Informatik II: Algorithmen und Datenstrukturen SS 2013

Vorlesung 1b, Mittwoch, 17. April 2013
(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

- Organisatorisches
  - Kleine Änderung am 1. Übungsblatt
- QuickSort
  - Beobachtung von gestern: MinSort wird mit wachsender Eingabegröße "unverhältnismäßig" langsamer
  - Heute: QuickSort, ein alternativer Sortieralgorithmus
  - Grundlegende Technik dabei: divide and conquer

QuickSort 1/6

UNI FREIBURG

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

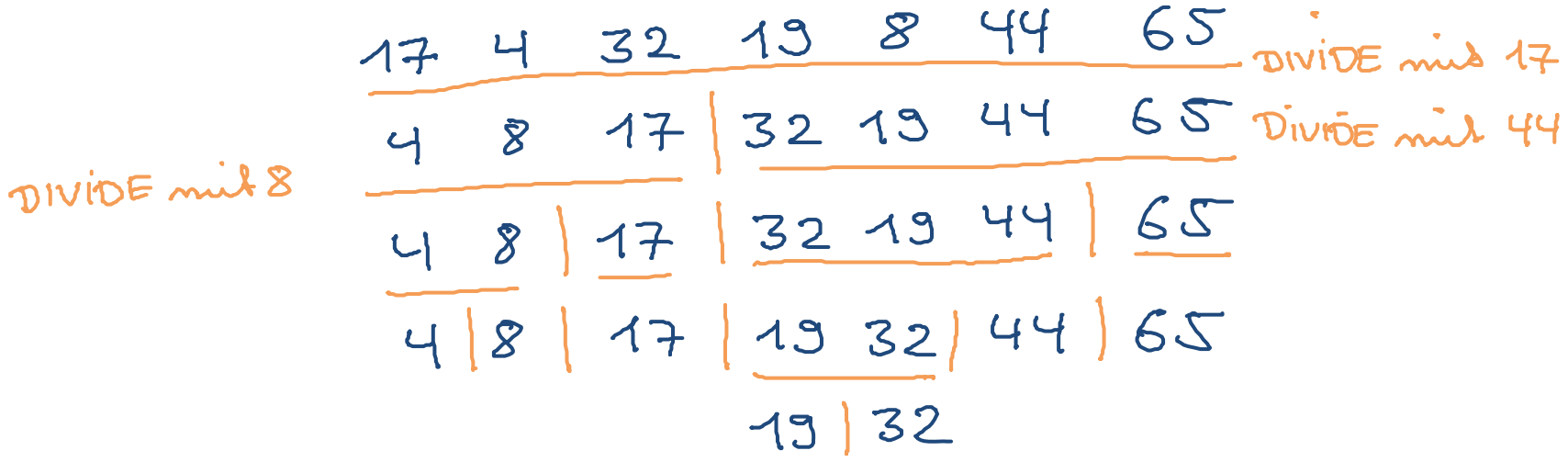
    Aber die Elemente im linken Teil sind untereinander (noch) nicht sortiert, und die Elemente im rechten Teil auch nicht
  - Sortiere jetzt die Elemente im linken Teil mit derselben Methode, und die Elemente im rechten Teil ebenso

    Das nennt man Rekursion = ein kleinerer 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 2/6

