Übersichtsblatt:
RadixSort

Was ist das?
RadixSort ist ein Sortierverfahren, das unter gewissen Voraussetzungen mit einer linearen Laufzeit auskommt. Somit eignet es sich sehr gut für die Sortierung großer Datenmengen.

Die Voraussetzungen:
Damit der Algorithmus korrekt funktioniert, ist es wichtig, dass die folgenden Bedingungen erfüllt sind:

- Die zu sortierenden Daten bestehen nur aus Zeichen eines endlichen Alphabets (z.B. Zahlen)
- Es besteht eine Totalordnung zwischen den Zeichen des Alphabets.
- Die Länge der Schlüssel ist durch eine bekannte Konstante begrenzt.

Funktionsprinzip:
Der Algorithmus besteht grundsätzlich aus zwei Phasen, die für jede Stelle des Schlüssels durchlaufen werden. Dabei wird mit der Stelle mit der niedrigsten Wertigkeit begonnen.

Partitionierungsphase:
Abhängig von der gerade betrachteten Stelle des Schlüssels wird dieser in eine Kiste gepackt. Beispiel: Wir betrachten die zweite Stelle des Schlüssels 42. Der Schlüssel wird in die Kiste 4 gepackt.

Sammelphase:
Nun geht man die Kisten nacheinander durch, beginnend mit der Kiste, die die niedrigste Wertigkeit hat, und legt alle Elemente aus der jeweiligen Kiste auf einen Stapel. Dabei bleibt die Reihenfolge der Elemente erhalten. Ist eine Kiste leer, geht man zur nächsten Kiste und fügt die Elemente aus dieser ebenfalls dem Stapel hinzu.

Sind alle Kisten leer wiederholt sich der Algorithmus mit der nächsten Stelle des Schlüssels.

Codebeispiel:
Das folgende Beispiel ist eine Java Methode, die den RadixSort-Algorithmus auf ein Feld von positiven Ganzzahlen anwendet. Als Obergrenze für die Länge der Schlüssel dient hier die natürliche Begrenzung des Datentyps Integer (32 Bit).

```
public static void radixSort(int[] a) {
     int     n;                              // Fachnummer
     int[]   nPart = new int[2];             // Anzahl der Elemente in den beiden Faechern
     int[][] part  = new int[2][a.length];   // die beiden Faecher haben die Groesse des Arrays a

     // Schleife ueber alle Bits der Schluessel (bei int: 32 Bit)
     for (int i=0; i<32; i++) {
         nPart[0] = 0;
         nPart[1] = 0;

         // Partitionierungsphase: teilt "a" auf die Faecher auf
         for (int j=0; j<a.length; j++) {
             n = (a[j]>>i)&1;              // ermittelt die Fachnummer: 0 oder 1
             part[n][nPart[n]++] = a[j];   // kopiert j-tes Element ins richtige Fach
         }

         // Sammelphase: kopiert die beiden Faecher wieder nach "a" zusammen
         System.arraycopy(part[0], 0, a, 0,        nPart[0]);
         System.arraycopy(part[1], 0, a, nPart[0], nPart[1]);
     }
 }
```

Quelle: Wikipedia

Laufzeit
Wenn die oben genannten Bedingungen erfüllt sind, erreicht der Algorithmus für ein Feld der Länge n mit Schlüsseln der Länge m eine Laufzeit von $O(n * m)$.