Khuyen Le Thi Minh s5128 – ASD 7

TASK DESCRIPTION:

Find an algorithm which determines intersection of two input convex polygons. Prove algorithm correctness and estimate its complexity.

ALGORITHM:

In order to find the intersection of two polygons, first we will find all the points that are intersection points between 2 edges, one from first polygon and second from second polygon. After that we'll add the point from one polygon that is completely inside second polygon by checking the x and y coordinates of that point.

```
construct Point
{
//we add the name for point for easier recognistion (the initiated for polygon should have it, but the points in intersection doen't necessary need to)
        char name;
// the position of x and y on the plane will be in double since we'll have to use them in calculation and we want to avoid of missing the decimals
        double x;
        double y;
}
construct Edge (Point begin, Point end){
//we add the name for edge for easier recognation, the name should be the cancatenation of the name of beginning and end points
        String name = name.of.begin + name.of.end;
   Point begin;
   Point end;
}
construct Polygon(Point[] points, Edge[] edges){
   Point[] points;
   Edge[] edges;
}
//we create separate class to find equation from the line. The construction will take in edge as an input.
construct Equation(Edge edge){
// since we'll calculate equation of the line, it's worth to remember the point of beginning and end of the line for later calculation
        Point begin = begin.point.of.edge;
        Point end = end.point.of.edge;
// we'll remember the name of edge to compare later
        String name = name.of.edge;
// number a in equation (y = ax + b)
        double x;
// number b in equation (y = ax + b)
        double y;
// we'll remember if the line from equation will be horizontal, vrtical or diagonal
        String type;
// we check if the position x of both beginning and end point of the line is the same, if it's then the line is vertical.
                if(position.x.of.beginning.point == position.x.of.end.point) then
                {
                        x = position.x.of.beginning.point
                        y = 0;
                        type = "pion";
                }
// we check if the position y of both beginning and end point of the line is the same, if it's then the line is horizontal.
                else if(position.y.of.beginning.point == position.y.of.end.point)
                {
                        x = 0;
                        y = position.y.of.beginning.point;
                        type = "poziom";
                }
//if the line is neither vrtical or horizontal, we calculate the number a and b in equation (y = ax + b). Since the calculation won't depend on number of points and edges, the time complexity is O(1)
                else
                {
//we calculate x using formula (a = (y2 - y1)/(x2 - x1))
```

```
                x = (position.y.of.end.point - position.y.of.beginning.point)/(position.x.of.end.point -
position.x.of.beginning.point);
//we calculate y using formula (b = y - ax)
                y = -x * position.x.of.beginning.point + position.y.of.beginning.point;
                type = "ukos";
            }fi;

}
// we create separate class to remember the point of intersection between 2 lines and the name of those lines
construct Intersect (String firstLine, String secondLine, Point point){
        String firstLine;
        String secondLine;
        Point point;
}
// we create the class to put together the calcukations.
construct Intersection {
// first we'll create the method to get all the equation from the edges of polygon. The method will take in the polygon
and the size of its point/ edge as input and return the array of Equations
        getAllEquation (Polygon polygon, number size)
        {
// we create the array that has the same size as the number of edges in polygon
            Equation[] result [size];
// we run through all edges in polygon and calculate for each of edges. Since the time complexity for calculation
equation is O(1) so the time complexity for this loop will be O(n) where n = size
            for(int i = 0; i< size; i++) do
            {
                result[i] = Equation(edge.i.from.polygon);
            } od;
//we return the array of all equation in polygon.
            return result;
        }
//we create the method to check if the intersection point between 2 lines is actually lying on those lines. The method
will take in as input: 2 equations as 2 lines, the position x and y of the intersection point and will return a boolean
value to state if it's true of flase.
        boolean checkIfLyingOnLines(Equation first, Equation second, double x, double y)
        {
// first we create the boolean value that we'll return later and set it as default on false.
            boolean isBelongTo = false;
//we will get the position x and y from the beginning and end points of both lines
            double first_y = positon.y.of.beginning.point.of.first.equation;
            double second_y = positon.y.of.end.point.of.first.equation;
            double first_x = positon.x.of.beginning.point.of.first.equation;
            double second_x = positon.x.of.end.point.of.first.equation;
            double third_y = positon.y.of.beginning.point.of.second.equation;
            double fourth_y = positon.y.of.end.point.of.second.equation;
            double third_x = positon.x.of.beginning.point.of.second.equation;
            double fourth_x = positon.x.of.end.point.of.second.equation;
//we check whether the point with position x and y is lying inside the rectangle create by 4 points from 2 lines in
input. If it's inside then we return the result as true. The time complexity for this calculation is O(1)
            if((((y >= first_y && y <= second_y) || (y <= first_y && y >= second_y)) && ((x >= first_x && x
<= second_x) || (x <= first_x && x >= second_x))) &&
                        (((y >= third_y && y <= fourth_y) || (y <= third_y && y >= fourth_y)) && ((x
>= third_x && x <= fourth_x) || (x <= third_x && x >= fourth_x)))          ) then
            {
                isBelongTo = true;
            }fi;
// we return the result
            return isBelongTo;
        }
//we create the method to get the points with maximum and minimum position x and y. The input would be the
polygon and its size. We'll return the result as array of points
        public static Point[] getMaxAndMin(Polygon polygon, number size)
        {
//The array of points would have size of 4 in the order: Max_X, Min_X, Max_Y, Min_Y. We initially set the all position in
result array to the first point in polygon and after that we'll switch it if we find the right one.
            Point[] result = {point[0].in.polygon, point[0].in.polygon, point[0].in.polygon, point[0].in.polygon};
```

```
//we create the loop running through all points in polygon to get the point with maximum and minimum value of x
and y. The time complexity of this loop would depend on the size of polygon so O(n) where n = size.
        for(int i = 0; i < size; i++) do
        {
//we check if the first point in result has lower x position than the point with position i in polygon, if yes then we
assign point with position i in polygon to first point in result array.
            if(position.x.of.result[0] < position.x.of.point.in.position.i.in.polygon) then
            {
                result[0] = point.in.position.i.in.polygon;
            }fi;
//we check if the second point in result has lower higher position than the point with position i in polygon, if yes then
we assign point with position i in polygon to second point in result array.
            if(position.x.of.result[1] > position.x.of.point.in.position.i.in.polygon) then
            {
                result[1] = point.in.position.i.in.polygon;
            }fi;
//we check if the third point in result has lower y position than the point with position i in polygon, if yes then we
assign point with position i in polygon to third point in result array.
            if(position.y.of.result[2] < position.y.of.point.in.position.i.in.polygon) then
            {
                result[2] = point.in.position.i.in.polygon;
            }fi;
//we check if the last point in result has higher y position than the point with position i in polygon, if yes then we
assign point with position i in polygon to last point in result array.
            if(position.y.of.result[3] > position.y.of.point.in.position.i.in.polygon) then
            {
                result[3] = point.in.position.i.in.polygon;
            }fi;
        } od;
//we return the result
        return result;
    }
//we create the methode to divide the edges of polygon in 4 part: top-left, top-tight, bottom-left, bottom-right with
the divide points as the points with maximum and minimum x and y value in polygon. This way we can latter chech if
the point with some x and y position is lying inside the polygon. We'll return the result as the 4-elements linked list
with list of the edges inside the side. The position of elements in result would be: TopLeft, TopRight, BottomLeft,
BottomRight.
    LinkedList(LinkedList(Edge)) getSides(Polygon polygon, number size)
    {
//we inicially create 4 linked lists for each side and 1 linked list that later we'll add 4 smaller list in.
        LinkedList(LinkedList(Edge)) result;
        LinkedList(Edge) topLeft;
        LinkedList(Edge) topRight;
        LinkedList(Edge) bottomLeft;
        LinkedList(Edge) bottomRight;
//we get the array with points with maximum and minimum position x and y in the polygon.
        Point[] minMax = getMaxAndMin(polygon);
//we have the loop that will run through all the edges in the polygon. The time complexity for this would be O(n)
where n = size
        for(int i = 0; i < size; i++)do
        {
//we check if the edge is lying on top-left side by checking if the position x of both beginning and end points are
higher or equals to position x in point with max y, and is less or equals than max x of the polygon; as well as if position
y of both beginning and end points of the edges are higher or equals to position y of the point with maximum x, as
well as if they're less or equals than max y. If yes then we add it to topLeft linked list.
            if(position.x.of.beginning.point.of.edge.with.position.i >= position.x.of.minMax[2] &&
position.x.of.beginning.point.of.edge.with.position.i <= position.x.of.minMax[0] &&
                position.y.of.beginning.point.of.edge.with.position.i >= position.y.of.minMax[0]
&& position.y.of.beginning.point.of.edge.with.position.i <= position.y.of.minMax[2] &&
                position.x.of.end.point.of.edge.with.position.i >= position.x.of.minMax[2] &&
position.x.of.end.point.of.edge.with.position.i <= position.x.of.minMax[0] &&
                position.y.of.end.point.of.edge.with.position.i >= position.y.of.minMax[0] &&
position.y.of.end.point.of.edge.with.position.i <= position.y.of.minMax[2]) then
            {
                topLeft.add.b;
            }fi;
