

# Algorithmen und Datenstrukturen

Wintersemester 2018/19

9. Vorlesung

## Sortieren in Linearzeit

# Reminder

Melden Sie sich unbedingt unter WueCampus an zur

Teilnahme am 1. Kurztest am 22.11.

...sonst können Sie nicht teilnehmen!

Anmeldefrist: Di, 20.11., 13:00 Uhr

... aber besser gleich *jetzt!*

# Reminder II

## **Auslandsaufenthalte für Studierende der Informatik**

Infoveranstaltung, heute (15.11.), SE 10 (ehem. TB Physik)

- 16:00 Partnerschaften und Fördermöglichkeiten der Uni Würzburg  
(Susanne Holzheimer, International Students Office)
- 16:20 Das TASSEP-Programm  
(Prof. Dr. Sebastian von Mammen)
- 16:30 Praktika und Abschlussarbeiten im Ausland  
(Prof. Dr. Tobias Hoßfeld und Kathrin Borchert)
- 16:40 Partnerschaften in der Informatik: Konkrete Beispiele  
(Prof. Dr. Alexander Wolff)
- 17:00 Einzelgespräche

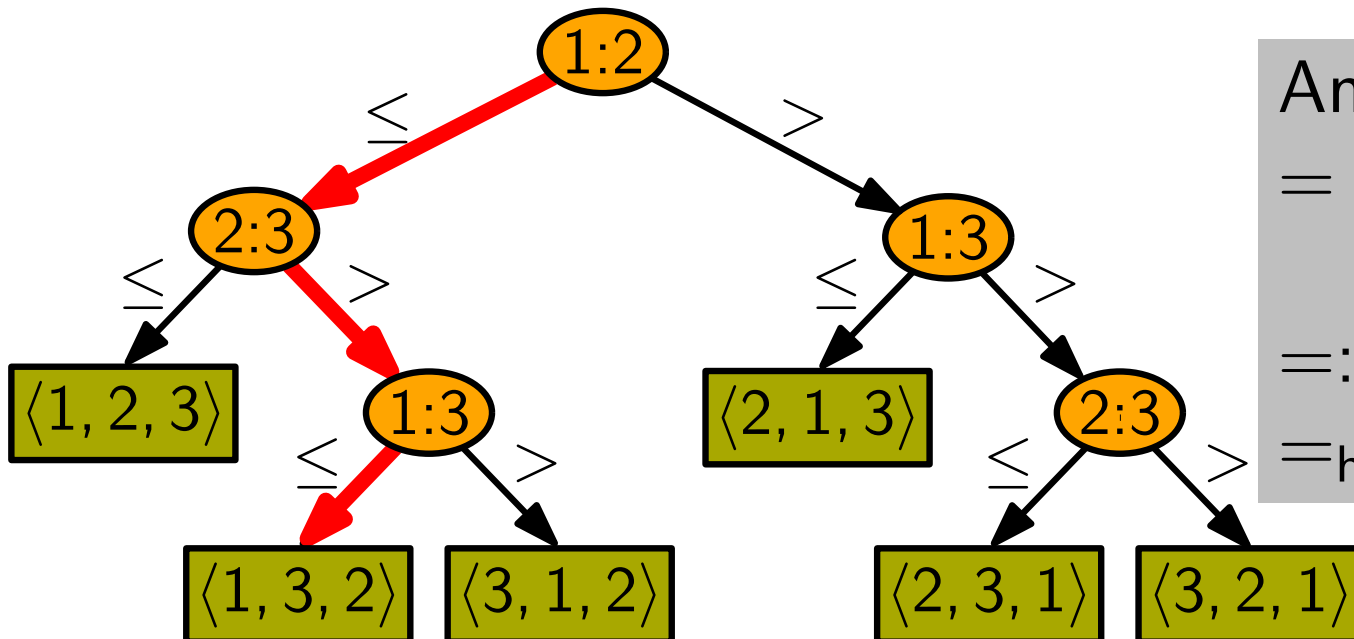
# Sortieren durch Vergleichen

Eingabefolge  $\langle a_1, a_2, \dots, a_n \rangle$   $\xrightarrow[\text{Schlüsselvergleiche}]{\text{Sortieralg.}}$  Ausgabe: sortierte Eingabe

Für festes  $n$  ist ein *vergleichsbasierter Sortieralg.* charakterisiert durch seinen *Entscheidungsbaum*:



