Hashing

Du: versuchst möglichst guten Algo zu bauen

Ich (Gegner): wähle die Eingabe für den Algo

Sichtweise 1: Worst-Case

- Schritt 1: Du legst fest, was der Algo tut
  (hier: Wahl der Hashfunktion)

- Schritt 2: Ich wähle Worst-Case Eingabe (für diese Hashfkt.)

  ~> Es gibt keine gute Hashfkt.

  ~> $\forall h \exists$ Eingabe die sehr schlecht ist

Sichtweise 2: Oblivious Adversary

- Schritt 1: Du legst Algo fest, dieser beinhalten zufällige Entscheidungen

- Schritt 2: Ich wähle Worst-Case Eingabe, kenne deinen Algo, aber kenne das Ergebnis der zufallsentsch. nicht

~> Universelle Familie von Hashfunktionen H

~> $\forall I \in$ Eingaben: Die meisten $h \in H$ sind gut für I (wenig Kollisionen)

~> Keine Aussage ob eine einzelne Hashfunktion gut ist.

Anmerkung:

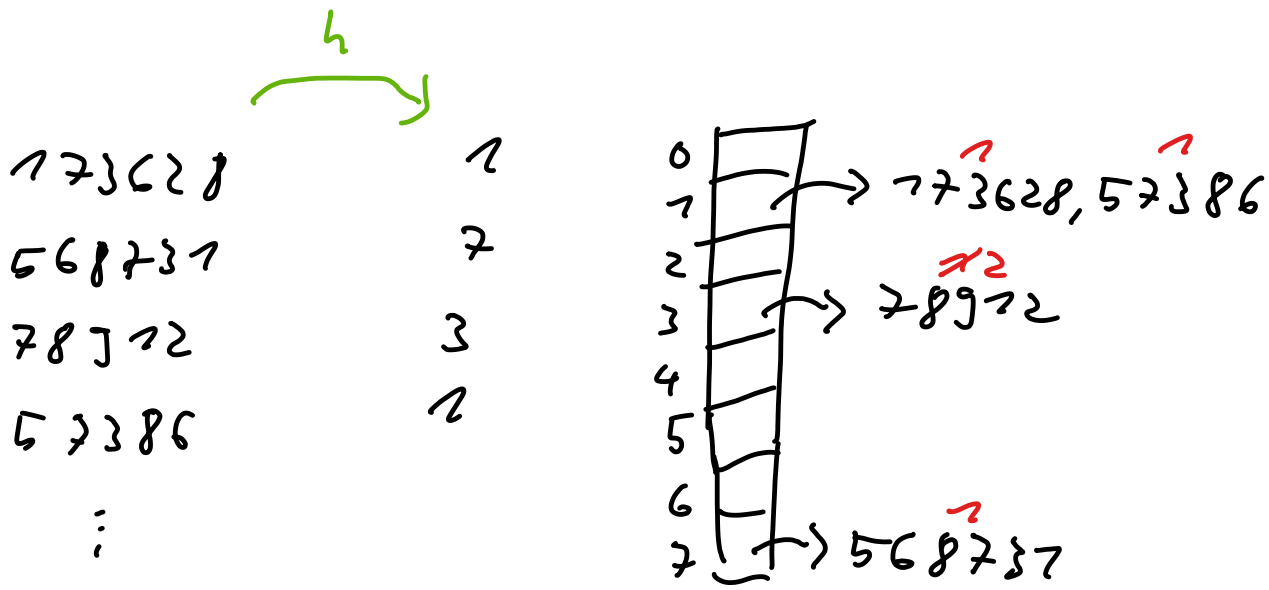
- Ich bin nicht wirklich euer Gegner, der euch böses will.
- Das ist aber trotzdem eine gute Vorstellung, um möglichst starke Garantien zu bekommen.

Sichtweise 3: Praxis

- Ich: Zeige dir ein paar Eingaben und verspreche, dass Eingaben in der Zukunft irgendwie ähnlich sind

- Du: Wähle Algo der für diese Eingaben super ist.

