

1 Stack

1.1 Einleitung

Ein Stack ist ein Stapelspeicher bzw. Kellerspeicher. Das Besondere am Stack ist, dass er für seine Funktionalität nur wenige Methoden braucht.

Um Elemente in den Stack zu legen wird die Methode *push()* benötigt. Dieses Element wird an vorderster bzw. oberster Stelle in die vorhandenen Daten eingebunden. Der Stack arbeitet nach dem LIFO-Verfahren. Die Abkürzung steht für „Last In First Out“ und das bedeutet, dass das zuletzt hinzugefügte Element auch als erstes wieder aus dem Stack herausgenommen werden muss.

Um dieses neueste Element aus dem Stack zu entfernen, wird die Methode *pop()* verwendet.

Außerdem wird noch mindestens eine Methode *size()* benötigt, die die Länge des bzw. die Anzahl der Elemente im Stack zurückgibt.

Sinnvoll ist vielleicht noch eine Methode *peek()*, die das oberste Element des Stacks zurückgibt, ohne dieses aus dem Stack zu entfernen.

In Java ist der Stack eine Klasse, die direkt von der Superklasse Vector erbt. Vector lässt sich über AbstractList und AbstractCollection auf die Klasse Object zurückführen. Damit stehen Stack auch sämtliche Methoden dieser Superklassen zur Verfügung.

1.2 Methodenbeschreibung

Im Folgenden werden die von Java für den Stack zur Verfügung gestellten Methoden beschrieben, die wichtig sind und in der Implementation verwendet wurden.

- *empty()*
 - prüft, ob der Stack Elemente enthält und liefert einen boolean-Wert zurück
- *peek()*
 - liefert das zuletzt zum Stack hinzugefügte Element zurück, ohne es aus dem Stack zu löschen
- *pop()*
 - gibt das zuletzt zum Stack hinzugefügte Element zurück und entfernt es aus dem Stack
- *push(E item)*
 - fügt das als Parameter übergebene Element dem Stack hinzu

- `size()`
 - gibt die Anzahl, der im Stack enthaltenen Elemente zurück

1.3 Effizienzanalyse

Bei der Methode „ExtractMin“, zeigt der Punkt bei 8000 Listenelementen eine geringere Zeit die benötigt wurde. Die restlichen Punkte dieses Graphen liegen auf einer Linie. Der Graph mit den Durchschnittswerten steigt konstant aber recht schnell an. Der rote Graph sieht dem grünen Graph sehr ähnlich, da auch er einen Ausreißerwert bei 8000 besitzt und die restlichen Punkte linear sind. Somit kann man hier feststellen, dass diese Methode einen linearen Aufwand hat.

