Musterlösung EFM 8.3.1

1. Zwei Klassen erstellen, Code kopieren, ausführen… Jep… läuft
2. Die innere for-Schleife begrenzt den zu durchsuchenden Bereich auf den vorderen Bereich. Der Vorteil des Bubblesorts ist, dass nach jedem die jeweils hinterste Zahl an der richtigen Position steht. Somit ist klar, dass nach Durchgang 1 die letzte Zahl an der richtigen Stellen und nach dem zweiten Durchgang die letzte und vorletzt Zahl an der richtigen Position stehen.
   Demnach wird durch die innere Schleife der Bereich ignoriert, der bereits sortiert wurde.

   6, 3, 2, 5, 1
   3, 2, 5, 1, **6**
   2, 3, 1, **5, 6**
   2, 1, **3, 5, 6**
   1, 2, 3, 5, 6

Musterlösung EFM 8.3.2

1. Die Aussage ist falsch. Denn eigentlich wächst die Sortierzeit mit der Anzahl N – 1 der Zahlen. Außerdem wird in dem Beispiel die verbesserte Version des Bubbelsorts verwendet. Hierbei ist der Aufwand Linear.
   Anzahl Vergleiche:
   $(n-1)^2$

   => $Vergleiche_{min} = (n-1)^2 = n^2-2n+1$
   => $Umspicherungen_{min} = 0$
   => Aufwand insgesamt: $n^2-2n+1+0 = O(n^2)$

2. (Beispiel Programm siehe Anhang)
   Die Behauptung ist nicht durch Messung beweisbar, da die Dauer immer abhängig von der Aufstellung des Arrays ist. Wenn dieser bereits zu einem großen Teil sortiert ist, verringert sich die Verarbeitungszeit trotz größerer Listen Länge.

Musterlösung EFM 8.4.1

1. Siehe Anhang (Liste.java | Zeile 46)
2. Siehe Anhang (Liste.java | Zeile 55)

Musterlösung EFM 8.4.2

1. Siehe Anlage (Liste.java | Zeile 67)
   Selection Sort ist schneller

Musterlösung EFM 8.5.1

1. Siehe Anlage (Liste.java | Zeile 97)
2. Siehe Anlage (Liste.java | Zeile 115)
   Insertion Sort ist schneller