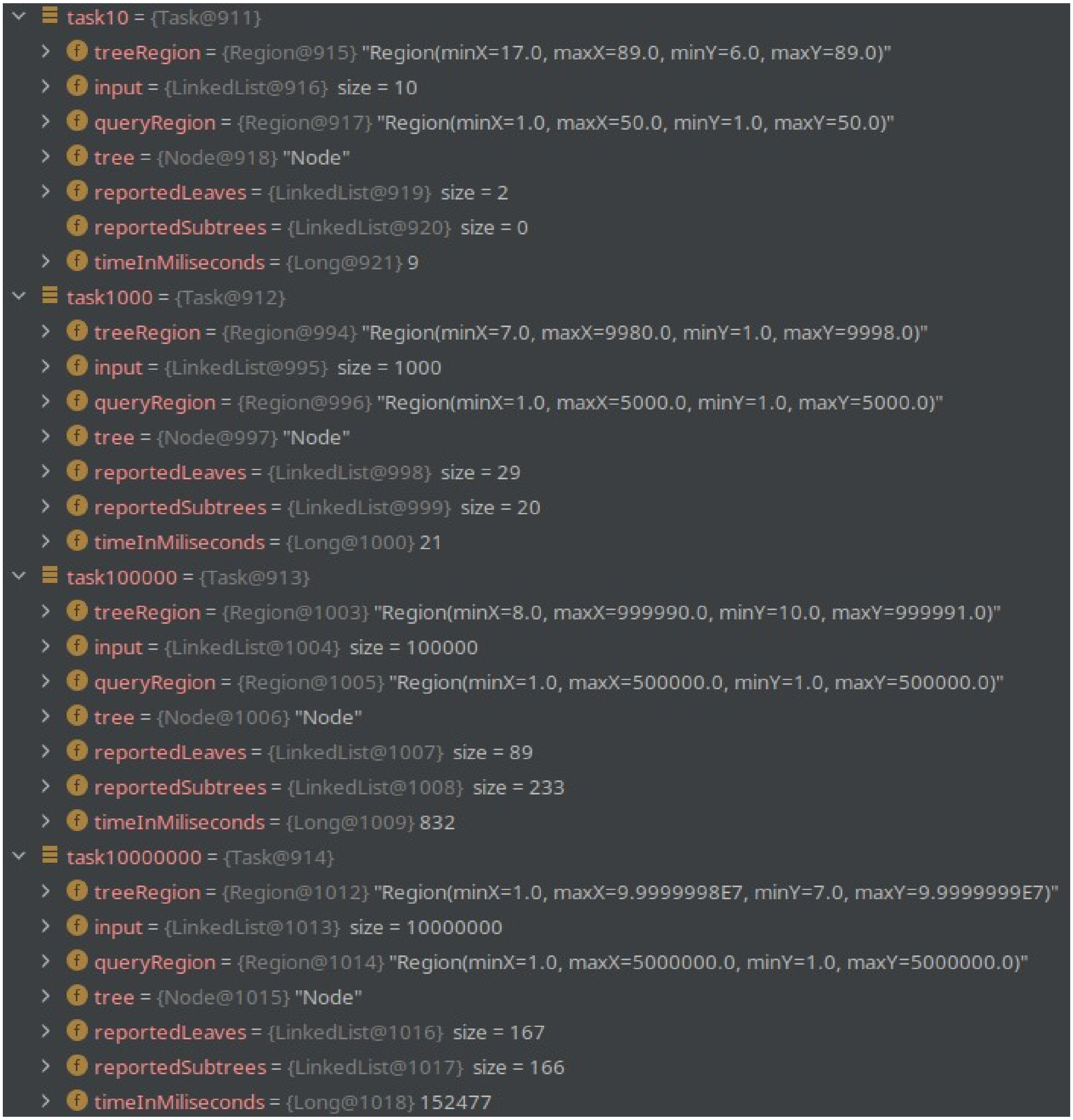
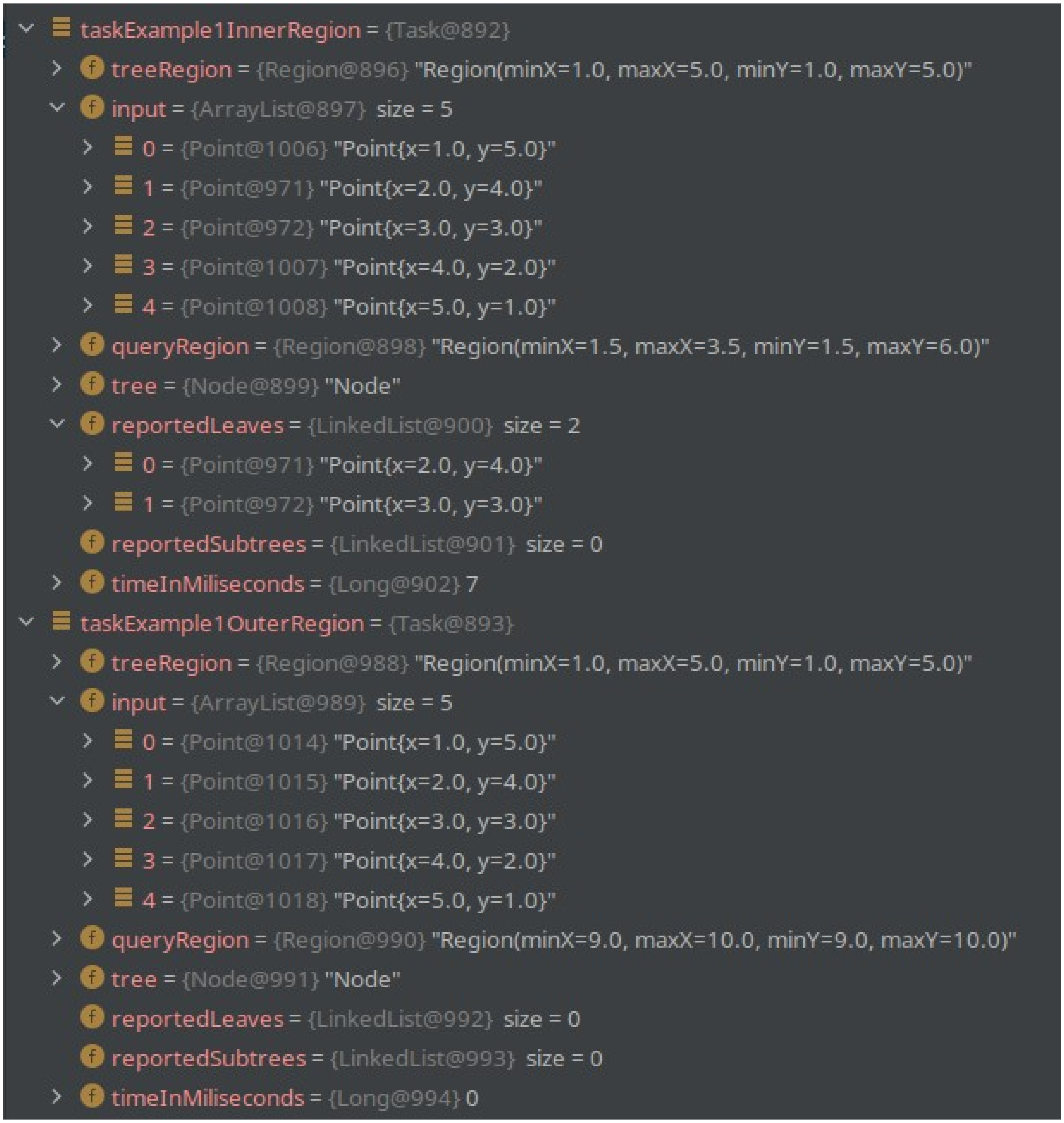
Błażej Piekoś 273210

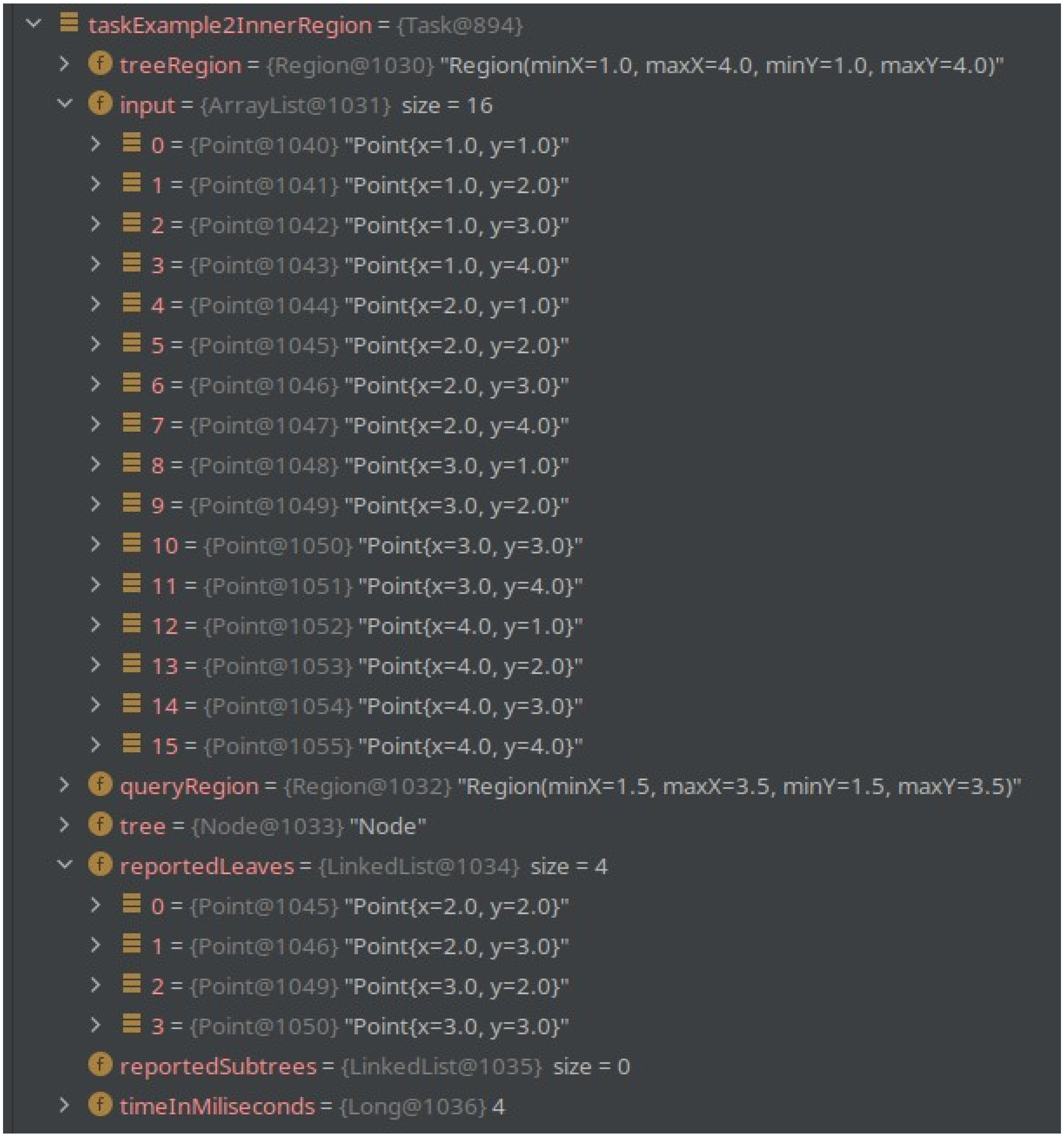
Wyniki:



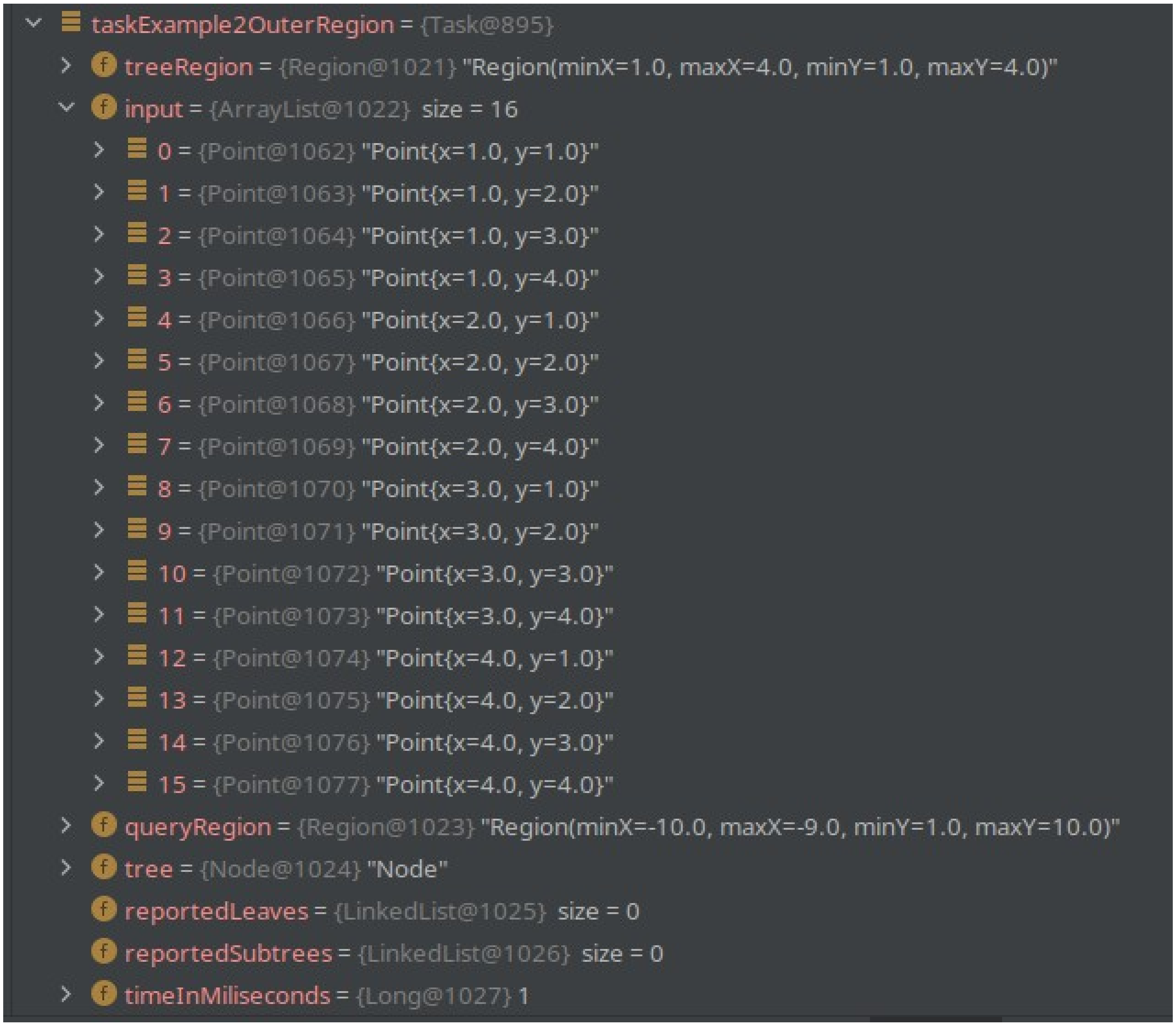
```
task10 = {Task@911}
    treeRegion = {Region@915} "Region(minX=17.0, maxX=89.0, minY=6.0, maxY=89.0)"
    input = {LinkedList@916}  size = 10
    queryRegion = {Region@917} "Region(minX=1.0, maxX=50.0, minY=1.0, maxY=50.0)"
    tree = {Node@918} "Node"
    reportedLeaves = {LinkedList@919}  size = 2
    reportedSubtrees = {LinkedList@920}  size = 0
    timeInMiliseconds = {Long@921} 9
task1000 = {Task@912}
    treeRegion = {Region@994} "Region(minX=7.0, maxX=9980.0, minY=1.0, maxY=9998.0)"
    input = {LinkedList@995}  size = 1000
    queryRegion = {Region@996} "Region(minX=1.0, maxX=5000.0, minY=1.0, maxY=5000.0)"
    tree = {Node@997} "Node"
    reportedLeaves = {LinkedList@998}  size = 29
    reportedSubtrees = {LinkedList@999}  size = 20
    timeInMiliseconds = {Long@1000} 21
task100000 = {Task@913}
    treeRegion = {Region@1003} "Region(minX=8.0, maxX=999990.0, minY=10.0, maxY=999991.0)"
    input = {LinkedList@1004}  size = 100000
    queryRegion = {Region@1005} "Region(minX=1.0, maxX=500000.0, minY=1.0, maxY=500000.0)"
    tree = {Node@1006} "Node"
    reportedLeaves = {LinkedList@1007}  size = 89
    reportedSubtrees = {LinkedList@1008}  size = 233
    timeInMiliseconds = {Long@1009} 832
