Universelle Familien von Hashfunktionen

Sei $U = \{0, \ldots, p-1\}$, wobei p eine Primzahl ist.
Es sei $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$.
Wir definieren

$$\mathcal{H} = \{\, h_{a,b} \mid 1 \leq a < p,\ 0 \leq b < p \}$$

Theorem

$\mathcal{H}$ ist eine universelle Familie von Hashfunktionen.

Es seien $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Wir wollen zunächst zeigen, daß die Funktion

$$f : (a, b) \mapsto (ax + b \bmod p, ay + b \bmod p)$$

für $a, b \in \{0, \ldots, p-1\}$ injektiv und somit auch bijektiv ist.

$$\begin{aligned}
&(ax + b \bmod p, ay + b \bmod p) = (a'x + b' \bmod p, a'y + b' \bmod p) \\
\Leftrightarrow &(ax + b - b' \bmod p, ay + b - b' \bmod p) = (a'x \bmod p, a'y \bmod p) \\
\Leftrightarrow &(b - b' \bmod p, b - b' \bmod p) = ((a' - a)x \bmod p, (a' - a)y \bmod p) \\
\Leftrightarrow &a' = a \wedge b' = b
\end{aligned}$$

Es seien $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Wir wollen zunächst zeigen, daß die Funktion

$$f : (a, b) \mapsto (ax + b \bmod p, ay + b \bmod p)$$

für $a, b \in \{0, \ldots, p-1\}$ injektiv und somit auch bijektiv ist.

$$\begin{aligned}
&(ax + b \bmod p, ay + b \bmod p) = (a'x + b' \bmod p, a'y + b' \bmod p) \\
\Leftrightarrow &(ax + b - b' \bmod p, ay + b - b' \bmod p) = (a'x \bmod p, a'y \bmod p) \\
\Leftrightarrow &(b - b' \bmod p, b - b' \bmod p) = ((a' - a)x \bmod p, (a' - a)y \bmod p) \\
\Leftrightarrow &a' = a \wedge b' = b
\end{aligned}$$

RWTH AACHEN UNIVERSITY

Nach wie vor gelte $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Für wieviele Paare (a, b) haben $c_x := ax + b \bmod p$ und $c_y := ay + b \bmod p$ den gleichen Rest modulo m?

Wir haben auf der letzten Folie bewiesen, daß sich für jedes Paar (a, b) ein eindeutiges Paar (c_x, c_y) ergibt. Für ein festes c_x gibt es nur

$$\lceil p/m \rceil - 1 = \left\lfloor \frac{p+m-1}{m} \right\rfloor - 1 \leq \frac{p-1}{m}$$

viele mögliche Werte von c_y mit $c_x \equiv c_y \mod m$ und $c_x \neq c_y$.

Weil p verschiedene Werte für c_x existieren, gibt es insgesamt höchstens $p(p-1)/m$ Paare der gesuchten Art.

$$\frac{|\{\, h \in \mathcal{H} \mid h(x) = h(y) \,\}|}{|\mathcal{H}|} \leq \frac{p(p-1)/m}{p(p-1)} \leq \frac{1}{m}$$

RWTH AACHEN UNIVERSITY

Nach wie vor gelte $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Für wieviele Paare (a, b) haben $c_x := ax + b \bmod p$ und $c_y := ay + b \bmod p$ den gleichen Rest modulo m?

Wir haben auf der letzten Folie bewiesen, daß sich für jedes Paar (a, b) ein eindeutiges Paar (c_x, c_y) ergibt. Für ein festes c_x gibt es nur

$$\lceil p/m \rceil - 1 = \left\lfloor \frac{p+m-1}{m} \right\rfloor - 1 \leq \frac{p-1}{m}$$

viele mögliche Werte von c_y mit $c_x \equiv c_y \mod m$ und $c_x \neq c_y$.

