A&W 2026

G-13 Timo Stucki, LFW E 13

Week 9

Minitest 5

Randomised Algorithms

Primality Testing

Finding Duplicates – Hash Functions

Sort and Select

Infos

ETH zürich

Feel free to contact me:

- tistucki@student.ethz.ch
- Discord: timostucki

Material:

- timostucki.com

Minitest 5

Randomised Algorithms

Error Probability Reduction

Recap?

Error Probability Reduction

On Cheatsheet! 🎉

Error Reduction

- **Monte Carlo Repetition:** An N-fold repetition of a Monte Carlo algorithm for $N = 4\varepsilon^{-2} \ln \delta^{-1}$ boosts the success probability from $\frac{1}{2} + \varepsilon$ to $\geq 1 - \delta$ (by outputing the majority answer).
- **One-sided MC Repetition:** An N-fold repetition for $N = \varepsilon^{-1} \ln \delta^{-1}$ boosts the success probability of a one-sided error Monte Carlo algorithm from ε to $\geq 1 - \delta$.
- **Target Shooting:** If the target-shooting algorithm identifies a set $S \subseteq U$ with $N \geq 3\frac{|U|}{|S|}\varepsilon^{-2} \ln(2/\delta)$ trials, then with probability $\geq 1 - \delta$ the output lies in the interval

$$\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right].$$

Error Probability Reduction

Satz 2.72. Sei $\mathcal{A}$ ein randomisierter Algorithmus, der nie eine falsche Antwort gibt, aber zuweilen '???' ausgibt, wobei

$$\Pr[\mathcal{A}(I) \text{ korrekt}] \geq \varepsilon.$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder ein Wert verschieden von '???' ausgegeben wird (und $\mathcal{A}_\delta$ diesen Wert dann ebenfalls ausgibt) oder bis $N = \varepsilon^{-1} \ln \delta^{-1}$-mal '???' ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls '???' ausgibt), so gilt für den Algorithmus $\mathcal{A}_\delta$, dass

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq 1 - \delta.$$

Error Probability Reduction

Satz 2.74. Sei $\mathcal{A}$ ein randomisierter Algorithmus, der immer eine der beiden Antworten JA oder NEIN ausgibt, wobei

$$\Pr[\mathcal{A}(I) = \text{JA}] = 1 \quad \text{falls } I \text{ eine JA-Instanz ist,}$$

und

$$\Pr[\mathcal{A}(I) = \text{NEIN}] \geq \varepsilon \quad \text{falls } I \text{ eine NEIN-Instanz ist.}$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder der Wert NEIN ausgegeben wird (und $\mathcal{A}_\delta$ dann ebenfalls NEIN ausgibt) oder bis $N = \varepsilon^{-1} \ln \delta^{-1}$-mal JA ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls JA ausgibt), so gilt für alle Instanzen I

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq 1 - \delta.$$

Error Probability Reduction

Satz 2.72. Sei $\mathcal{A}$ ein randomisierter Algorithmus, der nie eine falsche Antwort gibt, aber zuweilen '???' ausgibt, wobei

$$\Pr[\mathcal{A}(I) \text{ korrekt}] \geq \varepsilon.$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder ein Wert verschieden von '???' ausgegeben wird (und $\mathcal{A}_\delta$ diesen Wert dann ebenfalls ausgibt) oder bis $N = \varepsilon^{-1} \ln \delta^{-1}$-mal '???' ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls '???' ausgibt), so gilt für den Algorithmus $\mathcal{A}_\delta$, dass

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq 1 - \delta.$$

Satz 2.74. Sei $\mathcal{A}$ ein randomisierter Algorithmus, der immer eine der beiden Antworten JA oder NEIN ausgibt, wobei

$$\Pr[\mathcal{A}(I) = \text{JA}] = 1 \quad \text{falls } I \text{ eine JA-Instanz ist,}$$

und

$$\Pr[\mathcal{A}(I) = \text{NEIN}] \geq \varepsilon \quad \text{falls } I \text{ eine NEIN-Instanz ist.}$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathcal{A}$ solange aufruft, bis entweder der Wert NEIN ausgegeben wird (und $\mathcal{A}_\delta$ dann ebenfalls NEIN ausgibt) oder bis $N = \varepsilon^{-1} \ln \delta^{-1}$-mal JA ausgegeben wurde (und $\mathcal{A}_\delta$ dann ebenfalls JA ausgibt), so gilt für alle Instanzen I

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq 1 - \delta.$$

- **One-sided MC Repetition:** An N-fold repetition for $N = \varepsilon^{-1} \ln \delta^{-1}$ boosts the success probability of a one-sided error Monte Carlo algorithm from ε to $\geq 1 - \delta$.

