WBS TRAINING

Programmierung(2)

UNSER ♥ SCHLÄGT FÜR BILDUNG

Agenda

- Sortieralgorithmen (allgemein)
  - Definition
  - Motivation
- Bubblesort
  - Graphische Erläuterung
  - Pseudocode
- Quicksort
  - Graphische Erläuterung
  - Pseudocode
- Fachpraktische Anwendungen

Sortieralgorithmen – Definition

- Unter einem Sortieralgorithmus versteht man ein **Programm** (oder Unterprogramm), mit dessen Hilfe eine **Folge von Daten** (in unserem Fall werden dies ausschließlich die Felderinhalte eines Arrays sein) nach einem festgelegten Ordnungsprinzip (üblicherweise numerisch, alphabetisch oder chronologisch) **sortiert** werden.

- Eine solche Sortierung kann auf sehr unterschiedliche Weise geschehen und bemüht sich in der Regel um (mindestens) eine der drei folgenden **Qualitätsmerkmale**:
  - Die Sortierung braucht im Durchschnitt wenig Zeit
  - Die Sortierung braucht im Worst Case wenig Zeit
  - Die Sortierung benötigt wenig Speicherplatz

- Wir werden uns im Folgenden mit 2 berühmten Suchalgorithmen befassen, die wir aus sehr unterschiedlichen Gründen auswählen:
  - der **Bubblesort** ist besonders anschaulich zu erklären. Sein Algorithmus ist oft Gegentand von **Prüfungen**
  - der **Quicksort** ist deutlich schneller, benötigt wenig Speicherplatz und ist zudem eine interessante **rekursive** Lösung

Sortieralgorithmen – Motivation

- Die **Ausgabe** einer Abfolge von Daten hat oft in sortierter Form zu geschehen. So soll z.B. eine Namensliste üblicherweise in alphabetischer Reihenfolge erscheinen.

- Die Suche nach einem bestimmten Wert innerhalb einer Folge von Daten ist im Durchschnitt deutlich schneller zu leisten, wenn diese Daten sortiert sind:

  Offensichtlich findet man beispielsweise die Telefonnummer zu einem Personennamen X deutlich schneller, wenn die Namen innerhalb des verwendeten Telefonbuches alphabetisch sortiert sind.

- Auch die Bestimmung des Minimums und Maximums einer Datenmenge gelingt natürlich sehr leicht (und vor allem deutlich schneller), wenn diese Datenmenge sortiert vorliegt.

Bubblesort – Graphische Erläuterung

Wir betrachten im Folgenden das **unsortierte**, 6-Felder-lange Integer-Array **arr**.

Ausgangssituation:

Bubblesort – Graphische Erläuterung

Wir werden nun nacheinander jeweils **2 unmittelbar aufeinanderfolgende** Felder untersuchen.

Diese werden jeweils rot markiert:

Bubblesort – Graphische Erläuterung

Wir werden nun nacheinander jeweils **2 unmittelbar aufeinanderfolgende** Felder untersuchen.

Diese werden jeweils rot markiert:

Bubblesort – Graphische Erläuterung

Falls die markierten Felder bereits in der richtigen Reihenfolge nebeneinander stehen, so wird nichts geschehen:

2 ist kleiner als 6, also bleiben die Zahlen unverändert!

Bubblesort – Graphische Erläuterung

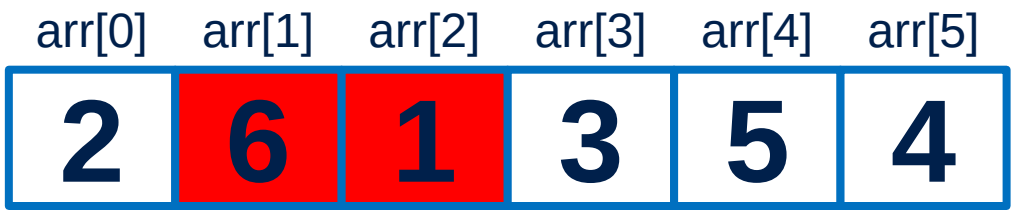
Wir betrachten nun das nächste Paar zweier unmittelbar aufeinanderfolgender Felder.

