TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P3

PDF erstellt am: 5. April 2022

1 Vorbereitung

Um diese Programmieraufgabe erfolgreich durchführen zu können, müssen folgende Schritte umgesetzt werden:

1. Laden Sie das Framework `P3.zip` aus TUWEL herunter.
2. Entpacken Sie `P3.zip` und öffnen Sie das entstehende Verzeichnis als Projekt in IntelliJ (nicht importieren, sondern öffnen).
3. Öffnen Sie die nachfolgend angeführte Datei im Projekt in IntelliJ. In dieser Datei sind sämtliche Programmiertätigkeiten durchzuführen. Ändern Sie keine anderen Dateien im Framework und fügen Sie auch keine neuen hinzu.
`src/main/java/exercise/StudentSolutionImplementation.java`
4. Füllen Sie Vorname, Nachname und Matrikelnummer in der Methode `StudentInformation provideStudentInformation()` aus.

2 Hinweise

Einige Hinweise, die Sie während der Umsetzung dieser Aufgabe beachten müssen:

- Lösen Sie die Aufgaben selbst und nutzen Sie keine Bibliotheken, die diese Aufgaben abnehmen.
- Sie dürfen beliebig viele Hilfsmethoden schreiben und benutzen. Beachten Sie aber, dass Sie nur die oben geöffnete Datei abgeben und diese Datei mit dem zur Verfügung gestellten Framework lauffähig sein muss.

3 Übersicht

In dieser Programmieraufgaben implementieren Sie einen Divide-and-Conquer-Algorithmus für das Problem des Auffindens eines dichtesten Punktpaares: Gegeben ist eine Menge von Punkten im zweidimensionalen Raum. Ziel ist es, ein Punktpaar mit der kleinsten Distanz zu bestimmen. Implizit wenden Sie dabei auch Quicksort an. Dabei vergleichen wir einige Möglichkeiten für die Auswahl des Pivotelements und studieren die Auswirkungen auf das Laufzeitverhalten.

4 Theorie

Die notwendige Theorie für Quicksort und die Bestimmung eines dichtesten Punktpaares mittels Divide-and-Conquer befindet sich in den Vorlesungsfolien „Divide-and-Conquer“. Lesen Sie sich die entsprechenden Abschnitte durch, bevor Sie mit der Aufgabe fortfahren.

5 Implementierung

Ehe in Abschnitt 5.6 der eigentliche Divide-and-Conquer-Algorithmus umgesetzt wird, werden Sie in den Abschnitten 5.1 bis 5.5 die dazu notwendigen Hilfsfunktionen implementieren.

Als Hilfestellung werden die Klassen `Point` und `PointPair` zur Verfügung gestellt, die im Folgenden kurz vorgestellt werden. Die Klasse `Point` verwaltet Punkte. Folgende Methoden, die Sie verwenden dürfen, stellt sie bereit:

- `double getX()`: Gibt die x-Koordinate eines Punktes zurück. Mittels `double x = p.getX()` kann zum Beispiel die x-Koordinate des Punktes `Point p` erfragt werden.
- `double getY()`: Gibt die y-Koordinate eines Punktes zurück. Mittels `double y = p.getY()` kann zum Beispiel die y-Koordinate des Punktes `Point p` erfragt werden.
- `double getDistance(Point p)`: Berechnet die euklidische Distanz zweier Punkte. Mittels `double d = p1.getDistance(p2)` kann zum Beispiel die euklidische Distanz zwischen den beiden `Point`-Objekten `p1` und `p2` berechnet werden.

Die Klasse `PointPair` verwaltet Punktpaare inklusive der Distanz des Punktpaares zueinander. Neue Punktpaare können Sie mit Hilfe der folgenden beiden **Konstruktoren** (= Methoden zur Erzeugung von Objekten) erzeugen:

- `PointPair(Point point1, Point point2)`. Mit Hilfe des Codes `PointPair pointPair = new PointPair(p1, p2)` können Sie aus den beiden `Point`-Objekten `p1` und `p2` ein neues Punktpaar erzeugen. Die Distanz zwischen beiden Punkten wird hierbei automatisch berechnet.

