Algorithmen und Datenstrukturen 1

Prof. Dr. Carsten Lecon

Wiederholung

- Beweis der Korrektheit:
  - Z.B. mittels Invarianten
- Komplexität
  - Laufzeit wird größenordnungsmäßig beschrieben
    - Meist O-Notation

Inhalt I

- Analyse von Algorithmen
  - Einführung
  - Beispiel
  - Analyse
  - Wachstum von Funktionen
- Entwurf von Algorithmen
  - Einführung
  - Teile und herrsche
  - Greedy-Verfahren
  - Backtracking

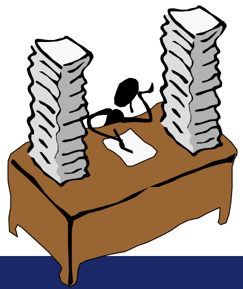
Divide & Conquer

- Teile und herrsche (Divide & Conquer):
  - Angeblich Ausspruch des französischen Königs Ludwig XI.
- Allgemeines Prinzip:
  - Teile eine Problem so lange in Teilprobleme, bis die Problemlösung trivial, offensichtlich oder stark vereinfach ist
- Prinzip (genauer):
  - Methode M zur Lösung eines Problems P der Größe n:
    - Direkte Lösung: Falls $n \le n_0$: Löse das Problem direkt
    - Teile (*divide*): Teile P in Teilprobleme P_1, P_2, …, P_k ($k \ge 2$)
    - Herrsche (*conquer*): Löse jedes Teilproblem P_i mit Methode M
    - Kombination (*merge, combine*): Setze die Teillösungen zusammen

