ON IMMEASURABLE MAGNITUDES

BRUNO ZIKI KONGAWI

Abstract. By deducing and exploiting simple statements from Proposition 28 of Euclid VII, a crude (algorithmic) method for finding prime numbers is devised. Additionally, in order to be as self-contained as possible, rough (yet fully compilable) programs written in the modern language Kotlin—originally used to expedite the writing and verification—are provided at the end to amply substantiate correctness and reproducibility (under certain suitable limits).

1. Introduction

This small, somewhat informal paper demonstrates a curious, counterintuitive method—based on easily deducible statements—for finding prime numbers (less than a certain magnitude). The sequence of ideas and technique propounded here (together with the preferred Euclidean terminology) represent to some degree a glimpse of some subtle, yet illuminating insights originally obtained while leisurely contemplating and experimenting with certain seemingly disparate ideas in metaphysics, physics, mathematics and philosophy (virtually indiscriminately).[1]

What follows is perhaps most worthy of the attention and appreciation of mathematicians engaged in the persistent pursuit of an elucidation of the fundamental laws governing (the distribution of) prime numbers insofar as it may offer a lateral, more fecund perspective—or vantage point—from which a relatively more intimate understanding of their mystifying nature (as it relates to their apparently "random" and obstinately unpredictable occurrence) can perhaps be attained or lucidly formulated in simpler terms.

Supposed hereinafter—alongside an exposure to very elementary notions of modern mathematics—is a minimal acquaintance with Euclid's *Elements* (particularly Euclid VII [1]) and the terminology contained therein.

It is worth remarking that definitions 1 and 2 of Euclid V are essentially definitions 3 and 5 of Euclid VII [1], respectively. If "magnitude" and "number" are generally assumed to mean the same in the sense of being treated as any extended (*i.e.* non-degenerate) entity, it follows that propositions and arguments based on either are (in principle) true for both. The terms *magnitude* and *number* shall be applied to both almost interchangeably and synonymously to preserve some fidelity to the origins and inspirations of the discovery and the manner in which it arose.

Date: January 3, 2020.

[1]An ulterior, more elaborate philosophical publication shall hopefully revisit some of those inspiring ideas, their prime motivations, and other conclusions one is naturally lead to deduce.

2. Primes as immeasurables

It is first observed that "*if two numbers be prime to one another, the sum will also be prime to each of them.*"[2]

Let p_1 and p_2 be two (distinct) magnitudes prime to one another[3] in the sense of being measured by one unit[4] alone. Let that unit be 1. Then it follows that the sum $p_1 + p_2$ will be prime to each of the original magnitudes, *i.e.* the magnitudes p_1, p_2 and $p_1 + p_2$ will be measured or "unified" by one (common) unit alone (that is 1). Euphemistically, it can be said that each will not measure the others, or simply, for convenience, that they are *relatively immeasurable.*[5]

Plainly, by virtue of the same statement borrowed from Euclid VII—*viz.* Proposition 28 [1]—it also follows that all magnitudes or numbers of the (varying) forms

$$p_1 \times p_1 + p_2$$
$$p_1 \times p_1 \times \cdots \times p_1 + p_2$$
$$p_1 + p_2 \times p_2$$
$$p_1 + p_2 \times p_2 \times \cdots \times p_2$$
$$p_1 \times p_1 + p_2 \times p_2$$
$$p_1 \times p_1 \times \cdots \times p_1 + p_2 \times p_2 \times \cdots \times p_2$$
$$etc.$$

will be relatively immeasurable or prime (*i.e* relatively to p_1 and p_2), since the relative primality or "immeasurability" between the two (successively multiplied) magnitudes in each sum is preserved. Thus, however many times either magnitude is repeatedly multiplied to itself, so long as the products or magnitudes (in the sum) remain prime to one another, magnitudes of the form $p_1^{s_1} + p_2^{s_2}$ will be relatively immeasurable or prime to p_1 and p_2 where s_1 and s_2 designate whole (natural) numbers.[6]

Generalizing, Proposition 28 of Euclid VII [1] can easily be extended to apply to more than two "seed"[7] (relatively) immeasurable magnitudes. Let p_1, p_2, and p_3 be (distinct) relatively immeasurable magnitudes, where each is prime (or immeasurable) to every other. Then, since the magnitudes p_1 and p_2 are prime to the magnitude p_3 (as is any to every other), it follows that the product (or composite) $p_1 \times p_2$ will be prime to the magnitude p_3 (because the composite $p_1 \times p_2$ and the magnitude p_3 will share "*one unit alone as a common measure*" [1]). Similarly, the product $p_1 \times p_3$ will be prime to the magnitude p_2 in the same way the product $p_2 \times p_3$ will be to the magnitude p_1. Therefore, the following will also be

---

[2]See Proposition 28 of Euclid VII [1].

[3]In Euclidean terminology, p_1 and p_2 "*are measured by an unit alone as a common measure*" (See Definition 12 of Euclid VII [1]).

[4]As defined by Euclid, an "*unit is that by virtue of which each of the things that exist is called one*" (see Definition 1 of Euclid VII [1]). Henceforth, when not stated, the unit is tacitly assumed to be 1.

[5]Although slightly deviating from Euclid's definition, this liberal way of describing magnitudes prime to one another shall be adopted henceforth. In modern terminology, however, they can be said to be *coprime* or *relatively prime.*

[6]See also Proposition 25 and Proposition 27 of Euclid VII [1].

[7]The use of the word "seed" here should not be misconstrued as to signify that the given magnitudes are parts of the ones one obtains from the numerical calculations.

immeasurable magnitudes (*i.e.* relatively to p_1, p_2 and p_3)

$$p_1 \times p_2 + p_3$$
$$p_1 \times p_3 + p_2$$
$$p_3 \times p_2 + p_1$$

or, if one substitutes p_1+p_2 for p_3 as obtained earlier (where $s_1 = s_2 = 1$), effectively using the result of a previous iteration or computation,

$$p_1 \times p_2 + p_1 + p_2$$
$$p_1 \times (p_1 + p_2) + p_2 = p_1^2 + p_1 \times p_2 + p_2$$
$$(p_1 + p_2) \times p_2 + p_1 = p_1 \times p_2 + p_2^2 + p_1$$

Exploiting this binary splitting technique to form (immeasurable or prime) sums, which fundamentally hinges on the (required) relative primality of any two magnitudes (as in Proposition 28 of Euclid VII [1]), one may wish to carry this generalization a bit further. Increasing the number of distinct "seed" (relatively) immeasurable magnitudes, an ever-growing number of (relatively) immeasurable magnitudes can be obtained by (1) exhaustively listing every possible way the given set can be partitioned or split in two disjoint subsets (such that the union of the two is the original set of "seed" immeasurable magnitudes); (2) successively multiplying all magnitudes in each subset (thus composing two magnitudes prime to one another from each partition, which is required to form a prime sum); and then finally (3) adding the two products of each partition to find a magnitude that cannot be measured by the starting set (or any of the "seed " immeasurable magnitudes).

This sequence of operations can then be continually applied to an (infinitely) growing number of (immeasurable) magnitudes by, for example, repeatedly adjoining any of the newly obtained magnitudes to the previous "seed" magnitudes (then taken collectively as the new "seed" set of the next iteration).

