Facharbeit zum Thema:

Vergleich von Listen mit Arrays des Moduls NumPy in Python

Facharbeit am Eduard-Spranger-Gymnasium Landau im Fach Informatik
Erstellt von Fabio Kron

Betreuende Lehrkraft des Fachbereichs Informatik: Frau Keller-Buttell
Abgabetermin: 10. Mai 2021

Kurzfassung

In dieser Facharbeit werden die Methoden `list.append()` für Listen und `numpy.append()` für Arrays der Programmbibliothek NumPy in Bezug auf Laufzeit- und Speicherentwicklung miteinander verglichen. Beide Methoden können verwendet werden, um Elemente an die dazugehörige Datenstruktur anzuhängen.

Wenn `numpy.append()` aufgerufen wird, um ein neues Element an das Array anzuhängen, wird das Array zuerst an eine neue Position im Speicher kopiert und erst dort wird das neue Element angehängt.

Bei der Verwendung von `list.append()` wird die Liste nur selten kopiert. Das ist unter anderem darauf zurückzuführen, dass systematisch Speicherplatz für neue Elemente reserviert wird, sodass dieser nicht durch andere Programmstrukturen blockiert werden kann. Dadurch hat die Liste an ihrer Position genug Platz, um neue Elemente anzuhängen und muss nicht zuerst an einen neuen Ort im Speicher kopiert werden.

Da das Kopieren einer Datenstruktur aufwendig ist, beansprucht die Methode `numpy.append()` in der Regel deutlich mehr Laufzeit als die Methode `list.append()`, während der Speicherbedarf der Datenstrukturen dabei relativ ähnlich ist.

Inhaltsverzeichnis

1 Einleitung

Bevor in die fachliche Thematik der Arbeit eingestiegen wird, wird in diesem Kapitel auf die Motivation, das Ziel sowie den Aufbau der Facharbeit eingegangen.

1.1 Motivation und Ziel der Facharbeit

Beim Entwickeln von Software hat man oft mehrere Möglichkeiten eine Lösung zu implementieren und es ist manchmal unklar, welche dieser Möglichkeiten sich am besten eignet. Die Ursache dafür ist häufig der Mangel an Wissen, wie eine potentielle Lösung funktioniert und welche Auswirkungen das hat. Dabei spielt manchmal auch Laufzeit und Speicherbedarf eine wichtige Rolle, da beides mit finanziellen Ressourcen verbunden ist.

Diese Facharbeit soll die Entwicklung der Laufzeit und des Speicherbedarfs der gleichnamigen Methoden `.append()`, die man sowohl im Zusammenhang mit Arrays der Programmbibliothek NumPy für Python als auch mit Listen in Python finden kann, vergleichen. Beide Methoden haben den selben Zweck und hängen Elemente an die Datenstrukturen an. Die Grundlage des Vergleichs bilden Messungen, die die Entwicklung des Speicherbedarfs und der Laufzeit festhalten.

Die gewonnenen Messdaten werden ausgewertet und interpretiert.

Hierbei handelt es sich jedoch nur um ein Fallbeispiel und die Erkenntnisse können nicht für andere Methoden der Datenstrukturen, die Verwendung der Methoden mit Elementen anderer Datentypen und ähnliche Fälle generalisiert werden.

1.2 Gliederung

Die Facharbeit gliedert sich wie folgt:

Kapitel 2: Grundlagen

Zu Beginn werden die theoretischen Grundlagen erläutert, um ein grundlegendes Verständnis der Thematik aufzubauen.

Kapitel 3: Durchführung

Daraufhin wird auf die Vorgehensweise bei den Messungen eingegangen und erklärt, worauf beim Erfassen der Messdaten geachtet wird.

Kapitel 4: Ergebnisse

In diesem Kapitel werden die Messdaten präsentiert, ausgewertet und interpretiert.

Kapitel 5: Fazit

Zum Abschluss werden die gewonnenen Erkenntnisse der Facharbeit bewertet. Außerdem wird verdeutlicht, welche Aspekte auf Grund des Umfangs der Facharbeit nicht behandelt werden konnten.

2 Grundlagen

In diesem Kapitel werden Grundlagen behandelt, die zum Verständnis der weiteren Kapitel der Facharbeit hilfreich sind.

2.1 Python

Python ist eine interpretierte, objektorientierte, dynamische Programmiersprache, die relativ leicht zu verstehen ist. Außerdem ist die Sprache ausgelegt für die schnelle Entwicklung von Programmen und eignet sich daher auch als Skriptsprache. Zusätzlich werden Module und Programmbibliotheken unterstützt, die bereits Lösungen für Problemstellungen bereitstellen. Module und Programmbibliotheken sind sozusagen Erweiterungen, die man verwenden kann, um mehr als nur die Standardmöglichkeiten, die die Programmiersprache bietet, nutzen zu können.

Quelle: [PYO1]

2.2 NumPy

NumPy ist eine externe Programmbibliothek, die zusätzlich zu Python installiert werden kann. Die Bibliothek bietet unter anderem die Verwendung von zusätzlichen Datenstrukturen wie Arrays an.

2.3 Datenstrukturen

Eine Datenstruktur ist ein gewisses System, nach dem Datensammlungen mehrerer Elemente strukturiert gespeichert werden können. Zwei häufig verwendete Datenstrukturen sind Arrays und Listen.

Quellen: [INF1], [INF2]

2.4 Anhängen von Elementen an Listen

Die Datenstruktur Liste ist eine endliche Folge von Elementen, der Elemente hinzugefügt oder entfernt werden können. Bei der Installation von Python sind Listen standardmäßig enthalten und werden in der Praxis sehr häufig verwendet. Ist in dieser Facharbeit von Listen die Rede, so handelt es sich um die Listen der Programmiersprache Python.

Beim Arbeiten mit Listen werden häufig Elemente angehängt. Dabei wird die Datenstruktur um das neue Element am Ende erweitert und die ursprünglichen Elemente bleiben erhalten.

Bei Listen wird dafür die Methode `.append()` auf die Liste mit dem neuen Element als Argument aufgerufen:

```
list.append(neuesElement)
```

Beim Anhängen von Elementen wird der belegte Speicherplatz durch die Liste manchmal erweitert. Beim Erweitern des Speicherplatzes der Liste wird zuzüglich ein Puffer von rund $\frac{1}{8}$ der benötigten Größe freigehalten. Dadurch muss der Speicherbedarf der Liste nicht immer verändert werden, wenn neue Elemente angehängt werden, sondern nur, wenn der freigehaltene Speicherplatz nicht mehr ausreicht. Reicht der Speicherplatz aus, so wird das neue Element, das der Liste angehängt werden soll, in den nächsten freien Platz des beanspruchten Speichers der Liste eingefügt. Da die Liste ihre Größe speichert und der benötigte Platz jedes Elements gleich groß ist, kann berechnet werden, wo das nächste Element gespeichert werden soll. Dadurch hat das Anhängen von Elementen nach [PYO3] in der Regel eine Laufzeitkomplexität von O(1).

Quellen: [INF3], [HTD1], [PYO2], [PYO3]

2.5 Anhängen von Elementen an Arrays

Arrays sind ähnlich wie Listen endliche Folgen von Elementen. Der große Unterschied ist jedoch, dass bei Arrays in der Regel die Größe nicht veränderbar ist, also keine Elemente hinzugefügt oder entfernt werden können. Bei den Arrays, die von der Programmbibliothek NumPy zur Verfügung gestellt werden, gibt es jedoch auch die Möglichkeit, dem Array Elemente hinzuzufügen oder zu entfernen. Wird in dieser Facharbeit der Begriff Array verwendet, so ist damit ein Array gemeint, das durch das Modul NumPy in Python bereitgestellt wird.

Wenn man einem Array Elemente hinzufügen will, wird das Array immer an einen neuen Ort verschoben und erst dann wird das neue Element angehängt. Es wird nicht wie bei Listen Speicher für neue Elemente freigehalten. Um Elemente anzuhängen wird hier die Methode `numpy.append()` verwendet, die das neue Array zurückgibt. Als Argumente müssen das alte Array und das neue Element übergeben werden:

```
neuesArray = numpy.append(altesArray, neuesElement)
```

Da das Array durch jedes Anhängen von Elementen aufwendig neu platziert werden muss, entwickelt sich nach [GIH1] beim Anhängen von mehreren Elementen ein exponentielles Wachstum mit der Laufzeitkomplexität $O(n^2)$.

Quellen: [NPY1], [GIH1]

2.6 Datentypen

Datentypen bestimmen, wie ein Computer gewisse Daten speichert. Beispielsweise werden Fließkommazahlen anders gespeichert als ganze Zahlen. Das hat auch Auswirkungen auf den Speicherplatz, der durch die Daten eingenommen wird. Bei ganzen Zahlen ist in Python beispielweise der Speicherbedarf von der Größe der Zahl abhängig.

Relevant für diese Facharbeit ist der Datentyp Float, der es ermöglicht Fließkommazahlen zu speichern. In Python ist die Größe dieses Datentyps konstant.

3 Durchführung

Dieses Kapitel geht auf die Umsetzung der Messungen ein. Es wird die Implementierung des Algorithmus zur Messung erläutert und erklärt, auf welchem System die Messungen durchgeführt werden und worauf noch geachtet wird.

Zunächst geht es um den Algorithmus zum Erfassen der Messdaten. Im wesentlichen hängt der Algorithmus in 10000 Wiederholungen den Datenstrukturen jeweils ein Element pro Wiederholung an und erfasst für jede dieser Wiederholungen folgende Daten für das Array und die Liste:

Gesamtlaufzeit:	Gesamte Laufzeit, die insgesamt benötigt wurde, um alle bisher hinzugefügten Elemente der Datenstruktur anzuhängen. Einheit: Sekunden
Laufzeitveränderung:	Benötigte Laufzeit für das Anhängen des letzten Elements. Einheit: Sekunden
Belegter Speicherplatz:	Gesamter Speicherbedarf der Datenstruktur Einheit: Bytes
Speicherveränderung:	Veränderung des belegten Speicherplatzes durch das Anhängen des letzten Elements. Einheit: Bytes

Die erfassten Daten werden in separaten Dateien gespeichert.

Als Element, das den Datenstrukturen angehängt wird, wird für jede Wiederholung eine Zufallszahl des Datentyps Float zwischen null und eins generiert, da diese eine konstante Größe hat und somit ungewöhnliche Entwicklungen durch verschieden große Elemente vermieden werden.

Insgesamt wird der Algorithmus 50 mal wiederholt, sodass 50 Dateien generiert werden, die jeweils Messdaten zum Hinzufügen von 10000 Elementen an Arrays und Listen enthalten. Dadurch wird es möglich den Effekt von Abweichungen durch äußere Faktoren zu minimieren und aussagekräftige Ergebnisse zu erhalten.

Für jede der 10000 Wiederholungen des Anhängen wird der Mittelwert aus den 50 Werten der 50 verschiedenen Dateien berechnet und zusätzlich die größte Abweichung zum Mittelwert.

Die Messungen werden in einer virtuellen Maschine mit einem Arbeitsspeicher von 4096MB, einem Prozessor mit einem Kern und einer Taktfrequenz von 3,00 GHz, dem Betriebssystem Ubuntu 20.04.2.0 und Python 3.8.5 als Interpreter durchgeführt.

Die folgende Abbildung zeigt einen Programmablaufplan des Algorithmus zum Dokumentieren der Laufzeit und des Speichers. Der abgebildete Algorithmus generiert eine Datei, die die Messdaten des hinzufügen von 10000 Elementen an einen Array und eine Liste enthält.