- `PointPair(Point point1, Point point2, double distance)`. Mit `PointPair pointPair = new PointPair(p1, p2, dist)` können Sie aus den beiden `Point`-Objekten `p1` und `p2` ein neues Punktpaar erzeugen. Die Distanz `double dist` wird dabei mit übergeben und daher nicht mehr neu berechnet. Dieser Konstruktor eignet sich dann, wenn Sie die Distanz zwischen den beiden Punkten bereits berechnet haben. Beachten Sie, dass die Korrektheit der Distanz **nicht** überprüft wird!

Die Klasse `PointPair` stellt folgende Methoden zur Verfügung:

- `Point getPoint1()`: Gibt den ersten gespeicherten Punkt des Punktpaares zurück. Mittels `Point p1 = pointPair.getPoint1()` kann zum Beispiel der erste Punkt des `PointPair`-Objekts `pointPair` erhalten werden.

- `Point getPoint2()`: Gibt den zweiten gespeicherten Punkt des Punktpaares zurück. Mittels `Point p2 = pointPair.getPoint2()` kann zum Beispiel der zweite Punkt des `PointPair`-Objekts `pointPair` erhalten werden.

- `double getDistance()`: Gibt die Distanz des Punktpaares zurück. Mittels `double dist = pointPair.getDistance()` kann zum Beispiel die Distanz der beiden Punkte des `PointPair`-Objekts `pointPair` erfragt werden.

- `double computeDistance()`: Berechnet und speichert die Distanz des Punktpaares und gibt sie zurück. Mittels `double dist = pointPair.computeDistance()` kann zum Beispiel die Distanz der beiden Punkte des `PointPair`-Objekts `pointPair` berechnet und gespeichert werden. Der einzige Unterschied zur Methode `double getDistance()` ist, dass die Distanz vor der Rückgabe neu berechnet und gespeichert wird.

Hinweis: Die Reihenfolge, in der die Punkte im Punktpaar abgelegt werden, ist für die Ergebnisüberprüfung nicht relevant. Für zwei `Point`-Objekte `p1` und `p2` sowie zwei `PointPair`-Objekte `PointPair pair1 = new PointPair(p1, p2)` und `PointPair pair2 = new PointPair(p2, p1)` liefert `pair1.equals(pair2)` stets `true`.

5.1 Insertion Sort

Implementieren Sie die Methode `public void insertionSort(Point[] points, int a, int b, boolean sortX)`. Ziel dieser Methode ist es, die Punkte des Arrays `Point[] points` im Intervall $[a, b)$ gemäß der x-Koordinaten (falls `sortX` gleich `true` ist) oder der y-Koordinaten (falls `sortX` gleich `false` ist) mit Hilfe von Insertion Sort zu sortieren. Die Punkte außerhalb des Intervalls $[a, b)$ sollen unverändert bleiben.

Sie können gerne Ihre Methode aus Programmieraufgabe 1 als Grundlage heranziehen. Die Methode hat keine Rückgabe.

Beispiel: Angenommen, das Array `Point[] points` besteht aus sechs Punkten. Die folgende Tabelle zeigt den Zustand des Arrays vor und nach dem Aufruf von `insertionSort(points, 1, 4, true)`

Index	0	1	2	3	4	5
vorher	(7.2|5.4)	(3.1|2.6)	(6.1|8.0)	(1.5|7.9)	(9.5|6.8)	(4.2|5.7)
nachher	(7.2|5.4)	(1.5|7.9)	(3.1|2.6)	(6.1, 8.0)	(9.5|6.8)	(4.2|5.7)

Nach dem Aufruf sind die Punkte an den Stellen 1 bis $4 - 1 = 3$ aufsteigend gemäß der x-Koordinaten sortiert. Die anderen Stellen bleiben unberührt.

Diese Methode kann unter anderem vom Divide-and-Conquer-Algorithmus beim Schritt *Kombiniere* eingesetzt werden, um die Punkte im 2δ-Streifen adäquat scannen zu können (siehe Vorlesungsfolien).

5.2 Brute-Force-Ansatz

Implementierten Sie die Methode `public PointPair bruteForce(Point[] points, int a, int b)`. In dieser Methode sollen Sie mit Hilfe von Brute-Force, also dem Ausprobieren aller Punktpaare, das dichteste Punktpaar für die Punkte des Arrays `Point[] points` im Intervall $[a, b)$ finden. Punkte außerhalb des Intervalls $[a, b)$ sollen bei der Suche ignoriert werden.

