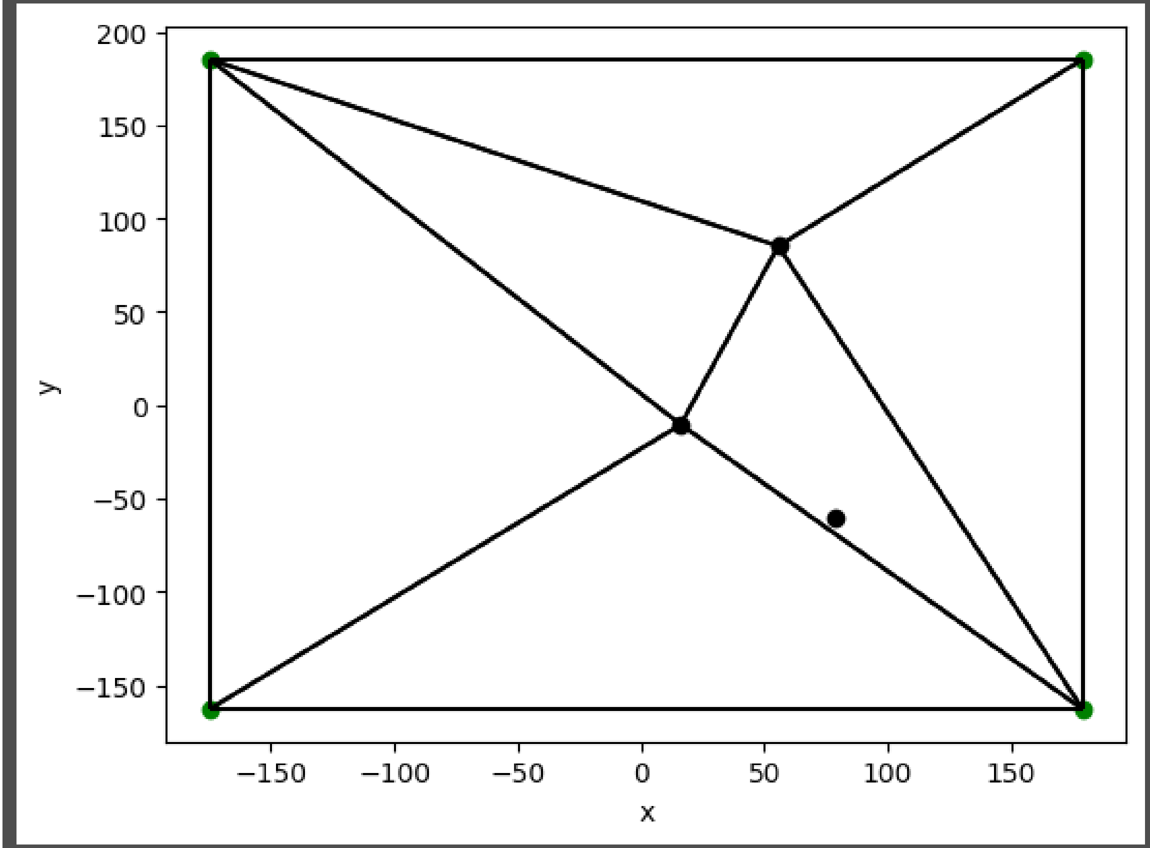

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

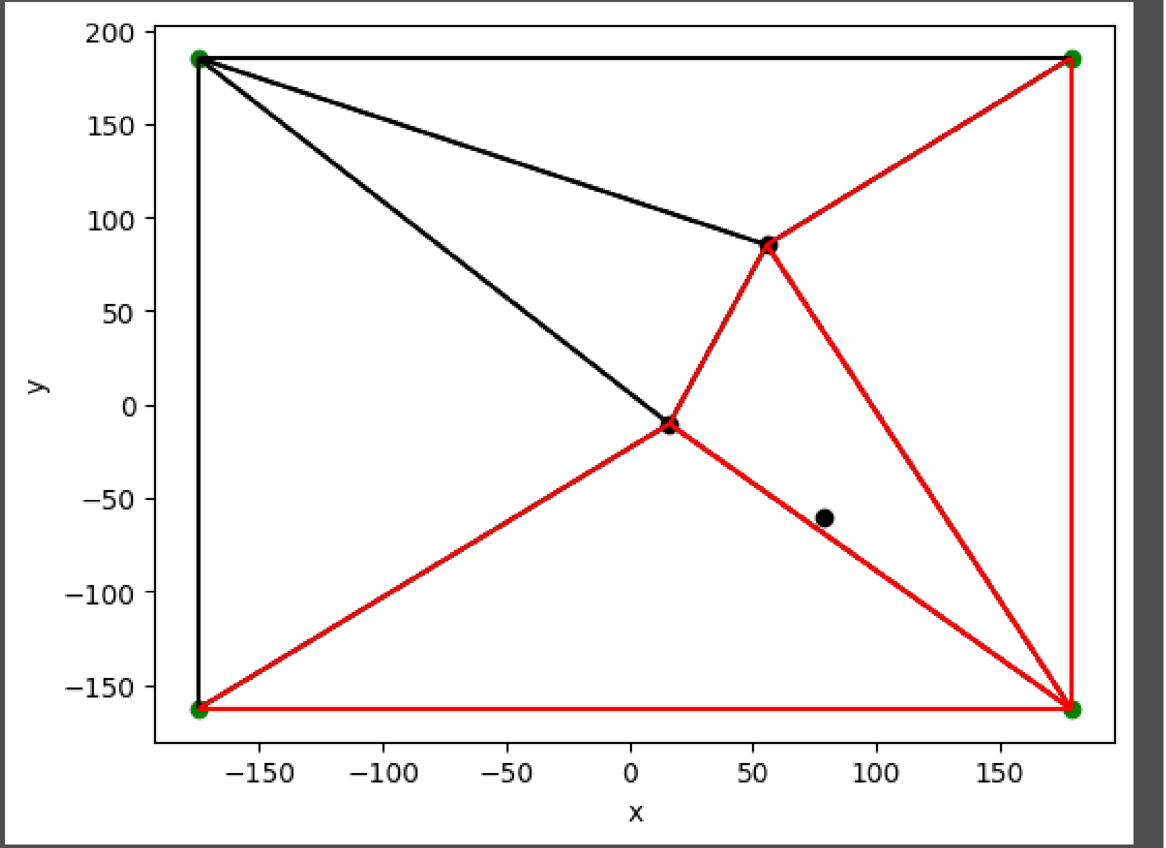