- innere Knoten = Vergleiche (o.B.d.A. immer  $\leq$ , z.B. „ $a_1 \leq a_2$ ?“)
- Blätter = sortierte Permutationen der Eingabe
- Kanten = Ergebnisse ( $\leq$ / $>$ ) eines Vergleichs



Anz. Vgl. im *worst case*  
 = Länge eines *längsten*  
 Wurzel-Blatt-Pfads  
 =: Höhe des Baums  
 = hier 3

Entscheidungsbaum für InsertionSort und  $n = 3$  [CLRS]

# Eine untere Schranke

**Frage:** Wie viele Vergleiche braucht *jeder* vergleichsbasierte Sortialg. im worst case um *n verschiedene* Objekte zu sortieren?

**M.a.W.** Gegeben

- ein beliebiger vergleichsbasierter Sortialgorithmus,
- eine Zahl  $n$  von verschiedenen Objekten, die man sortieren soll, welche Höhe hat der Entscheidungsbaum *mindestens*?

**Beob.:** Die Höhe ist eine Funktion der Blätteranzahl.  
 Anz. Blätter = Anz. Permutationen von  $n$  Obj. =  $n!$   
 Höhe Binärbaum mit  $B$  Blättern  $\geq \lceil \log_2 B \rceil$

---


$$\text{Höhe Entscheidungsbaum} \geq \log_2 n! = \sum_{i=1}^n \log_2 i$$

$$\geq \int_1^n \log_2 x \, dx = \frac{1}{\ln 2} \int_1^n \ln x \, dx = \frac{1}{\ln 2} \int_1^n 1 \cdot \ln x \, dx$$

$$= \frac{1}{\ln 2} \left( x \cdot \ln x \Big|_1^n - \int_1^n x \cdot \frac{1}{x} \, dx \right) = \frac{(n \ln n - 0) - (n - 1)}{\ln 2}$$

$$\in \Omega(n \log n)$$

partielle  
Integration

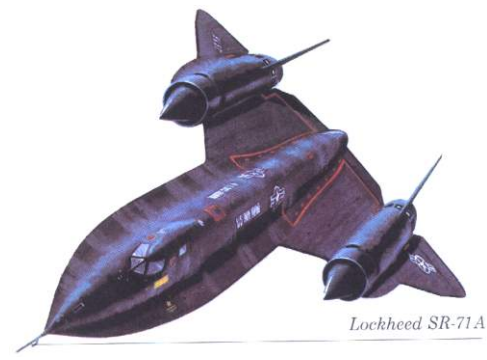
$$\int u' v = uv - \int uv'$$

# Resultat

**Satz.** Jeder vergleichsbasierte Sortieralg. benötigt im schlechtesten Fall  $\Omega(n \log n)$  Vergleiche um  $n$  Objekte zu sortieren.

**Korollar.** MergeSort und HeapSort sind asymptotisch worst-case optimal.

# Wir durchbrechen die Schallmauer



- (● SpaghettiSort sortiert Spaghetti nach Länge ;-)
- CountingSort sortiert Zahlen in  $\{0, \dots, k\}$
- RadixSort sortiert  $s$ -stellige  $b$ -adische Zahlen
- BucketSort sortiert gleichverteilte zufällige Zahlen



# CountingSort

- Idee:**
- 1) für jedes  $x$  in der Eingabe: zähle die Anzahl der Zahlen  $\leq x$
  - 2) benütze diese Information um  $x$  im Ausgabefeld direkt an die richtige Position zu schreiben

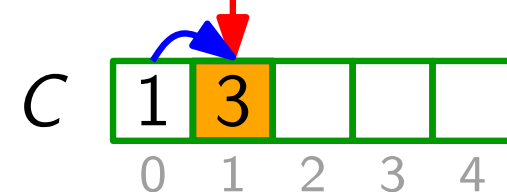
**Variable:**

$A$	Eingabefeld	$C$	Rechenfeld
$B$	Ausgabefeld	$k$	begrenzt das <i>Universum</i> : $\{0, \dots, k\}$

**Bsp:** 1a) Für jedes  $x$  in  $A$ , zähle die Anz. der Zahlen gleich  $x$



1b) Für jedes  $x$  in  $A$ , berechne die Anz. der Zahlen  $\leq x$





# CountingSort

- Idee:**
- 1) für jedes  $x$  in der Eingabe: zähle die Anzahl der Zahlen  $\leq x$
  - 2) benütze diese Information um  $x$  im Ausgabefeld direkt an die richtige Position zu schreiben