Weil p verschiedene Werte für c_x existieren, gibt es insgesamt höchstens $p(p-1)/m$ Paare der gesuchten Art.

$$\frac{|\{\, h \in \mathcal{H} \mid h(x) = h(y) \,\}|}{|\mathcal{H}|} \leq \frac{p(p-1)/m}{p(p-1)} \leq \frac{1}{m}$$

RWTH AACHEN UNIVERSITY

Nach wie vor gelte $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Für wieviele Paare (a, b) haben $c_x := ax + b \bmod p$ und $c_y := ay + b \bmod p$ den gleichen Rest modulo m?

Wir haben auf der letzten Folie bewiesen, daß sich für jedes Paar (a, b) ein eindeutiges Paar (c_x, c_y) ergibt. Für ein festes c_x gibt es nur

$$\lceil p/m \rceil - 1 = \left\lfloor \frac{p+m-1}{m} \right\rfloor - 1 \leq \frac{p-1}{m}$$

viele mögliche Werte von c_y mit $c_x \equiv c_y \mod m$ und $c_x \neq c_y$.

Weil p verschiedene Werte für c_x existieren, gibt es insgesamt höchstens $p(p-1)/m$ Paare der gesuchten Art.

$$\frac{|\{\, h \in \mathcal{H} \mid h(x) = h(y) \,\}|}{|\mathcal{H}|} \leq \frac{p(p-1)/m}{p(p-1)} \leq \frac{1}{m}$$

RWTH AACHEN UNIVERSITY

Nach wie vor gelte $x, y \in \{0, \ldots, p-1\}$, $x \neq y$.
Für wieviele Paare (a, b) haben $c_x := ax + b \bmod p$ und $c_y := ay + b \bmod p$ den gleichen Rest modulo m?

Wir haben auf der letzten Folie bewiesen, daß sich für jedes Paar (a, b) ein eindeutiges Paar (c_x, c_y) ergibt. Für ein festes c_x gibt es nur

$$\lceil p/m \rceil - 1 = \left\lfloor \frac{p+m-1}{m} \right\rfloor - 1 \leq \frac{p-1}{m}$$

viele mögliche Werte von c_y mit $c_x \equiv c_y \mod m$ und $c_x \neq c_y$.

Weil p verschiedene Werte für c_x existieren, gibt es insgesamt höchstens $p(p-1)/m$ Paare der gesuchten Art.

$$\frac{|\{\, h \in \mathcal{H} \mid h(x) = h(y) \,\}|}{|\mathcal{H}|} \leq \frac{p(p-1)/m}{p(p-1)} \leq \frac{1}{m}$$

RWTH AACHEN UNIVERSITY

Übersicht

Übersicht

Mengen

Der abstrakte Datentyp Menge sollte folgende Operationen unterstützen:

- $x \in M$?
- $M \to M \cup \{x\}$
- $M \to M \setminus \{x\}$
- $M = \emptyset$?
- wähle irgendein $x \in M$

Möglicherweise auch

- $M_1 \to M_2 \cup M_3$
- $M_1 \to M_2 \cap M_3$
- $M_1 \to M_2 \setminus M_3$
- ...

Mengen

Der abstrakte Datentyp Menge sollte folgende Operationen unterstützen:

- $x \in M$?
- $M \to M \cup \{x\}$
- $M \to M \setminus \{x\}$
- $M = \emptyset$?
- wähle irgendein $x \in M$

Möglicherweise auch

- $M_1 \to M_2 \cup M_3$
- $M_1 \to M_2 \cap M_3$
- $M_1 \to M_2 \setminus M_3$
- ...