For instance, given five "seed" (relatively) immeasurable magnitudes p_1, p_2, p_3, p_4 and p_5, all possible binary partitions will be

$$\{ p_1 \} \textit{ and } \{ p_2, p_3, p_4, p_5 \}$$
$$\{ p_1, p_2 \} \textit{ and } \{ p_3, p_4, p_5 \}$$
$$\{ p_1, p_3 \} \textit{ and } \{ p_2, p_4, p_5 \}$$
$$\{ p_1, p_4 \} \textit{ and } \{ p_2, p_3, p_5 \}$$
$$\{ p_1, p_5 \} \textit{ and } \{ p_2, p_3, p_4 \}$$
$$\{ p_2 \} \textit{ and } \{ p_1, p_3, p_4, p_5 \}$$
$$\{ p_2, p_3 \} \textit{ and } \{ p_1, p_4, p_5 \}$$
$$\{ p_2, p_4 \} \textit{ and } \{ p_1, p_3, p_5 \}$$
$$\{ p_2, p_5 \} \textit{ and } \{ p_1, p_3, p_4 \}$$
$$\{ p_3 \} \textit{ and } \{ p_1, p_2, p_4, p_5 \}$$
$$\{ p_3, p_4 \} \textit{ and } \{ p_1, p_2, p_5 \}$$
$$\{ p_3, p_5 \} \textit{ and } \{ p_1, p_2, p_4 \}$$
$$\{ p_4 \} \textit{ and } \{ p_1, p_2, p_3, p_5 \}$$
$$\{ p_4, p_5 \} \textit{ and } \{ p_1, p_2, p_3 \}$$

$$\{\, p_5 \,\}\ and\ \{\, p_1,\, p_2,\, p_3,\, p_4 \,\}$$

Thus, the following will constitute a set of magnitudes that cannot be measured by any of the "seed" immeasurable magnitudes (p_1, p_2, p_3, p_4 and p_5):

$$\begin{gathered}
p_1 + p_2 \times p_3 \times p_4 \times p_5 \\
p_1 \times p_2 + p_3 \times p_4 \times p_5 \\
p_1 \times p_3 + p_2 \times p_4 \times p_5 \\
p_1 \times p_4 + p_2 \times p_3 \times p_5 \\
p_1 \times p_5 + p_2 \times p_3 \times p_4 \\
p_2 + p_1 \times p_3 \times p_4 \times p_5 \\
p_2 \times p_3 + p_1 \times p_4 \times p_5 \\
p_2 \times p_4 + p_1 \times p_3 \times p_5 \\
p_2 \times p_5 + p_1 \times p_3 \times p_4 \\
p_3 + p_1 \times p_2 \times p_4 \times p_5 \\
p_3 \times p_4 + p_1 \times p_2 \times p_5 \\
p_3 \times p_5 + p_1 \times p_2 \times p_4 \\
p_4 + p_1 \times p_2 \times p_3 \times p_5 \\
p_4 \times p_5 + p_1 \times p_2 \times p_3 \\
p_5 + p_1 \times p_2 \times p_3 \times p_4
\end{gathered}$$

If then one considers any of the results, say the magnitude $p_1 \times p_3 + p_2 \times p_4 \times p_5$ (the third from the list), as a "branch" from which more immeasurable magnitudes can be obtained or computed, then the sequence of operations outlined above can be applied again as follows; the magnitude $p_1 \times p_3 + p_2 \times p_4 \times p_5$ is united to the "seed" set from which it was obtained to compose the new "seed" set for the next iteration: $\{\, p_1,\, p_2,\, p_3,\, p_4,\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \}$; the list of all possible (unique) binary partitions is then generated as (non-exhaustive list):

$$\begin{gathered}
\{\, p_1 \,\}\ and\ \{\, p_2,\, p_3,\, p_4,\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\} \\
\{\, p_1,\, p_2 \,\}\ and\ \{\, p_3,\, p_4,\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\} \\
\{\, p_1,\, p_2,\, p_3 \,\}\ and\ \{\, p_4,\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\} \\
\{\, p_1,\, p_2,\, p_4 \,\}\ and\ \{\, p_3,\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\} \\
\{\, p_1,\, p_2,\, p_5 \,\}\ and\ \{\, p_3,\, p_4,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\} \\
\vdots \\
\{\, p_4,\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\}\ and\ \{\, p_1,\, p_2,\, p_3 \,\} \\
\{\, p_4,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\}\ and\ \{\, p_1,\, p_2,\, p_3,\, p_5 \,\} \\
\{\, p_5 \,\}\ and\ \{\, p_1,\, p_2,\, p_3,\, p_4,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\} \\
\{\, p_5,\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\}\ and\ \{\, p_1,\, p_2,\, p_3,\, p_4 \,\} \\
\{\, p_1 \times p_3 + p_2 \times p_4 \times p_5 \,\}\ and\ \{\, p_1,\, p_2,\, p_3,\, p_4,\, p_5 \,\}
\end{gathered}$$

Finally, for each (unique) partition or pair of subsets, the (successive) product of all elements of one subset is added to that of the other to obtain a set of magnitudes prime to the original immeasurable magnitudes (which now includes $p_1 \times p_3 + p_2 \times$

$p_4 \times p_5$):

$$p_1 + p_2 \times p_3 \times p_4 \times p_5 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5)$$
$$p_1 \times p_2 + p_3 \times p_4 \times p_5 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5)$$
$$p_1 \times p_2 \times p_3 + p_4 \times p_5 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5)$$
$$p_1 \times p_2 \times p_4 + p_3 \times p_5 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5)$$
$$p_1 \times p_2 \times p_5 + p_3 \times p_4 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5)$$
$$\vdots$$
$$p_4 \times p_5 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5) + p_1 \times p_2 \times p_3$$
$$p_4 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5) + p_1 \times p_2 \times p_3 \times p_5$$
$$p_5 + p_1 \times p_2 \times p_3 \times p_4 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5)$$
$$p_5 \times (p_1 \times p_3 + p_2 \times p_4 \times p_5) + p_1 \times p_2 \times p_3 \times p_4$$
$$p_1 \times p_3 + p_2 \times p_4 \times p_5 + p_1 \times p_2 \times p_3 \times p_4 \times p_5$$

or

$$p_1 + p_1 \times p_2 \times p_3^2 \times p_4 \times p_5 + p_2^2 \times p_3 \times p_4^2 \times p_5^2$$
$$p_1 \times p_2 + p_1 \times p_3^2 \times p_4 \times p_5 + p_2 \times p_3 \times p_4^2 \times p_5^2$$
$$p_1 \times p_2 \times p_3 + p_1 \times p_3 \times p_4 \times p_5 + p_2 \times p_4^2 \times p_5^2$$
$$p_1 \times p_2 \times p_4 + p_1 \times p_3^2 \times p_5 + p_2 \times p_3 \times p_4 \times p_5^2$$
$$p_1 \times p_2 \times p_5 + p_1 \times p_3^2 \times p_4 + p_2 \times p_3 \times p_4^2 \times p_5$$
$$\vdots$$
$$p_1 \times p_3 \times p_4 \times p_5 + p_2 \times p_4^2 \times p_5^2 + p_1 \times p_2 \times p_3$$
$$p_1 \times p_3 \times p_4 + p_2 \times p_4^2 \times p_5 + p_1 \times p_2 \times p_3 \times p_5$$
$$p_5 + p_1^2 \times p_2 \times p_3^2 \times p_4 + p_1 \times p_2^2 \times p_3 \times p_4^2 \times p_5$$
$$p_1 \times p_3 \times p_5 + p_2 \times p_4 \times p_5^2 + p_1 \times p_2 \times p_3 \times p_4$$
$$p_1 \times p_3 + p_2 \times p_4 \times p_5 + p_1 \times p_2 \times p_3 \times p_4 \times p_5$$

Thus, it can be said that given n unique magnitudes, $p_1, \cdots, p_n$, "*which are measured by a unit alone as a common measure*" [1], then all magnitudes of the form

$$p_1 \times p_2 \times \cdots \times p_{m-1} \times p_m + p_{m+1} \times p_{m+2} \times \cdots \times p_{n-1} \times p_n$$

or, more compactly,

$$\prod_{i=1}^{m} p_i + \prod_{i=m+1}^{n} p_i$$

are immeasurable by (or prime to) the original immeasurable magnitudes where n is at least 2 and m is necessarily a whole number less than n.