- Beispiel



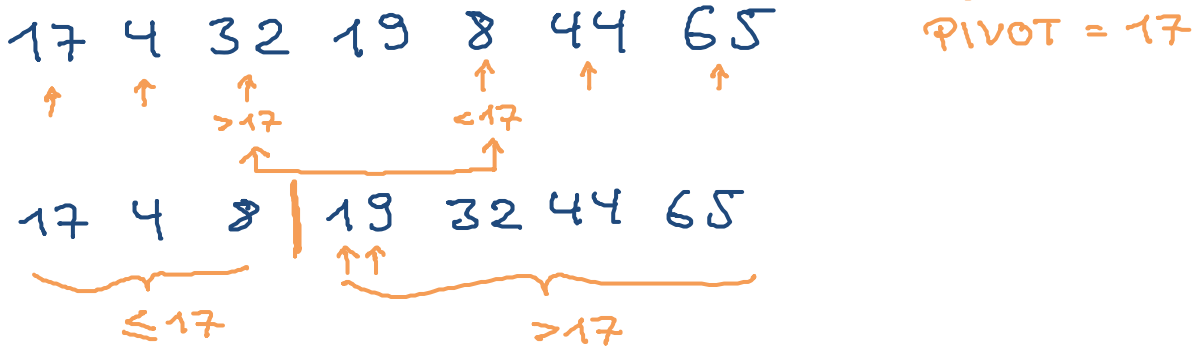
QuickSort 3/6

UNI FREIBURG

- Teilen des Eingabefeldes
  - Wie teilt man in zwei Teile, so dass alle Elemente im linken Teil ≤ alle Elemente im rechten Teil sind ?
  - Idee: wähle ein sogenanntes **Pivot-Element** x

    Alle Elemente < x kommen dann nach links, alle Elemente > x kommen nach rechts

    (wo die Elemente = x hinkommen, kann man sich aussuchen)
  - Man muss dazu nur einmal über das Feld iterieren:
    (von links und von rechts, bis man sich trifft)



QuickSort 4/6

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 später noch klar werden

QuickSort 5/6

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

    dass sich z.B. lösen lässt, indem man die Eingabe sortiert, aber das ist ja gerade das Problem, das wir lösen wollen

QuickSort 6/6

16 14 12 10 8 6 4 2

1 2 3 4 5 6 7 8 9 955 $\phi = 100$

MED = 5.6

UNI FREIBURG

- Wahl des Pivot-Elementes

1 17 32 45 69 2 99 108 12 37 3

  - Auswahl einer fixen Position, z.B. das erste Element

    Dann können die Teile sehr unterschiedlich groß werden

    5 19 8 32 17 (PIVOT)

    5 | 19 8 32 17

  - Auswahl einer zufälligen Position

    Dann ist es möglich, aber unwahrscheinlich, dass die Teile sehr unterschiedlich groß werden

  - Drei Elemente zufällig auswählen und davon den Median berechnen

    das ist einfach ... warum?

    Dann ist es immer noch möglich, aber noch unwahrscheinlicher, dass die Teile sehr unterschiedlich groß werden

    5 19 8 32 17 (PIVOT)

Divide-and-conquer 1/2

UNI FREIBURG

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

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

    Wenn man das implementiert, hat man in der Regel eine rekursive Funktion = eine Funktion, die sich selber aufruft
  - 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

- Ethymologie
  - 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 …

Rekursion beim Programmieren 1/2

UNI FREIBURG

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

```
// Compiles, but never ends.
int eternalDamnation(int x) {
   return eternalDamnation(x);
}
```

  - Aber auch wenn das nicht passiert, ist es nicht leicht zu verstehen, was beim Aufruf einer rekursiven Funktion tatsächlich passiert … deswegen jetzt ein Beispiel dazu

Rekursion beim Programmieren 2/2

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

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

    $F_0 = 0,\ F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$ für alle $n \geq 2$
  - Die programmieren wir jetzt zu Demonstrationszwecken mal rekursiv

    $T(n)$ = Laufzeit
    $T(n) \sim T(n-1) + T(n-2)$
    $\Rightarrow T(n) \sim F(n)$
  - Anmerkung: in der Praxis wird man Fibonacci-Zahlen aus zwei Gründen nicht rekursiv berechnen:
    1. Es gibt eine geschlossene Formel für F_n
    2. Die rekursive Berechnung hat Laufzeit $\sim F_n$, es geht aber auch einfach **iterativ** mit Laufzeit $\sim n$
       (einfach $F_0, F_1, F_2, \ldots, F_n$ der Reihe nach berechnen)

Literatur / Links

UNI FREIBURG

- QuickSort
  - http://en.wikipedia.org/wiki/Quicksort
  - Mehlhorn/Sanders: 5 Sorting and Selection
  - Cormen/Leiserson/Rivest: II.8 Quicksort
- Divide and Conquer
  - http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
  - Mehlhorn/Sanders: throughout the book
  - Cormen/Leiserson/Rivest: I.1.3.1 Divide-and-conquer