Auch diese werden natürlich wieder rot markiert:

Bubblesort – Graphische Erläuterung

Wir betrachten nun das nächste Paar zweier unmittelbar aufeinanderfolgender Felder.

Auch diese werden natürlich wieder rot markiert:



Bubblesort – Graphische Erläuterung

Falls die markierten Felder nicht in der richtigen Reihenfolge nebeneinander stehen, so müssen sie **getauscht** werden.

Da 6 größer als 1 ist, werden die beiden Felderinhalte getauscht:

Bubblesort – Graphische Erläuterung

Falls die markierten Felder nicht in der richtigen Reihenfolge nebeneinander stehen, so müssen sie **getauscht** werden.

Da 6 größer als 1 ist, werden die beiden Felderinhalte getauscht:

Bubblesort – Graphische Erläuterung

Dies ist das nächste Paar:

Da 6 größer als 3 ist, muss auch hier getauscht werden:

Bubblesort – Graphische Erläuterung

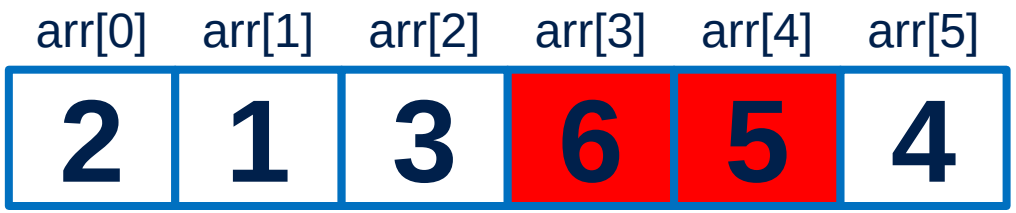
Dies ist das nächste Paar:

Da 6 größer als 3 ist, muss auch hier getauscht werden:

Bubblesort – Graphische Erläuterung

Das nächste Paar:

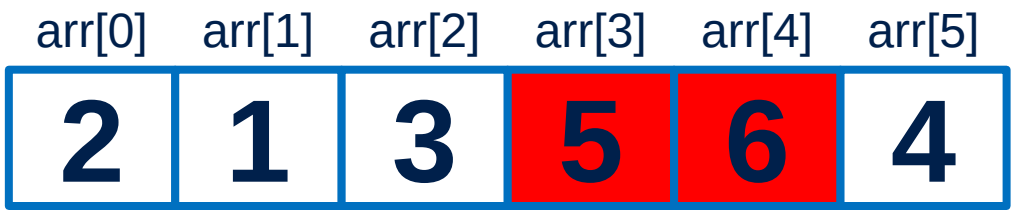
Da 6 größer als 5 ist, muss auch hier getauscht werden:



Bubblesort – Graphische Erläuterung

Das nächste Paar:

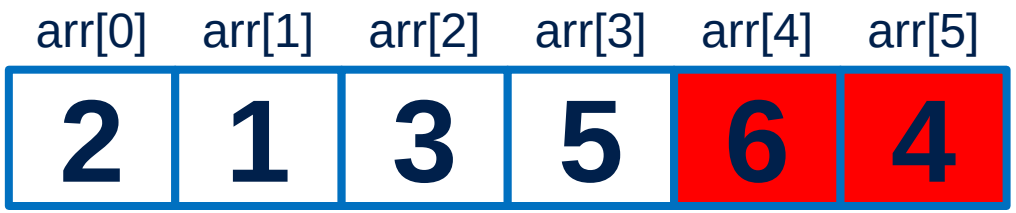
Da 6 größer als 5 ist, muss auch hier getauscht werden:



Bubblesort – Graphische Erläuterung

Das nächste Paar:

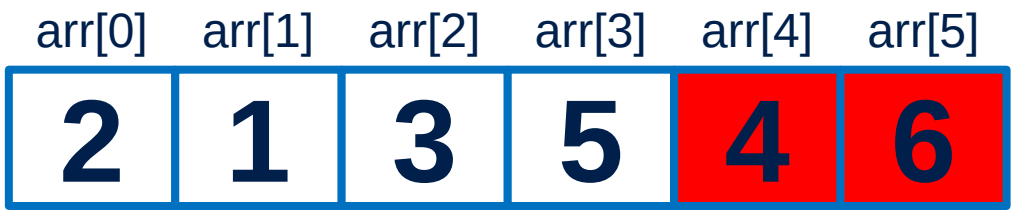
Da 6 größer als 4 ist, muss auch hier getauscht werden:



