OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Programmieren 1 – Informatik

Folien zur Quicksort-Übung

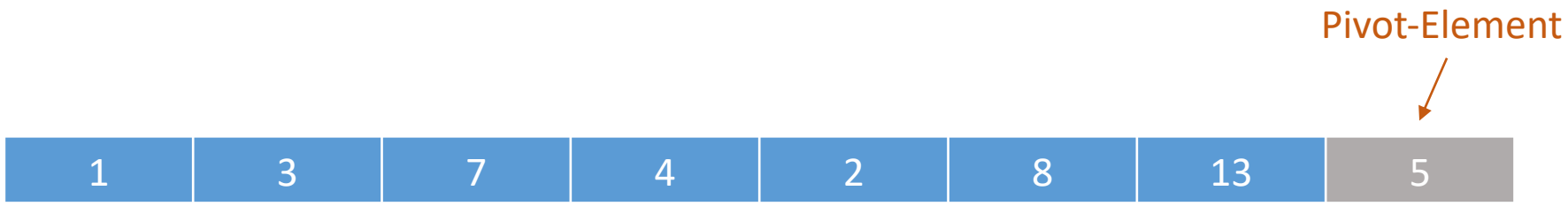
Prof. Dr.-Ing. Maike Stern

Quicksort

- In der Übung soll der Quicksort-Algorithmus implementiert werden
- Quicksort ist ein schneller Sortieralgorithmus, der das Teile-und-Herrsche-Prinzip einsetzt (divide and conquer)
- Divide bezieht sich hierbei auf das Unterteilen des ursprünglichen Arrays in Subarrays, die rekursiv bearbeitet werden (conquer)
- Die folgenden Folien erklären zunächst die grundsätzliche Idee hinter Quicksort sowie die Implementierung an einem Beispiel
- Zusätzliche Quellen (und Implementierungen) finden Sie im Internet, z.B.
  - https://www.linux-related.de/index.html?/coding/sort/sort_quick.htm
  - https://www.programiz.com/dsa/quick-sort
  - https://big-o.io/algorithms/comparison/quicksort/
- In C ist Quicksort in der stdlib-Headerdatei implementiert (qsort())

Quicksort

- Gegeben: Array mit n Elementen
- Algorithmus:
  - Auswählen eines sogenanntes Pivot-Elements (typischerweise das letzte Element im Array)

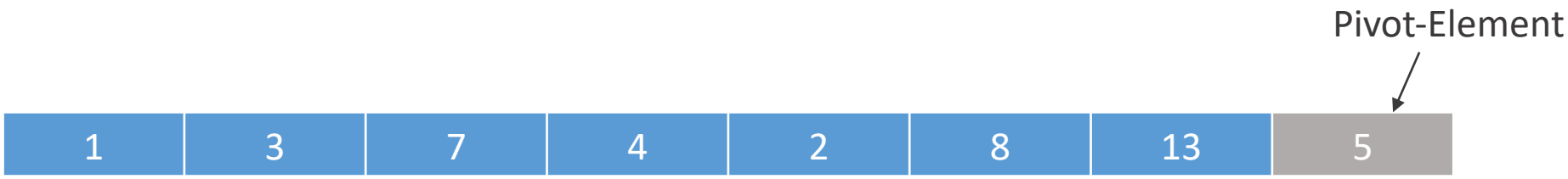


Quicksort

Algorithmus:

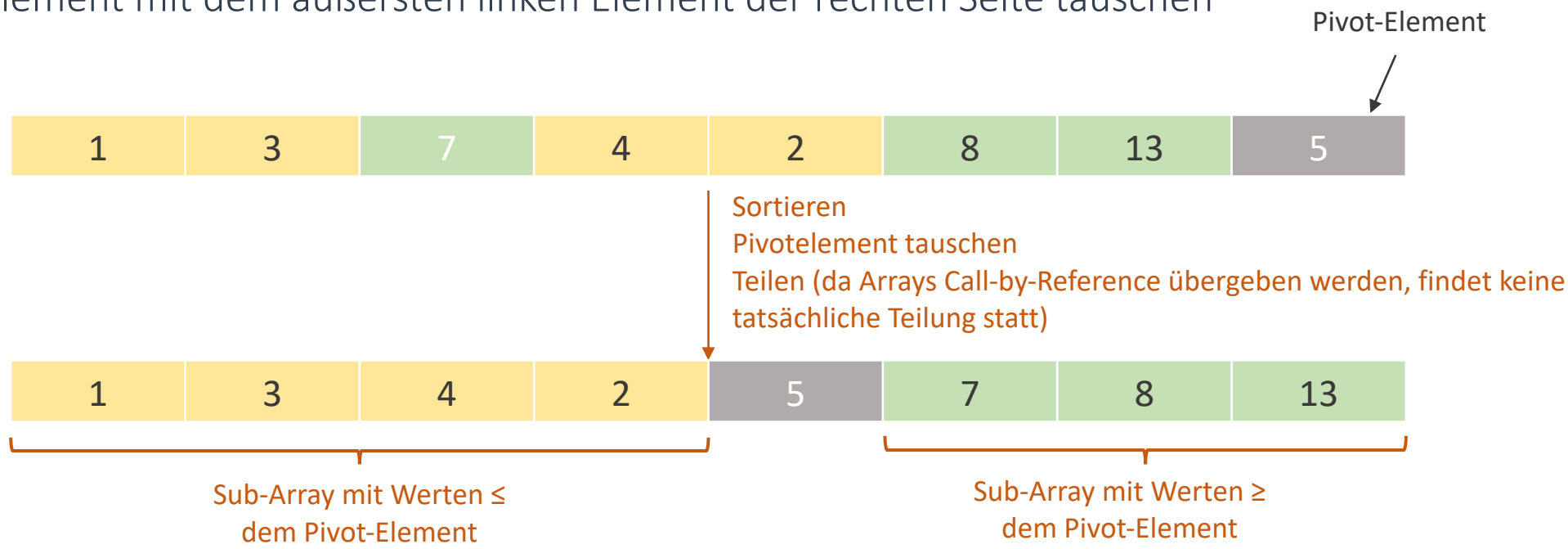
Gegeben: Array mit n Elementen

1. Auswählen eines sogenanntes Pivotelements
   - Das Pivotelement liegt idealerweise von der Wertigkeit her in der Mitte (bei Werten zwischen 0 und 10 wäre das ideale Element also 5), da dann die Rekursionstiefe gering wäre
   - Die Schwierigkeit ist natürlich, das mittlere Element zu finden
   - Eine Möglichkeit: Zufällig drei Elemente wählen und das mittlere Element als Pivot-Element verwenden (Name der Technik: median-of-three)
   - Im einfachsten Fall wird das letzte Array-Element verwendet



Quicksort

1. Auswählen eines sogenanntes Pivot-Elements (der Einfachheit halber das letzte Element im Array)
2. Sortieren der Elemente und Aufteilen des Arrays in zwei Sub-Arrays
   - 1. Sub-Array: Alle Elemente sollen kleiner oder gleich dem Pivot-Element sein
   - 2. Sub-Array: Alle Elemente sollen größer oder gleich dem Pivot-Element sein
   - Pivotelement mit dem äußersten linken Element der rechten Seite tauschen



