Keychain Shuffle

Problem H

Lukas Wolz und Michael Kohl

Seminar Algorithmen für Programmierwettbewerbe

Sommersemester 2020 - Universität Würzburg

Problem

Julia hat einen Schlüsselbund mit n Schlüsseln an einem einzigen Ring. Um den Überblick zu behalten, hat Sie diese Schlüssel immer in einer bestimmten Reihenfolge.
Aus Langeweile hat ihre Schwester Anna die Schlüssel an Julias Schlüsselbund eines Tages umsortiert.
Nun muss Julia alle Schlüssel in ihrem Bund wieder in die angestammte Reihenfolge bringen. Hierbei möchte Sie jedoch möglichst wenige Schlüssel aus dem Ring entnehmen und an anderer Stelle wieder einfügen müssen.

Beispiel

Merke: Schlüsselbund drehen ist möglich!

Ein- und Ausgabe

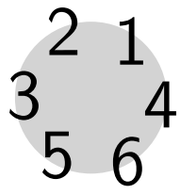
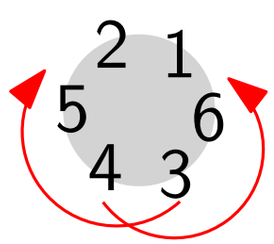
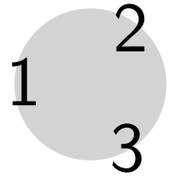
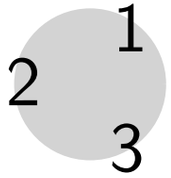
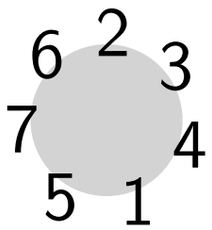
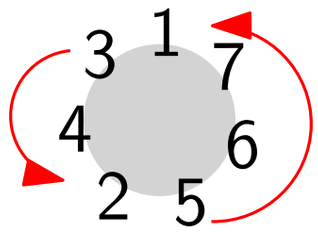
Eingabe:

- Eine Zeile mit einer Zahl n $(1 \leq n \leq 1000)$, die Anzahl der Schlüssel.
- Eine Zeile mit n Zahlen $k_1, \ldots, k_n$ $(1 \leq k_i \leq n$ für alle $i)$, die Original- bzw. Zielreihenfolge der Schlüssel am Bund.
- Eine Zeile mit n Zahlen $l_1, \ldots, l_n$ $(1 \leq l_i \leq n$ für alle $i)$, die geänderte Reihenfolge der Schlüssel am Bund.

Ausgabe:

- Eine Zahl z, welches für die minimale Anzahl notwendiger Verschiebeoperationen (Versetzungen) steht. Das Entnehmen eines Schlüssels an einer und das Einfügen jenes an einer anderen Stelle zählt dabei als eine Verschiebeoperation.

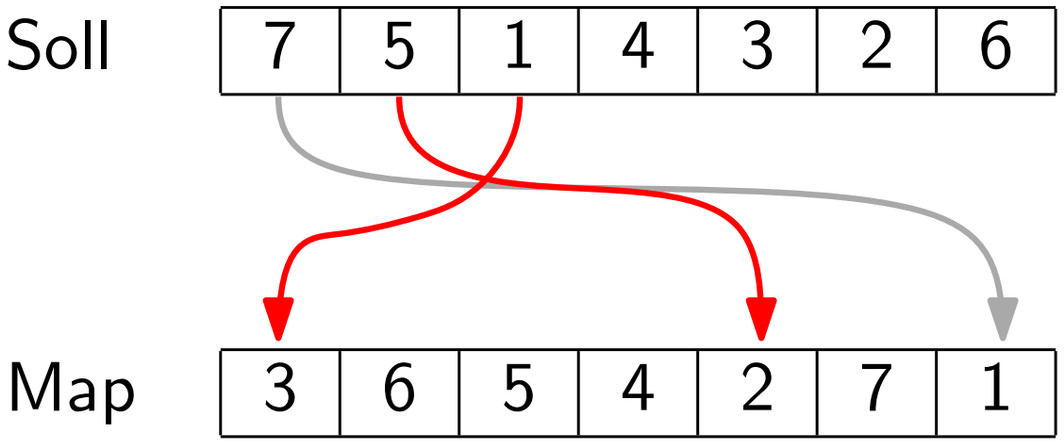
Weitere Beispiele

Original:		Geändert:		Ausgabe:
3 5 6 4 1 2		4 3 6 1 2 5		2
1 3 2		2 3 1		0
7 5 1 4 3 2 6		4 2 5 6 7 1 3		2

