Sortieralgorithmen

Sortieren durch Einfügen:

- Vergleichbar mit einem Kartenspiel, von dem immer eine Karte aufgenommen wird, die dann an die entsprechende Stelle eingefügt wird
- Pro Zug wird mehrfach verglichen, ob die einzufügende Karte größer oder kleiner als die vorhandene ist

Beispiel:

Karte 1: Ass	→ wird direkt aufgenommen
Karte 2: Fünf	→ wird mit Karte 1 verglichen und dahinter gesteckt
Karte 3: Sieben	→ wird erst mit Karte 1, dann mit Karte 2 verglichen und eingefügt

Prinzip der Sortierung:

Übertragen auf ein Array F, welches alle Karten enthält, ergibt sich je nach der Menge der sortierten Karten ein sortierter und ein nicht sortierter Bereich.
Läuft die Sortierung, wird immer ein Element aus dem nicht sortierten Bereich in den Speicher übernommen.
Danach wird der sortierte Bereich um eine Stelle des nicht sortierten Bereichs erweitert (Stelle des Wertes im Speicher).
Nun wird der Wert im Speicher mit dem (größten) Wert des sortieren Bereichs verglichen.
Ist der gespeicherte Wert größer, bleibt er an seiner Position und der Durchlauf ist beendet.
Ist er kleiner als der (größte) Wert des sortierten Bereichs, wird jener Wert um eine Stelle nach rechts kopiert. Er ist temporär also doppelt vorhanden. Jener Vorgang wird solange wiederholt, bis der Wert im Speicher größer ist, als der zu vergleichende Wert im sortierten Bereich oder die linke Grenze des sortierten Bereichs erreicht ist.

Algorithmus: Sortieren durch Einfügen

```
function Insert Sort (F,N)
     for i = 2 to N do
          me = F[i]       //einzusortierendes Element
          j = i-1         //j ist Index im sortierten Teil von F
          while (j > 0)UND(F[j] > me) do
               F[j+1] = F[j]
               j = j-1
               F[j+1] = me
          od
     od
END
```

Erläuterung:

- F = zu durchsuchendes Feld
- N = Anzahl der Feld- Elemente
- me = Merker (Speicher)
- while Schleife: durchläuft den sortierten Bereich solange, bis das Ende noch nicht erreicht ist und das einzusortierende Element (me) vor einem Element aus dem sortierten Bereich gehört
- Anweisung `F[j+1] = F[j]`: Verschiebe Element, das nach dem einzusortierenden Element kommt nach rechts

<u>Ablauf des Algorithmus</u>

Feld F:

5	1	8	3	9	2

Algorithmus Abfolge

```
i = 2
me = F[i] = 1
j = i-1 = 1
F[j] = 5 > me = 1
F[j+1]=F[j]
j = j-1 = 0
```

Feldreihenfolge

	5	5	8	3	9	2
	j	i				

me = 1

```
F[j+1] = me = 1
```

	1	5	8	3	9	2
j		i				

```
i = 3
me = F[i] = 8
j = i-1 = 2
F[j] = 5 < me = 8

F[j+1] = me = 8
```

	1	5	8	3	9	2
		j	i			

me = 8

```
i = 4
me = F[i] = 3
j = i-1 = 3
F[j] = 8 > me = 3
F[j+1]=F[j]
j = j-1 = 2
F[j] = 5 > me = 3
F[j+1]=F[j]
j = j-1 = 1
F[j] = 1 < me = 3

F[j+1] = me = 3
```

	1	5	8	8	9	2
			j	i		

me = 3

	1	5	5	8	9	2
		j		i		

me = 3

	1	3	5	8	9	2
	j			i		

```
i = 5
me = F[i] = 9
j = i-1 = 4
F[j] = 8 < me = 9
```

	1	3	5	8	9	2
				j	i	

me = 9

```
F[j+1] = me = 9
```

```
i = 6
me = F[i] = 2
j = i-1 = 5
F[j] = 9 > me = 2
F[j+1]=F[j]
j = j-1 = 4
F[j] = 8 > me = 2
F[j+1]=F[j]
j = j-1 = 3
F[j] = 5 > me = 2
F[j+1]=F[j]
j = i-1 = 2
F[j] = 3 > me = 2
F[j+1]=F[j]
j = j-1 = 1
F[j] = 1 < me = 2
```

	1	3	5	8	9	9
					j	i

me = 2

	1	3	5	8	8	9
				j		i

me = 2

	1	3	5	5	8	9
			j			i

me = 2

	1	3	3	5	8	9
		j				i

me = 2

```
F[j+1] = me = 2
```

	1	2	3	5	8	9
	j					i

Bemerkung: Die dicke Linie in den Tabellen teilt jeweils sortierten und nicht sortierten Bereich.