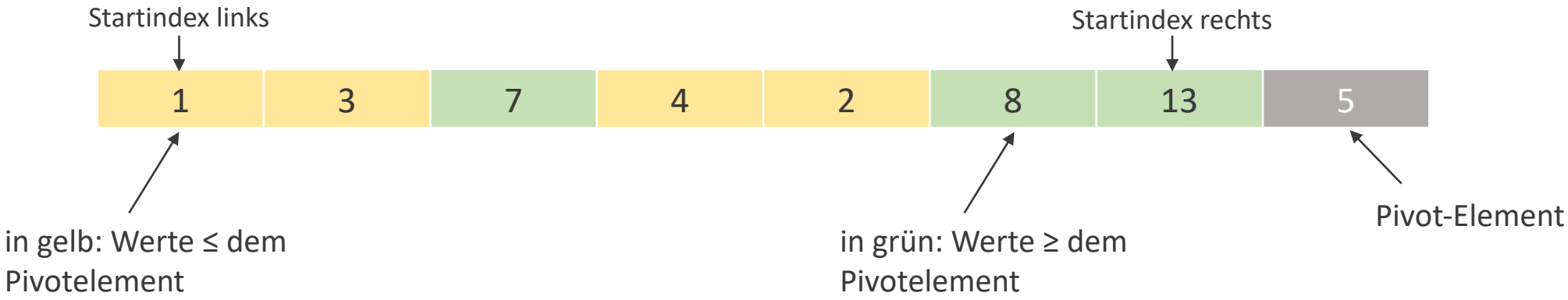
Quicksort

Funktionsweise:

1. Auswählen eines sogenanntes Pivot-Elements (der Einfachheit halber das letzte Element im Array)
2. Sortieren der Elemente und Aufteilen des Arrays in zwei Sub-Arrays
3. Rekursives Abarbeiten der Subarrays, bis nur noch ein Element im Subarray ist, das dann automatisch an der richtigen Stelle sitzt

Programmiertechnische Realisierung

1. Festlegen des letzten Arrayelements als Pivotelement

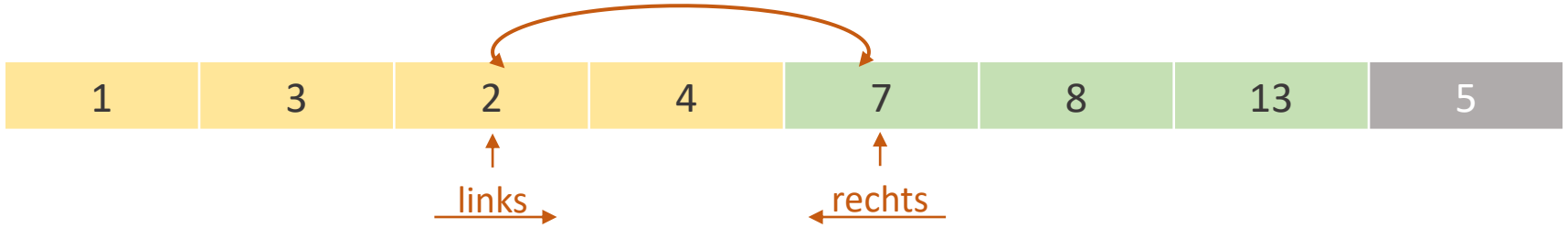


Quicksort
Programmiertechnische Realisierung

1. Festlegen des letzten Arrayelements als Pivotelement
2. Sortieren der Elemente, so dass alle Elemente ≤ dem Pivotelement nach links verschoben werden und alle Elemente ≥ dem Pivotelement nach rechts
   Umsetzung:
   - Zwei Pointer, die jeweils auf die Anfangs- bzw. Endadresse des Arrays zeigen (*links* und *rechts*)
   - Der linke Pointer durchläuft das Array von links nach rechts
     - Wird ein Element gefunden, das größer oder gleich dem Pivotelement ist, so bleibt der *linke* Pointer bei diesem Element stehen
   - Steht der linke Pointer, so durchläuft der rechte Pointer das Array von rechts nach links
     - Wird ein Element gefunden, das kleiner oder gleich dem Pivotelement ist, so bleibt der *rechte* Pointer bei diesem Element stehen

Quicksort
Programmiertechnische Realisierung

1. Festlegen des letzten Arrayelements als Pivotelement
2. Sortieren der Elemente, so dass alle Elemente ≤ dem Pivotelement nach links verschoben werden und alle Elemente ≥ dem Pivotelement nach rechts
   Umsetzung:
   - Zwei Pointer, die jeweils auf die Anfangs- bzw. Endadresse des Arrays zeigen (*links* und *rechts*)
   - Der linke Pointer durchläuft das Array von links nach rechts
     - Wird ein Element gefunden, das größer oder gleich dem Pivotelement ist, so bleibt der *linke* Pointer bei diesem Element stehen
   - Steht der linke Pointer, so durchläuft der rechte Pointer das Array von rechts nach links
     - Wird ein Element gefunden, das kleiner oder gleich dem Pivotelement ist, so bleibt der *rechte* Pointer bei diesem Element stehen
   - Stehen rechter und linker Pointer, so werden die Werte vertauscht.