Remapping

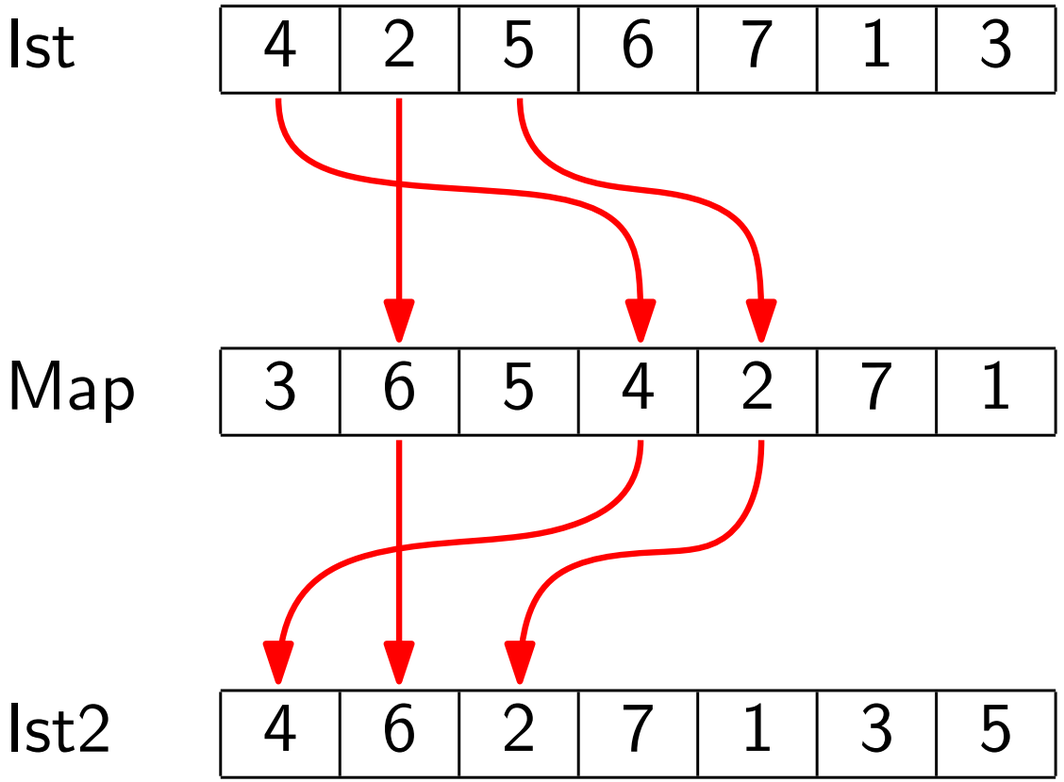
- Nachschlagen in Sollzustand $(k_1, \ldots, k_n)$ ist aufwendig
  - Suche vereinfachen?

