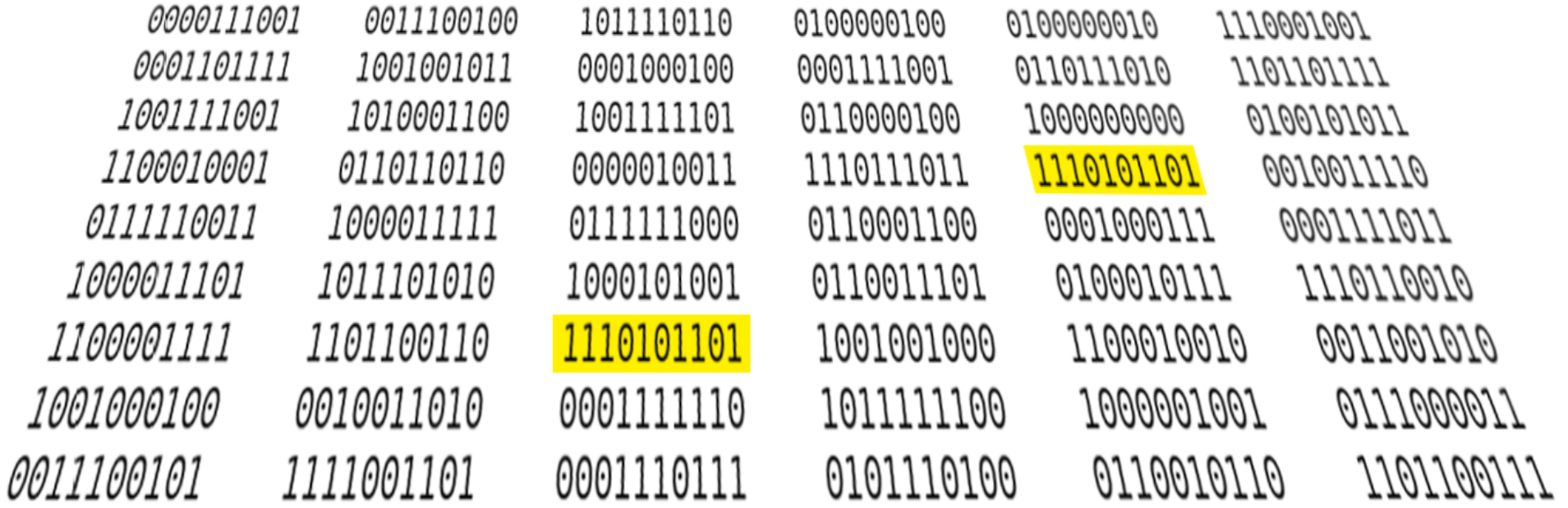
Karlsruher Institut für Technologie

Tutorium 5: Sortieren

Holger Ebhart | 20. Mai 2015

TUTORIUM ZUR VORLESUNG ALGORITHMEN I IM SS15



Gliederung

1. 4. Übungsblatt
2. Sortieren
   - Insertionsort
   - Mergesort
   - Quicksort
   - Heapsort
   - Untere Schranke
   - Aufgaben
3. Zu Übungsblatt 5

4. Übungsblatt

Insertionsort

Idee:

betrachte die Elemente an jeder Stelle 1, ..., n

schiebe das Element solange vor, bis der Vorgänger nicht mehr größer ist

Algorithmus:

```
1 | A:Array[1...n] of Digit
2 |         for i := 2 to n do
3 |                 j := i :Digit
4 |                 while j > 1 & A[j] < A[j-1] do
5 |                         swap(A, j, j-1)
6 |                         j--
7 |                 end
8 |         end
```

Laufzeit:

Average-Case: $O(n^2)$

Worst-Case: $O(n^2)$

Best-Case: $O(n)$

Insertionsort

Idee:

betrachte die Elemente an jeder Stelle 1, ..., n

schiebe das Element solange vor, bis der Vorgänger nicht mehr größer ist

Algorithmus:

```
1 | A:Array[1...n] of Digit
2 |         for i := 2 to n do
3 |                 j := i :Digit
4 |                 while j > 1 & A[j] < A[j-1] do
5 |                         swap(A, j, j-1)
6 |                         j--
7 |                 end
8 |         end
```