3. Wurden die Elemente (hier 2 und 7) vertauscht, läuft der *linke* Pointer wieder los, bis er ein Element findet, das ≥ dem Pivotelement ist (hier 7). Dann läuft der *rechte* Pointer los, bis er ein Element findet, das ≤ dem Pivotelement ist (hier 4). Jetzt ist aber der Index von *rechts* ≤ *links*, daher findet kein Tausch statt sondern der Durchlauf wird abgebrochen

4. Nach dem Abbruch wird das Pivotelement mit dem äußersten linken Element der rechten Seite getauscht (das ist der Index, an dem der *rechte* Pointer zuletzt einen ≥ Indexwert hatte als der *linke* Pointer, hier also die 7)

Quicksort
Programmiertechnische Realisierung

6. Nachdem das Pivotelement getauscht wurde sitzt es an der korrekten Stelle im Array und muss daher nicht mehr betrachtet werden. Es wird also gesetzt.

7. Anschließend werden zwei Subarrays gebildet, einmal die linke Seite vom gesetzten Pivotelement (also alle Werte die ≤ sind) und einmal die rechte Seite (also alle Werte die ≥ sind). Die Subarrays werden nacheinander rekursiv aufgerufen und entsprechend der Schritte 1 bis 7 abgearbeitet, bis nur noch ein Element vorhanden ist, das dann schon an der richtigen Stelle sitzt. Siehe nachfolgende Folien

Das Pivotelement (5) gilt als gesetzt. Die linke und die rechte Hälfte werden nacheinander rekursiv aufgerufen. Zuerst die linke Seite:

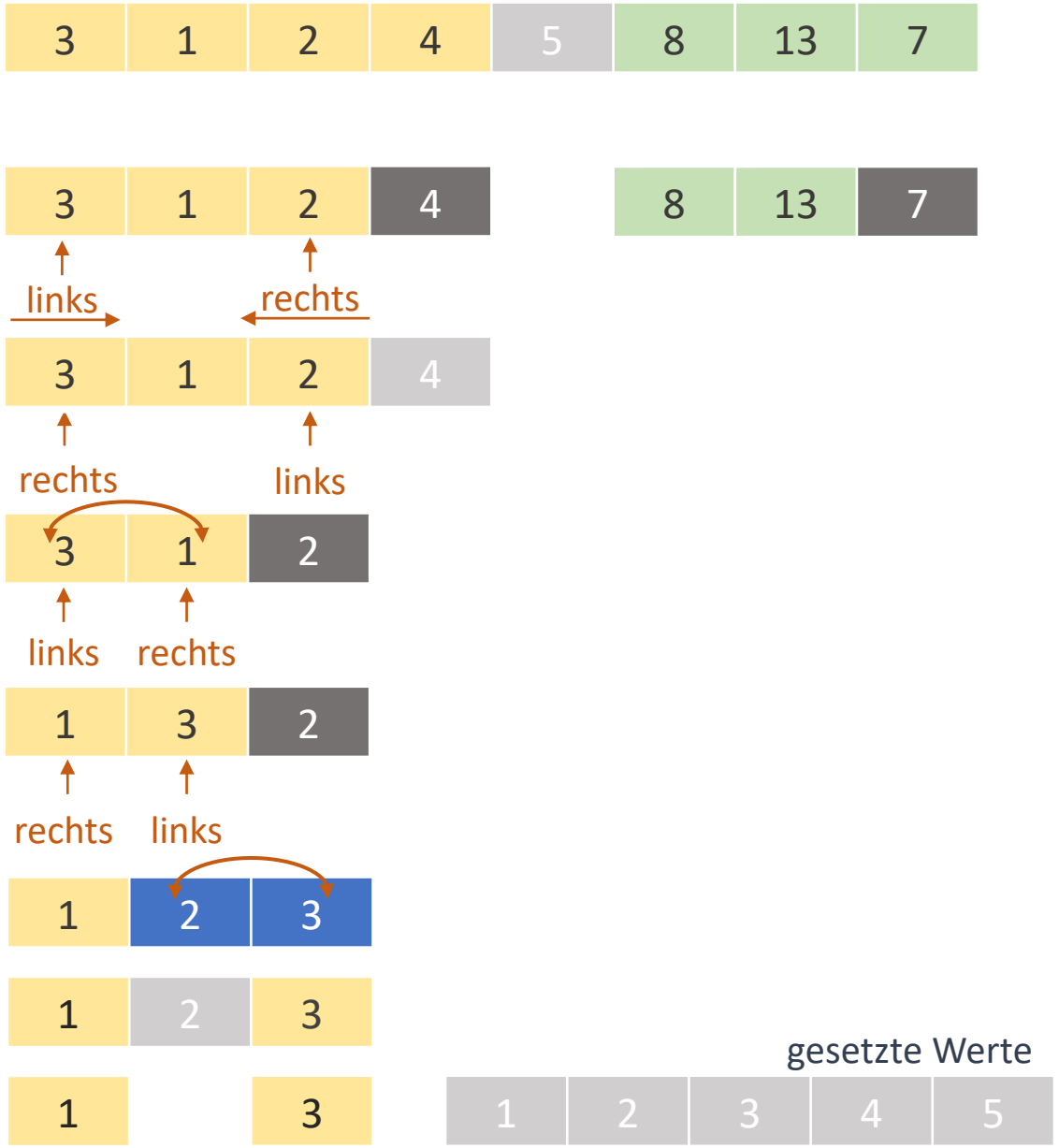
Das jeweils letzte Arrayelement in den Subarrays wird als Pivotelement festgelegt. Links: 4. Der linke und rechte Zeiger laufen entgegengesetzt durch das Array. Zuerst *links,* bis ein Element gefunden wird, das ≥ dem Pivotelement ist, dann bleibt *links* stehen. Dann läuft *rechts* los, bis ein Element gefunden wird, das ≤ dem Pivotelement ist, dann wird getauscht. Wenn der Index von *rechts* ≤ *links* ist, endet der Durchlauf. Da hier alle Werte ≤ dem Pivotelement sind, ist die 4 schon an der richtigen Stelle und wird gesetzt.