//we check if the edge is lying on top-right side by checking if the position x of both beginning and end points are less or equals to position x in point with max y, and is higher or equals than min x of the polygon; as well as if position y of both beginning and end points of the edges are higher or equals to position y of the point with minimum x, as well as if they're less or equals than max y. If yes then we add it to topRight linked list.
                        if(position.x.of.beginning.point.of.edge.with.position.i <= position.x.of.minMax[2] &&
position.x.of.beginning.point.of.edge.with.position.i >= position.x.of.minMax[1] &&
                                position.y.of.beginning.point.of.edge.with.position.i >= position.y.of.minMax[1]
&& position.y.of.beginning.point.of.edge.with.position.i <= position.y.of.minMax[2] &&
                                position.x.of.end.point.of.edge.with.position.i <= position.x.of.minMax[2] &&
position.x.of.end.point.of.edge.with.position.i >= position.x.of.minMax[1] &&
                                position.y.of.end.point.of.edge.with.position.i >= position.y.of.minMax[1] &&
position.y.of.end.point.of.edge.with.position.i <= position.y.of.minMax[2]) then
                        {
                                topRight.add.b;
                        }fi;
//we check if the edge is lying on bottom-left side by checking if the position x of both beginning and end points are higher or equals to position x in point with min y, and is less or equals than max x of the polygon; as well as if position y of both beginning and end points of the edges are less or equals to position y of the point with maximum x, as well as if they're higher or equals than min y. If yes then we add it to bottomLeft linked list.
                        if(position.x.of.beginning.point.of.edge.with.position.i >= position.x.of.minMax[2] &&
position.x.of.beginning.point.of.edge.with.position.i <= position.x.of.minMax[0] &&
                                position.y.of.beginning.point.of.edge.with.position.i <= position.y.of.minMax[0]
&& position.y.of.beginning.point.of.edge.with.position.i >= position.y.of.minMax[3] &&
                                position.x.of.end.point.of.edge.with.position.i >= position.x.of.minMax[2]. &&
position.x.of.end.point.of.edge.with.position.i <= position.x.of.minMax[0] &&
                                position.y.of.end.point.of.edge.with.position.i <= position.y.of.minMax[0]. &&
position.y.of.end.point.of.edge.with.position.i >= position.y.of.minMax[3]) then
                        {
                                bottomLeft.add.b;
                        }fi;
//we check if the edge is lying on bottom-right side by checking if the position x of both beginning and end points are less or equals to position x in point with min y, and is higher or equals than min x of the polygon; as well as if position y of both beginning and end points of the edges are less or equals to position y of the point with minimum x, as well as if they're higher or equals than min y. If yes then we add it to bottomRight linked list.
                        if(position.x.of.beginning.point.of.edge.with.position.i <= position.x.of.minMax[3] &&
position.x.of.beginning.point.of.edge.with.position.i >= position.x.of.minMax[1] &&
                                position.y.of.beginning.point.of.edge.with.position.i <= position.y.of.minMax[1]
&& position.y.of.beginning.point.of.edge.with.position.i >= position.y.of.minMax[3] &&
                                position.x.of.end.point.of.edge.with.position.i <= position.x.of.minMax[3] &&
position.x.of.end.point.of.edge.with.position.i >= position.x.of.minMax[1] &&
                                position.y.of.end.point.of.edge.with.position.i <= position.y.of.minMax[1] &&
position.y.of.end.point.of.edge.with.position.i >= position.y.of.minMax[3]) then
                        {
                                bottomRight.add.b;
                        }fi;
                }od;
//we add the 4 side lists to the result list in the order: topLeft, topRight, bottomLeft, bottomRight and return it.
                result.add.topLeft;
                result.add.topRight;
                result.add.bottomLeft;
                result.add.bottomRight;
                return result;
        }
//we create to method to check whwther the given point is lying inside given polygon. The method take in a point and polygon and will return a boolean value indicate true or false.
        boolean ifInsidePolygon(Point point, Polygon polygon)
        {
//we create the boolean that we'll return later as the result and set it on default as true.
                boolean isInside = true;
//we get the array with points with maximum and minimum position x and y in the polygon.
                Point[] minMax = Intersection.getMaxAndMin(polygon);
//we get the 4-elements linked list with edges divided in 4 sides.
                LinkedList(LinkedList(Edge)) sides = getSides(polygon);
//we create 4 boolean value that indicate if the point is inside the given side of polygon and set them initially on false.
                boolean isTopleft = false;
                boolean isTopRight = false;
        boolean isBottomLeft = false;
        boolean isBottomRight = false;
//now we check in wich side of polygon the given point would be.
//If the point is lying on top-left side then its position x is higher or equals to position x in point with max y, and is
less than max x of the polygon; as well as if position y of the point is higher than position y of the point with maximum
x, as well as if they're less to max y. If yes then we set isTopLeft to true.
        if(position.x.of.point >= position.x.of.minMax[2] && position.x.of.point. <=
position.x.of.minMax[0] &&
            position.y.of.point >= position.y.of.minMax[0] && position.y.of.point <=
position.y.of.minMax[2]) then
        {
            isTopleft = true;
        } fi;
//If the point is lying on top-right side then its position x is less or equals to position x in point with max y, and is
higher than max x of the polygon; as well as if position y of the point is higher than position y of the point with
maximum x, as well as if they're less or eqlas to max y. If yes then we set isTopRight to true.
        if(position.x.of.point <= position.x.of.minMax[2] && position.x.of.point >=
position.x.of.minMax[1] &&
            position.y.of.point >= position.y.of.minMax[1] && position.y.of.point <=
position.y.of.minMax[2]) then
        {
            isTopRight = true;
        }fi;
//If the point is lying on bottom-left side then its position x is higher or equals to position x in point with max y, and is
less than max x of the polygon; as well as if position y of the point is less than position y of the point with maximum x,
as well as if they're higher to max y. If yes then we set isBottomLeft to true.
        if(position.x.of.point >= position.x.of.minMax[2] && position.x.of.point <=
position.x.of.minMax[0] &&
            position.y.of.point <= position.y.of.minMax[0] && position.y.of.point >=
position.y.of.minMax[3]) then
        {
            isBottomLeft = true;
        }fi;
//If the point is lying on bottom-right side then its position x is less or equals to position x in point with max y, and is
higher than max x of the polygon; as well as if position y of the point is less than position y of the point with maximum
x, as well as if they're higher to max y. If yes then we set isBottomRight to true.
        if(position.x.of.point <= position.x.of.minMax[3] && position.x.of.point >=
position.x.of.minMax[1] &&
            position.y.of.point <= position.y.of.minMax[1] && position.y.of.point >=
position.y.of.minMax[3]) then
        {
            isBottomRight = true;
        }fi;
//in the case the point is lying on top-left part of polygon
        if(isTopleft) then
        {
//we run through all edges lying on this side. The time complexity would be O(n) where n = size of polygon.
            for(int i = 0; i< size.of.position[0].in.sides; i++) do
            {
//we create variable equation e where we store the euation with osition i in the list with position 0 in linked list sides
                Equation e = equation.with.position.i.in.sides[0];
//we check if the edge is diagonal since in the case of horizontal and vertical edges we don't need to compare here(it
would lying on the line). The time complexity for the double if statements would be O(1)
                if(type.of.e == "ukos") then
                {
//we create 2 variable to store the calculated position of x of point on the line with equation e where the y position is
equals to y position of given point, and position y of point on the line with equation e where the x position is equals to
x position of the given point.
//x = (y - b)/a since y = ax+b
//y = ax+b
                    double x = (position.y.of.point - number.y.of.e)/ number.x.of.e;
                    double y = (position.x.of.point * number.x.of.e) + number.y.of.e;
//since we're in the top-left part of polygon, if the given point has the value x higher than calculated x or position y
higher than calculated y the it's laying outside of the polygon. we can return the result as false.
                    if(position.x.of.point > x || position.y.of.point > y) then
                    {
                        isInside = false;
                        return isInside;
                    }fi;
                }fi;
            }od;
        }di;
//in the case the point is lying on top-right part of polygon
        if(isTopRight) then
        {
//we run through all edges lying on this side. The time complexity would be O(n) where n = size of polygon.
            for(int i = 0; i< size.of.position[1].in.sides; i++) do
            {
//we create variable equation e where we store the euation with osition i in the list with position 1 in linked list sides
                Equation e = equation.with.position.i.in.sides[1];
//we check if the edge is diagonal since in the case of horizontal and vertical edges we don't need to compare here(it would lying on the line). The time complexity for the double if statements would be O(1)
                if(type.of.e == "ukos") then
                {
//we create 2 variable to store the calculated position of x of point on the line with equation e where the y position is equals to y position of given point, and position y of point on the line with equation e where the x position is equals to x position of the given point.
//x = (y - b)/a since y = ax+b
//y = ax+b
                    double x = (position.y.of.point - number.y.of.e)/ number.x.of.e;
                    double y = (position.x.of.point * number.x.of.e) + number.y.of.e;
//since we're in the top-right part of polygon, if the given point has the value x less than calculated x or position y higher than calculated y the it's laying outside of the polygon. we can return the result as false.
                    if(position.x.of.point < x || position.y.of.point > y) ten
                    {
                        isInside = false;