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

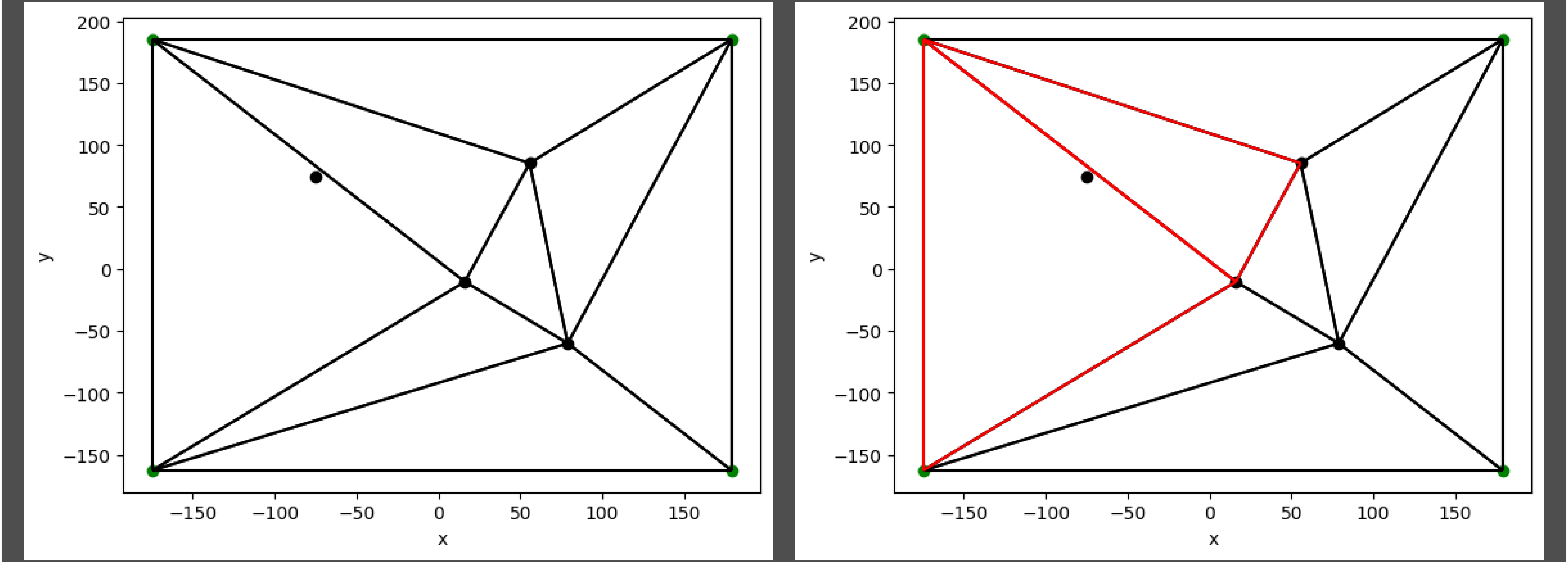

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