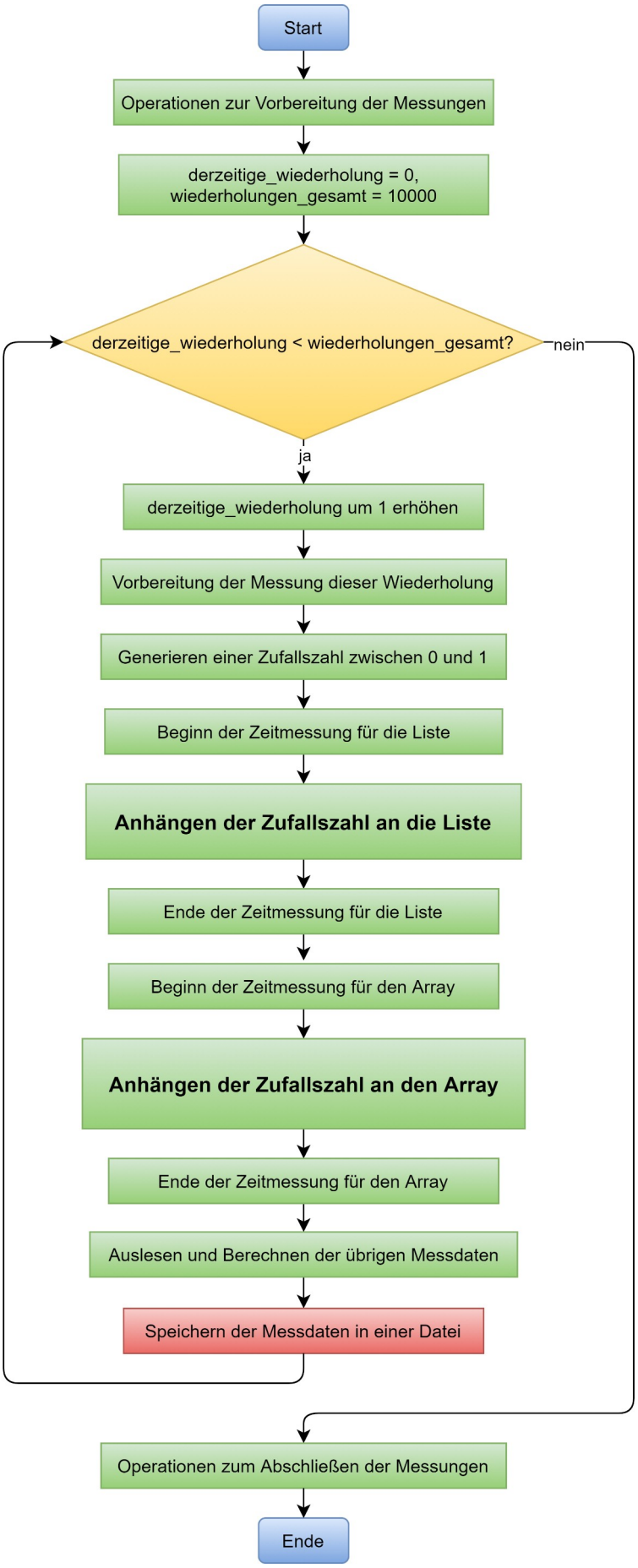

Abbildung 1: Programmablaufplan des Algorithmus zum Erfassen der Messdaten

4 Ergebnisse

Nun werden die Ergebnisse der Messungen zur Laufzeit und dem Speicher in Form von Diagrammen dargestellt und mit Bezug auf die Grundlagen interpretiert.

4.1 Speicherplatzverwaltung ist fest vorgegeben

In diesem Kapitel wird verdeutlicht, dass die Speicherplatzverwaltung fest vorgegeben ist.

Das folgenden Diagramm 1 zeigt die Mittelwerte der Speicherbelegung in Bytes nach Anzahl an hinzugefügten Elementen.

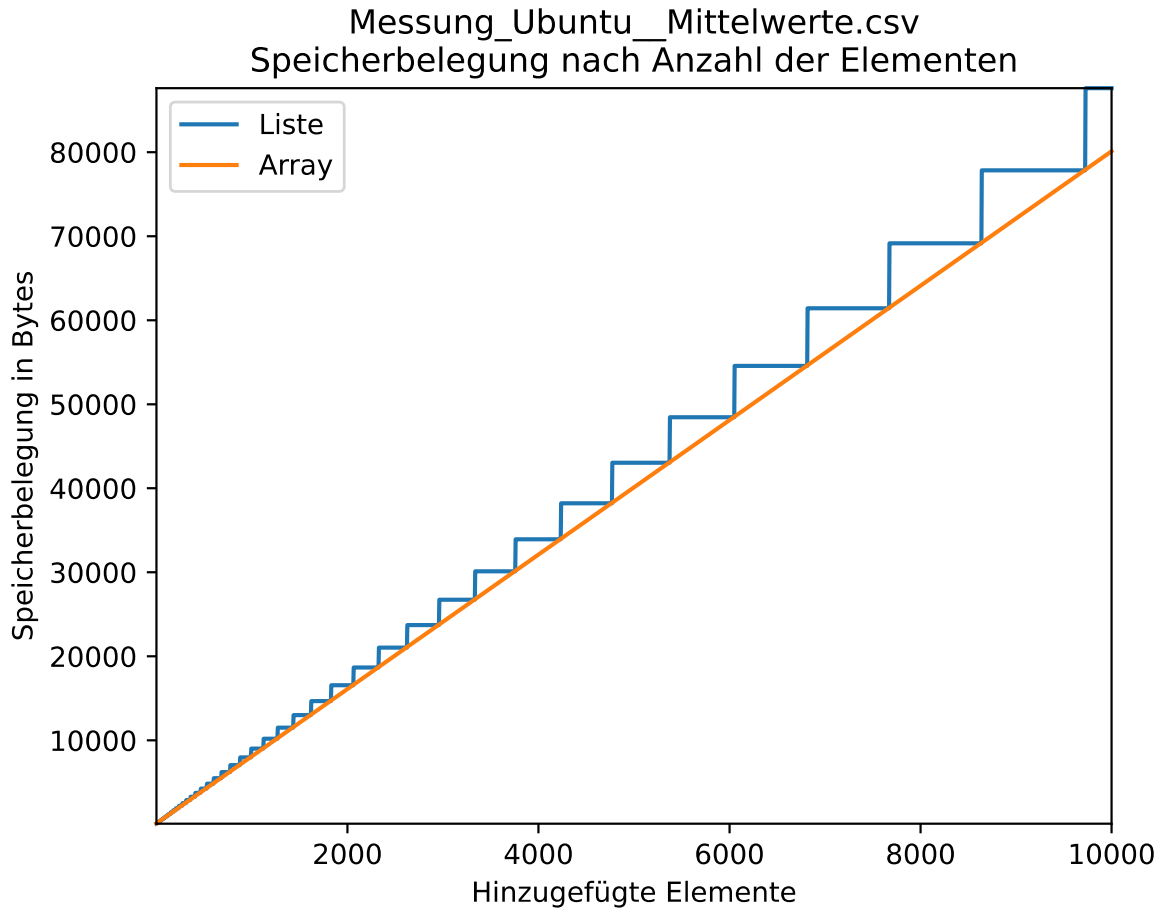

Diagramm 1: Speicherbedarf gesamt

Hier ist zu erkennen, dass der belegte Speicher des Arrays linear ansteigt, während der belegte Speicher der Liste stufenweise ansteigt. Es ist außerdem zu erkennen, dass sowohl die Speicherplatzveränderungen der Liste als auch die Abstände dazwischen systematisch größer werden.

Das nächste Diagramm 2 zeigt die Maximalabweichung der Messdaten zum belegten Speicher nach Anzahl an Elementen zu den Mittelwerten an.

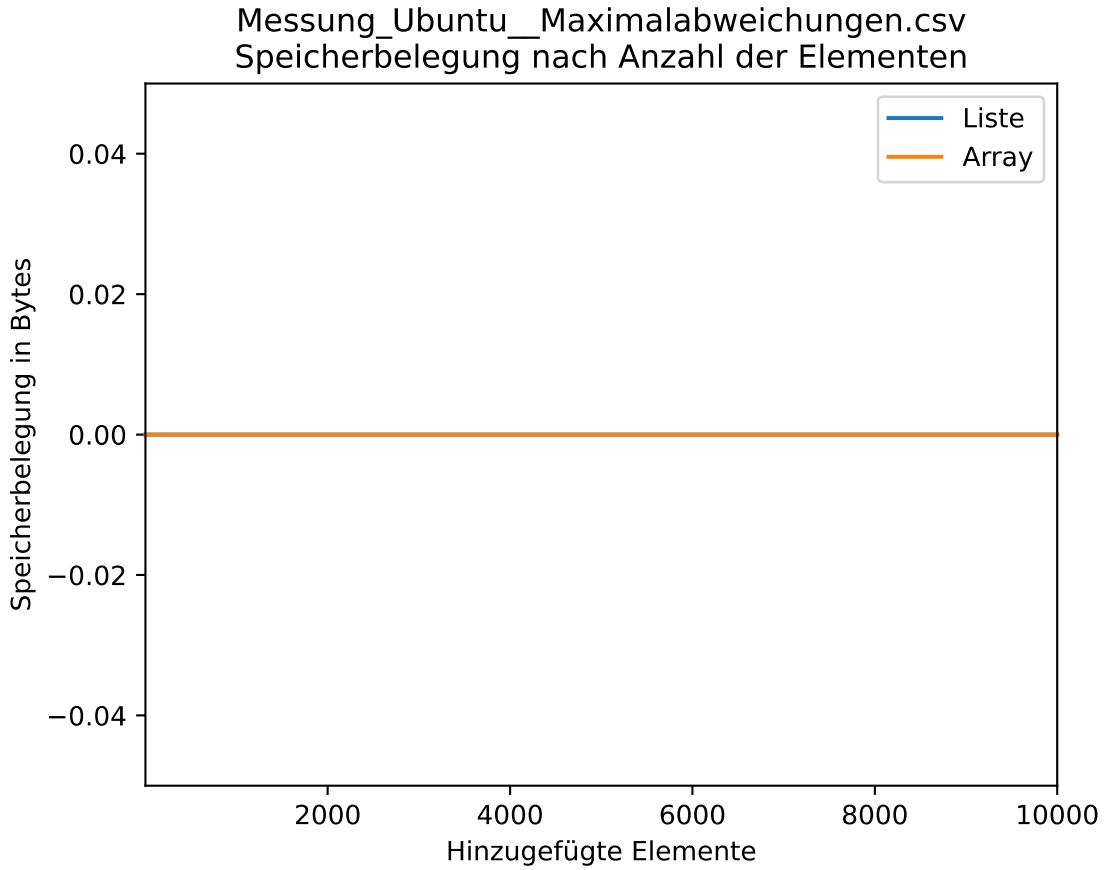

Diagramm 2: Maximalabweichungen Speicherbedarf gesamt

Die Maximalabweichung zum Mittelwert bei der Speicherbelegung beträgt sowohl für Array als auch für Liste für alle Messwerte null Bytes.

Dadurch lässt sich erschließen, dass die Speicherverwaltung fest vorgegeben ist.

In den nächsten Kapitel wird auf die Betrachtung einer konkreter Messungen zum Speicherplatzes verzichtet und auf die Mittelwerte zurückgegriffen, weil alle Messergebnisse zum Speicher identisch sind, wie sich durch die Maximalabweichung von null Bytes für alle Stellen erkennen lässt.

4.2 Speicherplatzverwaltung bei Arrays

Im folgenden Abschnitt wird die Entwicklung der Speicherplatzverwendung bei Arrays analysiert und in Bezug zu den Grundlagen gesetzt.

Durch Diagramm 3 wird verdeutlicht, wie sich der Speicherbedarf durch das Hinzufügen von Elementen verändert.

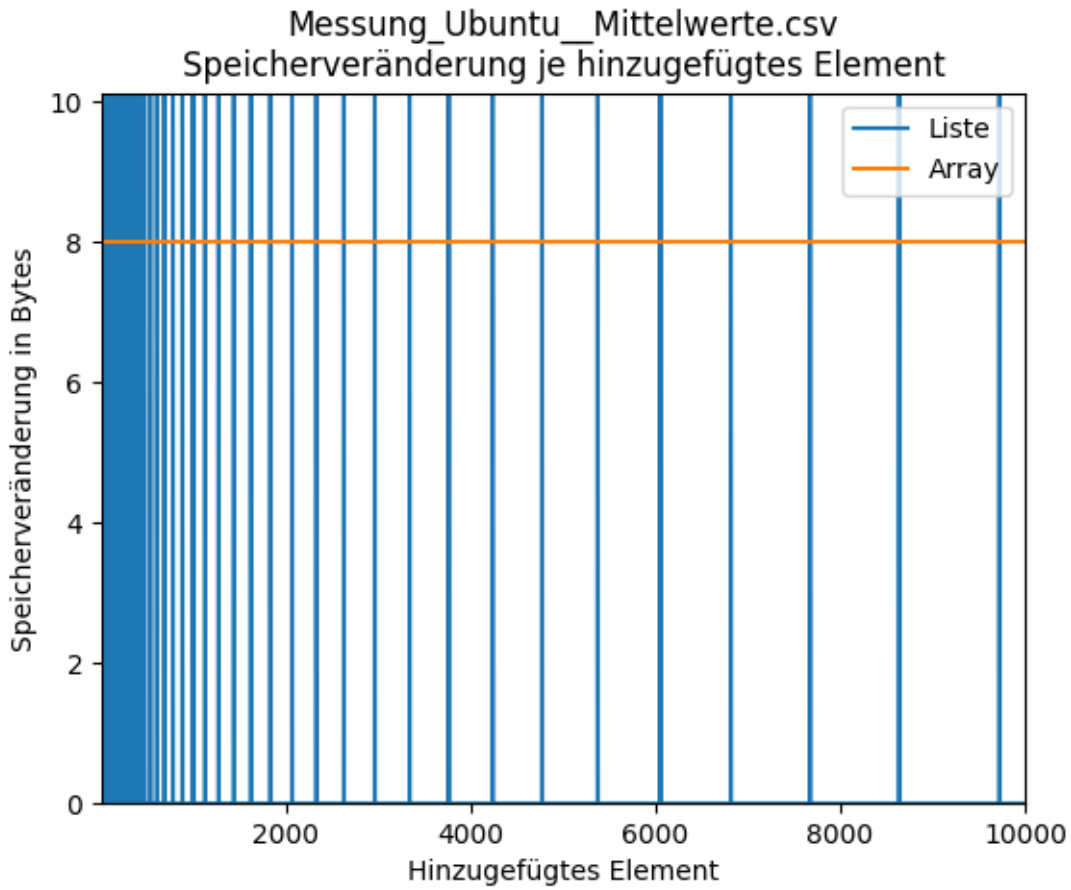

Diagramm 3: Speicherveränderung Array

In Diagramm 3 kann man ablesen, dass sich der Speicher für jedes hinzugefügte Element um acht Bytes vergrößert. Somit erklärt sich auch das lineare Wachstum, welches man in Diagramm 1 erkennen kann.

