Grundlagen

Aufgabe 1

$$\sum_{k=1}^{n}(2k-1) \Rightarrow 2 \cdot \sum_{k=1}^{n} k - \sum_{k=1}^{n} 1 \tag{1}$$

$$\text{Mit } \sum_{i=1}^{n}(i) = \frac{n \cdot (n+1)}{2} \tag{2}$$

$$\Rightarrow 2 \cdot \frac{n \cdot (n+1)}{2} - n \Rightarrow n^2 \tag{3}$$

Aufgabe 2

$$\prod_{k=1}^{n} 2 \cdot 4^k \Rightarrow \prod_{k=1}^{n} 2 \cdot \prod_{k=1}^{n} 4^k \tag{4}$$

$$\prod_{k=1}^{n} 2 = 2^n \tag{5}$$

$$\prod_{k=1}^{n} 4^k = 4^{\frac{n^2}{2}+\frac{n}{2}} \tag{6}$$

$$\Rightarrow 2^n \cdot 4^{\frac{n^2}{2}+\frac{n}{2}} \tag{7}$$

$$\Rightarrow 2^n \cdot 4^{\frac{n^2}{2}} \cdot 4^{\frac{n}{2}} \tag{8}$$

$$\Rightarrow 2^n \cdot 2^{n^2} \cdot 2^n \tag{9}$$

$$\Rightarrow 2^{n^2} \cdot 2^{2n} \Rightarrow 2^{n^2} \cdot 4^n \tag{10}$$

Aufgabe 3

$$\sum_{k=1}^{n} \frac{1}{k^2} \leq 1 + 1 - \frac{1}{n} \tag{11}$$

$$\text{Verankerung: } n = 1. \sum_{k=1}^{1} \frac{1}{k^2} = 1 \tag{12}$$

$$\text{Schritt: } \sum_{k=1}^{n+1} \frac{1}{k^2} \leq 1 + 1 - \frac{1}{n+1} \tag{13}$$

$$\sum_{k=1}^{n} \frac{1}{k^2} + \frac{1}{(n+1)^2} = (2 - \frac{1}{n}) + \frac{1}{(n+1)^2} \tag{14}$$

$$\Rightarrow 2 - \frac{1}{n} \tag{15}$$

$$\Rightarrow -\frac{1}{n} + \frac{1}{(n+1)^2} \leq -\frac{1}{n+1}, \text{ welches für alle n kleiner 1 stimmt} \tag{16}$$

Theoretische Aufgaben

Aufgabe 1

Der InsertionSort-Algorithmus ist relativ einfach zu erklären:
Eine Variable `key` wird iteriert, und anschliessend mit den vorderen Elementen verglichen. Ist das Element kleiner, wird es verschoben, wenn nicht, iteriert `key`.