task10000000 = {Task@914}
    treeRegion = {Region@1012} "Region(minX=1.0, maxX=9.9999998E7, minY=7.0, maxY=9.9999999E7)"
    input = {LinkedList@1013}  size = 10000000
    queryRegion = {Region@1014} "Region(minX=1.0, maxX=5000000.0, minY=1.0, maxY=5000000.0)"
    tree = {Node@1015} "Node"
    reportedLeaves = {LinkedList@1016}  size = 167
    reportedSubtrees = {LinkedList@1017}  size = 166
    timeInMiliseconds = {Long@1018} 152477
```



```
v taskExample1InnerRegion = {Task@892}
    > treeRegion = {Region@896} "Region(minX=1.0, maxX=5.0, minY=1.0, maxY=5.0)"
    v input = {ArrayList@897}  size = 5
        > 0 = {Point@1006} "Point{x=1.0, y=5.0}"
        > 1 = {Point@971} "Point{x=2.0, y=4.0}"
        > 2 = {Point@972} "Point{x=3.0, y=3.0}"
        > 3 = {Point@1007} "Point{x=4.0, y=2.0}"
        > 4 = {Point@1008} "Point{x=5.0, y=1.0}"
    > queryRegion = {Region@898} "Region(minX=1.5, maxX=3.5, minY=1.5, maxY=6.0)"
    > tree = {Node@899} "Node"
    v reportedLeaves = {LinkedList@900}  size = 2
        > 0 = {Point@971} "Point{x=2.0, y=4.0}"
        > 1 = {Point@972} "Point{x=3.0, y=3.0}"
      reportedSubtrees = {LinkedList@901}  size = 0
    > timeInMiliseconds = {Long@902} 7
v taskExample1OuterRegion = {Task@893}
    > treeRegion = {Region@988} "Region(minX=1.0, maxX=5.0, minY=1.0, maxY=5.0)"
    v input = {ArrayList@989}  size = 5
        > 0 = {Point@1014} "Point{x=1.0, y=5.0}"
