

1 Hashset

1.1 Einleitung

HashSet gehört zum java.util Paket. Es implementiert das Set-Interface und speichert seine Elemente in einer Hashtable.

1.2 Methodenbeschreibung

- `add(E e)` : - Fügt ein spezifisches Element dem Set hinzu, wenn es nicht schon vorhanden ist.
- `clear()` : - Entfernt alle Elemente von diesem Set.
- `clone()` : - Gibt eine oberflächliche Kopie des Sets zurück: Enthaltene Elemente werden nicht geklont.
- `contains(Object o)` : - Gibt true zurück wenn das Set das spezifische Element beinhaltet.
- `isEmpty()` : - Gibt true zurück wenn das Set keine Elemente enthält.
- `iterator()` : - Gibt einen Iterator über die Elemente im Set zurück.
- `remove(Object o)` : - Entfernt ein spezifisches Element vom Set, wenn es vorhanden ist.
- `size()` : - Gibt die Anzahl an Elementen dieses Sets zurück (Kardinalität).

1.3 Effizienzanalyse

Die Methode **extractMin** hat einen linearen Aufwand für alle drei gegebenen Graphen : $\theta(n)$

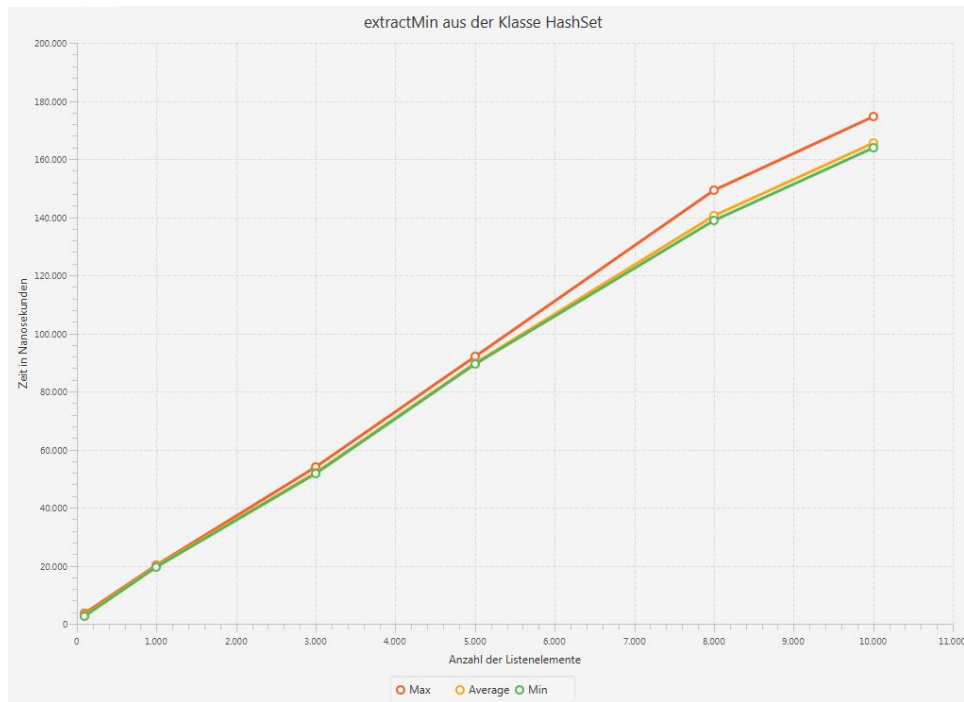


Abbildung 1: ExtractMin

Auch die Methode **getMin** besitzt einen linearen Aufwand wie alle drei Graphen zeigen : $\theta(n)$