Bubblesort – Graphische Erläuterung

Das nächste Paar:

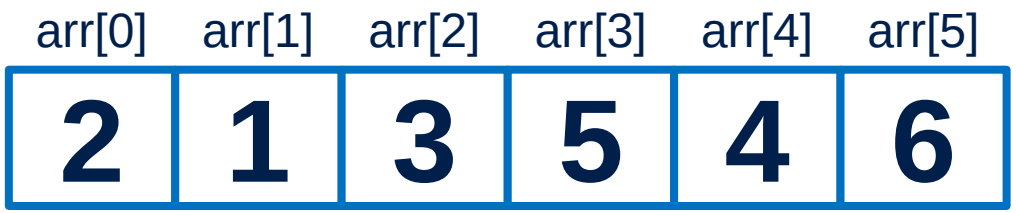
Da 6 größer als 4 ist, muss auch hier getauscht werden:



Bubblesort – Graphische Erläuterung

Die 6 ist nun (wie eine „Bubble“ in der Kohlensäure) nach oben gespült worden.

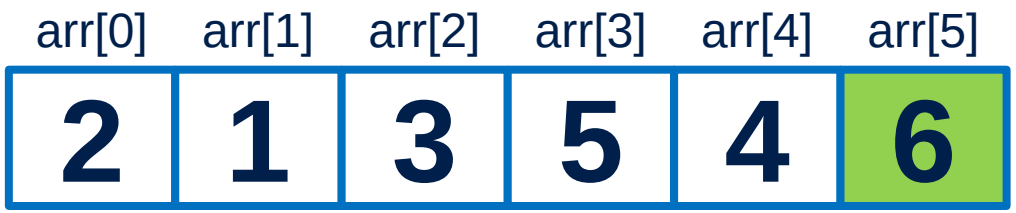
Sie hat ihre korrekte Position bereits erreicht und muss im Folgenden nicht mehr berücksichtigt werden:



Bubblesort – Graphische Erläuterung

Die 6 ist nun (wie eine „Bubble“ in der Kohlensäure) nach oben gespült worden.

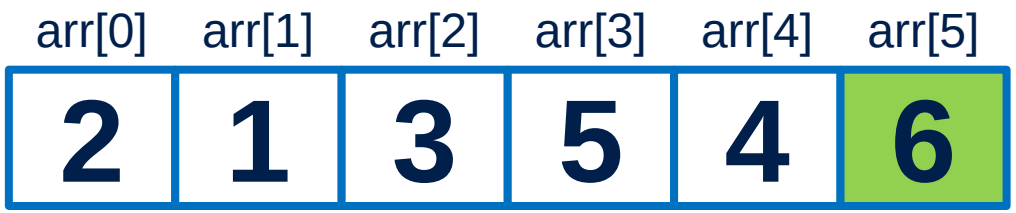
Sie hat ihre korrekte Position bereits erreicht und muss im Folgenden nicht mehr berücksichtigt werden:



Bubblesort – Graphische Erläuterung

Der nun schon bekannte Algorithmus beginnt von vorne.

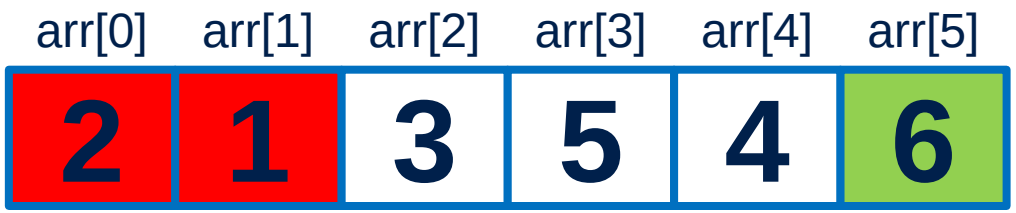
Erneut werden zwei unmittelbar aufeinanderfolgende Felder betrachtet:



Bubblesort – Graphische Erläuterung

Der nun schon bekannte Algorithmus beginnt von vorne.

