Münchhausen Matrices

Michael Brand

Faculty of Information Technology
Monash University
Clayton, VIC, Australia

michael.brand@alumni.weizmann.ac.il

Submitted: May 10, 2012; Accepted: Nov 27, 2012; Published: Dec 13, 2012
Mathematics Subject Classifications: 11B83, 05B20, 05B30, 05-04

Abstract

"The Baron's omni-sequence", $B(n)$, first defined by Khovanova and Lewis (2011), is a sequence that gives for each n the minimum number of weighings on balance scales that can verify the correct labeling of n identically-looking coins with distinct integer weights between 1 gram and n grams.

A trivial lower bound on $B(n)$ is $\log_3 n$, and it has been shown that $B(n)$ is $\mathrm{O}(\log n)$.

We introduce new theoretical tools for the study of this problem, and show that $B(n)$ is $\log_3 n + \mathrm{O}(\log \log n)$, thus settling in the affirmative a conjecture by Khovanova and Lewis that the true growth rate of the sequence is very close to the natural lower bound.

Keywords: Baron's Omni-sequence; Munchausen; coin weighing; verification

1 Introduction and definitions

Coin-weighing puzzles have been abundantly discussed in the mathematical literature over the past 60 years (see, e.g. [16, 7, 15, 5]). In coin-weighing problems one must typically identify a counterfeit coin from a set of identically-looking coins by use of balance scales, utilizing the knowledge that the counterfeit coin has distinctive weight. This can be generalized to the problem of identifying a coin, or a subset of the coins, based on distinctive weight characteristics, or, alternatively, to the problem of establishing the weight of a given coin.

This paper relates to a different kind of coin-weighing puzzle, which we call the Münchhausen coin-weighing problem. Consider the following question: given n coins with distinct integer weights between 1 gram and n grams, each labeled by a distinct integer label between 1 and n, what is the minimum number of weighings of these n coins

on balance scales that can prove unequivocally that all coins are labeled by their correct weight?

This question differs from classic coin-weighing problems in that we do not need to *discover* the weights, but only to determine whether or not a given labeling of weights is the correct one. To establish the weights one would require $\Omega(n \log n)$ weighings (as can be proved by reasoning similar to that which establishes lower bounds for comparative sorting [13, 3]), whereas merely verifying an existing labeling can be performed trivially in $O(n)$ weighings.

This question, which was inspired by a riddle that appeared in the Moscow Mathematical Olympiad [1], gives rise to an integer sequence, $B(n)$, that was studied in [11] and was dubbed there "The Baron's omni-sequence". It appears as sequence A186313 in the On-line Encyclopedia of Integer Sequences [10].

The present paper takes the view that, in studying this and related topics, [11] and [12] uncovered the tip of an iceberg, under which lies a wealth of hitherto unexplored mathematical questions that belong to a new class of problems. Namely, these are "verification problems".

In the remaining parts of this section, we define some terminology regarding weighings. The definitions present the subject of verification as resembling a design problem, and demonstrate how verification can be considered as a matrix property. The section will conclude with a re-definition of $B(n)$ as a special case and a formulation of the problem statement.

In Section 2 we develop new analytical tools which we believe to be of importance in a range of verification problems. We begin by presenting a simplified version of the tools. These allow the construction of matrices with the desired properties, but only for matrices whose dimensions satisfy certain conditions. For n values that satisfy these special conditions, the constructions prove an upper bound on $B(n)$ that is tighter than the best previously-known bound.

In Section 3, the tools developed previously are embellished. Further complications allow their use for matrices with general widths. We note that in both Section 2 and Section 3 the techniques used, even with the extra features, are fairly generic in what they accomplish and are straightforward to apply to other problems. Unfortunately, proving that the algorithms terminate successfully for any specific case, such as the Münchhausen coin-weighing problem, and, furthermore, proving bounds on their outputs for specific cases, does necessitate a certain level of technical detail, which forms large parts of the sections.

Lastly, in Section 4, we put the new tools to full use, designing with them an efficient strategy for solving the Münchhausen coin-weighing problem for any n. Our main theorems are proved by calculating the bound on $B(n)$ that this strategy implies.

1.1 Weigh-sets

First, we introduce the following notations. Let $\vec{\boldsymbol{n}}$ be the vector $(1, \ldots, n)^T$, $\vec{\mathbf{1}}_n$ be the $n \times 1$ all-ones vector and $\vec{\mathbf{0}}_n$ be the $n \times 1$ all-zeros vector, where the subscript n may be

omitted from the latter two notations when it is clear from the context. The name of a vector will also be used for its elements. For example, if $\vec{v}$ is a vector of length s, its elements will be denoted as $(v_1, \ldots, v_s)$. For a matrix, M, M_{ij} is its element in the i'th row and j'th column, where index counting begins with 1. M_{i*} and M_{*j} signify the i'th row and the j'th column of M, respectively.

For a set of coins X, a *weighing* can be described by an unordered pair of disjoint subsets of X, $\{L, R\}$. Set L is conceptually the subset placed on one pan of the balance scales and set R is placed on the other pan. Similarly, a *weigh-set* can be described as a set of weighings. Thus, the definition of a weigh-set of X as a set of unordered pairs of disjoint subsets of X is possibly reminiscent of the definition of a design (most generally describable as a set of disjoint subsets of some base set X). Like in the case of designs, it is often convenient to discuss weighings and weigh-sets in matrix notation. Consider the following.

If the elements of the set X are indexed $1, \ldots, n$, a vector $\vec{v}$ of length n with entries in $\{-1, 0, 1\}$ represents a weighing by placing in L all elements of X whose indices have a -1 entry in $\vec{v}$, and placing in R all those whose indices have a 1 entry. Similarly, an $m \times n$ matrix, M, with entries in $\{-1, 0, +1\}$ represents a weigh-set composed of m weighings: each row represents a weighing. (Note: this weigh-set in matrix notation should not be confused with a "weighing matrix", defined in [14], which relates to a different type of weighing problem.)

Clearly, the matrix notation has redundant information. The order of the rows can be permuted arbitrarily and each row can be inverted. A weigh-set matrix should therefore be taken as a representative of an entire set of possible representations of the same weigh-set. In this paper, we use set and matrix notation interchangeably when discussing weigh-sets. For example, though the variable M is always a matrix, the terminology "weigh-set M" should be taken as short-hand for "a weigh-set represented by matrix M".

Suppose that X is a set of n coins, in which the i'th coin weighs x_i. In the weigh-set M, the vector that is the result of the product $M\vec{x}$ describes the m imbalances in the m weighings described by M. On balance scales, we typically get the information of the sign of each entry in $M\vec{x}$. However, in other scenarios we may consider having more information (such as full information on the imbalance) or less (just the binary information of whether or not the two pans balance, omitting the sign information of any imbalance).

A weigh-set M is a *verification* for the set X with elements whose weights are described by the vector $\vec{x}$ if the vector of signs of the elements of $M\vec{x}$ is unique among all $M\vec{y}$ where $\vec{y}$ is a permutation of $\vec{x}$. A matrix (weigh-set) that is a verification of the coins weighing $1, \ldots, n$ grams is a *Münchhausen weigh-set* or *Münchhausen matrix*.

In addition, one can also consider "partial verifications". We say that a weigh-set, M, *establishes* a partition of X if the vector of signs of the elements of $M\vec{x}$ is different to that of any $M\vec{y}$ where $\vec{y} = \sigma(\vec{x})$ and σ is a permutation that does not respect the partition.

By contrast, we say that the weigh-set verifies X *subject to* the partition if the vector of signs of the elements of $M\vec{x}$ is unique among all $M\vec{y}$ where $\vec{y} = \sigma(\vec{x})$ and σ is a permutation that *does* respect the partition.

If A is a weigh-set that establishes some partition, P, of X, and B is a weigh-set that verifies X subject to it, $A \cup B$ is a verification of X.

1.2 The Baron's omni-sequence

The Baron's omni-sequence is the sequence whose n'th term, $B(n)$, is the minimum number for which there exists a $B(n) \times n$ Münchhausen matrix.

A trivial lower bound for $B(n)$ is $\log_3 n$. This bound is attained from the observation that there are at most 3^m distinct columns in an $m \times n$ $\{-1, 0, 1\}$-matrix, and a Münchhausen matrix cannot have duplicate columns, because if any two columns, i and j, were identical, permuting the coins by replacing the i'th and j'th coin would not have affected the weighing results.

In [11], Khovanova and Lewis construct $m \times n$ Münchhausen matrices with $m \leqslant \alpha \log_3 n + C$ for some constant C and an α value of approximately 2.96. This has since been improved to $\alpha = 2$ in [2], reducing the gap between the theoretical lower bound and the best known algorithm to a factor of 2.

It was conjectured in [11] that $B(n)$ is very close to $\log_3 n$, a statement that can, perhaps, be rigorously taken to mean that $B(n)$ is $\log_3 n + \mathrm{o}(\log_3 n)$. The main claim of this paper is:

Theorem 1. *The sequence $B(n)$ is $\log_3 n + \mathrm{O}(\log \log n)$.*

1.3 The Baron's homogeneous omni-sequence

All constructions in this paper (with the exception of constructions for $n < 4$, where these are impossible), belong to a special class of weigh-sets we call *homogeneous*. A *homogeneous weigh set* is a matrix, M, such that $M\vec{\boldsymbol{n}} = \vec{\mathbf{0}}$. By definition, homogeneous Münchhausen weigh-sets require that $M\vec{\boldsymbol{n}} = \vec{\mathbf{0}}$ and that $M\vec{v'} \neq \vec{\mathbf{0}}$ for $\vec{v'}$ that is a nontrivial permutation of $\vec{\boldsymbol{n}}$ (where by "nontrivial permutation" we mean any permutation other than the identity). This definition does not involve any use of the sign function, and therefore lends itself more easily to theoretical study. This may explain why in previous papers regarding verification [11, 12, 2] virtually all weigh-sets used are homogeneous.