                        return isInside;
                    }fi;
                }fi;
            }od;
        }fi;
//in the case the point is lying on bottom-left part of polygon
        if(isBottomLeft)
        {
//we run through all edges lying on this side. The time complexity would be O(n) where n = size of polygon.
            for(int i = 0; i< size.of.position[2].in.sides; i++)
            {
//we create variable equation e where we store the euation with osition i in the list with position 2 in linked list sides
                Equation e = equation.with.position.i.in.sides[2];
//we check if the edge is diagonal since in the case of horizontal and vertical edges we don't need to compare here(it would lying on the line). The time complexity for the double if statements would be O(1)
                if(type.of.e == "ukos")
                {
//we create 2 variable to store the calculated position of x of point on the line with equation e where the y position is equals to y position of given point, and position y of point on the line with equation e where the x position is equals to x position of the given point.
//x = (y - b)/a since y = ax+b
//y = ax+b
                    double x = (position.y.of.point - number.y.of.e)/ number.x.of.e;
                    double y = (position.x.of.point * number.x.of.e) + number.y.of.e;
//since we're in the bottom-left part of polygon, if the given point has the value x higher than calculated x or position y less than calculated y the it's laying outside of the polygon. we can return the result as false.
                    if(position.x.of.point > x || position.y.of.point < y)
                    {
                        isInside = false;
                        return isInside;
                    }
                }
            }
        }
//in the case the point is lying on bottom-right part of polygon
        if(isBottomRight) then
        {
//we run through all edges lying on this side. The time complexity would be O(n) where n = size of polygon.
            for(int i = 0; i< size.of.position[3].in.sides; i++)do
            {
//we create variable equation e where we store the euation with osition i in the list with position 3 in linked list sides
                Equation e = equation.with.position.i.in.sides[3];
//we check if the edge is diagonal since in the case of horizontal and vertical edges we don't need to compare here(it
would lying on the line). The time complexity for the double if statements would be O(1)
                if(type.of.e == "ukos") then
                {
//we create 2 variable to store the calculated position of x of point on the line with equation e where the y position is
equals to y position of given point, and position y of point on the line with equation e where the x position is equals to
x position of the given point.
//x = (y - b)/a since y = ax+b
//y = ax+b
                    double x = (position.y.of.point - number.y.of.e)/ number.x.of.e;
                    double y = (position.x.of.point * number.x.of.e) + number.y.of.e;
//since we're in the bottom-right part of polygon, if the given point has the value x less than calculated x or position y
less than calculated y the it's laying outside of the polygon. we can return the result as false.
                    if(position.x.of.point < x || position.y.of.point < y) then
                    {
                        isInside = false;
                        return isInside;
                    }fi;
                }fi;
            }od;
        }fi;
//if the point is lying inside the polygon we return isInside.
        return isInside;
    }
//we now finally have method to calculate the intersection of 2 polygons. The input would be 2 polygons with their
sizes. The result would be linked list with point.
    LinedList(Point) getIntersection(Polygon first, number size1, Polygon second, number size2)
    {
//we create a linked list that later we'll return as result
        LinkedList(Point) result;
//we create the linked list that we will store intersected points.
        List(Intersect) theInterescted
//we get the points with max and min position of x and y for both polygons.
        Point[] minMax1 = getMaxAndMin(first);
        Point[] minMax2 = getMaxAndMin(second);
//we check whether if first polygon is not lying inside second by checking if both max and min value of x and y from 1
polygon is between max and min value of second polygon. If it's true then we print message informing about it and
add all the points from first polygon to result linked list. The expected time complexity would be O(n) where n =
size1
        if(value.y.from.minMax1[2] < value.y.from.minMax2[2] && value.y.from.minMax1[3] >
value.y.from.minMax2[3] && value.x.from.minMax1[0] < value.x.from.minMax2[0] && value.x.from.minMax1[1] >
value.x.from.minMax1[1]) then
        {
            print.message."One polygon is inside another";
            for(int i = 0; i < size1; i++) do
            {
                result.add.point.in.position.i.from.first;
            }od;
        }
//we check whether if second polygon is not lying inside first one by checking if both max and min value of x and y
from 1 polygon is between max and min value of second polygon. If it's true then we print message informing about it
and add all the points from second polygon to result linked list. The expected time complexity would be O(n) where n
= size1
        else if(value.y.from.minMax2[2] < value.y.from.minMax1[2] && value.y.from.minMax2[3]
>value.y.from.minMax1[3] && value.x.from.minMax2[0] < value.x.from.minMax1[0] && value.x.from.minMax2[1] >
value.x.from.minMax1[1]) then
        {
            print.message."One polygon is inside another";
            for(int i = 0; i < size2; i++) do
            {
                result.add.point.in.position.i.from.second;
            }od;
        }
//if min x in first polygon is higher than max x in second or inverse, or whethere min y in first polygon is higher than max y in second or inverse then the polygons are not intersected and we print message informing about it
        else if(value.x.from.minMax1[1] > value.x.from.minMax2[0] || value.x.from.minMax2[1] > value.x.from.minMax1[0] || value.y.from.minMax1[3] > value.y.from.minMax2[2] || value.y.from.minMax2[3] > value.y.from.minMax1[2]) then
        {
            print.message.("The polygons are not intersect");
        }
//in other situation we'll start to calculate the intersection of two given polygons.
        else then
        {
//we calculate all equations based on edges from both polygons.
            Equation[] equationFirst = getAllEquation(first);
            Equation[] equationSecond = getAllEquation(second);
//now we run though the equations of first polygon and try to calculate the interseted point with each edges of second polygon. The time complexity for this calculation would be O(nm) where n = size1 and m = size2
            for(int i = 0; i < size1; i++) do
            {
                for(int j = 0; j < size2; j++) do
                {
//we create 2 variable to store the position x and y for intersected point. We set them as default to 0.
                        double x = 0;
                        double y = 0;
//we check if the equation in first polygon is vertical
                        if(type.of.equation.in.position.i.from.first == "pion") then
                        {
//we check if the equation in second polygon is not vertical, because if it's then the lines will be parallel. If the equation from second polygon is not a vertical line then we set x to value x from equation of first polygon.
                            if(type.of.equation.in.position.j.from.second != "pion") then
                            {
                                x = position.x.from.equation.in.position.i.from.first;
//we check if the equation in second polygon is horizontal, if it's true then  we set y to value y from equation of second polygon.
                                if(type.of.equation.in.position.j.from.second == "poziom") then
                                {
                                    y = position.y.from.equation.in.position.j.from.second;
                                }
//in other case we calculate y = ax + b where a is value x and b = y from equation from second polygon
                                else then
                                {
                                    y = x * position.x.from.equation.in.position.j.from.second + position.y.from.equation.in.position.j.from.second;
                                }fi;
                            }fi;
                        }
//in the case equation in first polygon is horizontal
                        else if(type.of.equation.in.position.i.from.first == "poziom") then
                        {
//we check if the equation in second polygon is not horizontal, because if it's then the lines will be parallel. If the equation from second polygon is not a horizontal line then we set x to value x from equation of first polygon.
                            if(type.of.equation.in.position.j.from.second != "poziom") then
                            {
                                y = position.y.from.equation.in.position.i.from.first;
//we check if the equation in second polygon is vertical, if it's true then  we set x to value x from equation of second polygon.
                                if(type.of.equation.in.position.j.from.second == "pion") then
                                {
```

```
                                                        x =
position.x.from.equation.in.position.j.from.second;
                                                    }
//in other case we calculate x with formula: x = (y-b)/a where a and b are values x and y from equation of second
polygon
                                                    else then
                                                    {
                                                        x = (y -
position.y.from.equation.in.position.j.from.second)/position.x.from.equation.in.position.j.from.second;
                                                    }fi;
                                                }fi;
                                            }
//in the case equation in first polygon is diagonal
                                            else then
                                            {
//we check if the equation in second polygon is vertical. If the equation from second polygon is a vertical line then we
set x to value x from equation of second polygon and calculate y = ax+b where a and b are values x and y from
equation from first polagon.
                                                if(type.of.equation.in.position.j.from.second == "pion") then
                                                {
                                                    x =
position.x.from.equation.in.position.j.from.second;
                                                    y = position.x.from.equation.in.position.i.from.first
* x + position.y.from.equation.in.position.i.from.first;
