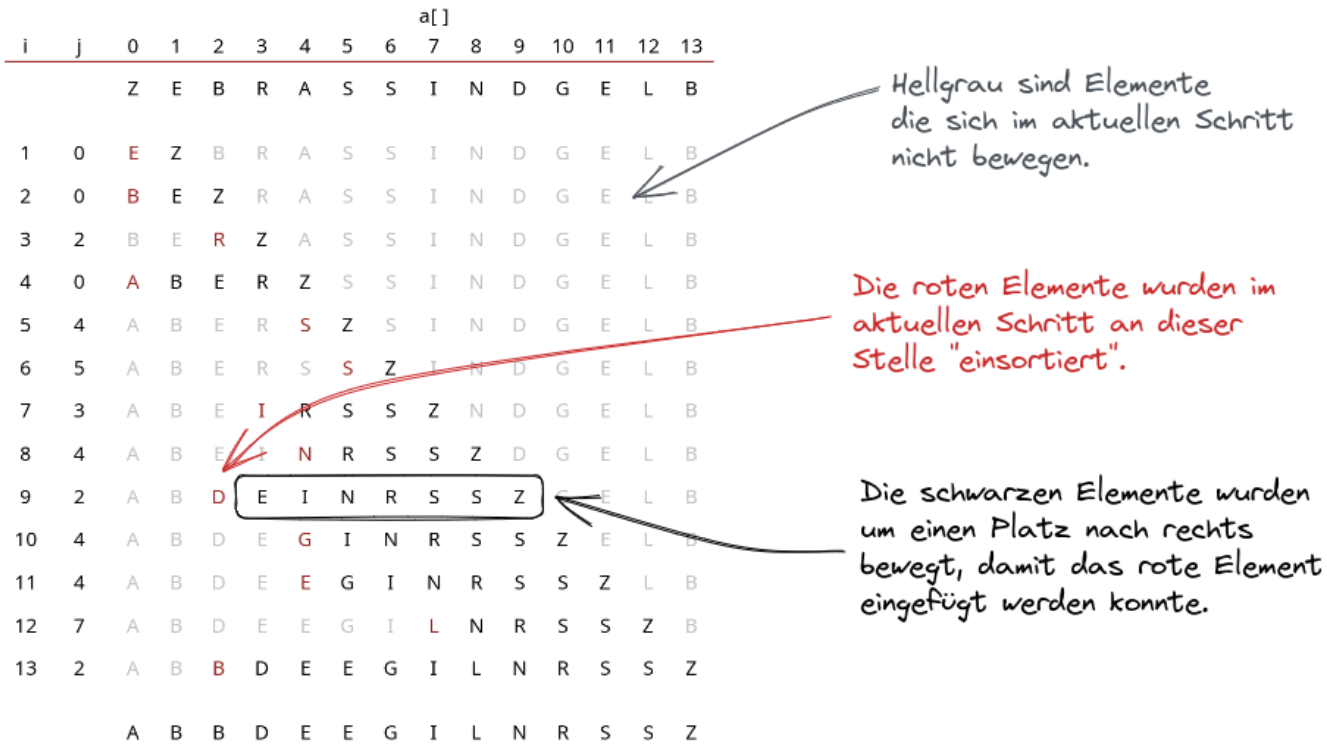
Insertion Sort

Während Selection Sort jeweils alle noch nicht bearbeiteten Elemente betrachtet hat, um das kleinste zu finden, orientiert sich **Insertion Sort** nach links: Es betrachtet jeweils ein Element und rückt dieses dann so weit nach links, bis es an seiner korrekten Position innerhalb der bislang betrachteten Elemente gelandet ist.

Beispiel

Für die Zeichenkette „ZEBRASSINDGELB“ sieht das dann folgendermaßen aus:



(A1)

- Implementiere im Bluej-Projekt https://codeberg.org/qg-info-unterricht/algs4-sort-bluej Selectionsort.
- Erzeuge mithilfe der `draw`-Methode eine Veranschaulichung des Sortiervorgangs wie im Bild oben. Du musst dazu die etwas angepasste Methode `drawInsertion` nutzen, um die Färbung für Insertionsort korrekt zu erzeugen.

(A2)

Wenn alle Elemente der zu sortierenden Liste denselben Sortier-Schlüssel haben (die zu sortierende Liste also z. B. nur aus "aaaaaaa" besteht), welches Verfahren ist dann effizienter: Insertion-Sort oder Selection-Sort? Begründe deine Antwort!

Tipp

Veranschauliche in beiden Verfahren das Sortieren einer Zeichenkette aus gleichen Buchstaben und überlege dir, welches Verfahren effizienter ist.

		a[ ]									
i	j	0	1	2	3	4	5	6	7	8	9
		a	a	a	a	a	a	a	a	a	a
1	1	a	a	a	a	a	a	a	a	a	a
2	2	a	a	a	a	a	a	a	a	a	a
3	3	a	a	a	a	a	a	a	a	a	a
4	4	a	a	a	a	a	a	a	a	a	a
5	5	a	a	a	a	a	a	a	a	a	a
6	6	a	a	a	a	a	a	a	a	a	a
7	7	a	a	a	a	a	a	a	a	a	a
8	8	a	a	a	a	a	a	a	a	a	a
9	9	a	a	a	a	a	a	a	a	a	a
		a	a	a	a	a	a	a	a	a	a

		a[ ]									
i	min	0	1	2	3	4	5	6	7	8	9
		a	a	a	a	a	a	a	a	a	a
0	0	a	a	a	a	a	a	a	a	a	a
1	1	a	a	a	a	a	a	a	a	a	a
2	2	a	a	a	a	a	a	a	a	a	a
3	3	a	a	a	a	a	a	a	a	a	a
4	4	a	a	a	a	a	a	a	a	a	a
5	5	a	a	a	a	a	a	a	a	a	a
6	6	a	a	a	a	a	a	a	a	a	a
7	7	a	a	a	a	a	a	a	a	a	a
8	8	a	a	a	a	a	a	a	a	a	a
9	9	a	a	a	a	a	a	a	a	a	a
		a	a	a	a	a	a	a	a	a	a

(A3)

Ein Sortieralgorithmus ist **stabil**, wenn er die ursprüngliche Reihenfolge von Elementen mit gleichem Schlüsselwert beibehält.

Das bedeutet, wenn zwei Elemente den gleichen Sortierschlüssel haben, bleibt ihre relative Position zueinander nach dem Sortieren unverändert.

Diese Eigenschaft ist besonders wichtig, wenn:

- Datensätze nach mehreren Kriterien sortiert werden sollen
- Die ursprüngliche Reihenfolge eine semantische Bedeutung hat

Als Beispiel: Wenn wir eine Liste von Personen erst nach Alter und dann nach Namen sortieren, ist es oft wichtig, dass die ursprüngliche Reihenfolge bei gleichaltrigen Personen erhalten bleibt.

Ein Gegenbeispiel ist der Selectionsort-Algorithmus, der nicht stabil ist, da beim Vertauschen von Elementen im unsortierten Bereich die ursprüngliche Reihenfolge von Elementen mit gleichem Wert verändert werden kann

(a) Überprüfe, ob deine Implementation von Selectionsort stabil ist.

(b) Ändere in der Klasse "Sorting" die Methode `less` so, dass diese auch dann 0 zurückgibt, wenn die zu vergleichenden Elemente gleich sind:

```
protected boolean less(String v, String w) {
    return v.compareTo(w) <= 0; // Kleiner gleich, nicht nur echt kleiner!
}
```

Überprüfe erneut, ob sein Selectionsort Algorithmus jetzt stabil ist.