We define *The Baron's homogeneous omni-sequence* analogously to the Baron's omni-sequence. For $n > 3$, $B'(n)$ is the smallest m for which there exists an $m \times n$ homogeneous Münchhausen matrix. This is also the smallest number of weighings required to verify the weights of coins weighing $1, \ldots, n$ grams by use of balance scales that give only the information of whether they balance or tip, but not to which direction they tip.

Because the weighing strategies described in this paper are all homogeneous, we effectively also prove

Theorem 2. *The sequence $B'(n)$ is $\log_3 n + \mathrm{O}(\log \log n)$.*

We conjecture that $\Theta(B(n) - \log_3 n) \equiv \Theta(B'(n) - \log_3 n)$, implying that the added information of the direction to which the scales tip (when they do) is not a significant help in the Münchhausen problem.

2 Special n values

We begin by constructing Münchhausen matrices for some special n values, noting that these constructions are the first known constructions that give a $\log_3 n + o(\log n)$ bound on $B(n)$ for any infinite family of n values. For this, we introduce two new notions: monotonicity and balance.

2.1 Monotonicity

In [11] and [2], all constructions iteratively establish partitions and then sub-partitions of the set of n coins. However, [11] also cites solutions found by computer searching, due to Konstantin Knop and Maxim Kalenkov, that reveal that the optimal strategy (at least for the range where computer searching is feasible) uses a different method. Consider, for example, the following weigh-set for $n = 19$, found by Maxim Kalenkov:

$$M = \begin{pmatrix} - & - & - & - & - & 0 & - & - & 0 & - & 0 & 0 & - & 0 & 0 & + & 0 & + & + \\ - & - & - & 0 & 0 & - & 0 & + & - & + & - & 0 & + & 0 & 0 & - & + & 0 & 0 \\ - & 0 & + & - & 0 & - & + & - & 0 & 0 & + & - & + & 0 & + & 0 & 0 & - & 0 \end{pmatrix},$$

where $+$ indicates 1 and $-$ indicates -1. To demonstrate that this is a Münchhausen matrix, consider the equation

$$(12, 7, 3)M = (-22, -19, -16, -15, -12, -10, -9, -8, -7, -5, -4, -3, -2, 0, 3, 5, 7, 9, 12).$$

The entries of the resulting vector form a strictly increasing list.

Formally, we define a *monotone* matrix to be any matrix, M, for which there exists a vector $\vec{w}$ such that the entries of the vector $\vec{w}M$ are monotone strictly increasing. The general claim is:

Lemma 2.1. *Any homogeneous monotone matrix is a Münchhausen matrix.*

Proof. By definition of a homogeneous matrix, M satisfies $M\vec{\boldsymbol{n}} = \vec{\mathbf{0}}$. Because M is monotone, we know that there exist vectors $\vec{w}$ and $\vec{c}$ such that $\vec{w}M = \vec{c}$ and $\vec{c}$ has monotone strictly-increasing entries. Therefore $\langle \vec{c} \cdot \vec{\boldsymbol{n}} \rangle = \vec{w}M\vec{\boldsymbol{n}} = 0$. The rearrangement inequality [8] states that the dot product of any two vectors, $\vec{x}$ and $\vec{y}$, with strictly-increasing entries is strictly larger than the dot product of $\vec{x}$ with any nontrivial permutation of $\vec{y}$. In particular, for any nontrivial permutation $\vec{v'}$ of $\vec{\boldsymbol{n}}$, $\langle \vec{c} \cdot \vec{v'} \rangle \neq 0$, so $M\vec{v'} \neq \vec{\mathbf{0}}$. □

2.2 Balance

A matrix M is *balanced* if $M\vec{\mathbf{1}} = \vec{\mathbf{0}}$.

Unlike monotonicity, which has roots in the works of Knop and Kalenkov, the concept of balance is an innovation of this paper. Balance allows "good" constructions to be repeated for different n values: if M is a balanced homogeneous $m \times n$ matrix, then $M\vec{v} = \vec{\mathbf{0}}$ not only for $\vec{v} = \vec{\boldsymbol{n}}$, but also for any $\vec{v}$ that is a linear progression of length

n. This allows us to create a new balanced homogeneous weighing $\{L', R'\}$ from an existing balanced homogeneous weighing $\{L, R\}$, by setting $L' = \{ax + b\colon x \in L\}$ and $R' = \{ax + b\colon x \in R\}$ for any integers a and b.

Furthermore, if $\{L_A, R_A\}$ and $\{L_B, R_B\}$ are two balanced homogeneous weighings, then so are $\{L_A \uplus L_B, R_A \uplus R_B\}$, if $L_A \uplus L_B$ and $R_A \uplus R_B$ (the disjoint unions) are well defined and do not intersect.

For matrices A and B, we use the standard block notation $(A\ B)$ and $\binom{A}{B}$ to indicate that A and B are placed side-by-side and one on top of the other, respectively, in a new matrix. Additionally, we denote by $(A \times k)$ and $[A \times k]$ the matrix defined by arranging k copies of A side-by-side and that created by copying each column of A k times, respectively. Otherwise stated: $(A \times k) \stackrel{\text{def}}{=} \vec{\mathbf{1}}_k^T \otimes A$ and $[A \times k] \stackrel{\text{def}}{=} A \otimes \vec{\mathbf{1}}_k^T$, where $\otimes$ signifies the Kronecker product. In all operations described, vectors can be used as matrices.

These operations are useful, because if the operands are balanced homogeneous weigh-sets, then so are the results.

Define the *t-th moment* of a vector $\vec{v}$ to be $\sum_i v_i i^t$. A vector is balanced if its 0-th moment is 0. It is homogeneous if its first moment is 0. We refer to the 0-th moment as the *imbalance*. The first moment will simply be referred to as *the moment*.

2.3 The basic idea

Clearly, both monotonicity and balance are desirable properties in constructing homogeneous Münchhausen matrices. The following hypothetical construction demonstrates the gist of how this is done.

Hypothetical Construct 1. *If M is a homogeneous, monotone, balanced weigh-set of dimensions $m \times n$, and $\{M_i\}_{i=1}^{\infty}$ is defined recursively as*

$$M_1 = M,$$

$$M_{i+1} = \begin{pmatrix} M \times n^i \\ [M_i \times n] \end{pmatrix},$$

then for all i, M_i is a homogeneous, monotone, balanced, Münchhausen weigh-set.

Proof. It is clear that all M_i are homogeneous and balanced. By Lemma 2.1, if M_i is homogeneous and monotone, it is also Münchhausen. We therefore only need to prove that all M_i are monotone.

Consider the vector $\vec{w}$ satisfying $\vec{w}M = \vec{c}$ with a monotone strictly-increasing $\vec{c}$. Choose a number C, larger than $\max(\vec{c}) - \min(\vec{c})$. Define recursively the following vector sequence.

$$\vec{w}_1 = \vec{w},$$

$$\vec{w}_{i+1} = (\vec{w}\ C\vec{w}_i),$$

where $(\vec{a}\ \vec{b})$ indicates that vector $\vec{b}$ is appended to vector $\vec{a}$ and $C\vec{w}$ is scalar multiplication. The vector $\vec{c}_k \stackrel{\text{def}}{=} \vec{w}_k M_k$ has monotone strictly-increasing entries. To see this, note that

the result at column j is $\sum_{l=1}^{i} c_{r(l)} C^l$, where $r(l) \equiv \lceil j/n^{l-1} \rceil \pmod{n}$. One can formally view these sums over all j as an enumeration in base C using the digits $c_1, \ldots, c_n$. The enumeration clearly forms a strictly-increasing list. □

Unfortunately, as it stands, this construction is merely hypothetical because the conditions required of M are never satisfied.

Lemma 2.2. *No homogeneous, balanced weigh-set is monotone.*

Proof. If M is a homogeneous, balanced weigh-set, then $M\vec{\boldsymbol{n}} = \vec{\mathbf{0}}$ and $M\vec{\mathbf{1}} = \vec{\mathbf{0}}$, and therefore also $M(a\vec{\boldsymbol{n}} + b\vec{\mathbf{1}}) = \vec{\mathbf{0}}$ for any choice of a and b. In particular, choosing $a = -1$, $b = n + 1$ we get $a\vec{\boldsymbol{n}} + b\vec{\mathbf{1}} = (n, \ldots, 1)^T$, which is a permutation on $\vec{\boldsymbol{n}}$. This means that there exists a nontrivial permutation on $\vec{\boldsymbol{n}}$, $\vec{v'}$, that yields $M\vec{v'} = \vec{\mathbf{0}}$, so M is not a Münchhausen matrix. By Lemma 2.1, it is not monotone. □

We therefore augment our requirements to "the next best thing". Let a weigh-set be *t-part piecewise-monotone* if its columns can be partitioned into at most t consecutive parts, such that each part forms a monotone weigh-set. A t-part piecewise-monotone weigh-set M is *k-regular* if each monotone part in the column-partition of M is exactly k columns wide. The qualifiers "t-part" and "k-regular" may be omitted if they are understood from the context.

Lemma 2.3. *Let M be a homogeneous, balanced, k-regular piecewise-monotone weigh-set of dimensions $m \times kt$, and let $\{M_i\}_{i=1}^{\infty}$ be a sequence of weigh-sets defined recursively as*

$$M_1 = M,$$

$$M_{i+1} = \begin{pmatrix} M \times k^i \\ [M_i \times k] \end{pmatrix},$$

then for all i, M_i is a homogeneous, balanced, k^i-regular t-part piecewise-monotone weigh-set of dimensions $mi \times k^i t$.