                                                }
//we check if the equation in second polygon is horizontal. If the equation from second polygon is a horizontal line
then we set y to value y from equation of second polygon and calculate x = (y-b)/a where a and b are values x and y
from equation of first polygon
                                                else if(type.of.equation.in.position.j.from.second ==
"poziom") then
                                                {
                                                    y =
position.y.from.equation.in.position.j.from.second;
                                                    x = (y -
position.y.from.equation.in.position.i.from.first)/position.x.from.equation.in.position.i.from.first;
                                                }
//in the case both equations are diagonal lines
                                                else then
                                                {
//in case the value x from both equations are different (if they're the same then they're parallel) we calcuate x and y
with formula: x = (y2-y1)/ (x1-x2) and y = ax + b where a nd b are values x and y from equation from first polyfon.
                                                    if(position.x.from.equation.in.position.i.from.first
!= position.x.from.equation.in.position.j.from.second)
                                                    {
                                                        x =
(position.y.from.equation.in.position.j.from.second -
position.y.from.equation.in.position.i.from.first)/(position.x.from.equation.in.position.i.from.first -
position.x.from.equation.in.position.j.from.second);
                                                        y =
position.x.from.equation.in.position.i.from.first * x + position.y.from.equation.in.position.i.from.first;
                                                    }fi;
                                                }fi;
                                            }fi;
//we check whether our just calculate point with position x and y are lying on the edges of both polygons
                                            boolean answer = Intersection.checkIfLyingOnLines(equationFirst[i],
equationSecond[j], x, y);
//if the answer is yes then
                                            if(answer) then
                                            {
//we check now if our x is lying between min and max x of both polygons as well as if our position y is lying between
min and max y from both polygons.
                                                if(x <= position.x.of.minMax1[0] && x <=
position.x.of.minMax2[0] && x >= position.x.of.minMax1[1] && x >= position.x.of.minMax2[1] &&
                                                    y <= position.y.of.minMax1[2] && y <=
position.y.of.minMax2[2] && y >= position.y.of.minMax1[3] && y >= position.y.of.minMax1[3]) then
                                                {
```

```
//if the point is lying insde both polygons we create new point from position x and y
                                                        Point p = Point(x,y);
//we create new intersected point with the name of the linnes from both polygons and point p that is their
intersection point
                                                        Intersect n =
Intersect(name.of.equation.in.position.i.from.first, name.of.equation.in.position.j.from.second, p);
//we add new Intersect n to our listof interse ted points.
                                                        theInterescted.add.n;
                                                }fi;

                                        }fi;
                                }od;
                        }od;
//now we check, if there is no intersected point between 2 polygons then they're not intersected and we print
message informing about it.
                        if(size.of.theInterescted == 0) then
                        {
                                print.message."The polygons are not intersect";
                        }
//in the case there are intersected points
                        else then
                        {
//we create 2 Linked List where we'll store the points from one polygon that is lying inside second polygon.
//we also create the linked list of points that should be include in the intersection
                                List(Point) firstInConsideration;
                                List(Point) secondInConsideration;
                                List(Point) toBeAdded;
//we run thought the all intersected points and add them to our result list. The time complexity should be
O((n+m)^2) where n = size1 and m = size2
                                for(int i = 0; i < size.of.theInterescted; i++)
                                {
//we create the boolean to check whetherthe point we want to add to result are already inside. We set it on default as
false.
                                        boolean isDuplicate = false;
//we run through the result list
                                        for(int j = 0; j < size.of.result; j++) do
                                        {
//we check if the position x and y from the point from theInterescted and result are the same then we set isDuplicate
to trueand break out the inner loop
                                                if(position.x.from.point.in.position.i.from.theInterescted ==
position.x.from.point.in.position.j.from.result && position.y.from.point.in.position.i.from.theInterescted ==
position.y.from.point.in.position.j.from.result ) then
                                                {
                                                        isDuplicate = true;
                                                        break;
                                                }fi;
                                        }od;
//we check if the point is not already in result, if not we add it to result.
                                        if(!isDuplicate) then
                                        {
                                                result.add.point.in.position.i.from.theInterescted;
                                        }fi;
                                }od;
//we run though the points of first polygon and check if their x position lying between min and max x of both
polygons and if their y is lying between min and max y of both polygons. If yes then we add them to linked list
firstInConsideration. The time complexity of this calculation should be O(n) where n = size1
                                for(int i = 0; i < size1; i++) do
                                {
                                        if(position.x.of.point.in.i.position.from.first <=
position.x.of.minMax1[0] && position.x.of.point.in.i.position.from.first <= position.x.of.minMax2[0] &&
position.x.of.point.in.i.position.from.first >= position.x.of.minMax1[1] && position.x.of.point.in.i.position.from.first
>= position.x.of.minMax2[1] &&
                                                position.y.of.point.in.i.position.from.first <=
position.y.of.minMax1[2] && position.y.of.point.in.i.position.from.first <= position.y.of.minMax2[2] &&
position.y.of.point.in.i.position.from.first >= position.y.of.minMax1[3] && position.y.of.point.in.i.position.from.first
>= position.y.of.minMax2[3]) then
```

```
                    {
                        firstInConsideration.add.point.in.i.position.from.first;
                    }fi;
                }od;
//we run though the points of second polygon and check if their x position lying between min and max x of both
polygons and if their y is lying between min and max y of both polygons. If yes then we add them to linked list
secondInConsideration. The time complexity of this calculation should be O(m) where m = size2
                for(int j = 0; j < size2; j++) do
                {
                    if(position.x.of.point.in.j.position.from.second <=
position.x.of.minMax1[0] && position.x.of.point.in.j.position.from.second <= position.x.of.minMax2[0] &&
position.x.of.point.in.j.position.from.second >= position.x.of.minMax1[1] &&
position.x.of.point.in.j.position.from.second >= position.x.of.minMax2[1] &&
                        position.y.of.point.in.j.position.from.second <=
position.y.of.minMax1[2] && position.y.of.point.in.j.position.from.second <= position.y.of.minMax2[2] &&
position.y.of.point.in.j.position.from.second >= position.y.of.minMax1[3] &&
position.y.of.point.in.j.position.from.second >= position.y.of.minMax2[3]) then
                    {
                        secondInConsideration.add.point.in.j.position.from.second;
                    }fi;
                }od;
//now we run thought the list of firstInConsideration and check if the points are really lyinginsde second polygons. if
yes the we add them to toBeAdded. The timecomplexity would be O(n^2)
                for(int i = 0; i < size.of.firstInConsideration; i++) do
                {
                    if(ifInsidePolygon(point.in.i.position.from.firstInConsideration,
second)) then
                    {
                        toBeAdded.add.point.in.i.position.from.firstInConsideration;
                    }fi;
                }od;
//now we run thought the list of secondInConsideration and check if the points are really lyinginsde first polygons. if
yes the we add them to toBeAdded. The timecomplexity would be O(m^2)
                for(int j = 0; j < size.of.secondInConsideration; j++)
                {

    if(ifInsidePolygon(ifInsidePolygon(point.in.j.position.from.secondInConsideration, first))
                    {

    toBeAdded.add.point.in.j.position.from.secondInConsideration;
                    }
                }
//now we run through all points in toBeAdded list to add them to result list. The time complexity would be O(2(n^2)
* (n+m)) = O((n^2) * (n+m))
                for(int i = 0; i < size.of.toBeAdded; i++)
                {
//we create a boolean variable to check if the points is not already been in the result list. We set it intially to false.
                    boolean isDuplicate = false;
//we run though result list and check if the x and y position from the point from toBeAdded match any point that is
already in result then we set isDuplicate to true.
                    for(int j = 0; j < size.of.result; j++)
                    {
                        if(position.x.of.point.in.position.i.from.toBeAdded ==
position.x.of.point.in.position.j.from.result && position.y.of.point.in.position.i.from.toBeAdded ==
position.y.of.point.in.position.j.from.result)
                        {
                            isDuplicate = true;
                        }
                    }
//if the point from toBeAdded is not already in result list, we add it in.
                    if(!isDuplicate)
                    {
                        result.add.point.in.position.i.from.toBeAdded;
                    }
                }
            }
```

```
                        }
//we return the result.
                        return result;
            }
}
```