Linearer Aufwand $\mathcal{O}(n)$

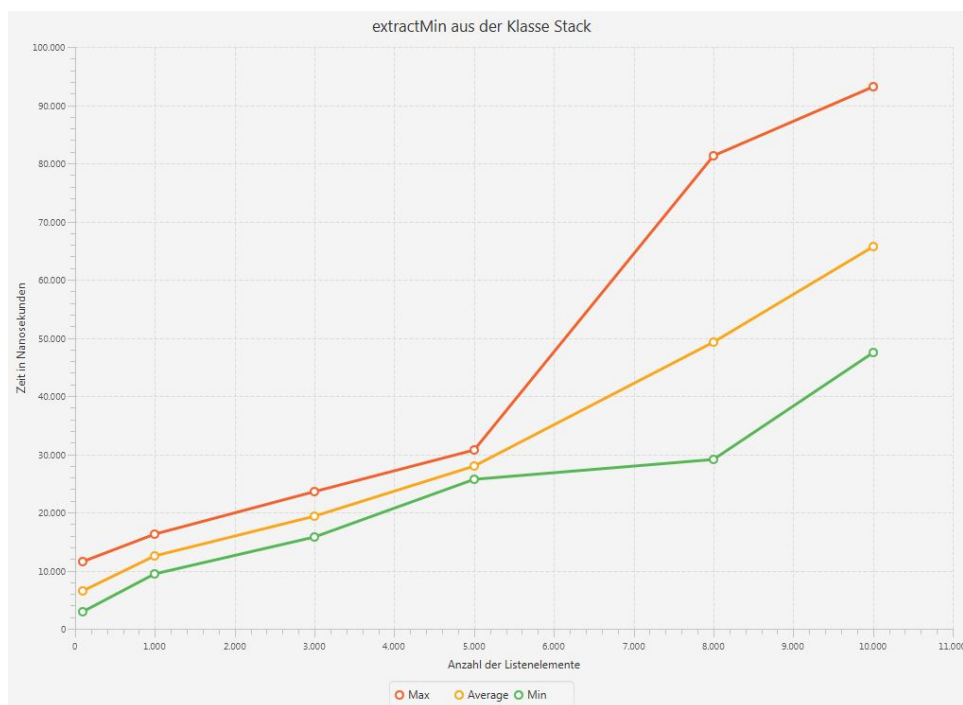


Abbildung 1: ExtractMin

Bei der Methode „GetMin“ sieht der untere Graph sehr linear aus. Auch der Durchschnittsgraph sieht linear aus, hat aber eine Tendenz zu einer logarithmischen Funktion. Der rote Graph ist logarithmisch, da der Graph hier bis 3000 Listenelemente ansteigt, danach abflacht und fast konstant ist. Zieht man hier eher die beiden unteren Graphen in Betracht lässt sich auch hier ein linearer Aufwand feststellen.
 Linearer Aufwand $\mathcal{O}(n)$

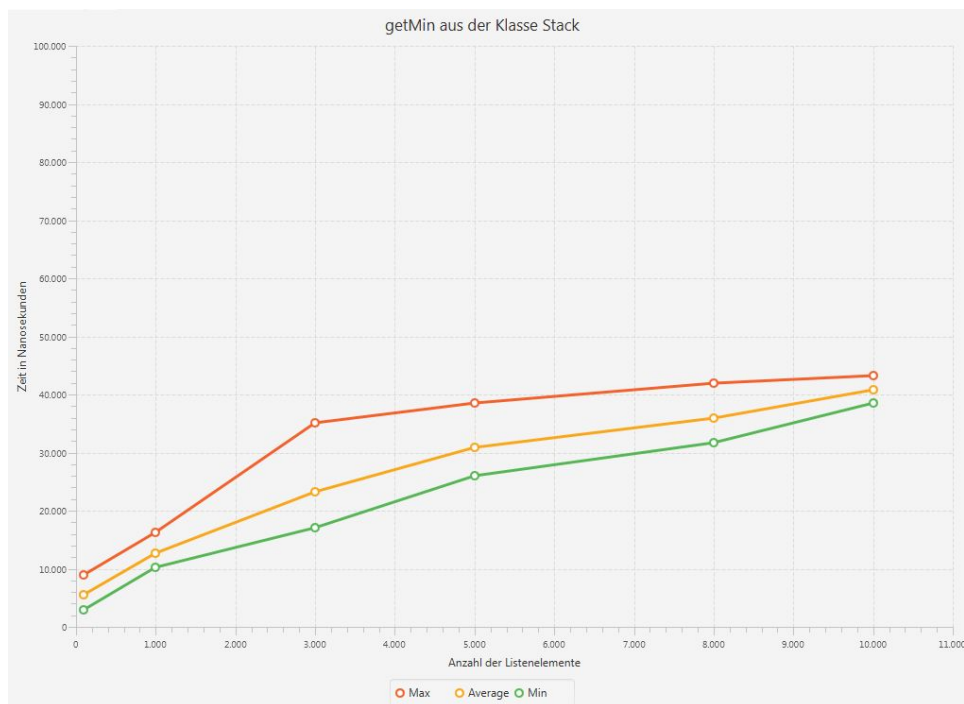


Abbildung 2: GetMin

Bei der Methode „insert“ ist der Durchschnittsgraph recht konstant gehalten. Beim grünen Graphen sieht man sehr deutlich, dass alle Werte exakt auf einer Linie sind und den gleichen Wert besitzen. Der Graph für die Maximalwerte steigt bis 5000 Listenelemente an und sinkt danach wieder, bis er fast den Startwert an Nanosekunden erreicht. Schlussfolgernd aus allen drei Graphen kann man sagen das diese Methode einen konstanten Aufwand besitzt.