Error Probability Reduction

On Cheatsheet! 🎉

Error Reduction

- **Monte Carlo Repetition:** An N-fold repetition of a Monte Carlo algorithm for $N = 4\varepsilon^{-2} \ln \delta^{-1}$ boosts the success probability from $\frac{1}{2} + \varepsilon$ to $\geq 1 - \delta$ (by outputing the majority answer).
- **One-sided MC Repetition:** An N-fold repetition for $N = \varepsilon^{-1} \ln \delta^{-1}$ boosts the success probability of a one-sided error Monte Carlo algorithm from ε to $\geq 1 - \delta$.
- **Target Shooting:** If the target-shooting algorithm identifies a set $S \subseteq U$ with $N \geq 3\frac{|U|}{|S|}\varepsilon^{-2} \ln(2/\delta)$ trials, then with probability $\geq 1 - \delta$ the output lies in the interval

$$\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right].$$

Error Probability Reduction

Satz 2.75. Sei $\varepsilon > 0$ und $\mathcal{A}$ ein randomisierter Algorithmus, der immer eine der beiden Antworten JA oder NEIN ausgibt, wobei

$$\Pr[\mathcal{A}(\mathrm{I}) \text{ korrekt}] \geq 1/2 + \varepsilon.$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $\mathrm{N} = 4\varepsilon^{-2} \ln \delta^{-1}$ unabhängige Aufrufe von $\mathcal{A}$ macht und dann die Mehrheit der erhaltenen Antworten ausgibt, so gilt für den Algorithmus $\mathcal{A}_\delta$, dass

$$\Pr[\mathcal{A}_\delta(\mathrm{I}) \text{ korrekt}] \geq 1 - \delta.$$

Error Probability Reduction

On Cheatsheet! 🎉

Error Reduction

- **Monte Carlo Repetition:** An N-fold repetition of a Monte Carlo algorithm for $N = 4\varepsilon^{-2} \ln \delta^{-1}$ boosts the success probability from $\frac{1}{2} + \varepsilon$ to $\geq 1 - \delta$ (by outputing the majority answer).
- **One-sided MC Repetition:** An N-fold repetition for $N = \varepsilon^{-1} \ln \delta^{-1}$ boosts the success probability of a one-sided error Monte Carlo algorithm from ε to $\geq 1 - \delta$.
- **Target Shooting:** If the target-shooting algorithm identifies a set $S \subseteq U$ with $N \geq 3\frac{|U|}{|S|}\varepsilon^{-2} \ln(2/\delta)$ trials, then with probability $\geq 1 - \delta$ the output lies in the interval

$$\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right].$$

Error Probability Reduction

Satz 2.76. Sei $\varepsilon > 0$ und $\mathcal{A}$ ein randomisierter Algorithmus für ein Maximierungsproblem, wobei gelte:

$$\Pr[\mathcal{A}(I) \geq f(I)] \geq \varepsilon.$$

Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathcal{A}_\delta$ den Algorithmus, der $N = \varepsilon^{-1} \ln \delta^{-1}$ unabhängige Aufrufe von $\mathcal{A}$ macht und die *beste* der erhaltenen Antworten ausgibt, so gilt für den Algorithmus $\mathcal{A}_\delta$, dass

$$\Pr[\mathcal{A}_\delta(I) \geq f(I)] \geq 1 - \delta.$$

(Für Minimierungsprobleme gilt eine analoge Aussage wenn wir „$\geq f(I)$“ durch „$\leq f(I)$“ ersetzen.)

