Gruppe E
Merge-Sort

Vorgehen:
Erarbeiten Sie zusammen das Verfahren „Merge-Sort“. Lesen Sie dazu den kurzen Einführungstext auf dieser Seite und teilen Sie die übrigen Teile untereinander auf:

1. Textbeschreibung
2. Struktogrammdarstellung
3. Pseudo-Code

Erklären Sie sich das Verfahren gegenseitig. Überlegen Sie, wie Sie es den anderen Schüler:innen erklären können. Entwerfen Sie eine Präsentation und ein Handout zu dem Verfahren, das Sie den übrigen Schüler:innen zur Verfügung stellen können. Bereiten Sie einen Kurzvortrag vor, in dem Sie das Verfahren vorstellen.

Begründen Sie auch, warum das Verfahren eine Sortierung zur Folge hat. Zählen Sie die Vergleiche, die ausgeführt werden für:

a = [1, 2, 3, 4, 5, 6]
b = [6, 5, 4, 3, 2, 1]

Entscheiden Sie, ob dieses Sortierverfahren stabil ist. Begründen Sie Ihre Entscheidung.

Dieser Algorithmus gehört zu denen, die das Prinzip „devide and conquer“ (teile und herrsche) umsetzen. Überlegen Sie, was mit diesem Prinzip gemeint sein könnte.

Teilung des Arrays in einen sortierten und einen unsortierten Teil

Bei den Sortieralgorithmen, die wir besprechen, wird das Array in einen sortierten und einen unsortierten Teil geteilt. In Ihrer Präsentation sollte immer deutlich werden, welcher Teil sortiert und welcher Teil unsortiert ist.
Um es einheitlicher zu gestalten, sollten Sie den sortierten Teil mit grüner und den unsortierten Teil mit roter Farbe verdeutlichen.

Ausgangspunkt der Sortierung ist folgendes unsortiertes Array

array = [18, 10, 38, 11, 23, 24, 45, 11]

In der grafischen Darstellung:

18	10	38	11	23	24	45	11

Merge-Sort: Textbeschreibung

Wie kann man sehr einfach eine Aufgabe lösen? Indem man sie aufteilt. Wir teilen unser Array einfach in der Mitte. Es entstehen die Arrays l (für links) und r (für rechts).

18	10	38	11	23	24	45	11

l :

18	10	38	11

r:

23	24	45	11

Da beide Arrays immer noch zu groß sind, teilen wir diese Arrays erneut in ll (linkes Teilarray von l), lr (rechtes Teilarray von l), rl (linkes Teilarray von r) und rr (rechtes Teilarray von r):

ll :

18	10

lr:

38	11

rl:

23	24

rr:

45	11

Da alle so entstandenen Teilarrays immer noch zu groß sind, um sortiert zu sein (nur Arrays mit 0 oder 1 Element sind sortiert), müssen alle noch einmal halbiert werden. So entstehen die Arrays bei Teilung von ll die Teilarrays lll = [18], llr=[10], ferner entstehen die Teilarrays lrl = [38], lrr = [11], rll = [23], rlr = [24], rrl = [45] und rrr = [11].

Jetzt werden die Teilarrays sukzessive wieder zusammengebaut. Dafür ist es am anschaulichsten, wenn wir uns die Arrays als Kartenstapel vorstellen. Um ll zu sortieren, sehen wir uns die Teilarrays lll und llr an. Da die oberste Karte von llr kleiner ist als die von lll. Deswegen wird die Karte von lll unter die vor llr gelegt. So erhalten wir das sortierte Array ll*=[10, 18]. Bei einem gleichen Vergleich erhalten wir die sortierten Arrays lr* =[11, 38], rl* = [23, 24], rr* = [11, 45]. Um l zu sortieren, vergleichen wir die Kartenstapel ll*=[10,18] und lr* = [11,38]:

l*	Oberste Karte ll*	Oberste Karte lr*	Wir nehmen ...
[]	10	11	10
[10]	18	11	11
[10,11]	18	38	18
[10,11,18]	_	38	38
[10,11,18,38]	_	_	

Rückseite beachten!

In ähnlicher Weise erhalten wir durch den Vergleich von rl* = [23, 24] und rr* = [11, 45] das sortierte Array r*:

r*	Oberste Karte rl*	Oberste Karte rr*	Wir nehmen ...
[]	23	11	11
[11]	23	45	23
[11,23]	24	45	24
[11,23,24]	_	45	45
[11,23,24,45]	_	_	

Jetzt können wir unser Ausgangsarray sortieren. In der folgenden Tabelle werden die Arrays aufgeführt, das „oberste“ ist farbig markiert. Ferner sind die Indizes der „obersten“ Elemente angegeben.

