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 1000 Aktien:**

Die Performance der Hashtabelle hängt von der Qualität der Hashfunktion und der Anzahl der Kollisionen ab. Wenn die Hashfunktion gut verteilt und die Hashtabelle angemessen groß ist, bleibt der durchschnittliche Aufwand für Einfügen, Suchen und Löschen nahe O(1). Bei 1000 Aktien und einer durchschnittlichen Performance der Hashtabelle würden die benötigten Operationen für Einfügen, Suchen und Löschen nahezu 1000 betragen. Im Vergleich 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 47-55):

Die quadSond-Methode verwendet Quadratisches Sondieren, um den richtigen Index in der Hashtabelle zu finden, falls Kollisionen auftreten.

* 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 (Codezeilen 83-89):

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. 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**

Zusammenfassend lässt sich sagen, dass die Leistung der Hashtabelle bei der Verwaltung von Aktien und Kursdaten maßgeblich von der anfänglichen Größe und dem kontinuierlichen Überwachen des Lastfaktors abhängt. Eine geringe Lastfaktor-Schwelle von beispielsweise 0,8 bei bis zu 1000 Aktien ist empfehlenswert. Rehashing der Hashtabelle ist ein wichtiger Schritt, um die Leistung bei einer wachsenden Anzahl von Elementen aufrechtzuerhalten. Auch die Wahl der Kollisionsbehandlungsmethode hat Auswirkungen auf die Leistung der Hashtabelle. Quadratisches Sondieren wurde im vorliegenden Fall gewählt. Insgesamt bietet die Hashtabelle im Vergleich zu Arrays und einfach verketteten Listen eine bessere Leistung bei Einfügen, Suchen und Löschen von Datensätzen, insbesondere bei einer großen Anzahl von Aktien und Kursdaten.