It must be noted that the order of the set $\{p_i\}$ in the expression $\prod_{i=1}^{m} p_i + \prod_{i=m+1}^{n} p_i$ (as it relates to the index i) are not necessarily (one-to-one) correspondent to that of their listing as in $p_1, \cdots, p_n$. In other words, given any set of unique

immeasurable magnitudes, one should only be concerned with all the ways the set can be partitioned in two to form $\prod_{i=1}^{m} p_i + \prod_{i=m+1}^{n} p_i$.

Moreover, since any of the magnitudes raised to any whole power (greater than zero) in its respective (successive) product will preserve the relative primality (or "immeasurability") of the two products for each partition (namely $p_1 \times p_2 \times \cdots \times p_{m-1} \times p_m$ and $p_{m+1} \times p_{m+2} \times \cdots \times p_{n-1} \times p_n$), all magnitudes of the form

$$p_1^{s_{1\cdot j}} \times p_2^{s_{2\cdot j}} \times \cdots \times p_{m-1}^{s_{(m-1)\cdot j}} \times p_m^{s_{m\cdot j}} + p_{m+1}^{s_{(m+1)\cdot j}} \times p_{m+2}^{s_{(m+2)\cdot j}} \times \cdots \times p_{n-1}^{s_{(n-1)\cdot j}} \times p_n^{s_{n\cdot j}}$$

or

$$\prod_{i=1}^{m} p_i^{s_{i\cdot j}} + \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}}$$

are also immeasurable by (or prime to) the original immeasurable magnitudes where $s_{1\cdot j}, \cdots, s_{n\cdot j}$ (or all elements of the set of exponents $\{s_{i\cdot j}\}$) are whole numbers and n is at least 2 while m is less than n as before.

It must be pointed out that the set of exponents $\{s_{i\cdot j}\}$ (in the expression $\prod_{i=1}^{m} p_i^{s_{i\cdot j}} + \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}}$) is the infinite (repeating) set of whole numbers, where i indicates the "index" associated with an immeasurable magnitude and j denotes that of *any* whole number. The expression "$i \cdot j$" then merely tells that for every i there are j (which tends to infinity: $j \to \mathbb{N}_\infty$); thus, for any immeasurable magnitude in any (sum) expression formed from any binary partition (as formulated above), any whole number can be applied as an exponent. The table below loosely (although imperfectly) serves to illustrate this, where each row represents all (infinite) possible repeated multiplications of an immeasurable magnitude, or every possible way each immeasurable magnitude can be raised to a whole power.[8]

$i \cdot j$	1	2	…	$\mathbb{N}_\infty$
p_1	$p_1^{s_{1\cdot 1}}$	$p_1^{s_{1\cdot 2}}$	…	$p_1^{s_{1\cdot \mathbb{N}_\infty}}$
p_2	$p_2^{s_{2\cdot 1}}$	$p_2^{s_{2\cdot 2}}$	…	$p_2^{s_{2\cdot \mathbb{N}_\infty}}$
p_3	$p_3^{s_{3\cdot 1}}$	$p_3^{s_{3\cdot 2}}$	…	$p_3^{s_{3\cdot \mathbb{N}_\infty}}$
⋮	⋮	⋮	⋱	⋮
p_n	$p_n^{s_{n\cdot 1}}$	$p_n^{s_{n\cdot 2}}$	…	$p_n^{s_{n\cdot \mathbb{N}_\infty}}$

⇓

$i \cdot j$	1	2	…	$\mathbb{N}_\infty$
p_1	$p_1^{s_1}$	$p_1^{s_2}$	…	$p_1^{s_{\mathbb{N}_\infty}}$
p_2	$p_2^{s_1}$	$p_2^{s_2}$	…	$p_2^{s_{\mathbb{N}_\infty}}$
p_3	$p_3^{s_1}$	$p_3^{s_2}$	…	$p_3^{s_{\mathbb{N}_\infty}}$
⋮	⋮	⋮	⋱	⋮
p_n	$p_n^{s_1}$	$p_n^{s_2}$	…	$p_n^{s_{\mathbb{N}_\infty}}$

⇓

[8]The infinity symbol "∞" in expressions such as "$1\cdot\mathbb{N}_\infty$" simply signifies that for the immeasurable magnitude of index 1, there are infinite ways it can be raised to a whole power; thus "$p_1^{\mathbb{N}_\infty}$" does not mean "p_1 raised to the power of infinity."

$i \cdot j$	1	2	...	$\mathbb{N}_\infty$
p_1	p_1^1	p_1^2	$\cdots$	$p_1^{\mathbb{N}_\infty}$
p_2	p_2^1	p_2^2	$\cdots$	$p_2^{\mathbb{N}_\infty}$
p_3	p_3^1	p_3^2	$\cdots$	$p_3^{\mathbb{N}_\infty}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
p_n	p_n^1	p_n^2	$\cdots$	$p_n^{\mathbb{N}_\infty}$

Proposition 28 hints that the relative primality (or "immeasurability") expressed as a difference may also hold true. Thus, to sum up briefly at the expense of some rigor, it is reasonable to (tentatively) infer that all magnitudes of the form

$$\overbrace{\underbrace{p_1^{s_{1\cdot j}} \times p_2^{s_{2\cdot j}} \times \cdots \times p_{m-1}^{s_{(m-1)\cdot j}} \times p_m^{s_{m\cdot j}}}_{p_a \,=\, \text{immeasurable by } p_b} \pm \underbrace{p_{m+1}^{s_{(m+1)\cdot j}} \times p_{m+2}^{s_{(m+2)\cdot j}} \times \cdots \times p_{n-1}^{s_{(n-1)\cdot j}} \times p_n^{s_{n\cdot j}}}_{p_b \,=\, \text{immeasurable by } p_a}}^{\text{immeasurable magnitude (relatively to the whole set)}}$$

or

$$\prod_{i=1}^{m} p_i^{s_{i\cdot j}} \pm \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}}$$

$$\left(\textit{where } \prod_{i=1}^{m} p_i^{s_{i\cdot j}} = p_a \textit{ and } \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}} = p_b\right)$$

are relatively immeasurable where $s_{1\cdot j}, \cdots, s_{n\cdot j}$ are whole numbers, n is at least 2 and m is less than n.[9]

Simple and innocuous as it is, the foregoing expression can be utilized in conjunction with the described steps to devise a crude algorithm or recipe by means of which prime numbers can confidently be found and laws governing their occurrence can perhaps be satisfactorily deciphered.

3. A simple demonstration

For a small, simple demonstration it suffices to initially think of only two smallest (whole) magnitudes or numbers which do not share any common measure (other than the unit, of course): 1 and 2.

Without loss of generality, and for the sake of simplicity and pragmatism, let $s_{1\cdot j}, \cdots, s_{n\cdot j}$, or all elements of the set of exponents $\{s_{i\cdot j}\}$, be 1 (*i.e.* only the first column in the table—the set $\{p_i^{s_{i\cdot 1}}\}$—is considered) and all negative magnitudes of the form

$$\prod_{i=1}^{m} p_i^{s_{i\cdot j}} - \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}}$$

be reduced to its positive (absolute) value

$$\left| \prod_{i=1}^{m} p_i^{s_{i\cdot j}} - \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}} \right|$$

[9]A complete proof of the subtraction case is not provided here, as correctness is deferred to the demonstration, which of course should not be regarded as a definite rigorous proof.

