



Objektorientierte Programmierung

Sortieren (Teil 4)

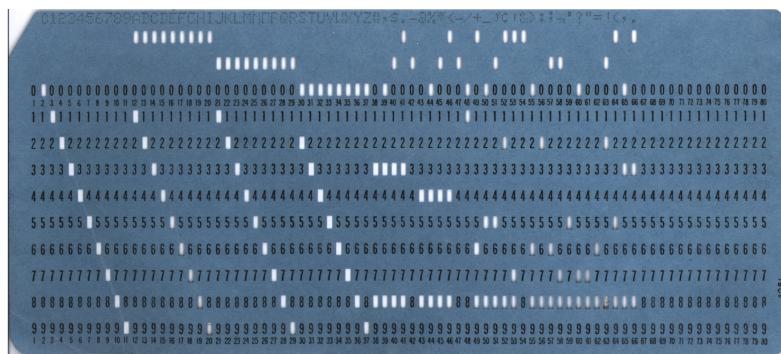
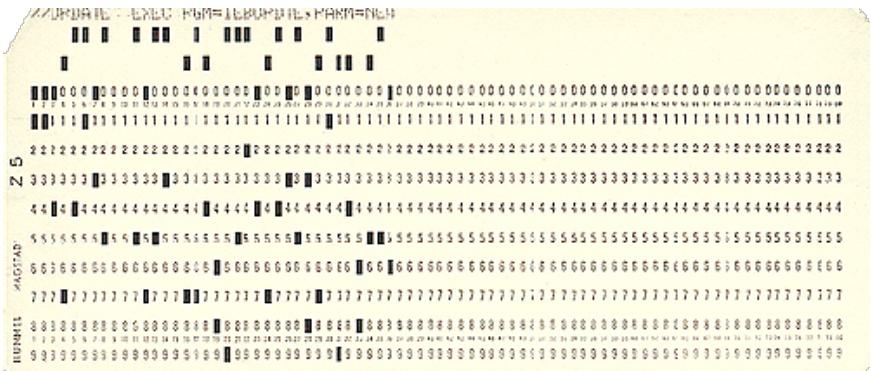
Imperativ!

SoSe 2020

Prof. Dr. Margarita Esponda



Radix sort



Der IBM Lochkartensortierer Modell 083, gebaut ab 1958.

Bildquelle: Wikipedia



Radix sort

1887 Entwickelt für das Sortieren von Lochkarten

- Zahlen werden ziffernweise sortiert
- die niedrigsten Stellenwerte zuerst
- alles nur mit einem stabilen Algorithmus (essentiell)
- auch gut für das Sortieren von zusammengesetzte Datenstrukturen
- **Beispiel:** Sortieren nach Datum



Radix sort

```
radixsort ( A, d) {  
    for i in range(d):  
        stablesort(A, i)
```



Countingsort



$T(n) = O(n)$



Bucket-Sort

- Die zu sortierenden Daten müssen **gleich verteilt** über den Wertebereich **[0,1)** sein.
- **Not-In-Place**
- linearer Aufwand **O(n)**
- zusätzlicher Speicherplatz (**O(n)**) wird benötigt

Bucket-Sort

Grundidee ist:

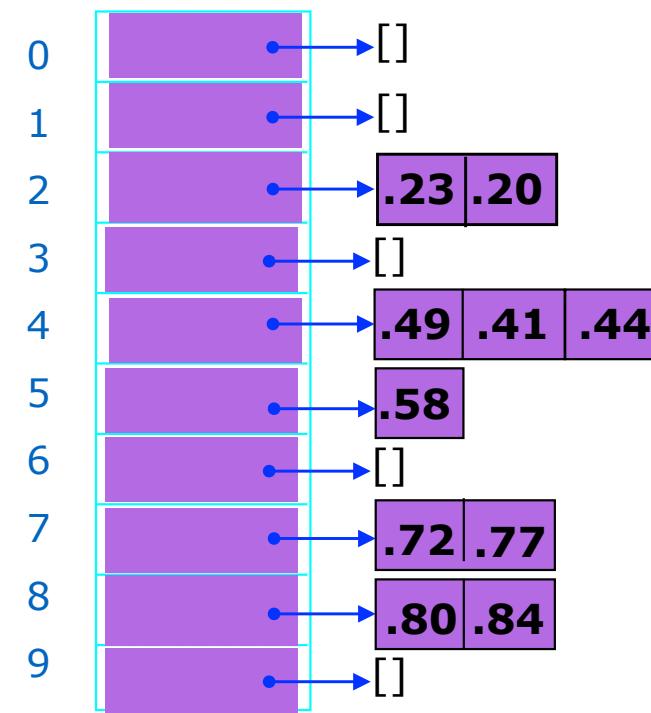
- den Wertebereich **[0,1)** in **m** kleinere Wert-Bereiche zu teilen und *Buckets* dafür zu definieren.
- Die Zahlen werden in den dazugehörigen *Buckets* verteilt und innerhalb diesen sortiert.
- Zum Schluss werden die Zahlen der Reihe nach aus den *Buckets* ausgegeben.
- Eine Hilfsarray von Listen kann für die *Buckets* verwendet werden.



