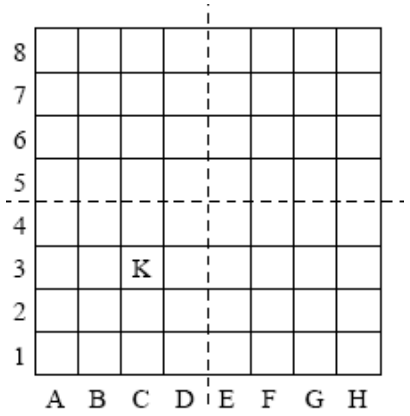
Übersichtsblatt:
Binäre Suche

Idee:

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, dass allerdings nur dann funktioniert, wenn man es mit sortierten Daten zu tun hat.



Beispiel:

Finde den König auf dem Schachbrett:

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 C 3

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

```
int search(A,links,rechts,s){
   if(links > rechts){
      return -1;
   }
   else{
      mitte = [(links + rechts) / 2]
      if(A[mitte] == s){ /* Schlüssel gefunden? */
         return mitte;
      }
      if(A[mitte] < s){  /* Suche in der rechten Hälfte */
         return search(A,mitte+1,rechts,s);
      }
      if(A[mitte] > s){  /* Suche in der linken Hälfte */
         return search(A,links,mitte-1,s);
      }
   }
}
```

Analyse:

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