Facharbeit Informatik

Behandlung und Erklärung des Rucksackproblems anhand einer eigenen Implementierung in Java

Simon Klemp

Facharbeit Informatik 15 / 16

Städtisches Gymnasium Rheinbach

Kurs: IFLK1

Lehrer: Herr Faßbender

Inhaltsverzeichnis

1 Einleitung

„Es war einmal ein tapferer Krieger aus dem Volk, der rettete die Prinzessin Tausendschön vor bösen Räubern[…]. Leider musste sich ihr Vater, der König, etwas als Belohnung ausdenken: […] Er gab ihm eine Kiste aus Holz und sagte ihm, er dürfe alles aus seiner Schatzkammer mitnehmen, was in die Kiste passt, jedoch muss der Deckel der Kiste sich am Ende noch schließen lassen. “[1] Die Frage, die sich der tapfere Mann jetzt stellt, ist, welche Sachen er wohl mitnehmen solle um den größtmöglichen Ertrag zu erlangen. Dabei greift das Rucksackproblem der Informatik, welches allgemein ein Optimierungsproblem der Kombinatorik ist. Aus einer bestimmten Menge von Objekten sollen diese ausgewählt werden, welche am Ende zusammengerechnet den größtmöglichen Wert haben, dabei jedoch das vorgegebene Grenzgewicht oder Volumen nicht überschreiten. Infolge der Facharbeit werde ich das Gewicht und nicht das Volumen betrachten, was aber an sich keinen Unterschied macht. Zu diesem Problem habe ich im Rahmen der Facharbeit ein Programm in der Programmiersprache Java implementiert, welches sich mit einigen der möglichen Algorithmen zur Berechnung der Menge der Objekte und deren Laufzeit befasst. Ich habe mir dieses Thema nicht selber ausgesucht, sondern es wurde mir vorgeschlagen. Nach einigem Zweifeln finde ich das Thema jetzt ziemlich interessant, wobei mich die Laufzeit der verschiedenen Algorithmen sehr interessiert, weshalb ich darauf auch genauer eingehen werde.

[1] Gallenbacher, J.(2008) , Abenteuer Informatik, Spektrum Akademischer Verlag; Auflage: 2.Aufl. Juni 2008

2 Algorithmen

2.1 Binärer- Algorithmus

Der Binäre-Algorithmus ist die einfachste aber auch laufzeitintensivste Variante der Algorithmen. Bei diesem wird jede Möglichkeit durchgegangen, wie die Objekte kombiniert werden könnten, um dann das Maximum aus diesen zu bestimmen. Es ist ein binärer Algorithmus, da man die Zahl der Möglichkeiten als Binärzahl darstellt. Hat man beispielsweise 4 Objekte, hat man 2^4 mögliche Kombinationen der Objekte. Dabei wird jede Kombination in einem String gespeichert. Bei der Zahl 15 wäre es zum Beispiel „1111“, wobei jede Stelle des Strings für ein mögliches Objekt steht, eine 1 steht für Hinzufügen und eine 0 für nicht Hinzufügen. In diesem Fall hat man also vier Objekte, welche alle hinzugefügt werden. Nun könnte man sich fragen, weshalb man bei 16 Kombinationsmöglichkeiten(2^4=16) nicht auch die Zahl 16 als Binärzahl errechnen und als mögliche Kombination nehmen könnte. Die Antwort ist ganz einfach: 16 wäre als Binärzahl „10000“ und passt zum einen nicht in das Muster der nur 4 Objekte, da diese 5 Stellen hat, zum anderen wird diese auch nicht benötigt. Normalerweise fängt man bei 1 an zu zählen und dann weiter bis 16, jedoch muss man hier auch den Fall 0 betrachten, da bei 0 keine Objekte eingepackt wären „0000“. [2]Man zählt also von 0 bis 15 und hat am Ende auch wieder 16 Kombinationen. Diese mögliche Variante der Betrachtung des Rucksackproblems ist jedoch, wie schon gesagt, sehr unelegant und aufwändig, weil schon bei wenigen Objekten eine lange Laufzeit entsteht. Das zeigt sich auch, wenn man sich klar macht, dass bei 100 Objekten etwa $1{,}2677*10^{30}$ Möglichkeiten durchprobiert werden müssen. Eine bessere Lösung wäre die Rekursive Betrachtung des Algorithmus.

2.2 Rekursiver-Algorithmus

Der Rekursive-Algorithmus ist ähnlich wie der binäre Algorithmus, jedoch in seiner Laufzeit und Eleganz diesem Meilen voraus. Bei diesem werden mögliche Objekte rekursiv zusammengeführt. Rekursiv bedeutet sich selbst aufrufend. Es gibt also eine Methode, welche sich selbst immer wieder mit einer erhöhten Zahl aufruft, bis eine bestimmte Bedingung erfüllt ist. Man fängt also bei 0 an. Dort passiert erstmal nichts. Jedoch muss man sich dann entscheiden ob man den nächsten Gegenstand einpackt

[2] Vgl. http://www.inf-schule.de/grenzen/komplexitaet/rucksackproblem/station_loesungsalgorithmus zuletzt aufgerufen am 25.03.2016

oder nicht, also macht man einen Rekursionsaufruf mit dem Gegenstand und einen ohne. Nun könnte man denken, dass die Laufzeit gleichgroß sei, wie auch beim binären Algorithmus, jedoch ist dies durch eine bestimmte Abfrage nicht der Fall, nämlich die Folgende:

```
if(aktGewicht - pGegenstaende[i][0] >= 0){
    ...
}
```

Diese wird abgefragt vor dem zweiten Rekursionsaufruf, also vor Einpacken des Objektes. Es wird abgefragt ob das Gewicht der bisher in dieser Kombination eingepackten Objekte mit dem Einpacken des aktuellen Objektes das Gesamtgewicht überschreitet. Hier ist das „aktGewicht“ das übergebliebene Restgewicht der bisher eingepackten Objekte. Wenn man also ein Gesamtgewicht von 10 hat und schon zwei Objekte, mit jeweils einem Gewicht von 2 eingepackt hat ist das „aktGewicht“ 6 -> 10 – 2*2 = 6. „pGegenstaende[i][0]“ gibt hierbei das Gewicht des einzupackenden Objektes an. Das, abgezogen von dem aktuellen Restgewicht, muss größer Null sein, was auch klar ist, da man das maximale Gewicht nicht überschreiten darf. Wenn das Objekt in diesem Fall zum Beispiel das Gewicht 8 hat funktioniert es nicht: 6 – 8 = -2. Das maximale mögliche Gewicht wurde also um 2 Einheiten überschritten. Durch diese Abfrage kann ein ganzer Rekursionsabschnitt entfallen, nämlich dieser, das aktuelle Objekt eingepackt zu haben. Dadurch wird einiges an Laufzeit gespart, aber diese Variante stark abhängig von den Objekten und dem maximalen Gewicht. Sie ist trotzdem in so gut wie jedem Fall schneller als die Binäre Betrachtungsweise. Dennoch kann man auch diese Variante noch verbessern, indem man sich schon berechnete Aufrufe in einem Array speichert, dadurch kann man sich ganze Rekursionsteile sparen. Man muss bei einem vorherigen Rekursionsaufruf nur abfragen, ob sich ein so schon berechneter Rekursionsaufruf im Array befindet. Wenn man zum Beispiel 4 Objekte hat kann man diese, vereinfacht zum Erklären, wieder als binär Code[3] darstellen, wobei eine 1 als eingepackt und eine 0 als nicht eingepackt gilt. Man hat in seinem Array schon den Wert des Rekursionsaufrufes „0011“ gespeichert, und jetzt möchte ein weiterer Rekursionsaufruf den Wert „1111“ errechnen. Dieser könnte bei „11“ die Rekursion beenden, indem er die bereits errechneten Werte aus dem Array entnimmt.

[3]Vgl. https://www.proggen.org/doku.php?id=algo:knapsack zuletzt aufgerufen am 25.03.2016

Dadurch kann man bei größeren Objektmengen einiges an Laufzeit einsparen. Ich habe diese Variante nicht in meiner Implementation verwendet, sondern nur die zuerst genannte Rekursionsvariante.