Erneut werden zwei unmittelbar aufeinanderfolgende Felder betrachtet:

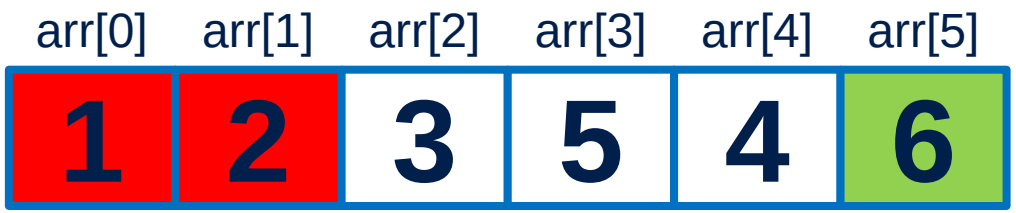


Tauschbedarf

Bubblesort – Graphische Erläuterung

Der nun schon bekannte Algorithmus beginnt von vorne.

Erneut werden zwei unmittelbar aufeinanderfolgende Felder betrachtet:

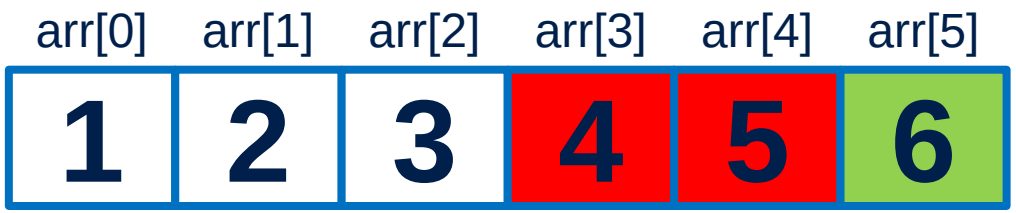


Bubblesort – Graphische Erläuterung

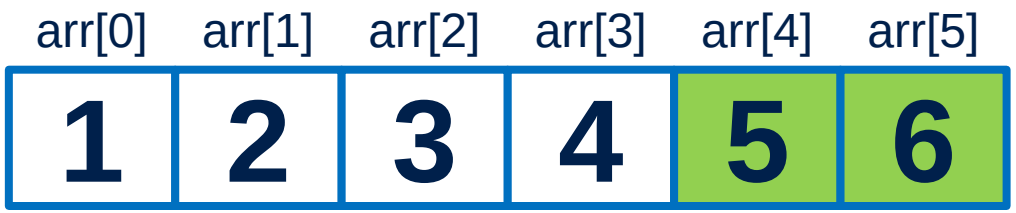
Bubblesort – Graphische Erläuterung

Bubblesort – Graphische Erläuterung

Bubblesort – Graphische Erläuterung



Bubblesort – Graphische Erläuterung

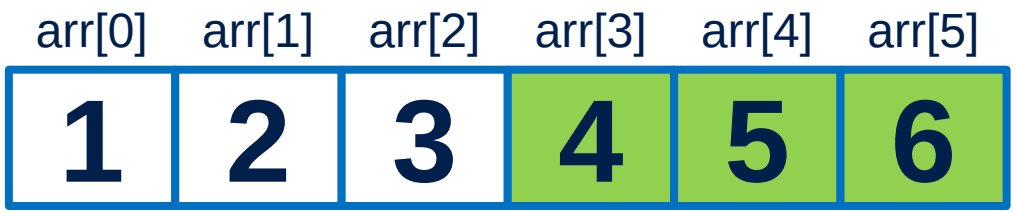


Bubblesort – Graphische Erläuterung

Bubblesort – Graphische Erläuterung

Bubblesort – Graphische Erläuterung

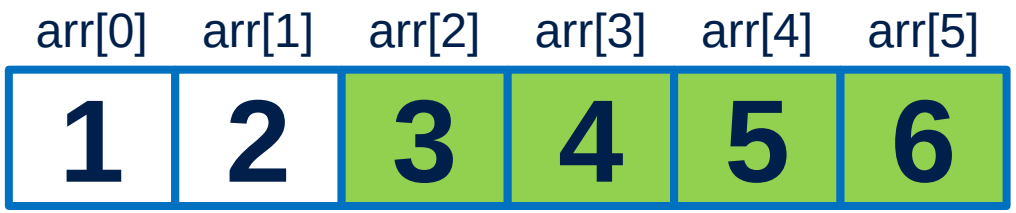
Bubblesort – Graphische Erläuterung



Bubblesort – Graphische Erläuterung

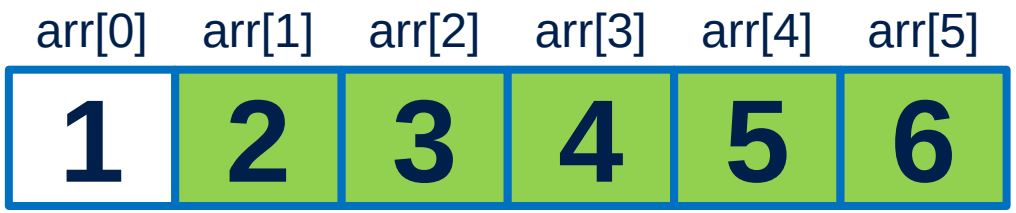
Bubblesort – Graphische Erläuterung

Bubblesort – Graphische Erläuterung

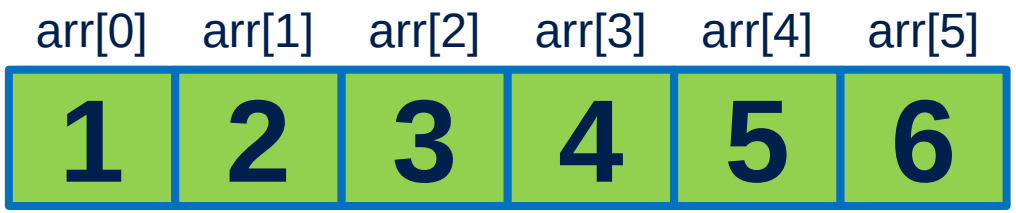


Bubblesort – Graphische Erläuterung

Bubblesort – Graphische Erläuterung



Bubblesort – Graphische Erläuterung



Bubblesort – Pseudocode

```
bubblesort(arr[ ], anzahlFelder)
{
        für(i=anzahlFelder; i>1; i=i-1)
        {
                für(j=0; j<i-1;j=j+1)
                {
                        wenn(arr[j]>arr[j+1])
                        {
                                hilfe=arr[j]