Laufzeit:

Average-Case: $\mathrm{O}(n^2)$

Worst-Case: $\mathrm{O}(n^2)$

Best-Case: $\mathrm{O}(n)$

Mergesort

Idee:
Splitte die Menge solange in zwei Teile, bis einelementige Mengen entstehen
merge die Mengen wieder zusammen und sortiere dabei

Laufzeit:
Average-Case: $\mathrm{O}(n \cdot \log n)$
Worst-Case: $\mathrm{O}(n \cdot \log n)$
Best-Case: $\mathrm{O}(n \cdot \log n)$

Mergesort

Idee:
Splitte die Menge solange in zwei Teile, bis einelementige Mengen entstehen
merge die Mengen wieder zusammen und sortiere dabei

Laufzeit:
Average-Case: $\mathrm{O}(n \cdot \log n)$
Worst-Case: $\mathrm{O}(n \cdot \log n)$
Best-Case: $\mathrm{O}(n \cdot \log n)$

Mergesort

Algorithmus:

```
1  procedure mergeSort(A:Array[1...n] of Digit, s,e:Digit)
2          if s = e then
3                  return A[s];
4          else
5                  merge(mergeSort(A, s, (s+e)/2),
                       mergesort(A, (s+e)/2 +1, e));
6
7  procedure merge(A:Array[1...n/m] of Digit)
8          C:Array[1...(m+n)] of Digit
9          for i := 1 to m+n do
10                 C[i] := min(A[l],B[j])
11                 if C[i] = A[l] then l++
12                 else j++
13         end
14         return c;
```

Median und Perzentile

- Median teilt eine Menge M in zwei gleichgroße Untermengen auf
  - $|M|$ ungerade $\rightarrow m = sort(M)_{\frac{|M|+1}{2}}$
  - $|M|$ gerade $\rightarrow m = \frac{sort(M)_{\frac{|M|}{2}} + sort(M)_{\frac{|M|}{2}+1}}{2}$
- Median ist $\frac{1}{2}$ Perzentil
- Wie berechnet man das $\frac{1}{3}$ Perzentil eines unsortierten Arrays mit n Elementen?
  Gehen Sie davon aus es sei eine Medianfunktion gegeben die den Median von m Elementen in $\mathrm{O}(m)$ berechnet. Stellen Sie sicher, dass ihr Algorithmus in $\mathrm{O}(n)$ läuft.

Median und Perzentile

- Median teilt eine Menge M in zwei gleichgroße Untermengen auf
  - $|M|$ ungerade $\rightarrow m = sort(M)_{\frac{|M|+1}{2}}$
  - $|M|$ gerade $\rightarrow m = \frac{sort(M)_{\frac{|M|}{2}} + sort(M)_{\frac{|M|}{2}+1}}{2}$
- Median ist $\frac{1}{2}$ Perzentil
- Wie berechnet man das $\frac{1}{3}$ Perzentil eines unsortierten Arrays mit n Elementen?
  Gehen Sie davon aus es sei eine Medianfunktion gegeben die den Median von m Elementen in $\mathrm{O}(m)$ berechnet. Stellen Sie sicher, dass ihr Algorithmus in $\mathrm{O}(n)$ läuft.

Median und Perzentile

- Median teilt eine Menge M in zwei gleichgroße Untermengen auf
  - $|M|$ ungerade $\rightarrow m = sort(M)_{\frac{|M|+1}{2}}$
  - $|M|$ gerade $\rightarrow m = \frac{sort(M)_{\frac{|M|}{2}} + sort(M)_{\frac{|M|}{2}+1}}{2}$
- Median ist $\frac{1}{2}$ Perzentil
- Wie berechnet man das $\frac{1}{3}$ Perzentil eines unsortierten Arrays mit n Elementen?
  Gehen Sie davon aus es sei eine Medianfunktion gegeben die den Median von m Elementen in $O(m)$ berechnet. Stellen Sie sicher, dass ihr Algorithmus in $O(n)$ läuft.

