Data Stream Algorithms Unplugged

Philip Bille[1][0000−0002−1120−5154] and Inge Li Gørtz[2][0000−0002−8322−4952]

Technical University of Denmark, Kgs. Lyngby, Denmark

Abstract. Data stream algorithms are an advanced topic in computer science that focus on processing massive data sets arriving as an online stream of items. The key algorithmic challenge is to process the items online in one or a few passes while only using very limited working space. Data stream algorithms are typically simple and elegant, but the mathematics behind their performance guarantees or correctness proofs often involve intricate combinatorial or probabilistic arguments. For many students, this is a significant barrier to effective learning and intuition-building in the area.
In this paper, we address this challenge and develop an unplugged activity that introduces data stream algorithms. The activity consists of a simple, unplugged, physical simulation of data stream algorithms using Lego bricks drawn one by one from an opaque bag and placed on a table or discarded according to the rules of the specific algorithm. In this model, we present a simulation of the classical Misra-Gries algorithm [*Sci. Comput. Program.*, 1982]. As a key feature, we provide an explicit, elementary proof of correctness that relies on the configuration of the Lego bricks assembled during the simulation. The proof is sufficiently simple that most students (with or without a few hints) can discover it on their own during a short group exercise. The activity can be easily adapted to different audiences, depending on their computer science and mathematical backgrounds (including high school students and computer science students), and we discuss these variants.

1 Introduction

Computer Science Unplugged (CS Unplugged) is a collection of activities that teach computer science without using computers or other digital technologies. Instead, CS Unplugged uses engaging demonstrations, puzzles, and games, as well as physical props such as cards, strings, paper, pens, and pencils, to explain concepts. CS Unplugged was popularized by Bell et al. in their book [7], which presents a collection of activities on fundamental algorithms for primary school children. Since then, numerous activities across a wide range of computer science topics and age groups have been designed, see e.g., [2,8,21,24,26,32,33,34,35,36], surveys [5, 6], and the homepage `www.csunplugged.org`.

Data stream algorithms are an advanced topic in computer science that focuses on processing massive streams of data arriving online. Typical examples include analyzing network traffic, online auctions, bank transactions, communication logs, atmospheric data, and astronomical data. The key features are

that algorithms must process the input data stream in one or a few passes and that their working space is limited, ideally much smaller than the input data stream. This captures the online, massive data-processing scenario where we cannot store the input, and the only feasible access to the data is by making a few passes over it. Data stream algorithms date back to the late 1970s and the topic is now an established and well-studied area with numerous results for a wide range of problems, see e.g., [1,4,9,10,11,12,13,14,15,19,22,23,27,28,29,30] and surveys [3,17,20,31].

In academic education, data stream algorithms are typically taught in advanced algorithms courses and require significant prerequisites in both computer science and mathematics. While most data stream algorithms are simple, elegant, and easy to learn, the mathematics behind their performance guarantees or correctness proofs often involve intricate combinatorial or probabilistic arguments. For many students, this is difficult and poses a significant barrier to effective learning and intuition-building in this area. Thus, a natural question is whether we can develop educational material to remedy this.

1.1 Contributions

Our goal in this paper is to address the above challenge and develop a new approach to introduce students to data stream algorithms and to better support their intuition in this area. To do so, we describe an unplugged activity that includes a physical simulation of the data stream model, a central algorithm in the model, and a simple, intuitive, and elementary proof of correctness that relies on the simulation's features. Specifically, we make the following contributions:

- We present a simple, unplugged model of data stream algorithms that uses Lego bricks of different colors, drawn one by one from an opaque bag and placed on a table or discarded according to the algorithmic rules. The "stream" of Lego bricks from the bag represents the online processing of items, and the table represents the algorithm's working memory.
- We consider the *k-frequent items problem*, that is, given an integer parameter k, we want to compute the items that appear with frequency more than n/k in a data stream of n items. In our unplugged model, this corresponds to computing which colors appear with frequency more than n/k.
- We implement the classical Misra-Gries algorithm [28] for the k-frequent items problem in the unplugged model. We show how to represent the algorithm's internal state using simple stacks of bricks. In addition, we maintain a special discard pile of groups of bricks that are clicked together and are not part of the algorithm's internal state but only used for analysis. We show how to use the discard pile to give a simple, intuitive, and elementary proof of correctness.
- Our activity can be easily adapted to different target audiences (including high school and computer science students), depending on their mathematical and computer science backgrounds and interests. We discuss several extensions and variants of our activity to accommodate these groups.

1.2 Outline

We first discuss the algorithmic background for the data stream model and the Misra-Gries algorithm in Section 2. We present the unplugged model and algorithms in Section 3. Finally, we discuss how to adapt the activity to different target audiences in Section 4.

2 Algorithmic Background

We discuss the algorithmic background of the data stream model, the frequent items problem, and its classical solutions.

2.1 Data Stream Model and the Frequent Items Problem

In the data stream model, the input is accessed sequentially as a stream $\mathcal{X} = x_1, \ldots, x_n$ of n items, and the goal is to process them using a working memory that is small compared to the length n of the stream and make as few passes over the input as possible. Ideally, we want to perform a single pass over the input and use space polylogarithmic in the length of the stream.

The *frequency* of an item x in $\mathcal{X}$, denoted f_x, is the number of copies of x in $\mathcal{X}$. An item x is *k-frequent* if $f_x > n/k$. Given a stream $\mathcal{X}$ and an integer parameter $k \geq 1$, the *k-frequent items problem* is to compute the k-frequent items in $\mathcal{X}$. If $k = 2$, we call the problem the *majority problem*.

2.2 Misra-Gries Algorithm

Misra-Gries [28] proposed a simple space-efficient two-pass solution that work as follows:

Pass 1: We maintain a table C of at most $k-1$ entries. Each table entry stores an item x with a count, denoted $C[x]$. Initially, C is empty. For each item x_i in $\mathcal{X}$,

- If $x_i \in C$, we increment the corresponding counter $C[x_i]$.
- If $x_i \notin C$ and C contains less than $k-1$ items, we insert x_i into C and set $C[x_i] = 1$.
- If $x_i \notin C$ and C contains $k-1$ items, then for each $x \in C$, we decrement $C[x]$ and delete the entry if $C[x]$ becomes 0.

Pass 2: Let C^* be the contents of the table C at the end of pass 1. We compute the frequency of each item in C^* by maintaining a counter for each item in C^* and incrementing the corresponding counter each time we see an element from C^* in $\mathcal{X}$. Finally, we output the items from C^* that are k-frequent.

The algorithm uses $O(k)$ space to store the table C, and the time is dominated by a constant number of dictionary operations for each item. With additional techniques [16, 25], these can be implemented in worst-case constant time. To show correctness, the main observation is that all k-frequent items in must be in C^* at the end of the first pass.

Lemma 1. *If an item x is k-frequent in $\mathcal{X}$, then $x \in C^*$ after the first pass.*

Proof. Consider the items in C^* and let $c = \sum_{x \in C^*} C[x]$ be the total count of them. We can partition all items in $\mathcal{X}$ into two sets: c items in C^* and $(n-c)/k$ groups such that each group contains k distinct items (each such group corresponds to a decrement operation in the third case of the algorithm). For contradiction, suppose there is another k-frequent item $x' \notin C^*$. Since the second set contains groups of k distinct items, it can have at most $(n-c)/k$ copies of x'. Thus, the first set must also contain at least one copy of x', contradicting the fact that $x' \notin C^*$.

The above proof is adapted from the original paper by Boyer and Moore [10] and extended to general k. Intuitively, the idea is that k distinct items can always be discarded without affecting the set of k-frequent items. A wide range of similar "grouping" arguments appear in other proofs [16, 18, 25, 28]. Unfortunately, we observe that for many students, such arguments are abstract, difficult to grasp, and hard to build a solid intuition for.