Das lineare Wachstum ist darauf zurückzuführen, dass bei jedem Hinzufügen eines Elements ein neuer Array aus dem alten Array und dem neuen Element erstellt wird und dabei kein Speicherplatz für das Hinzufügen von weiteren Elementen eingeplant wird.

4.3 Speicherplatzverwaltung bei Listen

Nun geht es um das Wachstum des belegten Speichers bei Listen.

Wie man in Diagramm 1 erkennen kann, wächst die Speicherbelegung stufenweise, anstatt linear wie bei Arrays. Das ist der Fall, weil bei jeder Vergrößerung des Speicherbedarfs der Liste rund $\frac{1}{8}$ des gerade benötigten Speicherplatzes zusätzlich reserviert wird. Dadurch muss für neue Elemente kein neuer Speicherplatz beansprucht werden, sofern noch genug Platz für neue Element freigehalten ist. Dadurch können Neupositionierungen der Listen vermieden werden.

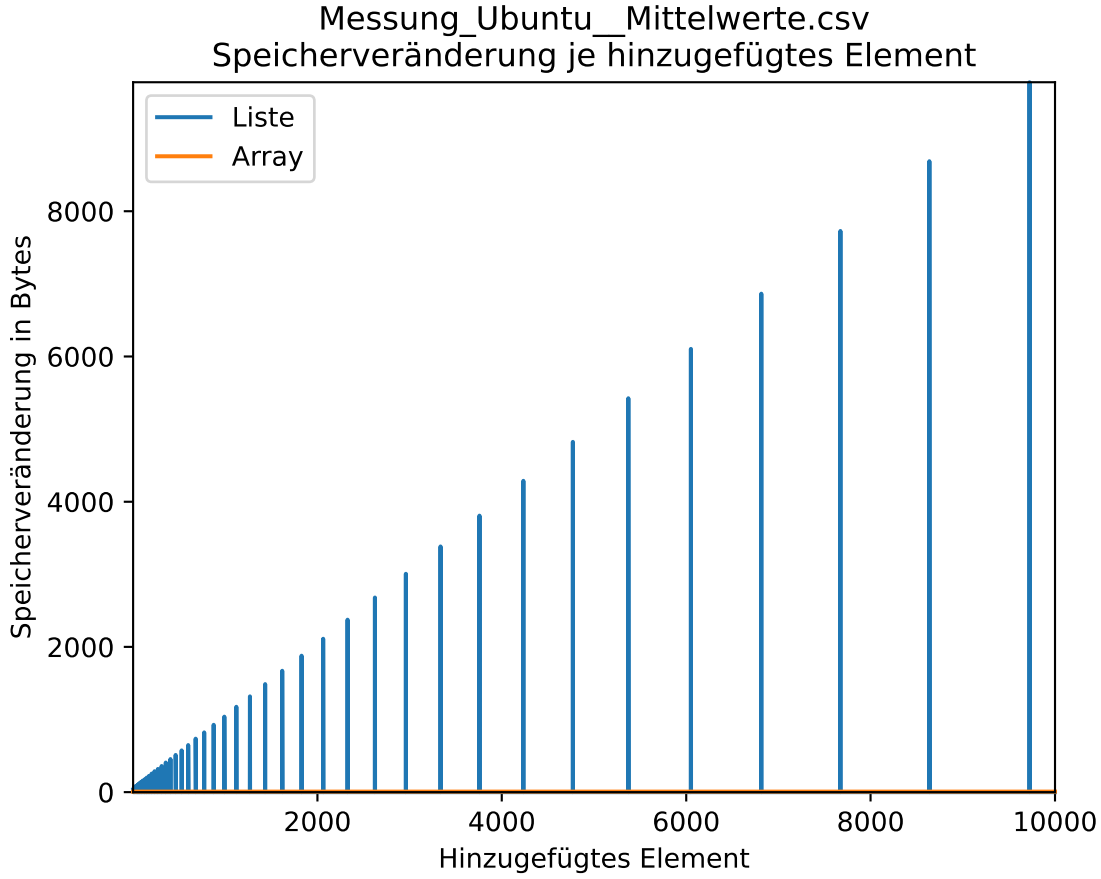

Diagramm 4: Speicherveränderung Liste

In Diagramm 4 ist abzulesen, dass die Größe der Speicherveränderungen linear steigt, dafür aber die Abstände zwischen den Speichervergrößerungen auch steigen, da nur dann der Speicher vergrößert wird, wenn kein freigehaltener Platz für neue Element vorhanden ist. Das ist unter anderem in Diagramm 1 sehr deutlich zu erkennen. Der belegte Speicherplatz durch die Liste steigt immer dann an, wenn diese ungefähr so viel Platz belegt wie das Array, bei dem kein Platz für neue Elemente reserviert wird. Das heißt auch, dass die Elemente, die den Listen hinzugefügt werden, sowohl bei Arrays und Listen genau gleich viel Speicherplatz in Anspruch nehmen, also acht Bytes (siehe 4.2).

In Diagramm 5 wird deutlich, dass nach rund 9700 hinzugefügten Elementen und einer Speicherbelegung von rund 78000 Bytes eine Vergrößerung des beanspruchten Speichers der Liste stattfindet.

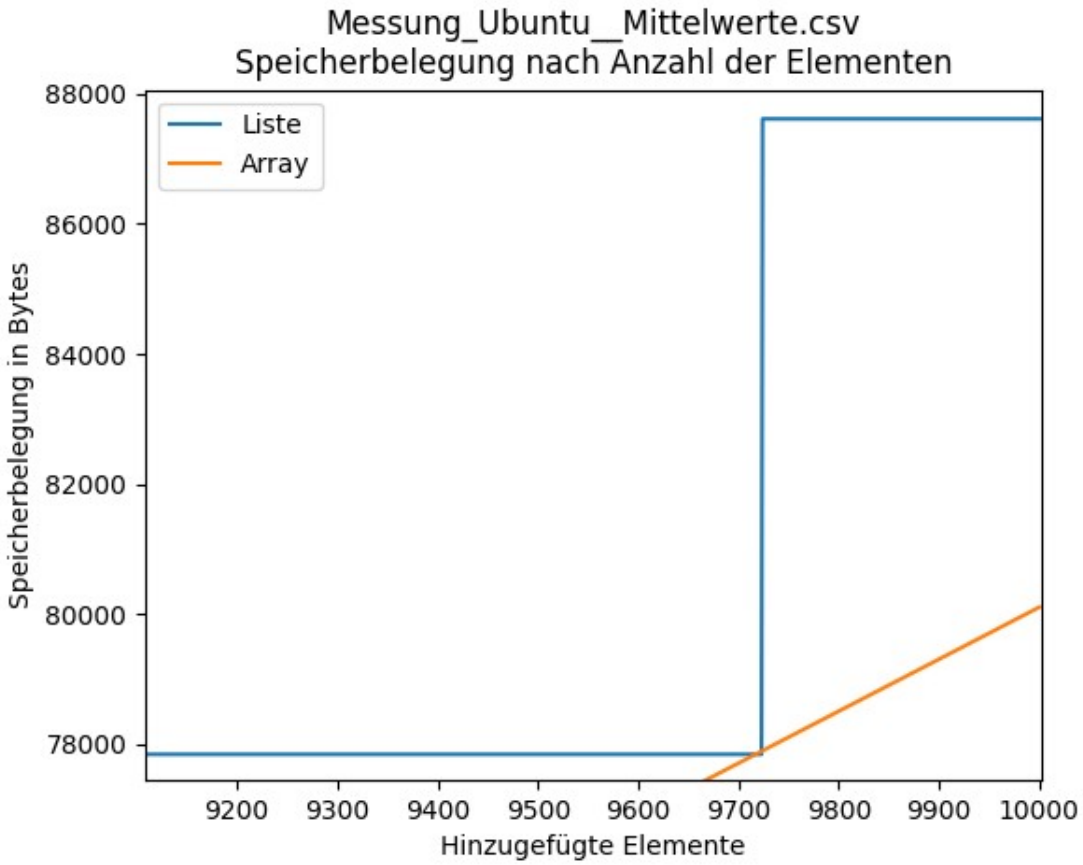

Diagramm 5: Speicherbedarf um Element 9700

In D6 kann man erkennen, dass bei genau der gleichen Speicherplatzvergrößerung nach rund 9700 hinzugefügten Elementen, der belegte Speicherplatz um rund 9800 Bytes steigt. Zum Vergleich: 78000Bytes * $\frac{1}{8}$ = 9750Bytes. Somit wird verdeutlicht, dass die Speichervergrößerung bei der Liste tatsächlich rund $\frac{1}{8}$ des zuvor belegten Speichers beträgt.

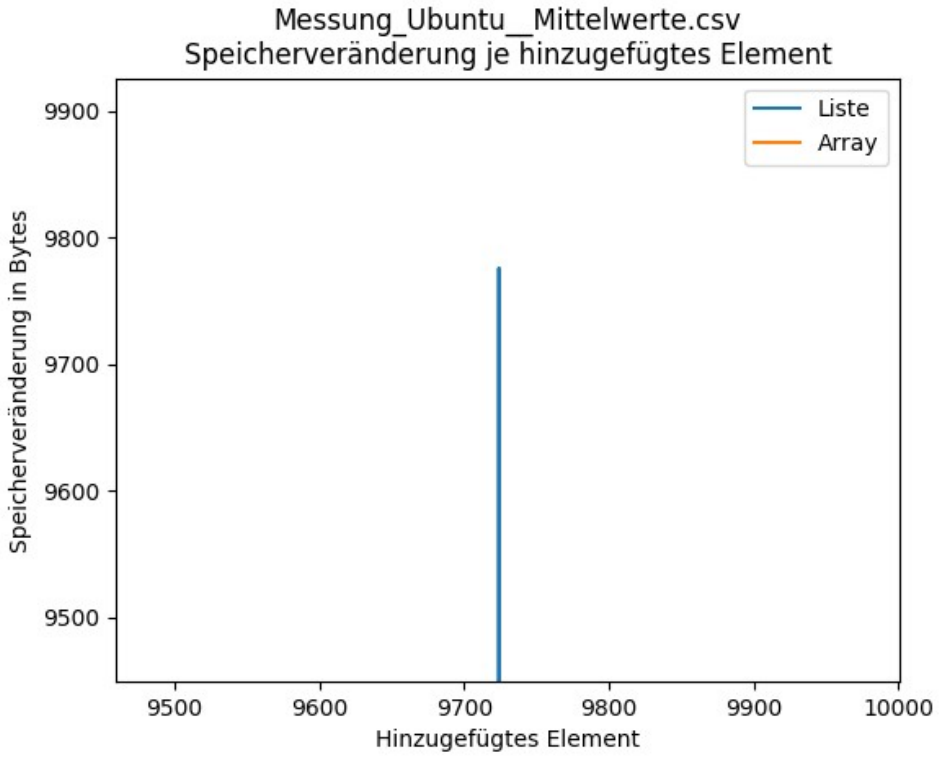

Diagramm 6: Speicherveränderung um Element 9700

4.4 Ungenauigkeiten bei Messungen der Laufzeit

Bevor in den nächsten Kapitel die gemessenen Laufzeiten zu den Arrays und Listen in Bezug auf die Grundlegenden Informationen analysiert und interpretiert werden, wird es zuerst um die Ungenauigkeiten der Messungen gehen. Die Diagramme dieses Abschnitts repräsentieren alle die Messwerte der gleichen zufällig ausgewählten Messung.

Diagramm 7 bildet die Laufzeitveränderung je hinzugefügtes Element ab.

