COMPUTER SCIENCE

HADES (High-end, Advanced, Data-driven, Enterprise-grade Sock sorting) - An algorithm for faster sock sorting

CelloClemens[a], *Henri*[b]

Sorting socks can often be a time-consuming task. This paper introduces the fastest method known in the scientific community to tackle this challenging task. Details required to implement the new algorithm and data structure will be introduced and discussed. Abundant application of this novel algorithm may be able to reduce the time required for sorting socks considerably.

Introduction

While sorting algorithms are one of the most discussed algorithms in the computer science community, application of this field to laundry is still quite new. In fact, no research is known to the authors connecting the fields of computer science and laundry sorting. A few definitions are required in order to establish a baseline for the algorithm discussed in the following paper.

Definitions

In this section, a few definitions that are common in the field of theoretical laundry science shall be introduced. These are required to understand the algorithm and its advantages.

Sock

Let Λ_a be the Set of laundry. The set of socks, $\Sigma \subset \Lambda_a$ is defined as $\Sigma := \{s \in \Lambda_a | \chi(s) = 1\}$[1], where $\chi(s)$ is the Euler characteristic of s. For every sock s there is an equal counterpart s^{-1} giving rise to the identity $s \cong s^{-1}$. The task commonly known as "sock sorting" is in fact the search for this isomorphism η and matching every sock s to its inverse s^{-1}.

Laundry Basket

Let $\Lambda \subseteq \Lambda_a$ be a set of laundry items. Then a laundry basket is a triplet $(\Lambda, +, -)$ representing a data structure that implements the following functions:

Figure 1: A pair of blue socks and a single orange sock.

- `get()` $\mathscr{L} \in \Lambda_a$, returns a uniformly random laundry item from the basket or $\mathscr{L}_0$, the Zero element of laundry, if there are no items left.
- `put(`$\mathscr{L} \in \Lambda_a$`)`, deposits the given laundry item into the basket.

Note that both operations run in $\mathcal{O}(1)$. Because of the nature of a laundry basket, finding a unique item requires transferring the content of the whole basket to a new basket—thus requiring $\mathcal{O}(n)$ operations, with n being the number of items currently inside the basket.

Prior Research

To fully appreciate the gravity of HADES, it has to first be discussed how most recent research tackles the problem of sock sorting. The following code describes the most recently developed sock sorting algorithm from the paper by my colleague[2] which is the current industry standard. Notice the method has a runtime complexity of $\mathcal{O}(n^2)$.

Algorithm 1 Conventional sock sorting

```
▷ initialise a new laundry basket with a given set of laundry
A ← Λ    ▷ WLOG assume ∀𝓛 ∈ Λ|𝓛 is a sock
matches ← []
repeat
    𝓛 ← A.get
    repeat▷ find the inverse Sock by checking all other socks
        𝓛′ ← A.get
    until 𝓛′ = 𝓛^−1
    matches.append((𝓛, 𝓛′))
until 𝓛 ≠ 𝓛_0
```

Concepts

The basis for every fast algorithm is a simple yet equally fast data structure. To enable the low run-

[a]Department for Theoretical Laundry Science, Karlsruhe Institute of Suffering and Sorrow (KISS), Germany
[b]Insititute of Laundry Sorting, Department for Socks, Karlsruhe Institute of Suffering and Sorrow (KISS), Germany

time achieved by HADES, the introduction of a new data structure—the "laundry rack"—is integral.

Figure 2: A blue drying rack, found in many households.

Laundry Rack

Let $\Lambda \subseteq \Lambda_a$ be a set of laundry. A laundry rack (See **Figure 2**) is a triplet $(\Lambda, +, -)$ representing a data structure that implements the following methods:

- `get`$(\mathscr{L} \in \Lambda)$: $\mathscr{L} \in \Lambda$, gets a specific laundry item from the laundry rack.
- `put`$(\mathscr{L} \in \Lambda)$, deposits a laundry item onto the laundry rack.
- `match`$(\mathscr{L} \in \Lambda)$: $(\mathscr{P} \in \Lambda \times \Lambda)|\mathscr{L}_0$, returns a tuple $(\mathscr{L}, \mathscr{L}^{-1})$ representing a pair of socks if $\mathscr{L}^{-1}$ is already on the laundry rack, $\mathscr{L}_0$ otherwise.