3 Unplugged Data Stream Algorithms and Frequent Items

We now present our new unplugged data-stream algorithm setup with Lego bricks, the k-frequent items problem in this setup, and our implementation of the Misra-Gries algorithm. We first present a generic version of our activity, then discuss how to adapt it for different target audiences in the following section.

3.1 Data Stream Algorithms with Lego Bricks

We need the following items for our unplugged, physical demonstration of data stream algorithms:

- A set of around 15-20 Lego bricks of different colors. The bricks will represent the items in the stream, and bricks of the same color represent copies of the same item. The set of bricks should include subsets with varying color frequencies to illustrate the algorithm's different outcomes. We recommend using the larger Duplo bricks, which are easily visible in larger classrooms, but any bricks that click together will suffice.
- An opaque bag that will hold the bricks. To simulate a pass of a data stream, we place our (chosen subset of) bricks into the bag and pull them out of the bag one by one. Thus, students will only see a single brick at a time and cannot see any future bricks.
- A table with a stable surface that is clearly visible to all students. We use this to illustrate the algorithm's behavior and analyze it by clicking together Lego bricks as we pull them out of the bag. Effectively, the table will serve as the algorithm's working memory.

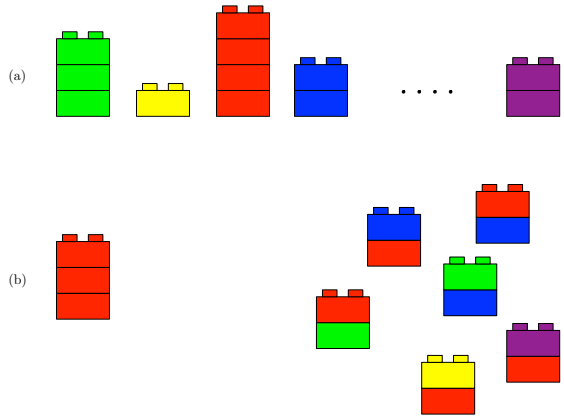

Fig. 1. (a) Stacks during the simple solution. (b) Stack and discard pile of the Misra-Gries algorithm with $k = 2$ after pass 1. There are 8 red bricks out of the 15 total, so red is the majority. We can have at most 1 red brick in each pair in the discard pile that can contain at most 7 pairs. Thus, at least 1 red brick must be in the stack.

This setup supports the main features of data stream algorithms, i.e., we process items online (pulling them out of the bag one by one) and have limited working memory (the table). To implement multiple passes, we may return the bricks to the bag to "restart" the stream. We note that the setup is simple and sufficiently general for our purposes, but it is not suited for all types of streaming algorithms. For instance, we can only deal with a limited number of distinct items (the available brick colors), and items are permuted randomly between passes (shuffled in the bag).

3.2 Frequent Items with Lego Bricks

We now show how to solve the k-frequent items problem in our Lego brick setup. We recommend demonstrating the algorithms with small k such as 2 (the majority problem), 3, or 4. We will represent the actions of our algorithms by building stacks of Lego bricks on the table using bricks pulled from the bag. Each stack consists of bricks of the same color, and the algorithm's working space will be represented by the total number of stacks. Thus, the k-frequent items problem in our setting is to compute the colors that appear more than n/k times in the stream of bricks, while using as few stacks on the table as possible and as few passes as possible.

3.3 Simple Solution

As a warm-up and to build intuition for the problem, we first present the naive single-pass solution to the audience. As discussed, we will maintain stacks of

bricks of the same color. Initially, all stacks are empty (the table is clear). For each brick x_i we pull out of the bag,

- If x_i matches a stack, we click x_i to the top of that stack.
- If x_i does not match any stack, we make a new stack consisting of x_i.

