

# ÜBERSICHTSBLATT: BINÄRE SUCHE

## Die Idee hinter dem Algorithmus:

Bei der binären Suche halbiert man mit jedem Vergleich das Suchintervall. Und grenzt somit den Bereich ein, in dem sich der Schlüssel befinden kann. Es ist ein verhältnismäßig einfaches und doch effektives Suchverfahren, das allerdings nur dann funktioniert, wenn man es mit sortierten Daten zu tun hat.

### Beispiel:

Auf einem Schachbrett soll die Position des Königs ermittelt werden. Man könnte nun sequenziell alle Felder (eins nach dem anderen) abfragen, oder die Suche nach dem folgenden Muster durchführen, wobei das Schachbrett durch jede Frage halbiert wird:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 8 |   |   |   |   |   |   |   |   |
| 7 |   |   |   |   |   |   |   |   |
| 6 |   |   |   |   |   |   |   |   |
| 5 |   |   |   |   |   |   |   |   |
| 4 |   |   |   |   |   |   |   |   |
| 3 |   |   | K |   |   |   |   |   |
| 2 |   |   |   |   |   |   |   |   |
| 1 |   |   |   |   |   |   |   |   |
|   | A | B | C | D | E | F | G | H |

### Suchabfragen:

|                          |                |   |      |
|--------------------------|----------------|---|------|
| Frage 1:                 | Ist er rechts? | - | NEIN |
| Frage 2:                 | Ist er unten?  | - | JA   |
| Frage 3:                 | Ist er rechts? | - | JA   |
| Frage 4:                 | Ist er unten?  | - | NEIN |
| Frage 5:                 | Ist er rechts? | - | NEIN |
| Frage 6:                 | Ist er unten?  | - | JA   |
| => König gefunden auf C3 |                |   |      |

Es waren nur 6 Vergleiche nötig um den König zu finden  
eine sequenzielle Suche hätte im schlimmsten Falle 63 Vergleiche benötigt.

## Pseudocode:

Eingabe: Feld A, Index „links“ und Index „rechts“ die den Bereich der Suche angeben, Schlüssel s

Ergebnis: Index i, so dass  $A[i] == s$  oder -1 wenn Schlüssel nicht gefunden

```
1 int search(A, links, rechts, s){
2     if(links > rechts){
3         return -1;
4     }
5     else{
6         mitte = [(links + rechts) / 2]
7         if(A[mitte] == s){ /* Schlüssel gefunden? */
8             return mitte;
9         }
10        if(A[mitte] < s){ /* Suche in der rechten Hälfte */
11            return search(A, mitte+1, rechts, s);
12        }
13        if(A[mitte] > s){ /* Suche in der linken Hälfte */
14            return search(A, links, mitte-1, s);
15        }
16    }
17 }
```

## Analyse:

Um die Laufzeit für n Elemente abzuschätzen gehen wir davon aus, dass die Anzahl n eine Zweierpotenz ist. Bei jedem rekursiven Aufruf wird die Anzahl der zu durchsuchenden Elemente halbiert. Damit bricht die Rekursion nach m Schritten ab. Im Allgemeinen braucht wir also etwa  $\log_2(n)$  Schritte um ein Element zu finden.