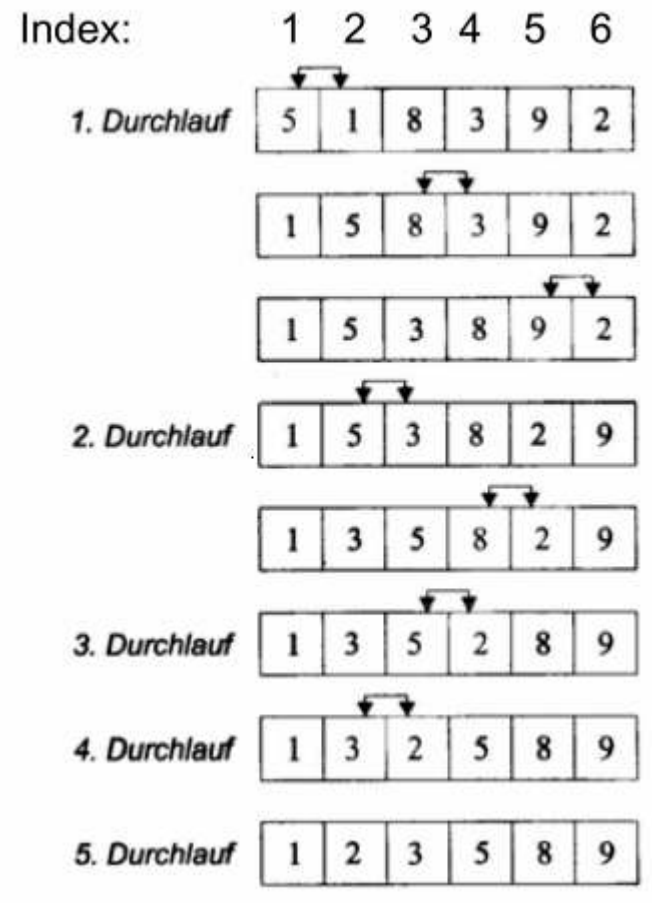
worst-case Abschätzung: $(N-1)\cdot(N-2) \approx N^2$

<u>Bubble-Sort</u>

Prinzip:

- Das Feld wird immer wieder durchlaufen
- Bei jedem Durchlauf werden benachbarte Elemente die nicht der Durchlauf werden benachbarte Elemente, die nicht der Sortierreihenfolge entsprechen, vertauscht
- Der Vorgang ist beendet, wenn keine Elemente mehr zu vertauschen sind

Ablauf des Sortiervorgangs:



Algorithmus: Bubble-Sort

```
function Bubble Sort (F,N)
     do
          vertauscht = 0
          for i = 1 to (N-1) do
               if (F[i] > F[i+1]) then
                    me = F[i]
                    F[i] = F[i+1]
                    F[i+1] = me
                    vertauscht = 1
               fi
          od
          N = N-1
     od
END
```

Erläuterung:
F = zu durchsuchendes Feld
N = Anzahl der Feldelemente
vertauscht = merke, ob vertauscht wurde

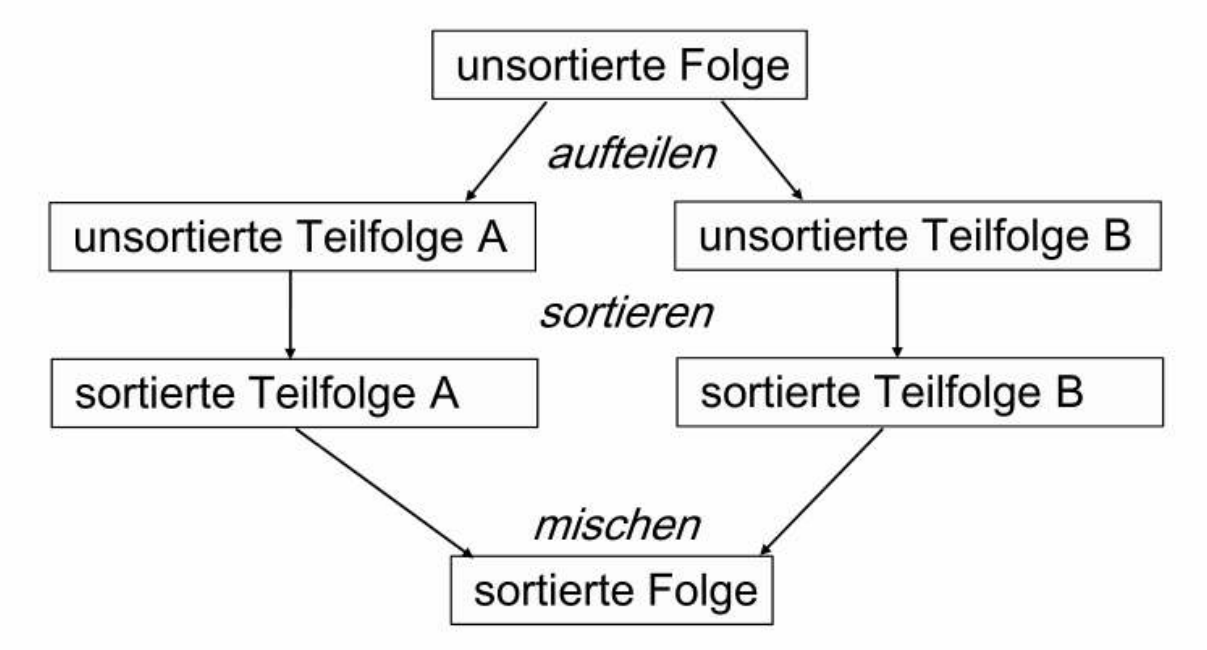
Wichtige Eigenschaft:
Bei jedem Durchlauf der for-Schleife wird das jeweils größte Element an das Ende bewegt. Deshalb braucht der nächste Durchlauf dieses Element nicht mehr zu betrachten. Es brauchen beim nächsten Durchlauf nur noch N-1 Elemente betrachtet zu werden.