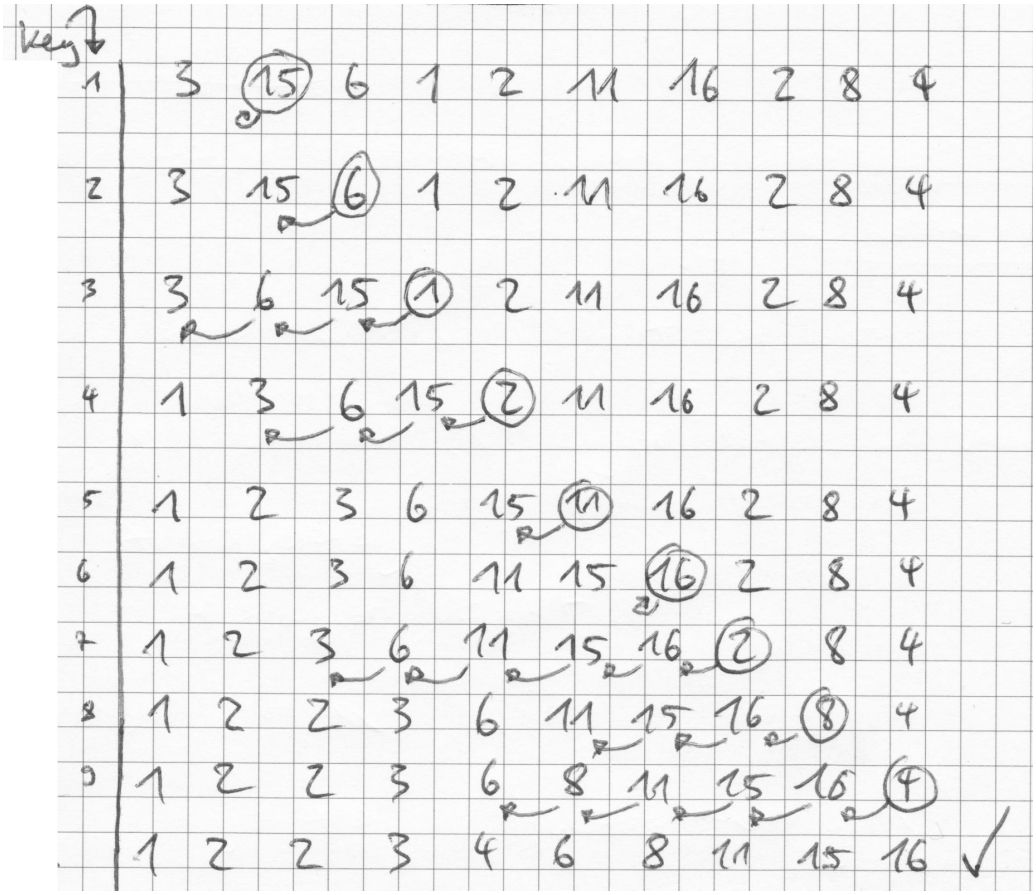

Abbildung 1: Funktionsweise des InsertionSort-Algorithmus.

Aufgabe 2

Der MergeSort-Algorithmus hingegen ist etwas schwerer zu erklären, und Abb. 1 soll versuchen aufzuzeigen, dass zuerst in Rekursionsschritten versucht wird, solange aufzuteilen, bis die kleinste Einheit von 2 Elementen erreicht wird.
Danach werden diese sortiert, und es werden die zwei (vorsortierten) neuen Elemente miteinander zusammengefügt[1], sodass auch diese Einheit wieder sortiert ist. Dies wird fortgesetzt bis die ganze Eingabe vollständig sortiert ist.

[1] Dies in einer Art, dass bei beiden vom Element 0 an aufwärts geschaut wird, welches der beiden zusammenzufügenden Teile das kleiner Element hat. Dieses wird herausgepickt, und dann wird wieder das nächstgrössere Element gesucht, solange, bis alle Elemente zum sortierten Teil hinzugefügt wurden.

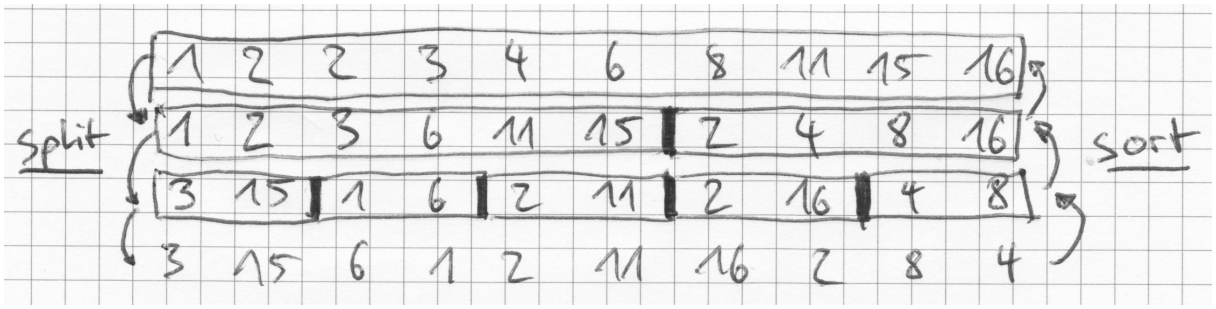

Abbildung 2: Funktionsweise des MergeSort-Algorithmus.