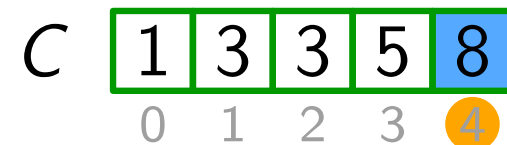
**Variable:**

$A$	Eingabefeld	$C$	Rechenfeld
$B$	Ausgabefeld	$k$	begrenzt das <i>Universum</i> : $\{0, \dots, k\}$

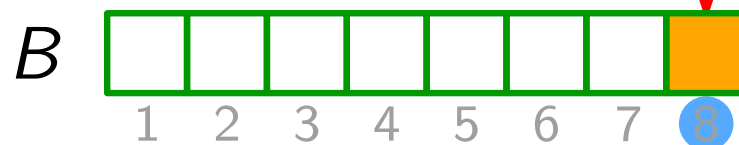
**Bsp:** 1a) Für jedes  $x$  in  $A$ , zähle die Anz. der Zahlen gleich  $x$



1b) Für jedes  $x$  in  $A$ , berechne die Anz. der Zahlen  $\leq x$



2) Schreibe jedes  $x$  in  $A$  direkt an die richtige Position in  $B$



# CountingSort

- Idee:**
- 1) für jedes  $x$  in der Eingabe: zähle die Anzahl der Zahlen  $\leq x$
  - 2) benütze diese Information um  $x$  im Ausgabefeld direkt an die richtige Position zu schreiben

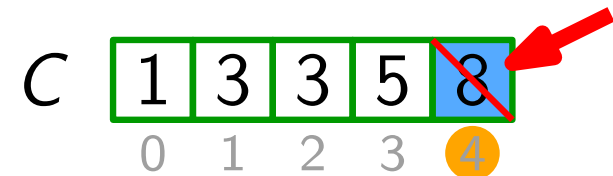
**Variable:**

$A$	Eingabefeld	$C$	Rechenfeld
$B$	Ausgabefeld	$k$	begrenzt das <i>Universum</i> : $\{0, \dots, k\}$

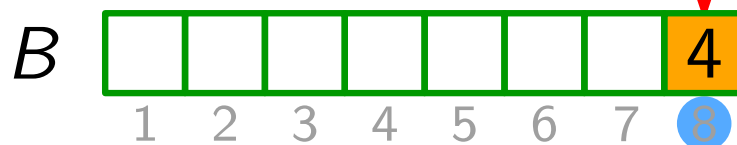
**Bsp:** 1a) Für jedes  $x$  in  $A$ , zähle die Anz. der Zahlen gleich  $x$



1b) Für jedes  $x$  in  $A$ , berechne die Anz. der Zahlen  $\leq x$



2) Schreibe jedes  $x$  in  $A$  direkt an die richtige Position in  $B$



# CountingSort

- Idee:**
- 1) für jedes  $x$  in der Eingabe: zähle die Anzahl der Zahlen  $\leq x$
  - 2) benütze diese Information um  $x$  im Ausgabefeld direkt an die richtige Position zu schreiben

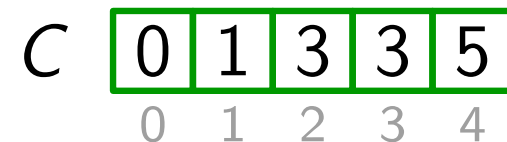
**Variable:**

$A$	Eingabefeld	$C$	Rechenfeld
$B$	Ausgabefeld	$k$	begrenzt das <i>Universum</i> : $\{0, \dots, k\}$

**Bsp:** 1a) Für jedes  $x$  in  $A$ , zähle die Anz. der Zahlen gleich  $x$



1b) Für jedes  $x$  in  $A$ , berechne die Anz. der Zahlen  $\leq x$



2) Schreibe jedes  $x$  in  $A$  direkt an die richtige Position in  $B$

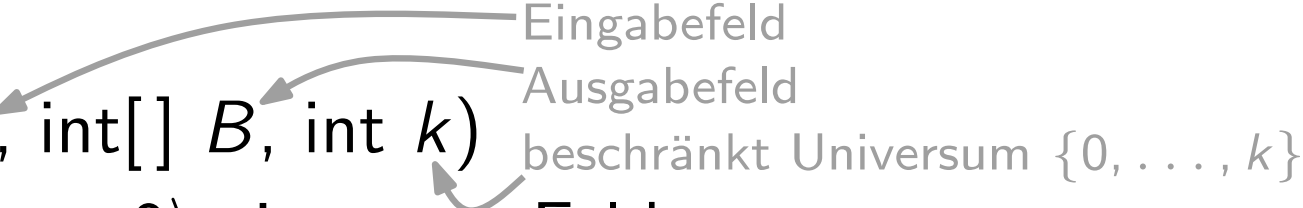


CountingSort ist *stabil*!


