Algorithmen und Datenstrukturen II

Arne Schmidt

Übung #1

18.04.2018

Übersicht

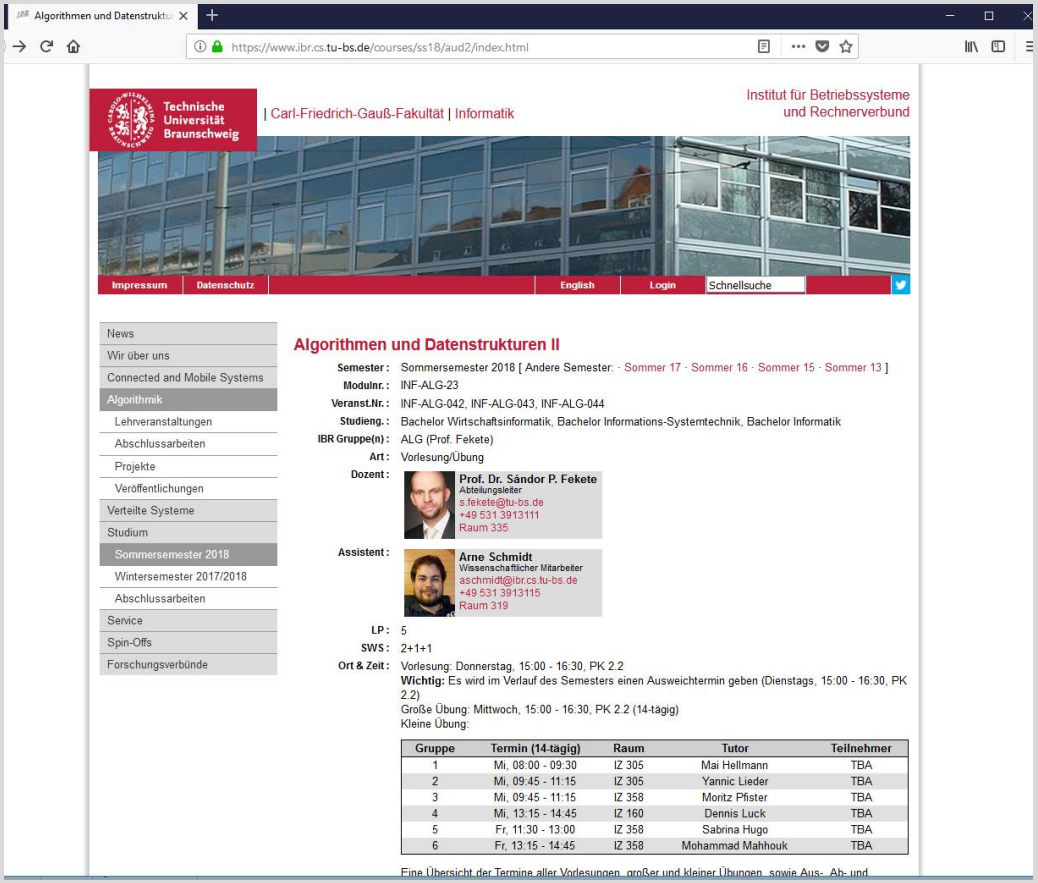
- Organisation
- Sortierverfahren
- Hausaufgaben: Programmierteil
- Datenverarbeitung

