Um die Korrektheit von Einfügesortieren zu beweisen besprechen wir zunächst noch kurz einige Begriffe und Notationen die für Korrektheitsbeweise von Algorithmen im Allgemeinen von Bedeutung sind: Eine *Vorbedingung* eines Algorithmus ist eine Bedingung von der wir annehmen, dass sie beim Start des Algorithmus erfüllt ist. Eine *Nachbedingung* ist eine Aussage von der wir zeigen wollen, dass sie nach Ausführung des Algorithmus erfüllt ist falls die Eingabe die Vorbedingung erfüllt hat. Eine Korrektheitsaussage hat also oft die Form "Falls die Vorbedingung ... erfüllt ist, dann ist nach Ausführung des Algorithmus ... die Nachbedingung ..." erfüllt.

Beweise mittels Schleifeninvarianten werden oft systematisch in die oben erwähnten Punkte 1 bis 3 unterteilt. Damit ist an fast jeder Stelle klar auf den Wert zu welchen Zeitpunkt man sich bezieht wenn man eine Programmvariable erwähnt: bei 1 auf den Beginn der Schleife und bei 3 auf das Ende der Schleife. Die einzige Ausnahme ist 2 wo man sich auf den Wert zu zwei unterschiedlichen Zeitpunkte bezieht: zu Beginn der i-ten Iteration und zu Beginn der $i+1$-ten Iteration. Eine gebräuchliche Notation für diese Situation besteht darin den Wert einer Programmvariablen $\mathtt{X}$ zu Beginn der i-ten Iteration einfach als $\mathtt{X}$ zu notieren und jenen zu Beginn der $i+1$-ten Iteration als $\mathtt{X}'$. Auch Varianten davon wie zum Beispiel $\mathtt{X}_{\text{alt}}$ und $\mathtt{X}_{\text{neu}}$ sind in Gebrauch.

Wir wollen nun in dieser kompakteren Notation die Korrektheit von Einfügesortieren beweisen. Dazu beginnen wir mit der folgenden Aussage über die innere Schleife.

Lemma 1.1. *Sei* EINFÜGEN$(\mathtt{A},\mathtt{i},\mathtt{x})$ *eine Abkürzung für die Zeilen 5-9 von Algorithmus 2. Dann gilt für* EINFÜGEN$(\mathtt{A},\mathtt{i},\mathtt{x})$ *unter der Vorbedingung (V)*

$i_0 = \mathtt{i}$, $n = \mathtt{A}$.*Länge,* $\mathtt{A}[1,\ldots,i_0] = m_1,\ldots,m_{i_0}$ *ist sortiert*
und $\mathtt{A}[i_0+1,\ldots,n] = m_{i_0+1},\ldots,m_n$

die Nachbedingung (N)

$\mathtt{A}[1,\ldots,i_0+1] = m_1,\ldots,m_i,\mathtt{x},m_{i+1},\ldots,m_{i_0}$ *ist sortiert*
und $\mathtt{A}[i_0+2,\ldots,n] = m_{i_0+2},\ldots,m_n$.

Beweis. Der Körper der Schleife in EINFÜGEN$(\mathtt{A},\mathtt{i},\mathtt{x})$ induziert die Abbildung $(\mathtt{A},\mathtt{i}) \mapsto (\mathtt{A}',\mathtt{i}')$ wobei

$$\mathtt{A}'[k] = \begin{cases} \mathtt{A}[\mathtt{i}] & \text{falls } k = \mathtt{i}+1 \\ \mathtt{A}[k] & \text{sonst} \end{cases} \qquad \text{und} \qquad \mathtt{i}' = \mathtt{i} - 1.$$

Für diese Schleife verwenden wir die Invariante $I(\mathtt{A},\mathtt{i})$:

