Informatik II: Algorithmen und Datenstrukturen SS 2015

Vorlesung 1b, Mittwoch, 22. April 2015

(QuickSort, Divide and Conquer, Rekursion)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Ein besserer Sortieralgorithmus
  - QuickSort: Idee, Beispiel, Wahl des "Pivot" Elementes
  - Divide and Conquer: das allgemeine Prinzip dahinter
  - Rekursion: kurze Wiederholung für die, die es wieder vergessen oder noch nie richtig verstanden haben

  Ü1: QuickSort implementieren und die Laufzeit messen

  Sie werden sehen, dass QuickSort oft (aber auch nicht immer) schneller ist als MinSort

  Warum genau sehen wir dann nächste Woche ...

QuickSort 1/9

UNI FREIBURG

- Informale Beschreibung des Algorithmus
  - Teile die Eingabe in einen linken und einen rechten Teil
  - Alle Elemente links sind ≤ alle Element rechts

    Aber die Elemente im linken Teil sind untereinander (noch) nicht sortiert, und die Elemente im rechten Teil auch nicht
  - Sortiere jetzt die Elemente im linken Teil mit derselben Methode, und die Elemente im rechten Teil ebenso

    Das nennt man Rekursion = ein kleineres Unterproblem von derselben Sorte mit demselben Verfahren lösen
  - Irgendwann werden die Teile so klein, dass das Sortieren trivial wird, dann endet die Rekursion

    Spätestens bei einem Teil mit nur noch einem Element

QuickSort 2/9

UNI FREIBURG

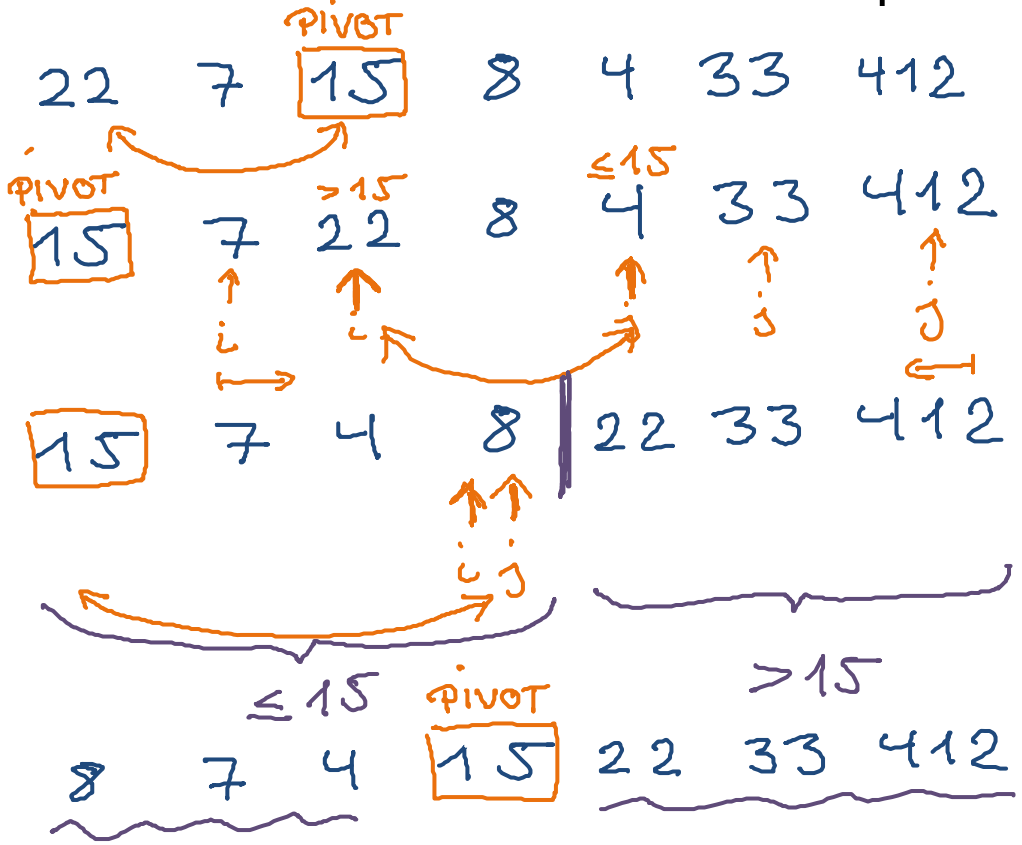
- Aufteilen in zwei Teile
  - Bedingung: alle Elemente im linken Teil sind ≤ alle Elemente im rechten Teil