Mengen können durch assoziative Arrays implementiert werden:

Java

```
public class Set⟨K⟩ {
  private final ADMap < K, ?> h;
  public Set() { h = new Hashtable⟨K, Integer⟩(); }
  public Set(ADMap < K, ?> m) { h = m; }
  public void insert(K k) { h.insert(k, null); }
  public void delete(K k) { h.delete(k); }
  public void union(Set⟨K⟩ U) {
    SimpleIterator⟨K⟩ it;
    for(it = U.iterator(); it.more(); it.step())
      insert(it.key()); }
  public boolean iselement(K k) { return h.containsKey(k); }
  public SimpleIterator⟨K⟩ iterator() {
  return h.simpleiterator(); }
  public List⟨K⟩ list() { return h.list(); }
```

ACHEN ERSITY

Übersicht

Insertion Sort

Wir sortieren ein unsortiertes Array, indem wir wiederholt Elemente in ein bereits sortiertes Teilarray einfügen.

Eingabe:

12	73	36	71	79	67	3	17	32	31	14	14	33	4	74	23

Ausgabe:

3	4	12	14	14	17	23	31	32	33	36	67	71	73	74	79

Insertion Sort

12	73	36	71	79	67	3	17	32	31	14	14	33	4	74	23

Algorithmus

procedure insertionsort(n) :
  for i $= 2, ..., n$ **do**
    j := i;
    while j ≥ 2 **and** a[j $- 1$] $>$ a[j] **do**
      vertausche a[j $- 1$] und a[j];
      j := j $- 1$
    od
  od

Insertionsort

RWTH AACHEN UNIVERSITY

Inversionen

Die Laufzeit von Insertion Sort ist $O(n^2)$.

Definition

Sei $\pi \in S_n$ eine Permutation. Die Menge der Inversionen von π ist

$$I(\pi) = \{ (i,j) \in \{1,\ldots,n\}^2 \mid i < j \text{ und } \pi(i) > \pi(j) \}.$$

Beim Sortieren durch Einfügen werden Schlüssel nur durch Vertauschungen benachbarter Elemente bewegt.
So eine Vertauschung verringert die Anzahl der Inversionen höchstens um eins.
Wenn die Eingabe genau falschherum sortiert ist, hat Insertion Sort die Laufzeit $\Theta(n^2)$.
Was ist die durchschnittliche Laufzeit?

Inversionen

Die Laufzeit von Insertion Sort ist $O(n^2)$.

Definition

Sei $\pi \in S_n$ eine Permutation. Die Menge der Inversionen von π ist

$$I(\pi) = \{ (i,j) \in \{1,\ldots,n\}^2 \mid i < j \text{ und } \pi(i) > \pi(j) \}.$$

Beim Sortieren durch Einfügen werden Schlüssel nur durch Vertauschungen benachbarter Elemente bewegt.
So eine Vertauschung verringert die Anzahl der Inversionen höchstens um eins.
Wenn die Eingabe genau falschherum sortiert ist, hat Insertion Sort die Laufzeit $\Theta(n^2)$.
Was ist die durchschnittliche Laufzeit?

RWTH AACHEN UNIVERSITY

Inversionen

Die Laufzeit von Insertion Sort ist $O(n^2)$.

Definition

Sei $\pi \in S_n$ eine Permutation. Die Menge der Inversionen von π ist

$$I(\pi) = \{ (i,j) \in \{1,\ldots,n\}^2 \mid i < j \text{ und } \pi(i) > \pi(j) \}.$$

Beim Sortieren durch Einfügen werden Schlüssel nur durch Vertauschungen benachbarter Elemente bewegt.
So eine Vertauschung verringert die Anzahl der Inversionen höchstens um eins.
Wenn die Eingabe genau falschherum sortiert ist, hat Insertion Sort die Laufzeit $\Theta(n^2)$.
Was ist die durchschnittliche Laufzeit?

Inversionen

Die Laufzeit von Insertion Sort ist $O(n^2)$.

Definition

Sei $\pi \in S_n$ eine Permutation. Die Menge der Inversionen von π ist

$$I(\pi) = \{ (i,j) \in \{1,\ldots,n\}^2 \mid i < j \text{ und } \pi(i) > \pi(j) \}.$$

Beim Sortieren durch Einfügen werden Schlüssel nur durch Vertauschungen benachbarter Elemente bewegt.
So eine Vertauschung verringert die Anzahl der Inversionen höchstens um eins.
Wenn die Eingabe genau falschherum sortiert ist, hat Insertion Sort die Laufzeit $\Theta(n^2)$.
Was ist die durchschnittliche Laufzeit?