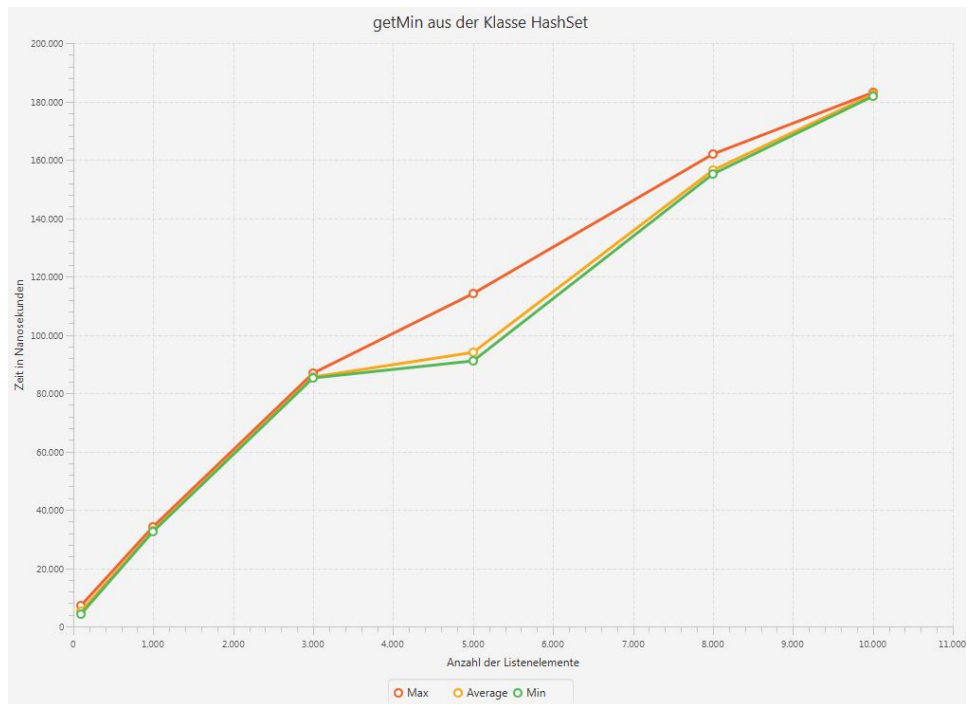


Abbildung 2: getMin

Wir können sehen, dass die Werte bei allen Graphen fast konstant bleiben. Deswegen hat die Methode **Insert** aus der Klasse Hashset einen Konstanten Aufwand : $\theta(1)$

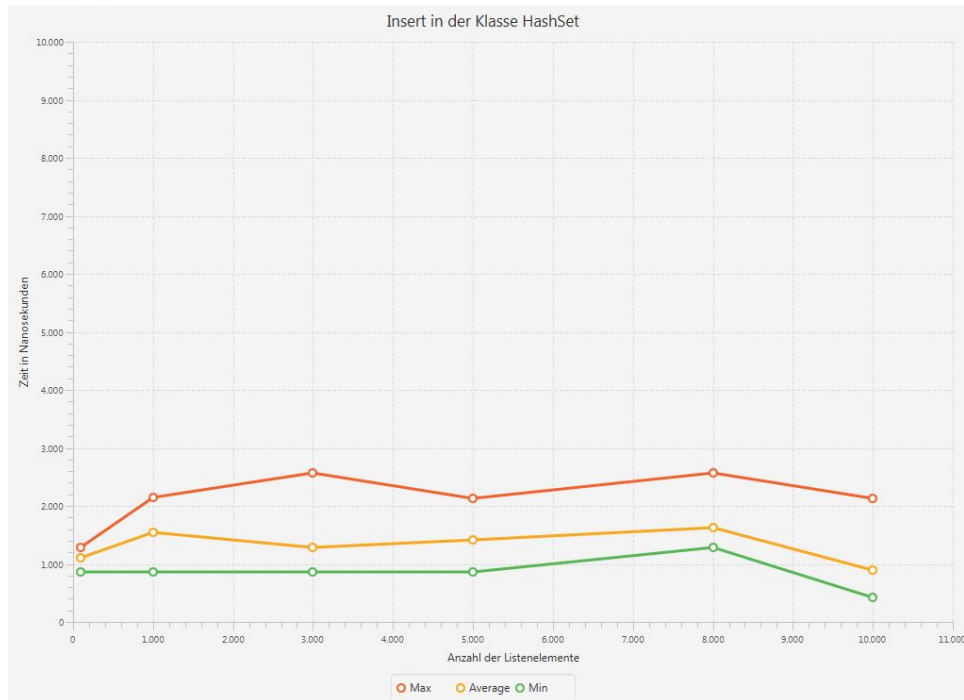


Abbildung 3: Insert

Die Methode **Merge** aus der Klasse Hashset hat einen Konstanten Aufwand. Bei einer Anzahl von 8000 Elementen weisen die Graphen eine Unregelmäßigkeit auf : $\theta(1)$