Das gefundene Punktpaar soll als `PointPair`-Objekt zurückgegeben werden. Um ein Punktpaar aus zwei Punkten zu erstellen, können Sie die Konstruktoren aus Abschnitt 5 verwenden. Ist $b - a \leq 1$, dann soll `null` zurückgegeben werden, da es bei höchstens einem betrachteten Punkt kein Punktpaar geben kann.

Diese Methode wird vom Divide-and-Conquer-Algorithmus für jene Teilprobleme verwendet, die aus höchstens drei Punkten bestehen. Zusätzlich wird diese Methode laufzeittechnisch mit dem Divide-and-Conquer-Algorithmus verglichen.

5.3 Pivotelement bestimmen

Implementierten Sie die Methode `double getPivotValue(Point[] points, int a, int b, String method, Random random)`. Ziel dieser Methode ist es, ein Pivotelement zu bestimmen, das später bei der Zerlegung des Problems in zwei Teilprobleme herangezogen wird. Das Pivotelement erfüllt dabei die Funktion der vertikalen Trennlinie L in den Vorlesungsfolien.

Wir betrachten drei Möglichkeiten, das Pivotelement zu bestimmen. Das Argument `String method` steuert dabei, welche Möglichkeit zur Anwendung kommt. Folgende Werte können für `method` an die Methode übergeben werden:

- `"Random"`: Die x-Koordinate eines zufälligen Punktes aus dem Intervall $[a, b)$ soll zurückgegeben werden.

  Für die Generierung von Zufallszahlen eignet sich das Objekt `Random random`, das als letztes Argument übergeben wird. Die Klasse `Random` stellt unter anderem folgende Methode zur Verfügung:

  - `nextInt(int bound)`: Erzeugt eine `int`-Zufallszahl aus der Menge $\{0, 1, \ldots, \text{bound} - 1\}$. Mit `random.nextInt(5)` kann zum Beispiel eine Zufallszahl aus der Menge $\{0, 1, 2, 3, 4\}$ generiert werden.

- `"First"`: Die x-Koordinate des Punktes beim Index `a` soll zurückgegeben werden.

- `"Median Of Three"`: Es sollen die x-Koordinaten der Punkte an den Stellen `a`, `(b-a-1) / 2 + a` (Punkt in der Mitte, ggf. abgerundet) und `b - 1` ausgelesen werden und der Median dieser drei Werte zurückgegeben werden. Beachte, dass bei `(b-a-1) / 2 + a` Nachkommastellen abgeschnitten werden, da nur ganzzahlige Werte verwendet werden.

Für den Fall, dass $b - a \leq 1$ gilt, soll jedenfalls -1 zurückgegeben werden, um zu zeigen, dass eine Zerlegung dieses Problems keinen Sinn mehr macht.

Hinweis: Mittels `s1.equals(s2)` können Sie zwei `String`-Objekte `s1` und `s2` auf Gleichheit überprüfen. `"Hallo".equals("hallo")` würde also zum Beispiel die Strings `"Hallo"` und `"hallo"` vergleichen und `false` zurückgeben, da Java case-sensitive ist. `"Hallo".equals("Hallo")` hingegen würde `true` liefern.

5.4 Array splitten

Implementieren Sie die Methode `int split(Point[] points, int a, int b, double pivot)`. Diese Methode soll **in linearer Laufzeit** – also insbesondere ohne Verwendung eines Sortieralgorithmus – ein $t \in (a, b)$ bestimmen und das Array `points` derart manipulieren, sodass Folgendes gilt:

- Die x-Koordinaten aller Punkte von `points` im Intervall $[a, t)$ sind kleiner oder gleich dem Pivotwert `pivot` und

- die x-Koordinaten aller Punkte von `points` im Intervall $[t, b)$ sind größer oder gleich dem Pivotwert `pivot`.

Das so gefundene t hat den Zweck, das aktuell betrachtete Problem mit dem Intervall $t \in (a, b)$ in zwei (kleinere) Teilprobleme mit den Intervallen $[a, t)$ bzw. $[t, b)$ zu zerlegen. Die Punkte des Arrays `points` außerhalb des Intervalls $[a, b)$ sollen dabei unverändert bleiben. Beachten Sie, dass t weder den Wert a noch den Wert b annehmen darf, da sonst leere Intervalle entstehen würden.