Justification of an algorithm's correctness:

The specification of the algorithm getIntersection becomes a pair <WP, WK>, where WP = (The two convex polygons that have finite number of points and edges) and WK = (the intersection of 2 polygons).

<u>Stop condition of the algorithm</u> – for getIntersection algorithm we have use few other algorithms, which every single of them are running the iteration from 0 to the size of polygons (n or m) or the sum of the size of polygons (n + m). Since both n and m are finite number then algorithm getIntersection and all algorithm that it uses will stops.

<u>Partial correctness of the algorithm</u> – for getIntersection algorithm for the initial condition WP = ((The two convex polygons that have finite number of points and edges) and WK = (the intersection of 2 polygons).
At the beginning, we would find all the intersected points between edges of two polygons. We do it by first calculating the equation from each edges from both figures. From there, when we take any two equation, one from one polygon, second from other we can find the intersection point between them (it they're not parallel or the same lines).
When we already find all intersected points, we need to determine which of them we can add in to our result list (list of the points that create the intersection area of 2 polygons) since some of them can lie completely outside the area of our polygons. Here we need to determine if the intersection point between 2 equations(that had been calculate from 2 edges) is actually lying on both of those edges. We also need to eliminate the duplicate value since many equations can intersect in the same point.
After finding all intersection points between edges. We now have to find the points that are completely lying inside the polygons.
To be able to determine it, we first will create the list of points that are candidate to our result list. Those that are lying between minimal and maximal value of coordinate x and coordinate y from both polygons.
From here we need to determine, in which part of second polygon the point from first polygon is lying. This is necessary, since from this we can determine the position between the point and the edges of another polygon to check if it's really lying inside the second polygons.
After finding the list of points that are lying completely inside both polygons, we check for duplicated points and add them to our result list.
The result of our algorithm will be the list of points that create the intersection area of 2 polygons
Because the getIntersection algorithm is partially correct in terms of specification <WP,WK> and we have shown the stop condition for this algorithm, it is also absolutely correct with respect to the considered specification.

Implementation of algorithm in Java:

```
import java.text.DecimalFormat;
import java.util.*;

class Point
{
            public char name;
            public double x;
            public double y;
            public Point(char name, double x, double y)
            {
                        this.name = name;
                        this.x = x;
                        this.y = y;
            }
            public Point(double x, double y)
            {
                        this.x = x;
                        this.y = y;
            }
            public char getName()
            {
                        return name;
            }
            public double getX()
            {
                        return x;
```

```
		}
		public double getY()
		{
			return y;
		}
		public String toString()
		{
			DecimalFormat df = new DecimalFormat("#.00");
			return df.format(x) + " " + df.format(y);
		}
}
class Edge {
		public String name;
    public Point begin;
    public Point end;
    public Edge(Point begin, Point end){
		this.name = new StringBuilder().append(begin.getName()).append(end.getName()).toString();
        this.begin = begin;
        this.end = end;
    }
    public String getEdgeName()
    {
		return name;
    }
    public Point getBegin(){
        return begin;
    }
    public Point getEnd(){
        return end;
    }
    public String toString()
    {
		return name;
    }
}
class Polygon {
    public Point[] points;
    public Edge[] edges;
    public Polygon(Point[] points, Edge[] edges){
        this.points = points;
        this.edges = edges;
    }
    public Point[] getPoints(){
        return points;
    }
    public Edge[] getEdges(){
        return edges;
    }
}
class Equation{
		Point begin;
		Point end;
		String name;
		double x;
		double y;
		String type;
		public Equation(Edge edge)
		{
			this.begin = edge.getBegin();
			this.end = edge.getEnd();
			this.name = edge.getEdgeName();
			if(edge.getBegin().getX() == edge.getEnd().getX())
			{
				this.x = edge.getBegin().getX();
				this.y = 0;
				this.type = "pion";
			}
			else if(edge.getBegin().getY() == edge.getEnd().getY())
			{
				this.x = 0;
				this.y = edge.getBegin().getY();
				this.type = "poziom";
			}
			else
```