2.3 Greedy-Algorithmus

Der Greedy-Algorithmus ist zwar der schnellste der bisher behandelten Algorithmen, jedoch ist er keine exakte Lösung des Problems. Greedy ist Englisch und steht im Deutschen für gierig, was auch auf den Algorithmus gut zutrifft, da dieser zwar schnell ein Ergebnis herausbringt, dieses jedoch in vielen Fällen nicht die optimale Lösung ist. Der Greedy-Algorithmus sortiert die ganzen Objekte zunächst nach ihrer Greedy-Zahl, diese kann man auch als Nutzwert bezeichnen. Um ihn zu erhalten muss man den Wert durch das Gewicht teilen, bei einem Gewicht von 10 und einem Wert von 15 wäre der Greedy-Wert oder die Greedy-Zahl 1,5. Bei den nun sortierten Werten werden solange die Werte mit den größten Greedy-Zahlen eingefügt, bis das nächste Objekt die Gewichtsbegrenzung überschreiten würde. Dieses wird dann übersprungen und das darauffolgende wird betrachtet. Das passiert solange, bis alle Objekte durchlaufen sind. Er bietet jedoch insofern keine genaue Lösung, als dass er nicht alle möglichen Kombinationen durchgeht. Wenn man zum Beispiel einen Behälter mit maximalem Gewicht 10 hat und 1 Objekt mit Größe 6 und Wert 12 hat dieses den Greedy-Wert 2 und 2 weitere Objekte mit jeweils dem Gewicht 5 und Wert 7 haben den Greedy-Wert 1,4. Dabei würde also zuerst das Objekt mit Größe 6 einsortiert werden, bei dem Restgewicht von 4 können aber die anderen beiden Objekte nicht mehr einsortiert werden. Hätte man jedoch die Möglichkeit ausprobiert diese beiden einzufügen, hätte man als Gesamtwert 14 anstatt 12 mit dem anderen. Jedoch ist die Abweichung des Greedy-Algorithmus in sehr großen Mengen an Objekten relativ gering und bietet eine sehr gute Laufzeit.[4]

[4] Vgl. https://www.proggen.org/doku.php?id=algo:knapsack zuletzt aufgerufen am 25.03.2016

3 Programmiersprache und Umgebungen

3.1 Java

Java wurde 1991 bei Sun Microsystems entwickelt, es sollte möglichst speichereffizient sein und eine geringe Rechenleistung beanspruchen. 1995 wurde dann der Name Java für die Programmiersprache eingeführt, Java wurde im Jahre 2010 an Oracle verkauft. Sie ist eine objektorientierte Programmiersprache, welche größtenteils plattformunabhängig ist[5] und ist grundlegend in drei Bereiche eingeteilt, zuerst die Programmiersprache an sich, also Java. Als nächstes gibt es das Java Development Kit (JDK), welches die Funktion hat das zuvor programmierte Programm in einen Bytecode umzuwandeln, eine Datei mit diesem Bytecode wird durch die Endung „.java“ angegeben. Zuletzt gibt es noch das Java Runtime Environment (JRE), welches zum Ausführen der Programme wichtig ist. Dieses liest den Bytecode und führt das Programm dementsprechend aus. Da diese Laufzeitumgebung weitgehend unabhängig vom Betriebssystem ausgeführt werden kann gibt es Java für eine Vielzahl an Geräten[6]. Von Oracle selber werden beispielsweise Laufzeitumgebungen für Linux, OS X, Solaris und Windows angeboten, jedoch kann man auch für andere Plattformen Laufzeitumgebungen zertifizieren lassen. Dadurch wird Java momentan auf Handys, Rechenzentren, Spielekonsolen und Supercomputern verwendet. Sie ist nach dem RedMonk-Programmiersprachenindex von 2015 auf Platz zwei der populärsten Programmiersprachen weltweit[7].

3.2 Bluej und Netbeans

Für meine Facharbeit habe ich sowohl Netbeans wie auch Bluej verwendet, diese sind beide Entwicklungsumgebungen für Java. Dabei ist Netbeans für eine grafische Ausgabe spezialisiert und Bluej als Entwicklungsumgebung speziell für Ausbildungszwecke. Netbeans ist komplett in der Programmiersprache Java geschrieben und ist auch hauptsächlich für diese gedacht, jedoch unterstützt sie auch die Programmiersprachen C und C++.[8] Es ist für grafische Ausgabefenster, eine

[5] Vgl. Fuchs, E.(2014) Java 8 Grundlagen Programmierung; Herdt Verlag; Ausgabe: 1.Aufl. Oktober 2008
[6] Vgl. https://www.java.com/de/about/ zuletzt aufgerufen am 25.03.2016
[7] Vgl. https://de.wikipedia.org/wiki/Java_(Programmiersprache) zuletzt aufgerufen am 25.03.2016
[8] Vgl. https://de.wikipedia.org/wiki/NetBeans_IDE zuletzt aufgerufen am 25.03.2016

sogenannte GUI, gedacht und erleichtert die Gestaltung dieser, da man die einzelnen Komponenten nicht als Text Code eingeben muss. Man kann aus einer Palette von Objekten, die man braucht, auswählen und diese in dem GUI Fenster anzupassen. Der Code dafür wird automatisch von Netbeans generiert. Bei Bluej steht die objektorientierte Programmierung im Mittelpunkt, dazu kann man zunächst Java Klassen erstellen, welche auf einer Startoberfläche als verkürzte UML-Diagramme angezeigt werden. Diese kann man dann durch Anklicken der Klassenkarte mit Methoden füllen. Einer der Vorteile von Bluej ist ein Compiler, welcher dem Benutzer nach Ausführung zurückgibt ob der Code korrekt war oder nicht und zeigt die gemachten Fehler an (jedoch nur Syntax Fehler). Ein weiterer Vorteil liegt in der Übersichtlichkeit der einzelnen Klassen, welche durch die Klassenkarten angegeben werden, aber auch der Methoden im Programm, welche farblich getrennt werden.

4 Programm

4.1 Erklärung der Komponenten und Funktion

Das Programm besteht aus 13 Klassen, wovon 5 Klassen als GUI und Aufrufen und Auswerten der Daten dienen, 6 Klassen zum Ausführen der Algorithmen, eine zum Erstellen eines zufälligen Arrays oder einer zufälligen Liste, eine weitere, welche das Objekt für die Liste bildet und zuletzt eine Main Methode, welche beim Starten des Programms aufgerufen wird (siehe Anhang UML-Diagramm). Bei der Implementation des Programmes unterscheide ich bei den Algorithmen jeweils zwischen Array und Liste und bei dem Greedy-Algorithmus auch noch zwischen den verschiedenen Sortieralgorithmen. Dies mache ich um einen Eindruck der unterschiedlichen Laufzeit der einzelnen Möglichkeiten zu bekommen, jedoch werde ich im Laufe der Facharbeit noch genauer auf die verschiedenen Laufzeiten eingehen. Im Folgenden werde ich die wichtigsten Schritte der Implementierung bei Ausführung des Programmes beschreiben. Wie schon gesagt startet das Programm mit dem Aufruf der main-Methode der Klasse GUI, welche wiederum die main-Methode der Klasse Startseite aufruft. Diese stellt eine einleitende Oberfläche mit allgemeinen Informationen dar. Nachdem man nun den Startbutton gedrückt hat wird die sogenannte actionPerformed Methode dieses Buttons aufgerufen, in welcher der aktuelle jPanel des jDialogs geändert wird. Der jDialog stellt in diesem Fall den Inhalt des Fensters dar

und ein jPanel ist einfach gesagt der Inhalt des jDialog, also ist der jPanel in diesem Fall ein darzustellender Inhalt auf dem jDialog. Nun wird der neue jPanel angezeigt, welchen ich vereinfacht „mainPage“ nenne. Auf diesem sind drei jButtons angezeigt, mit den Aufschriften „zurück“, „Einstellungen“ und „Laufzeitanalyse“. Bei Betätigen des Zurückbuttons kommt man auf die vorherige Startseite zurück. Zunächst sollte man jedoch auf den Einstellungsbutton klicken. Daraufhin öffnet sich ein kleines Einstellungsfenster. Man hat nun einige Auswahlmöglichkeiten, zunächst sollte man sich für einen der drei möglichen Algroithmen entscheiden. Das man nur einen wählen kann ist extra so gewählt, da die Daten später ausgegeben werden und dort nur ein Datenwert ausgegeben werden kann. Daraufhin hat man die Möglichkeit auf die Auswahl des Objektspeichers, also worin die Objekte oder Werte gespeichert werden. Wählt man nun das Array werden die später erzeugten Gewichte und dazugehörenden Werte in einem zweidimensionalen Array des Datentyps float gespeichert. Ein float ist eine Fließkommazahl, welche eine hohe Genauigkeit hat und hat eine Größe von bis zu 32 Bit. Wenn man andernfalls die Liste auswählt werden die Werte in einer Array Liste gespeichert, wobei der Vorteil dieser die Variabilität ist, da sie im Gegensatz zum Array dynamisch ist, also in ihrer Länge während der Laufzeit veränderbar. In dieser Liste werden Objekte des Typs „Objekt“ gespeichert, welche die Attribute Gewicht und Wert besitzt. Zuletzt kann man zwischen den Sortieralgorithmen unterscheiden, dabei gibt es den Selection, Insertion, Bubble und Quicksort. Diese kann man jedoch nur auswählen, wenn man zuvor den Greedy Algorithmus ausgewählt hat, da auch nur dort sortiert werden muss. Nun besteht auch noch die Möglichkeit Daten manuell einzugeben, indem man in das Textfeld in der Mitte des Bildschirmes mit der Aufschrift „EigenesGewicht“ das Gewicht des eigenen Objektes einträgt und in das Feld daneben mit der Aufschrift „EigenerWert“ den eigenen Wert einträgt. Dann muss man nur noch auf Add drücken und das Objekt wird zu einer eigenen Liste hinzugefügt. Die Anzahl der Objekte oben zeigt dabei die Anzahl der bisher in die Liste eingefügten Objekte an. Zuletzt gibt es noch einen Reset und einen setDefault button, wobei der Erste alles bisher Ausgewählte zurücksetzt und auch die intern gespeicherte Liste leert und der Zweite meine voreingestellten Standardeinstellungen auswählt, also den Binären Algorithmus mit einer Liste, 20 Objekten und einer Behältergröße von 15 .