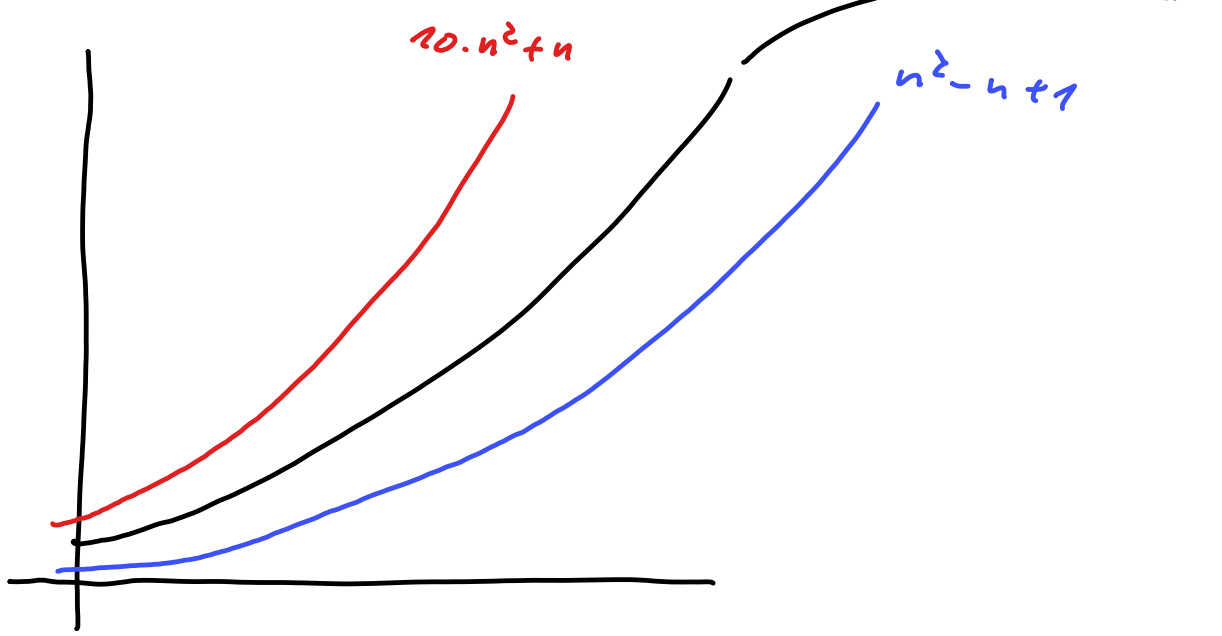
→ Es gibt eine Reihe erprobter Hashfunktionen, die in der Praxis gut sind.



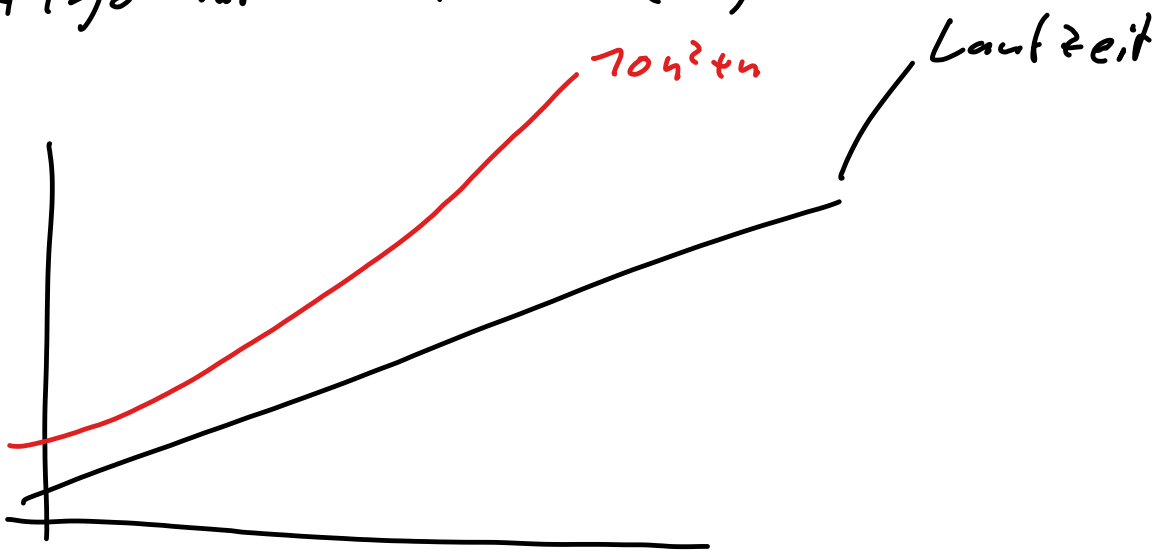
78912 }

Wir verwenden eine Hashmap M, bei der für einen Schlüssel K M[K] einen ganzzahligen Wert speichert.
Für jeden Schlüssel K in der Eingabe setze M[K] auf 1, wenn K bisher nicht in der Map ist und erhöhe M[K] um 1, wenn K schon drin ist
Zähle dabei mit, wie viele Schlüssel einen Wert > 1 erhalten.

Algo hat Laufzeit von $\Theta(n^2)$



Algo hat Laufzeit $O(n^2)$



Amortisierte Analyse ist eine Worst-Case Analyse

↳ Worst-Case über alle möglichen Eingaben.

Setting:

↳ Sequenz von Operationen einer DS

↳ manche der Operationen sind teuer

↳ viele sind günstig

Beispiel:

↳ n Operationen

↳ teure Operation Kostet $\Theta(n)$ } ⇒ naive Schlussfolgerung: $O(n^2)$

↳ alle bis auf eine Operation Kosten $\Theta(1)$

↝ Insgesamt nur $\Theta(n)$ Laufzeit

↝ Amortisiert $\Theta(1)$ pro Operation

Beispiel Dyn Array

↳ nur eine Operation: push_back

↳ push_back selten teuer

↳ vor jedem teuren push_back können immer viele günstige push_back