Der gefundene Wert t soll zurückgegeben werden. Gleichzeitig sollen nach der Ausführung der Methode die beiden oben genannten Bedingungen erfüllt sein. Wenn das Intervall $[a, b)$ zu klein ist (was der Fall ist, wenn $b - a \leq 1$ gilt), so soll -1 zurückgegeben werden. In dem Fall könnte nämlich kein geeignetes t gefunden werden.

Beispiel: Gegeben sei ein `Point`-Array mit folgendem Inhalt:

Index	0	1	2	3	4	5	6	7
Array	(8|2)	(7|2)	(4|9)	(3|5)	(3|3)	(5|3)	(2|7)	(1|8)

Für das Intervall $[a, b) = [2, 7)$ und das Pivotelement 5 könnte das Array am Ende so aussehen:

Index	0	1	2	3	4	5	6	7
Array	(8|2)	(7|2)	(4|9)	(3|5)	(3|3)	(2|7)	(5|3)	(1|8)

Der einzig gültige Wert für t ist 6: Alle x-Koordinaten der Punkte im Intervall $[a, t) = [2, 6)$ sind kleiner oder gleich 5 und alle x-Koordinaten der Punkte im Intervall $[t, b) = [6, 7)$ sind größer oder gleich 5. Für kein anderes $t \in (a, b)$ wäre dies der Fall.

Für dasselbe Intervall und das Pivotelement 3 könnte das Array am Ende so aussehen:

Index	0	1	2	3	4	5	6	7
Array	(8|2)	(7|2)	(2|7)	(3|5)	(3|3)	(5|3)	(4|9)	(1|8)

Überzeugen Sie sich davon, dass in diesem Fall $t = 3$, $t = 4$ und $t = 5$ gültig sind.

5.5 Kombinieren

Implementieren Sie die Methode `PointPair combination(Point[] points, int a, int b, int t, double delta, double L)`. Diese Methode soll den Schritt *Kombiniere* des Divide-and-Conquer-Algorithmus ausführen.

Das Array `Point[] points` enthält die Punkte und `int a` und `int b` markieren das Intervall $[a, b)$, welches das aktuelle Teilproblem begrenzt. `double delta` steht für das δ aus den Folien und steht für die derzeit kürzeste bekannte Distanz zweier Punkte. Der Parameter `double L` steht für die vertikale Trennlinie L aus den Folien und wird mit der Methode aus Abschnitt 5.3 bestimmt. Der Parameter `int t` steht für den gefundenen Index, er entspricht dem t, das von der Methode aus Abschnitt 5.4 bestimmt wird. Sie dürfen davon ausgehen, dass die Punkte innerhalb des Intervalls $[a, b)$ gemäß ihrer x-Koordinaten sortiert sind.

Die Umsetzung soll dabei in folgenden Schritten erfolgen:

1. Finde das kleinstmögliche Intervall $[\bar{a}, \bar{b})$, sodass alle Punkte innerhalb dieses Intervalls höchstens um δ von der vertikalen Trennlinie entfernt sind. Mit anderen Worten: Finde den Bereich jener Punkte innerhalb des 2δ-Streifens.

2. Sortiere die Punkte im Intervall $[\bar{a}, \bar{b})$ gemäß ihrer y-Koordinaten.

3. Scanne die Punkte im Intervall $[\bar{a}, \bar{b})$ in y-Reihenfolge.

Die Methode soll das dichteste Punktepaar bestimmen und als `PointPair`-Objekt zurückgeben (vgl. Abschnitt 5.2). Existiert kein Punktepaar (weil im 2δ-Streifen weniger als zwei Punkte liegen), so soll `null` zurückgegeben werden.

Hinweis: Sie können auf bereits implementierte Methoden innerhalb von `StudentSolutionImplementation.java` leicht zurückgreifen. Wenn Sie zum Beispiel Insertion Sort mit den Parametern $a = 1$ und $b = 4$ aufrufen wollen, um die Punkte des `Point`-Objektes `points` gemäß der x-Koordinaten zu sortieren, dann rufen Sie einfach `insertionSort(points, 1, 4, true)` auf.