# CountingSort

- Plan:**
- 1a) Für jedes  $x$  in  $A$ , zähle die Anz. der Zahlen gleich  $x$
  - 1b) Für jedes  $x$  in  $A$ , berechne die Anz. der Zahlen  $\leq x$
  - 2) Schreibe jedes  $x$  in  $A$  direkt an die richtige Position in  $B$


CountingSort(int[] *A*, int[] *B*, int *k*)



sei  $C[0..k] = \langle 0, 0, \dots, 0 \rangle$  ein neues Feld


**for**  $j = 1$  **to**  $A.length$  **do**  // (1a)

//  $C[i]$  enthält jetzt die Anz. der Elem. gleich  $i$  in  $A$

**for**  $i = 1$  **to**  $k$  **do**  // (1b)

//  $C[i]$  enthält jetzt die Anz. der Elem.  $\leq i$  in  $A$

**for**  $j = A.length$  **downto**  $1$  **do**

┌  // (2)

## Aufgabe:

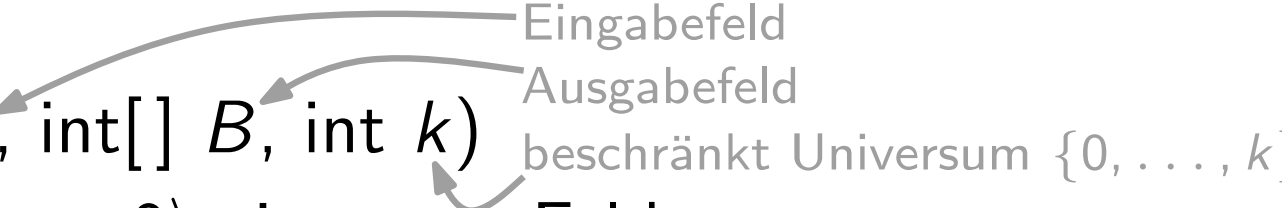
Fülle die Felder mit Code, der obige Idee umsetzt!

# CountingSort

**Laufzeit:**  
 $O(n + k)$

- Plan:**
- 1a) Für jedes  $x$  in  $A$ , zähle die Anz. der Zahlen gleich  $x$
  - 1b) Für jedes  $x$  in  $A$ , berechne die Anz. der Zahlen  $\leq x$
  - 2) Schreibe jedes  $x$  in  $A$  direkt an die richtige Position in  $B$

CountingSort(int[]  $A$ , int[]  $B$ , int  $k$ )



sei  $C[0..k] = \langle 0, 0, \dots, 0 \rangle$  ein neues Feld

**for**  $j = 1$  **to**  $A.length$  **do**  $C[A[j]] = C[A[j]] + 1$  // (1a)

//  $C[i]$  enthält jetzt die Anz. der Elem. gleich  $i$  in  $A$

**for**  $i = 1$  **to**  $k$  **do**  $C[i] = C[i] + C[i - 1]$  // (1b)

//  $C[i]$  enthält jetzt die Anz. der Elem.  $\leq i$  in  $A$

**for**  $j = A.length$  **downto** 1 **do**

$B[C[A[j]]] = A[j]$

$C[A[j]] = C[A[j]] - 1$  // (2)

# RadixSort

(Jahr, Monat, Tag)



**Frage:** Gegeben Liste von Menschen mit deren Geburtstagen.  
Wie würden Sie die Liste nach Alter sortieren?

## Drei (?) Lösungen:

- Geburtstage in Anz. Tage seit 1.1.1970 umrechnen, dann vergleichsbasiertes Sortierverfahren verwenden.
- Spezielle Vergleichsroutine schreiben und in vergleichsbasiertes Sortierverfahren einbauen.
- Liste  $3 \times$  sortieren: je  $1 \times$  nach Jahr, Monat, Tag.

*Aber in welcher Reihenfolge??*

RadixSort( $A, s$ )      Anz. Stellen (hier: 3)

**Laufzeit?**

**for**  $i = 1$  **to**  $s$  **do**      [1 = Index der *niederwertigsten* (!) Stelle]  
    └ sortiere  $A$  *stabil* nach der  $i$ -ten Stelle

└ z.B. mit CountingSort

# Beispiel

Sortiere  $A = \langle 25, 13, 31, 23, 11, 37, 15 \rangle$ :

Gemäß RadixSort erst nach Einern, dann (stabil) nach Zehnern.

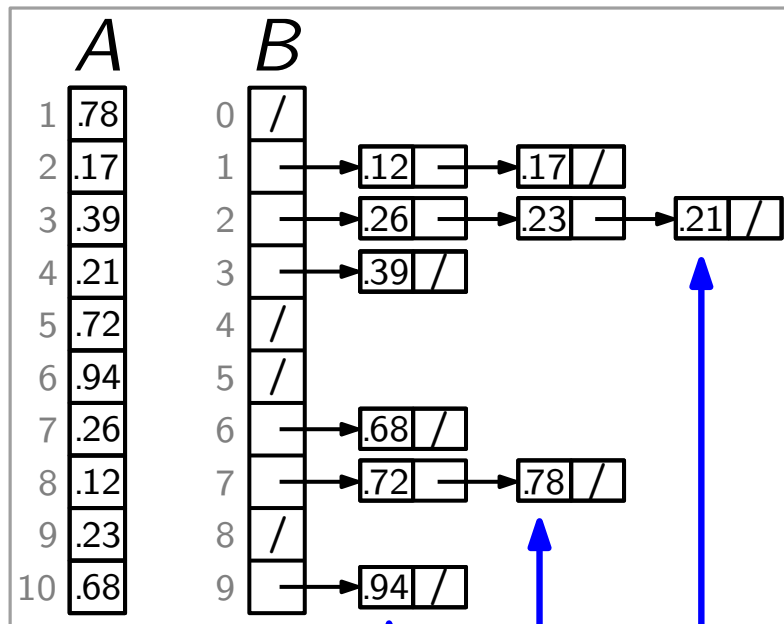
RadixSort( $A, s$ )

**for**  $i = 1$  **to**  $s$  **do**

$\perp$  sortiere  $A$  *stabil* nach der  $i$ -ten Stelle

# BucketSort

[CLRS]



(c) www.seafish.org

„Eimerinhalt“: Verkettete Liste von Elementen aus  $A$ .

Hilfsfeld  $B[0..n - 1]$ ;

jeder Eintrag entspricht einem „Eimer“ der Weite  $1/n$

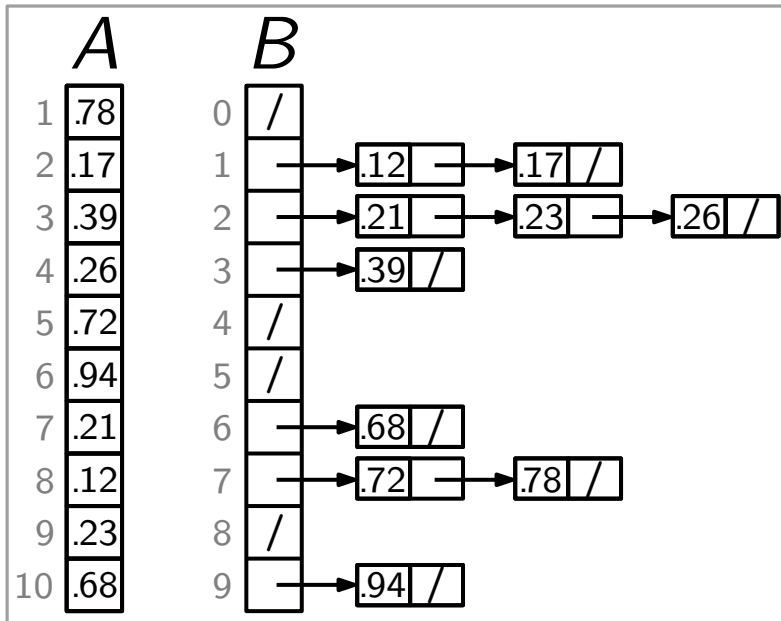
Eingabefeld  $A[1..n]$  enthält Zahlen,  
zufällig und gleichverteilt aus  $[0, 1)$  gezogen

[Im Bsp. auf 2 Nach-  
kommastellen gerundet!]