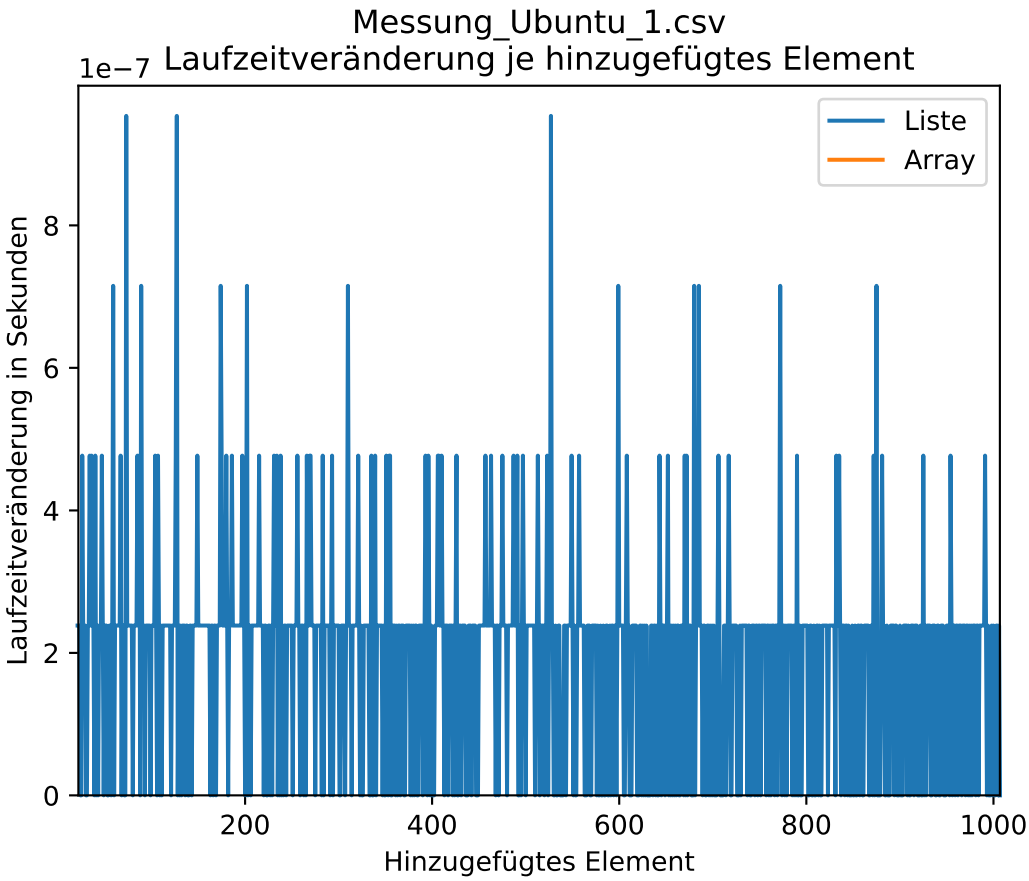

Diagramm 7: Beispielmessung Laufzeitveränderung Liste bis Element 1000

Hier ist zu erkennen, dass sich die abgebildeten Laufzeitveränderungen an verschiedenen Werten orientieren und alle ein Nullfaches, Zweifaches, Dreifaches oder Vierfaches vom kleinsten Wert über null seien. Dieser beträgt in diesem Diagramm etwas mehr als $2*10^{-7}$ Sekunden.

Außerdem zeigt das Diagramm, dass in manchen Fällen die Laufzeitveränderung null Sekunden sei. Das heißt, dass ein Element an die List hinzugefügt wird ohne dass Zeit vergeht. Das ist technisch nicht möglich.

Deshalb ist es vermutlich so, dass das Ergebnis der hier verwendete Methode der Zeitmessung nur in gleichmäßigen Schritten steigen kann, die in diesem Beispiel rund $2*10^{-7}$ Sekunden betragen.

Ein anderes interessantes Merkmal wird in Diagramm 8 deutlich.

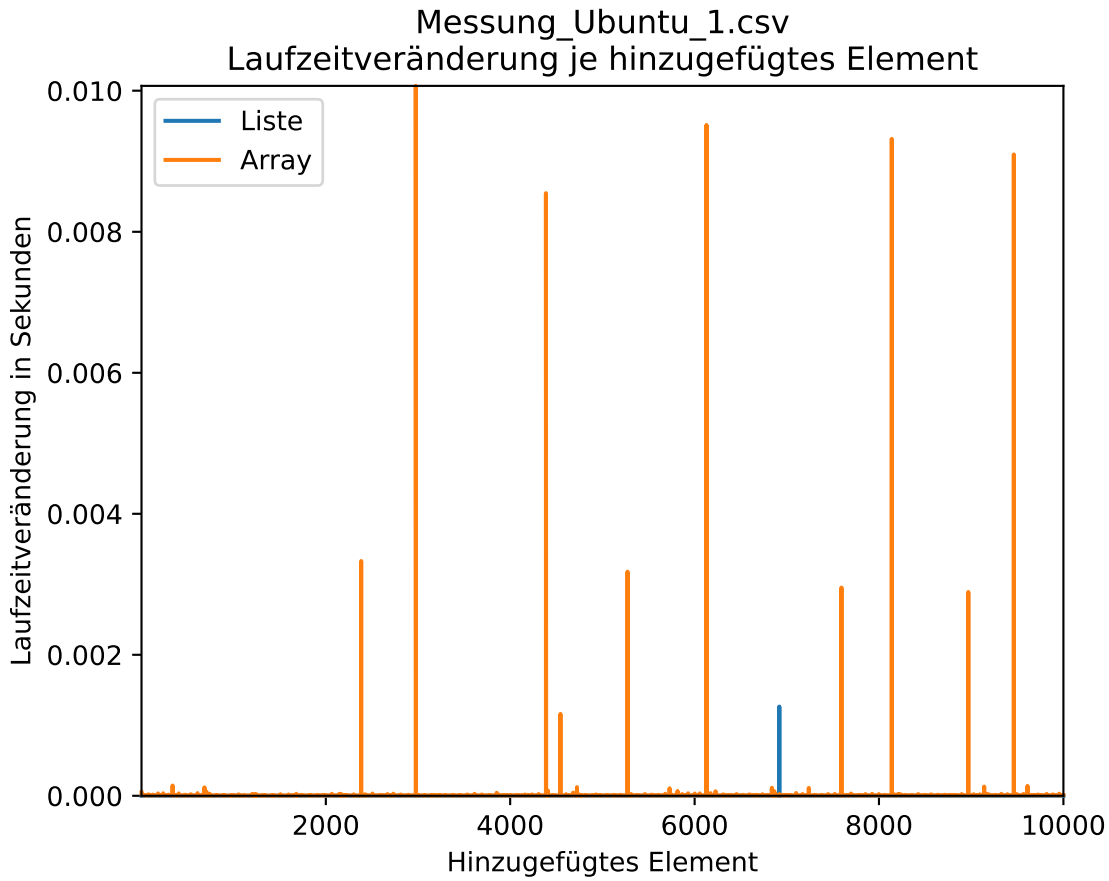

Diagramm 8: Beispielmessung Laufzeitveränderung gesamt

Hier ist zu erkennen, dass beim Hinzufügen der meisten Elemente die Laufzeitveränderung sehr klein ist, diese in Einzelfällen jedoch extrem hoch sein kann. Bei dieser Messung ist das vor allem beim Array der Fall. Da Arrays bei jedem Hinzufügen von Elementen neu platziert werden, lassen sich diese extremen Abweichungen nicht auf eine Speicherveränderung, sondern auf äußere Faktoren wie Hintergrundprozesse des Betriebssystems zurückführen.

4.5 Laufzeitentwicklung bei Listen

In diesem Kapitel geht es darum, wie sich die Laufzeit beim Anhängen von Elementen an eine Liste entwickelt. Diagramm 9 stellt die Mittelwerte der Laufzeitveränderung für jedes hinzugefügte Element dar.

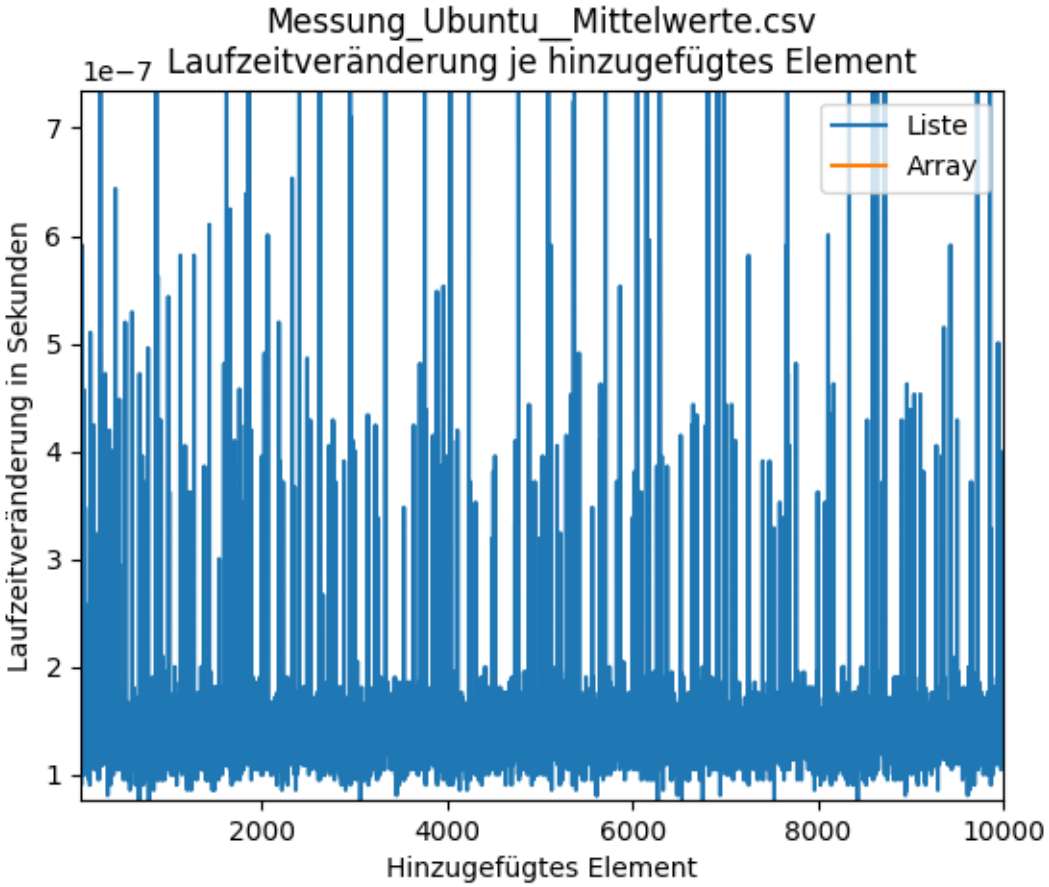

Diagramm 9: Mittelwerte Laufzeitveränderung Liste gesamt

Hier ist zu erkennen, dass abgesehen von einigen Ausreißern die meisten Mittelwerte der Laufzeitveränderungen zwischen $1*10^{-7}$ Sekunden und $2*10^{-7}$ Sekunden liegen. Es ist nicht zu erkennen, dass die Laufzeitveränderung beim Hinzufügen von Elementen an eine längere Liste größer wird. Dementsprechend ist die Laufzeitveränderung unabhängig von der Anzahl der Elemente in einer Liste. Wenn man von den extremen Abweichungen absieht, führt das ungefähr zu einem linearen Wachstum der Gesamtlaufzeit. Das gleichbleibende Verhalten der Laufzeitveränderung ist darauf zurückzuführen, dass beim Anhängen von Elementen die Liste nicht neu platziert werden muss, sondern die neuen Elemente durch den freigehaltenen Platz ohne großen Aufwand gespeichert werden können.

In Diagramm 10 ist das lineare Wachstum der Gesamtlaufzeit bei der Liste abgesehen von den großen Sprüngen durch die extremen Abweichungen zu erkennen .