RWTH AACHEN UNIVERSITY

Inversionen

Theorem

Eine zufällig gewählte Permutation $\pi \in S_n$ hat im Erwartungswert $n(n-1)/4$ Inversionen.

Beweis.

Es gibt $n(n-1)/2$ viele Paare (i,j) mit $1 \leq i < j \leq n$.
Wegen

$$\Pr[\pi(i) > \pi(j)] = \frac{1}{2} \text{ falls } i \neq j$$

gilt

$$E(|I(\pi)|) = \sum_{i<j} \Pr[\pi(i) > \pi(j)] = \frac{n(n-1)}{4}.$$

□

ACHEN ERSITY

Inversionen

Theorem

Eine zufällig gewählte Permutation $\pi \in S_n$ hat im Erwartungswert $n(n-1)/4$ Inversionen.

Beweis.

Es gibt $n(n-1)/2$ viele Paare (i,j) mit $1 \leq i < j \leq n$.
Wegen

$$\Pr[\pi(i) > \pi(j)] = \frac{1}{2} \text{ falls } i \neq j$$

gilt

$$E(|I(\pi)|) = \sum_{i<j} \Pr[\pi(i) > \pi(j)] = \frac{n(n-1)}{4}.$$

□

ACHEN ERSITY

Inversionen

Theorem

Jedes Sortierverfahren, das Schlüssel nur durch Vertauschungen benachbarter Elemente bewegt, benötigt im Durchschnitt $\Omega(n^2)$ Zeit.

Folgerung:

Wenn wir schneller sein wollen, müssen Schlüssel über weite Strecken bewegt werden.

Inversionen

Theorem

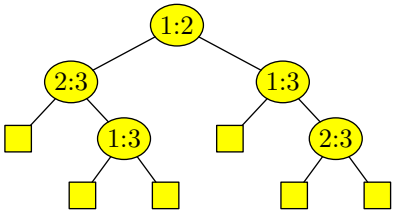
Jedes Sortierverfahren, das Schlüssel nur durch Vertauschungen benachbarter Elemente bewegt, benötigt im Durchschnitt $\Omega(n^2)$ Zeit.

Folgerung:

Wenn wir schneller sein wollen, müssen Schlüssel über weite Strecken bewegt werden.

Vergleichsbäume

Insertion Sort mit $n = 3$:



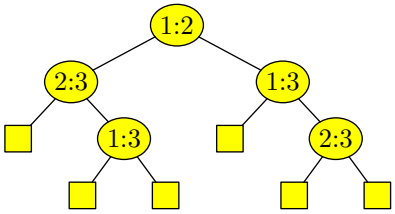
Jeder vergleichsbasierte Sortieralgorithmus hat einen Vergleichsbaum.

Die Wurzel ist der erste Vergleich.

Links folgen die Vergleiche beim Ergebnis kleiner.
Rechts folgen die Vergleiche beim Ergebnis größer.

Vergleichsbäume

Insertion Sort mit $n = 3$:



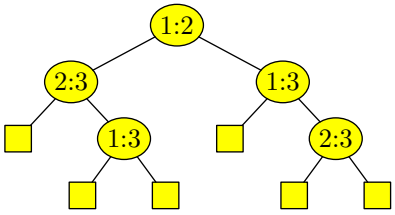
Jeder vergleichsbasierte Sortieralgorithmus hat einen Vergleichsbaum.

Die Wurzel ist der erste Vergleich.

Links folgen die Vergleiche beim Ergebnis kleiner.
Rechts folgen die Vergleiche beim Ergebnis größer.