Bucket-Sort

A

.58
.23
.49
.72
.77
.41
.20
.44
.80
.84

B

Bucket-Sort

```
def bucketsort (A, k):
    B = [ [] for j in range(k) ]
    n = len(A)
    for i in range(n):
        B[math.floor((A[i])*k)].append(A[i])
    for i in range(k):
        insertsort(B[i])
    R = []
    for i in range(k):
        R = R+B[i]
    return R
```

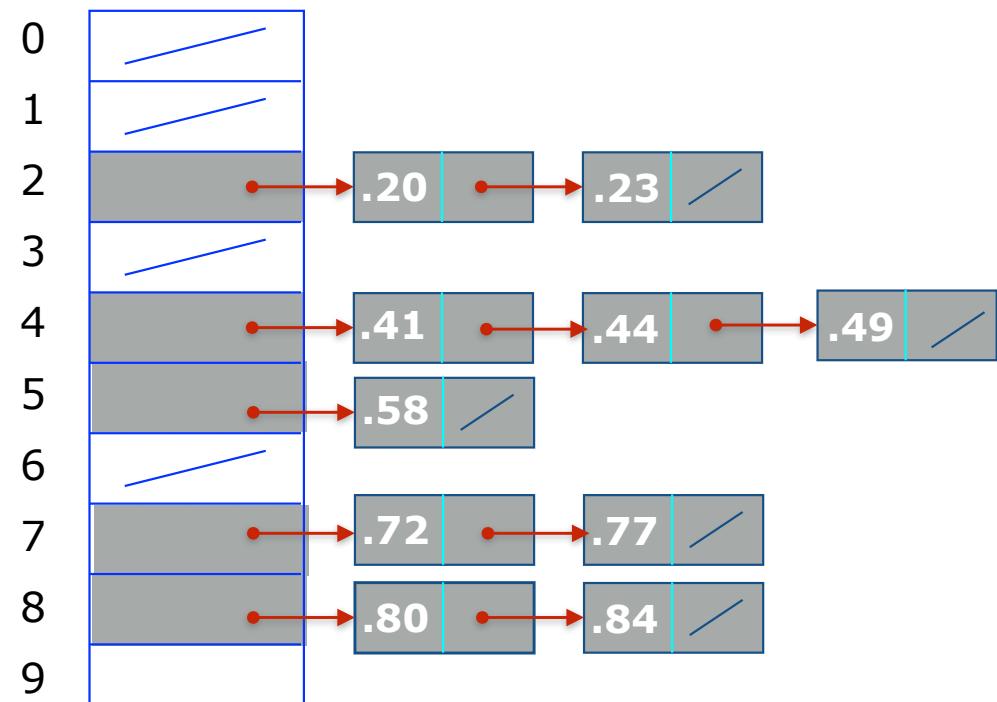
k ist z.B. gleich **10**, wenn nur eine Ziffer nach dem Koma für die Erstellung des Buckets berücksichtigt wird.