Aufgabe 3

Listing 1: Suche im unsortierten Feld

```
1  search(int[] A, int v)
2
3  for i = 0 to A.length()
4      if A[i] = v return v
5  return NULL
```

Bedingung: Das ganze Array wurde durchlaufen und der Index von v, wenn gefunden, ausgegeben.
Schleifeninvariante: Das Array wurde von 0 bis i-1 durchsucht.
Terminierungsbedingung: i = Länge des Arrays

Gültigkeit der Schleifeninvariante:

- **Vor der Schleife:** i muss korrekt initialisiert werden. Dies wird mit i = 0 erreicht, indem i den index des ersten Elements von A annimmt.
- **Während der Schleife:** i wird laufend nur um eins erhöht, und da dies gilt, wurde nach Durchlauf A immer bis A[i] durchsucht, und mit der if-Prüfung würde, falls v gefunden würde, die Schleife terminiert werden.
- **Terminierung:** Die Schleife kann während der Schleife, also währenddem $0 \leq i < A.length()$ gilt, abgebrochen werden, tut dies aber auf jeden Fall, wenn i die Länge des Arrays erreicht.

Worst-Case-Laufzeit für diesen Algorithmus ist $T(n) = O(n)$, also dass alle Elemente einmal durchlaufen werden müssen (wenn v nicht gefunden wurde).

Aufgabe 4

Listing 2: Suche in sortiertem Feld

```
6  search(int[] A, int v)
7
8  //dist: Grösse des zu durchsuchenden verbleibenden Feldes
9  dist = (A.length)/2 + 1
10 //k: Index des mittleren Feldes des aktuellen Suchfeldes
11 k = dist
12
13 while dist >= 1
```

```
14    //test if number is found
15    if A[k] = v then return k
16
17    //change k
18    dist = dist/2
19    if A[k] < v then k = k + dist
20    else if A[k] > v then k = k - dist
21
22  return NULL
```

Bedingung: Das ganze Array wurde durchlaufen und der Index von v, wenn gefunden, ausgegeben.
Schleifeninvariante: Ist der gesuchte Eintrag gerade an Stelle k, soll der Index zurückgegeben werden. Falls nicht, soll je nachdem die grössere oder kleinere verbleibende Hälfte untersucht werden.
Terminierungsbedingung: Wenn das gesuchte Element gefunden wurde oder der zu untersuchende Bereich kleiner als 1 Element ist, soll abgebrochen werden.

Gültigkeit der Schleifeninvariante:

- **Vor der Schleife:** k muss korrekt initialisiert werden. Dies wird mit k = dist = 'die Hälfte der Array-Länge' erreicht, indem k und dist den index des mittleren Elements von A annimmt.
- **Während der Schleife:** Entspricht der mittlere Eintrag dem gesuchten Element, wird dies zurückgegeben. Falls dies nicht der Fall ist, wird der Suchbereich halbiert und k erhält den Index der Mitte des Bereichs, dabei wird bei geraden Bereichen (welche 2 mittlere Elemente haben) immer die kleinere mögliche Mitte gewählt[2].
- **Terminierung:** Wenn der Wert gefunden wurde, der $A[k] = v$ entspricht, wird der Index zurückgegeben. Falls nicht, wird solange weitergesucht, bis das Suchfeld die Grösse 1 hat, was also heisst, dass das verbleibende Element entweder die Lösung ist oder nicht. Wenn ja, wird diese zurückgegeben, wenn nicht, hält die Schleife durch die Anfangsbedingung (dist ¿= 1) an. Es wird hier bewusst der Operator ¿= gewählt, denn wenn das Suchfeld auf 1 verkleinert wurde, muss dieses zuerst noch geprüft werden, ob sich das gesuchte Element darin befindet. Wenn dies nicht der Fall ist, wird das Suchfeld auf einen Wert unter 1 verkleinert und die Schleife bricht ab.

Worst-Case-Laufzeit für diesen Algorithmus ist $T(n) = O(log(n))$, weil für die Verdoppelung der Elemente (z.B. von 32 auf 64) nur 1 Schritt mehr gemacht werden muss (von 5 auf 6 Schritte). Dies natürlich, weil die Anzahl der zu durchsuchenden Elemente bei jedem Durchgang halbiert werden kann (die obere oder die untere Hälfte fällt ja jedesmal weg).