        > 1 = {Point@1015} "Point{x=2.0, y=4.0}"
        > 2 = {Point@1016} "Point{x=3.0, y=3.0}"
        > 3 = {Point@1017} "Point{x=4.0, y=2.0}"
        > 4 = {Point@1018} "Point{x=5.0, y=1.0}"
    > queryRegion = {Region@990} "Region(minX=9.0, maxX=10.0, minY=9.0, maxY=10.0)"
    > tree = {Node@991} "Node"
      reportedLeaves = {LinkedList@992}  size = 0
      reportedSubtrees = {LinkedList@993}  size = 0
    > timeInMiliseconds = {Long@994} 0
```



```
v taskExample2InnerRegion = {Task@894}
  > treeRegion = {Region@1030} "Region(minX=1.0, maxX=4.0, minY=1.0, maxY=4.0)"
  v input = {ArrayList@1031}  size = 16
    > 0 = {Point@1040} "Point{x=1.0, y=1.0}"
    > 1 = {Point@1041} "Point{x=1.0, y=2.0}"
    > 2 = {Point@1042} "Point{x=1.0, y=3.0}"
    > 3 = {Point@1043} "Point{x=1.0, y=4.0}"
    > 4 = {Point@1044} "Point{x=2.0, y=1.0}"
    > 5 = {Point@1045} "Point{x=2.0, y=2.0}"
    > 6 = {Point@1046} "Point{x=2.0, y=3.0}"
    > 7 = {Point@1047} "Point{x=2.0, y=4.0}"
    > 8 = {Point@1048} "Point{x=3.0, y=1.0}"
    > 9 = {Point@1049} "Point{x=3.0, y=2.0}"
    > 10 = {Point@1050} "Point{x=3.0, y=3.0}"
    > 11 = {Point@1051} "Point{x=3.0, y=4.0}"
    > 12 = {Point@1052} "Point{x=4.0, y=1.0}"
    > 13 = {Point@1053} "Point{x=4.0, y=2.0}"
    > 14 = {Point@1054} "Point{x=4.0, y=3.0}"
    > 15 = {Point@1055} "Point{x=4.0, y=4.0}"
  > queryRegion = {Region@1032} "Region(minX=1.5, maxX=3.5, minY=1.5, maxY=3.5)"