Proof. The proof is essentially the same as for Hypothetical Construct 1. □

Our main theorem in this section is

Theorem 3. *For any $\alpha > 1$ there exists a constant C such that there exist $m \times n$ homogeneous Münchhausen matrices with $m \leqslant \alpha \log_3 n + C$ for arbitrarily large values of n.*

The basis of the proof for Theorem 3 is Lemma 2.3. However, we still need to show that a suitable matrix M (as in Lemma 2.3) can be provided, and that the resulting t-part piecewise-monotone weigh-sets can be made into homogeneous Münchhausen matrices. The first part is discussed in Subsection 2.4. The second part is established by the following lemma.

Lemma 2.4. *Any homogeneous, t-part piecewise-monotone weigh-set can be extended into a Münchhausen weigh-set by adding $\mathrm{O}(t)$ further weighings into the set.*

Proof. Each part of the piecewise-monotone weigh-set is individually monotone. By the reasoning of Lemma 2.1, the weigh-set is a verification subject to its piecewise-monotone partition. To complete the piecewise-monotone weigh-set into a Münchhausen weigh-set, we add weighings that establish this partition. What we need to prove is that $\mathrm{O}(t)$ additional weighings suffice.

We prove, more generally, that for $1 \leqslant j < n$ it is possible to separate the coins $1, \ldots, j$ from the coins $j+1, \ldots, n$ in at most 8 weighings. Repeating this for each of the at-most $t-1$ partition separation points, the full partition will have been established and the proof completed.

To separate the coins $1, \ldots, j$ from the coins $j+1, \ldots, n$, we consider two cases:

Case 1: $2(j^2+j) \geqslant n^2+n$

Let x be the minimum integer such that $x^2+x \geqslant (n^2+n)-(j^2+j)$.

Case 1a: $x^2+x = (n^2+n)-(j^2+j)$

In this case, the set of coins $\{1, \ldots, x\}$ weighs the same as $\{j+1, \ldots, n\}$. Balancing them against each other establishes the partition $\{\{1, \ldots, x\}, \{x+1, \ldots, j\}, \{j+1, \ldots, n\}\}$, and, in particular, the partition we have set out to establish.

This is true because replacing any of the coins $1, \ldots, x$ would have made their pan heavier, whereas replacing any of the coins $j+1, \ldots, n$ would have made their pan lighter.

Case 1b: $x^2+x > (n^2+n)-(j^2+j)$

In this case, weighing $L = \{1, \ldots, x\}$ against $R = \{j+1, \ldots, n\}$ would have resulted in an imbalance of y, $1 \leqslant y < x$, in favor of the former. Instead, as a first weighing we balance $L \setminus \{y\}$ against R. To establish the partition, we must now merely verify the identity of coin y. However, in [12] it was already shown that at most 7 weighings are necessary to verify the weight of any single coin.

Case 2: $2(j^2+j) < n^2+n$

In this case, we can no longer use the system described above, because there would be overlaps between $1, \ldots, x$ and $j+1, \ldots, n$. Instead, we pick x as the maximal value that satisfies $x^2+x \leqslant (n^2+n)-(j^2-j)$. Note the use of j^2-j, not j^2+j.

Case 2a: $x^2+x \geqslant (n^2+n)-(j^2+j)$

Let $y = ((j^2+j)+(x^2+x)-(n^2+n))/2$. The value of y lies in the range $0 \leqslant y \leqslant j$. The weight of y can be verified in at most 7 weighings, following which balancing $\{1, \ldots, j\} \setminus \{y\}$ against $\{x+1, \ldots, n\}$ establishes the partition.

Case 2b: $x^2+x < (n^2+n)-(j^2+j)$

Balance $\{1, \ldots, j\}$ against $\{y, x+2, \ldots, n\}$, where y is chosen to satisfy equality of masses. The value of y is in the range $j+1 \leqslant y \leqslant x$, so the weighing is

well-defined. We complete the process, again, by verifying the weight of y in at most 7 additional weighings.

□

2.4 Constructing a base matrix

We now turn to the construction of the base matrix, M.

Lemma 2.5. *For every $m \geqslant 2$, there is a k-regular piecewise-monotone, homogeneous, balanced weigh-set of dimensions $m \times kt$, where $k > 3^{m-1}$ and $t \leqslant 5k$.*

We remark that any bound on t that is a polynomial in k would have sufficed to prove that $B'(n)$ is $\log_3 n + \mathrm{O}(\log\log n)$. We prove for a low degree polynomial because it improves the constants used in the $\mathrm{O}(\log\log n)$, but even so this bound is not tight.

Proof. We construct explicitly a matrix, M, satisfying the conditions of the lemma.

Let $\vec{w}$ of length m be defined by

$$w_i = \begin{cases} 2 \cdot 3^{m-1-i} & \text{if } i < m, \\ 1 & \text{if } i = m. \end{cases}$$

This will be the vector satisfying that $\vec{c} = \vec{w}M$ is monotone strictly-increasing in each $[ik+1, \ldots, (i+1)k]$ interval.

We choose k to be the largest prime number less than $2 \cdot 3^{m-1}$. To satisfy the conditions of the lemma we require that this value satisfies $3^{m-1} < k < 2 \cdot 3^{m-1}$, which is guaranteed by Bertrand's postulate, asserting the existence of a prime between x and $2x$. We note that for $m = 2$ we have $k = 5$, and that for $m > 2$ both upper and lower bounds are composite numbers. Hence, the inequalities are sharp in all cases.

Because M is required to be homogeneous, we know that for each of its rows, M_{i*}, we have $\langle M_{i*} \cdot \vec{\boldsymbol{n}} \rangle = \vec{\mathbf{0}}$. Because it is required to be balanced, we have $\langle M_{i*} \cdot \vec{\mathbf{1}} \rangle = \vec{\mathbf{0}}$. Both of these conditions are linear, so we know that they must also hold for $\vec{c}$. We begin by designing such a vector.

Vector $\vec{c}$ will be several concatenated copies of a vector $\vec{c'}$ of length k^2. By choosing $\vec{c'}$ to be balanced and homogeneous, $\vec{c}$ is also ensured to have these properties. Algorithm 1 shows how to construct $\vec{c'}$.

Each $[ik+1, \ldots, (i+1)k]$ interval in $\vec{c}$ is chosen by Algorithm 1 to hold one of the following possible lists of values, each of which is monotone strictly-increasing: $(-(k+1)/2, \ldots, (k-3)/2)$, $(-(k-1)/2, \ldots, (k-1)/2)$ or $(-(k-3)/2, \ldots, (k+1)/2)$, noting that k is a large prime and therefore odd. We call the first "choice -1", the second "choice 0" and the third "choice $+1$", according to the value of their associated h'_{i+1}. These choices all satisfy the monotonicity requirement. By choosing the same number of "choice -1" and "choice $+1$", balance (i.e., $\langle \vec{c} \cdot \vec{\mathbf{1}} \rangle = 0$) is also guaranteed.

The remaining condition is homogeneity: $\langle \vec{c'} \cdot \vec{\boldsymbol{n}} \rangle = 0$. Let us consider the vector $\vec{\tilde{c}}$ that would have been returned by Algorithm 1 had the "while" loop of step 5 been stopped after fewer iterations than is required by the algorithm.

Algorithm 1 Finding the vector $\vec{c'}$ as a function of k

1: $\vec{h'} \leftarrow \vec{\mathbf{0}}_k$
2: $T \leftarrow (k-1)(k+1)/12$
3: $s \leftarrow 1$
4: $f \leftarrow k$
5: **while** $T > 0$ **do**
6: **if** $T < f - s$ **then**
7: $f \leftarrow T + s$
8: **end if**
9: $h'_s \leftarrow 1$
10: $h'_f \leftarrow -1$
11: $T \leftarrow T - (f - s)$
12: $s \leftarrow s + 1$
13: $f \leftarrow f - 1$
14: **end while**
15: **for** $i \in 0, \ldots, k-1$ **do**
16: **for** $j \in 0, \ldots, k-1$ **do**
17: $c'_{ik+j+1} \leftarrow -(k-1)/2 + j + h'_{i+1}$
18: **end for**
19: **end for**
20: **return** $(c'_1, \ldots, c'_{k^2})$

If terminated after 0 iterations, $\vec{\tilde{c}}$ would have been the vector with all "choice 0" choices. Its moment, $\langle \vec{\tilde{c}} \cdot \vec{\boldsymbol{n}} \rangle$, can be calculated as $2kP_{(k-1)/2}$, where P_r is the r'th square pyramidal number,

$$P_r = 1^2 + \cdots + r^2 = r(r+1)(2r+1)/6,$$

because each $[ik+1, \ldots, (i+1)k]$ interval contributes $2P_{(k-1)/2}$ to the moment. This equals $k^2(k-1)(k+1)/12$.

At each iteration of the "while" loop, the moment is decreased by incrementing by 1 a list of k consecutive values starting at position $sk+1$, and decrementing by 1 a list of k consecutive values starting at position $fk+1$. This decreases the moment by $k^2(f-s)$.

The value of T in Algorithm 1 is initialized at step 2 and decreased at 11. These updates of T maintain the invariant $\langle \vec{\tilde{c}} \cdot \vec{\boldsymbol{n}} \rangle = k^2 T$. Because we know that $T = 0$ when the "while" loop terminates, $\langle \vec{c'} \cdot \vec{\boldsymbol{n}} \rangle = 0$, as required.

To prove that the algorithm terminates, one can simply compute the number of iterations required to cancel the initial moment. This number is $\lceil k/2 - \sqrt{k^2/6 + 1/12} \rceil$. Not only does this prove a successful termination of the algorithm, it also indicates that the middle $\lfloor k\sqrt{2/3} \rfloor$ parts of $\vec{c'}$ have at most one non-"0" choice.