One must now seek the set of magnitudes that $\{1,\ 2\}$[10] (or, more accurately, every element thereof except the unit) does not measure. For the first iteration I_1, there is only one way the set $\{1,\ 2\}$ can be expressed as two sets:

$$\{\,1\,\}\ and\ \{\,2\,\}$$

Therefore, one gets two immeasurable magnitudes from the addition and subtraction of the (respective) successive products of the proper subsets (stemming from the binary partition of the set):

$$\begin{gathered} 1^{s_{1\cdot j}} + 2^{s_{2\cdot j}}\ and\ \left|1^{s_{1\cdot j}} - 2^{s_{2\cdot j}}\right| \\ \Downarrow \\ 1^1 + 2^1 = 3\ and\ \left|1^1 - 2^1\right| = 1 \\ (when\ s_{1\cdot j} = s_{2\cdot j} = 1) \end{gathered}$$

Let $\mathbb{P}$ be the set of all (distinct) immeasurable magnitudes found or known so far at any given iteration; then $\mathbb{P} = \{1,\ 2,\ 3\}$.

For the second iteration I_2, one now seeks magnitudes that the set $\{1,\ 2,\ 3\}$ (or every element thereof except the unit) does not measure. The list of all unique binary partitions of the set $\mathbb{P}$ are

$$\begin{gathered} \{\,1\,\}\ and\ \{\,2,\,3\,\} \\ \{\,2\,\}\ and\ \{\,1,\,3\,\} \\ \{\,3\,\}\ and\ \{\,1,\,2\,\} \end{gathered}$$

from which the following immeasurable magnitudes are obtained:

$$\begin{gathered} 1^{s_{1\cdot j}} + 2^{s_{2\cdot j}} \times 3^{s_{3\cdot j}}\ and\ \left|1^{s_{1\cdot j}} - 2^{s_{2\cdot j}} \times 3^{s_{3\cdot j}}\right| \\ 2^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 3^{s_{3\cdot j}}\ and\ \left|2^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 3^{s_{3\cdot j}}\right| \\ 3^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}}\ and\ \left|3^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}}\right| \\ \Downarrow \\ 1^1 + 2^1 \times 3^1 = 7\ and\ \left|1^1 - 2^1 \times 3^1\right| = 5 \\ 2^1 + 1^1 \times 3^1 = 5\ and\ \left|2^1 - 1^1 \times 3^1\right| = 1 \\ 3^1 + 1^1 \times 2^1 = 5\ and\ \left|3^1 - 1^1 \times 2^1\right| = 1 \\ (when\ s_{1\cdot j} = s_{2\cdot j} = s_{3\cdot j} = 1) \end{gathered}$$

Thus $\mathbb{P} = \{1,\ 2,\ 3,\ 5,\ 7\}$. For the third iteration I_3, however, one must take into account that there is more than one new magnitude (*i.e.* "branch") from I_2: $\{5,\ 7\}$; and therefore, there is more than one way to decide on the composition of the set of the next iteration.

One approach to assemble the set of the next iteration would be to use the entirety of $\mathbb{P}$ as before. Instead, however, in order to maintain a progressively increasing order and "complete continuity" (in the sense of not having gaps), let the smallest magnitude from the set $\mathbb{P} \setminus \{the\ seed\ set\ of\ the\ previous\ iteration\}$ be adjoined to $\{the\ seed\ set\ of\ the\ previous\ iteration\}$ (from I_2), *i.e.* let the smallest from the set $\mathbb{P} \setminus \{1,\ 2,\ 3\,\} = \{5,\ 7\}$ be adjoined to the previous set $\{1,\ 2,\ 3\,\}$. Then,

[10]Evidently, one can argue that 1, being the unit that measures all, should not be included.

the set for the third iteration I_3 is $\{1, 2, 3, 5\}$, and the list of its (unique) binary partitions is:

$$\{1\}\ and\ \{2, 3, 5\}$$
$$\{1, 2\}\ and\ \{3, 5\}$$
$$\{1, 3\}\ and\ \{2, 5\}$$
$$\{1, 5\}\ and\ \{2, 3\}$$
$$\{2\}\ and\ \{1, 3, 5\}$$
$$\{3\}\ and\ \{1, 2, 5\}$$
$$\{5\}\ and\ \{1, 2, 3\}$$

from which the next collection of immeasurable magnitudes is calculated as

$$1^{s_{1\cdot j}} + 2^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\ and\ \left|1^{s_{1\cdot j}} - 2^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\right|$$
$$1^{s_{1\cdot j}} \times 2^{s_{2\cdot j}} + 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\ and\ \left|1^{s_{1\cdot j}} \times 2^{s_{2\cdot j}} - 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\right|$$
$$1^{s_{1\cdot j}} \times 3^{s_{2\cdot j}} + 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\ and\ \left|1^{s_{1\cdot j}} \times 3^{s_{2\cdot j}} - 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\right|$$
$$1^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} + 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}}\ and\ \left|1^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} - 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}}\right|$$
$$2^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\ and\ \left|2^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\right|$$
$$3^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\ and\ \left|3^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}}\right|$$
$$5^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}}\ and\ \left|5^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}}\right|$$
$$\Downarrow$$
$$1^1 + 2^1 \times 3^1 \times 5^1 = 31\, and\ \left|1^1 - 2^1 \times 3^1 \times 5^1\right| = 29$$
$$1^1 \times 2^1 + 3^1 \times 5^1 = 17\, and\ \left|1^1 \times 2^1 - 3^1 \times 5^1\right| = 13$$
$$1^1 \times 3^1 + 2^1 \times 5^1 = 13\, and\ \left|1^1 \times 3^1 - 2^1 \times 5^1\right| = 7$$
$$1^1 \times 5^1 + 2^1 \times 3^1 = 11\, and\ \left|1^1 \times 5^1 - 2^1 \times 3^1\right| = 1$$
$$2^1 + 1^1 \times 3^1 \times 5^1 = 17\, and\ \left|2^1 - 1^1 \times 3^1 \times 5^1\right| = 13$$
$$3^1 + 1^1 \times 2^1 \times 5^1 = 13\, and\ \left|3^1 - 1^1 \times 2^1 \times 5^1\right| = 7$$
$$5^1 + 1^1 \times 2^1 \times 3^1 = 11\, and\ \left|5^1 - 1^1 \times 2^1 \times 3^1\right| = 1$$
$$(when\ s_{1\cdot j} = s_{2\cdot j} = s_{3\cdot j} = s_{4\cdot j} = 1)$$

Thus $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, 17, 29, 31\}$. The set to use for the next iteration I_4 is $\{1, 2, 3, 5, 7\}$, which is the smallest of the set $\mathbb{P} \setminus \{1, 2, 3, 5\} = \{7, 11, 13, 17, 29, 31\}$ united with the previous set $\{1, 2, 3, 5\}$.