Vergleichsbäume

Insertion Sort mit $n = 3$:



Jeder vergleichsbasierte Sortieralgorithmus hat einen Vergleichsbaum.

Die Wurzel ist der erste Vergleich.

Links folgen die Vergleiche beim Ergebnis kleiner.
Rechts folgen die Vergleiche beim Ergebnis größer.

RWTH AACHEN UNIVERSITY

Vergleichsbäume

Lemma

Der Vergleichsbaum eines vergleichsbasierten Sortieralgorithmus hat mindestens $n!$ Blätter.

Beweis.

Wenn zwei Permutationen zum gleichen Blatt führen, wird eine von ihnen falsch sortiert.

Es gibt aber $n!$ viele Permutationen. □

Vergleichsbäume

Lemma

Der Vergleichsbaum eines vergleichsbasierten Sortieralgorithmus hat mindestens $n!$ Blätter.

Beweis.

Wenn zwei Permutationen zum gleichen Blatt führen, wird eine von ihnen falsch sortiert.

Es gibt aber $n!$ viele Permutationen. □

Theorem

Jeder vergleichsbasierte Algorithmus benötigt für das Sortieren einer zufällig permutierten Eingabe im Erwartungswert mindestens

$$\log(n!) = n \log n - n \log e - \frac{1}{2} \log n + O(1)$$

viele Vergleiche.

Beweis.

Sei T ein entsprechender Vergleichsbaum. Die mittlere Pfadlänge zu einem Blatt ist am kleinsten, wenn der Baum balanziert ist.
In diesem Fall ist die Höhe $\log(n!)$. Stirling-Formel:

$$n! = \frac{1}{\sqrt{2\pi n}} \left(\frac{n}{e}\right)^n (1 + O(n^{-1})).$$

ACHEN ERSITY

Theorem

Jeder vergleichsbasierte Algorithmus benötigt für das Sortieren einer zufällig permutierten Eingabe im Erwartungswert mindestens

$$\log(n!) = n \log n - n \log e - \frac{1}{2} \log n + O(1)$$

viele Vergleiche.

Beweis.

Sei T ein entsprechender Vergleichsbaum. Die mittlere Pfadlänge zu einem Blatt ist am kleinsten, wenn der Baum balanziert ist.
In diesem Fall ist die Höhe $\log(n!)$. Stirling-Formel:

$$n! = \frac{1}{\sqrt{2\pi n}} \left(\frac{n}{e}\right)^n \left(1 + O(n^{-1})\right).$$

ACHEN
ERSITY

□

Mergesort

Wir sortieren ein unsortiertes Array durch einen Divide-and-Conquer–Algorithmus:

Eingabe:

12	73	36	71	79	67	3	17	32	31	14	14	33	4	74	23

Ausgabe:

3	4	12	14	14	17	23	31	32	33	36	67	71	73	74	79

Frage: Wie zerteilen wir die Eingabe in zwei unabhängige Teilprobleme?

Mergesort

Wir sortieren ein unsortiertes Array durch einen Divide-and-Conquer–Algorithmus:

Eingabe:

12	73	36	71	79	67	3	17	32	31	14	14	33	4	74	23

Ausgabe:

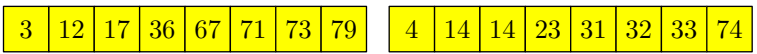
3	4	12	14	14	17	23	31	32	33	36	67	71	73	74	79

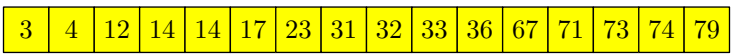
Frage: Wie zerteilen wir die Eingabe in zwei unabhängige Teilprobleme?

Mergesort

Eine einfache Strategie: Teilen in der Mitte.