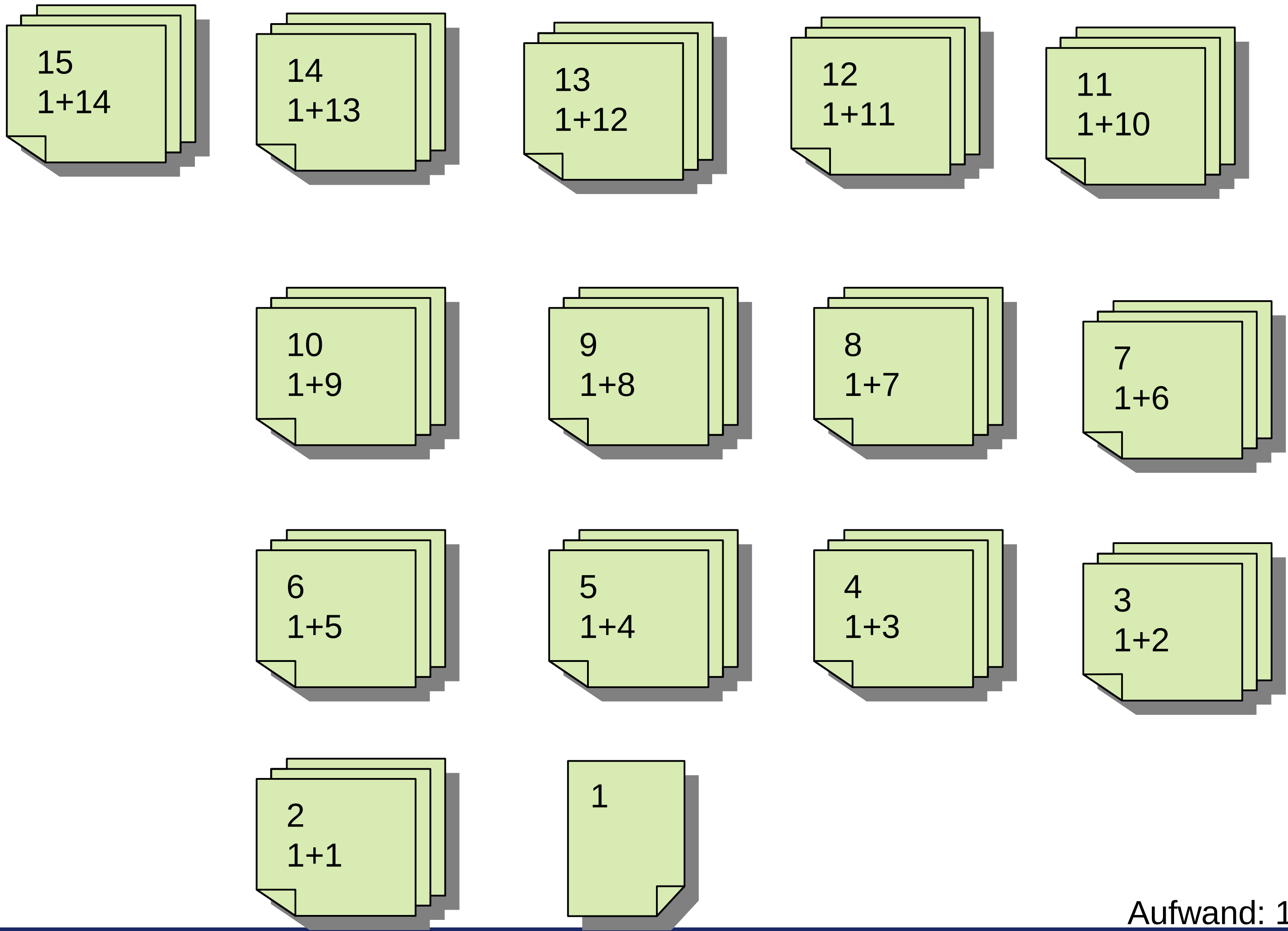
Divide & Conquer

- Beispiel 1:
  - Austeilen von n Übungsblättern an n Vorlesungsteilnehmende

- 1. Variante:
  - Jede(r) nimmt ein Blatt und reicht den Stapel weiter.
  - Aufwand?
    - O(n)
- 2. Variante („teile und herrsche“):
  - Jeder nimmt vom Stoß Blätter ein Blatt und reicht die Hälfte des Stapels an zwei Nachbarn weiter.
  - Aufwand?
    - O ($\log_2$ n)