Error Probability Reduction

On Cheatsheet! 🎉

Error Reduction

- **Monte Carlo Repetition:** An N-fold repetition of a Monte Carlo algorithm for $N = 4\varepsilon^{-2} \ln \delta^{-1}$ boosts the success probability from $\frac{1}{2} + \varepsilon$ to $\geq 1 - \delta$ (by outputing the majority answer).
- **One-sided MC Repetition:** An N-fold repetition for $N = \varepsilon^{-1} \ln \delta^{-1}$ boosts the success probability of a one-sided error Monte Carlo algorithm from ε to $\geq 1 - \delta$.
- **Target Shooting:** If the target-shooting algorithm identifies a set $S \subseteq U$ with $N \geq 3\frac{|U|}{|S|}\varepsilon^{-2} \ln(2/\delta)$ trials, then with probability $\geq 1 - \delta$ the output lies in the interval

$$\left[(1-\varepsilon)\frac{|S|}{|U|}, (1+\varepsilon)\frac{|S|}{|U|}\right].$$

Exercise S10

“Probabilistic method”

Exercise S10.1 – *Probabilistic method*

So far, we mainly used randomization to construct efficient algorithms. However, randomization can also be used to prove statements that don't involve randomness themselves. In fact, such proves can usually be phrased by describing a Monte-Carlo algorithm. Answer the two following questions. If you want to, you can describe your solution as a Monte-Carlo algorithm.

(a) You are given 1000 subsets $A_1, \ldots, A_{1000}$ of $\{1, \ldots, n\}$. Each subset has size at least 11. Can the numbers $1, \ldots, n$ be colored 'red' and 'blue', such that each set A_i contain at least one 'red' and at least one 'blue' number?

(b) You are given 10 points in the plane (i.e., $\mathbb{R}^2$). Can you place (filled) disks of radius 1 in the plane, such that (i) each of the 10 points is contained in a disk, and (ii) all disks are disjoint (i.e., their centers have distance at least 2)?
Remark: You may cover multiple points with the same disk. In particular, the task would be trivial if all points are very close to each other (then you only need one disk to cover all of them). Similarly, if all points are very far apart, you could use one disk per point without having to worry about disjointness.
Remark: a formal solution to this exercise would be very technical. Focus more on the ideas than on technical details

Primality Test

Primality Test

Satz 2.77 (Kleiner fermatscher Satz). Ist $n \in \mathbb{N}$ prim, so gilt für alle Zahlen $0 < a < n$

$$a^{n-1} \equiv 1 \mod n.$$

Primality Test

If n is a **prime number**, then the set of integers $\{0, 1, \ldots, n-1\}$ forms a **Field** under addition and multiplication modulo n. (remember Dsicrete Mathematics lecture)

In any field, a polynomial of degree d can have at most d roots. Specifically, for the quadratic congruence:

$$x^2 \equiv 1 \pmod{n}$$

for $0 < x < n$, there are exactly **two solutions** if n is prime:

- $x = 1$
- $x = n - 1$ (which is equivalent to $-1 \pmod{n}$)

Miller-Rabin-Primzahltest(n)

1: **if** $n = 2$ **then**
2: **return** 'Primzahl'
3: **else if** n gerade oder $n = 1$ **then**
4: **return** 'keine Primzahl'
5: Wähle $a \in \{2, 3, \ldots, n-1\}$ zufällig und
6: berechne $k, d \in \mathbb{Z}$ mit $n - 1 = d2^k$ und d ungerade.
7: $x \leftarrow a^d \pmod{n}$
8: **if** $x = 1$ or $x = n - 1$ **then**
9: **return** 'Primzahl'
10: **repeat** $k - 1$ mal
11: $x \leftarrow x^2 \pmod{n}$
12: **if** $x = 1$ **then**
13: **return** 'keine Primzahl'
14: **if** $x = n - 1$ **then**
15: **return** 'Primzahl'
16: **return** 'keine Primzahl'

Primality Test

The **Miller-Rabin** algorithm is a **Monte Carlo algorithm** used to determine whether a given integer n is prime.

- **Runtime:** $O(\ln n)$.
- **Correctness for Primes:** If n is a prime number, the algorithm always returns the answer **'Prime'**.
- **Accuracy for Composite Numbers:** If n is a composite number, the algorithm identifies it correctly by returning **'not a prime'** with a probability of at least $3/4$.
- **Error Reduction:** We can make the error probability (the chance of incorrectly labeling a composite number as prime) **arbitrarily small** by repeating the test multiple times, according to Satz 2.74.

