1. **Aufwandsabschätzung**:
   1. **Hashtabelle**:
      1. Einfügen: Im Durchschnitt O(1), im schlechtesten Fall O(n) (bei vielen Kollisionen)
      2. Suchen: Im Durchschnitt O(1), im schlechtesten Fall O(n) (bei vielen Kollisionen)
      3. Löschen: Im Durchschnitt O(1), im schlechtesten Fall O(n) (bei vielen Kollisionen)
   2. Array:
      1. Einfügen: Am Ende O(1), am Anfang oder in der Mitte O(n)
      2. Suchen: Im Durchschnitt O(n/2), im schlechtesten Fall O(n) (alle Elemente müssen durchsucht werden)
      3. Löschen: Am Ende O(1), am Anfang oder in der Mitte O(n) (alle Elemente müssen verschoben werden)
   3. Einfach verkettete Liste:
      1. Einfügen: Am Anfang O(1), am Ende oder in der Mitte O(n)
      2. Suchen: Im Durchschnitt O(n/2), im schlechtesten Fall O(n) (alle Elemente müssen durchsucht werden)
      3. Löschen: Am Anfang O(1), am Ende oder in der Mitte O(n) (alle Elemente müssen durchsucht werden)
2. **Füllgrad Hashtabelle:**

Die Performance der Hashtabelle ist abhängig von der Qualität der Hashfunktion und der Anzahl der Kollisionen. Eine gut verteilte Hashfunktion und eine angemessen große Hashtabelle führen dazu, dass der durchschnittliche Aufwand für Einfügen, Suchen und Löschen nahe O(1) bleibt. Bei einem Füllgrad von 1000 Aktien und einer Hashtabelle mit einer Kapazität von 1009, um Kollisionen zu minimieren, liegen die benötigten Operationen für Einfügen, Suchen und Löschen bei etwa 1000. Im Gegensatz dazu wären es etwa 500.000 Operationen für Suchen und Löschen in einem Array oder einer einfach verketteten Liste und 1.000 bis 500.000 Operationen für das Einfügen, abhängig von der Position.

1. **Beschreibung der Datenstruktur und der Codeabschnitte:**
   1. Stock-Klasse (Zeile 11-24):

Definiert die Aktienobjekte mit Kürzel, Name, WKN und einer Liste von Kursdaten (kursD). Die ADD-Methode fügt neue Kursdaten hinzu und stellt sicher, dass maximal 30 Datensätze gespeichert werden (18-21) und data gibt die letzten verfügbaren Kursdaten zurück.

* 1. Custom\_hash-Funktion (27-31):

Berechnet den Hashwert für einen gegebenen Schlüssel (das Kürzel der Aktie) und die Kapazität der Hashtabelle.

* 1. HashTable-Klasse (Zeile 34-102):

Implementiert die Hashtabelle, die die Aktienobjekte speichert. Die wichtigsten Methoden sind getitem (Suchen), setitem (Einfügen) und delitem (Löschen), die alle die quadSond-Methode verwenden, um Kollisionen mithilfe der Quadratischen Sondierung zu handhaben.

* 1. quadSond-Methode (Codezeilen 47-56):

verwendet Quadratisches Sondieren, um den richtigen Index in der Hashtabelle zu finden, falls Kollisionen auftreten. Im Falle einer Kollision wird der Index schrittweise erhöht, indem das Quadrat der Schritte hinzugefügt wird, bis ein freier Platz oder der gewünschte Schlüssel gefunden wird.

* 1. Die \_\_getitem\_\_-Methode (Codezeilen 57-61):

ruft die quadSond-Methode auf, um den richtigen Index in der Hashtabelle für den gegebenen Schlüssel zu berechnen, und gibt den Wert zurück, der dem Schlüssel zugeordnet ist.

* 1. Die resize-Methode aufgerufen (Codezeilen 91-99),

wird in der \_\_setitem\_\_-Methode, wenn die Anzahl der gespeicherten Elemente einen bestimmten Schwellenwert überschreitet (z.B. wenn der Lastfaktor größer als 0,8 wird). Die resize-Methode erstellt eine neue, größere Hashtabelle und fügt alle vorhandenen Elemente in die neue Tabelle ein. Auf diese Weise wird die Hashtabelle bei Bedarf vergrößert, um die Leistung bei zunehmender Anzahl von Elementen zu erhalten.

* 1. Die \_\_setitem\_\_-Methode (Codezeilen 67-81):

verwendet ebenfalls die quadSond-Methode, um den richtigen Index in der Hashtabelle für den gegebenen Schlüssel zu berechnen und fügt das Schlüssel-Wert-Paar in die Hashtabelle ein.