Then the list of binary partitions (for the iteration I_4) is

$$\{1\}\ and\ \{2, 3, 5, 7\}$$
$$\{1, 2\}\ and\ \{3, 5, 7\}$$
$$\{1, 3\}\ and\ \{2, 5, 7\}$$
$$\{1, 5\}\ and\ \{2, 3, 7\}$$
$$\{1, 7\}\ and\ \{2, 3, 5\}$$
$$\{2\}\ and\ \{1, 3, 5, 7\}$$
$$\{2, 3\}\ and\ \{1, 5, 7\}$$
$$\{2, 5\}\ and\ \{1, 3, 7\}$$

$$\{\,2,\,7\,\}\ and\ \{\,1,\,3,\,5\,\}$$
$$\{\,3\,\}\ and\ \{\,1,\,2,\,5,\,7\,\}$$
$$\{\,3,\,5\,\}\ and\ \{\,1,\,2,\,7\,\}$$
$$\{\,3,\,7\,\}\ and\ \{\,1,\,2,\,5\,\}$$
$$\{\,5\,\}\ and\ \{\,1,\,2,\,3,\,7\,\}$$
$$\{\,5,\,7\,\}\ and\ \{\,1,\,2,\,3\,\}$$
$$\{\,7\,\}\ and\ \{\,1,\,2,\,3,\,5\,\}$$

from which one obtains

$$1^{s_{1\cdot j}} + 2^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |1^{s_{1\cdot j}} - 2^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$1^{s_{1\cdot j}} \times 2^{s_{2\cdot j}} + 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |1^{s_{1\cdot j}} \times 2^{s_{2\cdot j}} - 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$1^{s_{1\cdot j}} \times 3^{s_{2\cdot j}} + 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |1^{s_{1\cdot j}} \times 3^{s_{2\cdot j}} - 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$1^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} + 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |1^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} - 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$1^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} + 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}\ and\ |1^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} - 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}|$$
$$2^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |2^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 3^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$2^{s_{1\cdot j}} \times 3^{s_{2\cdot j}} + 1^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |2^{s_{1\cdot j}} \times 3^{s_{2\cdot j}} - 1^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$2^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} + 1^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |2^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} - 1^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$2^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} + 1^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}\ and\ |2^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} - 1^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}|$$
$$3^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |3^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 5^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$3^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} + 1^{s_{3\cdot j}} \times 2^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |3^{s_{1\cdot j}} \times 5^{s_{2\cdot j}} - 1^{s_{3\cdot j}} \times 2^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$3^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} + 1^{s_{3\cdot j}} \times 2^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}\ and\ |3^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} - 1^{s_{3\cdot j}} \times 2^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}|$$
$$5^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}\ and\ |5^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 7^{s_{5\cdot j}}|$$
$$5^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} + 1^{s_{3\cdot j}} \times 2^{s_{4\cdot j}} \times 3^{s_{5\cdot j}}\ and\ |5^{s_{1\cdot j}} \times 7^{s_{2\cdot j}} - 1^{s_{3\cdot j}} \times 2^{s_{4\cdot j}} \times 3^{s_{5\cdot j}}|$$
$$7^{s_{1\cdot j}} + 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}\ and\ |7^{s_{1\cdot j}} - 1^{s_{2\cdot j}} \times 2^{s_{3\cdot j}} \times 3^{s_{4\cdot j}} \times 5^{s_{5\cdot j}}|$$

$$\Downarrow$$

$$211\ and\ 209$$
$$107\ and\ 103$$
$$73\ and\ 67$$
$$47\ and\ 37$$
$$37\ and\ 23$$
$$107\ and\ 103$$
$$41\ and\ 29$$
$$31\ and\ 11$$
$$29\ and\ 1$$
$$73\ and\ 67$$
$$29\ and\ 1$$
$$31\ and\ 11$$
$$47\ and\ 37$$
$$41\ and\ 29$$

$$37\,and\,23$$
$$(when\,s_{1\cdot j} = s_{2\cdot j} = \cdots = s_{5\cdot j} = 1)$$

Thus $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, 41, 47, 67, 73, 103, 107, 209, 211\}$.

An intriguing observation should almost immediately be apparent at this point: Excluding the unit, virtually all numbers somehow or other are prime with only one exception; decomposing 209 reveals that it can be expressed as 11×19. A little examination of this fact with knowledge of what can be stated with confidence about the elements of the set $\mathbb{P}$ (or the computed immeasurable magnitudes) in relation to the sets from which they were obtained provides an inkling as to the reason for the presence of a composite.

It can only be asserted that the found magnitudes are immeasurable (indivisible) by the set from which they were obtained. In other words, it is still possible to find *composite* magnitudes that are measurable (divisible) by other magnitudes (*i.e.* magnitudes that are not present in the "seed" set). Indeed, it is not required that the "seed" magnitudes be prime at all, but only that they be prime to *one another* (*i.e.* relatively immeasurable).

Notwithstanding, if one were curious enough in view of the observation, the exploration could readily be reduced to seeking auxiliary conditions (relatively to the current set or iteration) that must yield, with absolute certainty, prime numbers.

One could (1) investigate patterns, conditions, or properties in the combined exponents $s_{1\cdot j}, \cdots, s_{n\cdot j}$ perhaps in relation to their arrangements with respect to the immeasurable magnitudes in the sum (or difference) expression for which the calculated magnitudes are prime; or (2) resort to a more trivial criterion, which assumes the existence of a certain critical upper bound below which all calculated immeasurable magnitudes are prime (for every iteration or "seed" set) if the unit is ignored.[11]

Assuming that one can be certain that a given set of (relatively) immeasurable magnitudes contains all prime numbers up to its greatest element (*i.e.* all immeasurable magnitudes except the unit are the complete continuum of primes up to the greatest in the set), then there must exist a magnitude C below which all immeasurable magnitudes calculated from the set are prime.

Suppose p_g is the greatest magnitude (or prime number) of the given set of relatively immeasurable magnitudes forming a complete "prime continuum." Then the greatest composite magnitude made up of at most two magnitudes from the set is $p_g \times p_g$. Let $C = p_g \times p_g = {p_g}^2$.

Then, given a set of n unique (relatively) immeasurable magnitudes, $p_1, \cdots, p_n$, which contains all prime numbers up to its greatest magnitude p_g, all magnitudes of the form

$$\prod_{i=1}^{m} {p_i}^{s_{i\cdot j}} + \prod_{i=m+1}^{n} {p_i}^{s_{i\cdot j}}$$

and

$$\left| \prod_{i=1}^{m} {p_i}^{s_{i\cdot j}} - \prod_{i=m+1}^{n} {p_i}^{s_{i\cdot j}} \right|$$

[11]These two suggestions are not meant to be exhaustive, nor exclusive. In fact, few more are expressly omitted for further exploration by the author.

that are less than ${p_g}^2$, but greater than the unit, are prime, where m is less than n ($n \geq 2$). Let this statement be referred to as the *prime number statement*.

Returning to the point of departure, the "seed" set $\{1, 2\}$, and imposing the boundary condition at each iteration, one obtains what follows. For the first iteration I_1, C is $2^2 = 4$ (the greatest of the set squared). As before, from the binary partition of the set $\{1, 2\}$

$$\{\,1\,\}\ and\ \{\,2\,\}$$

one gets two immeasurable magnitudes (less than C):

$$1^{s_{1\cdot j}} + 2^{s_{2\cdot j}}\ and\ \left|1^{s_{1\cdot j}} - 2^{s_{2\cdot j}}\right|$$
$$\Downarrow$$
$$1^1 + 2^1 = 3\ and\ \left|1^1 - 2^1\right| = 1$$
$$(when\ s_{1\cdot j} = s_{2\cdot j} = 1)$$

Thus, $\mathbb{P} = \{1, 2, 3\}$. For the second iteration I_2, let the smallest of the set $\mathbb{P} \setminus \{1, 2\}$ be adjoined to $\{1, 2\}$. Thus, the set for I_2 is $\{1, 2, 3\}$ from which one gets $C = 3^2 = 9$ (the greatest of the set squared). As before, one gets three unique ways of splitting the set in two:

$$\{\,1\,\}\ and\ \{\,2, 3\,\}$$
$$\{\,2\,\}\ and\ \{\,1, 3\,\}$$
$$\{\,3\,\}\ and\ \{\,1, 2\,\}$$

from which the following immeasurable magnitudes (less than C) are obtained:

$$1^1 + 2^1 \times 3^1 = 7\ and\ \left|1^1 - 2^1 \times 3^1\right| = 5$$
$$2^1 + 1^1 \times 3^1 = 5\ and\ \left|2^1 - 1^1 \times 3^1\right| = 1$$
$$3^1 + 1^1 \times 2^1 = 5\ and\ \left|3^1 - 1^1 \times 2^1\right| = 1$$