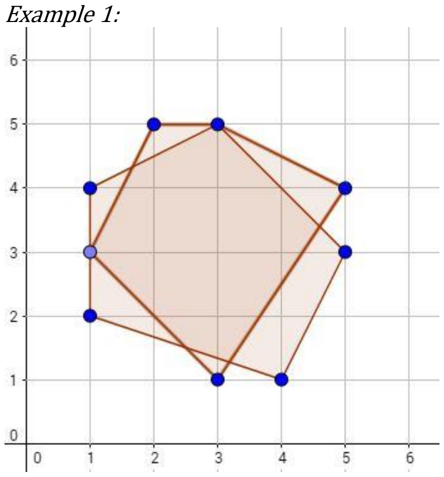
```
				{
					this.x = (edge.getEnd().getY()-edge.getBegin().getY())/(edge.getEnd().getX()-
edge.getBegin().getX());
					this.y = -((edge.getEnd().getY()-edge.getBegin().getY())/(edge.getEnd().getX()-
edge.getBegin().getX()))*edge.getBegin().getX()+edge.getBegin().getY();
					this.type = "ukos";
				}
		}
		public Point getEquationBegin()
		{
				return begin;
		}
		public Point getEquationEnd()
		{
				return end;
		}
		public String getEquationName()
		{
				return name;
		}
		public double getEquationX()
		{
				return x;
		}
		public double getEquationY()
		{
				return y;
		}
		public String getEquationType()
		{
		return type;
		}

		public String toString()
		{
				DecimalFormat df = new DecimalFormat("#.00");
				return name + " " + df.format(x) + " " + df.format(y) + " " + type;
		}

}
class Intersect
{
		public String firstLine;
		public String secondLine;
		Point point;
		public Intersect(String firstLine, String secondLine, Point point)
		{
				this.firstLine = firstLine;
				this.secondLine = secondLine;
				this.point = point;
		}
		public String getIntersectFLine()
		{
				return firstLine;
		}
		public String getIntersectSLine()
		{
				return secondLine;
		}
		public Point getIntersectPoint()
		{
				return point;
		}
		public String toString()
		{
				return firstLine + " " + secondLine + " " + point.getX() + " " + point.getY();
		}
}
public class Intersection {
		public static Equation[] getAllEquation(Polygon a)
		{
				Equation[] result = new Equation[a.getEdges().length];
				for(int i = 0; i< a.getEdges().length; i++)
				{
```

```
				result[i] = new Equation(a.getEdges()[i]);
			}
			return result;
		}

		public static boolean checkIfLyingOnLines(Equation first, Equation second, double x, double y)
		{
			boolean isBelongTo = false;
			double first_y = first.getEquationBegin().getY();
			double second_y = first.getEquationEnd().getY();
			double first_x = first.getEquationBegin().getX();
			double second_x = first.getEquationEnd().getX();

			double third_y = second.getEquationBegin().getY();
			double fourth_y = second.getEquationEnd().getY();
			double third_x = second.getEquationBegin().getX();
			double fourth_x = second.getEquationEnd().getX();
			if((((y >= first_y && y <= second_y) || (y <= first_y && y >= second_y)) && ((x >= first_x && x <= second_x)
|| (x <= first_x && x >= second_x))) &&
					(((y >= third_y && y <= fourth_y) || (y <= third_y && y >= fourth_y)) && ((x >= third_x
&& x <= fourth_x) || (x <= third_x && x >= fourth_x)))		)
			{
				isBelongTo = true;
			}

			return isBelongTo;
		}
		public static Point[] getMaxAndMin(Polygon polygon)
		{
//			MaxX, MinX, MaxY, MinY
			Point[] result = {polygon.getPoints()[0], polygon.getPoints()[0], polygon.getPoints()[0], polygon.getPoints()[0]};
			for(int i = 0; i < polygon.getPoints().length; i++)
			{
				if(result[0].getX() < polygon.getPoints()[i].getX())
				{
					result[0] = polygon.getPoints()[i];
				}
				if(result[1].getX() > polygon.getPoints()[i].getX())
				{
					result[1] = polygon.getPoints()[i];
				}
				if(result[2].getY() < polygon.getPoints()[i].getY())
				{
					result[2] = polygon.getPoints()[i];
				}
				if(result[3].getY() > polygon.getPoints()[i].getY())
				{
					result[3] = polygon.getPoints()[i];
				}
			}
			return result;
		}
		public static List<List<Edge>> getSides(Polygon a)
		{
//			4 elements list of list: TopLeft, TopRight, BottomLeft, BottomRight
			List<List<Edge>> result = new ArrayList<>();
			List<Edge> topLeft = new ArrayList<>();
			List<Edge> topRight = new ArrayList<>();
			List<Edge> bottomLeft = new ArrayList<>();
			List<Edge> bottomRight = new ArrayList<>();
			Point[] minMax = Intersection.getMaxAndMin(a);
			for(Edge b : a.getEdges())
			{
				if(b.getBegin().getX() >= minMax[2].getX() && b.getBegin().getX() <= minMax[0].getX() &&
						b.getBegin().getY() >= minMax[0].getY() && b.getBegin().getY() <=
minMax[2].getY() &&
						b.getEnd().getX() >= minMax[2].getX() && b.getEnd().getX() <=
minMax[0].getX() &&
						b.getEnd().getY() >= minMax[0].getY() && b.getEnd().getY() <=
minMax[2].getY())
				{
					topLeft.add(b);
				}
				if(b.getBegin().getX() <= minMax[2].getX() && b.getBegin().getX() >= minMax[1].getX() &&
                    b.getBegin().getY() >= minMax[1].getY() && b.getBegin().getY() <=
minMax[2].getY() &&
                    b.getEnd().getX() <= minMax[2].getX() && b.getEnd().getX() >=
minMax[1].getX() &&
                    b.getEnd().getY() >= minMax[1].getY() && b.getEnd().getY() <=
minMax[2].getY() )
            {
                topRight.add(b);
            }
            if(b.getBegin().getX() >= minMax[2].getX() && b.getBegin().getX() <= minMax[0].getX() &&
                    b.getBegin().getY() <= minMax[0].getY() && b.getBegin().getY() >=
minMax[3].getY() &&
                    b.getEnd().getX() >= minMax[2].getX() && b.getEnd().getX() <=
minMax[0].getX() &&
                    b.getEnd().getY() <= minMax[0].getY() && b.getEnd().getY() >=
minMax[3].getY())
            {
                bottomLeft.add(b);
            }
            if(b.getBegin().getX() <= minMax[3].getX() && b.getBegin().getX() >= minMax[1].getX() &&
                    b.getBegin().getY() <= minMax[1].getY() && b.getBegin().getY() >=
minMax[3].getY() &&
                    b.getEnd().getX() <= minMax[3].getX() && b.getEnd().getX() >=
minMax[1].getX() &&
                    b.getEnd().getY() <= minMax[1].getY() && b.getEnd().getY() >=
minMax[3].getY())
            {
                bottomRight.add(b);
            }
        }
        result.add(topLeft);
        result.add(topRight);
        result.add(bottomLeft);
        result.add(bottomRight);
        return result;
    }
    public static boolean ifInsidePolygon(Point point, Polygon polygon)
    {
        boolean isInside = true;
        Point[] minMax = Intersection.getMaxAndMin(polygon);
        List<List<Edge>> sides = Intersection.getSides(polygon);
        boolean isTopleft = false;
        boolean isTopRight = false;
        boolean isBottomLeft = false;
        boolean isBottomRight = false;
        if(point.getX() >= minMax[2].getX() && point.getX() <= minMax[0].getX() &&
                point.getY() >= minMax[0].getY() && point.getY() <= minMax[2].getY())
        {
            isTopleft = true;
        }
        if(point.getX() <= minMax[2].getX() && point.getX() >= minMax[1].getX() &&
                point.getY() >= minMax[1].getY() && point.getY() <= minMax[2].getY())
        {
            isTopRight = true;
        }
        if(point.getX() >= minMax[2].getX() && point.getX() <= minMax[0].getX() &&
                point.getY() <= minMax[0].getY() && point.getY() >= minMax[3].getY())
        {
            isBottomLeft = true;
        }
        if(point.getX() <= minMax[3].getX() && point.getX() >= minMax[1].getX() &&
                point.getY() <= minMax[1].getY() && point.getY() >= minMax[3].getY())
        {
            isBottomRight = true;
        }
        if(isTopleft)
        {
            for(int i = 0; i< sides.get(0).size(); i++)
            {
                Equation e = new Equation(sides.get(0).get(i));
                if(e.getEquationType() == "ukos")
                {
                    double x = (point.getY() - e.getEquationY())/ e.getEquationX();
                    double y = (point.getX() * e.getEquationX()) + e.getEquationY();
```

```
                                                if(point.getX() > x || point.getY() > y)
                                                {
                                                        isInside = false;
                                                        return isInside;
                                                }
                                        }
                                }
                        }
                        if(isTopRight)
                        {
                                for(int i = 0; i< sides.get(1).size(); i++)
                                {
                                        Equation e = new Equation(sides.get(1).get(i));
                                        if(e.getEquationType() == "ukos")
                                        {
                                                double x = (point.getY() - e.getEquationY())/ e.getEquationX();
                                                double y = (point.getX() * e.getEquationX()) + e.getEquationY();
                                                if(point.getX() < x || point.getY() > y)
                                                {
                                                        isInside = false;
                                                        return isInside;
                                                }
                                        }
                                }
                        }
                        if(isBottomLeft)
                        {
                                for(int i = 0; i< sides.get(2).size(); i++)
                                {
                                        Equation e = new Equation(sides.get(2).get(i));
                                        if(e.getEquationType() == "ukos")
                                        {
                                                double x = (point.getY() - e.getEquationY())/ e.getEquationX();
                                                double y = (point.getX() * e.getEquationX()) + e.getEquationY();
                                                if(point.getX() > x || point.getY() < y)
                                                {
                                                        isInside = false;
                                                        return isInside;
                                                }
                                        }
                                }
                        }
                        if(isBottomRight)
                        {
                                for(int i = 0; i< sides.get(3).size(); i++)
                                {
                                        Equation e = new Equation(sides.get(3).get(i));
                                        if(e.getEquationType() == "ukos")
                                        {
                                                double x = (point.getY() - e.getEquationY())/ e.getEquationX();
                                                double y = (point.getX() * e.getEquationX()) + e.getEquationY();
                                                if(point.getX() < x || point.getY() < y)
                                                {
                                                        isInside = false;
                                                        return isInside;
                                                }
                                        }
                                }
                        }
                        return isInside;
                }
                public static List<Point> getIntersection(Polygon first, Polygon second)
                {
                        List<Point> result = new ArrayList<>();
                        List<Intersect> theInterescted = new ArrayList<>();
                        Point[] minMax1 = Intersection.getMaxAndMin(first);
                        Point[] minMax2 = Intersection.getMaxAndMin(second);
                        if(minMax1[2].getY() < minMax2[2].getY() && minMax1[3].getY() > minMax2[3].getY() && minMax1[0].getX()
        < minMax2[0].getX() && minMax1[1].getX() > minMax1[1].getX())
                        {
                                System.out.println("One polygon is inside another");
                                for(int i = 0; i < first.getPoints().length; i++)
                                {
                                        result.add(first.getPoints()[i]);
                                        }
                              }
                              else if(minMax2[2].getY() < minMax1[2].getY() && minMax2[3].getY() > minMax1[3].getY() &&
minMax2[0].getX() < minMax1[0].getX() && minMax2[1].getX() > minMax1[1].getX())
                              {
                                        System.out.println("One polygon is inside another");
                                        for(int i = 0; i < second.getPoints().length; i++)
                                        {
                                                  result.add(second.getPoints()[i]);
                                        }
                              }
                              else if(minMax1[1].getX() > minMax2[0].getX() || minMax2[1].getX() > minMax1[0].getX() ||
minMax1[3].getY() > minMax2[2].getY() || minMax2[3].getY() > minMax1[2].getY())
                              {
                                        System.out.println("The polygons are not intersect");
                              }
                              else
                              {
                                        Equation[] equationFirst = Intersection.getAllEquation(first);
                                        Equation[] equationSecond = Intersection.getAllEquation(second);
                                        for(int i = 0; i < equationFirst.length; i++)
                                        {

                                                  for(int j = 0; j < equationSecond.length; j++)
                                                  {
                                                            double x = 0;
                                                            double y = 0;
                                                            if(equationFirst[i].getEquationType() == "pion")
                                                            {
                                                                      if(equationSecond[j].getEquationType() != "pion")
                                                                      {
                                                                                x = equationFirst[i].getEquationX();
                                                                                if(equationSecond[j].getEquationType() == "poziom")
                                                                                {

                                                                                          y = equationSecond[j].getEquationY();
                                                                                }
                                                                                else
                                                                                {
                                                                                          y =
x*equationSecond[j].getEquationX()+equationSecond[j].getEquationY();
                                                                                }
                                                                      }
                                                            }
                                                            else if(equationFirst[i].getEquationType() == "poziom")
                                                            {
                                                                      if(equationSecond[j].getEquationType() != "poziom")
                                                                      {
                                                                                y = equationFirst[i].getEquationY();
                                                                                if(equationSecond[j].getEquationType() == "pion")
                                                                                {
                                                                                          x = equationSecond[j].getEquationX();
                                                                                }
                                                                                else
                                                                                {
                                                                                          x = (y-
equationSecond[j].getEquationY())/equationSecond[j].getEquationX();

                                                                                }
                                                                      }
                                                            }
                                                            else
                                                            {
                                                                      if(equationSecond[j].getEquationType() == "pion")
                                                                      {
                                                                                x = equationSecond[j].getEquationX();
                                                                                y = equationFirst[i].getEquationX() * x +
equationFirst[i].getEquationY();
                                                                      }
                                                                      else if(equationSecond[j].getEquationType() == "poziom")
                                                                      {
                                                                                y = equationSecond[j].getEquationY();
                                                                                x = (y-
equationFirst[i].getEquationY())/equationFirst[i].getEquationX();
```

