Sortierverfahren

Suchen – in sortierten Folgen

Vereinfachende Annahmen:

- Folge = Feld von numerischen Werten.
- Auf jedes Element der Folge F kann über den Index i zugegriffen werden (mit F[i]). Erstes Element: F[1], letztes Element bei n Werten: F[n].
- Für die Feldelemente sind die bekannten Vergleichsoperatoren =, < und > definiert.
- Der für die Suche relevante Teil ist der numerische Wert, nach dem gesucht wird. Der gesuchte Wert ist der Suchschlüssel.

Sequenzielle Suche

- Die Folge wird sequenziell durchlaufen, beginnend beim ersten Element.
- In jedem Schritt wird das aktuelle Element mit dem Suchschlüssel verglichen.
- Sobald das gesuchte Element gefunden wurde, kann die Suche beendet werden.
- Wenn man am Ende der Folge ankommt, ohne das Suchelement gefunden zu haben, wird als Ergebnis ein spezielles Element NO_KEY zurückgegeben.

S equenzielle Suche: Algorithmus

- seqSearch (F, k) → p
  Eingabe: Folge F der Länge n, Suchschlüssel k
  Ausgabe: Position p des ersten Elements aus F, das gleich k ist, sonst NO_KEY

- ```
  for i:= 1 to n
        if F[i] = k then return i;
  oder
  return NO_KEY
  ```
  ```

Sequenzielle Suche: Implementierung

```
class seqSearch {

    public final static int NO_KEY = -1;

    static int search (int [] array, int key) {
      for (int i = 0; i < array.length; i++)
        if (array[i] == key) return i;
      return NO_KEY;
    }
```

Such-funktion

Fortsetzung

```
public static void main(String[] args) {
    int [] F = {2, 4, 5, 6, 7, 8, 9, 11};
    int key = 1;
     int res = NO_KEY ;
    int i = 0;

    print("search key = ");
    key = readInt();

    while (i < F.length && res == NO_KEY) {
      if (F[i] == key) res = i;
       i++;
     }