Thus $\mathbb{P} = \{1, 2, 3, 5, 7\}$. For the third iteration I_3, the set to use is the smallest of the set $\mathbb{P} \setminus \{1, 2, 3\}$ adjoined to $\{1, 2, 3\}$. Thus $C = 5^2 = 25$ (from the set $\{1, 2, 3, 5\}$) and the list of (unique) binary partitions (for I_3) as before is

$$\{\,1\,\}\ and\ \{\,2, 3, 5\,\}$$
$$\{\,1, 2\,\}\ and\ \{\,3, 5\,\}$$
$$\{\,1, 3\,\}\ and\ \{\,2, 5\,\}$$
$$\{\,1, 5\,\}\ and\ \{\,2, 3\,\}$$
$$\{\,2\,\}\ and\ \{\,1, 3, 5\,\}$$
$$\{\,3\,\}\ and\ \{\,1, 2, 5\,\}$$
$$\{\,5\,\}\ and\ \{\,1, 2, 3\,\}$$

from which one gets the following immeasurable magnitudes:

$$1^1 + 2^1 \times 3^1 \times 5^1 = 31\ and\ \left|1^1 - 2^1 \times 3^1 \times 5^1\right| = 29$$
$$1^1 \times 2^1 + 3^1 \times 5^1 = 17\ and\ \left|1^1 \times 2^1 - 3^1 \times 5^1\right| = 13$$
$$1^1 \times 3^1 + 2^1 \times 5^1 = 13\ and\ \left|1^1 \times 3^1 - 2^1 \times 5^1\right| = 7$$
$$1^1 \times 5^1 + 2^1 \times 3^1 = 11\ and\ \left|1^1 \times 5^1 - 2^1 \times 3^1\right| = 1$$
$$2^1 + 1^1 \times 3^1 \times 5^1 = 17\ and\ \left|2^1 - 1^1 \times 3^1 \times 5^1\right| = 13$$

$$3^1 + 1^1 \times 2^1 \times 5^1 = 13 \, and \, \left|3^1 - 1^1 \times 2^1 \times 5^1\right| = 7$$
$$5^1 + 1^1 \times 2^1 \times 3^1 = 11 \, and \, \left|5^1 - 1^1 \times 2^1 \times 3^1\right| = 1$$

Retaining only magnitudes or numbers that are less than $C = 5^2 = 25$, one gets $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, 17\}$. Although all elements of the set $\mathbb{P}$ except the unit are prime, $\mathbb{P}$ does not contain all prime numbers less than $C = 5^2 = 25$.

One is reminded that $s_{1 \cdot j} = s_{2 \cdot j} = \cdots = s_{n \cdot j} = 1$ (*i.e.* the set $\{s_{i \cdot 1}\}$); hence, the list cannot be thought of as exhaustive since only a minuscule subset of an infinity (comprising the whole table of repeated multiplications) is considered due to the constraint on the exponents. To remedy this, one can slightly, or gradually, increase the maximum exponent (*i.e.* consider more than one column) to allow more varied exponent combinations.

For the present experiment, let only *one* subset (the smallest if the number of elements of the whole "seed" set is odd) be allowed to have 2 as the maximum exponent (*i.e.* columns 1 and 2 are considered) while the other subset's maximum exponent remains 1 (*i.e.*, the magnitudes are constrained to the first column). Then one gets the following immeasurable magnitudes less than $C = 5^2 = 25$ for the third iteration I_3 (in no particular order and without eliminating superfluous duplicates[12]):

$$\begin{array}{ll}
1^1 \times 2^1 + 3^1 \times 5^1 = 17 & \left|1^1 \times 2^1 - 3^1 \times 5^1\right| = 13 \\
1^1 \times 2^2 + 3^1 \times 5^1 = 19 & \left|1^1 \times 2^2 - 3^1 \times 5^1\right| = 11 \\
1^2 \times 2^1 + 3^1 \times 5^1 = 17 & \left|1^2 \times 2^1 - 3^1 \times 5^1\right| = 13 \\
1^2 \times 2^2 + 3^1 \times 5^1 = 19 & \left|1^2 \times 2^2 - 3^1 \times 5^1\right| = 11 \\
1^1 \times 3^1 + 2^1 \times 5^1 = 13 & \left|1^1 \times 3^1 - 2^1 \times 5^1\right| = 7 \\
1^1 \times 3^2 + 2^1 \times 5^1 = 19 & 1^2 \times 3^1 + 2^1 \times 5^1 = 13 \\
\left|1^2 \times 3^1 - 2^1 \times 5^1\right| = 7 & 1^2 \times 3^2 + 2^1 \times 5^1 = 19 \\
1^1 \times 5^1 + 2^1 \times 3^1 = 11 & \left|1^1 \times 5^2 - 2^1 \times 3^1\right| = 19 \\
1^2 \times 5^1 + 2^1 \times 3^1 = 11 & \left|1^2 \times 5^2 - 2^1 \times 3^1\right| = 19 \\
2^1 + 1^1 \times 3^1 \times 5^1 = 17 & \left|2^1 - 1^1 \times 3^1 \times 5^1\right| = 13 \\
2^2 + 1^1 \times 3^1 \times 5^1 = 19 & \left|2^2 - 1^1 \times 3^1 \times 5^1\right| = 11 \\
2^1 \times 1^1 + 3^1 \times 5^1 = 17 & \left|2^1 \times 1^1 - 3^1 \times 5^1\right| = 13 \\
2^1 \times 1^2 + 3^1 \times 5^1 = 17 & \left|2^1 \times 1^2 - 3^1 \times 5^1\right| = 13 \\
2^2 \times 1^1 + 3^1 \times 5^1 = 19 & \left|2^2 \times 1^1 - 3^1 \times 5^1\right| = 11 \\
2^2 \times 1^2 + 3^1 \times 5^1 = 19 & \left|2^2 \times 1^2 - 3^1 \times 5^1\right| = 11 \\
2^1 \times 3^1 + 1^1 \times 5^1 = 11 & 2^1 \times 3^2 + 1^1 \times 5^1 = 23 \\
\left|2^1 \times 3^2 - 1^1 \times 5^1\right| = 13 & 2^2 \times 3^1 + 1^1 \times 5^1 = 17 \\
\left|2^2 \times 3^1 - 1^1 \times 5^1\right| = 7 & 2^1 \times 5^1 + 1^1 \times 3^1 = 13 \\
\left|2^1 \times 5^1 - 1^1 \times 3^1\right| = 7 & 2^2 \times 5^1 + 1^1 \times 3^1 = 23 \\
\left|2^2 \times 5^1 - 1^1 \times 3^1\right| = 17 & 3^1 + 1^1 \times 2^1 \times 5^1 = 13
\end{array}$$

[12]The repeated occurrence of the unit is omitted. The last section of this paper contains the program that produced these computations.