Note that the stack heights represent the frequency of each color. At the end, we inspect the stacks and check whether any stack contains more than n/k bricks, reporting them as k-frequent items.

Clearly, the naive solution correctly solves the problem. The main issue is that we need a stack on the table for each distinct color that appears in the stream. We explain to the students that the key challenge is to avoid these stacks, and our goal is to solve the problem with significantly fewer stacks.

3.4 Misra-Gries Algorithm

We then present the Misra-Gries algorithm.

Pass 1: We maintain at most $k-1$ stacks of bricks on the table. Each stack will contain bricks of the same color. Initially, all stacks are empty (i.e., the table is clear). For each brick x_i, we pull out of the bag,

- If x_i matches a color of a stack, we click x_i to the top of that stack.
- If x_i does not match any stack color and there are fewer than $k-1$ stacks, we create a new stack consisting of x_i.
- If x_i does not match any stack color and there are $k-1$ stacks, we click off the top of each stack. We then click the resulting top bricks and x_i together to form a group of k bricks, and put this group in a *discard pile*. For dramatic effect, we can throw these in a trash can or on the floor.

Pass 2: Unclick all the bricks, put them back in the bag, and remember only the colors C^* that were in the stacks at the end of pass 1. Start a new pass, and for each brick x_i we pull from the bag,

- If x_i matches a color in C^*, we click x_i to the top of that stack.
- If x_i does not match any color in C^* we ignore it.

At the end, we inspect the stacks, check whether each stack has a height greater than n/k, and report the colors of those that do as the k-frequent items.

We discuss the algorithm with the audience and, if needed, repeat the demonstration using different subsets of bricks to clarify its behavior and illustrate different outcomes. We then further discuss to what extent the algorithm satisfies the stated goals. We observe that it uses two passes and no more than $k-1$ stacks at any point in time. Thus, it remains to argue that it correctly solves the problem, and this will be the main focus of our discussion with the audience. The key property that we want to show is Lemma 1 adapted to our setting:

Lemma 2. *Any color that appears more than n/k times must be the color of a stack at the end of pass 1.*

We argue that this property must imply that the algorithm correctly solves the problem. Then we invite the audience to discuss why this property is true, either individually or in small groups. We recommend giving the audience about 5 minutes for this task. While doing so, we leave the stacks and the discard pile after a demonstration of pass 1 visible for the students. If needed, we provide the following hint: if a color appears more than n/k times, where can the bricks of that color be at the end of pass 1? And further: look closely at the discard pile. In our experience, most of our target audiences will discover a correct argument for Lemma 2 by realizing that since each group in the discard pile all have distinct colors, a frequent color must appear in a stack after pass 1 (see Fig. 1 for an illustration). More formally:

Proof. At the end of pass 1, bricks of a given color must be either in a stack or in the discard pile. The discard pile consists of at most n/k groups of k bricks. Each group of k bricks consists of bricks of k distinct colors. Hence, there can be at most n/k bricks of any given color in the discard pile. Therefore, if a color appears *more* than n/k times, that color must be in a stack at the end of pass 1.

Note that the discard pile helps to make the "grouping" argument more explicit since we can now physically point to the grouping.

4 Target Audiences

The above generic activity can be adapted to different audiences. We discuss our experiences with some of these below.

4.1 Late Primary School/High School Students

For this audience, our activity is best suited for demonstrating computational thinking and introducing students to clever algorithmic ideas that require non-trivial combinatorial insights. Thus, we focus on the topic mainly as an algorithmic puzzle and ignore programming details. To motivate the problem, we explain and discuss the importance of data streaming algorithms in real-world scenarios.

For this audience, we focus on the majority problem ($k = 2$). If possible, provide each group with a set of Lego bricks to help them test their ideas. Before we let the students discuss why Lemma 2 is true, we suggest to first let the students "play" with the proof of the algorithm by trying to give the bricks in an order that would break it, and then use the insight obtained to understand why the proof is correct. As a further exercise, we suggest that students generalize to larger k (starting with $k = 3$) using their own set of bricks.