  > tree = {Node@1033} "Node"
  v reportedLeaves = {LinkedList@1034}  size = 4
    > 0 = {Point@1045} "Point{x=2.0, y=2.0}"
    > 1 = {Point@1046} "Point{x=2.0, y=3.0}"
    > 2 = {Point@1049} "Point{x=3.0, y=2.0}"
    > 3 = {Point@1050} "Point{x=3.0, y=3.0}"
    reportedSubtrees = {LinkedList@1035}  size = 0
  > timeInMiliseconds = {Long@1036} 4
```



```
taskExample2OuterRegion = {Task@895}
  treeRegion = {Region@1021} "Region(minX=1.0, maxX=4.0, minY=1.0, maxY=4.0)"
  input = {ArrayList@1022}  size = 16
    0 = {Point@1062} "Point{x=1.0, y=1.0}"
    1 = {Point@1063} "Point{x=1.0, y=2.0}"
    2 = {Point@1064} "Point{x=1.0, y=3.0}"
    3 = {Point@1065} "Point{x=1.0, y=4.0}"
    4 = {Point@1066} "Point{x=2.0, y=1.0}"
    5 = {Point@1067} "Point{x=2.0, y=2.0}"
    6 = {Point@1068} "Point{x=2.0, y=3.0}"
    7 = {Point@1069} "Point{x=2.0, y=4.0}"
    8 = {Point@1070} "Point{x=3.0, y=1.0}"
    9 = {Point@1071} "Point{x=3.0, y=2.0}"
    10 = {Point@1072} "Point{x=3.0, y=3.0}"
    11 = {Point@1073} "Point{x=3.0, y=4.0}"
    12 = {Point@1074} "Point{x=4.0, y=1.0}"
    13 = {Point@1075} "Point{x=4.0, y=2.0}"
    14 = {Point@1076} "Point{x=4.0, y=3.0}"
    15 = {Point@1077} "Point{x=4.0, y=4.0}"
  queryRegion = {Region@1023} "Region(minX=-10.0, maxX=-9.0, minY=1.0, maxY=10.0)"
  tree = {Node@1024} "Node"
  reportedLeaves = {LinkedList@1025}  size = 0
  reportedSubtrees = {LinkedList@1026}  size = 0
  timeInMiliseconds = {Long@1027} 1
```

Kod:

```java
package model;

import com.google.common.collect.Lists;
import java.util.List;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class Input {

  private List<Point> points = Lists.newLinkedList();
  private Region region = new Region();

}

package model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class Node {

  private Double location;
  private NODE_TYPE type;
  private Point point;
  private Node parent;
  private Node left;
  private Node right;
  private Region region;

  public Node(
      Double location,
      NODE_TYPE type,
      Point point,
      Node left,
      Node right
  ) {
    this.location = location;
    this.type = type;
    this.point = point;
    this.left = left;
    this.right = right;
  }

  public boolean isLeaf() {
    return this.left == null && this.right == null;
  }

  public void setRegion(Region region) {
    this.region = new Region();
    this.region.of(region);
  }

  public boolean isLeft() {
    return this == this.parent.left;
  }

  public boolean isRight() {
    return this == this.parent.right;
  }

  @Override
  public String toString() {
    return "Node";
  }

  public enum NODE_TYPE {
    LEAF,
    VERTICAL_LINE,
    HORIZONTAL_LINE;
  }

}

package model;

import model.Node.NODE_TYPE;

public class NodeBuilder {

  /**
   * Stworzenie node'a na podstawie argumentów.