Praktische Aufgaben

Testprogramm

Siehe Anhang.

Auswertung der Testfälle

Gemäss der Abb. 3[3] lässt sich zeigen, dass für die gegebenen Datenpunkte relativ genau mit den Erwartungen aus der Theorie übereinstimmen.

[2]Der Eintrag wird durch den Typ (Integer) bedingten Wegfall von Nachkommastellen immer auf den kleineren möglichen Wert gesetzt. Bei vier Elementen zum Beispiel wird 2.5 auf 2 abgerundet, was das zweite Element als Resultat ergibt.
[3]Es wurde hier zur besseren Darstellung die logarithmische Skalierung der X-Achse gewählt.

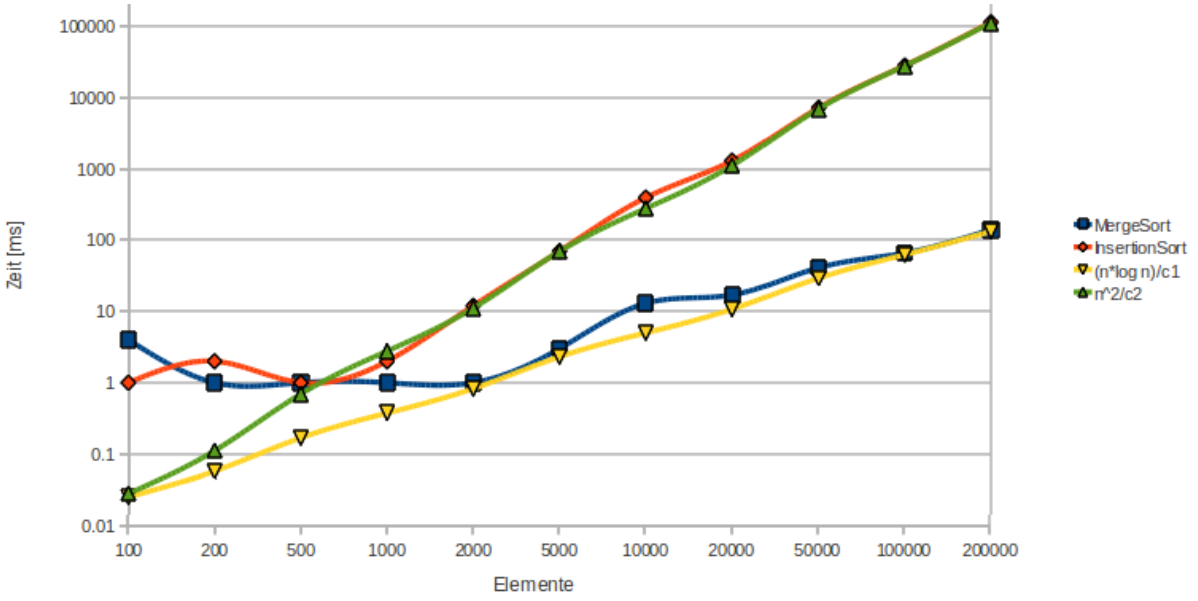

Abbildung 3: Darstellung der ermittelten Funktionswerte des Testprogramms.

Die Konstanten c1 und c2 wurden so gewählt, dass sich die Angleichung optimal anzeigen lässt[4]. Die Fluktuationen bei geringen Werten war zu erwarten, und bestätigt die Vermutung, dass bei diesen Werten von n andere Faktoren eine grössere Rolle spielen, welche aber dann bei grösseren n in den Hintergrund rücken.

Spezifikationen Computer:
Name: Acer Aspire One
Intel Atom N450 (1.66 GHz, 512 KB Cache)
1 GB RAM

Schätzung InsertionSort

Ermittle konstanter Faktor c aus Testprogramm: $c = \frac{T_i}{n_i^2}$, daraus folgt $c = 2.8 \cdot 10^{-9}$,
Daraus folgt bei $n = 10'000'000 \Rightarrow$ etwa 280'000 Sekunden, also etwa 3.25 Tage.

[4]Für diesen Fall mit den ermittelten Spezifikationen wurden folgende Werte gewählt: $c1 = 8000, c2 = 600$