Suche in Folgen

- Folge = Feld (Array)
- Schreibweise F[i] (i = Arrayindex)
- Nummerierung von 1 bis n (bei PHP: von 0 bis n-1)

Beispiel:

SUCHE (F, Key, n) → Suche nach dem Index i

F = Feld (Array)
Key = Feldelement, das gesucht wird
n = Größe des Arrays
Ergebnis = Index i vom Key (falls nichts gefunden wird, dann ist i = n-1)

1. Sequentielle Suche

```
function i = SequentiellSearch (F, n, Key)
  i = 1
  do while (i <= n)UND(F[i]< >key)
       i = i + 1
  od
End
```

i wird solange um 1 erhöht, bis i > n oder F[i] = Key ist.

Bewertung des Verfahrens

Wichtiges Kriterium: Aufwand, d.h. die Anzahl der notwendigen Schritte, um das Ergebnis zu finden. Häufig wird hierzu die Anzahl der Vergleiche oder Feld(Array)zugriffe herangezogen.

Meist erfolgt eine Fallunterscheidung, in der man:

- den besten Fall: 1
- den schlechtesten Fall: n
- dem Durchschnitt: $\sum_{i=1}^{n} i \cdot \frac{1}{n} = \frac{n+1}{2}$ (Erwartungswert, alle Elemente werden mit gleicher Wahrscheinlichkeit gesucht)

2. Binäre Suche

Suche über den mittleren Index (→ bei Sortierten Elementen)

Prinzip:

- Es wird in Grenzen gesucht (U = untere Grenze, O = obere Grenze)
- Pro Suchdurchlauf werden die Hälfte der Indexe ausgeschlossen
- Gesamte Teilfolge → mittleres Element finden über Ganzzahldivision $(U+O)/2$ → $\frac{n}{2}$ Elemente ausschließen → Vorgang widerholen, bis Key gefunden oder nicht

Bemerkung

- Interne Variblen:
  - u = untere Grenze des Suchbereichs
  - o = obere Grenze des Suchbereichs
  - gefunden = Merker, ob Element gefunden wurde (0 oder 1)

Im Pseudocode

```
function i = BinarySearch (F, n, Key)
      gefunden = 0
      u = 1
      o = n
      do while (gefunden = 0)UND(U <= 0)
            i = (u + o)/2              //Wähle Mitte
            if F[i] = Key then
                  gefunden = 1
            else
                  if F[i] > Key then
                        o = i - 1
                  else
                        u = i + 1
                  fi
            fi
      od
      if gefunden = 0 then
            i = n + 1
      fi
End
```

Beispiel 1: Suche nach einem Arrayelement

Array 1:

i	1	2	3	4	5	6	7	8
Key	3	7	10	15	21	37	40	51

```
function i = BinarySearch (F, 8, 10)
            i = (1 + 8)/2 = 4                 //F[4] = 15
            15 > 10
            o = i - 1 = 3
            i = (1 + 3)/2 = 2                 //F[2] = 7
            7 < 10
            u = i + 1 = 3
            i = (3 + 3)/2 = 3                 //F[3] = 10
            10 = 10
            od
End
```

Beispiel 2: Suche nach einem nicht vorhandenen Element (im Array 1)

```
function i = BinarySearch (F, 8, 5)
          i = (1 + 8)/2 = 4              //F[4] = 15
          15 > 10
          o = i - 1 = 3
          i = (1 + 3)/2 = 2              //F[2] = 7
          7 < 10
          u = i + 1 = 3
          i = (3 + 3)/2 = 3              //F[3] = 10
          10 > 5
          o = i - 1 = 2                  //ACHTUNG: u > o
          od
     i = n + 1
End
```

Aufwandsabschätzung für den schlechtesten Fall („worst case“) der binären Suche

- nach den ersten Vergleich müssen noch $\frac{n}{2}$ Elemente durchsucht werden
- nach den zweiten Vergleich müssen noch $\frac{n}{4}$ Elemente durchsucht werden
- nach den dritten Vergleich müssen noch $\frac{n}{8}$ Elemente durchsucht werden
- nach den x-ten Vergleich müssen noch $\frac{n}{2^x}$ Elemente durchsucht werden

➔ $\frac{n}{2^x} = 1$ → $x = \log_2(n)$

Zahlenbeispiele

Suchform	n = 10	n = 100	n = 1000
Sequentiell	10	100	1000
Binär	3,3	6,6	9,9

➔ Wird mehrfach in einer Folge gesucht, dann lohnt es, sich die Folge zu sortieren, damit die binäre Suche anwendbar ist.