Sollte eines der Pflichtfelder nicht ausgefüllt sein, wird in der Mitte des Fensters auf dem TextField eine Fehlermeldung ausgegeben, mit dem noch auszuwählenden Parameter und der Startbutton wird sich nicht drücken lassen. Wenn man aber alle notwendigen Felder ausgefüllt hat öffnet sich die Startseite mit jPanel „mainPage“ wieder, jedoch wurden nun Werte eingetragen und zwar in 2 Fenster des Typs Liste mit Inhaltsobjekt String. Die erste Liste mit Überschrift „Alle Objekte“ stellt genau diese dar, also alle erstellten Objekte. Wenn man als Anzahl zuvor 20 genommen hat, oder selbst 20 Objekte eingefügt hat, werden dort nun 20 Objekte dargestellt. Darunter gibt es noch eine Liste mit Überschrift „Ausgewählte Objekte“. In dieser werden alle Objekte angezeigt, welche zuvor durch den Algorithmus bestimmt wurden eingepackt zu werden. Darunter gibt es 3 Textfelder, wobei das Erste das Gesamtgewicht der ausgewählten Objekte anzeigt, das Zweite den gesamten Wert der Objekte und das Dritte die Laufzeit in Millisekunden. Wenn man nochmal den Einstellungsbutton betätigt werden alle Werte wieder gelöscht und es können in den Einstellungen wieder neue Sachen ausgewählt werden. Nun gibt es auch noch den Laufzeitanalyse Button. Dieser startet zunächst ein kleineres Fenster mit erneuten Einstellungen. Diese sind ähnlich wie die vorherigen, jedoch nicht identisch. Man kann mehrere Algorithmen und auch mehrere Speicherarten auswählen und wenn man den Greedy-Algorithmus verwendet auch mehrere Sortieralgorithmen. Das Textfeld in der Mitte der Seite zeigt wieder eine Fehlermeldung bei fehlenden Daten an. Wenn man alles richtig ausgewählt hat schließt sich das Einstellungsfenster und es öffnet sich ein weiteres Fenster, welches ein Koordinatensystem mit relativen Achseneinteilungen anzeigt. Dabei wird auf der X-Achse die Anzahl angezeigt, im Vergleich auf der Y-Achse die Laufzeit. In der Mitte wird nun ein Graph gezeichnet, welcher eine graue Farbe hat und mit den jeweiligen ausgewählten Parametern beschriftet ist. Wenn der Graph nicht direkt gezeichnet wird, liegt das an der Rechenzeit des Programms, da auch hierfür die Werte erstmal berechnet werden müssen. Unten auf der Seite sieht man dafür ein Textfeld, auf dem der aktuelle Status angezeigt wird. Wenn auf diesem eine Zahl angezeigt wird bedeutet das, das Programm ist noch am Rechnen und zwar bei der Anzahl der Objekte, welches die Zahl zeigt. Bei der Zahl 20 sind das also 20 Objekte, aus denen gerade die einsortierte Menge ausgewählt wird, nach dem zuvor ausgewählten Algorithmus. Wenn man mehrere Möglichkeiten ausgewählt hat zeigt

das Programm auch mehrere Graphen an, wobei jeder durch seinen Namen gekennzeichnet wird. Mit dieser Funktion des Programms kann man sehr gut eine Laufzeitanalyse machen, da man mehrere Algorithmen oder auch die Speicherart oder die Sortieralgorithmen vergleichen kann. Jedoch müssen hierbei mehrere Sachen beachtet werden. Zunächst sollte man nicht zu viele Sachen auswählen, da das Koordinatensystem zu unübersichtlich wird und es dann schwer ist mehrere Graphen zu vergleichen. Des Weiteren sollte man den GreedyAlgorithmus nicht mit den anderen Algorithmen an sich vergleichen, da bei diesem bei einer Menge von 27 Objekten, welches ich als das Maximum bei akzeptabler Laufzeit der anderen Algorithmen schätzen würde (bei meinem Computer), die Laufzeit so gering ist, dass ich den GreedyAlgorithmus extra betrachtet habe und bei diesem eigene Stauchungen und Streckungen der Achsen verwendet habe, um die Unterschiede in dem Algorithmus grafisch zu erkennen. Also, wie schon gesagt, liegt meiner Meinung nach das Maximum für die ersten zwei der drei Algorithmen bei etwa 27 vielleicht auch 28 Objekten. Jedoch dauert alles darüber hinaus eine lange Zeit, da die Laufzeit mit jedem neu hinzugefügten Objekt verdoppelt wird. Wohingegen beim GreedyAlgorithmus bei mir durchaus bis zu 3500 Objekte verwendet werden können. Wenn man nun die Seite wieder verlassen möchte muss man auf das normale Schließen des Fensters zurückgreifen, also der Button mit dem Kreuz oben rechts am Rand. Daraufhin wird wieder das 2. Panel der Startseite angezeigt, von dem man beliebig weiter probieren kann.

4.2 Erklärung der wichtigsten Implementations Schritte und Probleme

Zunächst brachte das Programm mich zu der Frage, wie ich die Einstellungen, welche auf der Einstellungsseite ausgewählt werden, an die Hauptseite während der Laufzeit der beiden übergebe. Eine Möglichkeit wäre die Hauptseite zu schließen und die ausgewählten Optionen mit Hilfe von Argumenten der Main-Methode zu übergeben, jedoch muss man dafür das Hauptfenster zunächst schließen und danach wieder öffnen, und das will ich nicht, da es zu viel Schließen und Öffnen von Fenstern bewirkt und das Programm zu dynamisch wirken lässt. Eine weiter möglichkeit wäre es eine Datei zu erstellen, die Werte zu speichern und dann wieder auszulesen, aber ich

Entschied mich dafür den Einstellungen das Objekt Startseite selber zu übergeben um danach auf diesem zu arbeiten:

```
private void jButtonEinstellungenActionPerformed(java.awt.event.ActionEvent evt) {
    Einstellungen.main(args, this);
    list1.removeAll();
    list2.removeAll();
    textFieldMaxGewicht.setText("");
    textFieldMaxWert.setText("");
    textFieldLaufzeit.setText("");
}
```

Das ist hier durch den Aufruf „Einstellungen.main(args, this)“ dargestellt, der Einstellungsmethode wird also das Objekt selber übergeben mit einem String args, welcher in diesem Fall leer ist. Daraufhin werden die Inhalte beider Listen gelöscht und die vorher dargestellten Texte auf den Textfeldern zurückgesetzt. In den Einstellungen wird das Objekt nun als Startseite gespeichert. Bei diesem werden dann nach Drücken des Starbuttons die ausgewählten jButtons ausgelesen und die Variablen auf der Startseite je nach Auswahl definieren, also den Algorithmus, den Speichermodus, die Anzahl, die Größe und den Sortieralgorithmus. Sollte eines der Felder nicht ausgewählt sein wird die entsprechende Fehlermeldung auf das Textfeld geschrieben und es passiert sonst nichts. Auf der Startseite werden dann die Daten mit Hilfe der Algorithmen ausgewertet, also die Menge an Objekten ausgewählt, welche den maximalen Wert bringen. Dazu werden die Klassen der jeweiligen ausgewählten Options Schritte aufgerufen, welche jedoch nur eine Implementation der anfangs genannten Algorithmen darstellen und keine großen Probleme bei der Implementation hervorrief.

Das einzige Problem, was dabei entstand, ist die Option des Rekursiven Arrays. Ich habe zwar im Internet eine Vielzahl von möglichen Ansätzen gesehen, bei dem nur ein Integer Wert berechnet wird, also der maximal mögliche Wert, dabei konnte man aber nicht die Objekte auslesen. Deshalb habe ich mir überlegt als Rückgabetyp der Methode ein float Array zu verwenden, wie das, in dem auch die anderen Objekte gespeichert wurden. Dabei muss man zwischen zwei Fällen unterscheiden, und zwar ob das Objekt eingepackt wird oder nicht. Sollte es nicht eingepackt werden ruft man einfach wieder die Methode auf, jedoch jetzt mit erhöhter Anzahl. Dazu wird jedes Mal ein Integerwert übergeben, welcher am Anfang 0 ist und mit jedem Aufruf um 1

inkrementiert wird. Das wird gemacht um nicht über die maximale Anzahl der Objekte hinaus aufzurufen und dient als Abbruchbedingung, also wenn der Integerwert i größer ist als die Anzahl der Objekte, die man hat, gibt man ein leeres Array zurück. Man darf hierbei nicht null zurückgeben, da in einem zweiten Rekursionsaufruf dem Array b das zurückgegebene Array des Rekursionsaufrufes angehängt wird, das kann jedoch nicht null sein. Zudem werden Werte zwischen dem Array a und b verglichen, wobei man auf Werte in dem Array zugreifen muss, und wenn dieses null ist gibt es eine Fehlermeldung.

