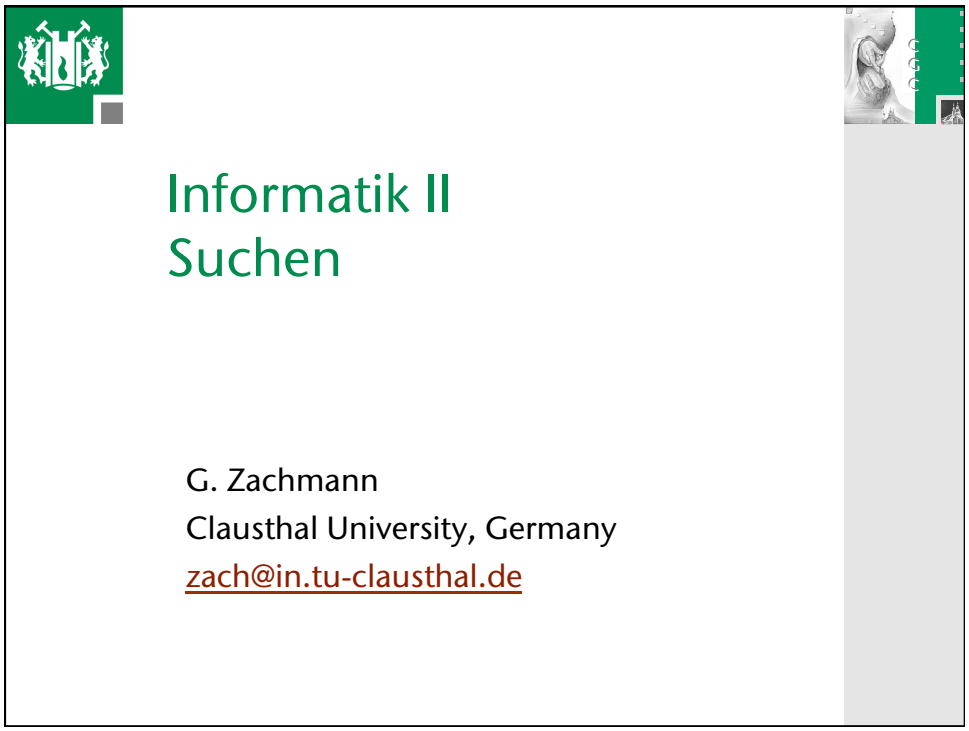
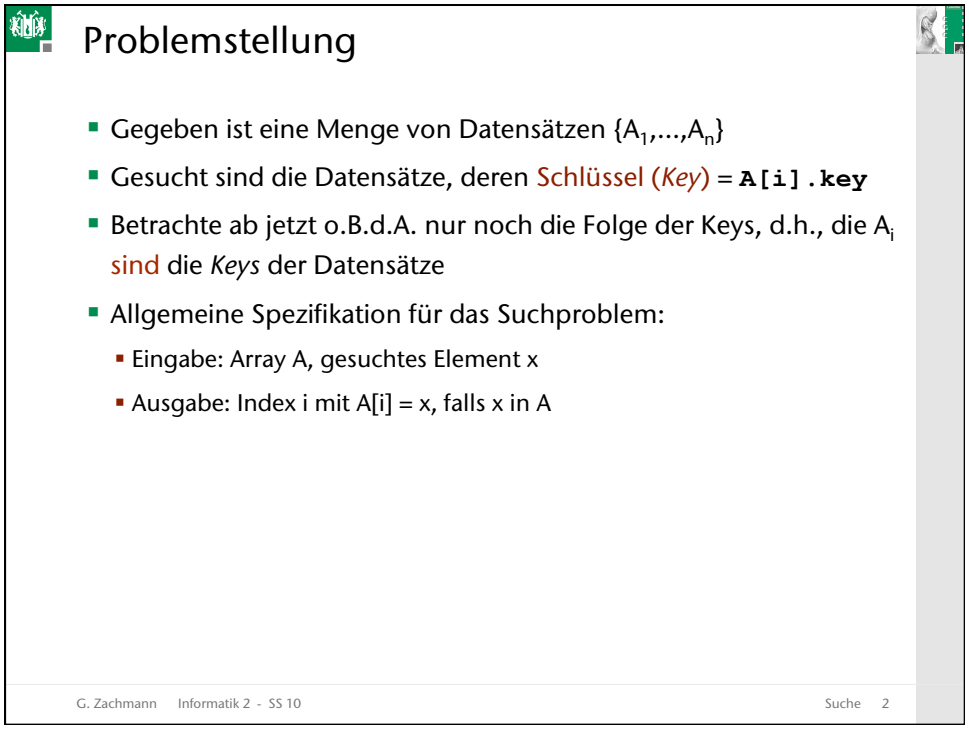