    Wie wir gleich sehen werden, ist das einfacher als ganz zu sortieren … das ist wesentlich für die ganze Idee
  - Idee: wähle ein Element, den sogenannten **Pivot**

    Ordne dann das Feld so um, dass alle Elemente links vom Pivot ≤ sind und alle Elemente rechts davon > sind

QuickSort 3/9

- Aufteilen in zwei Teile
  - Das kann man so implementieren, dass man "nur einmal über das Feld laufen muss" … hier ein Beispiel:



QuickSort 4/9

UNI FREIBURG

- Aufteilen in zwei Teile
  - Informale Beschreibung des Algorithmus:

    Tausche den Pivot ganz nach links

    Zwei "Zeiger" **i** und **j**, einer beginnt links einer rechts

    Mit **i** nach rechts, bis man auf Element **>** Pivot triff

    Mit **j** nach links, bis man auf Elemente **≤** Pivot trifft

    Tausche die Elemente an den Positionen **i** und **j**

    Alles bei **i** oder links davon dann ≤ alles bei **j** oder rechts davon

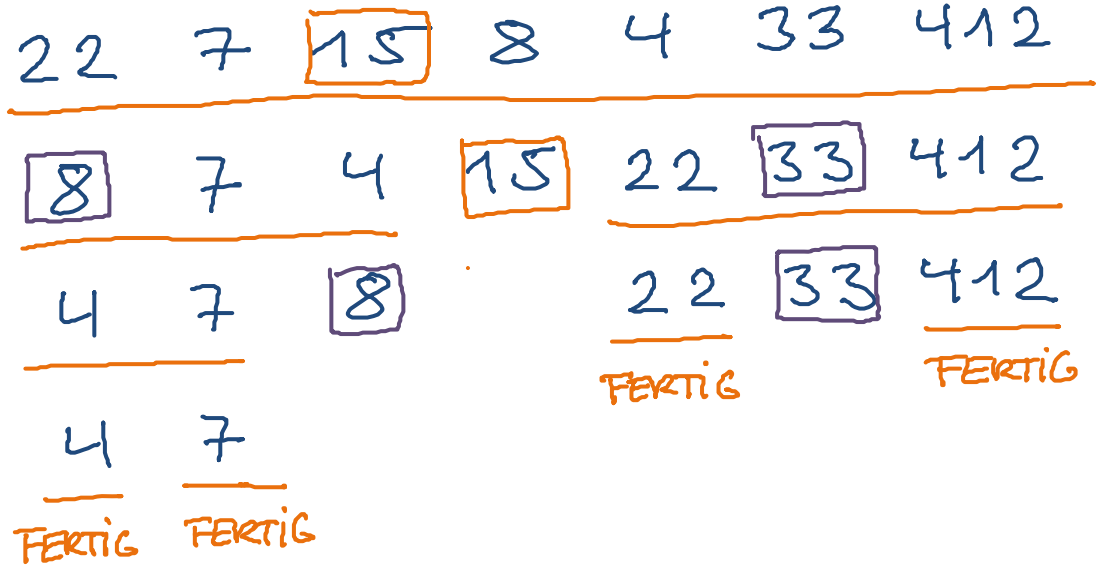
    Wiederhole so lange, bis sich **i** und **j** "treffen" ... dann "tausche" den Pivot an diesen Treffpunkt

    Die Details sollen Sie sich für das Ü1 selber überlegen

QuickSort 5/9

UNI FREIBURG

- Beispiel für den ganzen QuickSort Algorithmus



QuickSort 6/9

UNI FREIBURG

- Live Demos
  - Es gibt auf dem Netz einige schöne Visualisierungen, hier eine besonders intuitive und gut gemachte:

    http://me.dt.in.th/page/Quicksort

    Bei komplizierteren Algorithmen oft hilfreich, Sie aus mehreren Blickwinkeln anzuschauen

    Die eigene Implementierung ist dann immer der ultimative Test, ob man es ganz verstanden hat

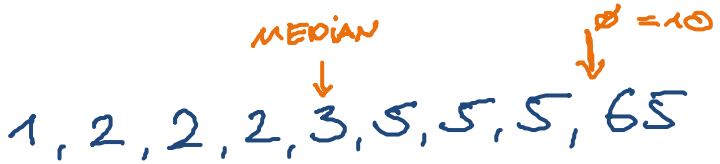
QuickSort 7/9

UNI FREIBURG

- Wahl des Pivot-Elements
  - Je nach Wahl des Pivot-Elementes können die Teile (fast) gleich groß oder sehr unterschiedlich groß sein
  - Es ist am besten, wenn die Teile (fast) gleich groß sind

