Tut 04

Tuesday, 7. May 2019 20:44

0. ORGANISATORISCHES

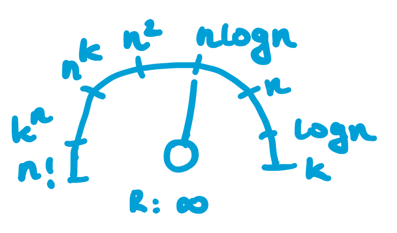
⚠ 1.Block-Test

- Start: 13.05. (Mo) 10^{00}
- Best before: 19.05. (So) 23^{59}

- HA3 Songs: Welcher Tutor hat den besten Musikgeschmack?
  → Spotify Playlist od!

1. SORTIERALGORITHMEN

SORTIERALGORITHMEN



- Sortieralgorithmen: Verwendung bei der binären Suche:
  → nur möglich wenn aufsteigend sortiert
- Wir lernen 3 unterschiedlich schnelle Sortieralgorithmen kennen

1.1 Stabilität

- Folie: Stabilität
- Ein Sortieralgorithmus arbeitet stabil, wenn bei gleichem Schlüssel die Reihenfolge erhalten bleibt

- Selectionsort & Quicksort nur mit Glück stabil
  → i.A. nicht stabil
- Insertionsort stabil (i.A.)

Def: Stabilität

Reihenfolge v. Elementen mit gleichem Sortierschlüssel nach Sortieren unverändert (evtl)

1.2 In-place vs. Out-of-place

- In-place wenn innerhalb des Arrays gearbeitet wird (kein zusätzliches Zwischenarray nötig)
  → aka. In-Situ (hudefak says that ?!)
- Out-of-place wenn beim Sortieren zusätzlich Speicher belegt wird (Zwschritt)
  → aka. Ex-Situ (again ?!)
- Selection, Insertion & Quicksort können In-place umgesetzt werden

1.3 Selectionsort

- Suche das Minimum und tausche wenn kleiner
  → wenn nicht, gehe weiter

Selectionsort

5	3	12	1	6	9
1	3	12	5	6	9
1	3	12	5	6	9
1	3	5	12	6	9
1	3	5	6	12	9
1	3	5	6	9	12
1	3	5	6	9	12

1.4 Insertionsort

- Element f. Element durchlaufen
- Jedes Element so lang verschoben, bis Position richtig ist

Insertionsort

5	3	12	4	10
3	5	12	4	10
3	5	12	4	10
3	5	4	12	10
3	4	5	12	10
3	4	5	12	10
3	4	5	10	12
3	4	5	10	12

• : Vgl mit
☐ : aktuell

- Programmcode

```
public class InsertionSort {
    public static void sort(int[] arr){
        for(int i=0; i<arr.length ; ++i)
            for(int j=i ; arr[j] < arr[j-1] && j>0 ; --j)
                swap(arr, j, j-1);
    }

    private static void swap(int [] arr, int x, int j){
        int a= arr[x];
        arr[x] = arr[y];
        arr[y] = a;
    }
}
```

1.5 Quicksort

- Selection / Insertionsort i.d.R. ausreichend schnell, jedoch n^2 im Avg.-Case
- Quicksort nlogn Avg., sehr vorteilhaft bei großen Datenmengen
- –"– basiert auf dem Teile- und herrsche Prinzip (?)
  → Zerlegung eines Großen Problemes in kleine Teilprobleme
  → rekursiver Algorithmus

1.5.1. Out-of-place

- Einfacher zu verstehen
- (1) Wähle ein 'Pivot'-Element
- (2) $\genfrac{}{}{0pt}{}{\text{Kleiner}(=)}{\text{größer}}$ Pivot ? $\genfrac{}{}{0pt}{}{\text{Linkes}}{\text{rechtes}}$ Teilarray
- (3) wiederhole (1) so lange bis 1 Elementiges Array

40	17	30	23	15	28	33	1	9	5	19	29	9	13	18
17	15	1	9	5	9	13	18	40	30	23	28	33	19	29
1	9	5	9	13	17	15	18	23	28	19	29	40	30	33
1	9	5	9	13	15	17	18	19	23	28	29	30	33	40
1	5	9	9	13	15	17	18	19	23	28	29	30	33	40

1.5.2 In-Place

- Denksport of the day

rekursiv

- Pivot: Letztes Element
- 2 zähler: Laufindex i, Grenzindex g, ==0
- arr[i] $\begin{cases} < \text{arr[Pivot]} & \text{swap arr[i++], arr[g++]} \\ > \text{arr[Pivot]} & \text{i++} \end{cases}$
- Arr Ende ? swap arr[g], arr[Pivot]
- Linkes TA: Links v. g, v.v.

40	17	30	23	15	28	40 > 28 → i++	
40	17	30	23	15	28	17 < 28 → sw, i++, g++	
17	40	30	23	15	28	30 > 28 → i++	
17	40	30	23	15	28	23 < 28 → sw, i++, g++	
17	23	30	40	15	28	15 < 28 → sw, i++ g++	
17	23	15	40	30	28	40 ↔ 28	
17	23	15	28	30	40	17 > 15, i++	30 < 40, sw, i++, g++
17	23	15	28	30	40	23 > 15, i++	40 ↔ 40
17	23	15	28	30	40	15 ↔ 17 (Pivot)	
15	23	17	28	30	40	23 > 17, i++	
15	23	17	28	30	40	23 ↔ 17 (Pivot)	
15	17	23	28	30	40	23 ↔ 23 (Pivot)	
15	17	23	28	30	40	puuh...	