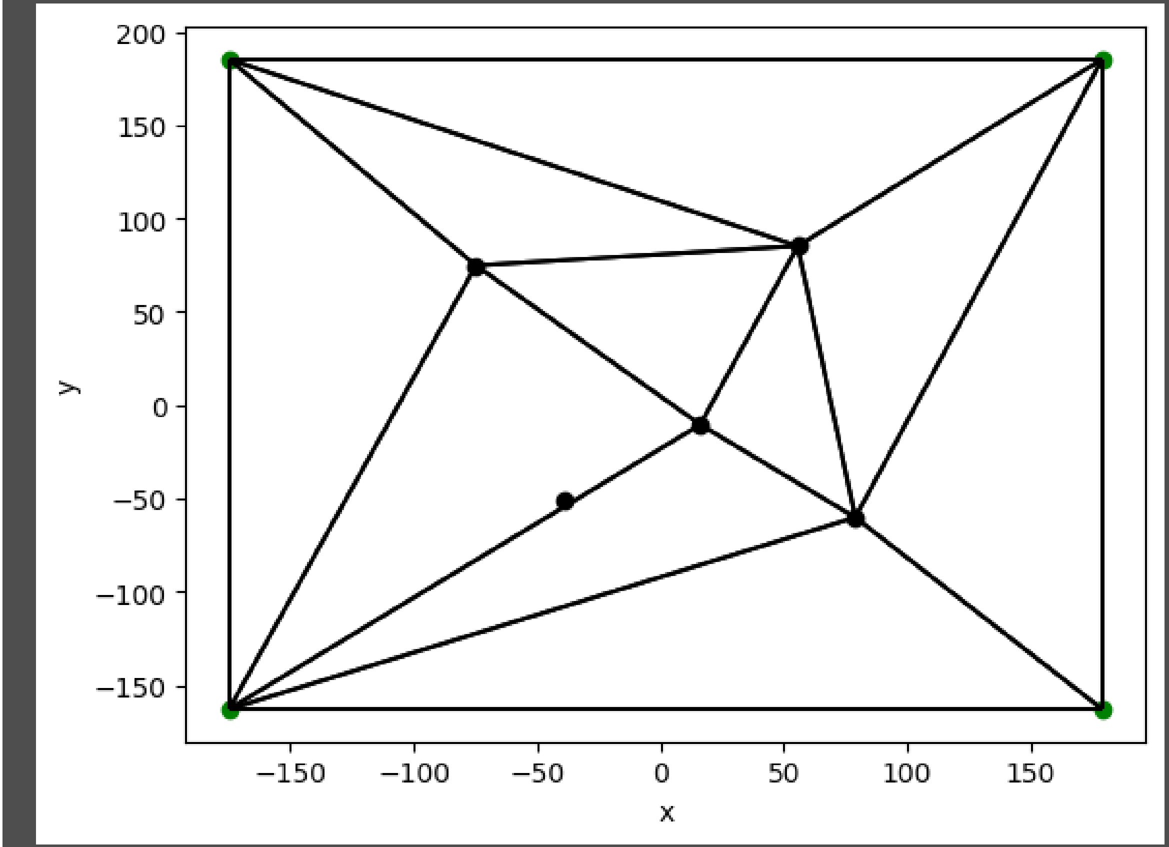

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

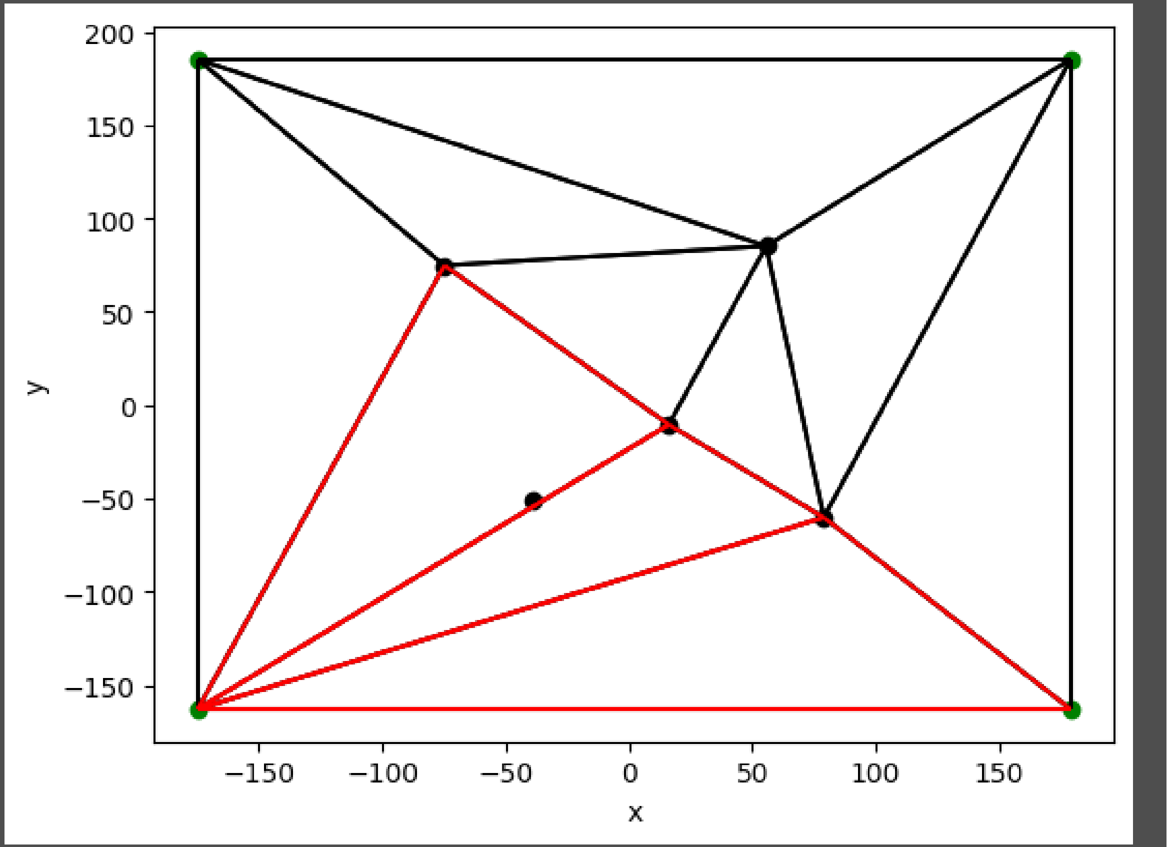