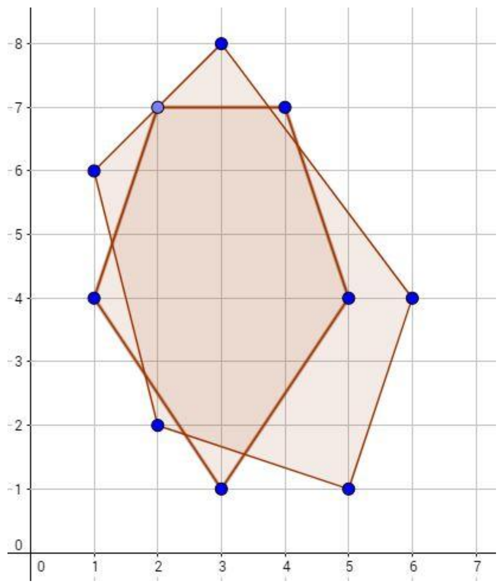
```
                        }
                        else
                        {
                            if(equationFirst[i].getEquationX() !=
equationSecond[j].getEquationX())
                            {
                                x = (equationSecond[j].getEquationY()-
equationFirst[i].getEquationY())/(equationFirst[i].getEquationX()-equationSecond[j].getEquationX());
                                y = equationFirst[i].getEquationX() * x +
equationFirst[i].getEquationY();
                            }
                        }
                    }
                    boolean answer = Intersection.checkIfLyingOnLines(equationFirst[i],
equationSecond[j], x, y);
                    if(answer)
                    {
                        if(x <= minMax1[0].getX() && x <= minMax2[0].getX() && x >=
minMax1[1].getX() && x >= minMax2[1].getX() &&
                            y <= minMax1[2].getY() && y <= minMax2[2].getY() &&
y >= minMax1[3].getY() && y >= minMax1[3].getY())
                        {
                            theInterescted.add(new
Intersect(equationFirst[i].getEquationName(), equationSecond[j].getEquationName(), new Point(x, y)));
                        }

                    }
                }
            }
            if(theInterescted.size() == 0)
            {
                System.out.println("The polygons are not intersect");
            }
            else
            {
                List<Point> firstInConsideration = new ArrayList<>();
                List<Point> secondInConsideration = new ArrayList<>();
                List<Point> toBeAdded = new ArrayList<>();
                for(Intersect k : theInterescted)
                {
                    boolean isDuplicate = false;
                    for(Point l : result)
                    {
                        if(k.getIntersectPoint().getX() == l.getX() &&
k.getIntersectPoint().getY() == l.getY())
                        {
                            isDuplicate = true;
                            break;
                        }
                    }
                    if(!isDuplicate)
                    {
                        result.add(k.getIntersectPoint());
                    }
                }
                for(Point k : first.getPoints())
                {
                    if(k.getX() <= minMax1[0].getX() && k.getX() <= minMax2[0].getX() &&
k.getX() >= minMax1[1].getX() && k.getX() >= minMax2[1].getX() &&
                            k.getY() <= minMax1[2].getY() && k.getY() <=
minMax2[2].getY() && k.getY() >= minMax1[3].getY() && k.getY() >= minMax2[3].getY())
                    {
                        firstInConsideration.add(k);
                    }
                }
                for(Point k : second.getPoints())
                {
                    if(k.getX() <= minMax1[0].getX() && k.getX() <= minMax2[0].getX() &&
k.getX() >= minMax1[1].getX() && k.getX() >= minMax2[1].getX() &&
                            k.getY() <= minMax1[2].getY() && k.getY() <=
minMax2[2].getY() && k.getY() >= minMax1[3].getY() && k.getY() >= minMax2[3].getY())
                    {
                        secondInConsideration.add(k);
                    }
                }
                for(Point k : firstInConsideration)
                {
                    if(Intersection.ifInsidePolygon(k, second))
                    {
                        toBeAdded.add(k);
                    }
                }
                for(Point k : secondInConsideration)
                {
                    if(Intersection.ifInsidePolygon(k, first))
                    {
                        toBeAdded.add(k);
                    }
                }
                for(Point k : toBeAdded)
                {
                    boolean isDuplicate = false;
                    for(Point l : result)
                    {
                        if(k.getX() == l.getX() && k.getY() == l.getY())
                        {
                            isDuplicate = true;
                        }
                    }
                    if(!isDuplicate)
                    {
                        result.add(k);
                    }
                }
            }
        }
        return result;
    }
```