Konstanter Aufwand $\mathcal{O}(1)$

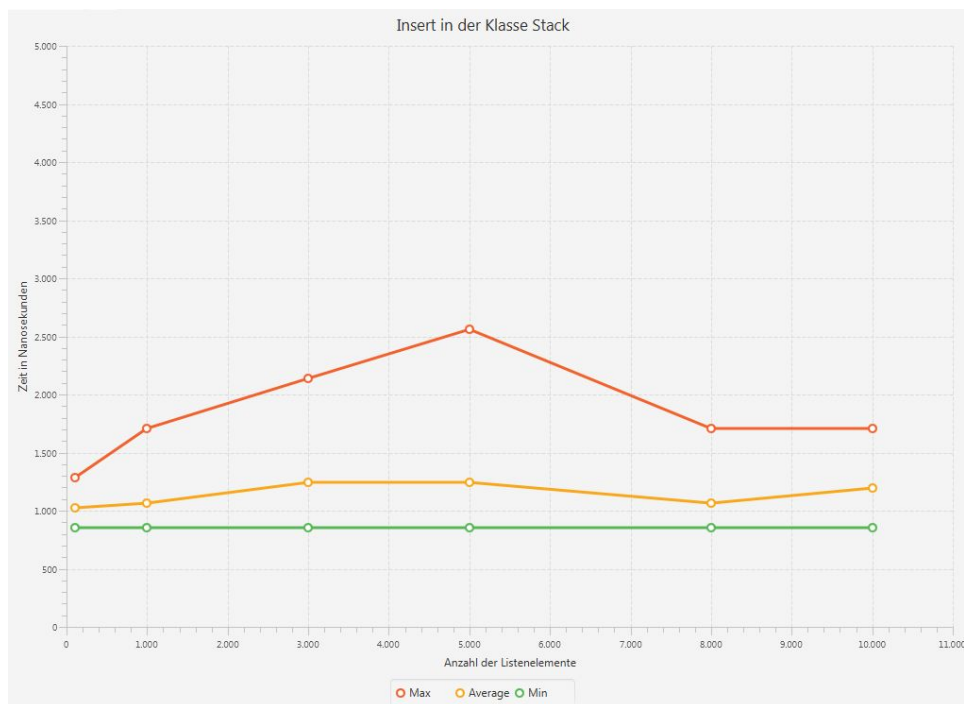


Abbildung 3: Insert

Die Methode „merge“ besitzt einen sehr konstanten Graphen für die Minimalwerte und die Durchschnittswerte, allerdings steigt hier der letzte Wert deutlich nach oben. Bei dem roten Graphen liegt der Minimalwert bei 1000 Listenelementen, danach steigt der Graph an, sinkt aber bei 8000 Listenelementen nochmal ab, um erneut anzusteigen. Konstanter Aufwand $\mathcal{O}(1)$