   *
   * @param location
   * @param type
   * @param point
   * @param left
   * @param right
   */
  public Node build(
      Double location,
      NODE_TYPE type,
      Point point,
      Node left,
      Node right
  ) {

    Node node = new Node(
        location,
        type,
        point,
        left,
        right
    );
    if (left != null) {
      left.setParent(node);
      node.setLeft(left);
    }
    if (right != null) {
      right.setParent(node);
      node.setRight(right);
    }
    return node;
  }

}

package model;

import java.util.Objects;
import java.util.UUID;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class Point {

  private UUID id = UUID.randomUUID();

  private double x;

  private double y;

  public Point(
      Double x,
      Double y
  ) {
    this.x = x;
    this.y = y;
  }

  @Override
  public boolean equals(Object o) {

    if (this == o) {
      return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }
    Point point = (Point) o;
    return Double.compare(point.x, x) == 0 && Double.compare(point.y, y) == 0;
  }

  @Override
  public int hashCode() {

    return Objects.hash(x, y);
  }

}

package model;

import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@Data
public class Region {

  private Double minX;
  private Double maxX;
  private Double minY;
  private Double maxY;

  /**
   * Stworzenie rejonu z prawodilowy przypisaniem wartosci minimalnych/maksymlanych
   *
   * @param minX
   * @param maxX
   * @param minY
   * @param maxY
   */
  public Region(
      Double minX,
      Double maxX,
      Double minY,
      Double maxY
  ) {

    double temporary;
    if (minX > maxX) {
      temporary = minX;
      minX = maxX;
      maxX = temporary;
    }
    if (minY > maxY) {
      temporary = minY;
      minY = maxY;
      maxY = temporary;
    }
    this.minX = minX;
    this.maxX = maxX;
    this.minY = minY;
    this.maxY = maxY;
  }

  /**
   * Skopiowanie wartosci z danego rejonu do instacji
   *
   * @param region
   */
  public void of(Region region) {

    this.minX = region.getMinX();
    this.maxX = region.getMaxX();
    this.minY = region.getMinY();
    this.maxY = region.getMaxY();
  }

}

package model;

import com.google.common.collect.Lists;
import java.util.Collections;
import java.util.List;
import model.Node.NODE_TYPE;
import service.RegionComparator;

public class Task {
```

```
public Region treeRegion = new Region(
    0.0,
    0.0,
    0.0,
    0.0
);
private List<Point> input = Lists.newArrayList();
private Region queryRegion;
private Node tree;
private List<Point> reportedLeaves = Lists.newLinkedList();
private List<Node> reportedSubtrees = Lists.newLinkedList();
private Long timeInMiliseconds = 0L;

public Task(
    Input input,
    Region queryRegion
) {
  this.input = input.getPoints();
  this.treeRegion = input.getRegion();
  this.queryRegion = queryRegion;
  this.run();
}

/**
 * Metoda fasadowa. Uruchamia poszczegolne etapy algorytmu.
 */
public void run() {

  try {
    long start = System.currentTimeMillis();
    this.tree = this.buildKdTree(
        this.input,
        0
    );
    this.tree.setRegion(this.treeRegion);
    this.defineRegion(this.tree);
    this.checkAffiliationToQueryRegion(this.tree);
    this.timeInMiliseconds = System.currentTimeMillis() - start;
  } catch (Exception exception) {
    System.out.println(exception.getMessage());
  }
}

/**
 * Walidacja obszaru zapytania
 *
 * @throws Exception
 */
public void validateQueryRegion() throws Exception {
  if (this.queryRegion != null) {
    if (this.queryRegion.getMinX() > this.queryRegion.getMaxX()
        || this.queryRegion.getMinY() > this.queryRegion.getMaxY()) {
      throw new Exception("Nieprawidłowy obszar zapytania");
    }
  }
}

/**
 * Walidacja danych wejsciowych
 *
 * @throws Exception
 */
public void validateAmountOfPoints() throws Exception {

  if (this.input.size() == 0) {
    throw new Exception("Nie podano punktów.");
  }
}

/**
 * Walidacja danych wejsciowych
 *
 * @throws Exception
 */
public void validateDifferenceOfX(
    Point left,
    Point right
) throws Exception {

  if (left.getX() == right.getX()) {
    throw new Exception("Współrzędne x są takie same w punktach");
  }
}

/**
 * Walidacja danych wejsciowych
 *
 * @throws Exception
 */
public void validateDifferenceOfY(
    Point left,
    Point right
) throws Exception {

  if (left.getY() == right.getY()) {
    throw new Exception("Współrzędne y są takie same w punktach");
  }
```

```
}

/**
 * Budowanie kd drzewa na podstawie punktow z argumentu metody. Złozoność O(n).