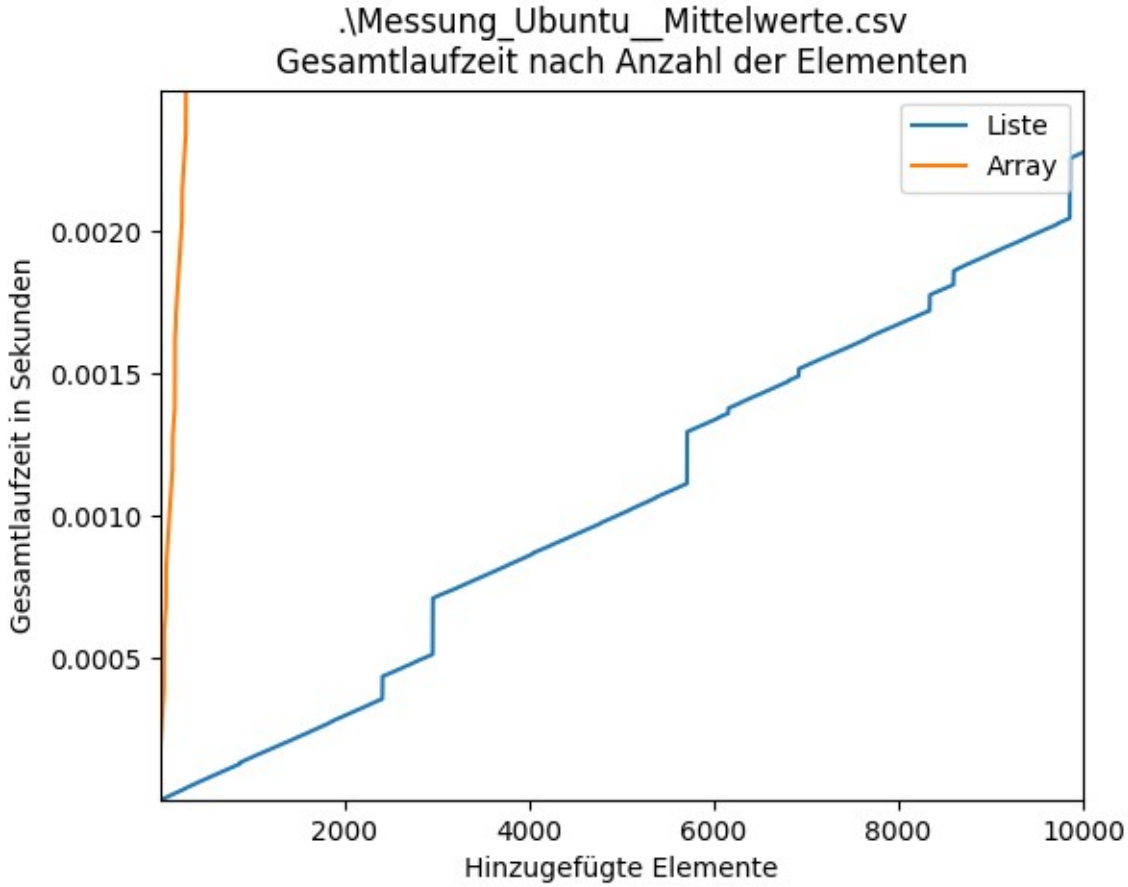

Diagramm 10: Mittelwerte Gesamtlaufzeit Liste gesamt

4.6 Laufzeitentwicklung bei Arrays

Im Vergleich zu der Laufzeitentwicklung bei Listen steht die Laufzeitentwicklung bei Arrays. Das folgende Diagramm 11 stellt die Mittelwerte der Laufzeitveränderung für jedes hinzugefügte Element bei Arrays dar.

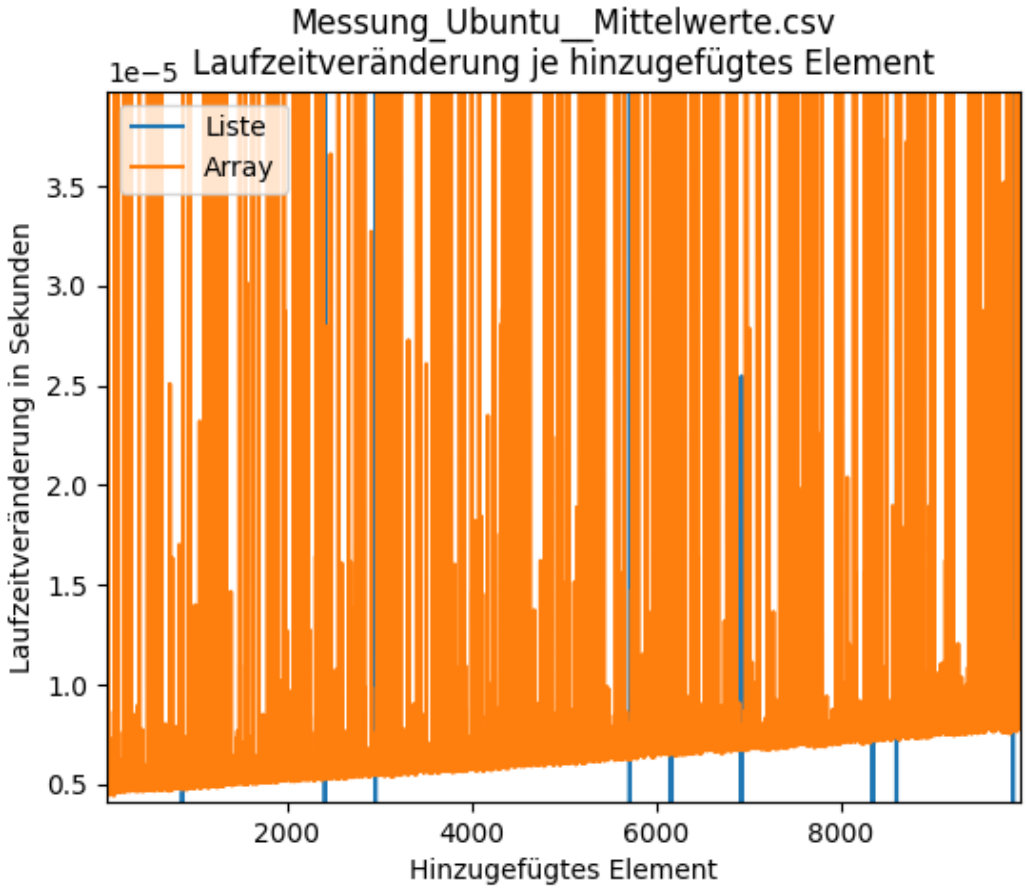

Diagramm 11: Mittelwerte Laufzeitveränderung Array Unterkante

Hier ist zu erkennen, dass abgesehen von den vielen Abweichungen die Laufzeitveränderung von anfangs rund 0,5 * 10^{-5} Sekunden auf fast 1 * 10^{-5} Sekunden nach 8000 Elementen linear ansteigt. Dieser lineare Anstieg wird dadurch verursacht, dass Arrays beim Hinzufügen von neuen Elementen aufwendig an einen neuen Ort positioniert werden und der Aufwand mit der Anzahl der Elementen im Array wächst.

Das führt zu einem exponentiellen Anstieg der Gesamtlaufzeit. Dieser ist jedoch undeutlich, da es extrem viele große Abweichungen gibt, bei denen die Laufzeitveränderung deutlich über den oben genannten Werten liegt. Diese Abweichungen kommen vermutlich durch Hintergrundprozesse, die nichts mit dem Hinzufügen von Elementen an Arrays zu tun haben. Die Abweichungen sind hier häufiger als bei Listen, da das Hinzufügen von Elementen bei Arrays durch das neue platzieren aufwendiger ist und somit die Wahrscheinlichkeit steigt, dass der Prozess durch Hintergrundprozesse unterbrochen wird. Der größere Aufwand bei Arrays ist auch in den Diagrammen zu erkennen:

In Diagramm 11 kann man ablesen, dass die kleinsten Mittelwerte beim Hinzufügen von Elementen an Arrays eine Laufzeitveränderung von etwas weniger als 0,5 * 10^{-5} Sekunden betragen während die kleinsten Mittelwerte nach Diagramm 9 bei Listen kleiner als $1*10^{-7}$ Sekunden sind.

Der oben angesprochene exponentielle Anstieg der Gesamtlaufzeit bei Arrays ist in Diagramm 12 auf Grund der vielen großen Abweichungen nur undeutlich zu erkennen.

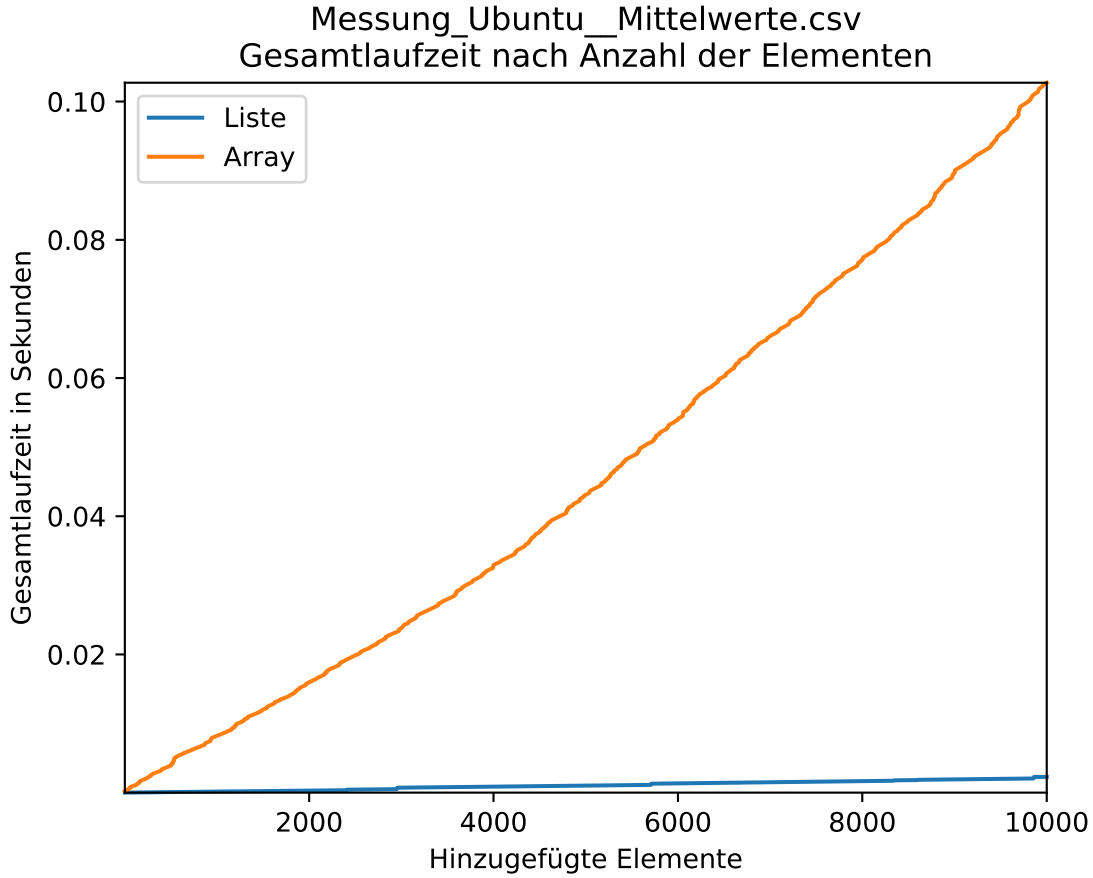

Diagramm 12: Mittelwerte Gesamtlaufzeit gesamt

Man kann den exponentiellen Anstieg jedoch leicht erahnen, da nach rund 5000 hinzugefügten Elementen die Gesamtlaufzeit ungefähr 0,04 Sekunden beträgt, während sie nach 10000 Elementen bei rund 0.10 Sekunden liegt. In dem Diagramm ist auch der Vergleich zwischen dem Hinzufügen von Elementen an Listen und Arrays abgebildet. Es ist sehr deutlich zu erkennen, dass Arrays dafür deutlich länger als Listen benötigen.

5 Fazit

Nun werden die gewonnenen Erkenntnisse bewertet und es wird einen Blick auf nicht behandelte Problemstellungen geworfen.

5.1 Bewertung

Beim Anhängen von Elementen an die Datenstruktur ist eindeutig festzustellen, dass Arrays für das Anhängen von Elementen eine deutlich höhere Laufzeit beanspruchen als Listen, während der Speicherbedarf der Datenstrukturen ungefähr ähnlich ist. Auch wenn das Anhängen von einzelnen Elementen an den Array auf diese Weise hilfreich sein kann, würde ich auf Grund der gewonnenen Erkenntnisse beim Anhängen einer großen Zahl von Elementen an die Datenstruktur eine Alternative wählen. Eine Möglichkeit wäre beispielsweise das Array in eine Liste umzuwandeln und dann Elemente anzuhängen.

Eventuell wäre es hilfreich gewesen, neben dem Mittelwert auch den Median als Mittel zur Auswertung der Daten zu verwenden. Dadurch wären die extremen Abweichungen nicht so stark ins Gewicht gefallen.

5.2 Ausblick

Das Analysieren einer Methode reicht nicht aus, um sich ein Gesamtbild über den Vergleich von Listen und Arrays zu verschaffen. Bei beiden Datenstrukturen spielen auch noch weitere Aspekte eine wichtige Rolle, die für einen grundlegenden Vergleich essentiell wären. Beispiele dafür sind weitere Methoden, spezifische Aspekte, die nur von der jeweiligen Datenstruktur geboten werden, oder das Verhalten bei Mehrdimensionalität. Auf Grund des Umfangs der Arbeit konnten diese Aspekte nicht umgesetzt werden.