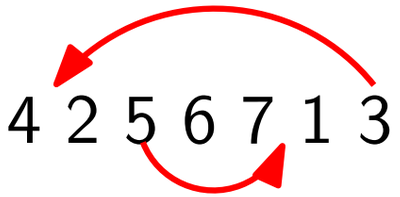
- Baue Map aus Sollzustand



Remapping

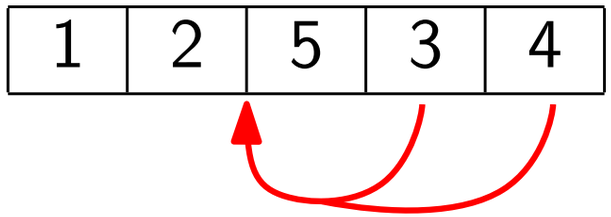
- Anwenden auf Ist-Zustand

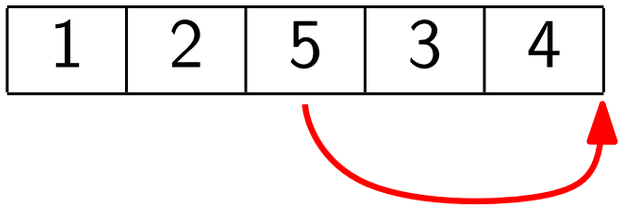




Verschiebeoperation

- Finde einen „guten“ Tausch

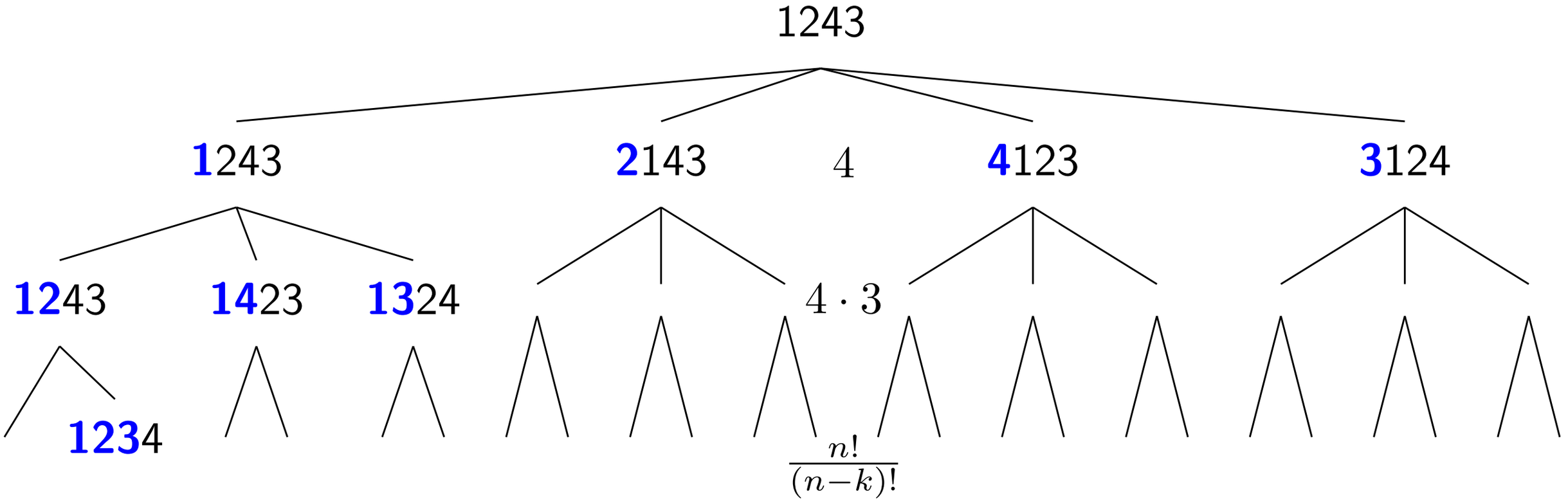




- Kein paarweises Tauschen

- Suchen mit Ausschneiden und Einfügen in $\mathcal{O}(n)$

Brute Force



- $n!$ Blätter : je $\mathcal{O}(n)$
- Worst Case Baum: $n \cdot \sum_{k=1}^{n-1} \frac{n!}{(n-k)!} = n \cdot \sum_{k=1}^{n-1} \frac{n!}{k!}$
- Gesamt $\mathcal{O}(n^n)$!!

Eine neue Sichtweise

Das Problem kann mit folgendem Lemma umformuliert werden:
Gegeben sei eine n-Permutationen π. Sei z die Anzahl minimal nötiger Verschiebeoperationen zum Sortieren (auf- oder abwärts) von π. Sei m die Länge der längsten monoton steigenden (bzw. fallenden) Teilfolge in π. Dann gilt:

$$z = n - m$$

Beispiel:

Permutation: 4 3 0 5 6 2 7 1

Beste Teilfolge:

Aufsteigend: 3 - 5 - 6 - 7

Absteigend: 4 - 3 - 2 - 1

Gegeben sei eine n-Permutationen π. Sei z die Anzahl minimal nötiger Verschiebeoperationen zum Sortieren (auf- oder abwärts) von π. Sei m die Länge der längsten monoton steigenden (bzw. fallenden) Teilfolge in π. Dann gilt:

$$z = n - m$$

Beweis :
$z \leq n - m$: Wenn man die nicht passenden $n - m$ Elemente aus der Folge entfernt und passend wieder einfügt, sortiert man die Folge mit $n-m$ Verschiebeoperationen.