Algorithm 2 shows how to construct matrix M.

By construction, when Algorithm 2 terminates, its output is a k-regular piecewise-monotone, homogeneous, balanced weigh-set of dimensions $m \times kt$, where $k > 3^{m-1}$.

Algorithm 2 Finding M as a function of $\vec{c'}$

1: The vector $\vec{w}$ is as defined earlier.
2: N' is a $0 \times k^2$ matrix.
3: N is a 0×0 matrix.
4: $\vec{c} \leftarrow \vec{\mathbf{0}}_0$
5: {N', N and $\vec{c}$ initially contain no elements, but we use them as recursion bases.}
6: **for** $i \in 1, \ldots, m$ **do**
7: **loop**
8: $\vec{c} \leftarrow (\vec{c}\ \vec{c'})$
9: $N \leftarrow (N\ N')$
10: $len \leftarrow$ length of $\vec{c}$
11: $\vec{v} \leftarrow \vec{\mathbf{0}}_{len}$
12: $\vec{r} \leftarrow \vec{c} - \sum_{l=1}^{i-1} w_l N_{l*}$
13: **for** $R \subseteq \{j \colon r_j = 3^{m-i-1}\}$ and $L \subseteq \{j \colon r_j = -3^{m-i-1}\}$ subject to $|R| = |L|$ **do**
14: **for** $j \in 1, \ldots, len$ **do**
15: **if** $j \in R$ or $r_j > 3^{m-i-1}$ **then**
16: $v_j \leftarrow 1$
17: **else if** $j \in L$ or $r_j < -3^{m-i-1}$ **then**
18: $v_j \leftarrow -1$
19: **else**
20: $v_j \leftarrow 0$
21: **end if**
22: **end for**
23: **if** $\langle v \cdot \vec{n} \rangle = 0$ **then**
24: **go to 28**
25: **end if**
26: **end for**
27: **end loop**
28: $N' \leftarrow \binom{N}{\vec{v}}$
29: $\vec{c'} \leftarrow \vec{c}$
30: N is an $i \times 0$ matrix.
31: $\vec{c} \leftarrow \vec{\mathbf{0}}_0$
32: **end for**
33: $M \leftarrow N'$
34: **return** M

To see this, note that Algorithm 2 maintains as an invariant throughout its execution that the absolute value of any element of $\vec{r}$ is at most 3^{m-i}. At the termination of the loop at step 6, $\vec{r}$, which at this time computes $\vec{c} - \vec{w}M$, will have become $\vec{\mathbf{0}}$. The equation $\vec{w}M = \vec{c}$ ensures all required monotonicity requirements. Homogeneity is ensured explicitly at step 23 and balance is a result of the invariant maintained throughout that the number of appearances of any value, x, in $\vec{r}$ is the same as the number of appearances

of $-x$. Under this invariant, the choice $|R| = |L|$ both ensures that each $\vec{v}$ tried in step 23 is balanced and ascertains that the invariant is retained in all future steps.

In order to complete the proof of the lemma, what still needs proving is that the algorithm terminates at all, and that it terminates with a sufficiently low t. We begin by proving termination.

The question of Algorithm 2's termination is the question of whether suitable R and L are always eventually found in the for-loop of step 13 that will allow escape from the endless loop of step 7 by satisfying the condition of step 23.

We separate here two cases: $i = m$ and $i \neq m$. At the last iteration of the main for-loop (step 6), the condition of step 23 is met immediately in the first pass through the loop of step 7 and using the only choices for R and L possible in the for loop of step 13, namely $R = L = \emptyset$.

The reason for this is that, as previously noted, Algorithm 2 maintains as an invariant throughout its execution that the absolute value of any element of $\vec{r}$ is at most 3^{m-i}. At $i = m$, $\vec{r}$ is therefore a $\{-1, 0, 1\}$-vector, so at the for-loop of step 14, $\vec{v}$ is chosen to be $\vec{r}$, reducing $\vec{w}M - \vec{c}$ to zero. (Throughout the process, $\vec{r}$ keeps a tally of this difference.) By construction $\langle \vec{c} \cdot \vec{\boldsymbol{n}} \rangle = 0$, and also each row of M other than the last is homogeneous. By linearity, we conclude that the last row of M must also be homogeneous.

For the case $i < m$, let us consider a variation on Algorithm 2, where the comprehensive search over all possible R and L combinations presented in step 13 is replaced by a much more efficient, though possibly sub-optimal, search. This change definitely cannot make Algorithm 2 terminate if it had otherwise not terminated, nor can it decrease the value of t at termination.

The simpler method to come up with (R, L) pairs is divided into three stages. First, we initialize $\vec{v}$. In a second step, we reduce the moment of $\vec{v}$ to almost zero. (Specifically, at the end of the process its absolute value is less than $2k$.) Finally, we zero the moment out completely.

We describe the process here as an algorithm that chooses $\vec{v}$ directly, not going through the process of picking out R and L first. The initialization is described in Algorithm 3. The rough correction stage is described in Algorithm 4. The fine correction stage is then described in Algorithm 5.

Let $\vec{v^0}$ be the value returned by Algorithm 3.

Algorithm 4 attempts to cancel out the moment of $\vec{v^0}$ by a method similar to that used in Algorithm 1 for the generation of $\vec{c'}$. Set R is assigned the lowest possible values, and set L is assigned the highest ones, subject to not overshooting the moment's target zone. Though it may be that $|R|$ and $|L|$ cannot be made large enough to counter all existing moment, in which case the algorithm will fail, reaching step 14, the one thing ensured by the algorithm is that it will not overshoot the mark: because there is a value that can be added to L at every interval of length k, the moment is decreased by at most $2k$ at every step, ensuring that the target zone is not missed.

Though this algorithm may not be able to cancel out the initial moment, repeatedly appending copies of this basic $\vec{v}$ to each other (which is what is done by the loop in step 7 of Algorithm 2) is sure to eventually result in a vector for which this Algorithm 4

Algorithm 3 Initializing $\vec{v}$

```
1: for j ∈ 1,...,s do
2:    if r_j > 3^{m-i-1} then
3:       v_j ← 1
4:    else if r_j < -3^{m-i-1} then
5:       v_j ← -1
6:    else
7:       v_j ← 0
8:    end if
9: end for
10: return v⃗
```

terminates successfully. Suppose that the vector's original length was kl and that it was appended to itself r times. If the initial moment was b, after r repetitions it has become rb. However, because there is a candidate for R and a candidate for L in every interval of length k, Algorithm 4 is sure to be able to cancel out a moment of order $\mathrm{O}(kr^2l^2)$. For a large-enough r, the moment-countering ability, growing as a quadratic function of r, will suffice to cancel the moment, which grows as a linear function of r, and the algorithm will terminate successfully.

Furthermore, by appending yet additional copies of the initial $\vec{v}$, countering the moment generated by r copies will require the cardinalities of R and L (which are equal) to be $\mathrm{O}(\sqrt{r})$. This means that in increasing r we also increase, without bound, the number of intervals that are unused by both R and L, and are therefore free to be used by Algorithm 5.

We now move to analyzing the fine-tuning algorithm. We prove that given enough "free" intervals (as guaranteed, given a large enough r), the fine-tuning algorithm terminates successfully. Algorithm 5 describes the fine-tuning algorithm for the case $moment > 0$. The case $moment < 0$ is symmetric, with the roles of negatives and positives reversed. (And the case $moment = 0$ requires no further moment-cancellation, of course.)

The way that Algorithm 5 works is by seeking pairs of potential values for R and for L, (r, l), such that $l - r = pos$ and such that $r - l = neg$. It then uses the minimal amount of each type of pair needed in order to cancel out the remaining moment. Because we chose k to be prime, pos and neg, satisfying $pos + neg = k$, are necessarily co-prime. Therefore, there is necessarily some $nump < neg$ that satisfies $neg|(moment + nump \cdot pos)$. The number of neg needed, $numn$ is then bounded by $nump \cdot pos/neg + moment/neg$, where the first part is bounded by pos and the second part by 4, because it is known that $neg > pos$, and therefore $neg > k/2$, whereas $moment < 2k$.

We now need to ensure that enough pairs with differences pos and neg exist. By the construction of $\vec{c}$, a pair with a difference of pos exists in every $[ik+1, \ldots, (i+1)k]$ interval not used by the rough-cancellation step. To obtain a difference of neg, we need two such consecutive intervals with the same "choice". Note, however, that in $\vec{c'}$ there are at most 5 cases where consecutive "choice"s are different (when considering $\vec{c'}$ as cyclic, which is how it is used in constructing $\vec{c}$).