Informatik II
Suchen

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de



Problemstellung

- Gegeben ist eine Menge von Datensätzen $\{A_1,...,A_n\}$
- Gesucht sind die Datensätze, deren Schlüssel (*Key*) = **`A[i].key`**
- Betrachte ab jetzt o.B.d.A. nur noch die Folge der Keys, d.h., die A_i sind die *Keys* der Datensätze
- Allgemeine Spezifikation für das Suchproblem:
  - Eingabe: Array A, gesuchtes Element x
  - Ausgabe: Index i mit A[i] = x, falls x in A

Lineare Suche

- Wenn nichts über die (An-)Ordnung der Elemente in der Datenstruktur bekannt ist, kann nur die lineare Suche benutzt werden:
  - Besuche der Reihe nach die Elemente aus dem Container, bis ein Element mit der gewünschten Eigenschaft gefunden wurde, oder alle besucht wurden
- Beispiel: suche gegebene Telefonnummer in einem Telefonbuch
- Aufwand
  - worst case: n Schleifendurchläufe
  - average case:

$$\frac{1}{n}(1+2+\ldots+n) = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}\frac{n(n+1)}{2} \approx \frac{n}{2}$$

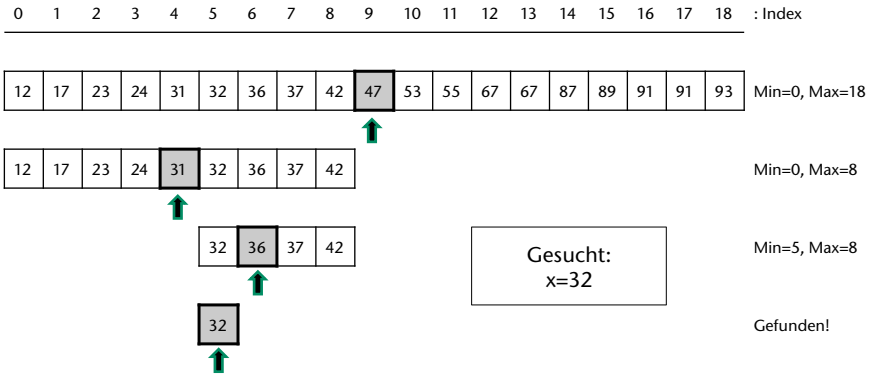
Binäre Suche

- Nutze die totale Ordnung auf den Elementen
- Annahme: die n Elemente seien nun aufsteigend sortiert (geordnet), d.h.

$$\forall i, 0 \leq i < n-1 : A_i \leq A_{i+1}$$

- Beispiel: suche nach einem Namen im Telefonbuch
- Lösungsstrategie: "Intervallhalbierung"
  - Schlage Telefonbuch in der Mitte auf, vergleiche gesuchten Namen mit einem Namen auf der Seite
  - Entscheide, in welcher Hälfte des Telefonbuches sich der gesuchte Name befindet (halbiert den Suchraum grob gerechnet)
  - Wiederhole Verfahren mit dieser Hälfte, bis richtige Seite gefunden ist

Beispiel



Der rekursive Algorithmus ...

```
def binsearch_start( A, key ):
    """ Return index i such that 0 <= i <= len(A)
        and A[i] == key, or, if no such i exists,
        returns -1. """
    if len(A) == 0:   return -1
    if len(A) == 1:
        if A[0] == key: return 0
        else:           return -1
    return binsearch_work( A, key, 0, len(A)-1 )
```

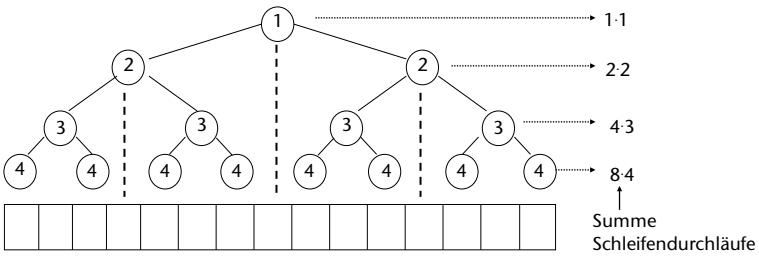
```
def binsearch_work( A, key, l, r ):
    """ The work horse for binsearch_start. """
    if r < l:
        return -1                    # k not found
    m = (l + r) / 2
    if key < A[m]:
        return binsearch_work( A, key, l, m-1 )
    if key > A[m]:
        return binsearch_work( A, key, m+1, r )
    return m
```

... und nicht-rekursiv

```
def binsearch( A, key ):
    l = 0
    r = len(A) - 1
    while l <= r:
        m = (l + r) / 2
        if key < A[m]:
            r = m - 1
        elif key > A[m]:
            l = m + 1
        else:
            return m
    return -1
```


Analyse

- Annahme: $n = 2^k - 1$
- Beispiel:



- Worst case: #Schleifendurchläufe = $k = \log(n+1) \approx \lceil \log n \rceil$
- Mittlere Anzahl Schleifendurchläufe: $\frac{1}{n}\sum_{i=1}^{k} i2^{i-1} = \frac{1}{n}((k-1)2^k + 1) \approx \log n$