 *
 * @param points
 * @param d
 */
public Node buildKdTree(
  List<Point> points,
  Integer d
) {

 if (points.size() == 1) {
  return new NodeBuilder().build(
    0d,
    NODE_TYPE.LEAF,
    points.get(0),
    null,
    null
  );
 } else if (points.size() == 0) {
  return null;
 }
 Integer axis = d % 2;
 NODE_TYPE type = axis == 0 ? NODE_TYPE.VERTICAL_LINE : NODE_TYPE.HORIZONTAL_LINE;
 double median = this.calculateMedian(
   axis,
   points
 );
 List<Point> leftPoints = Lists.newLinkedList();
 List<Point> rightPoints = Lists.newLinkedList();
 for (Point temporary : points) {
  double value = axis == 0 ? temporary.getX() : temporary.getY();
  if (value <= median) {
   leftPoints.add(temporary);
  } else {
   rightPoints.add(temporary);
  }
 }
 Node leftSubtree = this.buildKdTree(
   leftPoints,
   d + 1
 );
 Node rightSubtree = this.buildKdTree(
   rightPoints,
   d + 1
 );
 Node newNode = new NodeBuilder().build(
      median,
      type,
      null,
      leftSubtree,
      rightSubtree
  );

  return newNode;
}

/**
 * Liczenie mediany z punktow wzgledem osi x lub y z argumentu
 *
 * @param axis
 * @param points
 */
public Double calculateMedian(
    Integer axis,
    List<Point> points
) {

  Double median = axis == 0 ? this.calculateMedianByX(points) : this.calculateMedianByY(points);

  return median;
}

/**
 * Liczenie mediany punktow wzgledem osi x
 *
 * @param points
 */
public Double calculateMedianByX(
    List<Point> points
) {

  this.sortByX(points);
  Integer half = points.size() / 2;
  Double median = (points.size() % 2 == 0) ?
      ((points.get(half).getX() + points.get(half - 1).getX()) / 2)
      : points.get(half).getX();

  return median;
}

/**
 * Liczenie mediany punktow wzgledem osi y
 *
 * @param points
 */
```

```java
public Double calculateMedianByY(
    List<Point> points
) {

  this.sortByY(points);
  Integer half = points.size() / 2;
  Double median = (points.size() % 2 == 0) ?
      ((points.get(half).getY() + points.get(half - 1).getY()) / 2)
      : points.get(half).getY();

  return median;
}

/**
 * Sortowanie po osi x
 *
 * @param points
 */
public void sortByX(List<Point> points) {

  points.sort((p1, p2) -> {
    if ((p1.getX() < p2.getX()) || (p1.getX() == p2.getX() && p1.getY() < p2.getY())) {
      return -1;
    } else if (p1.getX() > p2.getX()) {
      return 1;
    } else {
      return 0;
    }
  });
}

/**
 * Sortowanie po osi y
 *
 * @param points
 */
public void sortByY(List<Point> points) {

  Collections.sort(points, (p1, p2) -> {
    if ((p1.getY() < p2.getY()) || (p1.getY() == p2.getY() && p1.getX() < p2.getX())) {
      return -1;
    } else if (p1.getY() > p2.getY()) {
      return 1;
    } else {
      return 0;
    }
  });
}
```

```
/**
 * Zdefiniowanie rejonu punktow node'a z argumentu metody. Rekurencyjnie sprawdza rejony dla
 * dzieci.
 *
 * @param node
 */
public void defineRegion(Node node) {

  if (!node.isLeaf()) {
    if (node.getParent() != null) {
      node.setRegion(
          node.getParent().getRegion()
      );
      Region nodeRegion = node.getRegion();
      Double location = node.getParent().getLocation();
      if (nodeRegion != null && location != null) {
        if (NODE_TYPE.HORIZONTAL_LINE.equals(node.getType())) {
          if (node.isLeft()) {
            nodeRegion.setMaxX(location);
          } else {
            nodeRegion.setMinX(location);
          }
        } else {
          if (node.isLeft()) {
            nodeRegion.setMaxY(location);
          } else {
            nodeRegion.setMinY(location);
          }
        }
        node.setRegion(nodeRegion);
      }
    }
    if (node.getLeft() != null) {
      this.defineRegion(
          node.getLeft()
      );
    }
    if (node.getRight() != null) {
      this.defineRegion(
          node.getRight()
      );
    }
  }
}

/**
 * Sprawdza czy node znajduje sie w obszarze zapytania i dodaje go do zgloszonych elementow
 *
 * @param node
 */
public void checkAffiliationToQueryRegion(Node node) {

  if (node.isLeaf()) {
    this.checkLeafAffiliationToQueryRegion(node);
  } else {
    if (node.getLeft().isLeaf()) {
      this.checkLeafAffiliationToQueryRegion(node.getLeft());
    } else {
      this.checkSubtreeAffiliationToQueryRegion(node.getLeft());
    }
    if (node.getRight().isLeaf()) {
      this.checkLeafAffiliationToQueryRegion(node.getRight());
    } else {
      this.checkSubtreeAffiliationToQueryRegion(node.getRight());
    }
  }
}

/**
 * Sprawdza czy lisc przynalezy do obszaru zapytania
 *
 * @param node
 */
public void checkLeafAffiliationToQueryRegion(Node node) {

  Boolean contained = this.containedInRegion(
      this.queryRegion,
      node.getPoint()
  );
  if (contained) {
    this.reportedLeaves.add(
        node.getPoint()
    );
  }
}

/**
 * Sprawdza czy poddrzewo znajduje sie w obszarze zapytania
 *
 * @param node
 */
public void checkSubtreeAffiliationToQueryRegion(Node node) {

  if (
      node != null
          && this.queryRegion != null