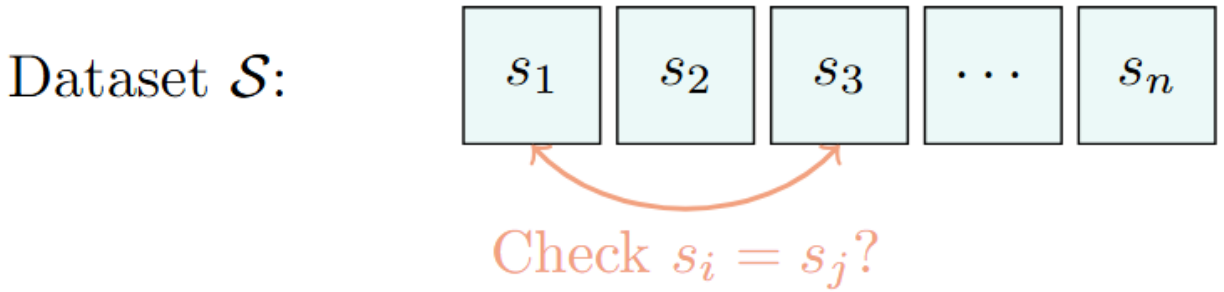
Finding Duplicates – Hash Functions

(Bloom Filters not shown yet in lecture)

Finding Duplicates

Problem Definition

We are given a dataset $\mathcal{S} := \{s_1, \ldots, s_n\}$ and must identify all pairs $1 \leq i < j \leq n$ where $s_i = s_j$. We assume that the original dataset is **read-only** and cannot be modified or reordered due to the size or nature of the individual records s_i (and that the equality check $s_i = s_j$ is expensive).



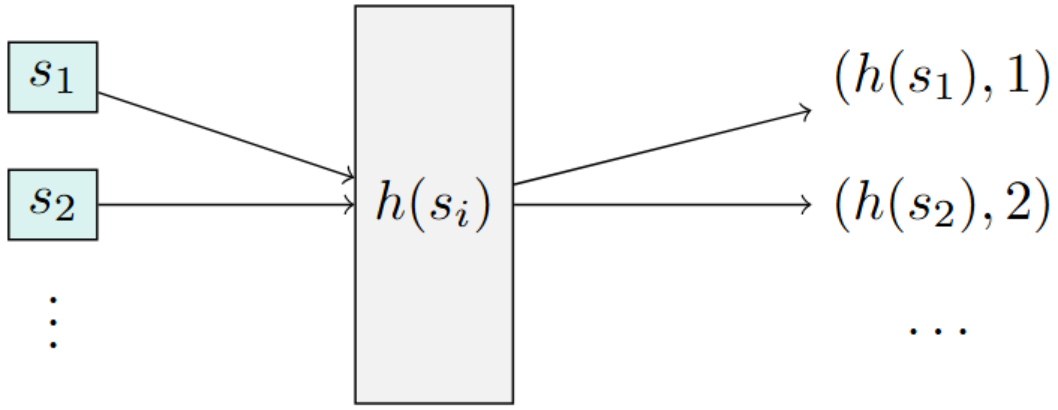
Finding Duplicates

The Hashing Approach

To avoid comparing every pair ($O(n^2)$), we map each element to a representative using a hash function $h : U \to \{1, \ldots, m\}$. We assume h is efficiently computable and distributes elements uniformly across the range $[m]$.

Create Pairs $(h(s_i), i)$

We iterate through the dataset once and create a new list of **ordered pairs**. Each pair contains the hash value and the index of the original element.