5.6 Dichtestes Punktpaar bestimmen mit Divide-and-Conquer

Im Folgenden ist die Methode `PointPair closestPair(Point[] points, int a, int b, double bestSoFar, String pivotMethod)` abgebildet, die Sie **nicht** mehr implementieren müssen und welche den Divide-and-Conquer-Algorithmus aus den Vorlesungsfolien umsetzt. Die Methode wird vom Framework aufgerufen und verwendet die in den Abschnitten 5.1 bis 5.5 implementierten Methoden.

```java
public PointPair closestPair(Point[] points, int a,
   int b, double bestSoFar, String pivotMethod,
   Random random) {
     if (b - a <= 3) {
         // Löse Teilproblem direkt
         insertionSort(points, a, b, true);
         return bruteForce(points, a, b);
     }
```

```java
// Zerlege das Problem in zwei Teilprobleme
double L = getPivotValue(points, a, b,
    pivotMethod, random);
int t = split(points, a, b, L);

PointPair res = null;

// Linkes Teilproblem lösen
PointPair links = closestPair(points, a, t,
    bestSoFar, pivotMethod, random);
if (links != null && links.getDistance() <
    bestSoFar) {
      bestSoFar = links.getDistance();
}
if (links != null && (res == null ||
    links.getDistance() < res.getDistance())) {
      res = links;
}

// Rechtes Teilproblem lösen
PointPair rechts = closestPair(points, t, b,
    bestSoFar, pivotMethod, random);
if (rechts != null && rechts.getDistance() <
    bestSoFar) {
      bestSoFar = rechts.getDistance();
}
if (rechts != null && (res == null ||
    rechts.getDistance() < res.getDistance())) {
      res = rechts;
}

// Stand sichern
Point[] copy = new Point[b - a];
for (int i = 0; i < b - a; i++) {
      copy[i] = points[i + a];
}

// Kombiniere
PointPair mitte = combination(points, a, b,
    bestSoFar, t, L);
```

```java
if (mitte != null && (res == null ||
    mitte.getDistance() < res.getDistance())) {
     res = mitte;
}

// Punkte zurücksetzen
for (int i = 0; i < b - a; i++) {
     points[i + a] = copy[i];
}

return res;
```

Zu Beginn wird diese Methode mit `a = 0`, `b = points.length` und `bestSoFar = Double.POSITIVE_INFINITY` aufgerufen. Der Parameter `bestSoFar` bezeichnet dabei die Distanz des bisher dichtesten Punktpaares und wird zu Beginn auf Unendlich gesetzt. Er erfüllt die Funktion von δ.

Das gefundene dichteste Punktepaar wird als `PointPair`-Objekt zurückgegeben, analog zum Brute-Force-Ansatz aus Abschnitt 5.2.

Anmerkung: Vor dem *Kombiniere*-Schritt müssen in dieser Implementierung die Punkte des Intervalls $[a, b)$ zwischengespeichert und nach dem *Kombiniere*-Schritt wieder zurückgesetzt werden. Der Grund dafür ist, dass im *Kombiniere*-Schritt die Punkte nach ihren y-Koordinaten sortiert werden, also die (für andere Rekursionsebenen notwendige) Sortierung gemäß der x-Koordinaten zunichte gemacht wird.

6 Testen

Führen Sie zunächst die `main`-Methode in der Datei `src/main/java/framework/Exercise.java` aus.

Anschließend wird Ihnen in der Konsole eine Auswahl an Testinstanzen angeboten, darunter befindet sich zumindest `abgabe.csv`:

```
Select an instance set or exit:
[1] abgabe.csv
[0] Exit
```

Durch die Eingabe der entsprechenden Ziffer kann entweder eine Testinstanz ausgewählt werden oder das Programm (mittels der Eingabe von `0`) verlassen werden. Wird eine Testinstanz gewählt, dann wird der von Ihnen implementierte Programmcode ausgeführt. Kommt es dabei zu einem Fehler, wird ein Hinweis in der Konsole ausgegeben.

Relevant für die Abgabe ist das Ausführen der Testinstanz `abgabe.csv`.

Die Testinstanzen `insertion-sort.csv`, `brute-force.csv`, `random.csv`, `first.csv`, `median-of-three.csv`, `split.csv` und `combination.csv` können Sie zum Testen der Unteraufgaben nutzen. `closest-pair.csv` enthält alle Instanzen, die den Divide-and-Conquer-Algorithmus aus Abschnitt 5.6 aufrufen.