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

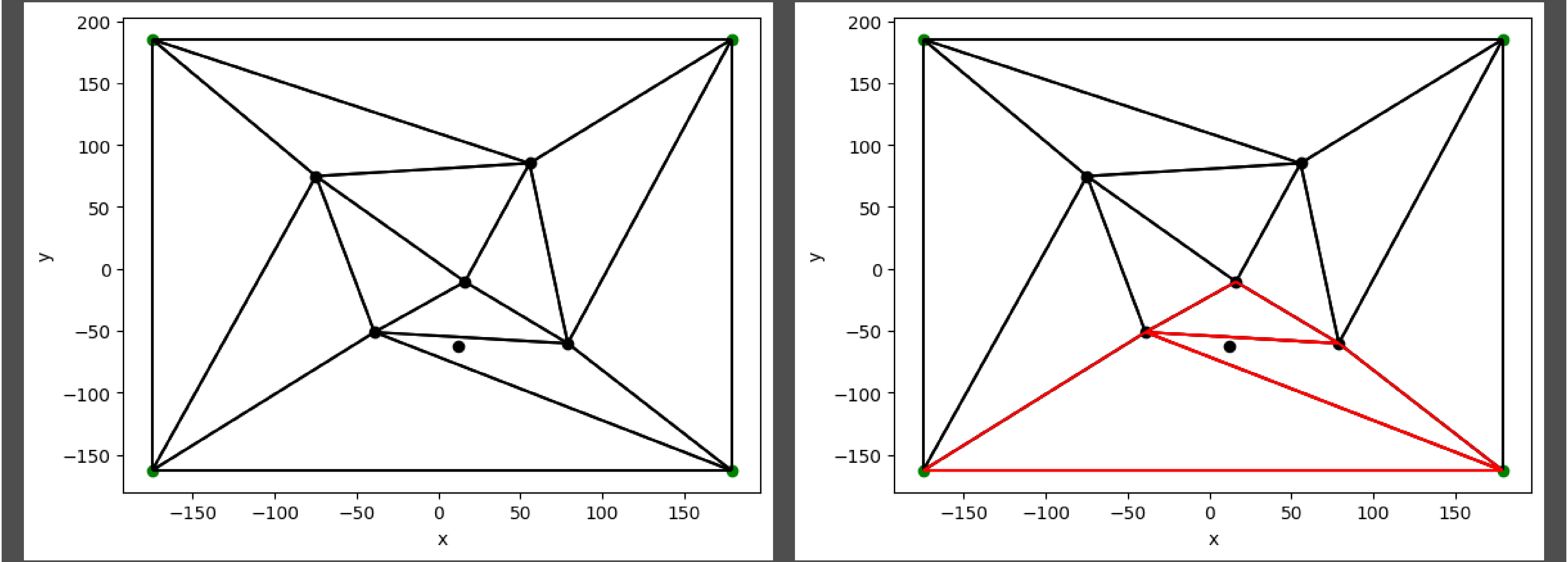

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