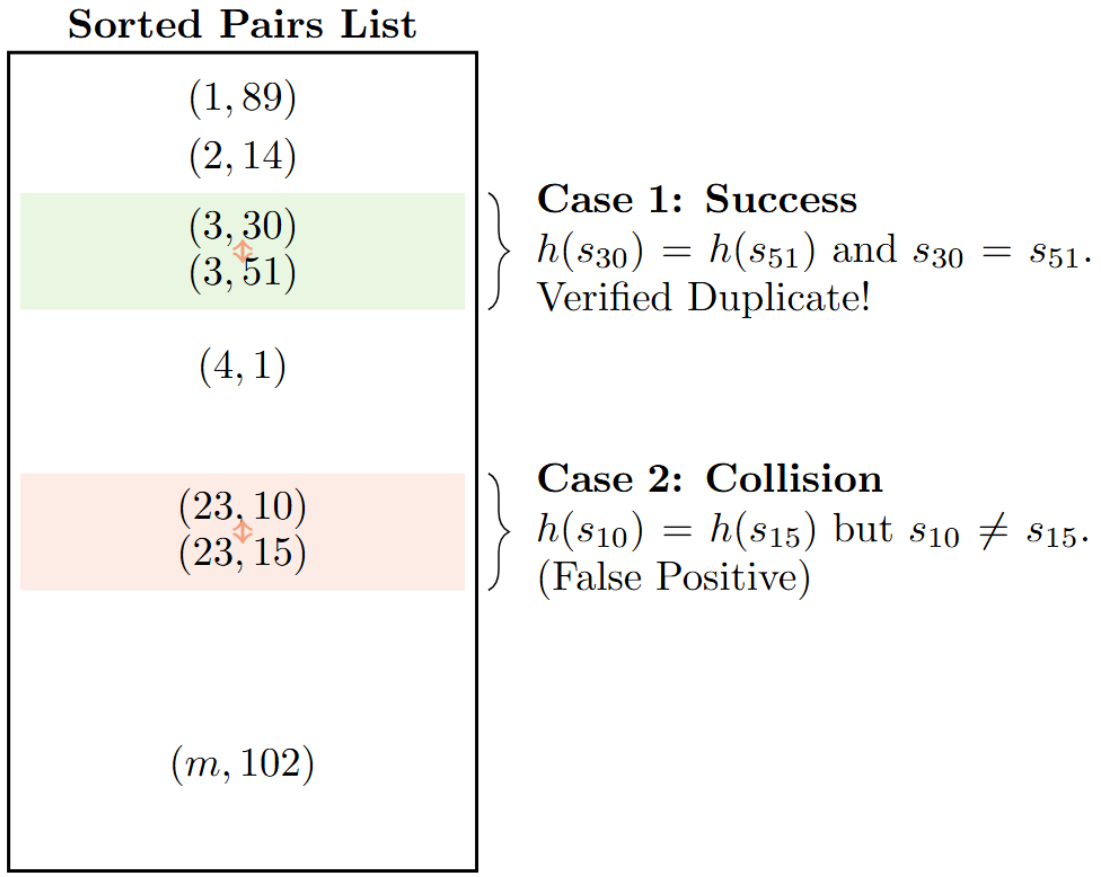
Finding Duplicates

Sorting

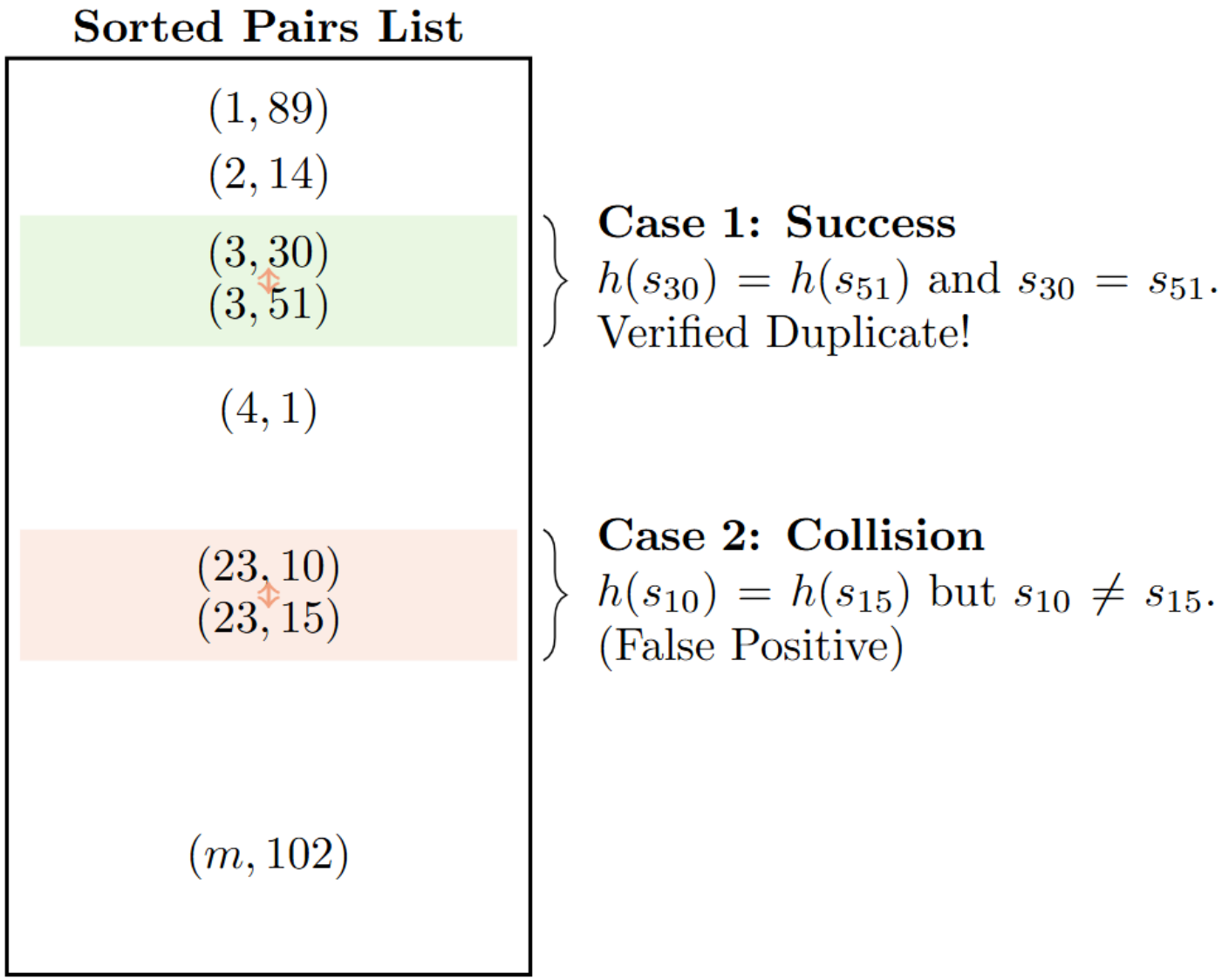
We sort the resulting list of pairs based on the hash values (the first coordinate). This groups potential duplicates together in the list.

Verification

We scan the sorted list. Only when consecutive entries share the same hash value k do we perform the expensive check on the original data s_i, s_j.



Finding Duplicates



Finding Duplicates

Analysis and Efficiency

To evaluate the algorithm, we determine the expected number of collisions, which includes actual duplicates and unintentional collisions.

Collision Probability

Let X_i be an indicator variable for dataset s_i that is 1 if an unintentional collision occurs with any $s_j \neq s_i$. Assuming a uniform hash distribution over $[m]$:

$$\Pr[X_i = 1] = 1 - \left(1 - \frac{1}{m}\right)^{n-1}$$

Finding Duplicates

Collision Probability

Let X_i be an indicator variable for dataset s_i that is 1 if an unintentional collision occurs with any $s_j \neq s_i$. Assuming a uniform hash distribution over $[m]$:

$$\Pr[X_i = 1] = 1 - \left(1 - \frac{1}{m}\right)^{n-1}$$

Expected Comparisons

By linearity of expectation, the total comparisons required are bounded by:

$$\begin{aligned}\mathbb{E}[\text{Total Collisions}] &\leq \text{Num. Duplicates} + \sum_{i=1}^{n} \mathbb{E}[X_i] \\ &\leq \text{Num. Duplicates} + n \cdot \left(1 - \left(1 - \frac{1}{m}\right)^{n-1}\right)\end{aligned}$$

Finding Duplicates

The choice of table size m controls the verification overhead:

- **Optimal Size**: Setting $m = n^2$ ensures $1 - (1 - \frac{1}{m})^{n-1} = O(1/n)$.
- **Efficiency**: This results in only $O(\text{Num. Duplicates})$ comparisons after sorting.
- **Memory**: $\Theta(n(\log n + \log m)) = \Theta(n \log n)$ for hash map storage.
- **Runtime**: $\Theta(n \log n)$ for sorting the hash map.