Organisation

Homepage:
https://www.ibr.cs.tu-bs.de/courses/ss18/aud2/

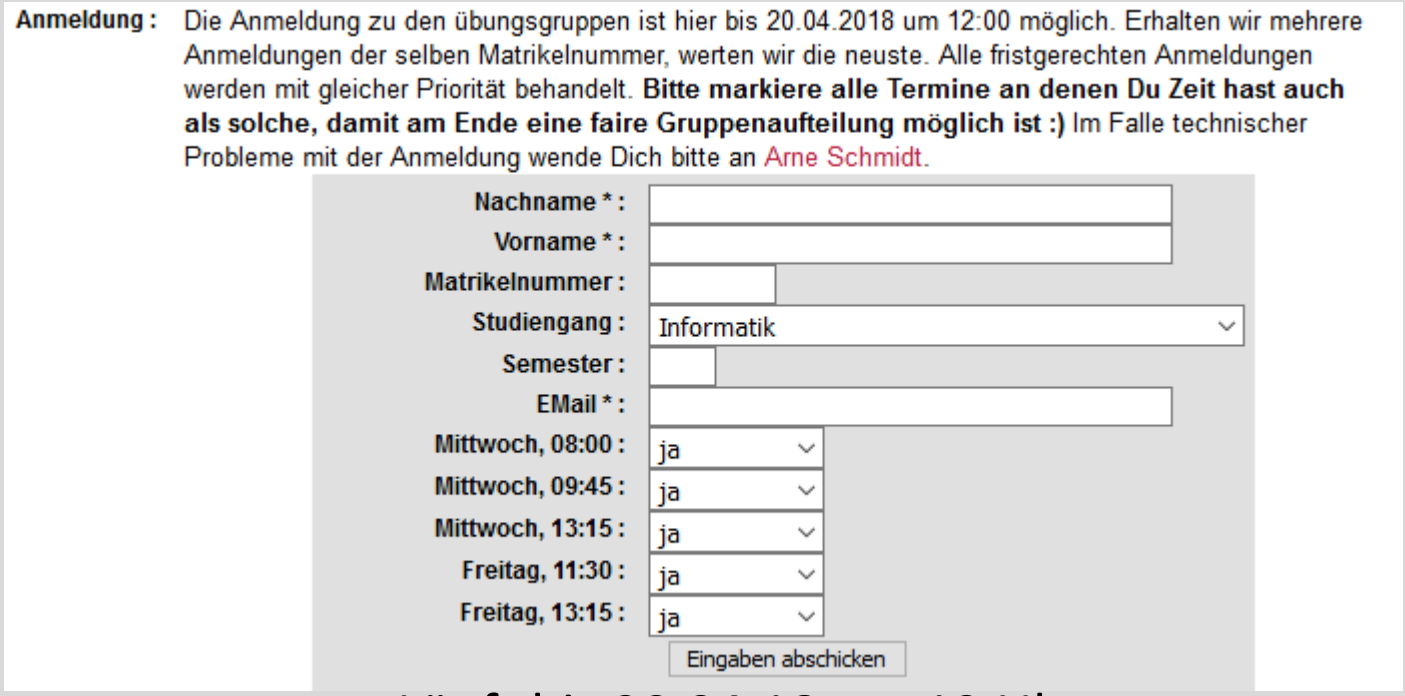
Dort gibt es:
Aktuelle Informationen,
Hausaufgaben,
Notizen zu Übungen
und Vorlesung, …

Neu: Es wird ein **Skript** geben. Das ist kein Ersatz für die Vorlesung!
Wenn Fehler gefunden werden, einfach eine Mail an mich schreiben.



Anmeldung

Anmeldung zu kleinen Übungen auf der Homepage



Läuft bis 20.04.18 um 12 Uhr.

Einteilung erfolgt voraussichtlich am selben Tag.

Semesterplan AuD II SS18

Woche	Vorlesung (Do.)	Gr. Übung (Mi.)	Kl. Übung (Mi./Fr.)	HA Ausgabe (Do.)	HA Abgabe (Do. bis 13:15 Uhr)	HA Rückgabe (in kl UE)
14	-					
15	1					
16	2	1		HA0*, HA1		
17	3**,4		1			HA0
18	5	2		HA2	HA1	
19	Christi Himmelfahrt		2			HA1
20	6	3		HA3	HA2	
21	Exkursionswoche					
22	7		3			HA2
23	8	4		HA4	HA3	
24	9		4			HA3
25	10	5		HA5	HA4	
26	11***,12		5			HA4
27	13	6			HA5	
28	TDSE	7	6			HA5

*: Mündliche Aufgaben, Bearbeitung in ersten kleinen Übungen; keine Bewertung

: Vorlesung vom 10.05.18 wird wegen Christi Himmelfahrt auf Dienstag, den **24.04.18, 15 Uhr, vorverlegt.