$\mathtt{i} \le i_0$,
$\mathtt{A}[1,\ldots,i_0+1] = m_1,\ldots,m_{\mathtt{i}},m_{\mathtt{i}+1},m_{\mathtt{i}+1},\ldots,m_{i_0}$,
$\mathtt{A}[i_0+2,\ldots,n] = m_{i_0+2},\ldots,m_n$
und falls $\mathtt{i} < i_0$ dann $x < m_{i+1}$.

und argumentieren wie folgt:

1. Zu Beginn der Schleife folgt $I(\mathtt{A},\mathtt{i})$ unmittelbar aus (V).
2. $I(\mathtt{A},\mathtt{i})$ impliziert $I(\mathtt{A}',\mathtt{i}')$ da erstens $\mathtt{i}' = \mathtt{i} - 1 \le^{I(\mathtt{A},\mathtt{i})} i_0 - 1 < i_0$ und damit bleibt $\mathtt{A}[i_0+2,\ldots,n]$ unverändert. Weiters ist

$$\begin{aligned} \mathtt{A}'[1,\ldots,i_0+1] &= \mathtt{A}[1,\ldots,\mathtt{i}],\mathtt{A}[\mathtt{i}],\mathtt{A}[\mathtt{i}+2,\ldots,i_0+1] \\ &=^{I(\mathtt{A},\mathtt{i})} m_1,\ldots,m_{\mathtt{i}},m_{\mathtt{i}},m_{\mathtt{i}+1},\ldots,m_{i_0} \\ &= m_1,\ldots,m_{\mathtt{i}'+1},m_{\mathtt{i}'+1},\ldots,m_{i_0}. \end{aligned}$$

Schließlich ist $\mathtt{i}' = \mathtt{i} - 1 < i_0$, also ist zu zeigen dass $x < m_{\mathtt{i}'+1} = m_{\mathtt{i}} =^{I(\mathtt{A},\mathtt{i})} \mathtt{A}[\mathtt{i}]$ was unmittelbar aus der Wahrheit der Schleifenbedingung folgt.

3. Bei Beendigung der Schleife ist $\mathtt{A}[1,\ldots,i_0+1] = m_1,\ldots,m_{\mathtt{i}},m_{\mathtt{i}+1},m_{\mathtt{i}+1},\ldots,m_{i_0}$ mit $x < m_{\mathtt{i}+1}$ und ($\mathtt{i} = 0$ oder ($\mathtt{i} \geq 1$ und $\mathtt{A}[\mathtt{i}] = m_i \leq x$)). Nach Ausführung von Zeile 9 ist damit $\mathtt{A}[1,\ldots,i_0+1] = m_1,\ldots,m_{\mathtt{i}},\mathtt{x},m_{\mathtt{i}}+1,\ldots,m_{i_0}$ sortiert und damit ist (N) gezeigt.

□

Satz 1.2. *Einfügesortieren ist korrekt.*

Beweis. Genauer gesagt wollen wir zeigen dass für EINFÜGESORTIEREN(A) unter der Vorbedingung (V*)

$n = \mathtt{A}.\mathit{Länge}$ und $A[1,\ldots,n] = p_1,\ldots,p_n$

die Nachbedingung (N*)

$\mathtt{A}[1,\ldots,n]$ ist sortierte Permutation von $p_1,\ldots,p_n$

gilt. Dazu verwenden wir für die äußere Schleife die Invariante $I(\mathtt{A},\mathtt{j})$:

$\mathtt{A}[1,\ldots,\mathtt{j}-1]$ ist sortierte Permutation von $p_1,\ldots,p_{\mathtt{j}-1}$ und
$\mathtt{A}[j,\ldots,n] = p_j,\ldots,p_n$.

und argumentieren wie folgt:

1. Zu Beginn der Schleife ist $\mathtt{j} = 2$ und damit folgt $I(\mathtt{A},\mathtt{j})$ unmittelbar aus (V*).

2. Aus $I(\mathtt{A},\mathtt{j})$ folgt (V) von Lemma 1.1 da $i_0 = \mathtt{j} - 1$ und $\mathtt{A}[1,\ldots,\mathtt{j}-1] = p_1,\ldots,p_{\mathtt{j}-1}$ sortiert ist. Also ist wegen Lemma 1.1 nach Ausführung von Zeile 9, und damit zu Beginn des nächsten Durchlaufs, die Nachbedingung (N) erfüllt. Also ist $\mathtt{A}'[1,\ldots,\mathtt{j}] = p_1,\ldots,p_{\mathtt{i}},\mathtt{x},p_{\mathtt{i}+1},\ldots,p_{\mathtt{j}-1}$ und sortiert und $\mathtt{A}'[\mathtt{j}+1,\ldots,n] = A[\mathtt{j}+1,\ldots,n] = p_{\mathtt{j}+1},\ldots,p_n$ woraus $I(\mathtt{A}',\mathtt{j}')$ folgt da $\mathtt{j} = \mathtt{j}' - 1$.

3. Bei Beendigung der Schleife ist $\mathtt{j} = n+1$ und damit folgt aus $I(\mathtt{A},\mathtt{j})$ dass $\mathtt{A}[1,\ldots,n]$ eine sortierte Permutation von $p_1,\ldots,p_n$ ist.

□

1.3 Aufwandsabschätzung

Der wichtigste Aspekt eines Algorithmus, neben seiner Korrektheit, ist typischerweise sein Ressourcenverbrauch, und hier vor allem: seine Laufzeit. Auch der Verbrauch anderer Ressourcen, zum Beispiel Speicherplatz, kann von Bedeutung sein, zentral ist aber in den allermeisten Situationen die Frage wie viel Zeit ein Algorithmus benötigt. Eine erste Antwort auf diese Frage bestünde darin, eine konkrete Implementierung des Algorithmus in einer bestimmten Programmiersprache anzufertigen, diese auf einem bestimmten Computer auf verschiedene Eingaben anzuwenden und die verbrauchte Zeit zu protokollieren. Derartige empirische Studien haben in der Informatik durchaus ihren Nutzen, allerdings haben sie die folgenden Nachteile:

1. Es ist unmöglich alle, typischerweise unendlich vielen, Eingaben auszuprobieren.

2. Die Ergebnisse hängen von der Implementierung, der Programmiersprache und vom Computer ab.

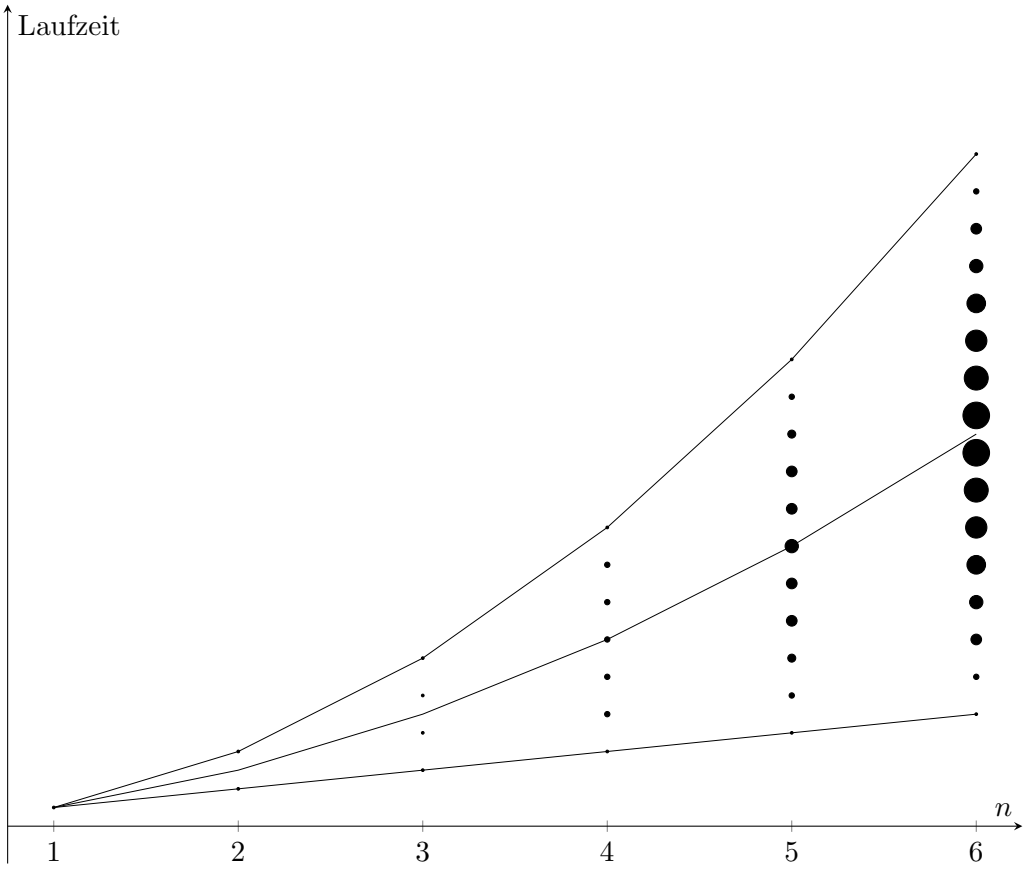

Abbildung 1.1: Laufzeit von Einfügesortieren für Datenfelder der Länge $n \leq 6$

Um zu erklären wie man diese Nachteile durch geeignete Abstraktionen vermeiden kann betrachten wir zunächst ein Beispiel. In Abbildung 1.1 ist die Laufzeit von Einfügesortieren für Eingabedatenfelder der Länge $n = 1, \ldots, 6$ dargestellt. Da es bei Sortieralgorithmen nur auf die Reihenfolge der Elemente in der Ordnung ankommt, wurden hier für festes n als Eingabe nur die $n!$ Permutationen von $\{1, \ldots, n\}$ betrachtet. Damit schränken wir unsere Betrachtung auch auf Fälle ein wo keine Elemente mehrfach vorkommen. Die Größe eines Punktes ist proportional zur Anzahl der Fälle mit dieser Laufzeit. Wir sehen dass es für festes n (abhängig von der Permutation) verschiedene Laufzeiten gibt und dass die Laufzeiten mit steigendem n wachsen. Zusätzlich sind in Abbildung 1.1 drei Kurven eingezeichnet: die Laufzeit im besten Fall (engl. *best case*), die Laufzeit im schlechtesten Fall (engl. *worst case*), sowie die Laufzeit im Durchschnittsfall (engl. *average case*).

Ein Verhalten wie das in Abbildung 1.1 dargestellte ist typisch für die Praxis: üblicherweise wächst die Laufzeit mit der Größe der Eingabe. Um zu einer abstrakteren Darstellung der Laufzeit zu kommen betrachten wir sie also nicht als eine Funktion der Menge erlaubter Eingaben, sondern als eine Funktion der Eingabegröße. Um der Tatsache Rechnung zu tragen, dass es für eine fixe Eingabegröße unterschiedliche Laufzeiten gibt betrachtet man die drei Laufzeiten im besten, im schlechtesten, und im durchschnittlichen Fall.

Zur Illustration führen wir nun eine Analyse der Komplexität von Einfügesortieren durch. Um die Analyse von Algorithmen zu vereinfachen und von Details der Implementierung und der Hardware zu abstrahieren (sh. oben) trifft man üblicherweise die folgenden Annahmen:

1. Die Instruktionen werden sequentiell ausgeführt (was auf Mehrprozessor-Systemen nicht immer der Fall sein muss).

2. Der Zeitbedarf jeder "elementaren Operation" ist konstant[1], also insbesondere ist er unabhängig von den Werten der Programmvariablen und wird nicht beeinflusst von Speicherhierachiekonzepten wie z.B. caching.

3. Wir arbeiten mit unbeschränkten Datentypen, also z.B. den natürlichen Zahlen und nicht, wie das in konkreten Implementierungen typischerweise der Fall ist mit, z.B., $\{0, \ldots, 2^{64} - 1\}$.