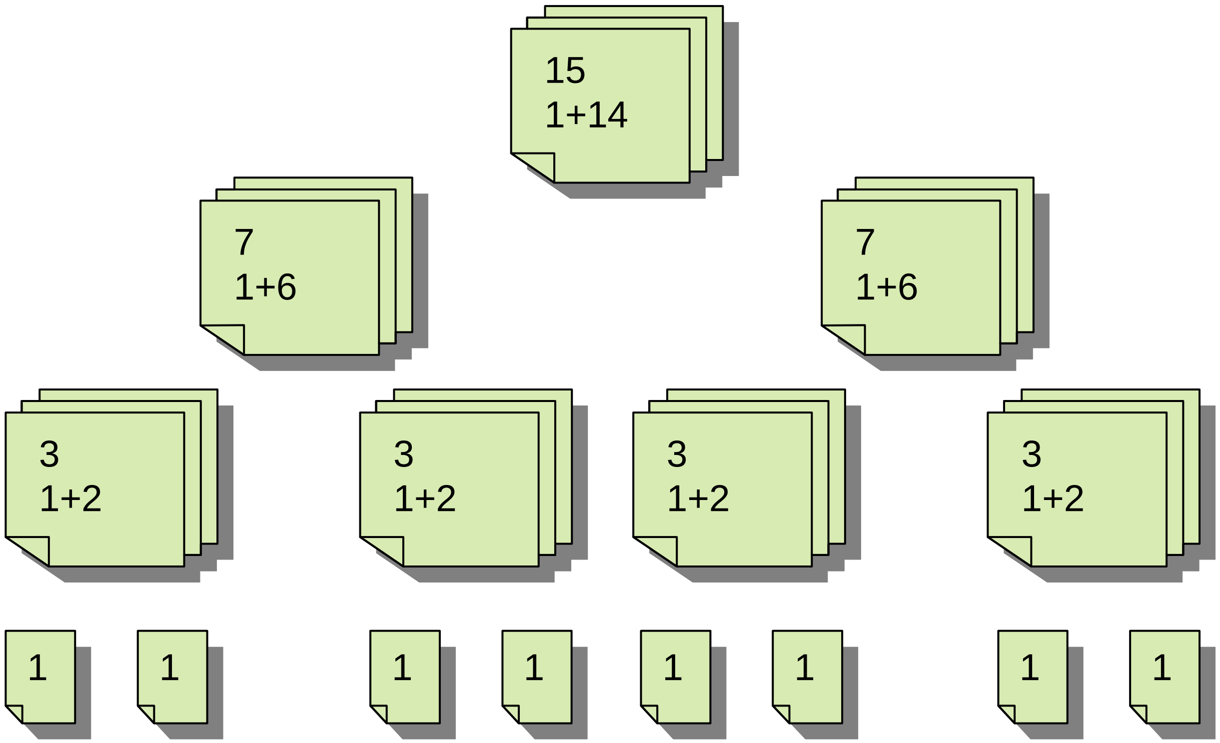


Divide & Conquer: Übungsblattverteilen



Aufwand: 15 Schritte → O(n)

Divide & Conquer: Übungsblattverteilen



Aufwand: 4 Schritte → $O(\log_2 n)$

Divide & Conquer

- Beispiel 2: Sortieralgorithmus (Details später)

  – Teilen: Aufteilen der Zahlenfolge A in zwei Abschnitte:
  A[p…q-1], mit a ≤ A[q] für alle a ε A[p…q-1]
  A[q+1…r], mit a ≥ A[q+1…r]
  (Index q steht zunächst nicht fest, wird im Algorithmus berechnet)
  (Element A[q] steht an der richtigen Position)

  – Herrschen: Sortiere die Folgen A[p…q-1] und A[q+1…r]
  (durch rekursiven Aufruf)

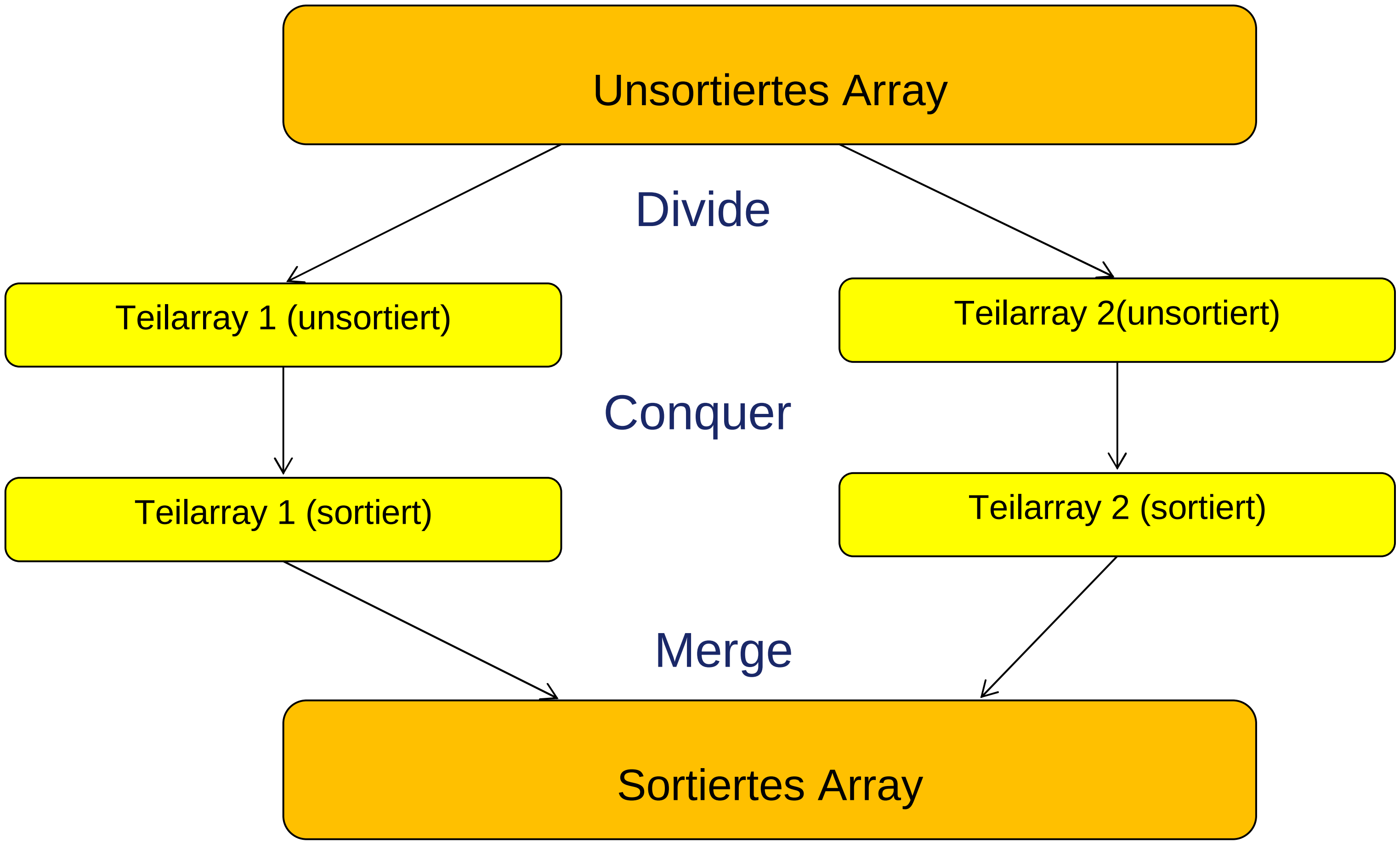
  – Kombinieren: (Automatisch, da *in place* sortiert wurde)