Bucket-Sort

A

.58
.23
.49
.72
.77
.41
.20
.44
.80
.84

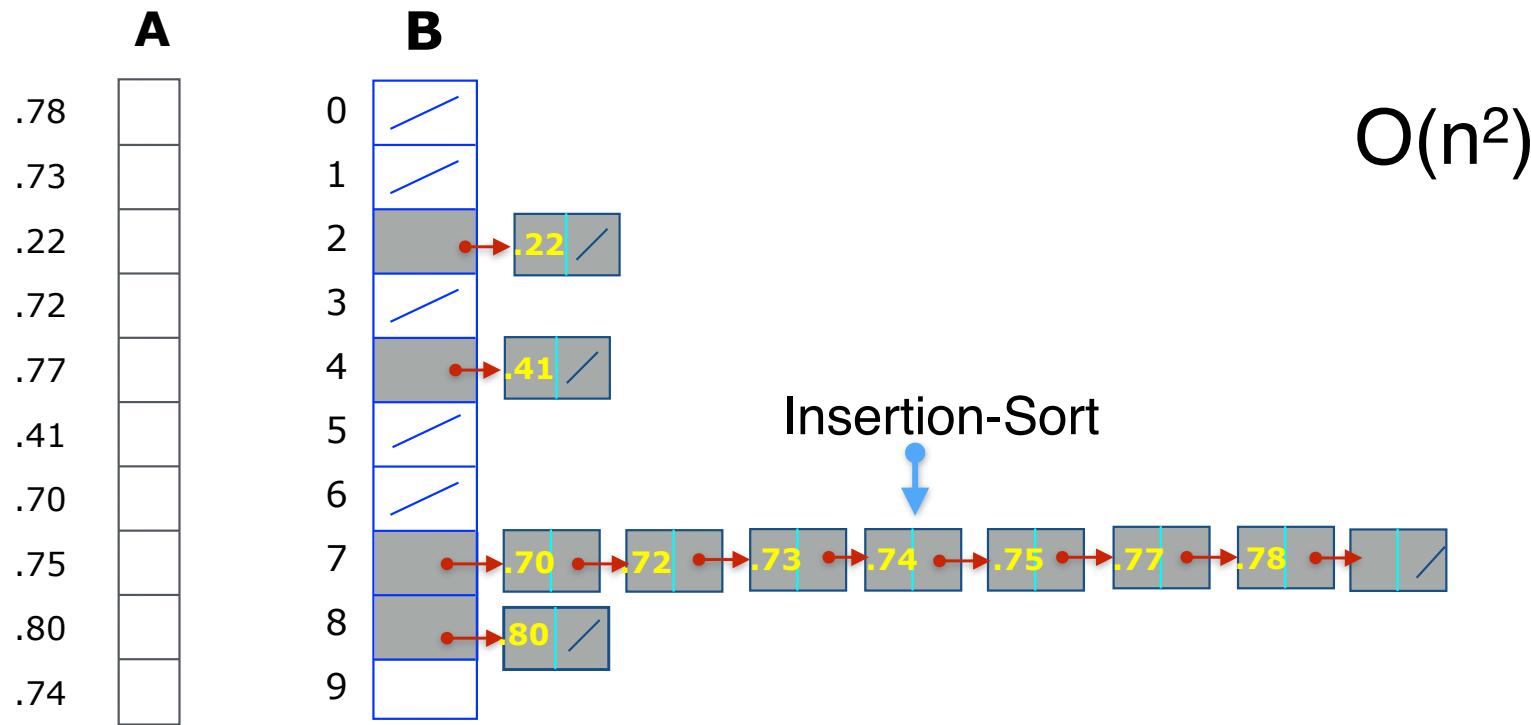
B

... mit verkettete Listen

Komplexität

Schlimmster Fall

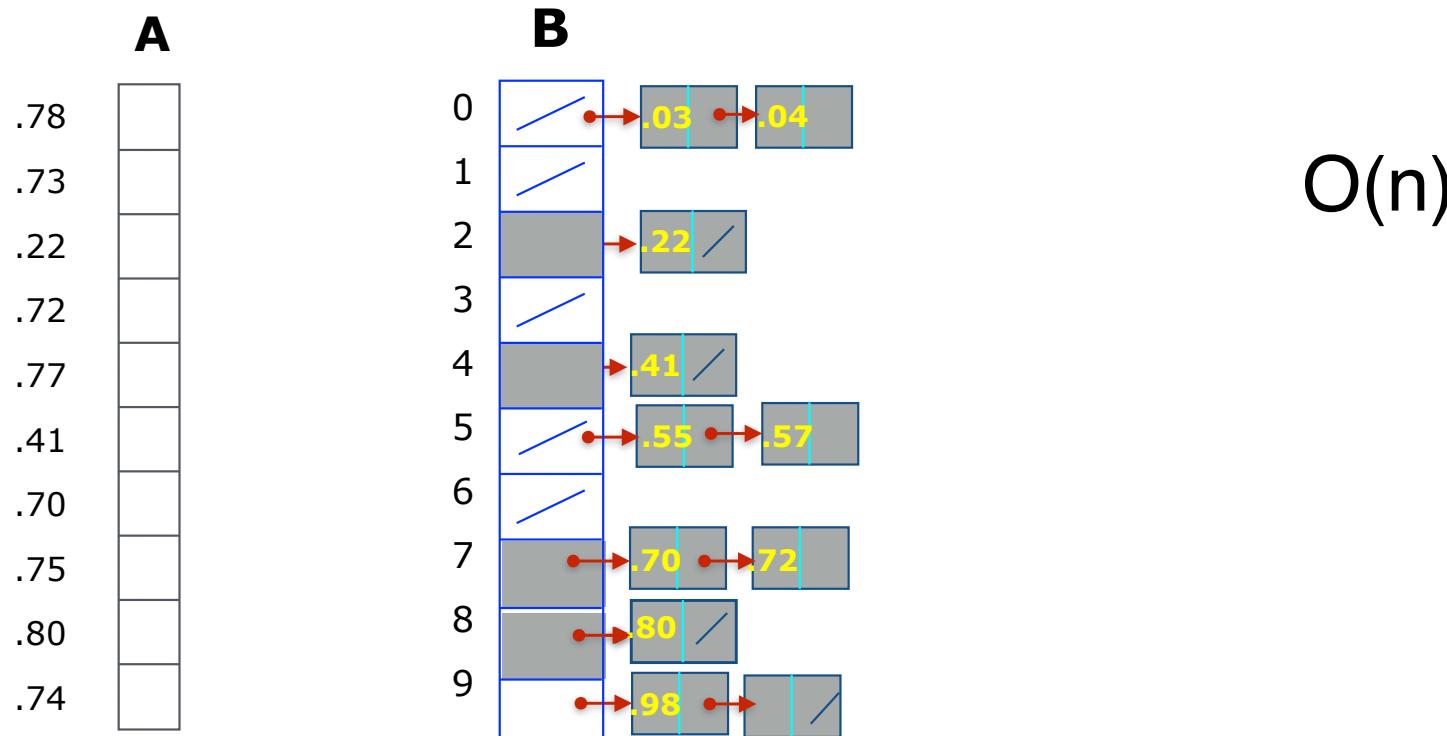
Die Zahlen sind gleich verteilt!



Komplexität

Durchschnittlicher Fall

Die Zahlen sind nicht gleich verteilt!





Welches ist die längste Zeichenfolge, die sich wiederholt?

a a b c f d e s a b a b c d f e r f c s d e a e d a b c f d e s a b e d a

a a b c f d e s a b a b c d f e r f c s d e a e d a b c f d e s a b e d a

Anwendungen

- Linguistik
- Bioinformatik (DNA-Analyse)
- Datenkompression
- Untersuchung von Plagiaten
- Antiviren-Software
- Musik-Analyse
- usw.

Lösung mit Brute-Force



Für alle Startpositionen (*i*, *j*) müssen wir das längste Präfix finden.

Anzahl der *i*, *j* Kombinationen = $(n-1) + (n-2) + \dots + 2 + 1 = O(n^2)$

Lösung mit Sortieralgorithmen

0	a a c f d e b d a b c f d e b e d a
1	a c f d e b d a b c f d e b e d a
2	c f d e b d a b c f d e b e d a
3	f d e b d a b c f d e b e d a
4	d e b d a b c f d e b e d a
5	e b d a b c f d e b e d a
6	b d a b c f d e b e d a