                                arr[j]=arr[j+1]
                                arr[j+1]=hilfe
                        }
                }
        }
}
```

Quicksort – Graphische Erläuterung (siehe Ordner Hilfsdokumente -> quicksort_Step_by_Step_Linkes_pivot)

Quicksort – **Pseudocode**

Quicksort – Pseudocode

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

quicksort ruft sich selbst auf, arbeitet also **rekursiv**.

Quicksort – Pseudocode

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Die Rekursion bricht ab, wenn das betrachtete Teilarray **weniger als 2 Felder** besitzt, die linke Grenze also nicht mehr echt kleiner als die rechte Grenze ist.

Quicksort – Pseudocode

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Die Funktion **sortiere** schauen wir uns im Folgenden noch genauer an.

Sie wählt ein Pivotelemet aus und sorgt dafür, dass die kleineren Werte des Arrays links vom Pivotelement und die größeren rechts vom Pivotelement platziert werden.

Ihr **Rückgabewert** ist die (endgültige) Position des Pivotelementes.

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Das Pivotelement teilt das Array in 2 Teilarrays.
Für beide kann erneut der **quicksort** gestartet werden.

Das **linke Teilarray** liegt zwischen einschließlich linker Grenze und ausschließlich Pivotelement.

Das **rechte Teilarray** liegt zwischen ausschließlich Pivotelement und einschließlich rechter Grenze.

