# Quicksort Algorithmus

Meine Aufgabe war es den Quicksort Algorithmus zu programmieren. Um die Funktionsweiße des Algorithmus zu verstehen habe ich einen sog. Pseudocode und ein Gif bekommen.

## Gliederung

* Funktionsweiße des Algorithmus
* Klassen Diagramm
* Vorstellung des Codes
* Rekursion
  + Unterschied Rekursiv / Iterativ
  + Vorteile der Rekursion im Vergleich zur Iteration
  + Nachteile der Rekursion

## Funktionsweiße des Algorithmus

Das besondere am Quicksort Algorithmus ist, dass die Elemente in der Liste nur vertauscht werden und nicht extern zwischengespeichert werden müssen. Solche Sortierverfahren nennt man auch In-Place-Algorithmus.

Hier wird die komplette Liste in immer kleiner werdenden Teillisten aufgeteilt und in sich sortiert. Sobald die Teillisten sortiert wurden ist auch die komplette Liste sortiert.

* Zunächst wird der Durchschnitt aller Elemente ermittelt.
* Im nächsten Schritt werden von links Werte gesucht, die größer als der Durchschnitt sind und von rechts Werte gesucht, die kleiner als der Durchschnitt ist.
* Wurde ein Wertepaar gefunden, werden die Positionen vertauscht.
* Dieser Schritt wird nun solange wiederholt, bis sich die linke und die rechte Suche treffen. Hier ist die Grenze der beiden Teillisten.
* In dem nächsten Schritt wird das Element, das am weitesten rechts in der Teilliste ist, mit dem Wert an der neuen Grenze vertauscht.
* Nun werden die neu entstandenen Teillisten im gleichen Verfahren sortiert.
* Das Verfahren wiederholt sich solange bis die Teillisten nur noch aus einem oder keinem Element bestehen.
* Ist dieser Punkt erreicht ist auch die komplette Liste fertig sortiert.

## Klassen Diagramm

* Main Programm erstell das Objekt myList von der Klasse Sorting diese Klasse ist dafür da um den Algorithmus auszuführen.
* List hat einen Konstruktor List und die Methoden fillList, getList und getPivot. Die Klasse ist erzeugt die Liste und befüllt diese mit Werten.
* Sorting hat die beiden Methoden sort und teile
* Sorting erstellt ein Objekt daten aus der Klasse List in Sorting wird der Algorithmus ausgeführt.