***: Vorlesung vom 12.07.18 wird wegen dem TDSE auf Dienstag, den **26.06.18**, 15 Uhr, vorverlegt.

Hausaufgaben I

- 6 Blätter
  - 1 unbewertet
  - 5 bewertet
- Drei Teile auf jedem Blatt
  - Allgemeiner Teil (A)
  - Theorie Teil (T)
  - Praxis Teil (P) → Programmieren der Algorithmen in Java
- Abgabe pro Blatt
  - Teil (A)
  - Teil (T) oder (P)
  - Zusammen insgesamt 30 Punkte
- Insgesamt 150 Punkte erreichbar. 50% müssen erreicht werden
- Keine Mindestpunktzahl pro Blatt!
- Abgabefrist zählt das Datum auf dem jeweiligen Blatt.

Hausaufgaben I

Hausaufgaben II

- Die Ausarbeitungen müssen **einzeln** (nicht mit Partner) abgegeben werden
- Man darf in Gruppen die Aufgaben diskutieren (und ggf. Teillösungen erarbeiten)
  - Vollständig ausgearbeitete Lösung nur alleine!

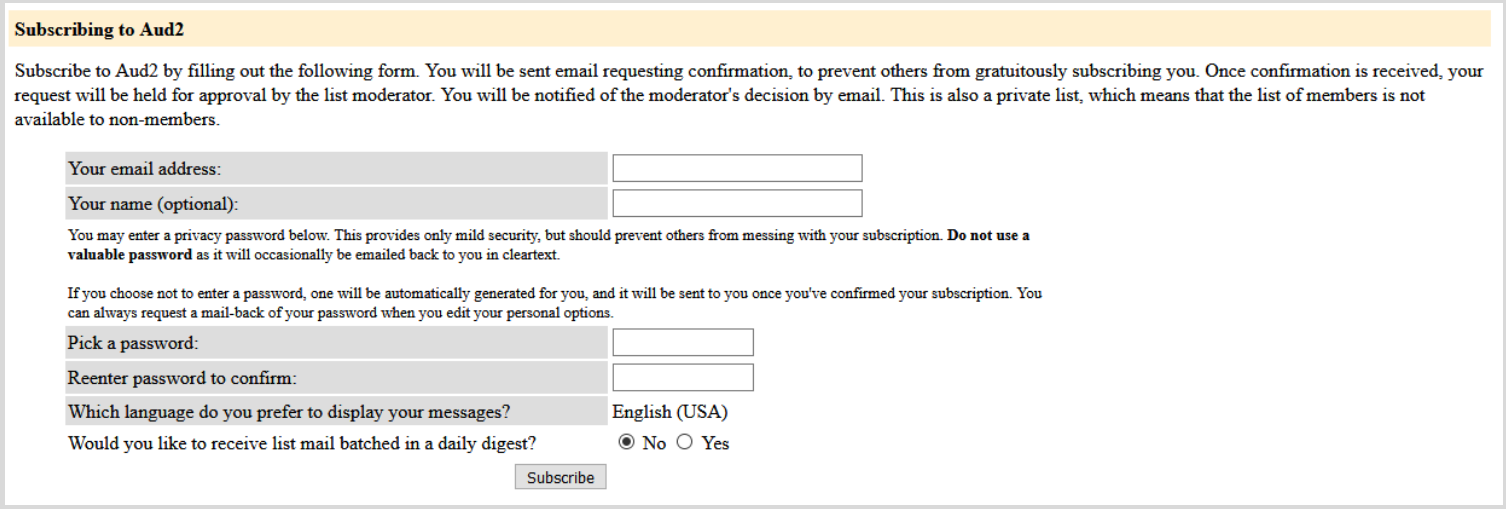
  (Wir müssen erkennen können, dass es eure Lösungen sind)
- Zu spät eingereichte Abgaben werden mit 0 Punkten bewertet

Mailingliste

Mailingliste

Es gibt eine Mailingliste zu dieser Vorlesung. Bitte meldet euch da an, denn wir werden sie nutzen, um kurzfristig Informationen zu verteilen. Bei technischen Schwierigkeiten wendet euch bitte an Arne Schmidt.

- Anmeldung über Homepage
- Information zur Vorlesung/Übung/Klausur/etc.
- Bietet Möglichkeit Fragen zu stellen
  - Wir (Prof. Fekete, Hiwis und ich) sind auf der Liste drauf und können Fragen beantworten
  - Ihr könnt euch untereinander Fragen beantworten!