Algorithm 4 Rough correction of moment

$moment \leftarrow \langle \vec{v} \cdot \vec{\boldsymbol{n}} \rangle$
$pos \leftarrow 2 \cdot 3^{m-i-1}$
$neg \leftarrow k - pos$
if $pos < neg$ **then**
  $target \leftarrow \{0, \ldots, 2k-1\}$
else
  $target \leftarrow \{1-2k, \ldots, 0\}$
end if
while $moment \notin target$ **do**
  $pvals \leftarrow \{l \colon r_l = 3^{m-i-1} \text{ and } v_l = 0\}$
  $nvals \leftarrow \{l \colon r_l = -3^{m-i-1} \text{ and } v_l = 0\}$
  if $pvals = \emptyset$ **then**
    {Failed to correct the moment.}
    quit
  end if
  if $moment > 0$ **then**
    $s \leftarrow \min(pvals)$
    $f \leftarrow \max(nvals)$
    $x \leftarrow \min(nvals)$
    while $moment - (x - s) > \max(target)$ and $x \neq f$ **do**
      $nvals \leftarrow nvals \setminus \{x\}$
      $x \leftarrow \min(nvals)$
    end while
    $v_s \leftarrow 1$
    $v_x \leftarrow -1$
    $moment \leftarrow moment - (x - s)$
  else
    $s \leftarrow \min(nvals)$
    $f \leftarrow \max(pvals)$
    $x \leftarrow \min(pvals)$
    while $moment - (s - x) < \min(target)$ and $x \neq f$ **do**
      $pvals \leftarrow pvals \setminus \{x\}$
      $x \leftarrow \min(pvals)$
    end while
    $v_s \leftarrow -1$
    $v_x \leftarrow 1$
    $moment \leftarrow moment - (s - x)$
  end if
end while
return $\vec{v}$

Algorithm 5 Fine correction of the moment

Require: $moment > 0$

$pvals \leftarrow \{l\colon r_l = 3^{m-i-1}$ and $v_l = 0\}$
$nvals \leftarrow \{l\colon r_l = -3^{m-i-1}$ and $v_l = 0\}$
$nump \leftarrow$ minimum nonnegative integer such that $neg | moment + nump \cdot pos$
$numn \leftarrow (moment + nump \cdot pos)/neg$
$pairs \leftarrow \{l\colon l \in pvals$ and $l - pos \in nvals\}$
if $|pairs| < nump$ **then**
  {Failed to correct the moment.}
  quit
end if
for $l \in nump$ lowest elements of $pairs$ **do**
  $v_l \leftarrow 1$
  $v_{l-pos} \leftarrow -1$
end for
$pvals \leftarrow \{l\colon r_l = 3^{m-i-1}$ and $v_l = 0\}$
$nvals \leftarrow \{l\colon r_l = -3^{m-i-1}$ and $v_l = 0\}$
$pairs \leftarrow \{l\colon l \in pvals$ and $l + neg \in nvals\}$
if $|pairs| < numn$ **then**
  {Failed to correct the moment.}
  quit
end if
for $l \in numn$ lowest elements of $pairs$ **do**
  $v_l \leftarrow 1$
  $v_{l+neg} \leftarrow -1$
end for
return $\vec{v}$

Because the fine-tuning algorithm is not guaranteed to be able to use any interval already utilized at the rough-cancellation step, consider the situation of the used intervals at the end of that previous step. The rough-cancellation algorithm uses the intervals of $\vec{v}$ corresponding to the lowest and highest valued elements of $\vec{\boldsymbol{n}}$, but it leaves in the middle one consecutive portion of unused intervals, with at most one used interval interrupting it.

We claim that if this consecutive portion is of length $2k + 13$ intervals or more, this guarantees that the fine-tuning algorithm will terminate successfully. The case that requires the greatest number of intervals for the fine-tuning step is $pos = k - 1$, $neg = 1$, with the original moment being $2k - 1$. This requires $k - 2$ *neg*-pairs (each of which requires 2 intervals) and 3 *pos*-pairs (each of which requires a single interval), for a total of $2k - 1$ intervals. Given that in any consecutive set of at most $3k$ intervals there cannot be more than 15 "choice" changes, and in each such "choice" change there is at most one interval that cannot be used to create a *neg*-sized interval, the total number of intervals needed can be bounded by the following computation: $2k - 4$ (number of intervals used

in *neg* pairs) +15 (number of intervals wasted over "choice" changes) +2 (number of intervals wasted because they or the interval after them was used in the rough-cancellation stage) $= 2k+13$. We note that in this worst-case scenario there is no need to add 3 more intervals for the *pos* pairs, as the *pos* pairs can use any of the 15 intervals unusable by *neg* pairs due to "choice" changes.

For large enough m we have $2k + 13 \leqslant 3k$, so the argument holds. The only m for which this does not hold is $m = 2$. We now show that for $m \geqslant 4$, $\vec{c} = (\vec{c'} \times 5)$ suffices for Algorithm 4 to leave this many unused intervals, ensuring the successful termination of Algorithm 2 with matrix dimensions meeting the condition $t \leqslant 5k$ of the lemma. We treat the case $m < 4$ separately.

First, let us consider the size of the moment that needs to be counteracted at each step. Our original $\vec{c'}$, of length k^2, was composed of k repetitions of either $-(k-1)/2, \ldots, (k-1)/2$ or $-(k-3)/2, \ldots, (k-3)/2$, and also at most $(k-1)/2$ pairs of $-(k+1)/2$ and $(k+1)/2$, values that appear, in total, at most once per k-length interval.

Consider the repetitions of $-(k-1)/2, \ldots, (k-1)/2$ or $-(k-3)/2, \ldots, (k-3)/2$ in $\vec{c}$. Each such repetition includes one $-x$ for every x. As a result, $\vec{v^0}$, taken over the same indices, is balanced. The greatest moment that can occur in such a segment is $(k^2-1)/4$, for a total of $(k^2-1)/4 \cdot k = (k^3-k)/4$ per repetition of $\vec{c'}$. The remaining $(-(k+1)/2, (k+1)/2)$ and $(-(k-1)/2, (k-1)/2)$ value pairs also result in a balanced set. Consider their moment. This moment is the result of "choice -1" and "choice $+1$" pairs taken at the $\vec{c'}$ determination stage. In designing $\vec{c'}$ we needed to cancel out a moment of $(k-1)k^2(k+1)/12$. For this, we chose (s, f) pairs, and made the s-interval into a "choice $+1$" and the f-interval into a "choice -1". This cancels out exactly $(f-s)k^2$. In terms of $\vec{v^0}$, such a move creates, at most, $2(f-s+1)k$ of moment. Knowing that the sum of all $f-s$ is $(k-1)(k+1)/12$, the total is bounded by $(k-1)k(k+1)/6 + 2pk$, where p is the number of pairs used, which cannot be larger than $(k-1)/2$. Adding all this together and multiplying by 5, because the claim is that $\vec{c}$ has at most 5 repetitions of $\vec{c'}$, we get $\frac{25}{12}k^3 + 5k^2 - \frac{85}{12}k$. The total moment for $\vec{v^0}$ at step $i = 1$ of Algorithm 2 is therefore bounded by $\frac{25}{12}k^3 + 5k^2 - \frac{85}{12}k$.

The choice of (R, L) at $i = 1$ must therefore create a $\vec{v}$ with at most this moment, because the two moments must cancel each other exactly. Let us call the moment created by the choice of (R, L) at iteration i by the name b_i. We have $b_1 \leqslant \frac{25}{12}k^3 + 5k^2 - \frac{85}{12}k$.

In subsequent steps, however, the moment can be significantly higher. The reason for this is that the values of $\vec{v^0}$ in all subsequent steps are reversed by the choice of (R, L) at all indices in $R \cup L$. (A value of 3^i can be described in one of two ways. It is either $\sum_{j=m-i}^{m} w_j$ or $w_{m-i-1} - \sum_{j=m-i}^{m} w_j$. Adding a position to R adds w_{m-i-1} to the equation, and therefore reverses all subsequent choices.) This adds a further $2b_1$ to the moment that needs to be corrected by b_2, so $|b_2| \leqslant 3b_1$. In total, we have the equation $|b_i| \leqslant \left(\frac{25}{12}k^3 + 5k^2 - \frac{85}{12}k\right) 3^{i-1}$.

On the other hand, consider how much moment can be countered at each step. In $\vec{c}$, every k-sized interval includes all numbers from $-(k-3)/2$ to $(k-3)/2$, and among them all the numbers from $-(3^{m-1}-1)/2$ to $(3^{m-1}-1)/2$, being a total of 3^{m-1} consecutive numbers. At iteration i of Algorithm 2, the candidates for R are exactly those positions

whose c_j is congruent to $3^{m-i-1} \pmod{2 \cdot 3^{m-i}}$ and the candidates for L are exactly those positions whose c_j is congruent to $-3^{m-i-1} \pmod{2 \cdot 3^{m-i}}$. (Only w_m is odd. All other w_i are even. However, $w_1/2, \ldots, w_{m-1}/2$ form the powers of 3, and it is well known that any integer can be expressed uniquely in base 3 using the digits $\{-1, 0, 1\}$. Therefore, every odd number in the range can be expressed by exactly 2 combinations of w_i, depending on whether w_m is added or subtracted. The numbers that are potentials for R in iteration i of Algorithm 2 are exactly those $2x + 1$ for which x has a 0 in its i'th digit in $\{-1, 0, 1\}$ base-3 representation and $x+1$ has a 1. Similarly, candidates for L are those where x has a -1 and $x + 1$ has a 0.)

Careful accounting shows that the number of candidates for R in every interval of length k is $(3^{i-1}+1)/2$. This is given by the following computation. We have $(3^{i-1}-1)/2$ due to this many full sets of all residues modulo $2 \cdot 3^{m-i}$. In addition, we have one more candidate due to the fact that remaining $\vec{c}$ entries include all residues in the range $0, \ldots, (3^{m-i}-1)/2$, and so, in particular, they also include one that is congruent to $3^{m-i-1} \pmod{2 \cdot 3^{m-i}}$. The total, $(3^{i-1} + 1)/2$, is also the number of candidates for L.

