Aufgabe 3

2. Berechnen Sie das Laufzeitverhalten für allgemeines N auf Basis der Implementierung.

```
for (int sz = 1; sz < N; sz = 3* sz) {

            for (int lo = 0; lo < N - sz; lo += 3 * sz) {

                merge(a, aux, lo, lo + sz - 1,
                    Math.min(lo + 2 * sz - 1, N - 1),
                    Math.min(lo + 3 * sz - 1, N - 1));
            }
        }
```

⇒ Dieser Bottom-Up-Merge-Sort benötigt zwischen $\frac{2}{3}N\lceil log_3 N\rceil$ und $N\lceil log_3 N\rceil$ Vergleiche und höchstens $6N\lceil log_3 N\rceil$ Arrayzugriffe, um ein Array der Länge N zu sortieren.

$\lceil log_3 N\rceil$ Durchläufe (äußere Schleife)

$\frac{N}{3*sz}$ Durchläufe (innere Schleife)

Arrays der Größe $3 * sz$:

- max. Vergleiche: $3 * sz$
- min. Vergleiche: $2 * sz$
- Arrayzugriffe: $6 * (3 * sz)$

Also erhalten wir für die innere Schleife:

- max. Vergleiche: $\frac{N}{3*sz} * 3 * sz = N$
- min. Vergleiche: $\frac{N}{3sz} * 2 * sz = \frac{2}{3}N$
- Arrayzugriffe: $\frac{N}{3*sz} * 6 * (3 * sz) = 6N$

3. Welche Optimierungen kennen Sie für MergeSort?

- Kleine Teilarrays (ab einem Schwellwert) mit Insertion Sort sortieren → Merge Sort benötigt zusätzlichen Speicher wegen den Hilfsarrays
- Vor dem Merge prüfen, ob das Zielarray bereits sortiert ist:
  - Wenn das Element in der Mitte kleiner gleich dem Element an Mitte + 1 ist, ist aufgrund der **Invariante** von Merge-Sort das gesamte Array sortiert.
  - Wenn das letzte Element kleiner gleich dem ersten Element ist, können die beiden Teilarrays getauscht werden (ersetzt die Vergleiche im Merge)