Subscribing to Aud2

Subscribe to Aud2 by filling out the following form. You will be sent email requesting confirmation, to prevent others from gratuitously subscribing you. Once confirmation is received, your request will be held for approval by the list moderator. You will be notified of the moderator's decision by email. This is also a private list, which means that the list of members is not available to non-members.

Your email address:

Your name (optional):

You may enter a privacy password below. This provides only mild security, but should prevent others from messing with your subscription. **Do not use a valuable password** as it will occasionally be emailed back to you in cleartext.

If you choose not to enter a password, one will be automatically generated for you, and it will be sent to you once you've confirmed your subscription. You can always request a mail-back of your password when you edit your personal options.

Pick a password:

Reenter password to confirm:

Which language do you prefer to display your messages? English (USA)

Would you like to receive list mail batched in a daily digest? ◉ No ○ Yes

Subscribe

Mehr Fragen!

Stellt eure Fragen:

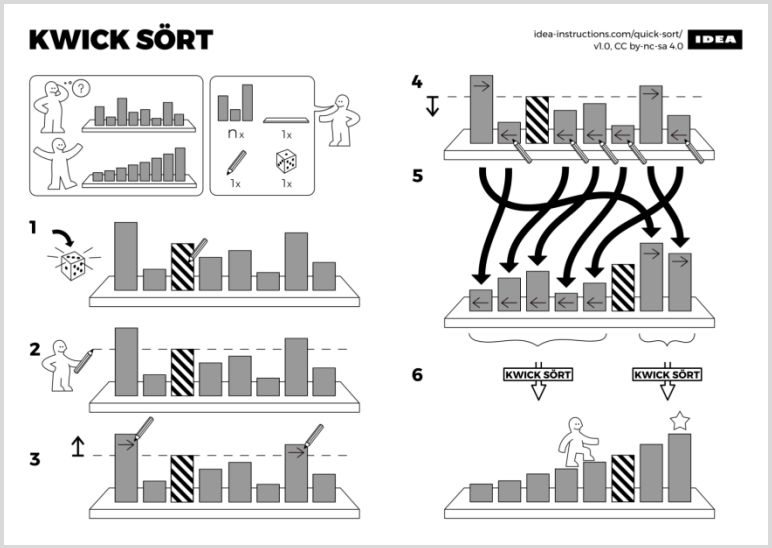
- Über die Mailingliste
- In: Vorlesung/Übung
- Euren Tutoren
- Sprechstunde bei mir (Mo. 09:30 - 10:30 Uhr)
- Sprechstunde bei Prof. Fekete (Mi. 13:15 - 14:00 Uhr)

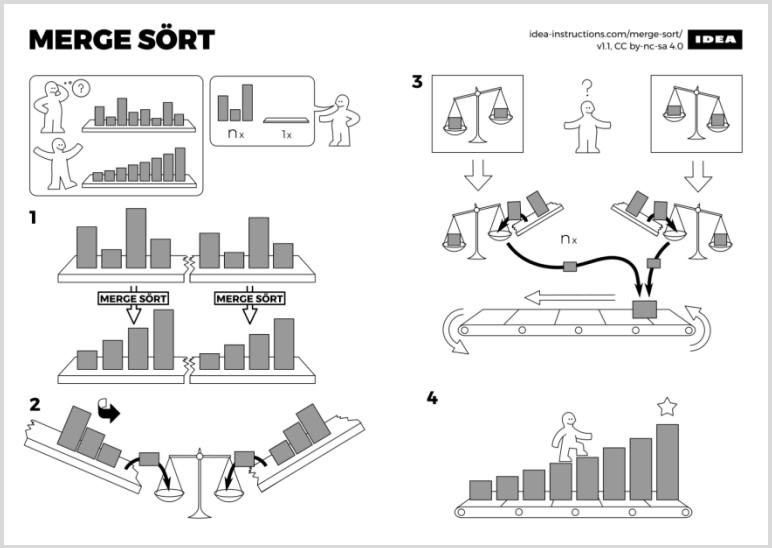
Fragen?