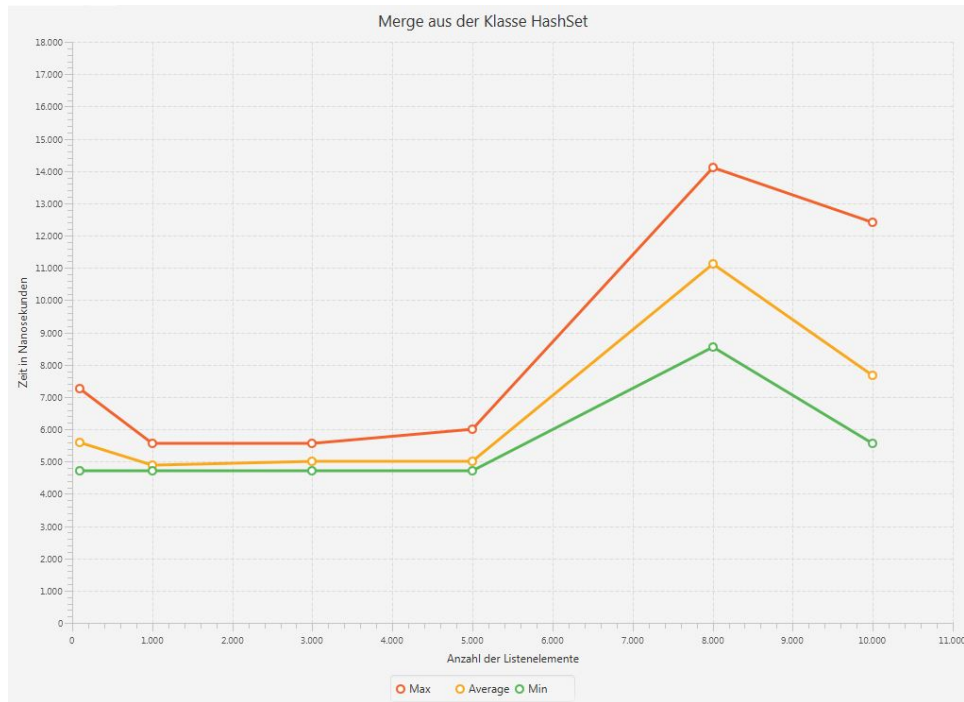


Abbildung 4: Merge

Die Methode **Remove** aus der Klasse HashSet hat einen Konstanten Aufwand, welcher durch die Graphen ersichtlich wirdt : $\theta(1)$

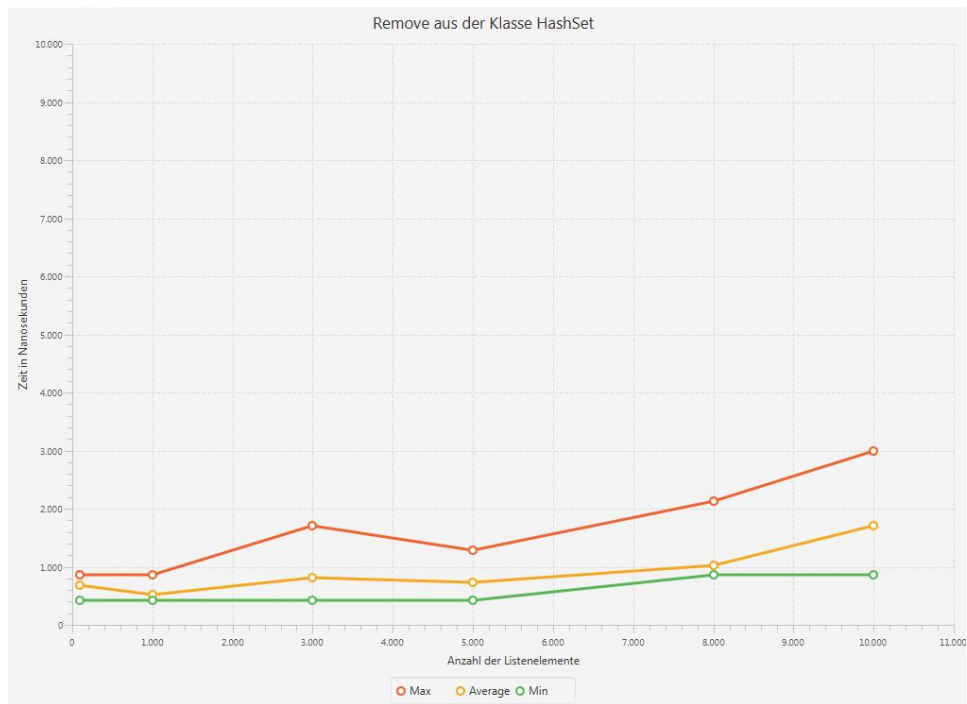


Abbildung 5: Remove

1.4 Schlussfolgerung

Alles in Allem bietet das HashSet in allen untersuchten Methoden einen sehr guten (konstanten) bis guten (linearen) Aufwand, weshalb sie sich für solch ein Einsatzgebiet was sich auf diese Funktionalitäten beschränkt zu empfehlen ist.