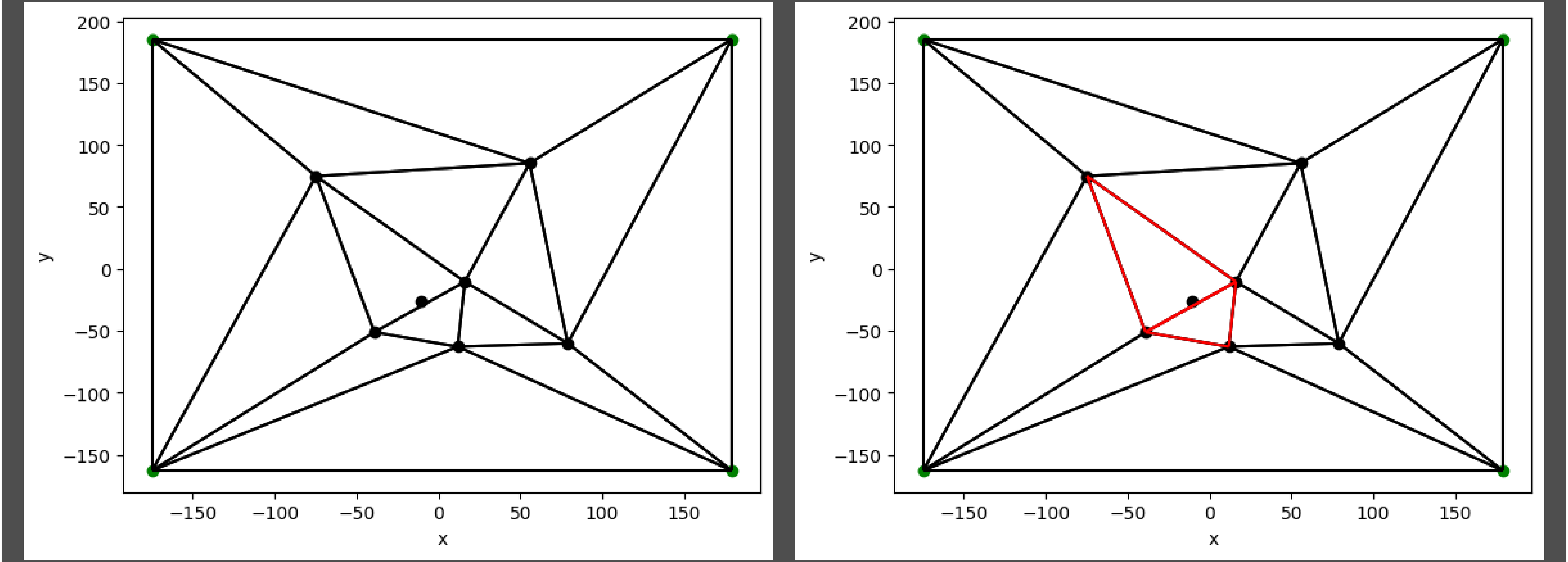

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

