Theoretische Aufgaben

Aufgabe 1

- Erfolgreiches Suchen: Bleibt $O(1)$.
- Erfolgloses Suchen: Sobald ein Element grösser ist als das gesuchte kann abgebrochen werden: $O(n) \to O(n)/2$. Dass heisst, die asymptotische Laufzeit bleibt $O(n)$, aber in der Praxis halbiert sich die Laufzeit.
- Einfügen: Element muss an richtiger Stelle eingesetzt werden, dass verlangsamt den Prozess: $O(n)$.
- Löschen: Bleibt $O(1)$.

Aufgabe 2

k	**h(k)**
51	519
52	137
53	755
54	373
55	991

Aufgabe 3

- **Lineare Sondierung**
  k=8: Überprüfe Feld [8] → füge 8 in Feld [8] ein.
  k=25: Überprüfe Feld [3] → füge 25 in Feld [3] ein.
  k=12: Überprüfe Feld [1] → füge 12 in Feld [1] ein.
  k=11: Überprüfe Feld [0] → füge 11 in Feld [0] ein.
  k=34: Überprüfe Feld [1], enthält bereits Schlüssel 12, überprüfe [2] → füge 34 in Feld [2] ein.
  k=22: Überprüfe Feld [0], besetzt, [1], [2], [3] auch, [4] ist verfügbar → füge 22 in Feld [4] ein.
  k=16: Überprüfe Feld [5] → füge 16 in Feld [5] ein.
  k=54: Überprüfe Feld [10] → füge 54 in Feld [10] ein.
  k=38: Überprüfe Feld [5], ist besetzt, [6] → füge 38 in Feld [6] ein.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
11	12	34	25	22	16	38	NULL	8	NULL	54

- **Quadratische Sondierung**
  k=8: Überprüfe Feld [8] → füge 8 in Feld [8] ein.
  k=25: Überprüfe Feld [3] → füge 25 in Feld [3] ein.
  k=12: Überprüfe Feld [1] → füge 12 in Feld [1] ein.
  k=11: Überprüfe Feld [0] → füge 11 in Feld [0] ein.
  k=34: Überprüfe Feld [1], enthält bereits Schlüssel 12, überprüfe [5][1] → füge 34 in Feld [5] ein.
  k=22: Überprüfe Feld [0], besetzt, [4] ist verfügbar → füge 22 in Feld [4] ein.
  k=16: Überprüfe Feld [5], besetzt, [9] ist verfügbar → füge 16 in Feld [9] ein.
  k=54: Überprüfe Feld [10] → füge 54 in Feld [10] ein.
  k=38: Überprüfe Feld [5], ist besetzt, [9] und [8] auch, [2] ist frei → füge 38 in Feld [2] ein.

[1] Berechnung des Feldes: $1 + c_1 \cdot 1^1 + c_2 \cdot 1^1 = 1 + 1 + 3 = 5$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
11	12	38	25	22	34	NULL	NULL	8	16	54

- **Doppeltes Hashing**
  k=8: Überprüfe Feld [8] → füge 8 in Feld [8] ein.
  k=25: Überprüfe Feld [3] → füge 25 in Feld [3] ein.
  k=12: Überprüfe Feld [1] → füge 12 in Feld [1] ein.
  k=11: Überprüfe Feld [0] → füge 11 in Feld [0] ein.
  k=34: Überprüfe Feld [1], enthält bereits Schlüssel 12, überprüfe [6][2] → füge 34 in Feld [6] ein.
  k=22: Überprüfe Feld [0], besetzt, [3] und [6] auch, [9] ist verfügbar → füge 22 in Feld [9] ein.
  k=16: Überprüfe Feld [5] → füge 16 in Feld [5] ein.
  k=54: Überprüfe Feld [10] → füge 54 in Feld [10] ein.
  k=38: Überprüfe Feld [5], ist besetzt, [3], [1], [10], [8] und [6] auch, [4] ist frei → füge 38 in Feld [4] ein.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
11	12	NULL	25	38	16	34	NULL	8	22	54

Aufgabe 4

Die Länge der Suche hängt davon ab, in welcher Reihenfolge die Schlüssel eingefügt wurden. Diese wiederum wird durch das Einfügen bestimmt. Beim linearen Sortieren wird aber der nächstoffene Platz belegt, und die Reihenfolge wird erhalten (analog zur Stabilität bei Sortieralgorithmen). Dies muss beim quadratischen Sortieren (in Abhängigkeit der gewählten Schlüssel c_1 und c_2) nicht unbedingt gewährleistet werden, da über eine bestimmte Anzahl von Elementen $c_1 \cdot i + c_2 \cdot i^2$ 'gesprungen' wird.

Aufgabe 5

???

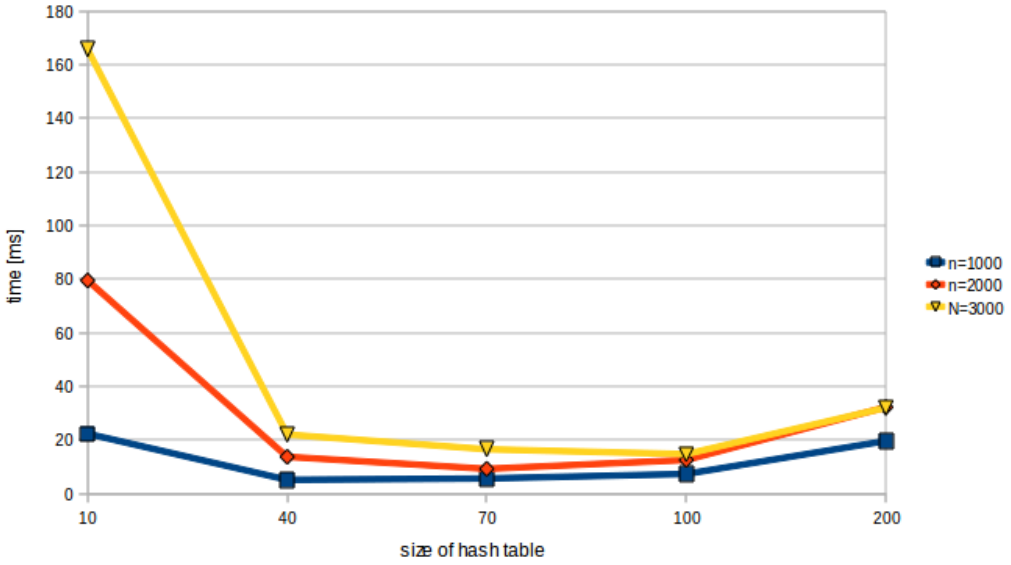
Praktische Aufgaben

Aufgabe 1

Siehe Datei `BouncingBallsSimulationImproved.java`.

[2] Berechnung: $1 + (1 + k \bmod 10) = 1 + 1 + 4 = 6$

Aufgabe 2



Bei grösseren Tabellen müssen wiederum sehr viele Punkte miteinander verglichen werden, was zur Verlangsamung führt, wie dies am Anfang der Fall war. Bei kleineren Tabellen kann effizienter verglichen werden, bis zu einer gewissen Grösse, wenn dann noch kleinere Felder gewählt werden, ist der Algorithmus wieder ineffizient. Es scheint, dass von den gewählten Intervallen bei n=1000 Partikel eine Tabelle mit Grösse m=40, bei n=2000 Grösse m=70 und bei n=3000 eine Grösse von m=100 optimal ist.