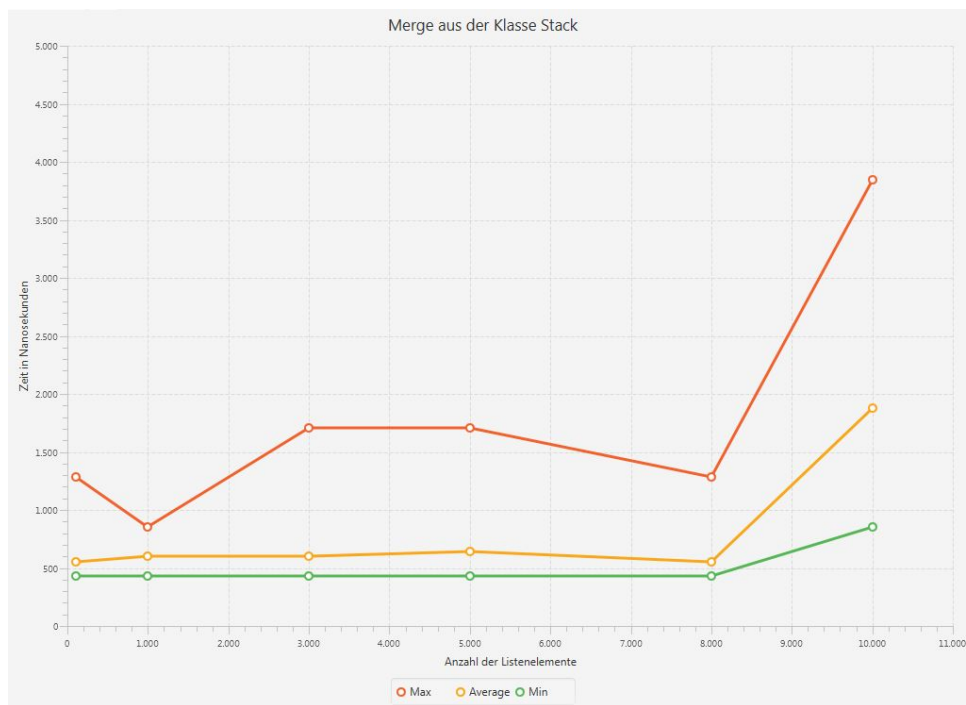


Abbildung 4: Merge

Bei der Methode „remove“ hat der Graph für die Minimalwerte einen logarithmischen Anklang. Auch die Durchschnittskurve ist logarithmisch, da sie bis 1000 Listenelemente ansteigt und dann stark abflacht. Der rote Graph weist noch keine logarithmische Funktion auf, da er recht konstant ansteigt. Betrachtet man mehr die 2 unteren Graphen, kann man erkennen, dass es sich um einen logarithmischen Aufwand handelt
 Logarithmischer Aufwand $\mathcal{O}(\log n)$

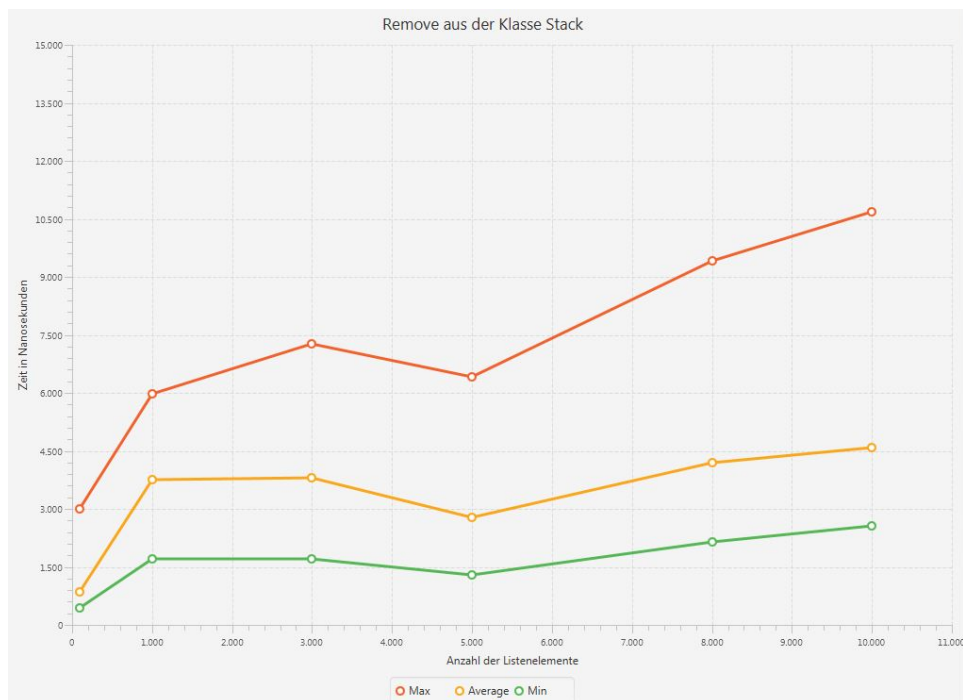


Abbildung 5: Remove

1.4 Schlussfolgerung

Insgesamt ist der Stack von ihrem Aufwand her sehr gut, da dieser meist linear ist. Die Methoden „insert“ und „merge“ sind besonders zu empfehlen, da ihr Aufwand konstant ist. Der logarithmische Aufwand der „remove“ Methode ist für wenige Listenelemente nicht so gut, sobald es aber um große Mengen geht kann man auch diese gut verwenden.

Die einzige Methode die gar nicht für den Stack geeignet ist, ist „decrease-Key“, da es nicht möglich war diese zu Implementieren.