Suppose that all candidates in the first $(3k - 13)/2$ intervals were placed in set R and all candidates in the last $(3k - 13)/2$ intervals were placed in set L. This would still leave $2k + 13$ free intervals, which we know is enough for fine tuning. To see how much moment can be corrected by this choice, note that we can pair each member r of R with a member l of L that is exactly $(7k + 13)/2$ intervals away from it. Even if l and r are at their closest positions within their respective intervals, this is still a difference of $(7k^2 + 11k)/2$. The total correctable moment is bounded from below by the product of this difference and the number of candidates used, $(3^{i-1} + 1)/2 \cdot (3k - 13)/2$. The product is $(3^{i-1} + 1)(\frac{21}{8}k^3 - \frac{29}{4}k^2 - \frac{143}{8}k)$, which is more than $3^{i-1}(\frac{21}{8}k^3 - \frac{29}{4}k^2 - \frac{143}{8}k)$. The algorithm succeeds if this is enough to counterbalance an initial moment of b_i, which is bounded from above by $\left(\frac{25}{12}k^3 + 5k^2 - \frac{85}{12}k\right) 3^{i-1}$.

Success of the algorithm depends, therefore, on the correctness of the inequality

$$\frac{25}{12}k^3 + 5k^2 - \frac{85}{12}k \leqslant \frac{21}{8}k^3 - \frac{29}{4}k^2 - \frac{143}{8}k,$$

or, equivalently, $13k^2 - 294k - 259 \geqslant 0$. This is true for $k \geqslant 24$, so in particular for the k values chosen for any $m \geqslant 4$.

We have shown that for $m \geqslant 4$, if the width of the matrices of Algorithm 2 ever reaches $5k^2$ (being 5 times the width at the start of the algorithm) the loop of step 7 will always be stopped at its very first iteration at step 23. The width will not increase beyond this point. We can therefore tweak Algorithm 2 to begin by choosing $\vec{c} = (\vec{c'} \times 5)$, thereby ensuring that this will also be the final width.

Algorithm 6 summarizes this final algorithm.

Algorithm 6 is guaranteed to work for $m \geqslant 4$. However, by simply running it on a computer it was possible to ascertain that it terminates successfully also in the cases $m \in \{2, 3\}$. Therefore, the lemma is proved for all $m \geqslant 2$. □

We can now complete the proof of this section's main claim, as follows.

Algorithm 6 Finding M, final

Find $\vec{c'}$ as in Algorithm 1.
$\vec{c} \leftarrow (\vec{c'} \times 5)$
$len \leftarrow 5k^2$
$M \leftarrow \mathbf{0}_{m \times len}$
$\vec{w} \leftarrow (2 \cdot 3^{m-2}, \ldots, 2, 1)$.
for $i \in 1, \ldots, m$ **do**
    $\vec{v} \leftarrow \vec{\mathbf{0}}_{len}$
    $\vec{r} \leftarrow \vec{c} - \sum_{l=1}^{i-1} w_l M_{l*}$
    Initialize $\vec{v}$ as in Algorithm 3
    Update $\vec{v}$ as in Algorithm 4.
    if $moment > 0$ **then**
        Update $\vec{v}$ as in Algorithm 5.
    else if $moment < 0$ **then**
        Update $\vec{v}$ as in Algorithm 5, with positives and negatives reversed.
    end if
    $M_{i*} \leftarrow \vec{v}$
end for
return M

Proof of Theorem 3. Let M be any k-regular piecewise-monotone, homogeneous, balanced weigh-set of dimensions $m \times kt$, then the sequence $\{M_i\}_{i=1}^{\infty}$ defined by Lemma 2.3 is a sequence of t-part piecewise-monotone weigh-sets with dimensions $m_i = mi$ and $n_i = k^i t$, being arbitrarily large dimensions.

By Lemma 2.4, adding $\mathrm{O}(t)$ further weighings to these weigh-sets is enough to make them Münchhausen. This is a number independent of n. We therefore have Münchhausen weigh-sets of dimensions $m'_i \times n_i$ equal to $(mi + \mathrm{O}(t)) \times (k^i t)$. These all satisfy $m'_i \leqslant \alpha \log_3 n_i + C$ with $\alpha = m/\log_3 k$.

All that is left is to prove that $m/\log_3 k$ can be made arbitrarily close to 1. However, recall that in Lemma 2.5 M was constructed with $k > 3^{m-1}$, meaning that $\alpha < m/(m-1)$, which can be made arbitrarily close to 1. □

Though not necessary for the main proofs of the paper, it is still interesting to note that Theorem 3 immediately implies the following.

Corollary 2.1. *There exists a sequence n values for which $B(n)$ is $\log_3 n + \mathrm{o}(\log n)$.*

3 General n values

We now turn to the question of arbitrary n values. Dealing with these requires strengthening the tools built in Section 2.

Let $\mathcal{M}(m, k, t, n)$ be the set of homogeneous, $m \times n$ matrices (weigh-sets), such that $M \in \mathcal{M}(m, k, t, n)$ if and only if there exists a vector $\vec{w}$ of length m, such that $\vec{c} = \vec{w}M$

satisfies the following properties:

$$\begin{aligned} &c_i \leqslant c_{i+1} \quad (i \not\equiv n \pmod t), \\ &c_i < c_{i+1} \quad (i \equiv n \pmod t \text{ but } i \not\equiv n \pmod{kt}). \end{aligned}$$

This definition of $\mathcal{M}(m,k,t,n)$ is a generalization of monotonicity. If M is a k-regular t-part piecewise monotone, homogeneous matrix, then $M \in \mathcal{M}(m,k,1,kt)$. Furthermore, $[M \times r]$ is in $\mathcal{M}(m,k,r,rkt)$ and $([M \times r] \times s)$ is in $\mathcal{M}(m,k,r,rkts)$. In Section 2, we proved that by joining $\mathcal{M}(m,k,k^i,k^r t)$ weigh-sets for $i = 0,\ldots,r-1$, one can construct a Münchhausen weigh-set of dimensions $(mr + \mathrm{O}(t)) \times k^r t$. In this section, we use the more generalized form in order to replace $k^r t$ by a general n.

However, before this can be done, we must first prove that the sets we use are nonempty.

Lemma 3.1. *For every $m \geqslant 3$ and any positive t, there exist a $k > 3^{m-1}$, such that $\mathcal{M}(m,k,t,n)$ is nonempty for $n \geqslant 5tk^2$.*

Proof. The proof for Lemma 3.1 follows the general outline of the construction in the proof for Lemma 2.5. We begin by choosing the same $\vec{w}$ and k as before, construct a suitable $\vec{c}$, and then build a matrix $M \in \mathcal{M}(m,k,t,n)$ row by row.

The final algorithm used will follow the framework of Algorithm 6. As is done there, the length of $\vec{c}$ here is known in advance. In our case, it is n. In this case, there is no need to first construct $\vec{c'}$ and then duplicate it, as was done in Algorithm 2. Instead, we construct $\vec{c}$ directly. The process for doing so is similar to that described in Algorithm 1. However, three complications need to be accounted for:

1. The matrix width, n, may not be a multiple of kt. For this reason, a construction that duplicates a basic construction of any width is not possible. In the construction used here we will follow Algorithm 1 for the last $\lfloor n/kt \rfloor\, kt$ columns, and will have special handling for the first $n \bmod kt$ ones.

2. The moment that needs to be counterbalanced in Algorithm 1 is known to be a multiple of k^2 (the equivalent of which in this algorithm would be a multiple of k^2t^2), and can therefore be tackled by "choice $+1$" and "choice -1" pairs. Here, the initial moment is general. We handle this by first reducing the moment from its original value, I, to $I \bmod k^2t^2$, as was done before. Next, we use Lagrange's four-square theorem [9] to represent the remainder as a sum of 4 squares (each of which is necessarily smaller than k^2t^2). A moment of a^2, for $a \leqslant kt$, can be created by shifting by 1 the a highest values in one kt-length interval, and by -1 the lowest values in the kt-length interval immediately following it.

The new algorithm for choosing $\vec{c}$ is presented as Algorithm 7.

There are two potential pitfalls in Algorithm 7. First, in step 26, we need to make sure that the "while" loop terminates successfully. This has already been discussed regarding Algorithm 1, but we repeat the discussion here briefly. The total moment when entering

Algorithm 7 Finding the vector $\vec{c}$ as a function of k, t and n

1: $b \leftarrow \lfloor n/kt \rfloor$
2: $\vec{c} \leftarrow \vec{\mathbf{0}}_n$
3: {Choosing the last bkt elements of $\vec{c}$.}
4: **for** $i \in 0, \ldots, b-1$ **do**
5: **for** $j \in 0, \ldots, k-1$ **do**
6: **for** $r \in 1, \ldots, t$ **do**
7: $c_{n-bkt+ikt+jt+r} \leftarrow -(k-1)/2 + j$
8: **end for**
9: **end for**
10: **end for**
11: {Choosing the remaining elements of $\vec{c}$.}
12: $rem \leftarrow n - bkt$
13: $offset \leftarrow \lfloor rem/2t \rfloor$
14: $parity \leftarrow 1 - (\lfloor rem/t \rfloor \mod 2)$
15: **for** $j \in 0, \ldots, (offset \times t) - 1$ **do**
16: $c_{n-bkt-j} \leftarrow offset - \lfloor j/t \rfloor$
17: **end for**
18: **for** $j \in offset \times t, \ldots, rem - 1$ **do**
19: $c_{n-bkt-j} \leftarrow offset - \lfloor j/t \rfloor - parity$
20: **end for**
21: {Rough moment cancellation.}
22: $\vec{h} \leftarrow \vec{\mathbf{0}}_b$
23: $moment \leftarrow \sum_{i=1}^{n} c_i i$
24: $s \leftarrow 1$
25: $f \leftarrow b$
26: **while** $moment \geqslant k^2t^2$ **do**
27: **if** $moment < (f-s)k^2t^2$ **then**
28: $f \leftarrow \lfloor moment/k^2t^2 \rfloor + s$
29: **end if**
30: $h_s \leftarrow 1$
31: $h_f \leftarrow -1$
32: $moment \leftarrow moment - (f-s)k^2t^2$
33: $s \leftarrow s + 1$
34: $f \leftarrow f - 1$
35: **end while**

the "while" loop at step 26 is bounded by $(k-1)k(k+1)(b+1)t^2/12$. (Here, multiplication by b stems from the b complete kt-sized intervals, and the extra $+1$ stems from the effect of the remaining elements. Explicitly computing the moment generated by these elements shows that it can never be greater than that of a complete interval.) In all but the last iteration, the counterbalance introduced by adding a new pair of $(+1, -1)$-choices is