Median und Perzentile

- Median teilt eine Menge M in zwei gleichgroße Untermengen auf
  - $|M|$ ungerade $\rightarrow m = sort(M)_{\frac{|M|+1}{2}}$
  - $|M|$ gerade $\rightarrow m = \frac{sort(M)_{\frac{|M|}{2}} + sort(M)_{\frac{|M|}{2}+1}}{2}$
- Median ist $\frac{1}{2}$ Perzentil
- Wie berechnet man das $\frac{1}{3}$ Perzentil eines unsortierten Arrays mit n Elementen?
  Gehen Sie davon aus es sei eine Medianfunktion gegeben die den Median von m Elementen in $\mathrm{O}(m)$ berechnet. Stellen Sie sicher, dass ihr Algorithmus in $\mathrm{O}(n)$ läuft.

Quicksort

Idee:
tue bis nur noch einelementige Mengen existieren
nehme Pivot m und bilde Mengen mit Elementen $<$,$>$,= m
konkateniere die Mengen der Reihenfolge nach

Laufzeit:
Average-Case: $\mathrm{O}(n \cdot \log n)$
Worst-Case: $\mathrm{O}(n^2)$
Best-Case: $\mathrm{O}(n \cdot \log n)$

Quicksort

Idee:
tue bis nur noch einelementige Mengen existieren
nehme Pivot m und bilde Mengen mit Elementen $<$,$>$,= m
konkateniere die Mengen der Reihenfolge nach

Laufzeit:
Average-Case: $\mathrm{O}(n \cdot \log n)$
Worst-Case: $\mathrm{O}(n^2)$
Best-Case: $\mathrm{O}(n \cdot \log n)$

Quicksort

Algorithmus:

```
1  procedure quickSort(A:Array[1...n] of Digit)
2          pick pivot m
3          B,C,D:Array of Digit
4          a=1,b=1,c=1:Digit
5          for i := 1 to n do
6                  if A[i] > m then B[b++] := A[i]
7                  else if A[i] < m then D[d++] := A[i]
8                  else C[c++] := A[i]
9          end
10         return quickSort(B) + C + quickSort(D)
11
12 procedure merge(A:Array[1...n/m] of Digit)
13         C:Array[1...(m+n)] of Digit
14         for i := 1 to m+n do
15                 C[i] := min(A[l],B[j])
16                 if C[i] = A[l] then l++
17                 else j++
18         end
```

Heapsort

Idee:
Füge alle Elemente in einen Heap ein
entnehme stets das kleinste Element

Laufzeit:
Average-Case: $O(n \cdot \log n)$
Worst-Case: $O(n \cdot \log n)$
Best-Case: $O(n \cdot \log n)$
Algorithmus:

```
1 procedure heapSort(A:Array[1...n] of Digit)
2         H := buildHeap(A):Heap
3         for i := 1 to n do
4                 A[i] := H.deleteMin()
5         end
6         return A;
```

Heapsort

Idee:
Füge alle Elemente in einen Heap ein
entnehme stets das kleinste Element

Laufzeit:
Average-Case: $\mathrm{O}(n \cdot \log n)$
Worst-Case: $\mathrm{O}(n \cdot \log n)$
Best-Case: $\mathrm{O}(n \cdot \log n)$
Algorithmus:

```
1 procedure heapSort(A:Array[1...n] of Digit)
2         H := buildHeap(A):Heap
3         for i := 1 to n do
4                 A[i] := H.deleteMin()
5         end
6         return A;
```

Untere Schranke

Vergleichsbasiertes Sortieren von n Elementen:

- $n!(< 2^H)$ verschiedene Möglichkeiten
- baue binären Entscheidungsbaum mit den $n!$ Möglichkeiten (mit max. 2^H Blättern)
- Baum hat Höhe $n \cdot \log n$
- $\rightarrow n \cdot \log n$ Vergleiche

Aufgabe 1

Es sei die folgende Zahlenfolge gegeben:
$Z_1 = (6, 8, 7, 33, 56, 1, 14, 16, 29)$
Sortieren sie Z_1 mit

- Insertionsort
- Mergesort
- Quicksort ($Pivot = \lfloor \frac{|Z_i|}{2} \rfloor$)

und geben Sie jeweils die Anzahl benötigter Vergleiche an.

