Radixsort

Radixsort ist ein **nicht-vergleichender** Sortieralgorithmus. Er funktioniert besonders gut beim Sortieren von Ganzzahlen, darum betrachten wir hier zunächst auch nur diesen Fall.

Radixsort sortiert die Elemente einer Liste, indem er sie nach einzelnen Ziffern (Stellenwerten) gruppiert, meistens beginnend mit der niederwertigsten Ziffer. Für jede Ziffer der zu sortierenden Zahlen wird ein sogenannter "Bucket" verwendet, in den die Zahlen einsortiert werden. Nachdem die Elemente auf die "Buckets" verteilt wurden, werden sie wieder eingesammelt, was dazu führt, dass die Sortierung für die betrachtete Stelle korrekt ist. Dieser Vorgang wird für die Ziffern jeder Stelle wiederholt, bis alle Stellen abgearbeitet sind, dann ist die Liste sortiert.

In der Animation kann man das Prinzip von Radix-Sort sehen. Zunächst werden die Elemente entsprechend ihrer Einerstelle in die Buckets sortiert und anschließend wieder in das ursprüngliche Array zurück übernommen, beginnend bei Bucket 9. Jetzt befinden sich die Elemente in der korrekten Reihenfolge bezüglich der Einerstelle. Anschließend verfährt man mit der Zehnerstelle entsprechend. Wenn man darauf achtet, die Reihenfolge von Elementen mit gleichen Stellenwerten beizubehalten, sind diese entsprechend der Einerstelle vom vorigen Durchlauf an der korrekten Position. Das Vorgehen wiederholt man für die weiteren Stellen der Elemente und erhält so eine sortierte Liste.

Radixsort stellt **besondere Anforderungen an die zu sortierenden Elemente** (es muss eine Stellenweise, lexikalische Ordnung für die Elemente existieren) - die zu Beginn dieses Kapitels eingeführte Verallgemeinerung mit Hilfe der `compareTo`-Methode, beliebige Objekte "sortierbar zu machen", funktioniert für Radix-Sort nicht! Dafür hat Radixsort eine Zeitkomplexität von ca. O(n) - das lässt sich bei vergleichsbasierten Sortierverfahren nicht erreichen.

Weitere Wichtige Eigenschaften:

- Nicht-vergleichend: Radixsort vergleicht keine Elemente direkt miteinander, sondern nutzt die Ziffernpositionen.
- Bei entsprechender Implementierung ist Radixsort stabil: Die Reihenfolge gleichwertiger Elemente bleibt wie in der Ausgangsliste erhalten.
- Laufzeit: Die Zeitkomplexität beträgt im Allgemeinen $O(k \cdot (b+n))$, wobei k die maximale Schlüssellänge (Anzahl der Ziffern), b die Basis (z.B. 10 bei Dezimalzahlen) und n die Anzahl der zu sortierenden Elemente ist
- Radixsort ist besonders effizient, wenn die Anzahl der Ziffern (bzw. die Schlüssellänge) im Vergleich zur Anzahl der Elemente klein ist

---

(A1)

Überlege, welche Eigenschaften die Elemente einer Liste haben müssen, wenn man sie mit Radix-Sort sortieren möchte. Gib weitere Beispiele an, die Listen beinhalten die dich aus zahlen bestehen.

(A2)

Implementiere Radix-Sort im Programmgerüst der entsprechenden Klasse in der Bluej-Vorlage von https://codeberg.org/qg-info-unterricht/algs4-sort-bluej.

A2.1 Betrachte zunächst den Konstruktor: Notiere dir die Bedeutung von n, $digits$ und a im Kontext von Radix-Sort. Welche Aufgabe hat die for-Schleife?

Kontrolliere deine Überlegung mit Hilfe entsprechender Eingaben, indem du das erzeugte Objekt untersuchst.

A2.2 Als "Buckets" verwenden wir ein "Array aus ArrayLists":

```
// Buckets als Array aus ArrayLists erzeugen
ArrayList<Integer>[] bucket = new ArrayList[10];
for(int i=0; i<10; i++) {
     bucket[i] = new ArrayList<>();
}
```

`bucket[7]` ist die ArrayList, die alle Elemente mit der betrachteten Ziffer 7 aufnimmt. Analog für alle anderen Ziffern, die in den Zahlen vorkommen können (0-9).

A2.3 Implementiere die beiden Abschnitte "Verteilen" und "Einsammeln" so, dass die eingegebenen Zahlen anschließend nach ihrer Einer-Ziffer sortiert sind. Beachte die Anmerkungen in der Vorlage!

A2.4 Wiederhole den Verteilen/Einsammeln-Zyklus nun für jede Stelle der Zahlen, indem du eine Schleife um diese Abschnitte herum erzeugst - beachte die Kommentare im Code.

Du musst dir noch überlegen, wie du der Reihe nach Einer-, Zehner, Hunderter-Stelle ermitteln kannst, um bei jedem neuen Durchlauf in die passenden Buckets zu sortieren.

Außerdem musst du festlegen, wie oft diese äußere Schleife durchlaufen werden soll - `digits` hilft hier weiter.

Lösungsvorschlag

```
    // Radixsort sortiert die ArrayList
    public void sort(ArrayList<Integer> a) {

        // Unsortiertes Array ausgeben
        System.out.println(Arrays.toString(a.toArray()));

        // Buckets als Array aus ArrayLists erzeugen
        ArrayList<Integer>[] bucket = new ArrayList[10];
```

```
        for(int i=0; i<10; i++) {
            bucket[i] = new ArrayList<>();
        }

        for(int d=0 ; d<digits; d++) {
            int q=1;
            for(int i=0; i<d; i++) {
                q=q*10;
            }
            // Verteilen
            for(int z=0; z<a.size(); z++) {
                int ziel_bucket = (a.get(z)/q)%10;
                bucket[ziel_bucket].add(a.get(z));
            }
            // Einsammeln
            int pos = a.size()-1;
            for(int b=9; b>=0; b--) {
                while(!bucket[b].isEmpty()) {
                    a.set(pos, bucket[b].removeLast());
                    pos--;
                }
            }
            // Zwischen/Endergebnis
            System.out.println(Arrays.toString(a.toArray()));
        }

    }
```