Diese Facharbeit war nur ein kleiner Schritt auf dem Weg, die optimalen Lösungen für Probleme beim Entwickeln von Software zu finden. Vermutlich gibt es eine unendliche Anzahl an Problemen beim Programmieren, die auf verschieden Weisen gelöst werden können. Somit wären auch andere dieser Fallbeispiele Material für weitere Analysen, die den Rahmen dieser Arbeit jedoch überschreiten.

6 Anhang

6.1 Quellenverzeichnis

[GIH1] **Github**, „Why is np.append so slow?“, https://github.com/numpy/numpy/issues/17090, Zugriff am 26.4.2021

[HTD1] **Hacking the Desert**, „Exploring Python’s Implementation of List.append() in C“, https://lisantra.wordpress.com/2020/04/24/python-internals-list-append/, Zugriff am 26.4.2021

[INF1] **Inf-Schule**, „Was tun Datenstrukturen?“, https://www.inf-schule.de/algorithmen/suchbaeume/datenstrukturen/was, Zugriff am 26.4.2021

[INF2] **Inf-Schule**, „Fachkonzept – Datenstrukturen“, https://www.inf-schule.de//programmierung/imperativeprogrammierung/konzepteimp/datenstrukturen/konzept_datenstrukturen, Zugriff am 26.4.2021

[INF3] **Inf-Schule**, „Fachkonzept- Liste“, https://www.inf-schule.de/algorithmen/listen/datenstrukturliste/konzept_liste, Zugriff am 26.4.2021

[NPY1] **NumPy**, „numpy.append“, https://numpy.org/doc/stable/reference/generated/numpy.append.html#numpy.append, Zugriff am 26.4.2021

[PYO1] **Python**, „What is Python? Executive Summary“, https://www.python.org/doc/essays/blurb/, Zugriff am 26.4.2021

[PYO2] **Python**, „Design and History FAQ“, https://docs.python.org/3/faq/design.html, Zugriff am 26.4.2021

[PYO3] **Python**, „TimeComplexity“, https://wiki.python.org/moin/TimeComplexity, Zugriff am 26.4.2021