4.2 Graduate Algorithms

For this audience, the main use is to introduce students with a background in algorithms to the data stream model and its main challenges. We motivate the problem as in Section 4.1. We focus on general k and the proof of Lemma 2. Afterward, we relate the unplugged model and Misra-Gries algorithm to the standard (plugged) data stream model and Misra-Gries algorithm.

Furthermore, we propose a set of advanced exercises with elegant solutions using the unplugged model.

- Let f_i be the frequency of color i in the stream $\mathcal{X}$. Prove that the estimated frequency $\hat{f}_i$ of color i is at least $f_i - \frac{n-c^*}{k}$, where c^* is the sum of the values of the counters in C^*. *Solution: the total number of bricks in the discard pile is at most $n - c^*$. Thus, each color can occur at most $\frac{n-c^*}{k}$ times in the discard pile.*
- Give an example of a stream that illustrates why the algorithm cannot be used to compute colors with frequency *at least* n/k (instead of *more than* n/k). *Solutions: There are several solutions to this. E.g., a stream of bricks of k colors that each occur n/k times. All bricks of the kth color are at the end of the stream. This will result in all stacks being empty at the end of the stream.*
- Assume you have two streams $\mathcal{X}_1$ and $\mathcal{X}_2$ and run the Misra-Gries algorithm on each of them independently with the same k. Now merge the two results as follows. If a color has a stack in both streams, add the stacks together. You now have at most $2(k-1)$ stacks. Let h be the value of the kth largest of these stacks. Remove a brick h from each stack, keeping only the non-empty stacks. Prove that any color that appears more than n/k times must be a color of a non-empty stack. *Solution: Merge the two discard piles into one. All the groups in the merged discard pile have the same property as before. When removing h bricks from each stack, do it in h rounds. In each round, remove a brick from each non-empty stack, click them together, and put them in a new discard pile. The important observation here is that each of the newly created groups has at least k bricks, and the bricks in a group have distinct colors. Thus, a color cannot be discarded more than n/k times.*

As above, we suggest providing each group with a set of Lego bricks to help them test their ideas.

4.3 Public Outreach

The unplugged activity is also suitable for more popular talks aimed at promoting computer science ideas and computational thinking. In this scenario, we use the same version of the activity as in Section 4.1. However, we emphasize the connection to real-world practical problems.

5 Conclusion and Open Problems

We presented an unplugged activity for introducing data stream algorithms through a physical simulation with Lego bricks. The activity captures the central constraints of the streaming model—online processing and limited memory—while making the Misra-Gries algorithm and its correctness proof concrete and visually accessible. In particular, the discard pile provides a tangible representation of the grouping argument, allowing students to discover the key invariant themselves rather than encountering it only as an abstract combinatorial proof. The activity is simple to run, requires minimal materials, and can be adapted to audiences ranging from high school students to graduate algorithms students and public outreach settings.

Several natural open problems remain. First, it would be valuable to design and evaluate unplugged activities for other core data streaming techniques, such as Count-Min, sampling-based methods, or frequency-moment estimators. Second, a systematic classroom study could measure how the physical grouping argument affects student understanding compared with a traditional lecture-based presentation. Finally, it would be interesting to explore whether the unplugged model can support deeper advanced topics while preserving the simplicity and intuition that make the activity effective.

6 Acknowledgments

We thank Gad M. Landau for the initial inspiration for this work.