# BucketSort

[CLRS]



BucketSort(Feld A von Zahlen in  $[0, 1)$ )

$n = A.length$

lege Feld  $B[0..n - 1]$  von Listen an

**for**  $j = 1$  **to**  $n$  **do**

└ füge  $A[j]$  in Liste  $B[\lfloor n \cdot A[j] \rfloor]$  ein

**for**  $i = 0$  **to**  $n - 1$  **do**

└ sortiere Liste  $B[i] = [\frac{i}{n}, \frac{i+1}{n}) \cap A$

hänge  $B[0], \dots, B[n - 1]$  aneinander  
kopiere das Ergebnis nach  $A[1..n]$

**Korrektheit?** 2 Fälle: –  $A[i]$  und  $A[j]$  in der gleichen Liste  
–  $A[i]$  und  $A[j]$  in verschiedenen Listen

**Laufzeit?** – *erwartet*, hängt von den zufälligen Zahlen in A ab  
– hängt vom Sortieralgorithmus in Zeile 6 ab;  
wir nehmen InsertionSort: schnell auf kurzen Listen!

# Erwartete Laufzeit von BucketSort

$$T_{BS}(n) = \Theta(n) + \sum_{i=0}^{n-1} T_{IS}(n_i) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

$$\begin{aligned} E[T_{BS}(n)] &= E[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)] \\ &= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \end{aligned}$$

↑  $\Theta(n)$

**Behauptung:**  $E[n_i^2] \leq 2 - \frac{1}{n}$

*Beweis.* Def. Indikator-ZV  $X_j := 1$ , falls  $A[j]$  in Eimer  $i$  fällt. *fest!*

$$\Rightarrow n_i = \sum_{j=1}^n X_j \quad E[X_j] = \Pr[X_j = 1] = 1/n$$

$$\begin{aligned} \Rightarrow n_i^2 &= \left( \sum_{j=1}^n X_j \right)^2 = \sum_{j=1}^n \sum_{k=1}^n X_j X_k \\ &= \sum_{j=1}^n X_j^2 + \sum_{j=1}^n \sum_{k \neq j} X_j X_k \end{aligned}$$

# Erwartete Laufzeit von BucketSort

Behauptung:  
 $E[n_i^2] \leq 2 - \frac{1}{n}$

Es gilt  $n_i^2 = \sum_{j=1}^n X_j^2 + \sum_{j=1}^n \sum_{k \neq j} X_j X_k$

$$\Rightarrow E[n_i^2] = \sum_{j=1}^n E[X_j^2] + \sum_{j=1}^n \sum_{k \neq j} E[X_j X_k]$$

Behandle die beiden Typen von Erwartungswerten getrennt:

$$\begin{aligned} E[X_j^2] &= 1 \cdot \Pr[X_j^2 = 1] + 0 \cdot \Pr[X_j^2 = 0] && \text{unabhängig von } j! \\ &= 1 \cdot \Pr[X_j = 1] + 0 \cdot \Pr[X_j = 0] = 1 \cdot \frac{1}{n} + 0 = \frac{1}{n} \end{aligned}$$

$$E[X_j X_k] = E[X_j] \cdot E[X_k] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$

für  $j \neq k$  sind  $X_j$  und  $X_k$  unabhängig

unabh. von  $j$  und  $k$ !

Fasse die Zwischenergebnisse zusammen:

$$\begin{aligned} E[n_i^2] &= \sum_{j=1}^n E[X_j^2] + \sum_{j=1}^n \sum_{k \neq j} E[X_j X_k] \\ &= n \cdot \frac{1}{n} + n \cdot (n-1) \cdot \frac{1}{n^2} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n} \end{aligned}$$

□

# Zusammenfassung

- Jedes *vergleichsbasierte* Sortiervverfahren braucht im schlechtesten Fall  $\Omega(n \log n)$  Vergleiche für  $n$  Zahlen.
- **CountingSort** sortiert Zahlen in  $\{0, \dots, k\}$  (*stabil!*)  
Laufzeit für  $n$  Zahlen:  $O(n + k)$
- **RadixSort** sortiert  $s$ -stellige  $b$ -adische Zahlen  
Laufzeit für  $n$  Zahlen:  $O(s \cdot (n + b))$
- **BucketSort** sortiert gleichverteilte zufällige Zahlen  
*erwartete* Laufzeit für  $n$  Zahlen:  $O(n)$

**Bem.** Die Idee mit den (gleichgroßen) Eimern ist natürlich nicht nur auf Zufallszahlen beschränkt, aber hier lässt sie sich hübsch analysieren.