Example 1:



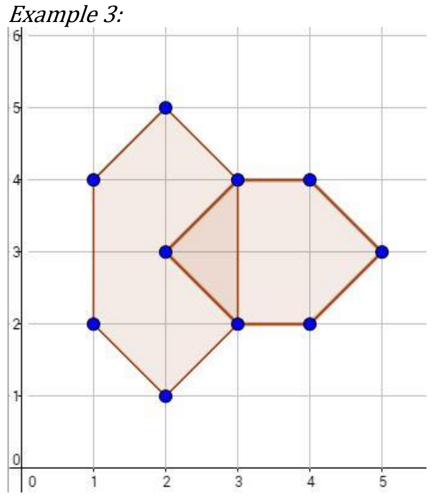
The polygon 1 points are: {Point('A', 1, 2), Point('B', 1, 4), Point('C', 3, 5), Point('D', 5, 3), Point('E', 4, 1)}
The polygon 1 edges are: {Edge(A,B), Edge(B,C), Edge(C,D), Edge(D,E), Edge(E,A)}
The polygon 1 points are: {Point('F', 1, 3), Point('G', 2, 5), Point('C', 3, 5), Point('H', 5, 4), Point('I', 3, 1)}
The polygon 1 edges are: {Edge(F,G), Edge(G,C), Edge(C,H), Edge(H,I), Edge(I,F)}
The result will be list of points: { Point(1.00, 3.00), Point(1.67, 4.33), Point(3.00, 5.00), Point(4.60, 3.40), Point(3.18, 1.27), Point(2.50, 1.50)}

Example 2:



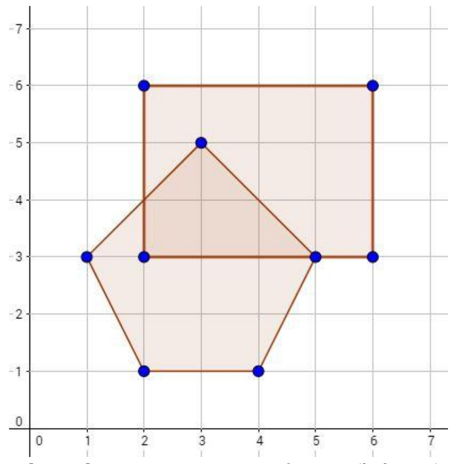
The polygon 1 points are: { Point('A', 2, 2), Point('B', 1, 6), Point('C', 3, 8), Point('D', 6, 4), Point('E', 5, 1)}
The polygon 1 edges are: { Point('F', 3, 1), Point('G', 1, 4), Point('H', 2, 7), Point('I', 4, 7), Point('J', 5, 4)}
The polygon 1 points are: { Edge(A,B), Edge(B,C), Edge(C,D), Edge(D,E), Edge(E,A)}
The polygon 1 edges are: { Edge(F,G), Edge(G,H), Edge(H,I), Edge(I,J), Edge(J,F)}
The result will be list: { Point(1.80, 2.80), Point(1.29, 4.86), Point(2.00, 7.00), Point(3.75, 7.00), Point(4.20, 6.40), Point(2.43, 1.86), Point(3.36, 1.55), Point(5.00, 4.00)}

Example 3:



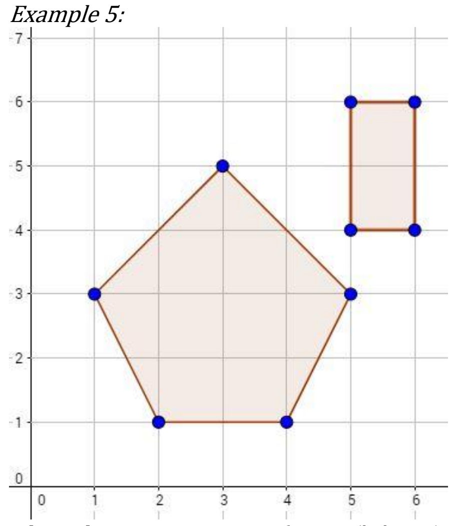
The polygon 1 points are: { Point('A', 1, 4), Point('B', 2, 5), Point('C', 3, 4), Point('D', 3, 2), Point('E', 2, 1), Point('F', 1, 2)}
The polygon 1 edges are: { Edge(A,B), Edge(B,C), Edge(C,D), Edge(D,E), Edge(E,F), Edge(F,A)}
The polygon 1 points are: { Point('G', 2, 3), Point('C', 3, 4), Point('H', 4, 4), Point('I', 5, 3), Point('J', 4, 2), Point('D', 3, 2)}
The polygon 1 edges are: { Edge(G,C), Edge(C,H), Edge(H,I), Edge(I,J), Edge(J,D), Edge(D,G)}
The result will be 3 elements list: { Point(3.00, 4.00), Point(3.00, 2.00), Point(2.00, 3.00)}

Example 4:



The polygon 1 points are: { Point('A', 1, 3), Point('B', 3, 5), Point('C', 5, 3), Point('D', 4, 1), Point('E', 2, 1)}
The polygon 1 edges are: { Edge(A,B), Edge(B,C), Edge(C,D), Edge(D,E), Edge(E,A)}
The polygon 1 points are: { Point('F', 2, 3), Point('G', 2, 6), Point('H', 6, 6), Point('I', 6, 3)}
The polygon 1 edges are: { Edge(F,G), Edge(G,H), Edge(H,I), Edge(I,F)}
The result will be list: { Point(2.00, 4.00), Point(5.00, 3.00), Point(3.00, 5.00), Point(2.00, 3.00)}

Example 5:



The polygon 1 points are: { Point('A', 1, 3), Point('B', 3, 5), Point('C', 5, 3), Point('D', 4, 1), Point('E', 2, 1)}
The polygon 1 edges are: { Edge(A,B), Edge(B,C), Edge(C,D), Edge(D,E), Edge(E,A)}
The polygon 1 points are: { Point('F', 5, 4), Point('G', 5, 6), Point('H', 6, 6), Point('I', 6, 4)}
The polygon 1 edges are: { Edge(F,G), Edge(G,H), Edge(H,I), Edge(I,F)}
The result will be a message: The polygons are not intersect

Estimation of algorithm's complexity:

Time complexity – time complexity for calculation 1 equation O(1)
time complexity for calculation all equations for both polygon with size n and m: O(n) + O(m)
time complexity for checking if 1 intersection lying on the 2 lines: O(1)
time complexity for getting the point ith maximum and minimum x and y: O(n) + O(m)
time complexity for dividing edges into 4 sides: O(n) + O(m)
time complexity for checking if the point is inside polygon: O(n)+O(n)+O(n)+O(n)
*time complexity for calculation for getIntersection: O(n) || O(m) || (O(nm)+O(n) + O(m) + O((n+m)^2) + O(n) + O(m) + O(n^2)´+ O(m^2) + O((n^2) * (n+m)))*
*Since n^2 run faster than (n+m) then the time complexity for the algorithm would be: O((n^2) * (n+m)) = O(n^3 + mn^2)*

Space complexity – the getInterscetion algorithm only uses many auxiliary variables that are independently of the value of the arguments and and m, however in the same time it use many linked list that the size of the lists are depend on the arguments na and m, the longer list that we could expected would be the list of possible intersection poins and the possible points that lying inside both figures and the cost of those lists would be T(n+m) so the space complexity for the algorithm would be O(n+m)