4. Wir nehmen an, dass wir mit den behandelten Zahlen exakt rechnen können. Der Einfluß von Rundungsfehlern bei der Verwendung von Gleitkommazahlen zur Darstellung reeller Zahlen spielt in der numerischen Mathematik eine wichtige Rolle, in dieser Vorlesung stellt er nur einen Nebenaspekt dar.

Wir können also voraussetzen dass für $i = 2, \ldots, 10$ eine Konstante c_i existiert, welche die Zeit angibt, die zur Ausführung von Zeile i benötigt wird. Sei A das Eingabedatenfeld und n die Länge von A. Dann belaufen sich die Kosten der Anwendung von Einfügesortieren auf das Datenfeld A auf

$$T(A) = (c_2 + c_{10})n + (c_3 + c_4 + c_9)(n-1) + (c_5 + c_8)\sum_{j=2}^{n} t_j + (c_6 + c_7)\sum_{j=2}^{n}(t_j - 1)$$

wobei t_j angibt, wie oft die Bedingung der inneren Schleife überprüft wird, wenn der äußere Schleifenzähler j ist. Die t_j hängen offensichtlich von A ab. Was sind nun mögliche Werte für die t_j ?

[1] Wir geben hier keine präzise Definition davon, was als elementare Operation zu verstehen ist. Jedenfalls dazu gehören: arithmetische Operationen wie Addition und Multiplikation, schreibender und lesender Zugriff auf Variablen sowie die Ausführung von Kontrollstrukturen wie Bedingungen, Schleifenköpfen, etc.

Im besten Fall ist das Eingabedatenfeld A bereits sortiert. Dann gilt nämlich $t_j = 1$ für $j = 2, \ldots, n$ und wir erhalten

$$T_{\mathrm{b}}(n) = (c_2 + c_{10})n + (c_3 + c_4 + c_5 + c_8 + c_9)(n-1),$$

d.h. also $T_{\mathrm{b}}(n) = kn + l$ für geeignete Konstanten k und l. Mit anderen Worten: die Laufzeit hängt linear von der Größe der Eingabe ab.

Im schlechtesten Fall muss jedes $A[j]$ mit *allen* Elementen in $A[1], \ldots, A[j-1]$ verglichen werden. Das geschieht dann wenn das Eingabedatenfeld absteigend sortiert ist. Dann ist also $t_j = j$ für $j = 2, \ldots, n$ und wir erhalten $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$ sowie $\sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$ aus der gaußschen Summenformel und damit

$$T_{\mathrm{s}}(n) = (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} + \frac{c_8}{2})n^2 + (c_2 + c_3 + c_4 + c_9 + c_{10} + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + \frac{c_8}{2})n - c_3 - c_4 - c_5 - c_8 - c_9,$$

d.h. also $T_{\mathrm{s}}(n) = kn^2 + ln + m$ für geeignete Konstanten k, l und m. Im schlechtesten Fall hängt die Laufzeit also quadratisch von der Größe der Eingabe ab.

Bei der Analyse des Durchschnittsfalls eröffnet sich eine zusätzliche Schwierigkeit: es ist a priori nicht klar, welche Wahrscheinlichkeitsverteilung den Eingabedaten zugrunde liegt. Diese kann auch je nach Anwendung recht unterschiedlich ausfallen. Der Einfachheit halber arbeitet man oft mit einer (in geeignetem Sinn) uniformen Wahrscheinlichkeitsverteilung. In diesem Fall, der Sortierung von Datenfeldern, bedeutet das, dass wir für ein Eingabedatenfeld $A[1], \ldots, A[n]$ und seine Sortierung $A[\pi(1)], \ldots, A[\pi(n)]$ annehmen dass jede Permutation $\pi \in S_n$ gleich wahrscheinlich ist. D.h. jedes $\pi \in S_n$ tritt mit Wahrscheinlichkeit $\frac{1}{n!}$ auf. Man bezeichnet diese Annahme auch als *Permutationsmodell*. Dieses Modell für den allgemeinen Fall setzt auch die in Abbildung 1.1 betrachteten Eingabedaten für beliebige $n \geq 1$ fort. Wir verwenden die folgende Eigenschaft einer zufällig gewählten $\pi \in S_n$: Sei $1 \leq i \leq j \leq n$, dann ist

$$\mathrm{W}(\pi(j) \text{ ist das } i\text{-t-größte Element in } \{\pi(1), \ldots, \pi(j)\}) = \frac{1}{j}.$$