    Warum genau wird bei der Bearbeitung des Ü1 klar werden … und spätestens nächste Woche

QuickSort 8/9



UNI FREIBURG

- Wahl des Pivot-Elementes
  - Am besten wäre es, genau den **Median** der Eingabe zu wählen, das ist gerade das Element so dass die Hälfte der Elemente ≤ sind und die Hälfte ≥
  - Aber die Bestimmung des Median ist selber ein schwieriges Problem

    Allerdings leichter als Sortieren, was nicht so offensichtlich ist … wenn es interessiert:

    Time Bounds for Selection

    Blum, Floyd, Pratt, Rivest, Tarjan

    Journal of Computer and System Science **1973**

QuickSort 9/9

UNI FREIBURG

- Wahl des Pivot-Elementes, Alternativen
  - Auswahl einer fixen Position, z.B. das erste Element

    Dann kann ein Teil leicht sehr groß werden

    5 7 [43] 8 4 → 4 7 5 8 [43] (PIVOT)
  - Auswahl einer zufälligen Position

    Möglich, dass ein Teil sehr groß wird, aber unwahrscheinlich

    SIEHE OBEN … $P(\text{Pivot} = 43) = \frac{1}{n}$ … n = Eingabegröße
  - Median von drei zufälligen Elementen

    Immer noch möglich, aber noch unwahrscheinlicher

    17 2 [38] 5 9 [114] 12 [42] 7 8 3

Divide and Conquer 1/2

UNI FREIBURG

- Allgemeines Prinzip
  - Teile das gegebene Problem in zwei oder mehr Teile

    Meistens ist es gut, wenn Teile gleich oder ähnlich groß sind
  - Löse diese Teile mit demselben Verfahren

    Führt in der Regel zu einer **rekursiven** Funktion … siehe gleich
  - Irgendwann kommt man so zu Teilen, die so klein sind, dass man sie trivial / mit einem einfachen Verfahren lösen kann

    Spätestens, wenn ein Teil nur noch aus einem Element besteht
  - Manchmal muss man noch etwas tun um die Lösungen für die Teilproblem zu einer Gesamtlösung zusammenzufügen

    Bei QuickSort war das nicht nötig, da ist man durch die Art des Aufteilens schon fertig, wenn man links und rechts sortiert hat

Divide and Conquer 2/2

UNI FREIBURG

- Etymologie
  - Latein: Divide et impera
  - Deutsch: Teile und herrsche
  - Französisch: Diviser pour régner
  - Spanisch: Divide y vencerás
  - Griechisch: Διαίρει και βασίλευε
  - Usw.
  - Das kommt eigentlich aus der Kriegsführung

    So wie der Campus hier übrigens auch

    Wir versuchen, das Beste daraus zu machen …

Rekursion 1/2

UNI FREIBURG

- Ist im Prinzip ganz einfach
  - Eine Funktion ruft sich als Teil Ihres Codes selber wieder auf
  - Man muss nur darauf achten, dass das Ganze irgendwann aufhört, so wie das hier z.B. **nicht**

```
# Never ending recursion.
def eternalDamnation():
    eternalDamnation()
```

Rekursion 2/2

UNI FREIBURG

- Ein einfaches aber nicht-triviales Beispiel
  - **Fibonacci-Zahlen** … die sind ja rekursiv definiert:

    $F_1 = 1,\ F_2 = 1,\ F_n = F_{n-1} + F_{n-2}$ für alle $n \geq 3$

    $1_{F_1}, 1_{F_2}, 2_{F_3}, 3, 5, 8, 13, \ldots$
  - Die programmieren wir jetzt rekursiv … und lassen uns dabei zum Verständnis die Rekursionsstufen ausgeben

    Die rekursive Implementierung ist hier rein zu Demonstrationszwecken und nicht praktikabel:

    Es gibt nämlich eine geschlossene Formel für F_n die man einfach in konstanter Zeit auswerten kann

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

Literatur / Links

UNI FREIBURG

- QuickSort
  - http://en.wikipedia.org/wiki/Quicksort
  - Mehlhorn/Sanders: 5 Sorting and Selection
- Divide and Conquer
  - http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
  - Mehlhorn/Sanders: über das ganze Buch verteilt