7	d a b c f d e b e d a
8	a b c f d e b e d a
9	b c f d e b e d a
10	c f d e b e d a
11	f d e b e d a
12	d e b e d a
13	e b e d a
14	b e d a
15	e d a
16	d a
17	a

["a a c f d e b d a b c f d e b e d a",
 " a c f d e b d a b c f d e b e d a",
 " c f d e b d a b c f d e b e d a",
 " f d e b d a b c f d e b e d a".
 " d e b d a b c f d e b e d a",
 " e b d a b c f d e b e d a",
 " b d a b c f d e b e d a",
 " d a b c f d e b e d a",
 " a b c f d e b e d a",
 " b c f d e b e d a",
 " c f d e b e d a",
 " f d e b e d a",
 " d e b e d a",
 " e b e d a", Suffixen oder
 " b e d a", Startpositionen innerhalb
 " e d a", der gesamten
 " d a", Zeichenketten
 " a"]

Lösung mit Sortieralgorithmen

```
17  a
 0  a a c f d e b d a b c f d e b e d a
 8  a b c f d e b e d a
 1  a c f d e b d a b c f d e b e d a
 9  b c f d e b e d a
 6  b d a b c f d e b e d a
14  b e d a
10  c f d e b e d a
 2  c f d e b d a b c f d e b e d a
16  d a
 7  d a b c f d e b e d a
 4  d e b d a b c f d e b e d a
12  d e b e d a
 5  e b d a b c f d e b e d a
13  e b e d a
15  e d a
11  f d e b e d a
 3  f d e b d a b c f d e b e d a
```

