

MergeSort (Fortsetzung)

Beispiel für einen Funktionsablauf:

F=

10	5	7	8	3	1
----	---	---	---	---	---

```
function Mergesort (F, Li(1), Re(6))
    mitte = 3
    Mergesort (F,1,3) → 

|   |   |    |   |   |   |
|---|---|----|---|---|---|
| 5 | 7 | 10 | 8 | 3 | 1 |
|---|---|----|---|---|---|


    Mergesort (F,4,6) → 

|   |   |    |   |   |   |
|---|---|----|---|---|---|
| 5 | 7 | 10 | 1 | 3 | 8 |
|---|---|----|---|---|---|


    Merge (F, Li(1), mitte(3), Re(6))
fi
END
```

Funktion Merge

```
function Merge (F, Li, mitte, Re)
    nL = mitte - Li + 1
    nR = Re - mitte
    Erzeuge Feld L mit (nL + 1) Elementen
    Erzeuge Feld R mit (nR + 1) Elementen
    for i = 1 to nL do
        L[i] = F[Li + i - 1]
    od
    for i = 1 to nR do
        R[i] = F[Mitte + i]
    od
    L[nL + 1] = ∞
    R[nR + 1] = ∞
    i = 1
    j = 1
    for K = Li to Re do
        if L[i] ≤ R[j] then
            F[K] = L[i]
            i = i + 1
        else
            F[K] = R[j]
            j = j + 1
        fi
    od
    Lösche Feld L
    Lösche Feld R
END
```

Variablen:

L = Hilfsarray zur Speicherung
des linken Teilfeldes

R = Hilfsarray zur Speicherung
des rechten Teilfeldes

nL = Anzahl der Elemente in L

nR = Anzahl der Elemente in R

L =

5	7	10	∞
---	---	----	---

R =

1	3	8	∞
---	---	---	---

Beispiel: Funktionsdurchlauf mit einem 4-Feld Array

F =

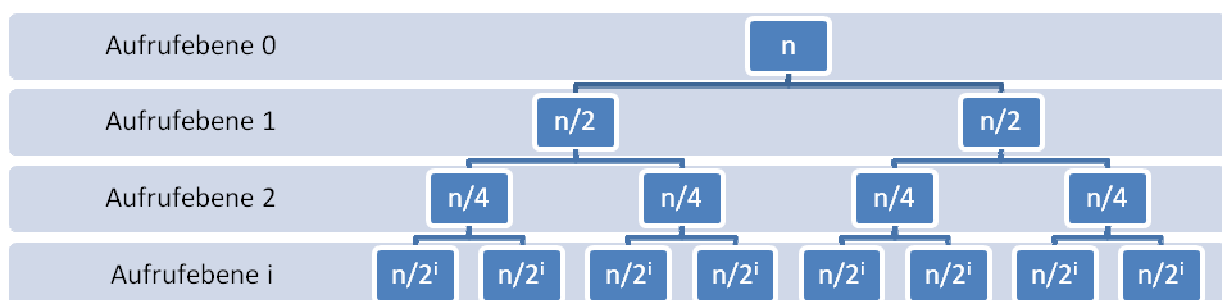
3	8	2	5
---	---	---	---

Anmerkung: Mergesort = mSort

```
mSort (F, 1, 4)
  mitte = (1 + 4)/2 = 2
  mSort (F, 1, 2)
    mitte = 1
    mSort (F, 1, 1)
    mSort (F, 2, 2)
    Merge (F, 1, 1, 2)
  mSort (F, 3, 4)
    mitte = 3
    mSort (F, 3, 3)
    mSort (F, 4, 4)
    Merge (F, 3, 3, 4)
  Merge (F, 1, 2, 4)
END
```

Aufwandsabschätzung MergeSort

Mit jedem Aufruf wird eine Teilfolge von n Elementen sortiert. Dies erfolgt durch zwei rekursive Aufrufe mit jeweils $n/2$ Elementen.



Die letzte Aufrufebene hat nur noch ein Element zu sortieren:

$$\frac{n}{2^i} = 1 \Rightarrow i = \log_2(n) \text{ ist die Anzahl der Aufrufebenen}$$

Anzahl der Vergleiche auf jeder Ebene (finden innerhalb der Schleife for K=Li to Re do statt)

Aufsummieren der Vergleiche in den einzelnen Ebenen. Diese entsprechen der Anzahl der Elemente der Ebenen (s. Grafik oben).

$$\text{Ergebnis: } \approx n \cdot \log_2(n)$$

Vergleich der Sortierverfahren in Bezug auf die Aufwandsabschätzung

N = Anzahl Elemente

BS = BubbleSort

IS = InsertSort

MS = MergeSort

N	BS und IS	MS
2	4	2
50	2.500	282
100	10.000	664

Man sieht deutlich, dass MergeSort immer eine bessere Abschätzung hat, als Bubble oder InsertSort.

QuickSort

Prinzip:

- Eine Folge wird in zwei Teilfolgen geteilt (Referenzelement ist die Mitte)
- Eine Folge enthält nur Elemente, die kleiner als das Referenzelement sind
- Eine Folge enthält nur Elemente, die größer als das Referenzelement sind
- Merge wird vermieden
- Problem: Wird das äußerste Element als Referenz gewählt, hat ein Stapel keine Werte (Verbesserung jenes Problems ist per stochastischer Abschätzung möglich)

Beispiel:

