Laufzeitanalyse. Zeilen 3 und 4 benötigen konstante Laufzeit in $\mathcal{O}(1)$. In Iteration i der äußeren Schleife hat die innere `for`-Schleife genau $n-1-i$ Iterationen. Daher ist die Anzahl der Aufrufe

$$\sum_{i=0}^{n-2}(n-1-i) = \sum_{k=1}^{n-1} \frac{n(n-1)}{2} \in \Theta(n^2)$$

$\square$

Korrektheitsbeweis. Zeige mittels vollständiger Induktion über k folgende Schleifeninvariante: Für $k = 0, 1, \ldots, n-2$ gilt nach k Iterationen der äußeren `for`-Schleife

$$A[i] \not\succ A[j] \text{ für } i = 0, \ldots, k-1 \text{ und } j = i+1, \ldots, n-1$$

d. h. die Felder $A[0], \ldots, A[k-1]$ sind topologisch aufsteigend sortiert und keiner ist größer als eines der übrigen Felder $A[k], \ldots, A[n-1]$.
Induktionsanfang ($k = 0$): Trivial (leere Aussage.)
Induktionsschluss ($k \to k+1$):

Die Behauptung gelte für ein beliebiges, aber fest gewähltes k, d. h.

$$A[i] \not\succ A[j] \text{ für } i = 0, \ldots, k-1 \text{ und } j = i+1, \ldots, n-1$$

Betrachte nun $(k+1)$-te Iteration der äußeren `for`-Schleife (d. h. $i = k$). Nach Durchlauf der inneren `for`-Schleife ($j = k+1, \ldots, n-1$) gilt

$$A[k] \not\succ A[j] \text{ für } j = k+1, \ldots, n-1$$

Gemeinsam implizieren diese Aussagen die Schleifeninvariante für $k+1$. $\square$

```python
1 def selectionSort(A):
2 # sort the given list A
3   n = len(A)
4   for i in range(0, n - 1):
5     for j in range(i + 1, n):
6       if A[i] % A[j] == 0:
7         A[i], A[j] = A[j], A[i]
```

Bemerkung. Für partielle Ordnungen gibt SelectionSort eine topologische Sortierung zurück.

8.3 Insertion Sort

Algorithm 1: InsertionSort

Input: Array A der Länge n, totale Ordnung $\preceq$
Output: A sortiert

```
1 for i ← 1 to n − 1 do
2     s ← A[i]
3     k ← i while k > 0 ∧ s ≺ A[k − 1] do
4         A[k] ← A[k − 1]
5         k ← k − 1
6     end
7     A[k] ← s
8 end
```

```python
def insertionSort(A):
  n = len(A)
  for i in range(1, n):
    s = A[i]
    k = i
    while k > 0 and s < A[k - 1]:
      A[k] = A[k - 1]
      k -= 1
    A[k] = s
```

Theorem 8.1. *Sortieren durch Einfügen ist korrekt und hat Laufzeit* $\Theta(n^2)$.

Bemerkung. Es sei $\preceq$ nur eine partielle Ordnung.

- Sortieren durch Einfügen liefert im Allgemeinen keine topologische Sortierung.
- Gegenbeispiel: Ist $\preceq$ die Teilbarkeitsrelation, so liefert InsertionSort bei Eingabe $A = [4, 5, 2]$ die Ausgabe $A = [4, 5, 2]$.
- Für die Aussgabe ist in diesem Fall lediglich

$$A[0] \not\succeq A[1] \not\succeq \ldots \not\succeq A[n-1]$$

8.4 Divide and Conquer

Idee für einen rekursiven Sortieralgorithmus:

(i) Teile zu sortierende Menge A in zwei etwa gleich große Teilmengen auf.

(ii) Sortiere beide Teilmengen (rekursiv).

(iii) Füge die beiden Sortierungen zu einer Sortierung von A zusammen.

Algorithm 2: Die Merge-Operation

Input: Sortierungen $\pi_A = \{1, 2, \ldots, |A|\} \to A, \pi_B = \{1, 2, \ldots, |B|\} \to B$
Output: Sortierung $\pi : \{1, 2, \ldots, |A + B|\} \to A \cup B$

$a \leftarrow 1$
$b \leftarrow 1$
$c \leftarrow 1$ **while** $a \leq |A| \wedge b \leq |B|$ **do**
    if $\pi_A(a) \preceq \pi_B(b)$ **then**
        $\pi(c) \leftarrow \pi_A(a)$
        $a \leftarrow a + 1$
    end
    else
        $\pi(c) \leftarrow \pi_B(b)$
        $b \leftarrow b + 1$
    end
    $c \leftarrow c + 1$
end
while $a \leq |A|$ **do**
    $\pi(c) \leftarrow \pi_A(a)$
    $a \leftarrow a + 1$
    $c \leftarrow c + 1$
end
while $b \leq |B|$ **do**
    $\pi(c) \leftarrow \pi_B(b)$
    $b \leftarrow b + 1$
    $c \leftarrow c + 1$
end

Lemma 8.2. *Die Merge-Operation arbeitet korrekt. Sie benötigt* $\leq |A| + |B| - 1$ *Vergleiche und hat Laufzeit* $\mathcal{O}(|A| + |B|)$.