* 1. Die \_\_delitem\_\_-Methode(lazy deletion) (Codezeilen 84-89):

ermöglicht es, ein Schlüssel-Wert-Paar aus der Hashtabelle zu entfernen, indem der entsprechende Schlüssel angegeben wird. Diese Methode verwendet die quadSond-Methode, um den richtigen Index in der Hashtabelle für den gegebenen Schlüssel zu berechnen. Wenn das Schlüssel-Wert-Paar gefunden wird, wird es durch (None, None) ersetzt, und die Größe der Hashtabelle wird reduziert. Dieser Ansatz wird als "Lazy Deletion" bezeichnet, da der entsprechende Bucket-Eintrag auf einen speziellen Wert gesetzt wird, der als gelöscht betrachtet wird. Wenn der Schlüssel nicht gefunden wird, wird ein KeyError ausgelöst.

* 1. Bucket-Funktion: (Codezeilen 63-65):

Die Funktion nimmt zwei Argumente: HashTable, eine Instanz der HashTable-Klasse, und key, den Schlüssel, für den der Bucket zurückgegeben werden soll. Die Funktion ruft die quadSond-Methode der HashTable-Instanz auf, um den richtigen Index in der Hashtabelle für den gegebenen Schlüssel zu berechnen. Dieser Index wird als index gespeichert. Die Funktion überprüft, ob es einen Bucket (in diesem Fall eine Liste) am berechneten Index (HashTable.buckets[index]) gibt. Wenn es einen Bucket gibt, wird dieser als Einzelelement-Liste zurückgegeben. Wenn es keinen Bucket gibt (d.h., der Wert ist None), gibt die Funktion eine leere Liste zurück.

**Fazit:**

Insgesamt kann gesagt werden, dass die Leistung der Hashtabelle bei der Verwaltung von Aktien- und Kursdaten stark von der anfänglichen Größe und dem kontinuierlichen Überwachen des Lastfaktors abhängt. Eine geringe Lastfaktorschwelle von 0,8 bei bis zu 1000 Aktien ist empfehlenswert, um eine Überlastung der Hashtabelle zu vermeiden und die Leistung zu erhalten. Das Rehashing der Hashtabelle wird bei zunehmender Anzahl der Elemente zu einem wichtigen Schritt, um eine gute Leistung aufrechtzuerhalten.

Die Wahl der Kollisionsbehandlungsmethode hat erheblichen Einfluss auf die Leistung der Hashtabelle. Im konkreten Fall wurde das quadratische Sondieren gewählt, da es im Allgemeinen bessere Ergebnisse liefert als andere Methoden wie die lineare Suche. Andererseits führt die Verwendung einer einfach verketteten Liste zu einer Verschlechterung der Leistung der Hashtabelle, da jeder Zugriff auf ein Element in der Liste einen linearen Durchlauf erfordert, was die Leistung beeinträchtigt.

Insgesamt bietet die Hashtabelle im Vergleich zu Arrays und einfach verketteten Listen eine bessere Leistung beim Einfügen, Suchen und Löschen von Aktien- und Kursdaten.

Quellen:

<https://www.w3schools.com/python/>

<https://www.udemy.com/course/python-bootcamp/>

<https://www.youtube.com/watch?v=wWgIAphfn2U&list=PLqM7alHXFySGwXaessYMemAnITqlZdZVE>

<https://www.youtube.com/watch?v=APAbRkrqDVI>

<https://www.youtube.com/watch?v=KyUTuwz_b7Q>

<https://www.youtube.com/watch?v=shs0KM3wKv8>

<https://www.youtube.com/watch?v=hESTgiuQfoo>

Custom\_hash-Funktion:

"Data Structures and Algorithms in Python" von Michael T. Goodrich, Roberto Tamassia und Michael H. Goldwasser

Quadratisches sondieren:

Data Structures and Algorithms in Python" von Michael T. Goodrich, Roberto Tamassia und Michael H. Goldwasser (Wiley, 2013), insbesondere Kapitel 10: Maps, Hash Tables und Skip Lists.

"Introduction to Algorithms" von Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein

Lazy Deletion(\_\_delitem\_ methode):

"Data Structures and Algorithms in Python" von Michael T. Goodrich, Roberto Tamassia und Michael H. Goldwasser (Wiley, 2013), insbesondere Kapitel 10: Maps, Hash Tables und Skip Lists.

"Data Structures: A Pseudocode Approach with C++" von Richard F. Gilberg und Behrouz A. Forouzan

Hashtabellen:

"Introduction to Algorithms" von Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein

"Data Structures and Algorithms in Python" von Michael T. Goodrich, Roberto Tamassia und Michael H. Goldwasser