References

1. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th STOC*, pages 20–29, 1996.
2. P. P. Angara, U. Stege, A. MacLean, H. A. Müller, and T. Markham. Teaching quantum computing to high-school-aged youth: a hands-on approach. *IEEE Trans. Quantum Eng.*, 3:1–15, 2021.
3. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21st PODS*, pages 1–16, 2002.
4. Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68(4):702–732, 2004.
5. A. Battal, G. Afacan Adanır, and Y. Gülbahar. Computer science unplugged: A systematic literature review. *J. Educ. Tech. Sys.*, 50(1):24–47, 2021.
6. T. Bell, F. Rosamond, and N. Casey. Computer science unplugged and related projects in math and computer science popularization. In *The multivariate algorithmic revolution and beyond: Essays dedicated to Michael R. Fellows on the occasion of his 60th Birthday*, pages 398–456. 2012.
7. T. Bell, I. H. Witten, and M. R. Fellows. *Computer Science Unplugged - an enrichment and extension programme for primary-aged children.* 2002.

8. C. A. Benavides-Escola, E. Martín-Barroso, M. Zapata-Cáceres, and M. Román-González. Improvement of computational thinking skills through unplugged activities in upper secondary education. In *Proc. 19th WiPSCE*, 2024.
9. L. Bhuvanagiri, S. Ganguly, D. Kesh, and C. Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proc. 17th SODA*, pages 708–713, 2006.
10. R. S. Boyer and J. S. Moore. MJRTY—a fast majority vote algorithm. *Automated reasoning: essays in honor of Woody Bledsoe*, pages 105–117, 1991.
11. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. 29th ICALP*, pages 693–703, 2002.
12. M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proc. 35th STOC*, pages 30–39, 2003.
13. D. Coppersmith and R. Kumar. An improved data stream algorithm for frequency moments. In *Proc. 15th SODA*, pages 151–156, 2004.
14. G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.
15. G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.
16. E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proc. 10th ESA*, pages 348–360, 2002.
17. C. Demetrescu and I. Finocchi. Algorithms for data streams. In *Handbook of Applied Algorithms: Solving Scientific, Engineering and Practical Problems*, pages 241–269. Wiley Online Library, 2008.
18. M. J. Fischer and S. L. Salzberg. Finding a majority among n votes: Solution to problem 81–5. *J. Algorithms*, 3(4):376–379, 1982.
19. G. Frahling, P. Indyk, and C. Sohler. Sampling in dynamic data streams and applications. In *Proc. 21. SoCG*, pages 142–149, 2005.
20. P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *External memory algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 50:39–70, 1999.
21. R. I. Greenberg. Educational magic tricks based on error-detection schemes. In *Proc. 22nd ITiCSE*, pages 170–175, 2017.
22. P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proc. 36th STOC*, pages 373–380, 2004.
23. P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proc. 27th STOC*, pages 202–208, 2005.
24. R. Jacob and F. Silvestri. Unplugging dijkstra's algorithm as a mechanical device. In *Proc. 7th CMSC*, pages 39–49, 2024.
25. R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1):51–55, 2003.
26. L. Lehner and M. Landman. Unplugged decision tree learning–a learning activity for machine learning education in k-12. In *Proc. 7th CMSC*, pages 50–65, 2024.
27. A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proc. 10th ICDT*, pages 398–412, 2005.
28. J. Misra and D. Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.
29. R. Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978.
30. J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoret. Comput. Sci.*, 12(3):315–323, 1980.

31. S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.
32. S. Soviani, J. Kusnendar, and H. Prabawa. Learning how computer's work with combining cs-unplugged and raspberry pi. In *J. Phys.: Conf. Ser.*, volume 1280, page 032025, 2019.
33. X. Suo, O. Glebova, D. Liu, A. Lazar, and D. Bein. A survey of teaching pdc content in undergraduate curriculum. In *Proc 11th CCWC*, pages 1306–1312, 2021.
34. S. Szeider. Large and parallel human sorting networks. In *Proc. 7th CMSC*, pages 194–204, 2024.
35. B. Tonbuloğlu and İ. Tonbuloğlu. The effect of unplugged coding activities on computational thinking skills of middle school students. *Inform. Edu.*, 18(2):403–426, 2019.
36. P. Virtue. Gans unplugged. In *Proc. 35th AAAI*, volume 35, pages 15664–15668, 2021.