Ergebnis	l*	r*	Wir nehmen ...
[]	0: [**10**,11,18,38]	0: [**11**,23,24,45]	10
[10]	1: [10,**11**,18,38]	0: [**11**,23,24,45]	11
[10,11]	2: [10,11,**18**,38]	0: [**11**,23,24,45]	11
[10,11,11]	2: [10,11,**18**,38]	1: [11,**23**,24,45]	18
[10,11,11,18]	3: [10,11,18,**38**]	1: [11,**23**,24,45]	23
[10,11,11,18,23]	3: [10,11,18,**38**]	2: [11,23,**24**,45]	24
[10,11,11,18,23,24]	3: [10,11,18,**38**]	3: [11,23,24,**45**]	38
[10,11,11,18,23,24,38]	4: [10,11,18,38]	3: [11,23,24,**45**]	45
[10,11,11,18,23,24,38, 45]	4: [10,11,18,38]	4: [11,23,24,45]	

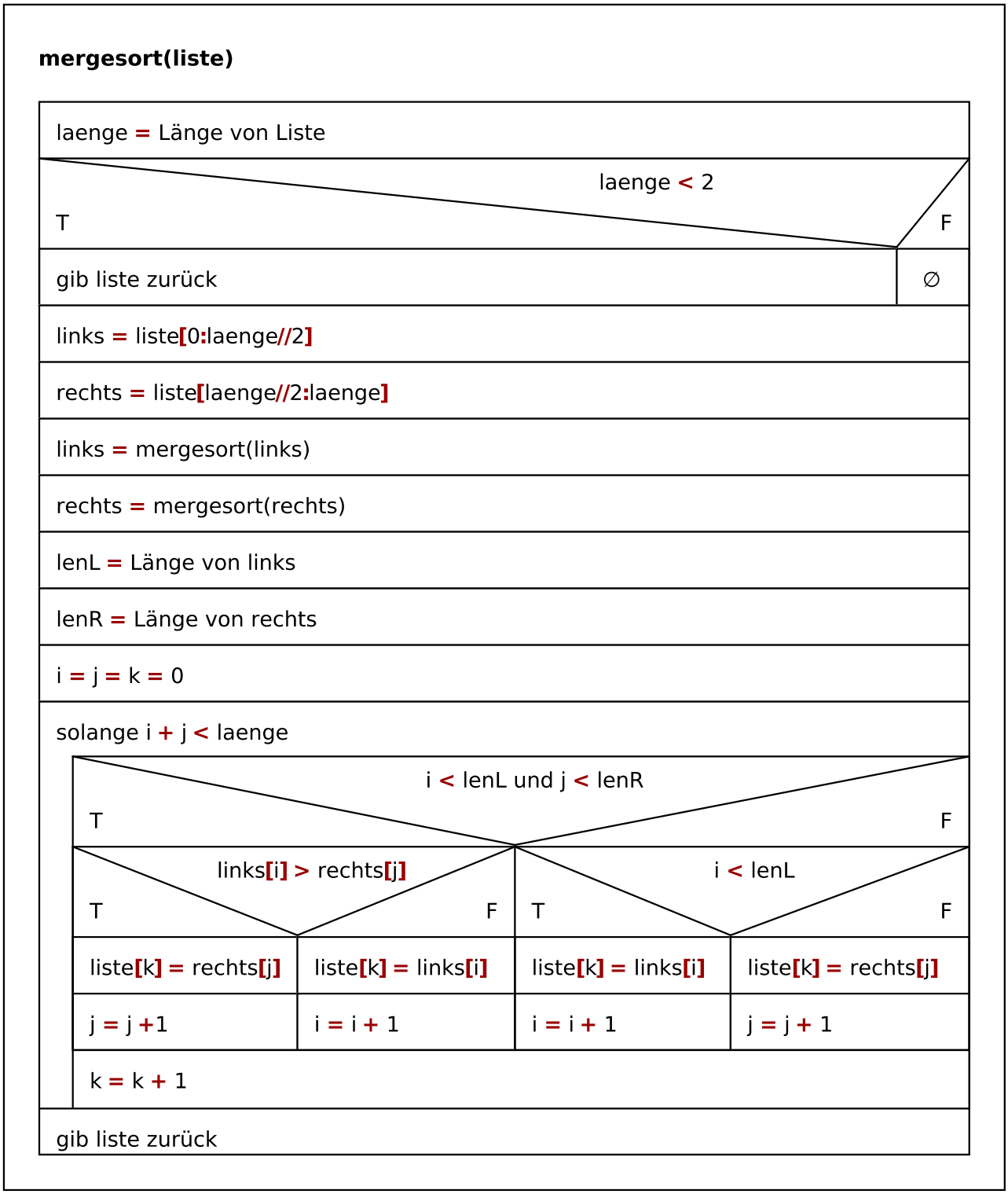
Merge-Sort: Pseudo-Code

```
funktion mergesort(liste)
      laenge = Länge der Liste
      wenn lange < 2
            gib liste zurück
      links = alle Elemente von Liste vom Index 0 bis zum Index laenge//2 - 1
      rechts = alle Elemente von Liste vom Index laenge//2 bis zum Index laenge-1
      links = mergesort(links)
      rechts = mergesort(rechts)
      lenL = Länge von links
      lenR = Länge von rechts
      i = 0
      j = 0
      k = 0
      solange i + j < laenge
            wenn i < lenL und j < lenR
                  wenn links[i] > rechts[j]
                        liste[k] = rechts[j]
                        j = j + 1
                  sonst
                        liste[k] = links[i]
                        i = i + 1
            sonst
                  wenn i < lenL
                        liste[k] = links[i]
                        i = i + 1
                  wenn j < lenR
                        liste[k] = rechts[j]
                        j = j + 1
            k = k + 1
      gib liste zurück
```

Übertragen Sie diesen Pseudo-Code in eine Python-Funktion mergesort(liste) und testen Sie ihn.

Spielen Sie den Code mit den Karten durch.

Merge-Sort: Struktogramm



Führen Sie mit einem kurzen Array:
test = [20, 10, 30, 5, 10]
einen Trockentest durch.