7 Evaluierung

Wenn der von Ihnen implementierte Programmcode mit der Testinstanz `abgabe.csv` ohne Fehler ausgeführt werden kann, dann wird nach dem Beenden des Programms im Ordner `results` eine Ergebnis-Datei mit dem Namen `solution-abgabe.csv` erzeugt.

Die Datei `solution-abgabe.csv` beinhaltet erstellte Bäume und Zeitmessungen der Ausführung der Testinstanz `abgabe.csv`, welche in einem Web-Browser visualisiert werden können. (Auch Ergebnis-Dateien anderer Testinstanzen können zu Testzwecken visualisiert werden.) Öffnen Sie dazu die Datei `visualization.html` in Ihrem Web-Browser und klicken Sie rechts oben auf den Knopf *Ergebnis-Datei auswählen*, um `solution-abgabe.csv` auszuwählen.

Beantworten Sie basierend auf der Visualisierung die Fragestellungen aus dem folgenden Abschnitt.

8 Fragestellungen

Öffnen Sie `solution-abgabe.csv`. Um die Optimallösung eines gegebenen Problems zu bestimmen, wurde Divide-and-Conquer mit den in Abschnitt 5.3 beschriebenen Möglichkeiten, das Pivotelement zu bestimmen, ausgeführt (*Random*, *First* und *Median Of Three*). Zusätzlich wird auch der Brute-Force-Ansatz zu Vergleichszwecken angewendet, der Ihre Methode aus Abschnitt 5.2 verwendet (*Brute Force*).

Die Algorithmen wurden jeweils auf zwei unterschiedlichen Sets von Instanzen ausgeführt; die beiden abgebildeten Laufzeitgrafiken stehen dabei für die Instanzsets *set1* und *set2*.

Bearbeiten Sie folgende Aufgaben- und Fragestellungen:

1. Erklären Sie, wie die Umsetzung des Divide-and-Conquer-Algorithmus aus Abschnitt 5.6 mit den Erläuterungen auf den Vorlesungsfolien zusammenhängt.

   Beantworten Sie außerdem folgende Fragen:

   - Wo und wie fließt dabei Quicksort implizit hinein?
   - Warum würde der Einsatz eines Sortieralgorithmus in Abschnitt 5.4 der Idee von Quicksort widersprechen? Mit anderen Worten: Warum ist es wichtig, dass die Methode `int split(Point[] points, int a, int b, double pivot)` eine lineare Laufzeit hat? Erfüllt Ihre Implementierung dieser Methode die Voraussetzung der linearen Laufzeit?

2. In Abschnitt 5.5 haben Sie in Schritt 2 die Punkte des Intervalls $[\bar{a}, \bar{b})$ nach ihren y-Koordinaten sortiert. Deckt sich Ihre Implementierung mit den Erläuterungen der Vorlesung auf Folie 79? Falls nein: Wie könnte die Idee dieser Folie umgesetzt werden? Eine Beschreibung genügt, eine Implementierung ist nicht notwendig.

3. Durch Klicken auf den Gruppennamen in der Legende neben der Plots lassen sich einzelne Gruppen aus- bzw. einblenden. Blenden Sie in allen Grafiken alle Kategorien bis auf *Brute Force* aus. Beschreiben Sie das Laufzeitverhalten anhand der Plots für *Brute Force* und erklären Sie basierend auf Ihrer Implementierung, wie dieses Laufzeitverhalten zustande kommt.

   Drücken Sie im Anschluss in der Menüleiste rechts über dem Plot auf den Fotoapparat, um die Plots als Bild zu speichern.

4. Blenden Sie nun in allen Grafiken nur die Kategorien *Random* und *Median Of Three* ein. Beschreiben Sie das Laufzeitverhalten. Sind Unterschiede feststellbar? Wenn ja, wie erklären Sie sich diese bzw. welche Rückschlüsse auf die unterschiedlichen Instanzsets können Sie daraus ziehen? Erstellen Sie nun auch ein Bild dieser Plots.

5. Blenden Sie nun zusätzlich zu den Kategorien *Random* und *Median Of Three* die Kategorie *First* ein. Sind Unterschiede feststellbar?