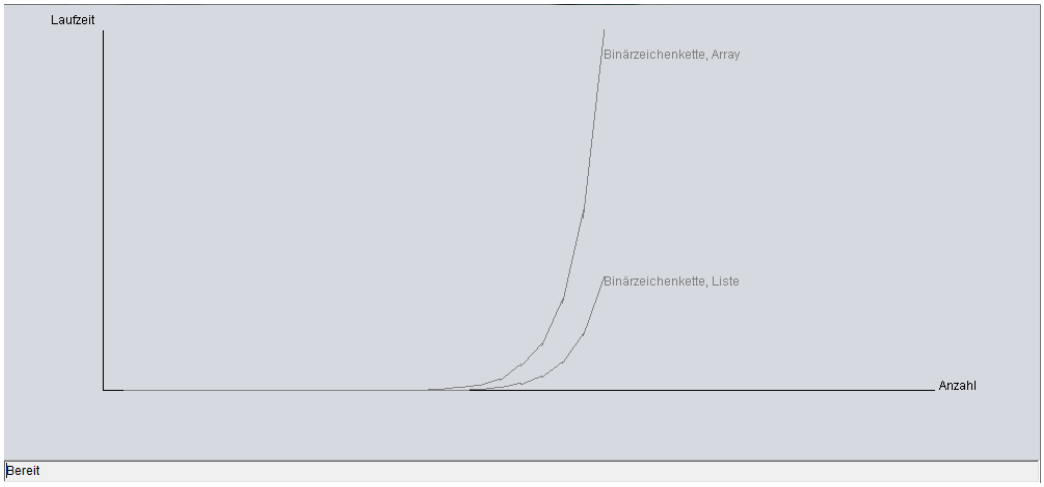
Ich habe kein Tool für netbeans gefunden, welches einen Graphen zeichnen kann, wodurch ein weiteres Problem der Implementation von der Ausrichtung des Graphen handelt. Das erste Problem, welches sich ergibt, ist, dass bei Java Graphics das Koordinatensystem nicht, wie normalerweise, unten rechts beginnt, sondern oben links in der Ecke, wobei die Y-Achse senkrecht herunter geht und die X-Achse waagerecht nach rechts verläuft. Es ist also aufgebaut wie der 4. Quadrant im bekannten Koordinatensystem, nur dass die Y-Werte positiv sind. Dabei habe ich mir zunächst geeignete Koordinatenachsen gezeichnet, wobei man darauf achten muss, dass bei der Graphics Funktion von Java mit der Anweisung „g.drawLine“ eine Linie gezogen wird, bei der die Koordinaten des Anfangspunktes also X1 und Y1 an den ersten beiden Stellen stehen und die des Ausgangspunktes X2 und Y2 an dritter und vierter Stelle stehen. Also: g.drawLine(X1,Y1,X2,Y2). Das g steht hierbei für die Grafik, sie wird also dazu aufgefordert die Linie zu ziehen. Des Weiteren musste ich einen geeigneten Faktor für die Laufzeit und die Anzahl zu finden, damit man auch noch etwas erkennen kann. Dafür habe ich den Graphen der einzelnen Funktionen mit dem Faktor 20 in X-Richtung gestreckt und mit einem Divisor von j*3 gestaucht. J steht hierbei für eine Variable, welche mit der gerade durchgegangen Anzahl identisch ist, also wird er bei 10 Objekten durch 30 geteilt. Der Vergleich der einzelnen Graphen wird dadurch nicht falsch, da jeder der darstellbaren Grafen gleich behandelt wird, also jeder X-Wert mit 20 multipliziert und jeder Y-Wert durch j*3 geteilt.

Als letztes Problem der Facharbeit lässt sich die Laufzeitanalyse der Sortieralogrithmen nennen. Dabei sind die wichtigen Algorithmen der Laufzeitanalyse der Selection,

Insertion und Bubble-sort und der Quicksort. Als erstes ergab sich das Problem, dass der Graph nicht richtig dargestellt wurde, sondern verzogen und mit vielen Ungenauigkeiten. Dazu habe ich einige Analysen laufen lassen, wie beispielsweise Mittelwertbildung vieler Durchläufe verschiedener Algorithmen und die Anzeige der Laufzeit bei vielen Algorithmen um Ungenauigkeiten zu erkennen. Dabei bin ich letztendlich zu dem Ergebnis gekommen, dass dies zum einen an der Ungenauigkeit der Laufzeitanalyse mit dem System.currenTimeMillis Befehl kommt, aber auch von der Auslastung des Ausgangssystems. Letzteres habe ich herausgefunden, indem ich eine Methode zunächst alleine und dann 2 mal parallel ausgeführt habe, die nacheinander die Laufzeit für das Auswählen von Objekten mit dem gleichen Array anzeigt. Zu der Ungenauigkeit des „System.currentTimeMillis()“ Befehls bringen mich mehrere Fakten. Zunächst sollte man beachten, dass die Laufzeit des Algorithmus in den einzelnen Schritten so klein ist, dass es gerade im Bereich von 0 – 1000 Objekten zu starken Fehlern im Zusammenhang mit dem Befehl kommt, da dort die Laufzeit im knappen Nanosekundenbereich verläuft, und auch bei 3000 Objekten erst bei ungefähr 40 Millisekunden verläuft. Deshalb habe ich daraufhin den System.nanoTime() Befehl verwendet, welcher im Vergleich ein eindeutig besseres Bild abgibt, wobei der einzige Nachteil bei diesem eine relativ lange Auslesezeit ist. Der zweite Teil des letzten Problems ist, dass die Sortieralgorihtmen eine eindeutig unterschiedlich lange Sortierzeit haben, was nur insofern ein Problem darstellt, als dass uns in der Schule beigebracht wurde, dass diese Algorithmen eine etwa gleich lange Laufzeit zu verzeichnen haben bei gleichen Ausgangsbedingungen. Das verwirrt zunächst, da auch im Internet keine Seite zu finden ist, auf der die Laufzeit der 3 Sortieralgorithmen miteinander vergleichen wird. Deshalb habe ich mir die Klassen, in welchen wir in der Schule die Sortieralgorithmen implementiert haben, näher angeschaut und dort eine Laufzeitanalyse eingebaut, die jeweils die 3 Algorithmen das gleiche Array 1000-mal sortieren lässt und dabei die Laufzeit misst, und am Ende einen Mittelwert bildet. Dabei komme ich jedoch auch auf unterschiedliche Laufzeiten, wodurch das Problem, was dann genaugenommen gar kein Problem ist, gelöst wird, also die Sortieralgorithmen doch eine unterschiedliche Laufzeit haben.

5 Laufzeitanalyse

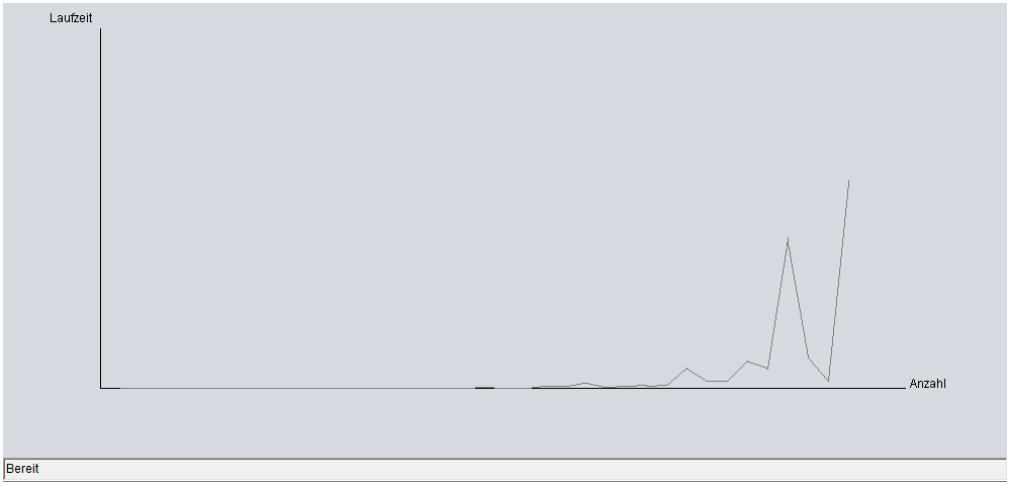
Nun, die Laufzeitanalyse stellt eines der wichtigsten Attribute des Programms dar, da es auch eines der wichtigsten Bestandteile des Rucksackproblems darstellt. Hierbei ist jedoch zu beachten, dass der Greedy-Algorithmus keine optimale Darstellung des Rucksackproblems darstellt. Er ist zwar sehr schnell, auch bei vielen Objekten, jedoch, wie bereits gesagt, bildet er nicht den best Case. Dahingegen kann man den Binären wie auch den Rekursiven sehr gut miteinander vergleichen. Wenn man das folgende Koordinatensystem betrachtet sieht man den Vergleich von dem Binären Algorithmus mit einer Liste und einem Array:



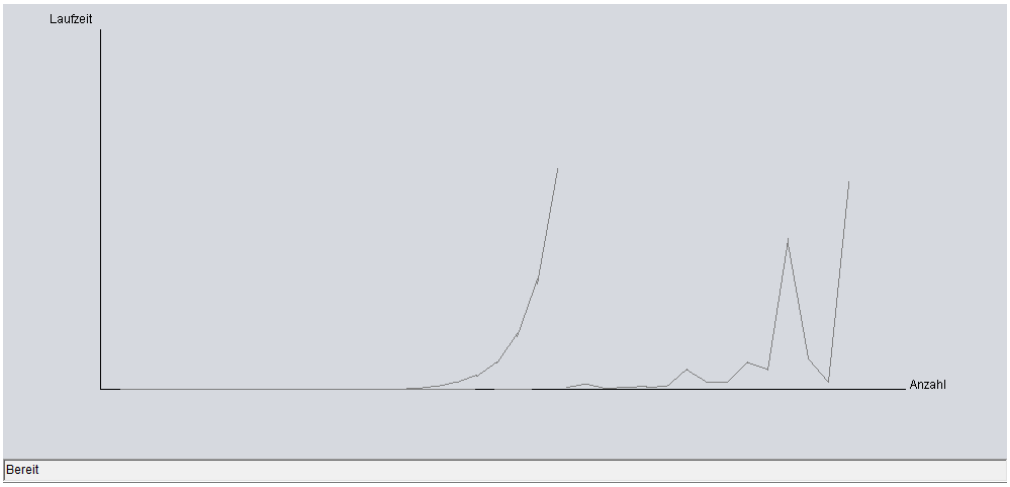
Man sieht, dass die Liste als Speichertyp der Objekte um einiges schneller ist. Es braucht nur in etwa ein Viertel im Vergleich zum Array zum Einsortieren der 25 Objekte. Zunächst stellt man sich die Frage, weshalb das so ist. Diese lässt sich in diesem Fall relativ einfach beantworten, indem man sich zunächst einmal klar macht was die Vor- und Nachteile einer Liste, beziehungsweise eines Arrays, sind. Eine Liste ist beispielsweise eine dynamische Datenstruktur, was heißt, ihre Größe ist während der Laufzeit veränderbar. Wohingegen ein Array statisch ist, also man muss einmal eine Größe dieses festlegen und diese kann nicht mehr verändert werden. Wenn man sich diese Tatsache klar gemacht hat und sich dann die Methode anschaut ist es eigentlich klar, dass das Array um einiges länger braucht als die Liste. In der Binärzeichenketten Methode wird zum Abrufen der Daten aus einer Binärzeichenkette die Methode gibDaten aufgerufen, in welcher eine neue Liste oder auch Array angelegt wird. Diese wird hier angelegt um aus den Daten der Kombination, also den Einsen

oder Nullen, diejenigen Objekte auszulesen, welche eingefügt werden und welche nicht. Nun weiß man aber vorher nicht welche Objekte eingefügt werden sollen oder welche nicht, da ja die 1 und 0 Folge jedes Mal anders ist. Dadurch muss man bei der Variante des Arrays ein neues Array erzeugen, welches die Größe der gesamt möglichen Anzahl hat, wodurch in sehr vielen Fällen einiges an unnötigem Speicher verwendet wird, da viele Stellen des Array leer bleiben. Dahingegen muss man bei der Liste also zu Beginn keine feste Größe festlegen, wodurch viel Laufzeit in diesem Fall eingespart wird. Man kann dies auch nachweisen, dazu habe ich in der Methode gibDaten eine Zeitmessung implementiert, welche die Zeit jeweils vom Aufruf der Methode bis zum Beenden der Methode misst und jedes Mal die Zahlen addiert. Hierbei kommen bei 25 Objekten für das Array auf eine Laufzeit von etwa 47 Sekunden bei einer Gesamtlaufzeit von ungefähr 52 Sekunden. Bei der Liste waren es jedoch nur 9 Sekunden bei einer gesamten Laufzeit von 14 Sekunden. Das zeigt zum einen, dass vergleichsweise dieselbe Zeit zum Anlegen des Randomtypes und zum Füllen der Kombination Strings gebraucht werden, was auch klar ist, da sich diese Methoden in den beiden Klassen kaum unterscheiden. Zum anderen beweist es aber auch, dass der Hauptunterschied der beiden bei dieser Methode liegt. Dabei liegt die Differenz bei ungefähr 4/5, also die Laufzeit der Methode des Arrays ist im Durchschnitt um 500% länger als die der Liste. Dazu zu sagen ist noch, dass die Objekte, welche einsortiert werden, keinen Unterschied bei der Laufzeit machen, sowie auch die Größe des Behälters in welchem einsortiert wird, da in keinem Schritt mit dieser verglichen wird und alle möglichen Kombinationen durchgegangen werden, unabhängig von dem Wert oder Gewicht der Objekte.

Dahingegen spielen diese Faktoren eine sehr große Rolle bei dem Rekursiven Algorithmus, wie der folgende Graf zeigt:



Der Graph scheint beinahe zufällig zu steigen und zu fallen, jedoch liegt das an der angegebenen Größe und den Werten, welche erzeugt werden. Wenn es sehr große Werte im Bereich der Größe eines Objektes gibt, dauert der Algorithmus meist kürzer, als wenn man kleinere Größen bei den Objekten oder eine größere Größe bei dem Behälter verwendet. Wie ich im früheren Verlauf der Facharbeit bereits gesagt habe, liegt das an der Abfrage, ob das Objekt noch hineinpasst oder nicht. Um zu verdeutlichen, dass es wirklich daran liegt, habe ich hier einmal diese Abfrage aus dem Code gelöscht und es bildet sich etwa der gleiche Graf wie bei dem Binären Algorithmus:

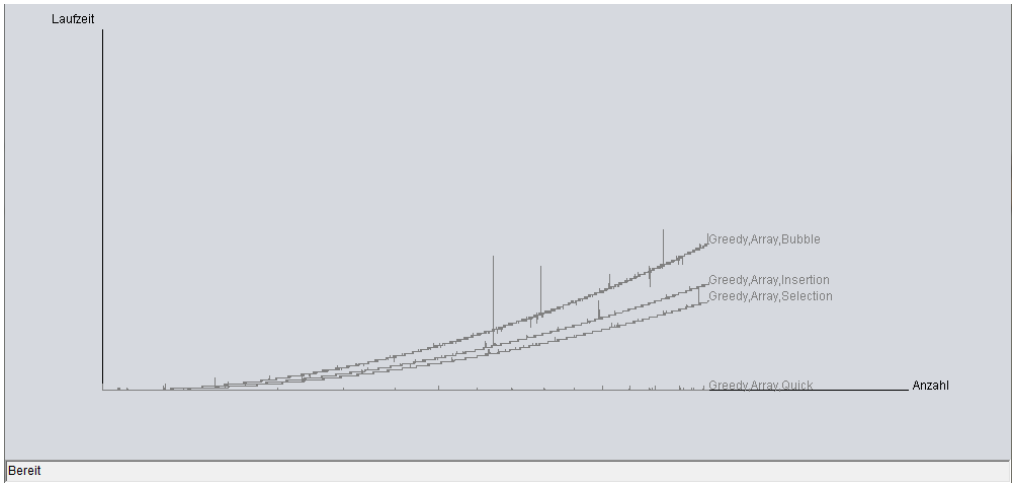


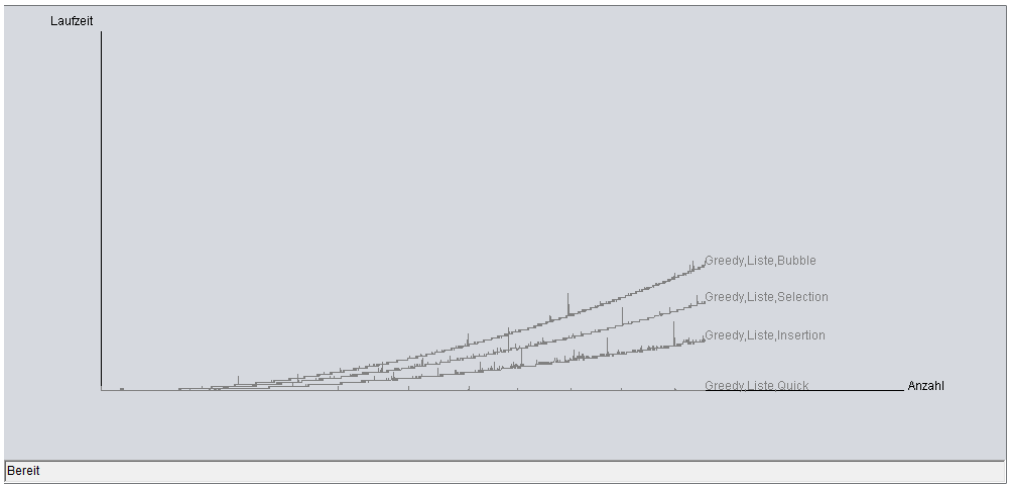
Das es fast der gleiche Graph ist wie bei der Binären Variante hängt damit zusammen, dass nun auch beim Rekursiven Algorithmus jede Variante durchgegangen werden muss und dadurch auch die Laufzeit von 2^n erreicht wird. Um letztendlich aber doch eine genaue Laufzeitanalyse des anfänglichen Rekursiven Algorithmus zu erhalten kann man, durch viele Messungen und Mittelwertbildungen dieser auch bei diesem eine

Laufzeit zum Vergleichen ermitteln. Dafür habe ich eine extra Klasse geschrieben, welche bei einem Array oder Liste mit 30 Objekten die optimalen Objekte von dem Rekursiven Algorithmus auswählen lässt und dabei die Laufzeit misst. Dies macht sie x-mal und teilt die gesamte Laufzeit daraufhin durch die Anzahl, dadurch erlangt man den Mittelwert. Bei dem Rekursiven Algorithmus mit einem Array mit einem Mittelwert von 400 Aufrufen kommt eine durchschnittliche Laufzeit von 2111 Millisekunden, also 2,111 Sekunden heraus, dahingegen hat man bei der Liste eine durchschnittliche Laufzeit von 20 Millisekunden, also 0,02 Sekunden. Es zeigt sich also, auch hier ist die Liste um einiges effektiver als das Array. Die Gründe sind fast dieselben wie auch bei dem Binären Algorithmus. In der Rekursiven Methode muss das zurückgegebene Array dem bereits bestehenden angehängt werden, in das das Objekt, welches in diesem Fall eingepackt wird, hinzugefügt wird. Dieses Objekt kann aber nicht in ein Array mit null Referenz eingefügt werden, weshalb man wieder ein neues Array erzeugen muss, welches, wie ich schon bei der Fehlerbehebung gesagt habe, die maximal mögliche Anzahl haben muss. Bei der Liste ist dies wieder relativ schnell erledigt, da diese ja dynamisch ist und dadurch die Objekte einfach angehangen werden können, wodurch man viel Speicher und Laufzeit spart.

