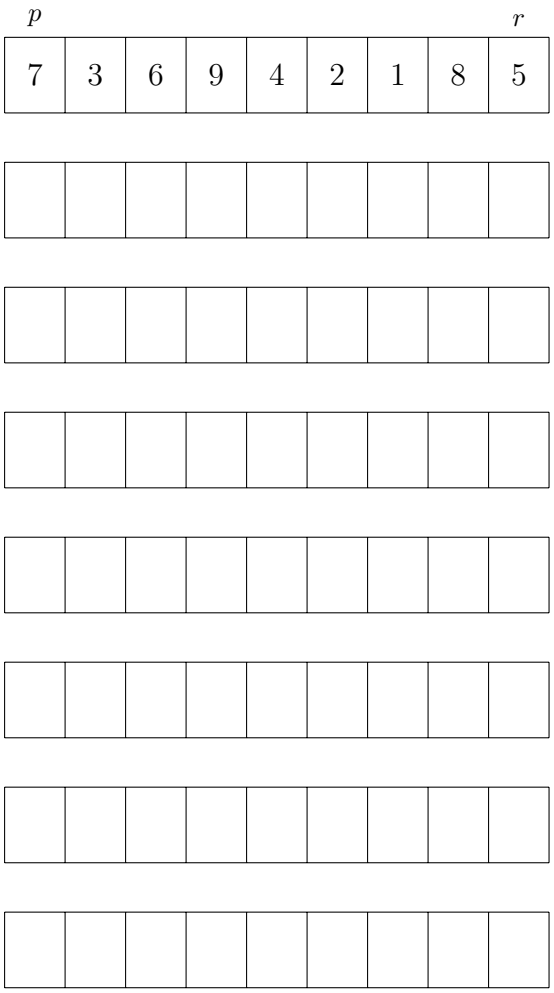
Aufgabe 1

(a) Zeige schrittweise, wie die Funktion PARTITION im Quicksort-Algorithmus die unten gegebene Liste verarbeitet. Notiere nur jeden neuen Zustand der Liste.

(b) Welchen Wert gibt die Funktion PARTITION in (a) für $p = 10$ zurück?

(c) Gib die Laufzeitkomplexität von PARTITION in der O-Notation an.



Aufgabe 2

(a) Wie viele Vertauschungen führt PARTITION(A, 1, 5) auf der folgenden Liste aus? Zähle auch „unsichtbare“ Vertauschungen.

1				5
5	6	7	8	9

(b) Welchen Wert gibt PARTITION(A, 1, 5) in (a) zurück?

(c) Wie oft wird PARTITION(A, ?, ?) bei der Ausführung von QUICKSORT(A, 1, 5) auf der Liste in (a) aufgerufen?

Aufgabe 3

Gib die Laufzeitkomplexität von „normalem“ Quicksort für ein Array der Länge n in den folgenden drei Fällen an.

Best Case	Average Case	Worst Case

Aufgabe 4

Beschreibe mindestens eine Strategie, mit der man beim Quicksort-Algorithmus die Worst Case-Laufzeit verhindern kann.

Aufgabe 5

Beschreibe möglichst detalliert (evtl. mit Skizzen), wie MERGE(A, 8, 10, 14) auf den unten dargestellten Teil des Arrays A operiert.

	7	8	9	10	11	12	13	14	15
A	...	2	4	9	1	3	5	6	...

Aufgabe 6

Zeige tabellarisch, wie Mergesort die folgende Liste sortiert, indem du den neuen Zustand der Liste nach jedem Aufruf von Merge protokollierst. Achte auf die richtige Reihenfolge der Änderungen, die sich aus den rekursiven Aufrufen ergibt.

12	9	17	3	8	1	21	4

Aufgabe 7

Vergleiche Quicksort und Mergesort in Bezug auf.

(a) Speicherbedarf

(b) Stabilität

Gehe jeweils von einem Array mit n Elementen aus.

Aufgabe 8

Zeige, wie Countingsort das folgende Array (stabil) sortiert.

$A = [4, 2, 0, 4, 3, 2, 4, 3, 2]$

Aufgabe 9

Charakterisiere Countingsort bezüglich

(a) Voraussetzungen

(b) Laufzeitkomplexität

(c) Speicherbedarf

(d) Stabilität