Divide & Conquer

- Beispiel 3: **Mergesort**
  - **Divide**: Teile eine n-elementige Sequenz in zwei Teilsequenzen mit je n/2 Elementen.
  - **Conquer**: Sortiere die beiden Teilsequenzen mittels Mergesort.
  - **Combine (Merge)**: Mische die beiden Teilsequenzen, so dass eine sortierte Sequenz resultiert.

  - Rekursion!

Divide & Conquer: Mergesort



Divide & Conquer: Mergesort

- Ablauf Hilfsfunktion **`merge`**:
  - **Divide**:
    - Erstellung von zwei Teilarrays (Ergebnis steht im Original-Array)
  - **Conquer** (Sortieren):
    - Schleife (Variable k) über die Anzahl der linken plus rechten Seite
    - Sukzessive Elemente aus der linken und rechten Seite nehmen, so dass eine sortierte Folge resultiert („Reißverschlussverfahren“)
  - **Merge**
    - Entfällt, da in A bereits durch die Teil-Sortierungen die richtigen Elemente stehen
  - Beispiel:
    - Folge 9 8 7 4 5 6 3 2 1

Divide & Conquer (Pseudocode - 1)

```
function merge(A, p, q, r)
  if Länge(A) ≤ 1 then return
  n1 := q-p+1
  n2 := r-q
  erzeuge Arrays L[1..n1+1] und R[1..n2+1]
  for i=1 to n1 do
    L[i] := A[p+i-1]
  end for
  for j=1 to n2 do
    R[j] := A[q+j]
  end for
  L[n1+1] = ∞
  R[n2+1] = ∞
```

Divide & Conquer (Pseudocode - 2)

```
i := 1
j := 1
for k=p to r do
  if L[i] ≤ R[j]
    A[k] := L[i]
    i := i+1
  else
    A[k] := R[j]
    j := j+1
  end if
end for
```

Divide & Conquer: Mergesort

- Beweis der Korrektheit:
  - Schleifeninvariante: Array A[p..k-1] ist sortiert
  - Initialisierung:
    - Vor dem ersten Schleifendurchlauf mit k=p ist Array A[p..k-1] leer.
      Die Invariante ist erfüllt.
  - Aufrechterhaltung:
    - Wenn L[i]≤R[j], dann ist L[i] kleinstes noch nicht einsortiertes Element.
      Dann enthält A[p..k] die k-p+1 kleinsten Elemente.
      Zusammen mit der Erhöhung von i und k garantiert das die Einhaltung der Invarianten.
    - Analog für L[i]>R[j].
  - Terminierung:
    - Nach Ende der Schleife enthält A[p..r] die r-p+1 kleinsten Elemente.
      Also sind alle Element einsortiert.