## Vorstellung des Codes

* Die Klasse mainProgram ist dafür da, um den Suchalgorithmus auszuführen.
* Es wird zunächst ein neues Objekt von Sorting namens myList erstellt.
* Mit myList wird dann die rekursive Methode sort ausgeführt, gleichzeitig werden die Übergabeparameter gesetzt, die die Grenzen der Liste angeben.
* In der zweiten Klasse List geht es hauptsächlich darum die Liste zu erzeugen und diese danach mit Werten zu befüllen.
* Die Klasse List beginnt mit dem Konstruktor, der in diesem Fall die Methode fillList ausführt, sobald die Klasse List aufgerufen wird.
* Darunter wird die Variable list als Typ ArrayList deklariert.
* Die Methode fillList, die im Konstruktor aufgerufen wird erzeugt zunächst das Objekt list und befüllt diese im Folgenden Schritt mit Werten.
* Nach dem befüllen wird die unsortierte Liste in der Konsole ausgegeben
* Die Methode getList ist dafür da, um die Liste an andere Klassen zu übergeben.
* Die Methode getPivot berechnet den Durchschnitt der jeweiligen Teilliste indem er mit einer Schleife die Liste durchgeht und die Werte addiert. Danach wird die Summe durch die Anzahl der Elemente geteilt.
* In der Klasse Sorting steckt der Algorithmus.
* Wie schon in der Main Methode gesehen benötigt die Methode sort zwei Werte, die Grenze Links und die Grenze Rechts.
* Solange die Werte von der Linken Grenze kleiner als die der Rechten Grenze sind wird der Algorithmus ausgeführt.
* Den Wert für teiler erhält das Programm indem es die Methode teile mit den Werten der Grenze von links und rechts ausführt. Dieser Wert wird benötigt um die Grenzen der Teillisten zu setzen.
* Jetzt wird die Methode sort zweimal in sich selber Aufgerufen diesmal aber mit anderen Grenzen. Hier werden also die Bereiche der Teillisten gesetzt.
* In der Methode teile, die von sort aufgerufen wird, werden zunächst die Parameter der Grenzen lokal zwischengespeichert.
* Außerdem wird der Wert der am weitesten Rechts ist als Pivot Element deklariert.
* Nun Folgt die Schleife, in der das eigentliche sortieren der Elemente stattfindet.
* Das Ziel dieser while-Schleife ist es, alle Elemente die kleiner als der Durschnitt sind, auf die linke Seite zu verschieben und alle Elemente die größer als der Durschnitt sind auf die rechte Seite zu verschieben.
* Die erste for-Schleife fängt von links an und geht die Elemente solange durch, bis es die rechte Grenze erreicht hat und die Werte kleiner als der Durchschnitt sind. Wird aber ein Element in dem Bereich gefunden der größer als der Durchschnitt ist wird die Position in dem Wert i gespeichert und die for-Schleife bricht ab.
* In der zweiten for-Schleife spielt sich das gleiche ab, hier beginnt die Suche aber von rechts und die Schleife bricht ab, wenn ein Wert der kleiner als der Durschnitt ist gefunden wurde. Diese Position wird in j gespeichert.
* Jetzt prüft das Programm ob i kleiner als j ist. Ist das der Fall werden die Elemente vertauscht.
* Nachdem zwei Elemente vertauscht wurden prüft die while-Schleife erneut, ob i immer noch kleiner als j ist. Ist das der Fall geht die Suche an der Stelle der vertauschten Elemente weiter, bis sich die beiden Suchdurchläufe Treffen, also bis i nichtmehr kleiner als j ist.
* Wurde die Teilliste also erfolgreich durchsucht, wird das Pivot Element mit der Stelle vertauscht, an der die Suchdurchläufe gestoppt sind.
* Jetzt wird die Teilweiße sortierte Liste in der Konsole ausgegeben und der Wert i wird an die Variable teiler übergeben.
* Jetzt können die nächsten sort Durchläufe mit den neuen Grenzen durchgeführt werden.
* Sort wird also solange ausgeführt bis es nur noch Teillisten mit der länge von einem Element gibt, den erst danach ist die Bedingung das die linke Grenze kleiner als die rechte Grenze ist nicht mehr erfüllt und die Methode sort bricht ab.

## Rekursion

Das besondere an der sort Methode ist, dass sie sich selber aufruft bis die Abbruch Bedingung erfüllt wurde. Und genau dieses wiederaufrufen nennt man Rekursion.

Unter Rekursion versteht man in der Programmierung ein Verfahren, bei dem sich eine Methode selbst aufruft, sodass, ähnlich einer Endlosschleife, ein potentiell unendlicher Programmablauf entsteht. Im Grunde wird das Problem in mehrere kleine Teilprobleme oder in meinem Fall Teillisten zerlegt. Diese Teilprobleme werden gelöst und anschließen werden die Teillösungen wieder zu einer Gesamtlösung vereint.

Setzt man die Abbruchbedingung korrekt, so kann durch Rekursion ein Algorithmus oft recht einfach mehrfach wiederholt werden.

Die Rekursion steht der Iteration gegenüber, viele Probleme können entweder iterativ oder aber auch rekursiv gelöst werden. Man kann dabei aber nicht sagen ob das rekursive Verfahren pauschal besser als das iterative ist. Bei komplexen Berechnungen sind Rekursionen oft leichter lesbar und liefern den kompakteren Code, während iterative Lösungen oft effizienter und leichter verständlich sind.

Ein einfaches Beispiel das man sowohl rekursiv als auch iterativ lösen kann ist die Fakultätsberechnung.