Sort and Select

Sort and Select

QuickSelect(A, ℓ, r, k)

1: $p \leftarrow \text{Uniform}(\{\ell, \ell+1, \ldots, r\})$ ▷ wähle Pivotelement zufällig
2: $t \leftarrow$ **Partition**(A, ℓ, r, p)
3: **if** $t = \ell + k - 1$ **then**
4: **return** $A[t]$ ▷ gesuchtes Element ist gefunden
5: **else if** $t > \ell + k - 1$ **then**
6: **return** **QuickSelect**$(A, \ell, t-1, k)$ ▷ gesuchtes Element ist links
7: **else**
8: **return** **QuickSelect**$(A, t+1, r, k-t)$ ▷ gesuchtes Element ist rechts

Sort and Select

QuickSelect

- **Goal**: Determine the k-th smallest value in a sequence of distinct numbers.
- **Efficiency**: Solves the selection problem in expected $O(n)$ steps, faster than the $O(n \log n)$ required for full sorting.
- **Logic**: Uses a random pivot p and `PARTITION` to split elements. It recurses only into the side containing the k-th element.

Expected Runtime Analysis

The total runtime T is the sum of comparisons $(r_i - \ell_i)$ over all random calls N.

$$T = \sum_{i=1}^{N} (r_i - \ell_i)$$

To simplify the sum, we group calls by the size of the remaining search space.

- **Definition of N_j:** N_j is the number of `QuickSelect` calls where the problem size $r_i - \ell_i + 1$ falls within the range:

$$(3/4)^j n < r_i - \ell_i + 1 \leq (3/4)^{j-1} n$$

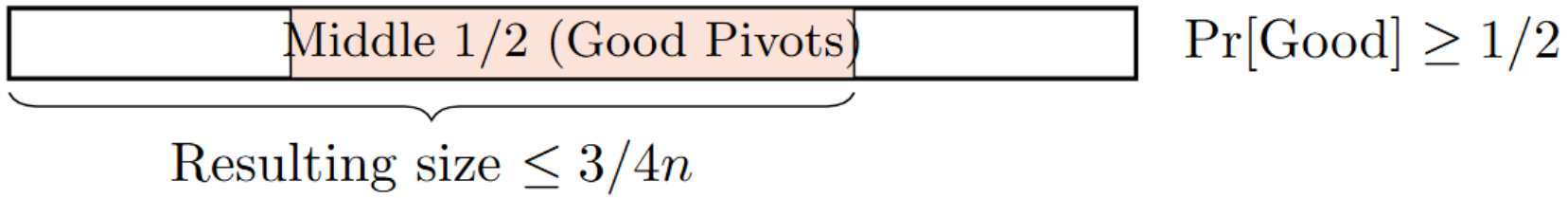
- **Bounding T:** By replacing every $r_i - \ell_i$ in a group with its upper bound $(3/4)^{j-1} n$:

$$T \leq \sum_{j=1}^{\infty} N_j \cdot (3/4)^{j-1} n$$

Sort and Select

3/4-Reduction Principle

- **Good Pivots**: Choosing one of the middle 1/2 elements as the pivot reduces the problem size to at most 3/4 of the current size.
- **Success Probability**: There is at least a 1/2 probability per call to pick such a pivot.
- **Expected** N_j: Because each attempt has a $\geq 1/2$ chance of moving to the next size range $j+1$, the expected number of calls spent in range j is $\mathbb{E}[N_j] \leq 2$.



Expected Runtime Analysis

The total runtime T is the sum of comparisons $(r_i - \ell_i)$ over all random calls N.

$$T = \sum_{i=1}^{N} (r_i - \ell_i)$$

To simplify the sum, we group calls by the size of the remaining search space.

- **Definition of N_j:** N_j is the number of `QuickSelect` calls where the problem size $r_i - \ell_i + 1$ falls within the range:

$$(3/4)^j n < r_i - \ell_i + 1 \leq (3/4)^{j-1} n$$

- **Bounding T:** By replacing every $r_i - \ell_i$ in a group with its upper bound $(3/4)^{j-1} n$:

$$T \leq \sum_{j=1}^{\infty} N_j \cdot (3/4)^{j-1} n$$

Sort and Select

By linearity of expectation:

$$\mathbb{E}[T] \leq n \cdot \sum_{j=1}^{\infty} \mathbb{E}[N_j] \cdot (3/4)^{j-1} \leq 2n \sum_{j=1}^{\infty} (3/4)^{j-1} = 2n \cdot \frac{1}{1-3/4} = 8n$$

Conclusion: The expected runtime is linear, proving $\mathbb{E}[T] \in O(n)$.