$z \geq n - m$: Sei z gegeben. Entfernt man die entsprechenden z Elemente aus der Folge ergibt sich eine sortierte Teilfolge der Länge $n - z$, womit die Permutation sicherlich eine monoton steigende (bzw. fallende) Teilfolge dieser Länge enthält.

Folgen des Lemmas

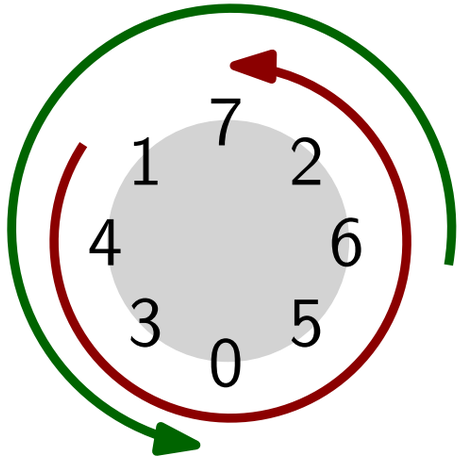
Sei P_π die n-elementige Menge aller n-Permutationen, welche man erhält, wenn man iterativ immer das erste Element an die letzte Stelle der Folge verschiebt.

Dann lässt sich das Keychain-Shuffle Problem wie folgt formulieren:

Sei π die dem Problem entsprechende Permutation. Für jedes $p \in P_\pi$ finde die Länge der längsten Teilfolgen (aufsteigend oder absteigend) m_p. Berechne $m = \max \{m_p \mid p \in P_\pi\}$. Gib $n - m$ zurück.

Beispiel:

Permutation: 4 3 0 5 6 2 7 1



Beste Teilfolge:

Aufsteigend: 1 - 3 - 5 - 6 - 7

Absteigend: 6 - 2 - 1 - 0

Merke: Zur Beantwortung der Aufgabenstellung müssen wir sogar nur die Länge der besten Teilfolge kennen!

Längste Teilfolge

- **0 1 4 6 2 3 5**
  - greedy 0 - 1 - 4 - 6
  - statt 0 - 1 - 2 - 3 - 5

- Führe Hilfsliste

0	1	2	3	5

  Überschreiben erlaubt!

- **Achtung:** Ungültige Teilfolge
  - nur Länge ist entscheidend

Längste Teilfolge

- **0 1 4 6 2 3 5**
  - Längste Teilfolge: 0 - 1 - 2 - 3 - 5
  - Start bei 4: | 4 | 5 |
  - Start bei 2: | 2 | 3 | 4 | 6 |
- Länge hängt vom gewählten Startpunkt ab
  - Betrachte alle Startpunkte
- Einsortieren ist aufwendig
  - Binäre Suche $\mathcal{O}(\log\ n)$

Algorithmus

- Einlesen & Remapping $\mathcal{O}(n)$
- Schleife für Startpunkte $\mathcal{O}(n)$
  - Schleife für Teilfolge $\mathcal{O}(n)$
    - Größer als letztes Element in der Hilfsliste: anhängen $\mathcal{O}(1)$
    - Größer als erstes Element in der Hilfsliste: ersetzen $\mathcal{O}(\log n)$
  - Merke Maximum m $\mathcal{O}(1)$

- Wiederhole für absteigende Teilfolgen $\mathcal{O}(n^2 \log n)$
- Ausgabe $\mathcal{O}(1)$