$$\begin{array}{ll}
\left|3^1 - 1^1 \times 2^1 \times 5^1\right| = 7 & 3^2 + 1^1 \times 2^1 \times 5^1 = 19 \\
3^1 \times 1^1 + 2^1 \times 5^1 = 13 & \left|3^1 \times 1^1 - 2^1 \times 5^1\right| = 7 \\
3^1 \times 1^2 + 2^1 \times 5^1 = 13 & \left|3^1 \times 1^2 - 2^1 \times 5^1\right| = 7 \\
3^2 \times 1^1 + 2^1 \times 5^1 = 19 & 3^2 \times 1^2 + 2^1 \times 5^1 = 19 \\
3^1 \times 2^1 + 1^1 \times 5^1 = 11 & 3^1 \times 2^2 + 1^1 \times 5^1 = 17 \\
\left|3^1 \times 2^2 - 1^1 \times 5^1\right| = 7 & 3^2 \times 2^1 + 1^1 \times 5^1 = 23 \\
\left|3^2 \times 2^1 - 1^1 \times 5^1\right| = 13 & 3^1 \times 5^1 + 1^1 \times 2^1 = 17 \\
\left|3^1 \times 5^1 - 1^1 \times 2^1\right| = 13 & 5^1 + 1^1 \times 2^1 \times 3^1 = 11 \\
\left|5^2 - 1^1 \times 2^1 \times 3^1\right| = 19 & 5^1 \times 1^1 + 2^1 \times 3^1 = 11 \\
5^1 \times 1^2 + 2^1 \times 3^1 = 11 & \left|5^2 \times 1^1 - 2^1 \times 3^1\right| = 19 \\
\left|5^2 \times 1^2 - 2^1 \times 3^1\right| = 19 & \left|5^2 \times 1^2 - 2^1 \times 3^1\right| = 19 \\
5^1 \times 2^1 + 1^1 \times 3^1 = 13 & \left|5^1 \times 2^1 - 1^1 \times 3^1\right| = 7 \\
5^1 \times 2^2 + 1^1 \times 3^1 = 23 & \left|5^1 \times 2^2 - 1^1 \times 3^1\right| = 17 \\
5^1 \times 3^1 + 1^1 \times 2^1 = 17 & \left|5^1 \times 3^1 - 1^1 \times 2^1\right| = 13
\end{array}$$

As predicted, all found numbers (less than C) are repeatedly prime and the set $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23\}$ is exactly the complete set of prime numbers less than $C = 5^2 = 25$ (the upper bound of the computation of the iteration I_3) if the unit is ignored.

How far can this program be carried? For the fourth iteration I_4, the new "seed" set is obtained by adjoining the smallest from the set $\mathbb{P} \setminus \{1, 2, 3, 5\}$ to the set $\{1, 2, 3, 5\}$. Thus the set for the current iteration is $\{1, 2, 3, 5, 7\}$ from which one obtains the following immeasurable magnitudes less than $C = 7^2 = 49$:

$$\begin{array}{ll}
1^1 \times 5^1 + 2^1 \times 3^1 \times 7^1 = 47 & \left|1^1 \times 5^1 - 2^1 \times 3^1 \times 7^1\right| = 37 \\
\left|1^1 \times 5^2 - 2^1 \times 3^1 \times 7^1\right| = 17 & 1^2 \times 5^1 + 2^1 \times 3^1 \times 7^1 = 47 \\
\left|1^2 \times 5^1 - 2^1 \times 3^1 \times 7^1\right| = 37 & \left|1^2 \times 5^2 - 2^1 \times 3^1 \times 7^1\right| = 17 \\
1^1 \times 7^1 + 2^1 \times 3^1 \times 5^1 = 37 & \left|1^1 \times 7^1 - 2^1 \times 3^1 \times 5^1\right| = 23 \\
1^2 \times 7^1 + 2^1 \times 3^1 \times 5^1 = 37 & \left|1^2 \times 7^1 - 2^1 \times 3^1 \times 5^1\right| = 23 \\
\left|1^2 \times 7^2 - 2^1 \times 3^1 \times 5^1\right| = 19 & 2^1 \times 3^1 + 1^1 \times 5^1 \times 7^1 = 41 \\
\left|2^1 \times 3^1 - 1^1 \times 5^1 \times 7^1\right| = 29 & \left|2^1 \times 3^2 - 1^1 \times 5^1 \times 7^1\right| = 17 \\
2^2 \times 3^1 + 1^1 \times 5^1 \times 7^1 = 47 & \left|2^2 \times 3^1 - 1^1 \times 5^1 \times 7^1\right| = 23 \\
2^1 \times 5^1 + 1^1 \times 3^1 \times 7^1 = 31 & \left|2^1 \times 5^1 - 1^1 \times 3^1 \times 7^1\right| = 11 \\
\left|2^1 \times 5^2 - 1^1 \times 3^1 \times 7^1\right| = 29 & 2^2 \times 5^1 + 1^1 \times 3^1 \times 7^1 = 41 \\
2^1 \times 7^1 + 1^1 \times 3^1 \times 5^1 = 29 & 2^2 \times 7^1 + 1^1 \times 3^1 \times 5^1 = 43 \\
\left|2^2 \times 7^1 - 1^1 \times 3^1 \times 5^1\right| = 13 & 3^1 \times 2^1 + 1^1 \times 5^1 \times 7^1 = 41 \\
\left|3^1 \times 2^1 - 1^1 \times 5^1 \times 7^1\right| = 29 & 3^1 \times 2^2 + 1^1 \times 5^1 \times 7^1 = 47 \\
\left|3^1 \times 2^2 - 1^1 \times 5^1 \times 7^1\right| = 23 & \left|3^2 \times 2^1 - 1^1 \times 5^1 \times 7^1\right| = 17 \\
3^1 \times 5^1 + 1^1 \times 2^1 \times 7^1 = 29 & \left|3^2 \times 5^1 - 1^1 \times 2^1 \times 7^1\right| = 31 \\
3^1 \times 7^1 + 1^1 \times 2^1 \times 5^1 = 31 & \left|3^1 \times 7^1 - 1^1 \times 2^1 \times 5^1\right| = 11
\end{array}$$

$$5^1 + 1^1 \times 2^1 \times 3^1 \times 7^1 = 47 \quad \left|5^1 - 1^1 \times 2^1 \times 3^1 \times 7^1\right| = 37$$
$$\left|5^2 - 1^1 \times 2^1 \times 3^1 \times 7^1\right| = 17 \quad 5^1 \times 1^1 + 2^1 \times 3^1 \times 7^1 = 47$$
$$\left|5^1 \times 1^1 - 2^1 \times 3^1 \times 7^1\right| = 37 \quad 5^1 \times 1^2 + 2^1 \times 3^1 \times 7^1 = 47$$
$$\left|5^1 \times 1^2 - 2^1 \times 3^1 \times 7^1\right| = 37 \quad \left|5^2 \times 1^1 - 2^1 \times 3^1 \times 7^1\right| = 17$$
$$\left|5^2 \times 1^2 - 2^1 \times 3^1 \times 7^1\right| = 17 \quad 5^1 \times 2^1 + 1^1 \times 3^1 \times 7^1 = 31$$
$$\left|5^1 \times 2^1 - 1^1 \times 3^1 \times 7^1\right| = 11 \quad 5^1 \times 2^2 + 1^1 \times 3^1 \times 7^1 = 41$$
$$\left|5^2 \times 2^1 - 1^1 \times 3^1 \times 7^1\right| = 29 \quad 5^1 \times 3^1 + 1^1 \times 2^1 \times 7^1 = 29$$
$$\left|5^1 \times 3^2 - 1^1 \times 2^1 \times 7^1\right| = 31 \quad 5^1 \times 7^1 + 1^1 \times 2^1 \times 3^1 = 41$$
$$\left|5^1 \times 7^1 - 1^1 \times 2^1 \times 3^1\right| = 29 \quad 7^1 + 1^1 \times 2^1 \times 3^1 \times 5^1 = 37$$
$$\left|7^1 - 1^1 \times 2^1 \times 3^1 \times 5^1\right| = 23 \quad \left|7^2 - 1^1 \times 2^1 \times 3^1 \times 5^1\right| = 19$$
$$7^1 \times 1^1 + 2^1 \times 3^1 \times 5^1 = 37 \quad \left|7^1 \times 1^1 - 2^1 \times 3^1 \times 5^1\right| = 23$$
$$7^1 \times 1^2 + 2^1 \times 3^1 \times 5^1 = 37 \quad \left|7^1 \times 1^2 - 2^1 \times 3^1 \times 5^1\right| = 23$$
$$\left|7^2 \times 1^1 - 2^1 \times 3^1 \times 5^1\right| = 19 \quad \left|7^2 \times 1^2 - 2^1 \times 3^1 \times 5^1\right| = 19$$
$$7^1 \times 2^1 + 1^1 \times 3^1 \times 5^1 = 29 \quad 7^1 \times 2^2 + 1^1 \times 3^1 \times 5^1 = 43$$
$$\left|7^1 \times 2^2 - 1^1 \times 3^1 \times 5^1\right| = 13 \quad 7^1 \times 3^1 + 1^1 \times 2^1 \times 5^1 = 31$$
$$\left|7^1 \times 3^1 - 1^1 \times 2^1 \times 5^1\right| = 11 \quad 7^1 \times 5^1 + 1^1 \times 2^1 \times 3^1 = 41$$
$$\left|7^1 \times 5^1 - 1^1 \times 2^1 \times 3^1\right| = 29$$