Zuletzt gibt es noch den GreedyAlgorithmus, bei welchem einige Probleme, die ich bereits beschrieben habe, sehr viel Zeit in Anspruch genommen haben. Seine Laufzeit ist im Vergleich zu den anderen sehr gering. Da die Laufzeit der anderen Algorithmen durchschnittlich exponentiell steigen würde, habe ich mir ausgerechnet, dass der BinäreAlgorithmus bei 100 Objekten schon eine Laufzeit von etwa $598789{,}9*10^{12}$ Jahren hätte, wohingegen der GreedyAlgorithmus eine Laufzeit von etwa 2 Millisekunden hat. Nun, bei diesem kann man die einzelnen Sortieralgorithmen vergleichen, sowie zwischen Liste und Array, welche aber hierbei zunächst relativ wenig Unterschied zu machen scheinen. Zwischen den einzelnen Sortieralgorithmen gibt es aber einen relativ großen Unterschied. Dabei ist der Quicksort der schnellste, mit einer Laufzeit, welche Proportional zu $n*\log(n)$ steigt. Der Bubblesort braucht durchschnittlich etwa $0.5 * n^2$ Vergleiche und $0.25 * n^2$ Vertauschungen, wohingegen der Selectionsort etwa $0.5 * n^2$ Vergleiche, aber nur n Vertauschungen braucht. Zuletzt gibt es noch den Insertionsort, der in etwa $0.25 * n^2$ Vergleiche und $0.125 * n^2$ Vertauschungen benötigt. Wenn man sich die Anzahl der Vergleiche ansieht, wird

direkt klar, dass der Bubblesort der Langsamste sein wird, da er in mindestens einer Hinsicht schlechter als die anderen Sortieralgorithmen ist. Dabei benötigt dieser gleich viele Vergleiche wie der Selectionsort aber dafür 0.25*n mal mehr Vertauschungen. Der Insertionsort braucht in beiderlei Hinsicht weniger als der Bubblesort, aber der Vergleich zwischen Insertion und Selectionsort fällt ein wenig schwerer aus. Der Selectionsort benötigt zwar doppelt so viele Vergleiche, jedoch etwa n*0.125 Vertauschungen weniger.[9] Welcher der beiden der bessere ist hängt also davon ab, wie schnell verglichen oder vertauscht werden kann, wobei der Selectionsort bei schnellem Vergleichen und der Insertionsort bei schnellem Vertauschen besser ist. An den zwei folgenden Grafiken erkennt man diese Eigenschaften exemplarisch:





[9]Vgl. http://www.gym1.at/informatik/fachdidaktik/suchsort/index.htm zuletzt aufgerufen am 29.03.2016

Es werden jeweils 4 Graphen gezeigt, bei denen bis 3000 Objekte verglichen werden bei den jeweiligen Sortieralgorithmen. Das erste Bild zeigt die Sortierung auf einem Array und das zweite auf einer Liste. Jetzt muss man sich noch einmal an die Eigenschaften des Arrays und der Liste erinnern, welche ich im vorherigen Verlauf bereits genannt habe. Wenn man zunächst auf die Vergleiche achtet fällt als Unterschied auf, dass man beim Array die Werte direkt abrufen kann, das geht auch bei der Arrayliste direkt, jedoch muss dort aus dem Objekt noch der Wert bestimmt werden. Es gibt aber auch einen Unterschied im Tauschen der Objekte. Nun, beim Array werden zunächst einmal immer 2 Objekte bei einem Tauschvorgang getauscht und es müssen auch immer 2 Objekte temporär gespeichert werden. Das liegt an der Struktur des Arrays, das durch die 2 Werte, welche gebraucht werden, 2 Dimensional ist, also eine Dimension für den Wert und eine für das Gewicht. Dahingegen wird bei einer Liste immer nur ein Objekt gespeichert und getauscht. Also ist hier der Aufwand geringer als bei dem Array. Dadurch ist auch eindeutig, dass der Insertionsort bei der Liste der schnellste ist, da man bei der Liste schneller tauschen kann und der Selectionsort bei dem Array, da dort das Auslesen der Werte und dadurch der Vergleich schneller ist. Es zeigt auch, dass der Bubblesort der langsamste ist, was auch vorher schon durch die durchschnittliche Laufzeit angegeben wurde und der Quicksort im allgemeinen der schnellste ist, wobei es, wie gesagt, keinen sichtbaren Unterschied zwischen Array und Liste gibt. Deshalb habe ich auch hierfür eine Methode geschrieben, welche den Durchschnitt für 10000 sortierende Objekte berechnet, wo bei dem Array etwa 5,7 Sekunden und bei der Liste 6,4 Sekunden gebraucht werden. Das zeigt, dass hier das Array auf lange Sicht besser geeignet ist, als die Liste, da es bei 10000 Objekten bereits einen Unterschied von etwa 0,7 Sekunden gibt.

6 Fazit

Abschließend würde ich meine Facharbeit trotz einiger Probleme als erfolgreich einschätzen. Es hat sich bei Behandlung des Rucksackproblems gezeigt, dass auch ein anfangs relativ einfach scheinendes Problem sehr komplex sein kann. Es gibt mehre Algorithmen zur Behandlung dieses, wobei ich drei Algorithmen verwendet habe welche man verschieden Implementieren kann. Das Programm, welches ich programmiert habe, ist bei Fallanwendung oder auch der Theoretischen Analyse ein sehr hilfreiches Werkzeug, da man sehr gut die Unterschiede der Algorithmen, Speicherarten, Objektanzahlen oder Speichergrößen sehen kann. Als effektivster Algorithmus hat sich meiner Meinung nach der Greedy Algorithmus mithilfe des Quicksorts auf einem Array herausgestellt, da dieser, trotz der relativen Ungenauigkeit des Ergebnisses eine sehr gute Laufzeit hat und der Effektivste ist. Möchte man jedoch ein genaues Ergebnis, ist der Rekursive Algorithmus auf einer Liste am besten geeignet.

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

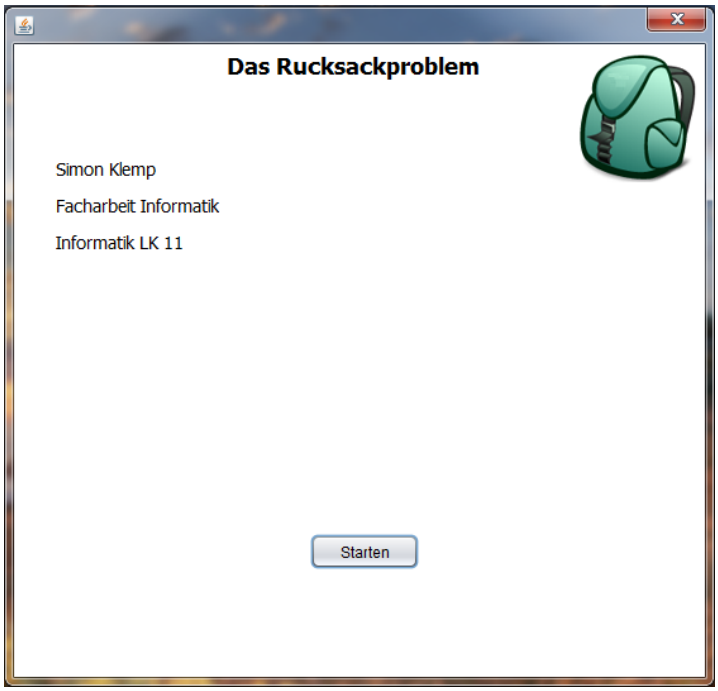
Literaturverzeichnis

- **Gallenbacher, J.(2008) Abenteuer Informatik, Spektrum Akademischer Verlag; Auflage: 2. Aufl. Juni 2008**
- **Fuchs, E.(2014) Java 8 Grundlagen Programmierung, Herdt Verlag; Ausgabe: 1.Aufl. Oktober 2014**
- **Befehle zu folgenden Java API gesucht:**
    - **java.util.ArrayList:**
    - **https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **java.awt.BorderLayout:**
    - **https://docs.oracle.com/javase/7/docs/api/java/awt/BorderLayout.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **java.awt.Color:**
    - **https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **java.awt.Graphics:**
    - **https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **java.text.NumberFormat:**
    - **https://docs.oracle.com/javase/7/docs/api/java/text/NumberFormat.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **java.text.DecimalFormat:**
    - **https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **java.math.RoundingMode:**
    - **https://docs.oracle.com/javase/7/docs/api/java/math/RoundingMode.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
    - **javax.swing:**
    - **https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html**
    - **Zuletzt Aufgerufen am 03.04.2016**
- **https://de.wikipedia.org/wiki/Rucksackproblem#Mathematische_Formulierung**
    - **Zuletzt Aufgerufen am 29.03.2016**
- **https://www.proggen.org/doku.php?id=algo:knapsack**
    - **Zuletzt Aufgerufen am 29.03.2016**
- **https://de.wikipedia.org/wiki/NetBeans_IDE**
    - **Zuletzt Aufgerufen am 29.03.2016**
- **http://www.inf-schule.de/grenzen/komplexitaet/rucksackproblem**
    - **Zuletzt Aufgerufen am 29.03.2016**