Previously on AuD…

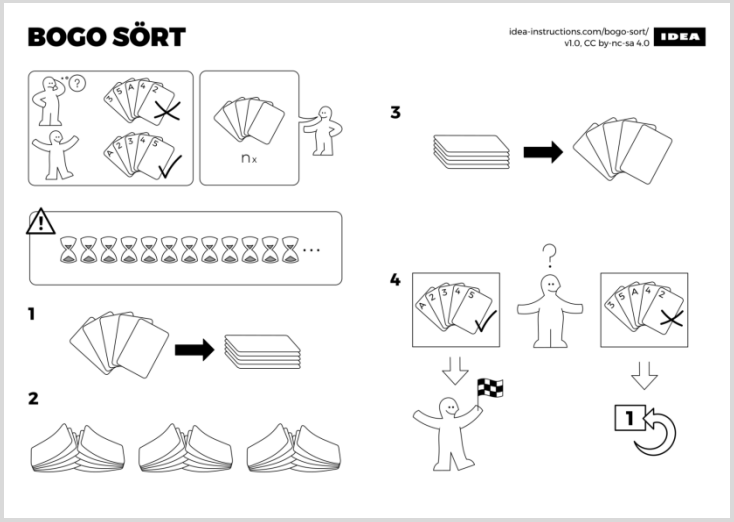
(Sortierverfahren)

Ein paar Sortierverfahren





Nicht ganz ernst gemeint ;-)



Laufzeit – Sortieren

Wie lange dauern diese Algorithmen?

Algorithmus	Best-Case	Average-Case	Worst-Case
Quicksort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Mergesort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Theorem:

Jedes vergleichsbasierte Verfahren benötigt $\Omega(n \log n)$ Schritte.

Laufzeit

Sei $f(n)$ die Laufzeit eines Algorithmus mit Inputgröße n.

O-Notation:

Können wir garantieren, dass $f(n) \leq c_1 \cdot g(n)$ ab irgendeinem n gilt so schreiben wir $f(n) \in O(g(n))$.

Maximale Laufzeit

Ω-Notation:

Können wir garantieren, dass $f(n) \geq c_2 \cdot g(n)$ ab irgendeinem n gilt so schreiben wir $f(n) \in \Omega(g(n))$.

Mindestlaufzeit

Hausaufgaben (Programmierteil)

Das Programm - Items

```java
/**
 * Item with weight and value.
 * Internal use only.
 */
class KnapsackItem implements Comparable<KnapsackItem> {
    private long weight;
    private long value;

    public KnapsackItem(long weight, long value) {
        this.weight = weight;
        this.value = value;
    }

    public long getWeight() {
        return weight;
    }

    public long getValue() {
        return value;
    }

    public int compareTo(KnapsackItem item) {
        final long score = item.getWeight() * getValue();
        final long itemScore = getWeight() * item.getValue();
        final long diff = score - itemScore;

        // prevent overflow
        if (diff < 0) {
            return -1;
        } else if (diff > 0) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Das Programm - Instanz

```java
/**
 * Instance of the Knapsack Problem.
 * Total weight to be packed: getCapacity()
 * #items: getNumItems()
 * weight/value of i-th item: getWeight(i) / getValue(i)
 * sort() sorts items by (value/weight), in descending order
 */
class KnapsackInstance {
    private long capacity;
    private ArrayList<KnapsackItem> items;

    public KnapsackInstance(int size) {
        capacity = 0;
        items = new ArrayList<KnapsackItem>(size);
    }

    public long getCapacity() {
        return capacity;
    }

    public void setCapacity(long capacity) {
        this.capacity = capacity;
    }

    public void append(long weight, long value) {
        items.add(new KnapsackItem(weight, value));
    }

    public long getWeight(int i) {
        return items.get(i).getWeight();
    }

    public long getValue(int i) {
        return items.get(i).getValue();
    }

    public int getNumItems() {
        return items.size();
    }

    public void sort() {
        Collections.sort(items);
        Collections.reverse(items); // descending order
    }

    public String toString() {
        String description = "capacity=" + getCapacity();

        for (int i = 0; i < getNumItems(); ++i) {
            description += " / (weight_" + i + ", value_" + i + ") = (";
            description += getWeight(i);
            description += ", " + getValue(i) + ")";
        }