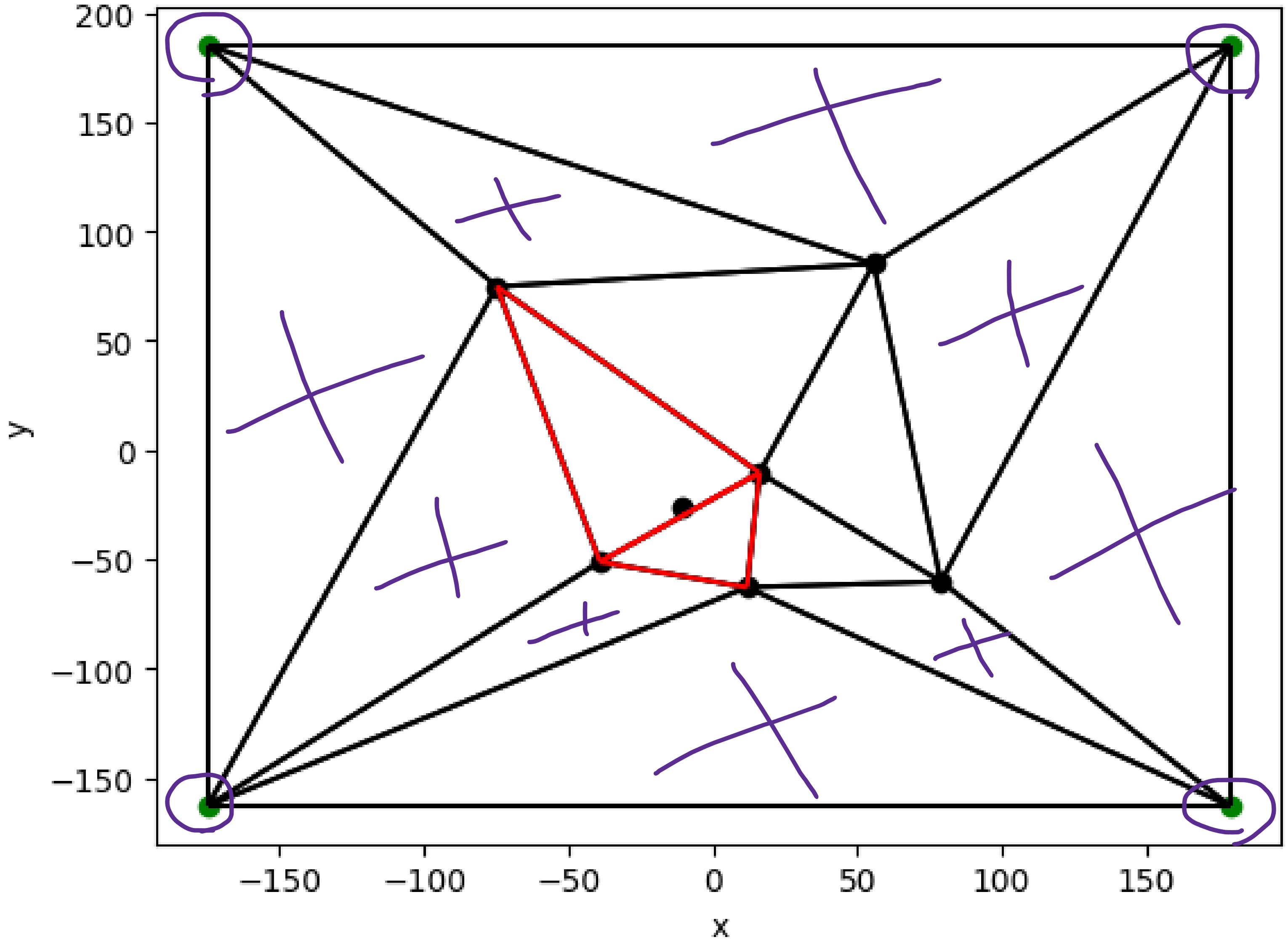

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x
200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