Divide & Conquer

- Laufzeitabschätzung von **`merge`**: O(n):
  - n=r-p+1
  - Konstante Zeit für Zuweisungen und Array-Erstellung
  - Kopieren: $\Theta(n_1+n_2)$
  - n Iterationen der for-Schleife

Divide & Conquer

Algorithmus Mergesort

```
function mergesort(A,p,r)
  if p<r then
    q := |(p+r)/2 |
    mergesort(A,p,q)
    mergesort(A,q+1,r)
    merge(A,p,q,r)
  end if
end function
```

Divide & Conquer

- Allgemeine Laufzeitabschätzung von rekursiven Algorithmen:
- Wenn kleine Problemgröße $n \leq c \rightarrow$ konstante Zeit: $O(1)$
- Sonst:
  - *a* Unterprobleme (Teile von *Divide & Conquer*)
  - Zeitaufwand der Unterprobleme: *n/b* der Zeit des Gesamtaufwands
  - *D(n)*: Aufwand für Teilen (*Divide*)
  - *C(n)*: Aufwand für Zusammenführung (*Combine*)

  - → Zeitaufwand: $T(n) = a\,T(n/b) + D(n) + C(n)$ *(„Mastertheorem“)*

Divide & Conquer

- Laufzeitabschätzung für Mergesort
- Vereinfachte Annahme (o.B.d.A): n ist Potenz von 2
- *Divide*: Konstante Zeit: O(1)
- *Conquer*: Die beiden Teilprobleme der Größe n/2 werden rekursiv gelöst, also 2T(n/2)
- *Merge*: O(n) (s.o.: **`merge`**)

- → Aufwand für n>1:
  T(n) = 2T(n/2) + O(n)
- Laufzeit O(n log n) [vergleiche Tafelanschrieb]

Inhalt I

- Analyse von Algorithmen
  - Einführung
  - Beispiel
  - Analyse
  - Wachstum von Funktionen
- Entwurf von Algorithmen
  - Einführung
  - Teile und herrsche
  - Greedy-Verfahren
  - Backtracking

Greedy-Algorithmus

- Prinzip: Gier (engl. *greed* = Gier)
  - Das Beste (Größte, Günstigste) zuerst
  - In jedem Schritt wird die im Moment günstigste Wahl getroffen
    - Dies ist die lokal günstigste Wahl
    - Die endgültige Abfolge von Schritten muss nicht global optimal sein
  - Warum dann überhaupt dieses Verfahren?
    - Bei manchen Optimierungsproblemen ist die Laufzeit zur Erlangung des globalen Optimums sehr ungünstig.

Greedy-Algorithmus

- Beispiel:
  - Gegeben:
    - ein Betrag W an Wechselgeld
    - eine Menge B von Münzwerten
  - Gesucht:
    - Folge von Münzwerten mit
      - möglichst kurzer Länge
      - Gesamtwert W
  - Beispielwerte:
    - W = 98
    - B = {50, 20, 10, 5, 2, 1}

Greedy-Algorithmus

```
function Münzwechsel(W,B)
  while W≠0 do
    b=sucheGrößteMünze(B,W)
    zahleAus(b)
    W = W-b
  end while
end function
```

Greedy-Algorithmus

- Beispielwerte (2):
  - Wechselgeld W = 60
  - Münzwerte B = {41, 20, 1}

- → Greedy-Algorithmus liefert suboptimales Ergebnis!