36: **for** $i \in 0, \ldots, b-1$ **do**
37: **for** $j \in 1, \ldots, kt$ **do**
38: $c_{n-bkt+ikt+j} \leftarrow c_{n-bkt+ikt+j} + h_{i+1}$
39: **end for**
40: **end for**
41: {Fine moment cancellation.}
42: $s \leftarrow s-1$
43: Find a_1, a_2, a_3 and a_4 satisfying $a_1^2 + a_2^2 + a_3^2 + a_4^2 = \mathit{moment}$.
44: **for** $i \in 1, \ldots, 4$ **do**
45: **repeat**
46: $s \leftarrow s+1$
47: **until** $h_s = h_{s+1} = 0$
48: **for** $j \in 0, \ldots, a_i - 1$ **do**
49: $c_{n-bkt+skt-j} \leftarrow c_{n-bkt+skt-j} + 1$
50: $c_{n-bkt+skt+j+1} \leftarrow c_{n-bkt+skt+j+1} - 1$
51: **end for**
52: $s \leftarrow s+1$
53: **end for**
54: **return** $\vec{c}$

known. The first pair counterbalances by $(b-1)k^2t^2$, the next by $(b-3)k^2t^2$, and so on. The number of pairs needed, p, not including the last pair, is therefore $p = \lfloor \tilde{p} \rfloor$, where $\tilde{p}$ is the lowest solution to

$$bk^2t^2\tilde{p} - k^2t^2\tilde{p}^2 = (k-1)k(k+1)(b+1)t^2/12,$$

or, equivalently, to

$$k\tilde{p}^2 - bk\tilde{p} + (k-1)(k+1)(b+1)/12 = 0.$$

Using some calculus, and utilizing the information that $n \geqslant 5tk^2$, as per the claim of the lemma, and therefore $b \geqslant 5k$, as well as the knowledge of which k values are admissible in the present construction, we conclude that the ratio $\tilde{p}/b$ peaks at $(k, b) = (17, 85)$, so

$$\frac{\tilde{p}}{b} \leqslant \frac{1}{2} - \sqrt{\frac{343707}{368475}} \approx 0.017. \tag{1}$$

There are, therefore, many more pairs than are actually needed at this step. Let $\alpha = 1/2 - \sqrt{343707/368475}$.

The second potential pitfall for Algorithm 7 is that in step 47 we need to ensure that enough unused intervals are left for fine-tuning. This corresponds to the equation $b - 2p \geqslant 12$, because we require 2 intervals for the remaining rough-cancellation pair, 8 for the sum of squares, and a further 2 as the maximal amount of space wasted due to the sum of squares requiring two consecutive "choice 0" intervals each. A stricter condition is $b - 2\tilde{p} \geqslant 12$. From equation (1), we see that this condition is met for $b \geqslant 13$, so certainly also for $b \geqslant 5k$, because $k \geqslant 5$.

With this computed $\vec{c}$, we can now proceed with the equivalent of Algorithm 6. This remains essentially unchanged, but has the following modification. In the original Algorithm 6, moment cancellation was a two-step process: first rough-cancellation, then fine-tuning. We now add a third step, before the original two, meant to simplify handling of the variable t.

Recall that the algorithms of Section 2 deal with matrices divided into k-length intervals, whereas here we deal with matrices divided into kt-length intervals. It would be most convenient if we could treat the last bkt columns of matrix M as $[M' \times t]$. In fact, there are only 8 of the b intervals of the matrix where M diverges from an $[M' \times t]$ format, and we can simply avoid using them in moment cancellation, so they can safely be ignored. (These 8 intervals are the 2×4 used in the sum of squares.)

In Section 2 we already investigated how to correct the moment of a matrix like M' and have already established how much moment can be corrected in such a matrix. The problem is that any such action that has an effect of magnitude B on M' will have an effect of magnitude Bt^2 on M. This method can therefore only handle moments that are multiples of t^2. In M, however, the moment is general.

To solve this problem, we add 4 more intervals to the list of non-stretched intervals, for a total of 12. Again, we use Lagrange's four-square theorem to construct four numbers, $a_1, \ldots, a_4$, each at most t, such that the addition of $\sum_{i=1}^{4} a_i^2$ to the moment will make it divide by t^2. Once this is done, the original moment cancellation algorithm is applied on the M' matrices.

To perform the initial moment cancellation, we pick from each of the 4 chosen intervals one element from *pvals* and one element from *nvals*, each repeated t times and each known to exist in every interval, and make the a_i with the highest indices among the t repetitions of the value taken from *pvals* into 1 in $\vec{v}$, and the a_i with the lowest indices of the *nvals* into -1. Though this will only create a counterbalance of magnitude a_i^2 when the intervals are consecutive, it always creates a counterbalance that is congruent to a_i^2 in modulo t^2.

Finally, the last remaining obstacle is that the algorithm may fail in one of the two moment cancellation routines that were already introduced in Section 2. The calculation here is similar to the one used in the proof for Lemma 2.5, but we augment it because the moment may be larger and because the ability to counterbalance may be diminished (both due to the added features of $\vec{c}$).

We first revise our estimate regarding how many intervals are needed in order to ascertain that the fine-tuning step succeeds. As before, the case that requires the greatest number of intervals for fine-tuning is $pos = k - 1$, $neg = 1$, with the original moment being $2k - 1$. This requires $k - 2$ *neg*-pairs (each of which requires 2 intervals) and 3 *pos*-pairs (each of which requires a single interval), for a total of $2k - 1$ intervals. The difference, however, is that this time there is no partitioning of $\vec{c}$ into copies of a base $\vec{c'}$. Therefore, the total number of intervals that are unusable due to "choice" changes is bounded by 5. We describe explicitly the structure of $\vec{v}$ here.

The lowest indices of $\vec{v}$ correspond to "+1" choices and the highest indices are the reverse. Between these, there may be at most one other "−1" choice, from step 27 of Algorithm 7. In the lowest-most "choice 0" intervals, 4 pairs of intervals are used

for handling modulo k^2t^2 moment cancellation in $\vec{c}$. These are not used in moment cancellation on $\vec{v}$.

Other than these 4 pairs, any two consecutive "+1", "−1" or "0" choices can be used at this point for the fine tuning in the choice of *neg* pairs. The intervals that are unusable due to "choice" changes can be used either for the *pos* pairs or for the 4 intervals used for ensuring that the moment divides by t^2. Because there are no more than 5 intervals affected by "choice" changes, but $4+3$ (or more) interval roles that do not require pairing, no interval is truly unusable due to a "choice" change. We therefore reach the conclusion that if $(2k-1)+2\times 4+4 = 2k+11$ of the b complete kt-sized intervals are unused at the end of the rough moment-cancellation stage, these are enough to complete fine tuning properly.

We now turn to the question of how much needs to be counterbalanced at the rough-cancellation stage. The calculation repeats the one that was presented in the proof of Lemma 2.5, with the necessary changes and adding in the effects of the new elements. We summarize here briefly by enumerating the elements to be accounted for and their effect.

1. The $-(k-1)/2,\ldots,(k-1)/2$ progression in "choice 0" intervals accounts for $(k^2-1)t^2/4$ per interval. The $-(k-3)/2,\ldots,(k-3)/2$ progression in "choice 1" and "choice −1" intervals accounts for less. Together with the remaining types of complete intervals, these amount to no more than $b(k^2-1)t^2/4$.

2. "choice 1" and "choice −1" have pairs of $(-(k-1)/2,(k-1)/2)$ and $(-(k+1)/2,(k+1)/2)$ shared among them. Originally, (f,s) pairs were chosen in $\vec{c}$ so as to counterbalance at most $(k-1)k(k+1)(b+1)t^2/12$. Each such pair counterbalanced $(f-s)k^2t^2$. In the present context, each such pair causes at most $2(f-s+1)kt^2$ moment, so the total of all pairs is bounded by $(k-1)(k+1)(b+1)t^2/6+2pkt^2$, where p is the number of pairs, known to satisfy $p\leqslant \alpha b$.

3. The 4 pairs of interval used to form a sum of squares in creating $\vec{c}$ differ from regular intervals by the removal of at most t elements and their replacement by at most t others. This change is complemented by a change in the neighboring interval. Hence, this contributes at most $4kt^2$ to the moment, per pair, or $16kt^2$ in total. (The fact that the elements that have not been replaced may have shifted positions does not enter the calculation, because we have already taken into account the maximal moment attainable by any interval, as long as its elements are balanced.)

4. The first $n-bkt$ indices can be considered in two parts: first, we have $n \bmod t$ indices that contribute no more than $t^2/2$ to the moment. The rest of the at most $(k-1)t$ indices are balanced, and cannot contribute more moment than $(k-1)^2t^2/4$. In total, this is bounded by $k^2t^2/4$.

5. The cancellation actions of previous i iterations contribute twice their original moment. As a result, all of the above has to be multiplied by 3^{i-1}.

6. In adding the sum of squares that cancels the moment of $\vec{v}$ modulo t^2, each added pair can cause an inadvertent moment of at most kt^2, for a total of $4kt^2$. (This does

not accrue a 3^{i-1} factor, because it is part of the $\vec{v}$ moment cancellation process, not part of the initial moment of $\vec{c}$.)