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – Pseudocode

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

```
sortiere(array, links, rechts)
{
    pivot=array[links]  // erstes Element
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – Pseudocode

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1 // erstes Element hinter Pivotelement
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

Quicksort – Pseudocode

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts // letztes Element

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

i startet im Array von links und wird schrittweise um 1 erhöht, **j** startet von rechts und wird jeweils um 1 verringert. Wenn **i** nicht mehr kleiner als **j**, dann wurden alle Felder des Arrays (außer dem Pivotelement) abgearbeitet.

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Solange **i<rechts**, solange befindet sich i im zu sortierenden Array.

Solange **array[i]<pivot**, solange gibt es keinen Handlungsbedarf, da array[i] bereits im richtigen Teilarray liegt.

Daher kann mit **i++** einfach weitergezählt werden.

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Solange **j>links**, solange befindet sich j im zu sortierenden Array.

Solange **array[i]>=pivot**, solange gibt es keinen Handlungsbedarf, da array[i] bereits im richtigen Teilarray liegt.

Daher kann mit **j--** einfach heruntergezählt werden.

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Nur wenn die beiden vorangegangen Schleifen verlassen wurden, ist …

a) … an der Stelle **i** ein Element gefunden worden, das sich im linken Teilarray befindet, aber in das rechte gehört, und

b) … an der Stelle **j** ein Element gefunden worden, das sich im rechten Teilarray befindet, aber in das linke gehört.

Diese werden dann entsprechend **getauscht**.

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Nach der verschachtelten Schleife gibt es 2 Möglichkeiten:

a) j ist der größte Index aller Felder, die kleiner als das Pivot-Element sind
   => array[j] und pivot müssen getauscht werden, damit das Pivot-Element rechts von allen kleineren Werten liegt (und links von allen größeren)
   => j ist nun der Index des Pivot-Elementes

b) j ist bereits der Index des Pivot-Elementes
   => array[j] und pivot sind identisch und müssen nicht getauscht werden
   (dieser Fall tritt nur ein, wenn kein Arrayfeld kleiner als das Pivot-Element ist)

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

WBS TRAINING

Quicksort – **Pseudocode**

```
quicksort(array,links,rechts)
{
        wenn(links<rechts)
        {
                pivotEndposition=sortiere(array,links,rechts)
                quicksort(array, links, pivotEndposition-1)
                quicksort(array, pivotEndposition+1,rechts)
        }
}
```

Nach der verschachtelten Schleife gibt es 2 Möglichkeiten:

a) j ist der größte Index aller Felder, die kleiner als das Pivot-Element sind
   => array[j] und pivot müssen getauscht werden, damit das Pivot-Element rechts von allen kleineren Werten liegt (und links von allen größeren)
   => **j** ist nun der **Index des Pivot-Elementes**

b) **j** ist bereits der **Index des Pivot-Elementes**
   => array[j] und pivot sind identisch und müssen nicht getauscht werden
   (dieser Fall tritt nur ein, wenn kein Arrayfeld kleiner als das Pivot-Element ist)

```
sortiere(array, links, rechts)
{
    pivot=array[links]
    i=links+1
    j=rechts

    solange(i < j)
    {
        solange(i<rechts UND array[i]<pivot)
        {
            i++
        }

        solange(j>links UND array[j]>=pivot)
        {
            j--
        }

        wenn(i < j)
        {
            tausche array[i] mit array[j]
        }
    }

    wenn(array[j]<pivot)
    {
        tausche array[j] mit array[links]
    }

    return j
}
```

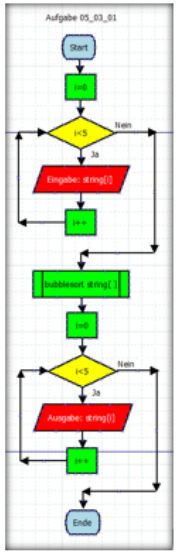
WBS TRAINING

Sortieralgorithmen – Gemeinsame Übung A_05_03_01

WBS TRAINING

Aufgabe_05_03_01

Gegeben sei der folgende PAP:



Aufgabenstellung:
Bitte erstellen Sie dazu einen geeigneten **Quellcode** in ANSI C.

WBS TRAINING

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!