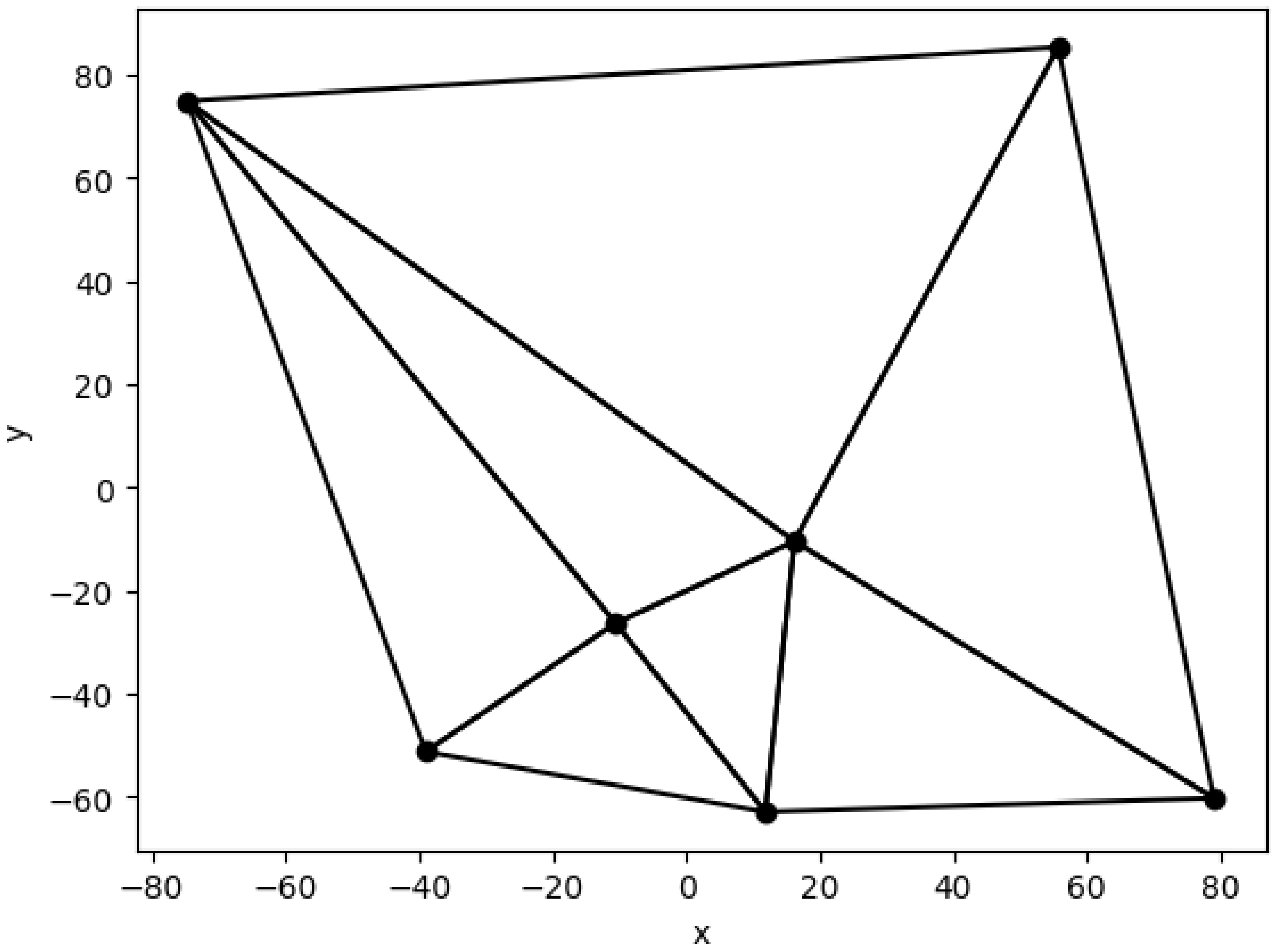

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

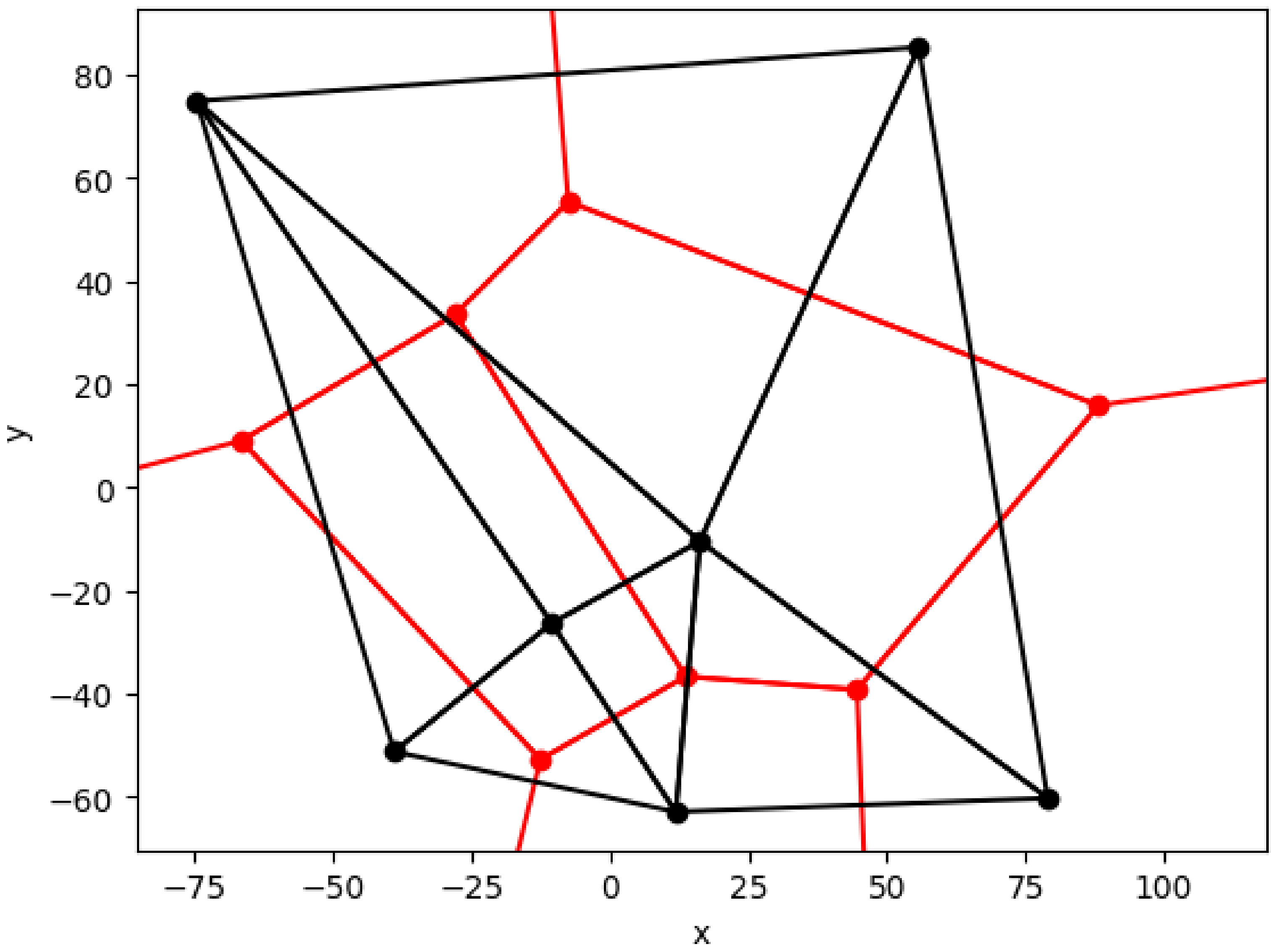

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x


200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x
200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x

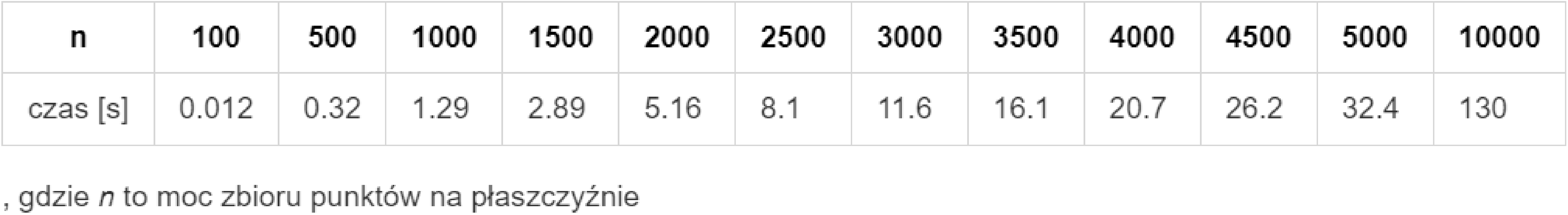

200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x
200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x


200
150
100
50
0
−50
−100
−150
y
−150
−100
−50
0
50
100
150
x


80
60
40
20
0
−20
−40
−60
y
−80
−60
−40
−20
0
20
40
60
80
x


80
60
40
20
0
−20
−40
−60
y
−75
−50
−25
0
25
50
75
100
x

Testy czasowe algorytmu



n	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	10000
czas [s]	0.012	0.32	1.29	2.89	5.16	8.1	11.6	16.1	20.7	26.2	32.4	130

, gdzie *n* to moc zbioru punktów na płaszczyźnie

Źródła:

- https://en.wikipedia.org/wiki/Voronoi_diagram
- https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm
- https://www.baeldung.com/cs/voronoi-diagram