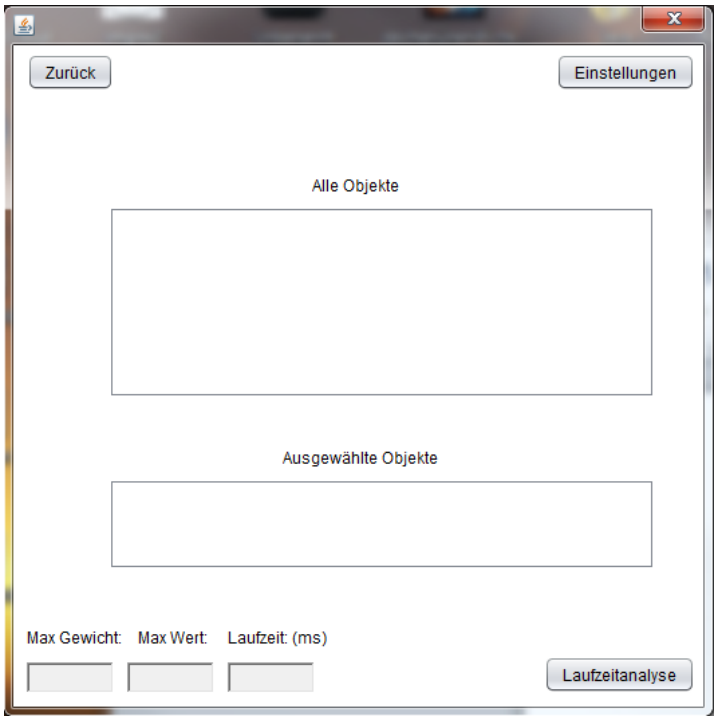
- **http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo15.php**
  - **Zuletzt Aufgerufen am 29.03.2016**

- **http://parco.iti.kit.edu/henningm/Seminar-AT/seminar-arbeiten/Schmitz_final.pdf**
  - **Zuletzt Aufgerufen am 29.03.2016**
- **https://de.wikipedia.org/wiki/Java_(Programmiersprache)**
  - **Zuletzt Aufgerufen am 29.03.2016**
- **http://www.tinohempel.de/info/info/ti/rucksackproblem.htm**
  - **Zuletzt Aufgerufen am 29.03.2016**
- **http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/gif/gif_26.pdf**
  - **Zuletzt Aufgerufen am 29.03.2016**
- **http://th.informatik.uni-mannheim.de/teach/Alg-0910/material/lecture_knapsack.pdf**
  - **Zuletzt Aufgerufen am 29.03.2016**
- **https://de.wikipedia.org/wiki/BlueJ**
  - **Zuletzt Aufgerufen am 29.03.2016**
- **https://de.wikipedia.org/wiki/Rucksackproblem**
  - **Zuletzt Aufgerufen am 29.03.2016**

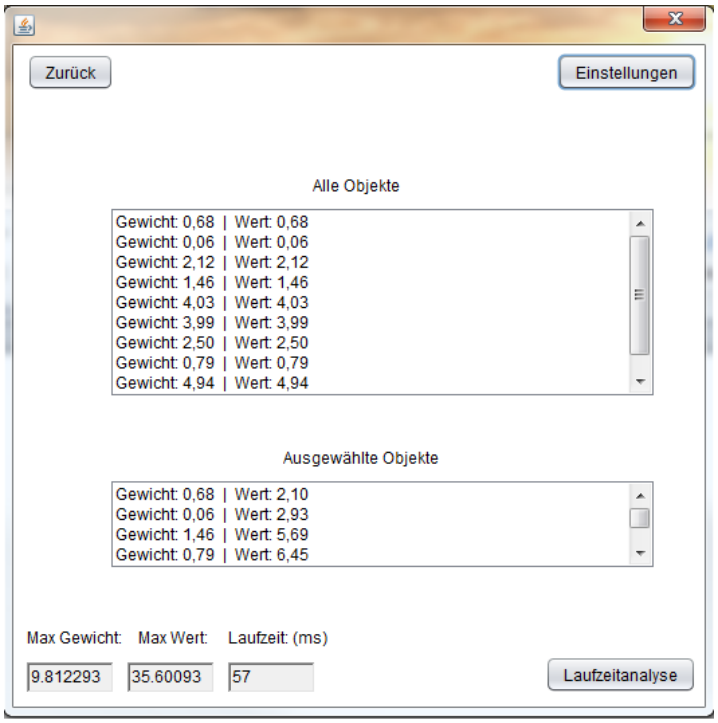
Anhang:



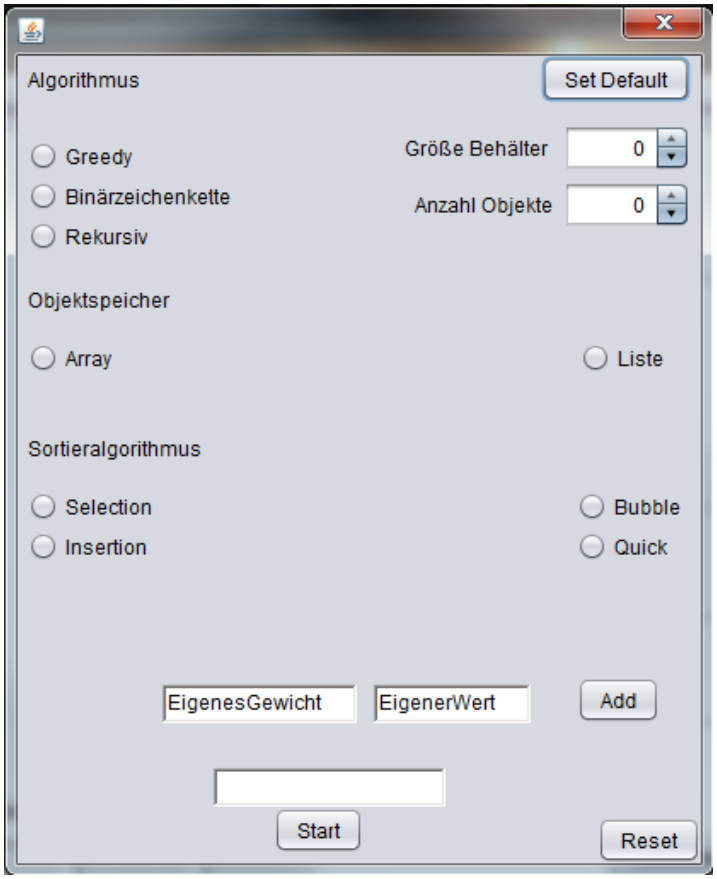
-Startseite des Programms



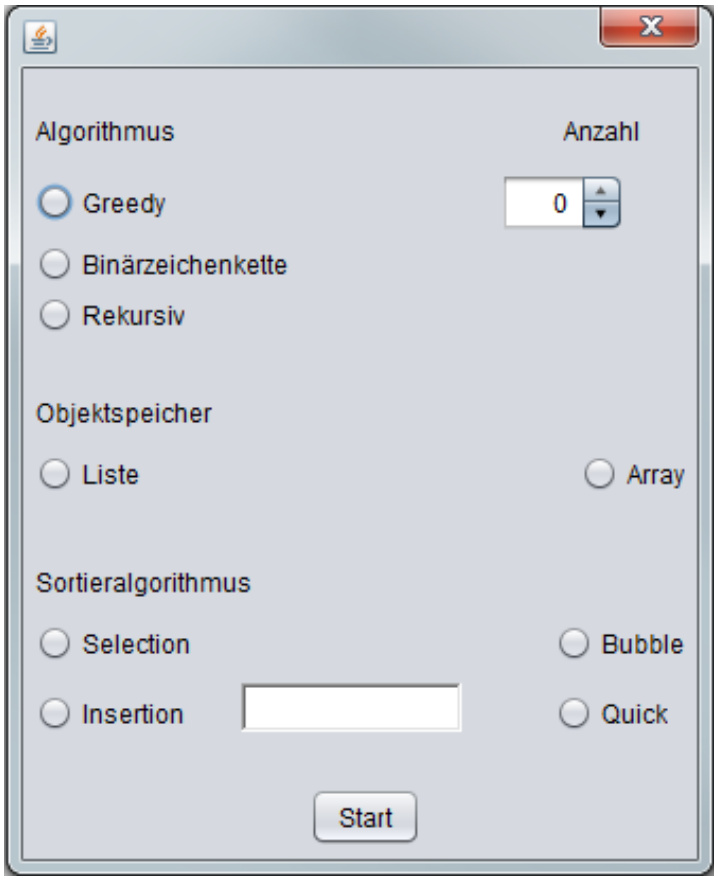
-Mainpage des Programms



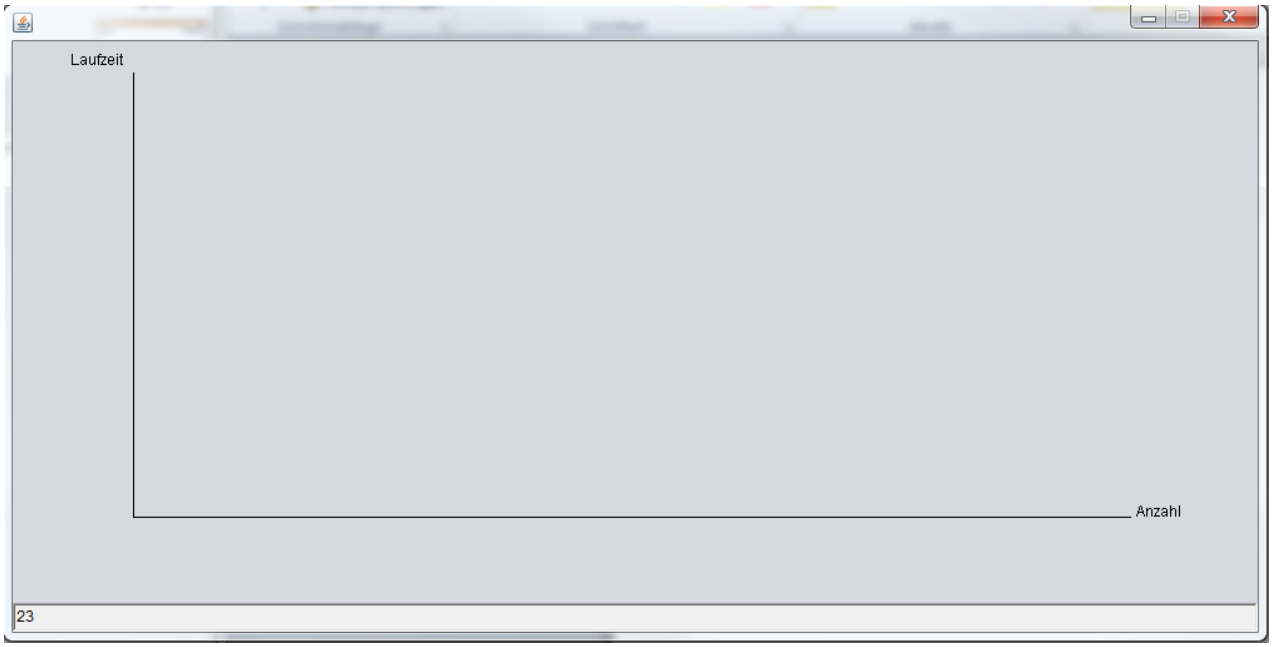
-Mainpage mit Inhalten gefüllt



-1. Einstellungsseite

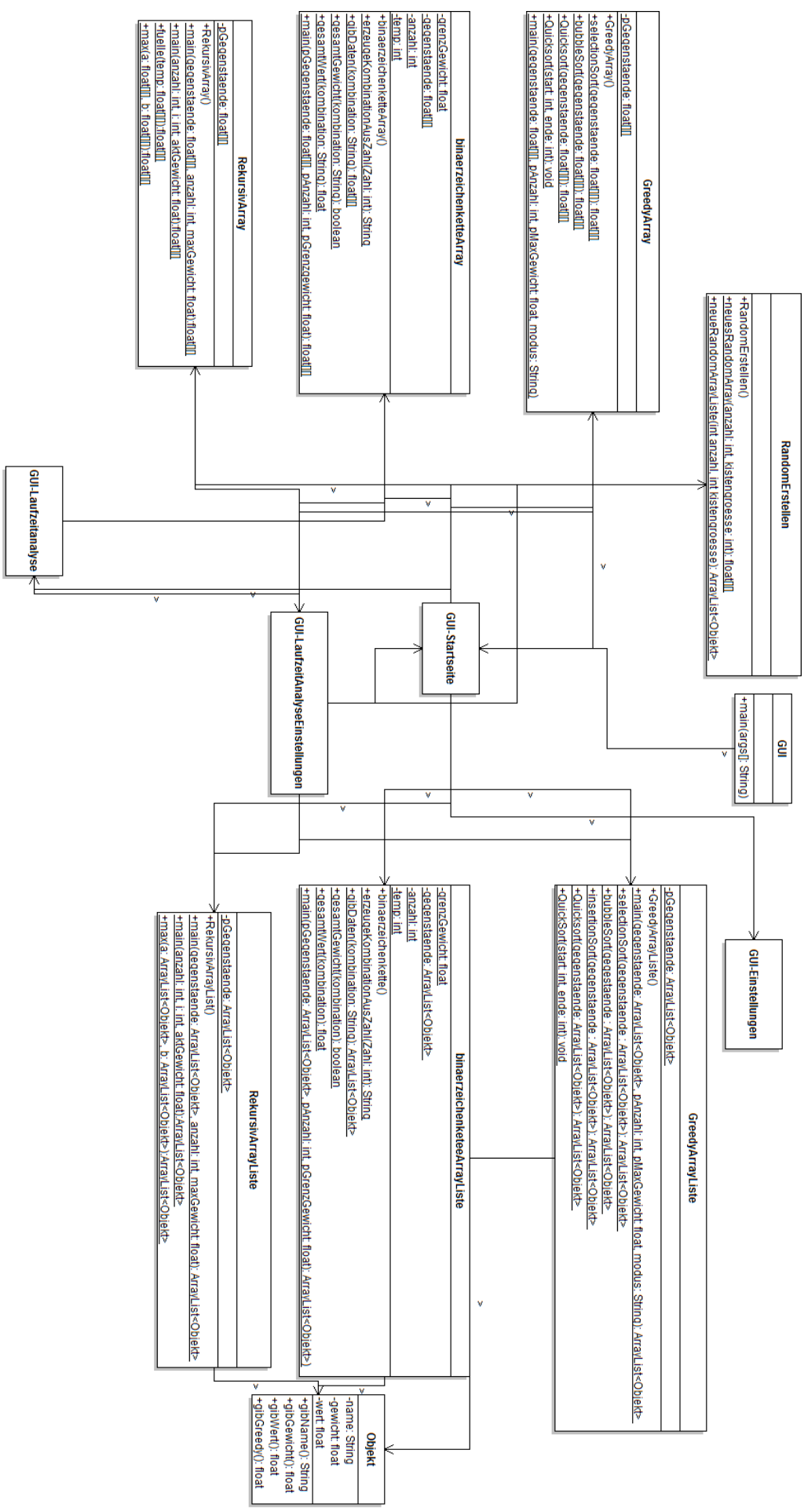


-2. Einstellungsseite



-Laufzeitanalyse Ohne Graphen

-UML Klassendiagramm:


RandomErstellen
+RandomErstellen()
+neuesRandomArray(anzahl: int, kistengroesse: int): float[]
+neueRandomArrayListe(int anzahl, int kistengroesse): ArrayList<Objekt>
GreedyArray
-pGegenstaende :float[][]
+GreedyArray()
+selectionSort(gegenstaende :float[][]): float[][]
+bubbleSort(gegenstaende :float[][]): float[][]
+Quicksort(gegenstaende :float[][]): float[][]
+Quicksort(start int, ende :int): void
+main(gegenstaende :float[][], pAnzahl: int, pMaxGewicht :float, modus: String)
binaerzeichenketteArray
-grenzGewicht :float
-gegenstaende :float[][]
-anzahl: int
-temp: int
+binaerzeichenketteArray()
+erzeugeKombinationAusZahl(Zahl: int): String
+qibDaten(kombination :String[]): float[][]
+gesamtGewicht(kombination :String[]): boolean
+gesamtWert(kombination :String[]): float
+main(pGegenstaende :float[][], pAnzahl: int, pGrenzgewicht :float): float[][]
RekursivArray
-pGegenstaende :float[][]
+RekursivArray()
+main(gegenstaende :float[][], anzahl: int, maxGewicht :float):float[][]
+main(anzahl: int, i: int, aktGewicht :float):float[][]
+fuelle(temp :float[][]):float[][]
+max(a :float[][], b :float[][]):float[][]
GUI-Laufzeitanalyse
GUI-Startseite
GUI-LaufzeitAnalyseEinstellungen
GUI
+main(args[]: String)
GUI-Einstellungen
GreedyArrayListe
-pGegenstaende :ArrayList<Objekt>
+GreedyArrayListe()
+main(gegenstaende :ArrayList<Objekt>, pAnzahl: int, pMaxGewicht :float, modus: String):ArrayList<Objekt>
+selectionSort(gegenstaende : ArrayList<Objekt>):ArrayList<Objekt>
+bubbleSort(gegenstaende : ArrayList<Objekt>):ArrayList<Objekt>
+insertionSort(gegenstaende :ArrayList<Objekt>):ArrayList<Objekt>
+Quicksort(gegenstaende :ArrayList<Objekt>):ArrayList<Objekt>
+QuickSort(start int, ende :int): void
binaerzeichenketteeArrayListe
-grenzGewicht :float
-gegenstaende :ArrayList<Objekt>
-anzahl: int
-temp: int
+binaerzeichenkette()
+erzeugeKombinationAusZahl(Zahl: int): String
+qibDaten(kombination :String[]):ArrayList<Objekt>
+gesamtGewicht(kombination): boolean
+gesamtWert(kombination): float
+main(pGegenstaende :ArrayList<Objekt>, pAnzahl: int, pGrenzGewicht :float):ArrayList<Objekt>
RekursivArrayListe
-pGegenstaende :ArrayList<Objekt>
+RekursivArrayList()
+main(gegenstaende :ArrayList<Objekt>, anzahl: int, maxGewicht :float): ArrayList<Objekt>
+main(anzahl: int, i: int, aktGewicht :float):ArrayList<Objekt>
+max(a :ArrayList<Objekt>, b :ArrayList<Objekt>):ArrayList<Objekt>
Objekt
-name: String
-gewicht :float
-wert :float
+qibName(): String
+qibGewicht(): float
+qibWert(): float
+qibGreedy(): float