     println ("search result: " + res);
  }
}
```

Aufwand

- Wichtigstes Kriterium für Beurteilung von Suchverfahren: Aufwand
- Wie wird der Aufwand berechnet?
  - Notwendige Schritte mit den Vergleichen = Anzahl der Schleifendurchläufe.
  - Der Aufwand wird (sinnvoller Weise) nicht absolut betrachtet, sondern in Abhängigkeit von der Länge n der Folge.

Sequentielle Suche: Aufwand

	Anzahl der Vergleiche
bester Fall	1
schlechtester Fall	n
Durchschnitt (erfolgreiche Suche)	n/2
Durchschnitt (erfolglose Suche)	n

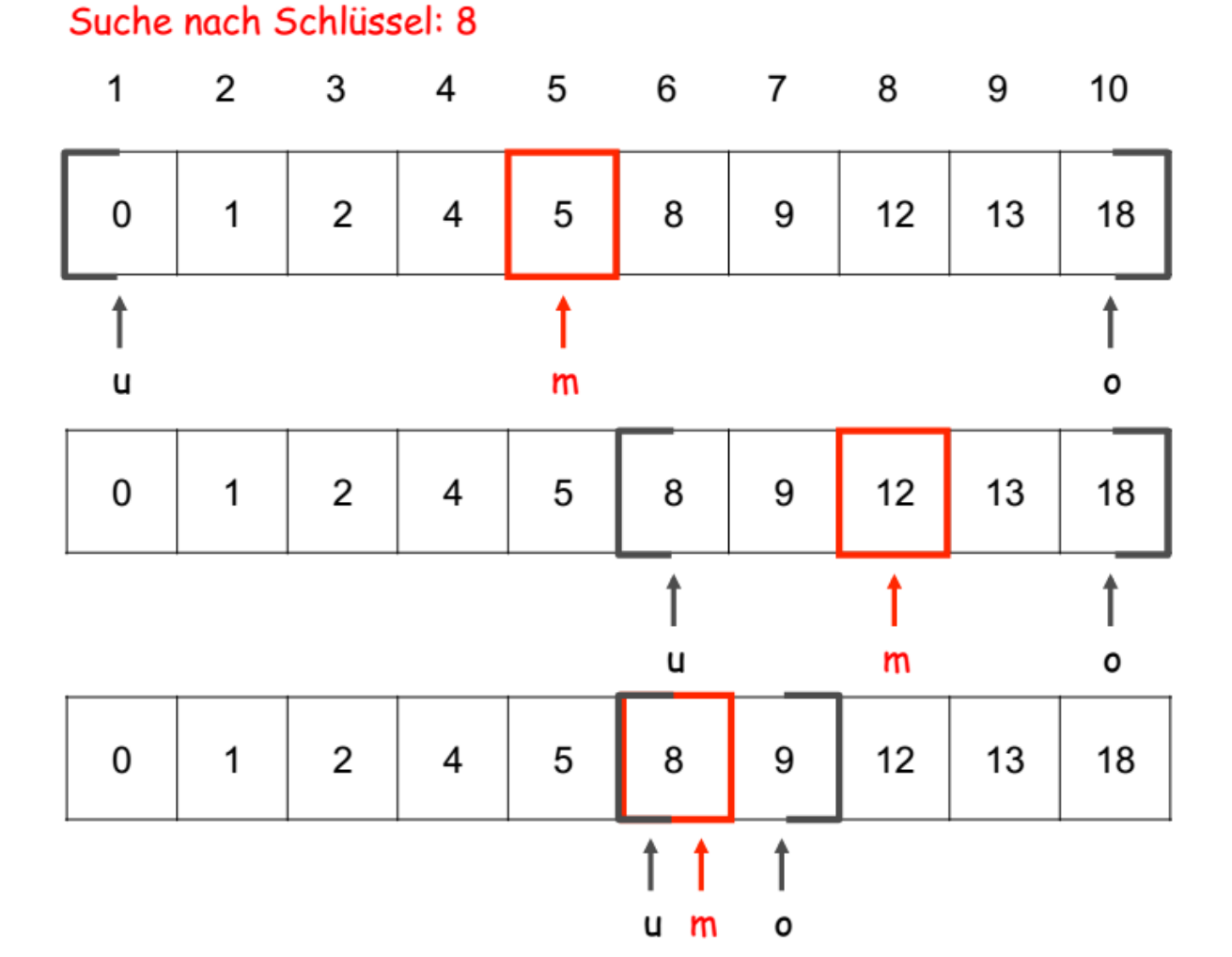
Binäre Suche: Beschreibung des Algorithmus

- Wähle den mittleren Eintrag und prüfe, ob gesuchter Wert in der ersten oder in der zweiten Hälfte der Folge ist.
- Fahre analog Schritt 1. mit der Hälfe fort, in der sich der Eintrag befindet.
- Realisierungsvarianten:
  - iterativ oder rekursiv.

Binäre Suche: Algorithmu s

```
binarySearch (F, k) → p
Eingabe:    Folge F der Länge n, Suchschlüssel k
Ausgabe:    Position p des ersten Elements aus F, das gleich k ist, sonst NO_KEY
u := 1; o := n;
while u <= o
        m := (u+o)/2;
        if F[m] = k then return m;  // gefunden !
        else if k < F[m] then
                o := m – 1;    // suche in der unteren Hälfte weiter
                else
                u := m + 1;    // suche in der oberen Hälfte weiter
                fi;
        fi;
od;
return NO_KEY
```

Binäre Suche: Beispiel



Binäre Suche: Aufwand

	Anzahl der Vergleiche
bester Fall	1
schlechtester Fall	$\approx \log_2 n$
Durchschnitt (erfolgreiche Suche)	$\approx \log_2 n$
Durchschnitt (erfolglose Suche)	$\approx \log_2 n$

Binäre Suche vs. sequenzielle Suche (im Mittel)

Anz. Elemente / Verfahren	10	100 (10^2)	1.000 (10^3)	10.000 (10^4)
sequenziell (n/2)	≈ 5	≈ 50	≈ 500	≈ 5000
binär ($\log_2$ n)	≈ 3.3	≈ 6.6	≈ 9.9	≈ 13.3

Binäre Suche: Implementierung (JAVA)

```
static int search (int [] array, int key) {
   int u = 0, o = array.length - 1;
   while (u <= o) {
     int m = (u + o) / 2;
     if (array[m] == key) return m; // gefunden !
     else if (key < array[m])
       o = m-1; // suche in der unteren Hälfte weiter
     else u = m+1; // suche in der oberen Hälfte weiter
   }
   return NO_KEY;
 }
```