Diese Aussage werden wir im nächsten Kapitel beweisen. Die mittlere Anzahl von Aufrufen des inneren Schleifenkopfs ist dann also

$$\bar{t}_j = \frac{1}{j}1 + \frac{1}{j}2 + \cdots \frac{1}{j}j = \frac{1}{j}\frac{j(j+1)}{2} = \frac{j+1}{2}.$$

Damit erhalten wir

$$\begin{aligned}
\sum_{j=2}^{n} \bar{t}_j &= \frac{1}{2}\sum_{j=3}^{n+1} j = \frac{(n+1)(n+2)}{4} - \frac{3}{2} = \frac{n^2+3n}{4} - 1, \\
\sum_{j=2}^{n}(\bar{t}_j - 1) &= \frac{1}{2}\sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{4} \text{ und} \\
T_{\mathrm{d}}(n) &= (\frac{c_5}{4} + \frac{c_6}{4} + \frac{c_7}{4} + \frac{c_8}{4})n^2 \\
&\quad + (c_2 + c_3 + c_4 + c_9 + c_{10} + \frac{3c_5}{4} - \frac{c_6}{4} - \frac{c_7}{4} + \frac{3c_8}{4})n \\
&\quad - c_3 - c_4 - c_5 - c_8 - c_9,
\end{aligned}$$

d.h. also $T_{\mathrm{d}}(n) = kn^2 + ln + m$ für geeignete Konstanten k, l und m.

Mit dieser Analyse von Einfügesortieren haben wir also in einem gewissen Sinn eine Darstellung der Laufzeit auf *allen* Eingaben erhalten, womit wir das oben angesprochene erste Problem als behandelt betrachten wollen.

Was das zweite Problem angeht ist die entscheidende Beobachtung, dass sich die Wahl einer Programmiersprache, eines Compilers, etc. nur auf die c_i auswirkt, nicht aber darauf wie die Laufzeit von n abhängt. Insbesondere haben wir rechnerisch gezeigt, dass diese Abhängigkeit, wie man aufgrund von Abbildung 1.1 bereits vermuten könnte, im besten Fall linear ist und im schlechtesten und durschnittlichen Fall quadratisch. Um diese Information auf formale Weise darzustellen verwendet man die Landau-Symbole, auch "Groß-O-Notation" genannt ($O(f)$ steht für "Ordnung von f").

Definition 1.4. Sei $f : \mathbb{N} \to \mathbb{R}$ eine Funktion. Wir definieren

$$O(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} \text{ so dass } \forall n \geq n_0 \colon |g(n)| \leq c \cdot |f(n)|\}.$$

Falls $g \in O(f)$ sagen wir dass f eine *asymptotisch obere Schranke* von g ist. Sei weiters

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0, n_0 \in \mathbb{N} \text{ so dass } \forall n \geq n_0 \colon c \cdot |f(n)| \leq |g(n)|\}.$$

Falls $g \in \Omega(f)$ sagen wir dass f eine *asymptotisch untere Schranke* von g ist. Schließlich definieren wir noch

$$\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0, d > 0, n_0 \in \mathbb{N} \text{ so dass } \forall n \geq n_0 \colon c \cdot |f(n)| \leq |g(n)| \leq d \cdot |f(n)|\}.$$

Falls $g \in \Theta(f)$ sagen wir dass f eine *asymptotisch scharfe Schranke* von g ist.