All these operations (especially `match`) run in $\mathcal{O}(1)$, making iteration over all n laundry items to find a pair $(\mathscr{L}, \mathscr{L}^{-1})$ obsolete.

Algorithm

Making use of the novel advanced features of a "drying rack" we are able to implement the following algorithm in $\mathscr{O}(n)$:

Algorithm 2 HADES

```
▷ initialise a new laundry basket with a given set of laundry
A ← Λ    ▷ WLOG assume ∀𝓛 ∈ Λ|𝓛 is a sock
matches ← []
⅂ ← []          ▷ Create a new empty drying rack
repeat
    𝓛 ← A.get
    result ← ⅂.match(𝓛)
    if result ≠ 𝓛₀ then
        matches.append(result)
    else
        ⅂.put(𝓛)
    end if
until 𝓛 ≠ 𝓛₀
```

As evident from the algorithm above, only one loop performing operations which are all in $\mathscr{O}(1)$ is required—putting the algorithm in a $\mathscr{O}(n)$ runtime complexity class. Assuming that $\forall \mathscr{L} \in \Lambda \exists \mathscr{L}^{-1} | \mathscr{L} \cong \mathscr{L}^{-1}$, the algorithm always yields a correct solution for the problem[3].

Discussion and Results

To evaluate the algorithm's performance, it has been executed on different platforms consisting of diverse hardware:

Hardware	Algorithm	Runtime [s]
Myself	Conventional (n=20)	352.7
Myself	HADES (n=20)	92.3
Myself	Conventional (n=100)	42069
Myself	HADES (n=100)	420.69
Roommate	Conventional (n=10)	91.7
Roommate	HADES (n=10)	-2
Girlfriend	n.a.	n.a.

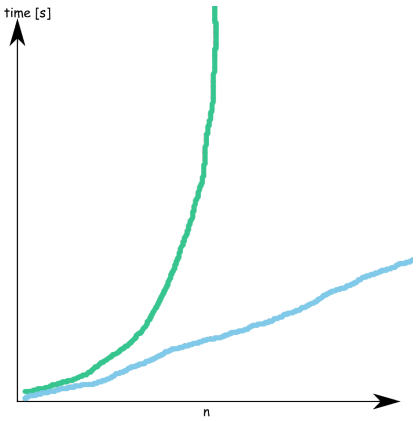

Figure 3: Comparative statistical time analysis of both algorithms. The graph depicts algorithm runtime (y-axis) and graphs it against input size (x-axis). Data for HADES in blue, for conventional sock sorting in green.

From the above data it is evident that HADES bears a clear advantage in comparison to the conventional algorithm when it comes to sock sorting. Utilising advanced statistical modelling, we calculated a speedup factor of about $3.14159265358979323846264 33 \cdot n$. The data also illustrates the scalability of the algorithm and its adaptability to different hardware.

Conclusion

It can be concluded that the algorithm presented in this paper is greatly superior to the conventional method of sorting socks. It will probably revolutionise not only the field of laundry science but also have great impact in the industry.

The data structures outlined above may become abundantly used and become the future industry standard. Although the field of laundry science is still rather new, there are still a lot of open questions to be answered. However, it is unlikely that a faster sock sorting algorithm than HADES can be developed.

Acknowledgements

We shall use this historic opportunity to thank the Journal of Immaterial Science for publishing great ~~memes~~ research. We also want to thank our university for giving us this opportunity for ~~depression and self-loathing~~ research and personal advancement. It is also only appropriate to thank the air. Without it, no laundry would be dry and we would not have written this paper.

Notes and References

[1]Yes, some socks have holes. So what?!

[2]My Colleague et al. "A first approach to sock sorting". In: *Journal of Laundry Science* (13 1969).

[3]Proof is left as an exercise to the reader.