1. Teile das Array in der Mitte.
2. Sortiere beide Hälften.
3. Mische beide in eine sortierte Folge.



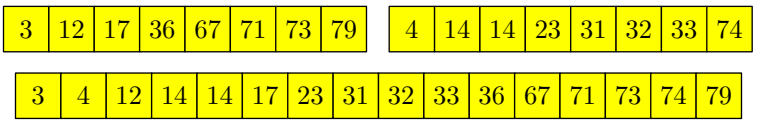


RWTH AACHEN UNIVERSITY

Mergesort

Eine einfache Strategie: Teilen in der Mitte.

1. Teile das Array in der Mitte.
2. Sortiere beide Hälften.
3. Mische beide in eine sortierte Folge.

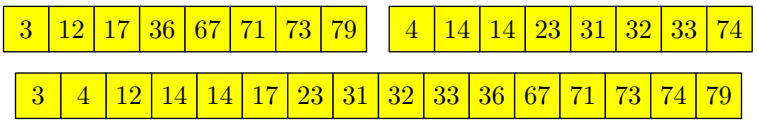


RWTH AACHEN UNIVERSITY

Mergesort

Eine einfache Strategie: Teilen in der Mitte.

1. Teile das Array in der Mitte.
2. Sortiere beide Hälften.
3. Mische beide in eine sortierte Folge.

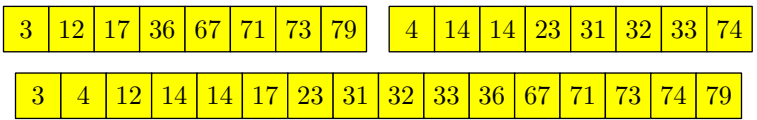


RWTH AACHEN UNIVERSITY

Mergesort

Eine einfache Strategie: Teilen in der Mitte.

1. Teile das Array in der Mitte.
2. Sortiere beide Hälften.
3. Mische beide in eine sortierte Folge.

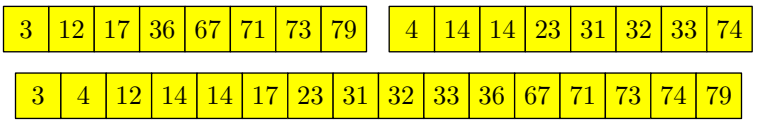


RWTH AACHEN UNIVERSITY

Mergesort

Eine einfache Strategie: Teilen in der Mitte.

1. Teile das Array in der Mitte.
2. Sortiere beide Hälften.
3. Mische beide in eine sortierte Folge.

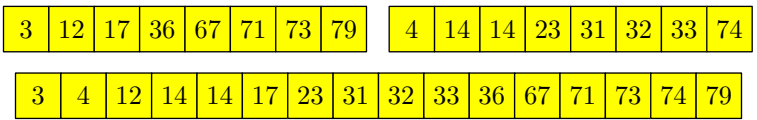


RWTH AACHEN UNIVERSITY

Mergesort

Eine einfache Strategie: Teilen in der Mitte.

1. Teile das Array in der Mitte.
2. Sortiere beide Hälften.
3. Mische beide in eine sortierte Folge.



RWTH AACHEN UNIVERSITY

Mergesort

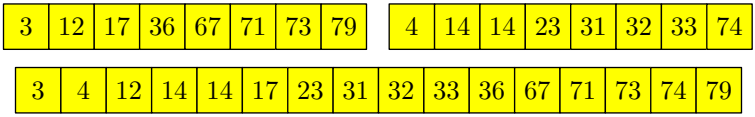
RWTH AACHEN UNIVERSITY

Bottom-up Mergesort

RWTH AACHEN UNIVERSITY

Mergesort

Mischen ist der schwierige Teil.



Algorithmus, der $a[l], \ldots, a[m-1]$ mit $a[m], \ldots, a[r]$ mischt:

Algorithmus

```
i := l; j := m; k := l;
while k ≤ r do
  if a[i] ≤ a[j] and i < m or j > r
  then b[k] := a[i]; k := k + 1; i := i + 1
  else b[k] := a[j]; k := k + 1; j := j + 1 fi
od;
for i = l, ..., r do a[k] := b[k] od
```