Wenn ja, wie erklären Sie sich diese bzw. welche Rückschlüsse auf die unterschiedlichen Instanzsets können Sie daraus ziehen? Erstellen Sie nun auch je ein Bild dieser Plots.

Blenden Sie nun zusätzlich die Kategorie *Brute Force* ein. Vergleichen Sie das Laufzeitverhalten von *Brute Force* mit *Random*, *First* und *Median Of Three*. Sind Unterschiede feststellbar? Wenn ja, wie erklären Sie sich diese? Erstellen Sie auch nun je ein Bild dieser Plots.

6. Wenn das Pivotelement ungünstig gewählt wird, so kann es in der rekursiven Implementierung zu einem `StackOverflowError` kommen. Ein `StackOverflowError` tritt dann auf, wenn zu viele Rekursionsaufrufe offen sind. Erklären Sie, unter welchen Umständen ein solcher Fehler verstärkt auftritt bzw. auftreten kann. Verorten Sie eine erhöhte Gefahr für diesen Fehler bei den hier betrachteten Instanzsets bzw. angewendeten Methoden?

7. In Abschnitt 5.4 haben wir gefordert, dass $t \in (a, b)$ liegen muss. Anders als bei Quicksort muss nämlich das Pivotelement entweder der linken oder der rechten Seite zugeordnet werden. Angenommen, wir hätten stattdessen zugelassen, dass $t \in [a, b]$ liegt. Benennen Sie ein konkretes Szenario (also eine konkrete (kleine) Beispielinstanz und eine Methode für die Auswahl des Pivotelements), in dem der Aufruf des Divide-and-Conquer-Algorithmus aus Abschnitt 5.6 zu einer Endlosrekursion führen würde.

Falls sich im Zuge der Evaluierung die Darstellung der Plots auf ungewünschte Weise verändert (z.B. durch die Auswahl eines zu kleinen Ausschnitts), können Sie mittels Doppelklick auf den Plot oder Klick auf das Haus in der Menüleiste die Darstellung zurücksetzen.

Fügen Sie Ihre Antworten in einem Bericht gemeinsam mit den erstellten Bildern der Plots zusammen.

9 Abgabe

Laden Sie die Datei `src/main/java/exercise/StudentSolutionImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P3* hoch. Fassen Sie diesen Bericht mit den anderen für das zugehörige Abgabegespräch relevanten Berichten in einem PDF zusammen und geben Sie dieses in der TUWEL-Aktivität *Hochladen Bericht Abgabegespräch 1* ab.

10 Nachwort

Die Aufgabenstellung in der euklidischen Ebene ein dichtestes Paar von Punkten zu finden ist eine fundamentale im Bereich der Geometrie, und natürlich gibt es auch viele praxisrelevante Verallgemeinerungen davon. Im Dreidimensionalen, beispielsweise, ist es in der Luftraumüberwachung wichtig rasch Alarm zu schlagen, wenn sich zwei Flugobjekte zu sehr nähern. Mit einer Verallgemeinerung der hier betrachteten Methode kann das kritischste Paar von Flugobjekten effizient gefunden werden. Eine andere Verallgemeinerung betrachtet nicht nur Punkte sondern allgemeinere Objekte mit einer räumlichen Ausdehnung. Hierfür ist es schon etwas schwieriger zu einem effizienten Algorithmus zu kommen, jedoch kann auch hier auf das Divide-and-Conquer Prinzip aufgebaut werden.

Allgemeiner sind viele Aufgabenstellungen, die eine geometrische Interpretation bzw. Struktur haben, häufig effizienter lösbar als abstraktere Problem ohne einer solchen, da eben ein rekursives Unterteilen des Problems in einer räumlichen Dimension eine gute Ausgangslage zur Anwendung des Divide-and-Conquer Prinzips bietet. Beispielsweise gibt es auch spezialisierte Algorithmen zum Finden eines minimalen Spannbaums von Punkten in der euklidischen Ebene, die effizienter sind als Prims bzw. Kruskals Algorithmen.

Neben Algorithmen werden wir auch Datenstrukturen kennenlernen, die das Prinzip des Divide-and-Conquer verfolgen, und solche spielen bei geometrischen Problemstellungen auch oft eine zentrale Rolle. Mehr zum Thema Algorithmen für geometrische Probleme erfahren Sie in unserer LVA *Algorithmic Geometry*.