6.2 Abbildungsverzeichnis

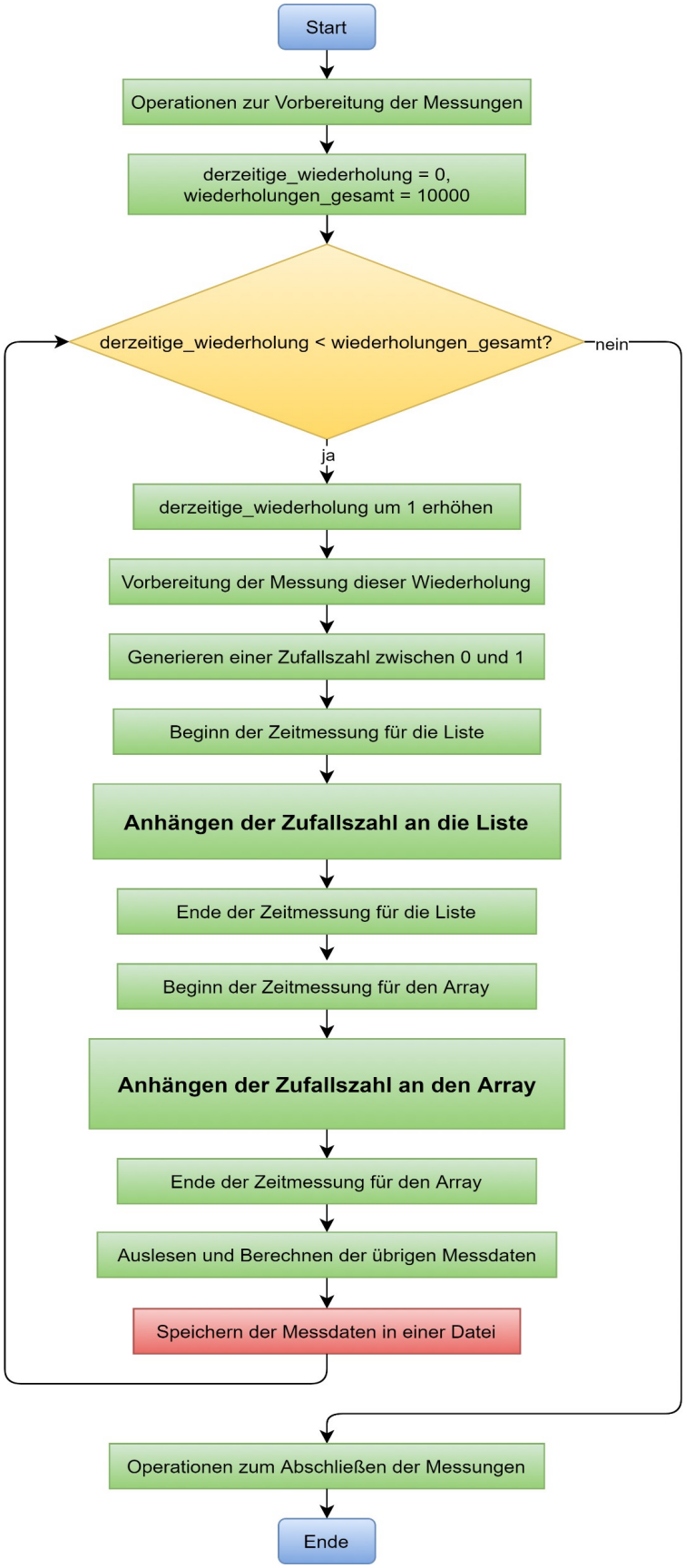

Abbildung 1: Programmablaufplan des Algorithmus zum Erfassen der Messdaten

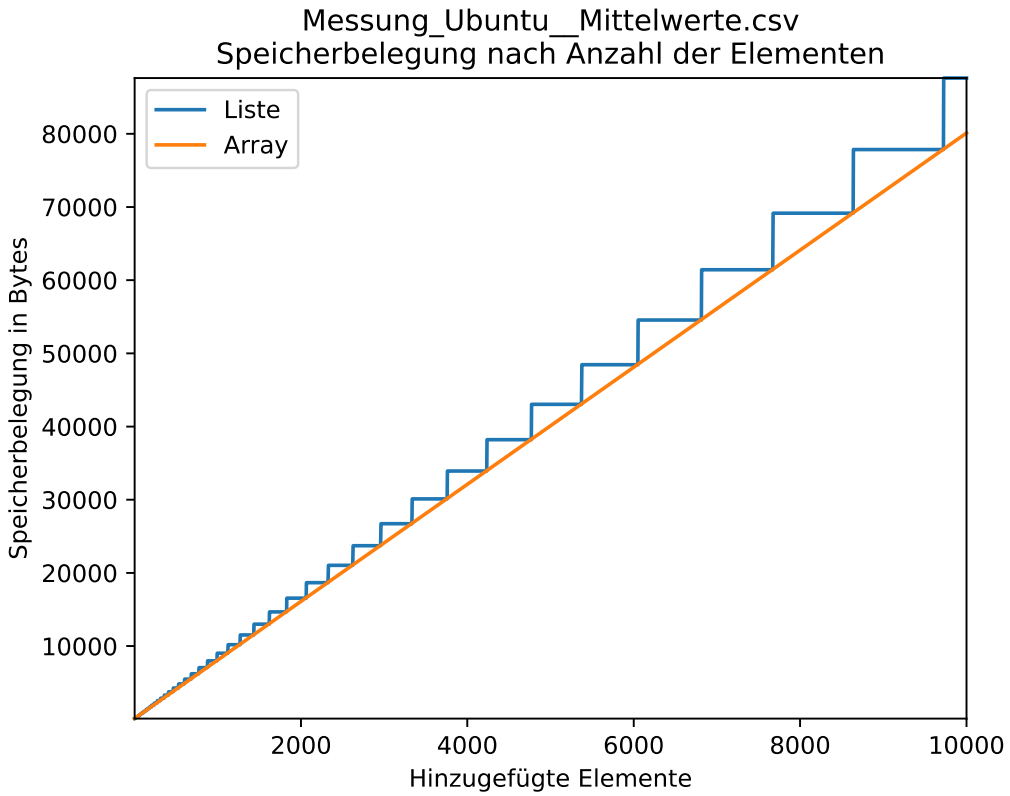

Diagramm 1: Speicherbedarf gesamt

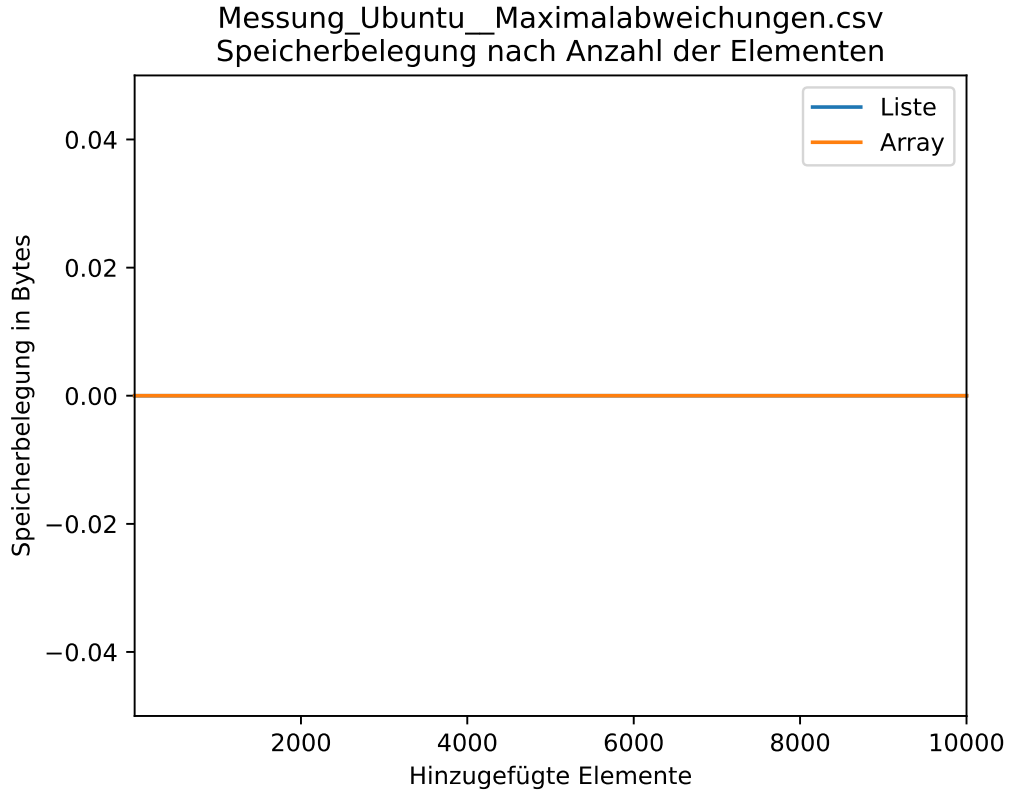

Diagramm 2: Maximalabweichungen Speicherbedarf gesamt

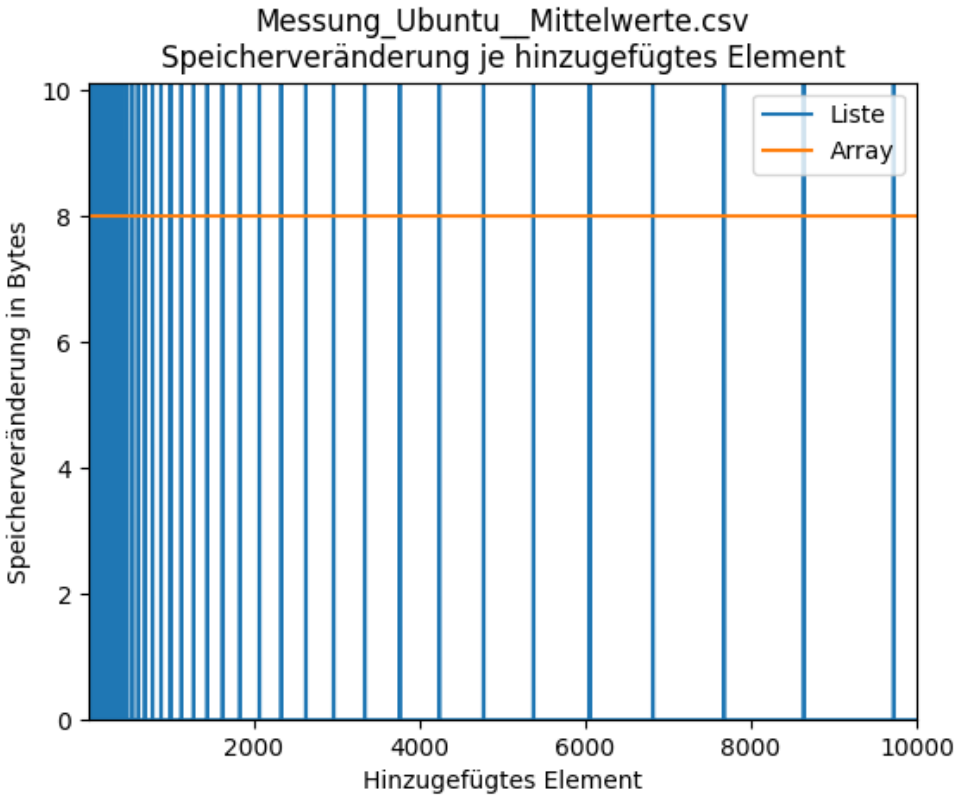

Diagramm 3: Speicherveränderung Array

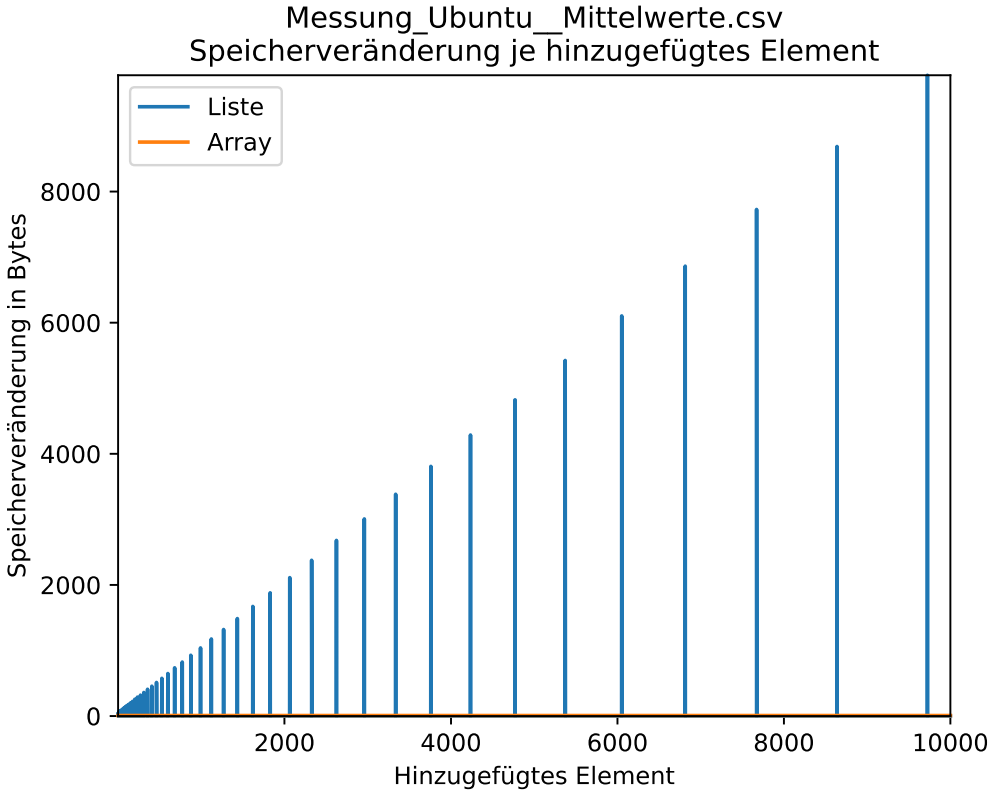

Diagramm 4: Speicherveränderung Liste

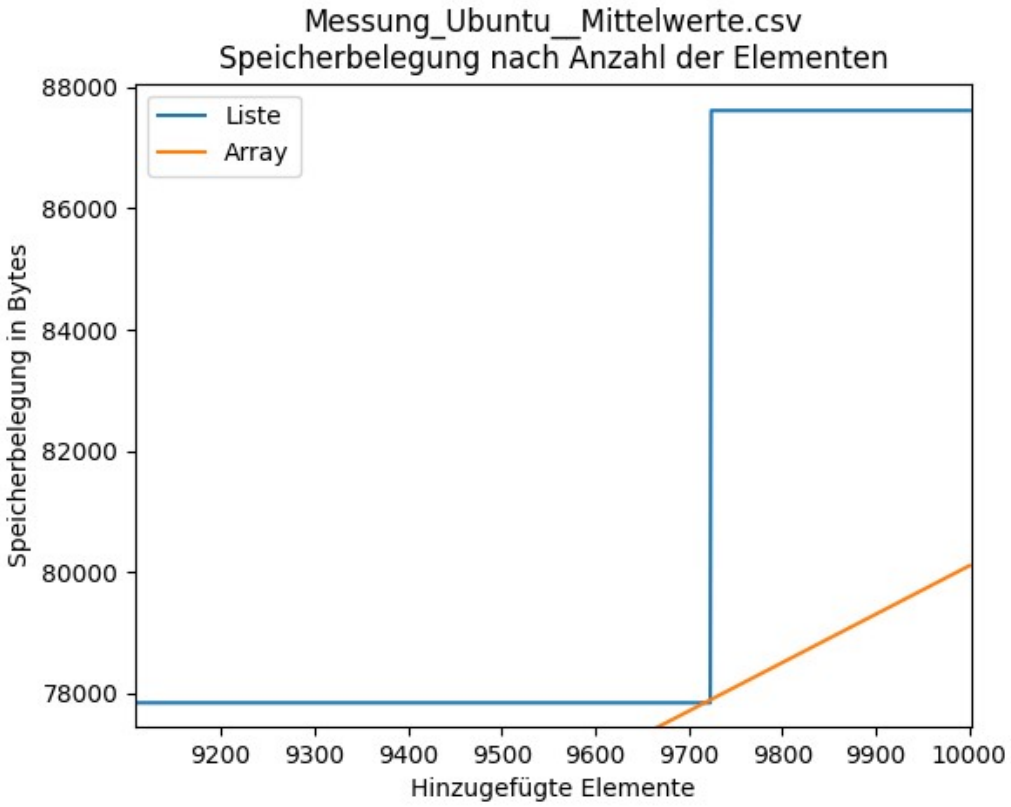

Diagramm 5: Speicherbedarf um Element 9700

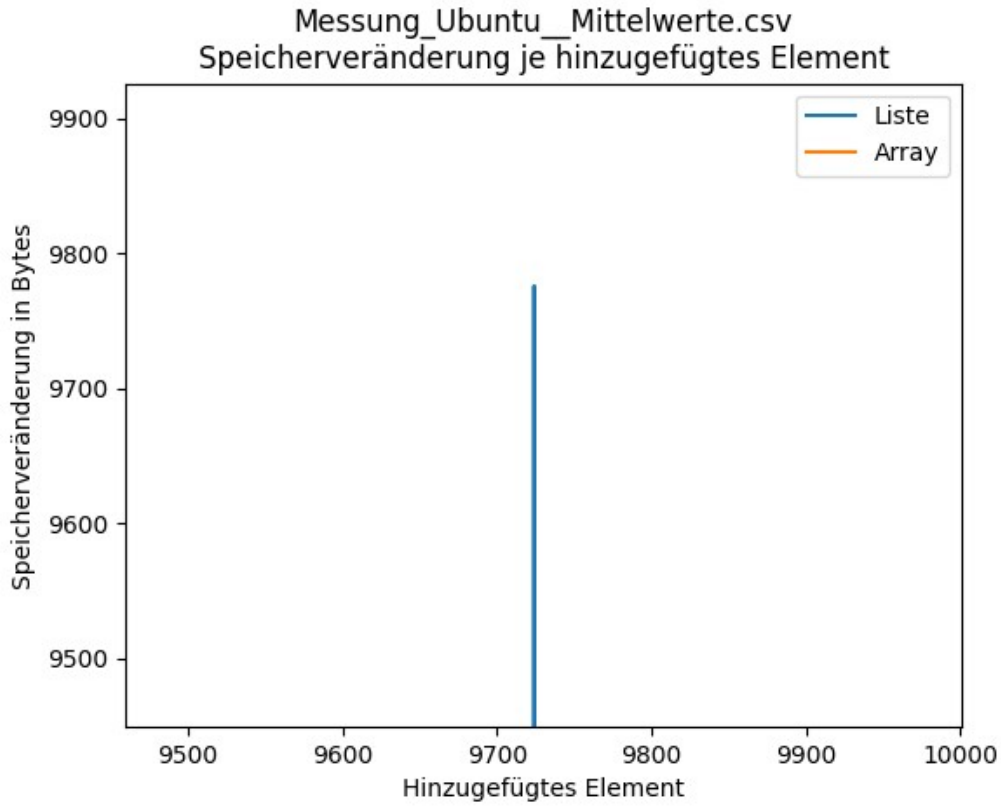

Diagramm 6: Speicherveränderung um Element 9700

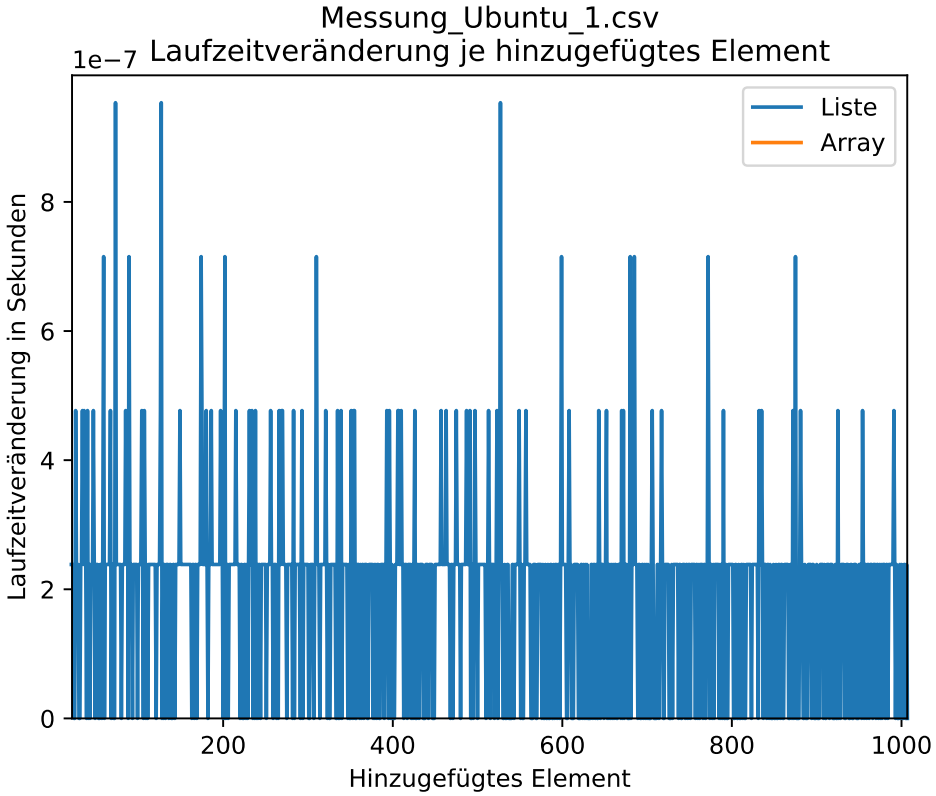

Diagramm 7: Beispielmessung Laufzeitveränderung Liste bis Element 1000

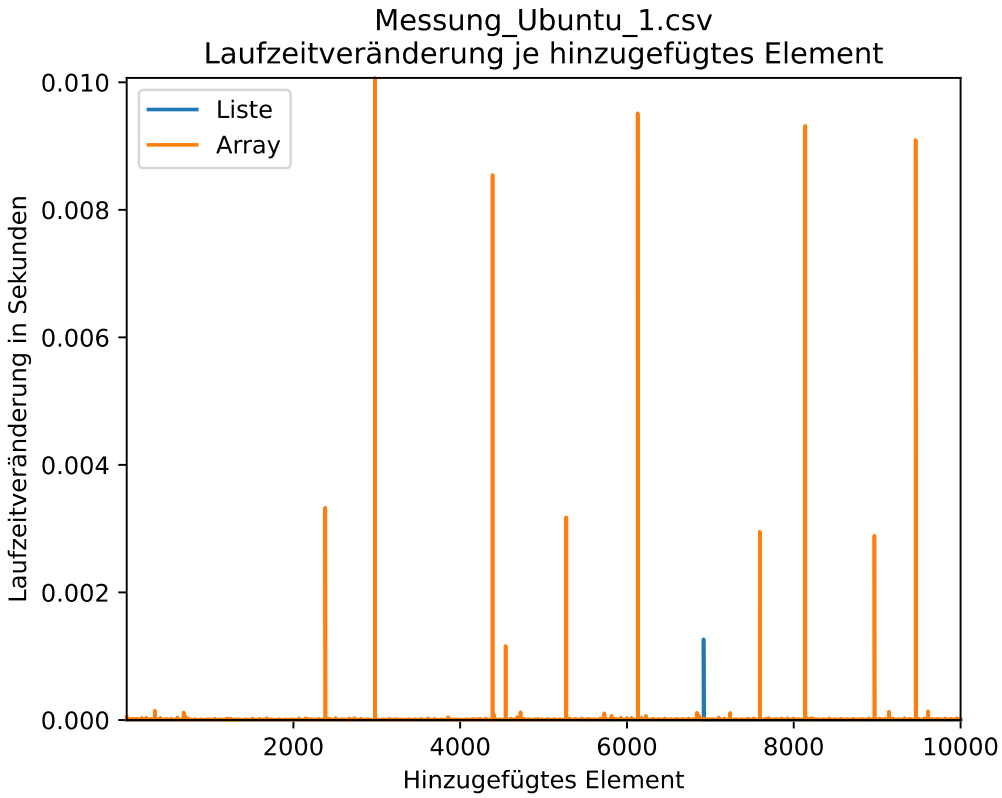