Aufgabe 2

Welcher der folgenden Algorithmen sollte für welche Zahlenfolge gewählt werden. Begründen Sie warum.
Algorithmen:

- Insertionsort
- Mergesort
- Quicksort ($Pivot = \lfloor \frac{|Z_i|}{2} \rfloor$)

Zahlenfolgen:

- $Z_1 = 8, 7, 6, 5, 4, 3, 2, 1$
- $Z_2 = 1, 8, 2, 3, 7, 4, 6, 5$
- $Z_3 = 1, 2, 3, 4, 5, 6, 7, 8$

Inversionen

Geben Sie die Anzahl an Inversionen in folgenden Folgen an. Geben Sie bei a) und b) zusätzlich noch jeweils alle Inversionen an.

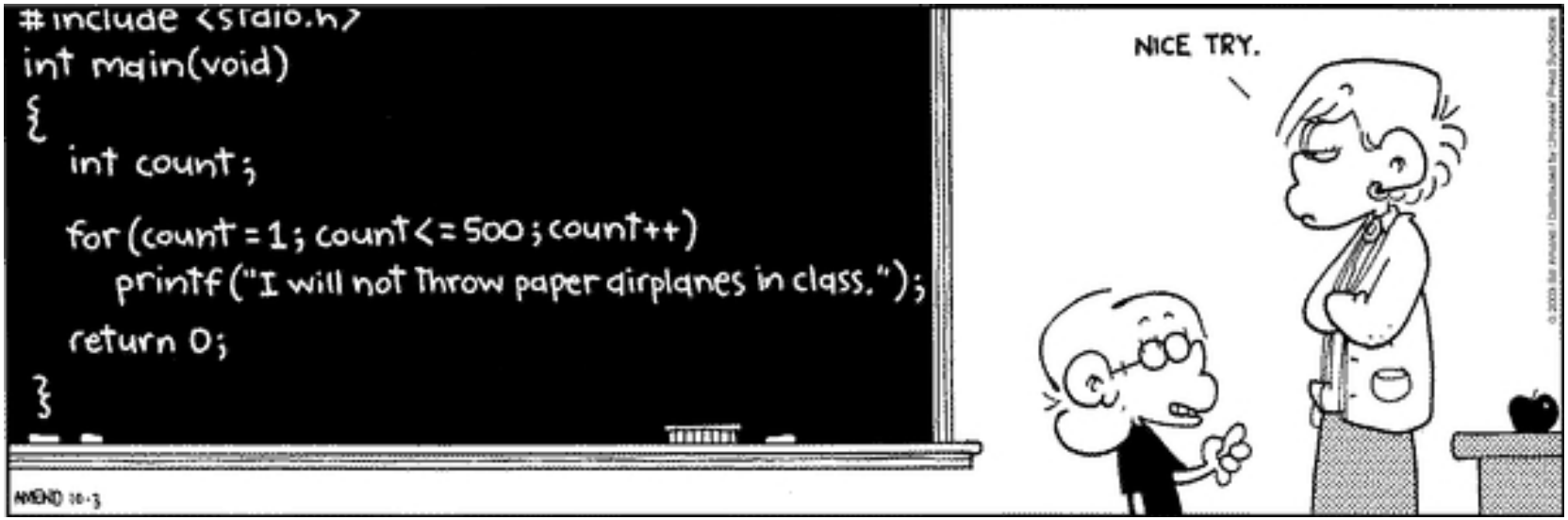
a) $F_1 = 1, 3, 2, 5, 4, 0$

b) $F_2 = 34, 21, 34, 56, 99$

c) $F_3 = 67, 5, 87, 32, 34, 55, 3, 99, 82, 49$

d) $F_4 = 9, 8, 7, 6, 5, 4, 3, 2, 1$

Vielen Dank für eure Aufmerksamkeit!
Bis zum nächsten Mal.



stackoverflow.com