worst-case Abschätzung: $\approx N^2$

Rekursives Sortieren (Merge-Sort)

Prinzip:

- Das zu sortierende Feld wird in zwei Hälften aufgeteilt.
- Beide Hälften werden durch rekursive Anwendung von Merge-Sort sortiert und anschließend gemischt.
- Der rekursive Aufruf endet, wenn ein Feld zu sortieren ist, das nur aus einem einzigen Element besteht (ein Feld mit nur einem Element ist bereits sortiert)



Algorithmus: Merge-Sort (Erster Teil)

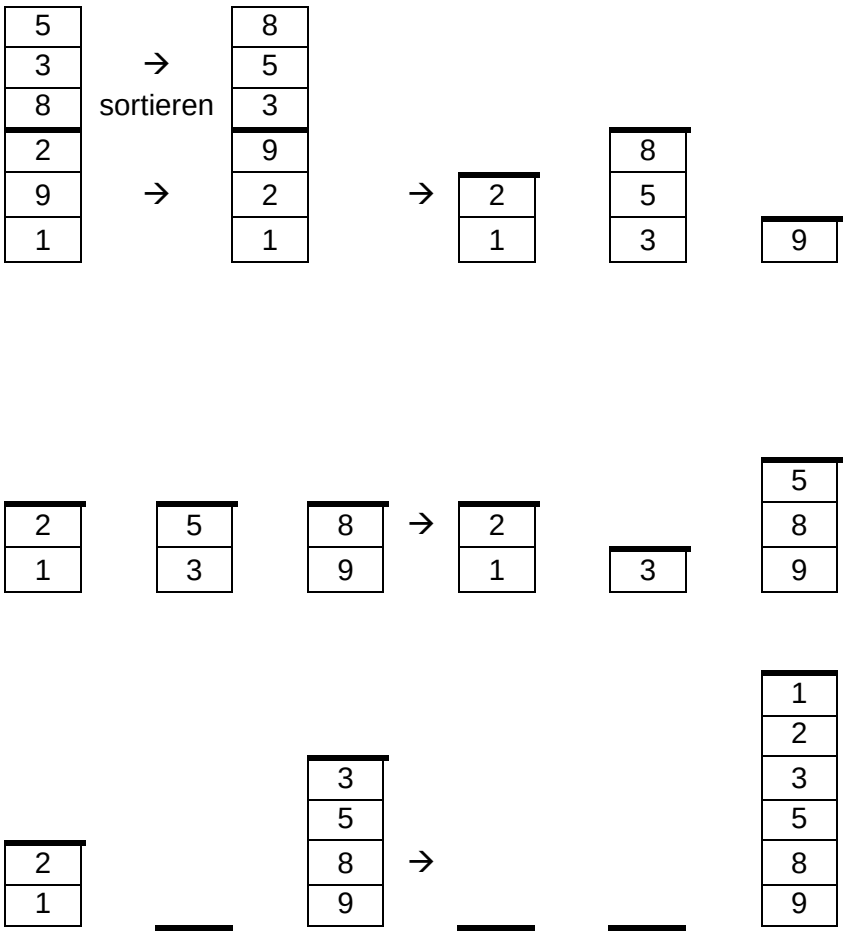
```
function Merge Sort (F, Li, Re)
     if (Li < Re) then          //mehr als ein Element zu sortieren
          mitte = (Li + Re)/2
          Merge Sort (F, Li, mitte)
          Merge Sort (F, mitte+1, Re)

          Merge(F, Li, mitte, Re)    //Misch-Funktion
     fi
END
```

Erläuterung:

- F = zu durchsuchendes Feld
- Li = linke Indexgrenze, bis zu der sortiert werden soll
- Re = rechte Indexgrenze, bis zu der sortiert werden soll

Beispiel: Zwei Kartenstapel (Mischfunktion)



Bemerkung: Im letzten Schritt kann der gesamte Stapel aufleget werden, weil er bereits sortiert ist und somit keine Mischung mehr notwendig ist.