Diagramm 8: Beispielmessung Laufzeitveränderung gesamt

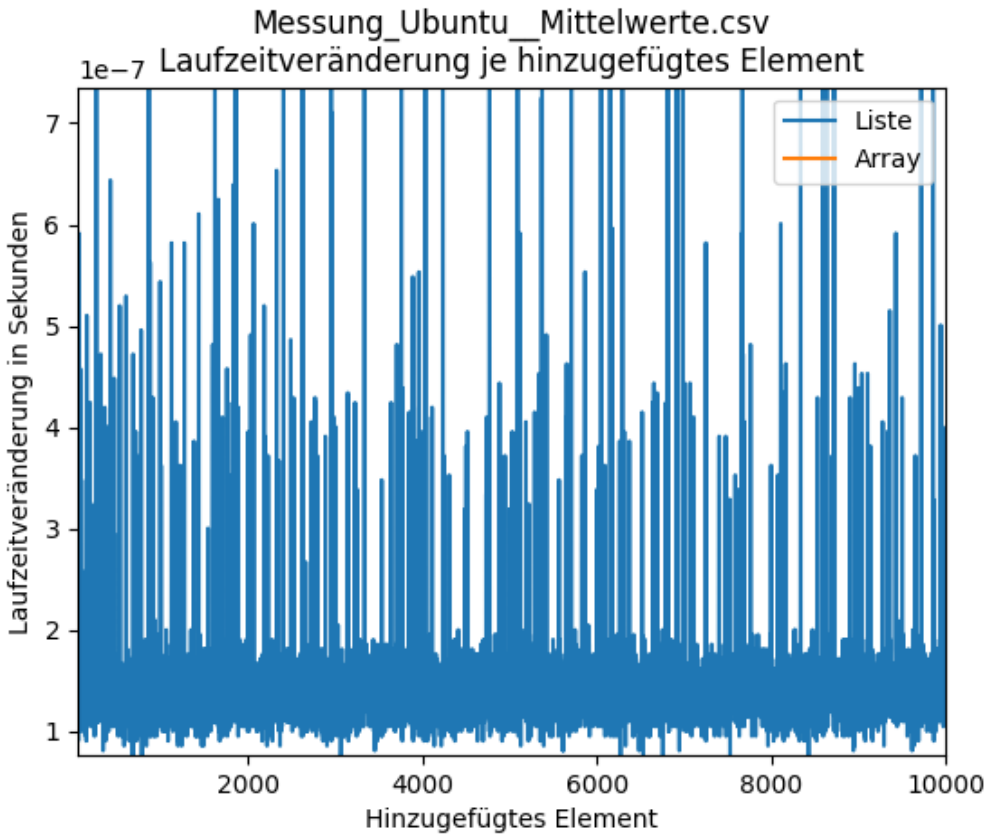

Diagramm 9: Mittelwerte Laufzeitveränderung Liste gesamt

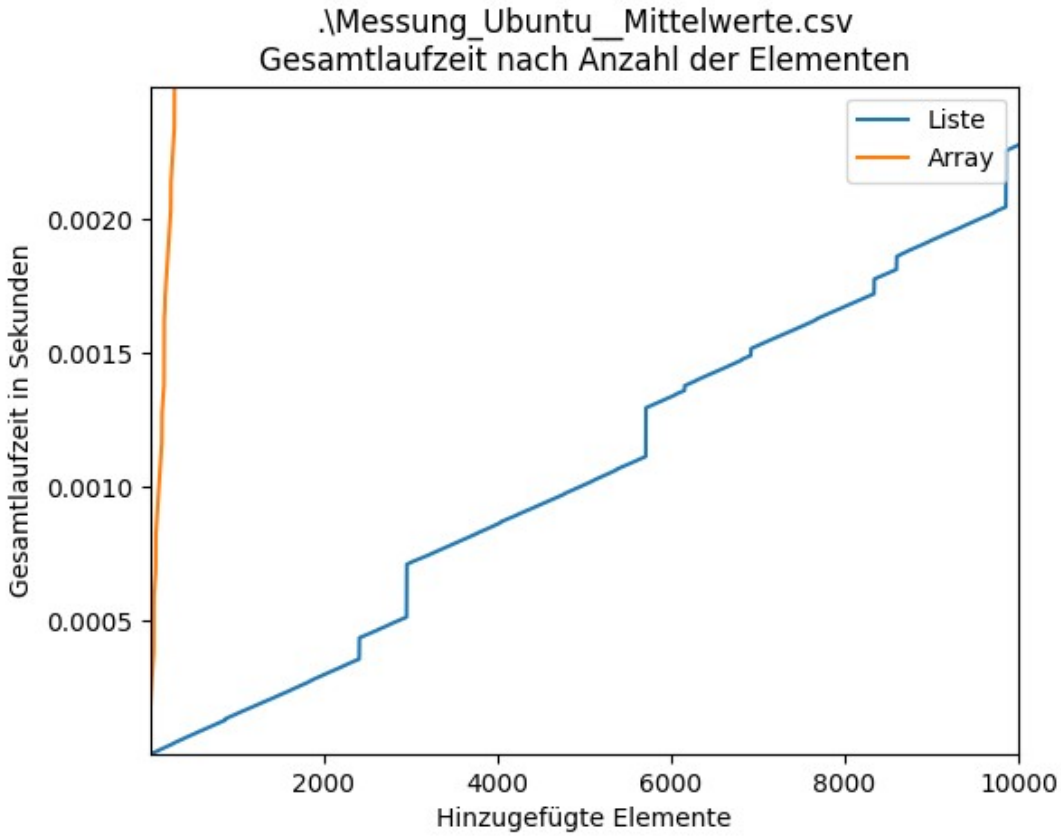

Diagramm 10: Mittelwerte Gesamtlaufzeit Liste gesamt

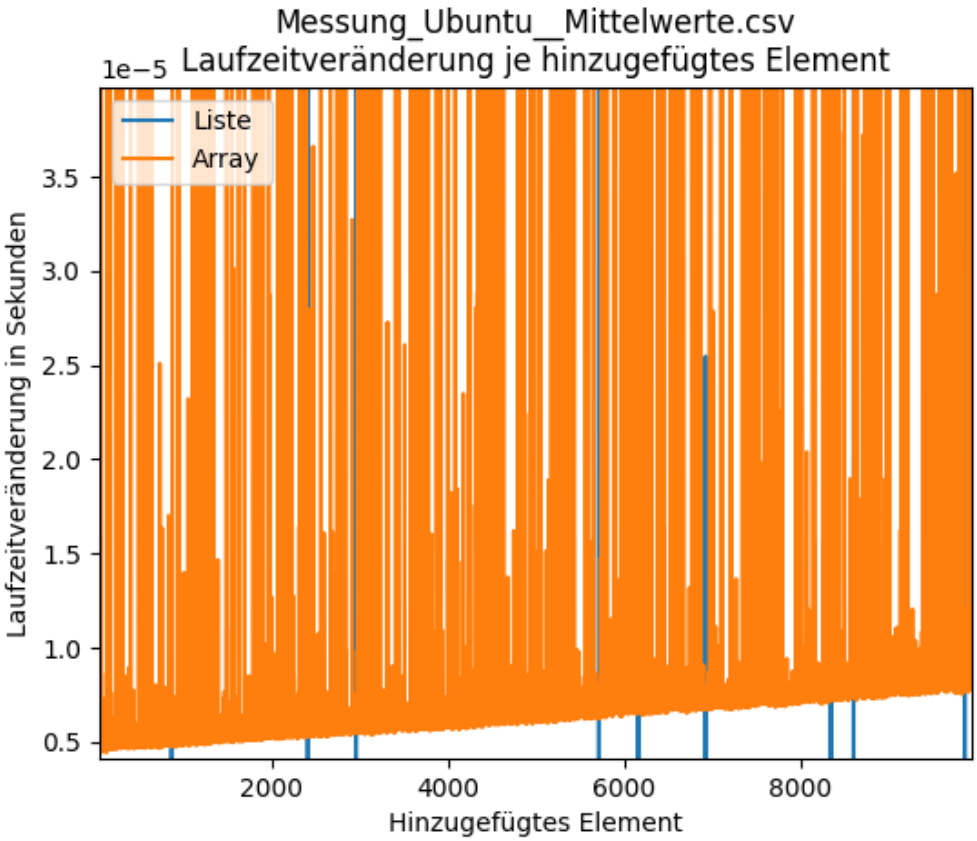

Diagramm 11: Mittelwerte Laufzeitveränderung Array Unterkante

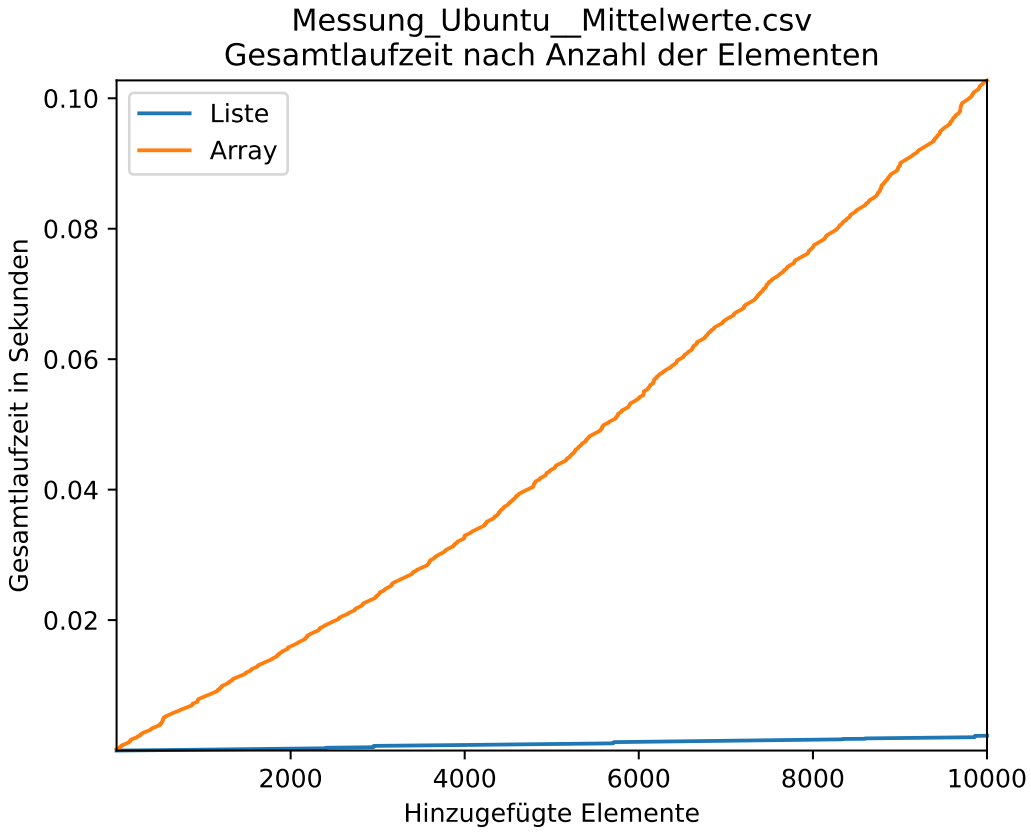

Diagramm 12: Mittelwerte Gesamtlaufzeit gesamt

Erklärung über die selbstständige Anfertigung

Ich versichere hiermit, dass ich die vorliegende Arbeit eigenständig ohne unerlaubte fremde Hilfe angefertigt habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Entlehnungen aus anderen Arbeiten sind an den betreffenden Stellen als solche kenntlich gemacht.

_____

Ort, Datum

_____

Unterschrift