          && new RegionComparator(
          node.getRegion(),
```

```java
            this.queryRegion).isNodeRegionContainedInQueryRegion()
    ) {
      this.reportedSubtrees.add(node);
    } else if (
        node != null
            && this.queryRegion != null
            && new RegionComparator(
            node.getRegion(),
            this.queryRegion).isNodeRegionIntersectionOfQueryRegion()
    ) {
      this.checkAffiliationToQueryRegion(node);
    } else {
      if (node.isLeaf()) {
        this.checkLeafAffiliationToQueryRegion(node);
      }
    }
  }

  /**
   * Sprawdza czy punkt z argumentu znajduje sie danym rejonie z argumentu
   *
   * @param region
   * @param point
   */
  public Boolean containedInRegion(
      Region region,
      Point point
  ) {

    Boolean pointInRegion = point.getX() >= region.getMinX() && point.getX() <= region.getMaxX()
        && point.getY() >= region.getMinY() && point.getY() <= region.getMaxY();

    return pointInRegion;
  }

}

package service;

import com.google.common.collect.Lists;
import com.google.common.collect.Sets;
import java.util.List;
import java.util.Random;
import java.util.Set;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import model.Input;
import model.Point;
import model.Region;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class InputBuilder {

  private Integer amount;

  /**
   * Tworzy liste z losowymi wartosciami oraz rejon w obrebie ktorego znajduja sie te wartosci
   */
  public Input build() {

    List<Point> points = Lists.newLinkedList();
    List<Double> xPoints = this.buildListOfRandomDouble();
    List<Double> yPoints = this.buildListOfRandomDouble();
    Double minX = xPoints.get(0);
    Double maxX = xPoints.get(0);
    Double minY = yPoints.get(0);
    Double maxY = yPoints.get(0);
    for (int i = 0; i < this.amount; i++) {
      Point point = new Point(
          xPoints.get(i),
          yPoints.get(i)
      );
      points.add(point);
      if (point.getX() < minX) {
        minX = point.getX();
      }
      if (point.getX() > maxX) {
        maxX = point.getX();
      }
      if (point.getY() < minY) {
        minY = point.getY();
      }
      if (point.getY() > maxY) {
        maxY = point.getY();
      }
    }
    Region region = new Region(
        minX,
        maxX,
        minY,
        maxY
    );
    Input input = new Input(
        points,
        region
```

```java
    );

    return input;
  }

  /**
   * Tworzy liste z losowymi wartosciami
   */
  public List<Double> buildListOfRandomDouble() {

    Set<Double> doubles = Sets.newLinkedHashSet();
    double randomDouble;
    int randomInt;
    Random random = new Random();
    while (doubles.size() != this.amount) {
      randomInt = random.nextInt(this.amount * 10);
      randomInt = randomInt < 0 ? randomInt * -1 : randomInt;
      randomDouble = randomInt;
      doubles.add(randomDouble);
    }
    List<Double> result = doubles.stream().toList();

    return result;
  }

}

package service;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import model.Point;
import model.Region;

@AllArgsConstructor
@NoArgsConstructor
@Data
public class RegionComparator {

  private Point topRightQueryPoint;
  private Point bottomLeftQueryPoint;
  private Point topRightNodePoint;
  private Point bottomLeftNodePoint;

  public RegionComparator(
      Region node,
      Region query
  ) {
    this.topRightQueryPoint = new Point(
      query.getMaxX(),
      query.getMaxY()
    );
    this.bottomLeftQueryPoint = new Point(
      query.getMinX(),
      query.getMinY()
    );
    this.topRightNodePoint = new Point(
      node.getMaxX(),
      node.getMaxY()
    );
    this.bottomLeftNodePoint = new Point(
      node.getMinX(),
      node.getMinY()
    );
  }

  /**
   * Sprawdza czy node znajduje sie w obszarze zapytania
   */
  public Boolean isNodeRegionContainedInQueryRegion() {

    boolean condition1 = this.topRightNodePoint.getX() <= this.topRightQueryPoint.getX();
    boolean condition2 = this.topRightNodePoint.getY() <= this.topRightQueryPoint.getY();
    boolean condition3 = this.bottomLeftNodePoint.getX() >= this.bottomLeftQueryPoint.getX();
    boolean condition4 = this.bottomLeftNodePoint.getY() >= this.bottomLeftQueryPoint.getY();
    boolean result = condition1 && condition2 && condition3 && condition4;

    return result;
  }

  /**
   * Sprawdza czy node przecina obszar z zapytania
   */
  public Boolean isNodeRegionIntersectionOfQueryRegion() {

    boolean condition1 = this.topRightQueryPoint.getX() < this.bottomLeftNodePoint.getX();
    boolean condition2 = this.bottomLeftQueryPoint.getX() > this.topRightNodePoint.getX();
    boolean condition3 = this.topRightQueryPoint.getY() < this.bottomLeftNodePoint.getY();
    boolean condition4 = this.bottomLeftQueryPoint.getY() > this.topRightNodePoint.getY();
    boolean result = condition1 || condition2 || condition3 || condition4 ? false : true;

    return result;
  }

}
```