        return description;
    }
}
```

Das Programm – Lösung I

```
113
114 /**
115  * This class stores solutions of a Knapsack instance.
116  * The i-th boolean indicates whether the i-th item has been chosen.
117  * The resulting weight/value/feasibility of the current solution is tracked.
118  * Copy instances with copyFrom(...).
119  */
120 class KnapsackSolution {
121     KnapsackInstance knapsack;
122     double[] choices;
123     long weight;
124     long value;
125
126     public KnapsackSolution(KnapsackInstance kp) {
127         knapsack = kp;
128         choices = new double[kp.getNumItems()];
129         clear();
130     }
131
132     public double get(int i) {
133         return choices[i];
134     }
135
136     public void set(int i, double b) {
137         if (choices[i] != b) {
138             weight -= choices[i]*knapsack.getWeight(i);
139             value -= choices[i]*knapsack.getValue(i);
140             choices[i] = b;
141             weight += (long)(b*(double)knapsack.getWeight(i));
142             value += (long)(b*(double)knapsack.getValue(i));
143         }
144     }
145
146     public long getWeight() {
147         return weight;
148     }
149
150     public long getValue() {
151         return value;
152     }
153
154
155     public int getSize() {
156         return choices.length;
157     }
158
159     public double getCapacity() {
160         return knapsack.getCapacity();
161     }
162
163     public boolean isFeasible() {
164         return getWeight() <= knapsack.getCapacity();
165     }
166
167     public void clear() {
168         for (int i = 0; i < choices.length; ++i) {
169             choices[i] = 0;
170         }
171
172         weight = 0;
173         value = 0;
174     }
```

Das Programm – Lösung II

```
175
176     public void copyFrom(KnapsackSolution solution) {
177         knapsack = solution.knapsack;
178         choices = new double[solution.choices.length];
179         for (int i = 0; i < solution.choices.length; ++i) {
180             choices[i] = solution.choices[i];
181         }
182         weight = solution.weight;
183         value = solution.value;
184     }
185
186     public String toString() {
187         String description
188                 = (isFeasible() ? "" : "INFEASIBLE! ")
189                 + "Total Weight: " + getWeight()
190                 + " / total value: " + getValue() + " / items:";
191         for (int i = 0; i < choices.length; ++i) {
192             if (choices[i] == 1) {
193                 description += " " + i;
194             }else if (choices[i] > 0){
195                 description += " (" + String.format("%.2f",choices[i]) + ")*" + i;
196             }
197         }
198
199         return description;
200     }
201 }
```

Das Programm – Hauptteil I

```
203 /**
204  * Here are the actual algorithms!
205  */
206 public class knapsack1234567 {
207     // Parses input. Do not edit.
208     public static KnapsackInstance readInstance(String filePath) {
209         try{
210         File file = new File(filePath);
211         Scanner scanner = new Scanner(file);
212         KnapsackInstance kp = new KnapsackInstance(scanner.nextInt());
213 
214         while (scanner.hasNextLong()) {
215             long capacity = scanner.nextLong();
216 
217             if (scanner.hasNextLong()) {
218                 long weight = scanner.nextLong();
219                 long value = scanner.nextLong();
220                 kp.append(weight, value);
221             } else {
222                 kp.setCapacity(capacity);
223             }
224         }
225 
226         kp.sort();
227         return kp;
228         }catch(Exception e){}
229         return null;
230     }
231 
232     public static void writeToCSV(Algorithmus alg, String name, KnapsackSolution ks, long time_init, long time_solve){
233         String file_name = alg + ".csv";
234         System.out.println("value: " + (int)ks.getValue());
235         System.out.println(time_init+"ms");
236         System.out.println(time_solve+"ms");
237 
238         try{
239         Path filePath = Paths.get(file_name);
240         if (!Files.exists(filePath)) {
241             Files.createFile(filePath);
242             Files.write(filePath, "Datei Name;Itemzahl;Kapazität;Wert;Zeit (Einlesen);Zeit (Lösen)\n".getBytes(), StandardOpenOption.APPEND);
243         }
244         String s = name + ";"+ ks.getSize()+ ";"+ ks.getCapacity() +";" + (int)(ks.getValue()) + ";" + time_init + ";" + time_solve + "\n";
245         Files.write(filePath, s.getBytes(), StandardOpenOption.APPEND);
246         }catch(Exception e){}
247     }
```

Das Programm – Hauptteil II