- **Gesamtlaufzeit:** $\mathcal{O}(n^2 \log n)$

Laufzeit

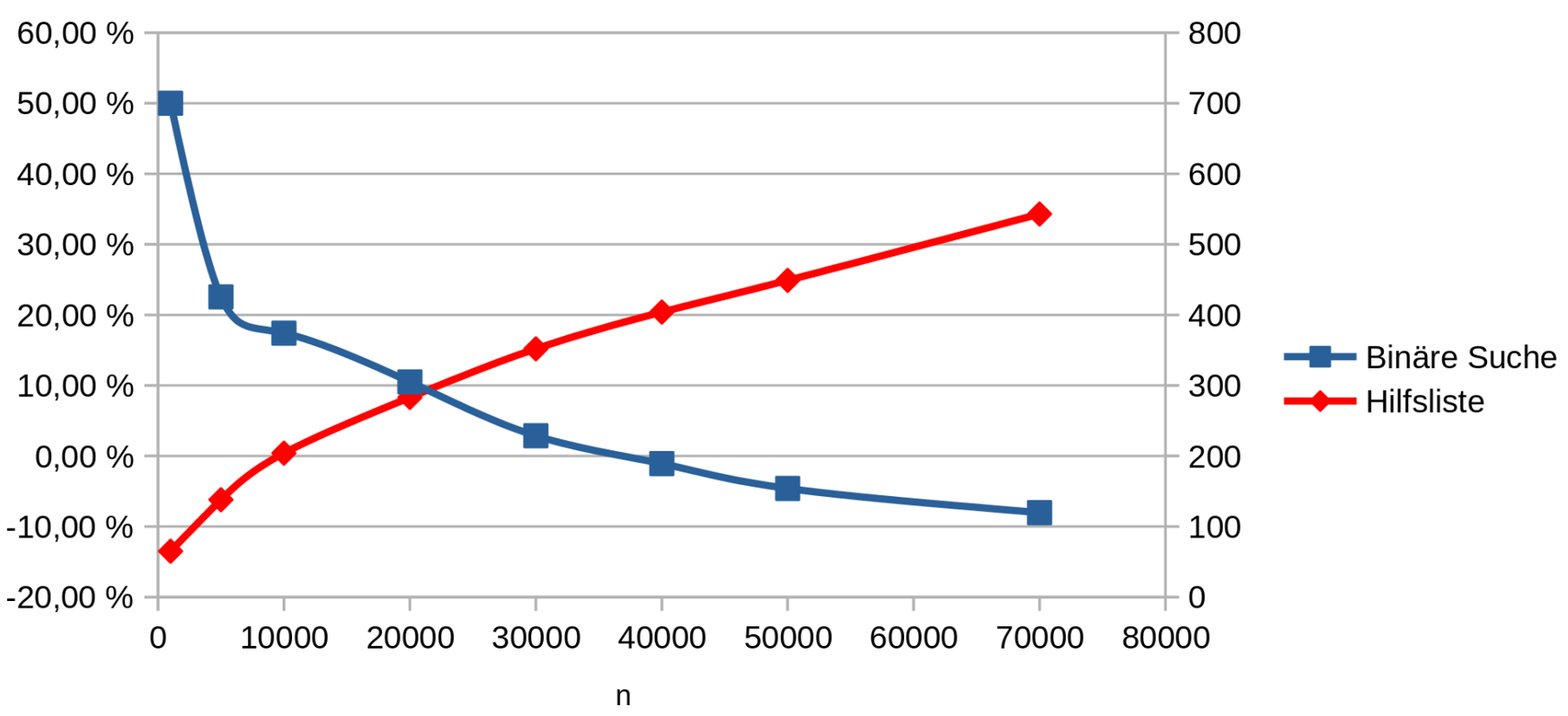
- Abbruchbedingungen

0	1	2	3	5	4

  - Abbrechen, falls von Startpunkt keine längere Teilfolge möglich

- Zufällige Eingabe mit $n = 1000$
  - Binäre Suche: 54 ms
  - Iterativ: 36 ms
  - etwa 935 Verschiebeoperationen ($\sigma \approx 2,5$)

Laufzeit



- Binäre Suche lohnt $n \approx 40.000$
- Hilfsliste wächst langsam

Tipps

- Datenstruktur
- Vorsicht mit Index
- Gegenrichtung

- Binäre Suche
  - `auto it = lower_bound(res.begin(), ...);`

Viel Erfolg