Die restlichen Werte werden als neues Subarray rekursiv aufgerufen. Der linke Pointer startet bei 3, einem Wert ≥ 2, der rechte Pointer startet bei 1, einem Wert ≤ 2, daher werden 1 und 3 getauscht.

Links und *rechts* laufen ein Element weiter, so dass der Index von *rechts* ≤ *links* ist – der Durchlauf wird beendet.

Das Pivotelement (2) wird mit dem äußersten linken Element der rechten Seite getauscht (3) und das Pivotelement ist gesetzt.

Da die verbleibenden Subarrays jeweils nur noch ein Element enthalten sind beide Pointer (links und rechts) auf demselben Element, der Durchlauf wird daher direkt beendet und 1 & 3 sind gesetzt. Die Rekursion wird beendet und die rechte Seite wird rekursiv aufgerufen.



Die linke und rechte Seite werden nacheinander rekursiv aufgerufen.
Rechte Seite:

Pivotelement: 7
Der linke Pointer beginnt bei 8, einem Wert ≥ dem Pivotelement, so dass links anhält. Der rechte Pointer beginnt bei 13, einem Wert ≥ dem Pivotwert. Da der rechte Pointer nach einem Wert ≤ dem Pivotwert sucht, läuft er weiter nach links. Schon im nächsten Schritt sind beide Pointer auf demselben Element, so dass der Durchlauf beendet wird.

Anschließend wird das Pivotelement mit dem äußersten linken Element der rechten Seite getauscht, die 7 ist gesetzt.

Die letzten zwei Elemente werden als Subarray rekursiv aufgerufen. Die 8 ist das Pivotelement.

Die Pointer stehen beide auf der 13, daher wird der Durchlauf direkt wieder beendet. Das Pivotelement 8 wird mit der 13 getauscht und die 8 ist gesetzt.

Die 13 bleibt als letztes Element übrig und ist bereits an der korrekten Stelle, am Ende des Arrays.