```
248
249     public static KnapsackSolution bruteForce(KnapsackInstance kp) {
250
251     }
252
253     public static KnapsackSolution fractional(KnapsackInstance kp) {
254
255     }
256
257     public static KnapsackSolution greedy(KnapsackInstance kp) {
258
259     }
260
261     public static KnapsackSolution dynamicProgramming(KnapsackInstance kp) {
262
263     }
264
265     public static KnapsackSolution branchAndBound(KnapsackInstance kp) {
266
267     }
268
269     public static KnapsackSolution greedyK(KnapsackInstance kp, int k) {
270
271     }
```

Das ist der Teil den ihr ausfüllen müsst!

Das Programm – Hauptteil III

```
273     public enum Algorithmus{
274         GREEDY, FRACTIONAL, BRANCH, DYNAMIC, BRUTEFORCE, APPROXIMATION
275     }
276
277     public static void main(String[] args) {
278         Algorithmus alg = Algorithmus.DYNAMIC;
279         long time_init, time_solve;
280         Timestamp time_start = new Timestamp(System.currentTimeMillis());
281         KnapsackInstance kp = readInstance(args[0]);
282         Timestamp time_end = new Timestamp(System.currentTimeMillis());
283         time_init = time_end.getTime()-time_start.getTime();
284         System.out.println(kp);
285         KnapsackSolution ks = new KnapsackSolution(kp);
286         time_start = new Timestamp(System.currentTimeMillis());
287         switch (alg){
288             case FRACTIONAL:
289                 ks.copyFrom(fractional(kp));
290                 break;
291             case GREEDY:
292                 ks.copyFrom(greedy(kp));
293                 break;
294             case BRUTEFORCE:
295                 ks.copyFrom(bruteForce(kp));
296                 break;
297             case BRANCH:
298                 ks.copyFrom(branchAndBound(kp));
299                 break;
300             case DYNAMIC:
301                 ks.copyFrom(dynamicProgramming(kp));
302                 break;
303             case APPROXIMATION:
304                 ks.copyFrom(greedyK(kp, 1));
305                 break;
306             default:
307                 System.out.println("Opps...something went wrong. Correct enum chosen?");
308                 break;
309
310         }
311         time_end = new Timestamp(System.currentTimeMillis());
312         time_solve = time_end.getTime()-time_start.getTime();
313         System.out.println(ks);
314         writeToCSV(alg, args[0], ks, time_init, time_solve);
315     }
316 }
```

Hier wählt ihr den Algorithmus aus

Instanzen