In total, this sums up to

$$3^{i-1}t^2\left(b\left(\frac{5}{12}k^2+2\alpha k-\frac{5}{12}\right)+\frac{5}{12}k^2+16k-\frac{1}{6}\right)+4kt^2,$$

and to guarantee the success of the algorithm, this should be less than the moment that can be produced in the rough-moment-cancellation step, minus $2k+11$ unused intervals. The moment that can be generated in this way is bounded from below by

$$\frac{1}{2}(3^{i-1}+1)\left\lfloor\frac{1}{2}(b-(2k+11))\right\rfloor\left(\left\lceil\frac{1}{2}(b+(2k+11))\right\rceil-1\right)kt^2.$$

Substituting in $b\geqslant 5k$, we get that this is true if $k\geqslant 16$, which is true if $m\geqslant 3$. □

We remark that $b\geqslant 5k$ is not a tight bound. For a large enough m, the methods used here continue to work for any $b\geqslant\beta k$ when $\beta>\frac{5+\sqrt{61}}{3}\approx 4.27$.

4 Bounding $B(n)$ and $B'(n)$

We now turn to proving Theorem 2, of which our main theorem, Theorem 1, is a direct corollary due to $B'(n)\geqslant B(n)$.

Proof of Theorem 2. We construct, explicitly, a homogeneous $m\times n$ Münchhausen matrix for any n, with an m value that is $\log_3 n+\mathrm{O}(\log\log n)$.

Specifically, we construct a homogeneous $\tilde{m}\times n$ matrix that is $\mathrm{O}(1)$-part piecewise-monotone, with a $\tilde{m}$ value that is $\log_3 n+\mathrm{O}(\log\log n)$, then use Lemma 2.4 to make it into a Münchhausen matrix by adding $\mathrm{O}(1)$ weighings.

The homogeneous $\mathrm{O}(1)$-part piecewise-monotone weigh-set is, in turn, the union of s homogeneous weigh-sets, $\{M_i\}_{i=1}^s$, with $M_i\in\mathcal{M}(m_i,k_i,t_i,n)$, for all i, where the parameters m_i, k_i, t_i and s are determined as follows.

1. $\forall i, t_i=\prod_{j=1}^{i-1}k_j$.
2. $\forall i, m_i=\left\lfloor\log_3\left(\frac{3}{2}\sqrt{n/5t_i}\right)\right\rfloor$.
3. $\forall i$, k_i is chosen to be the largest prime smaller than $2\times 3^{m_i-1}$.
4. The parameter s is chosen to be maximal, subject to $\forall i, m_i\geqslant 4$.

The description above defines all parameters uniquely, as is demonstrated by Algorithm 8.

Furthermore, by Lemma 3.1, there exist weigh-sets M_i that match these specifications.

Algorithm 8 Choosing the parameters m_i, k_i, t_i and s

$s \leftarrow 0$
$t_1 \leftarrow 1$
$m_1 \leftarrow \left\lfloor \log_3 \left(\frac{3}{2}\sqrt{n/5} \right) \right\rfloor$
while $m_{s+1} \geqslant 4$ **do**
  $s \leftarrow s + 1$
  $k_s \leftarrow \max\{k \colon k \text{ prime and } k < 2 \times 3^{m_s - 1}\}$
  $t_{s+1} \leftarrow t_s k_s$
  $m_{s+1} \leftarrow \left\lfloor \log_3 \left(\frac{3}{2}\sqrt{n/5t_{s+1}} \right) \right\rfloor$
end while

By the same reasoning as in Lemma 2.3, the weigh-set constructed in this fashion is $\lceil n/t_s \rceil$-part piecewise-monotone, which, due to the criterion for choosing s, is $\mathrm{O}(1)$-part piecewise-monotone, as necessary. We have shown, therefore, the construction of a Münchhausen matrix with dimensions $m \times n$, for an arbitrary n, where m is $\mathrm{O}(1) + \sum_{i=1}^{s} m_i$. We now turn to proving that $m - \log_3 n$ is $\mathrm{O}(\log \log n)$.

As we have seen, the halting criterion ensures that n/t_s is $\mathrm{O}(1)$ and, as a result, k_s is also $\mathrm{O}(1)$ and $n/\prod_{i=1}^{s} k_i$ is $\mathrm{O}(1)$. The value $\log_3 n$ is therefore $\mathrm{O}(1) + \sum_{i=1}^{s} \log_3 k_i$. The value $m - \log_3 n$ can therefore be written as $\mathrm{O}(1) + \sum_{i=1}^{s} (m_i - \log_3 k_i)$.

A simple upper bound on $m_i - \log_3 k_i$ follows from Bertrand's postulate, which we have already used amply in the construction, according to which $k_i > 3^{m_i - 1}$, and therefore $m_i - \log_3 k_i < 1$. This ensures that $m - \log_3 n$ is $\mathrm{O}(s)$. We therefore turn to bounding s from above.

Consider the list of k_i values. By construction, k_i is greater than $1/3$ the largest power of 3 that is smaller than $\frac{3}{2}\sqrt{n/5t_i}$. Specifically, it is greater than $\sqrt{n/5t_i}/6$. On the other hand, k_{i+1} is smaller than $2/3$ the largest power of 3 that is smaller than $\frac{3}{2}\sqrt{n/5t_{i+1}}$. It is therefore smaller than $\sqrt{n/5t_{i+1}} = \sqrt{n/5t_i k_i}$, leading to the equation

$$k_{i+1} < \frac{6k_i}{\sqrt{k_i}} = 6\sqrt{k_i}. \tag{2}$$

Consider the sequence $\tilde{k}_i$, defined as follows, for a general value of n and for k_i and s values calculated from it: $\tilde{k}_i \stackrel{\text{def}}{=} \min_n\{k_{s+1-i}/36\}$. The sequence $\tilde{k}_i$, up to an order reversal and division by 36, gives a lower bound for the k_i for any n. In particular, because $m_s \geqslant 4$, we have $\tilde{k}_1 \geqslant \frac{53}{36} > 1$, and due to Equation (2) we have $\tilde{k}_{i+1} > \tilde{k}_i^2$. In particular, for all i, $\tilde{k}_i \geqslant \gamma^{2^i}$, for $\gamma = \sqrt{53/36}$.

Clearly, $k_1 \leqslant n$, so $\tilde{k}_s \leqslant n/36$, leading to the equation $n/36 \geqslant \gamma^{2^s}$, so s is $\mathrm{O}(\log \log n)$. □

We remark that throughout we have only used Bertrand's postulate, certifying the existence of a prime between a and $2a$. However, the largest prime smaller than $2a$, for a large enough $2a$, is known to be quite close to $2a$. Specifically, a result by Dusart

[4] is that for a large enough x a prime always exists between x and $\left(1 + (2\ln^2 x)^{-1}\right) x$. Integrating this tighter bound into our calculation of $m_i - \log_3 k_i$ it is possible to show that m is $\log_3 n + \delta \log_2 \log_B n + \mathrm{O}(1)$, where $\delta = \log_3 \frac{3}{2}$ and the choice of B is arbitrary, and only affects the $\mathrm{O}(1)$.

Though, mathematically speaking, $\delta \log_2 \log_B n$ is clearly an unbounded function of n, it is interesting to note that even when choosing $B = 2$, the value of this function only reaches 3 when $n > 10^{80}$, which is more than the estimated number of atoms in the universe [6]. It goes up to 4 at $n > 10^{551}$ and up to 5 at $n > 10^{3604}$. These numbers demonstrate that $B(n)$ is, indeed, very close to $\log_3 n$, as was conjectured in [11].

Acknowledgements

The author wishes to thank Ian Wanless for introducing him to the problem.

References

[1] Problems from the last round of LIV Moscow Mathematical Olympiad. *Kvant*, 9:70–71, 1991.

[2] M. Brand. Tightening the bounds on the Baron's omni-sequence. *Discrete Mathematics*, 312(7):1326–1335, 2012.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*, Section 8.1: Lower bounds for sorting, pages 165–168. MIT Press and McGrow-Hill, 2nd edition, 2001.

[4] Pierre Dusart. Inégalités explicites pour $\psi(X)$, $\theta(X)$, $\pi(X)$ et les nombres premiers. *C. R. Math. Acad. Sci. Soc. R. Can.*, 21(2):53–59, 1999.

[5] F. J. Dyson. 1931. the problem of the pennies. *The Mathematical Gazette*, 30(291):231–234, 1946.

[6] J. Gribbin. *In Search of the Big Bang: Quantum Physics and Cosmology*. Heinemann, 1986.

[7] L. Halbeisen and N. Hungerbühler. The general counterfeit coin problem. *Discrete Mathematics*, 147(1–3):139–150, 1995.

[8] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge, at the University Press, 1952. 2d ed.

[9] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory*, volume 84 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1990.

[10] T. Khovanova and J.B. Lewis. A186313. oeis.org/A186313. [Online; accessed 10 March 2012].

[11] T. Khovanova and J.B. Lewis. Baron Münchhausen redeems himself: Bounds for a coin-weighing puzzle. *The Electronic Journal of Combinatorics*, 18(1):37, 2011.

[12] Tanya Khovanova, Konstantin Knop, and Alexey Radul. Baron Münchhausen's sequence. *J. Integer Seq.*, 13(8):Article 10.8.7, 18, 2010.

[13] D.E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*, Section 5.3.1: Minimum-Comparison Sorting, pages 180–197. Addison-Wesley, 2nd edition, 1997.

[14] Alexander M. Mood. On Hotelling's weighing problem. *Ann. Math. Statistics*, 17:432–446, 1946.

[15] C.A.B. Smith. The counterfeit coin problem. *Mathematical Gazette*, 31:31–39, 1947.

[16] H. Steinhaus. *Mathematical Snapshots*. Dover Publications, 3rd edition, 1999.