Die Suffixe werden sortiert

Lösung mit Sortieralgorithmen

```

17  a
 0  a a c f d e b d a b c f d e b e d a
 8  a b c f d e b e d a
 1  a c f d e b d a b c f d e b e d a
 9  b c f d e b e d a
 6  b d a b c f d e b e d a
14  b e d a
10  c f d e b e d a
 2  c f d e b d a b c f d e b e d a
16  d a
 7  d a b c f d e b e d a
 4  d e b d a b c f d e b e d a
12  d e b e d a
 5  e b d a b c f d e b e d a
13  e b e d a
15  e d a
11  f d e b e d a
 3  f d e b d a b c f d e b e d a

```

Die Präfix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Präfix gesucht.

Lösung mit Sortieralgorithmen

```

17  a
 0  a a c f d e b d a b c f d e b e d a
 8  a b c f d e b e d a
 1  a c f d e b d a b c f d e b e d a
 9  b c f d e b e d a
 6  b d a b c f d e b e d a
14  b e d a
10  c f d e b e d a
 2  c f d e b d a b c f d e b e d a
16  d a
 7  d a b c f d e b e d a
 4  d e b d a b c f d e b e d a
12  d e b e d a
 5  e b d a b c f d e b e d a
13  e b e d a
15  e d a
11  f d e b e d a
 3  f d e b d a b c f d e b e d a

```

Die Präfix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Präfix gesucht.

Lösung mit Sortieralgorithmen

```

17  a
 0  a a c f d e b d a b c f d e b e d a
 8  a b c f d e b e d a
 1  a c f d e b d a b c f d e b e d a
 9  b c f d e b e d a
 6  b d a b c f d e b e d a
14  b e d a
10  c f d e b e d a
 2  c f d e b d a b c f d e b e d a
16  d a
 7  d a b c f d e b e d a
 4  d e b d a b c f d e b e d a
12  d e b e d a
 5  e b d a b c f d e b e d a
13  e b e d a
15  e d a
11  f d e b e d a
 3  f d e b d a b c f d e b e d a

```

Die Präfix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Präfix gesucht.

Lösung mit Sortieralgorithmen

```

17  a
0   a a c f d e b d a b c f d e b e d a
8   a b c f d e b e d a
1   a c f d e b d a b c f d e b e d a
9   b c f d e b e d a
6   b d a b c f d e b e d a
14  b e d a
10  c f d e b e d a
2   c f d e b d a b c f d e b e d a
16  d a
7   d a b c f d e b e d a
4   d e b d a b c f d e b e d a
12  d e b e d a
5   e b d a b c f d e b e d a
13  e b e d a
15  e d a
11  f d e b e d a
3   f d e b d a b c f d e b e d a

```

Die Präfix-Länge jeder zwei benachbarten Suffixe werden verglichen und der längste Präfix gesucht.

Implementierung

```
def generate_suffixes(A):
    """ Find all suffixes of the String A """
    n = len(A)
    suffixes = [0 for i in range(len(A))]

    for i in range(n):
        suffixes[i] = A[i:n]

    return suffixes
```

0	a a c f d e b d a b c f d e b e d a
1	a c f d e b d a b c f d e b e d a
2	c f d e b d a b c f d e b e d a
3	f d e b d a b c f d e b e d a
4	d e b d a b c f d e b e d a
5	e b d a b c f d e b e d a
6	b d a b c f d e b e d a
7	d a b c f d e b e d a
8	a b c f d e b e d a
9	b c f d e b e d a
10	c f d e b e d a
11	f d e b e d a
12	d e b e d a
13	e b e d a
14	b e d a
15	e d a
16	d a
17	a

Implementierung

```
def LRS_Algorithmus(A):
    B = generate_suffixes(A)
    B.sort()
    seq = find_longest_seq(B)
    return seq
```

Implementierung

```
def find_longest_seq(B):
    max = 0

    for i in range(len(B)-1):
        min_len = min(len(B[i]),len(B[i+1]))
        j = 0
        while j<min_len and B[i][j]==B[i+1][j]:
            j += 1
        if j > max:
            max = j
        if j == min_len:
            seq = B[i][:j+1]
        else:
            seq = B[i][:j]
    return seq
```