```
20
     1     50     31
     2     12     16
     3      8     40
     4     20     31
     5     15     42
     6     27      1
     7     29      9
     8     49      9
     9     24     27
    10     26     18
    11     41     39
    12      5     47
    13     39     28
    14      2     50
    15     45     34
    16     22     32
    17      6     23
    18     12     14
    19     34     13
    20     47     50
274
```

Ganz einfach auch zum selber machen!

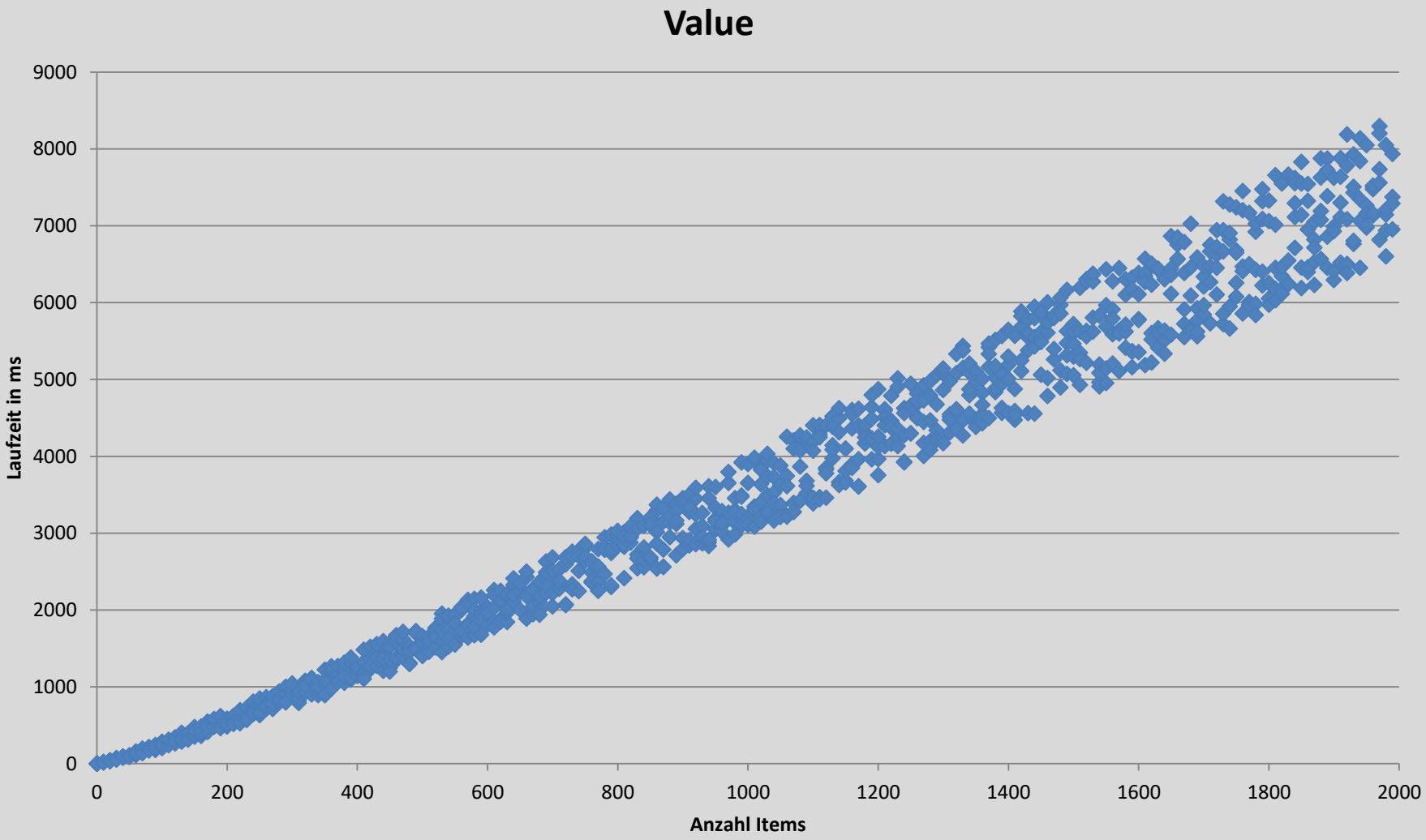
- Erste Zeile: Anzahl der Objekte
- Danach für jedes Objekt:
  - Nummer
  - Gewicht
  - Wert
- Letzte Zeile: Kapazität

Was tun mit gelösten Instanzen?

Visualisierung hilft die Ergebnisse zu verstehen.

Dafür eignen sich zum Beispiel X-Y-Scatter-Plots.
Man benötigt dazu einfach Paare von Daten wie
(Inputgröße, Laufzeit) oder (Inputgröße, Lösungswert), etc.

Was tun mit gelösten Instanzen?



Was tun mit gelösten Instanzen?

Visualisierung hilft die Ergebnisse zu verstehen.

Dafür eignen sich zum Beispiel X-Y-Scatter-Plots.
Man benötigt dazu einfach Paare von Daten wie
(Inputgröße, Laufzeit) oder (Inputgröße, Lösungswert), etc.

Wenn man weiter gehen möchte, kann man auch
Durchschnitt, Mediane, Quartile, usw. Darstellen.

Für unsere Zwecke brauchen wir das hier nicht.

Erstellt werden können solche Diagramme mit:

- Excel
- LibreOffice Calc
- GnuPlot
- Matlab
- Etc.

https://imgflip.com/memetemplate/63997228/Thats-All-Folks