Thus $\mathbb{P} = \{1,\ 2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19,\ 23,\ 29,\ 31,\ 37,\ 41,\ 43,\ 47\}$; one obtains (repeatedly) exactly all prime numbers less than $C = 7^2 = 49$ if the unit is ignored.

For the fifth iteration I_5, the "seed" set is $\{1,\ 2,\ 3,\ 5,\ 7,\ 11\}$, from which one gets the series of (non-exhaustively listed) computations (where $C = 11^2 = 121$):

$$\left|1^2 \times 2^1 \times 7^2 - 3^1 \times 5^1 \times 11^1\right| = 67 \quad \left|1^2 \times 2^2 \times 7^2 - 3^1 \times 5^1 \times 11^1\right| = 31$$
$$\left|1^1 \times 2^1 \times 11^1 - 3^1 \times 5^1 \times 7^1\right| = 83 \quad \left|1^1 \times 2^2 \times 11^1 - 3^1 \times 5^1 \times 7^1\right| = 61$$
$$\left|1^1 \times 3^1 \times 5^2 - 2^1 \times 7^1 \times 11^1\right| = 79 \quad \left|1^1 \times 3^2 \times 5^1 - 2^1 \times 7^1 \times 11^1\right| = 109$$
$$\left|1^1 \times 3^2 \times 5^2 - 2^1 \times 7^1 \times 11^1\right| = 71 \quad \left|1^2 \times 3^1 \times 5^2 - 2^1 \times 7^1 \times 11^1\right| = 79$$
$$\left|1^2 \times 3^2 \times 5^1 - 2^1 \times 7^1 \times 11^1\right| = 109 \quad \left|1^2 \times 3^2 \times 5^2 - 2^1 \times 7^1 \times 11^1\right| = 71$$
$$\left|1^1 \times 3^1 \times 7^1 - 2^1 \times 5^1 \times 11^1\right| = 89 \quad \left|1^1 \times 3^1 \times 7^2 - 2^1 \times 5^1 \times 11^1\right| = 37$$
$$\left|1^1 \times 3^2 \times 7^1 - 2^1 \times 5^1 \times 11^1\right| = 47 \quad \left|1^2 \times 3^1 \times 7^1 - 2^1 \times 5^1 \times 11^1\right| = 89$$
$$\left|1^2 \times 3^1 \times 7^2 - 2^1 \times 5^1 \times 11^1\right| = 37 \quad \left|1^2 \times 3^2 \times 7^1 - 2^1 \times 5^1 \times 11^1\right| = 47$$
$$1^1 \times 3^1 \times 11^1 + 2^1 \times 5^1 \times 7^1 = 103 \quad \left|1^1 \times 3^1 \times 11^1 - 2^1 \times 5^1 \times 7^1\right| = 37$$
$$\left|1^1 \times 3^2 \times 11^1 - 2^1 \times 5^1 \times 7^1\right| = 29 \quad 1^2 \times 3^1 \times 11^1 + 2^1 \times 5^1 \times 7^1 = 103$$
$$\left|1^2 \times 3^1 \times 11^1 - 2^1 \times 5^1 \times 7^1\right| = 37 \quad \left|1^2 \times 3^2 \times 11^1 - 2^1 \times 5^1 \times 7^1\right| = 29$$
$$\left|1^1 \times 5^1 \times 3^2 - 2^1 \times 7^1 \times 11^1\right| = 109 \quad \left|1^1 \times 5^2 \times 3^1 - 2^1 \times 7^1 \times 11^1\right| = 79$$
$$\left|1^1 \times 5^2 \times 3^2 - 2^1 \times 7^1 \times 11^1\right| = 71 \quad \left|1^2 \times 5^1 \times 3^2 - 2^1 \times 7^1 \times 11^1\right| = 109$$
$$\left|1^2 \times 5^2 \times 3^1 - 2^1 \times 7^1 \times 11^1\right| = 79 \quad \left|1^2 \times 5^2 \times 3^2 - 2^1 \times 7^1 \times 11^1\right| = 71$$
$$1^1 \times 5^1 \times 7^1 + 2^1 \times 3^1 \times 11^1 = 101 \quad \left|1^1 \times 5^1 \times 7^1 - 2^1 \times 3^1 \times 11^1\right| = 31$$

$$\begin{array}{cc} \left|1^1 \times 5^2 \times 7^1 - 2^1 \times 3^1 \times 11^1\right| = 109 & 1^2 \times 5^1 \times 7^1 + 2^1 \times 3^1 \times 11^1 = 101 \\ 1^1 \times 5^1 \times 11^1 + 2^1 \times 3^1 \times 7^1 = 97 & \left|1^1 \times 5^1 \times 11^1 - 2^1 \times 3^1 \times 7^1\right| = 13 \\ \left|2^1 \times 3^1 \times 5^2 - 1^1 \times 7^1 \times 11^1\right| = 73 & \left|2^2 \times 5^1 \times 7^1 - 1^1 \times 3^1 \times 11^1\right| = 107 \\ \vdots & \vdots \end{array}$$

At the end of all computations of the iteration I_5, one gets $\mathbb{P} = \{1,\ 2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19,\ 23,\ 29,\ 31,\ 37,\ 41,\ 43,\ 47,\ 61,\ 67,\ 71,\ 73,\ 79,\ 83,\ 89,\ 97,\ 101,\ 103,\ 107,\ 109\}$.[13] As expected, all found numbers except the unit are prime. However, as before, the set $\mathbb{P}$ does not contain all prime numbers less than $C = 11^2 = 121$. In order to try fill the gaps in the "continuum" of prime numbers, one could try various (whole) exponents, *i.e.* consider more columns or cells from the table of repeated multiplications.

Let *one* subset be allowed to have 5 as the maximum exponent (while the other subset's maximum exponent remains 1). Then the remaining prime numbers less than $C = 121$ are (only listing those that were not previously found):

$$\begin{array}{cc} \vdots & \vdots \\ \left|7^1 \times 2^4 - 5^1 \times 3^1 \times 1^1 \times 11^1\right| = 53 & \left|7^1 \times 2^5 - 5^1 \times 3^1 \times 1^1 \times 11^1\right| = 59 \\ \left|2^3 \times 3^1 \times 7^1 - 5^1 \times 1^1 \times 11^1\right| = 113 & \vdots \\ \vdots & \vdots \end{array}$$

Thus $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113\}$, which is, indeed, the complete list of prime numbers less that $C = 11^2 = 121$ if the unit is ignored.

It should be emphasized before moving forward that the insistence of finding *all* prime numbers below C (at each iteration) should not be seen as a crucial criteria for determining the correctness of the deduced prime number statement, as the algorithmic approach presented here is merely a convenient one, which was mostly designed to demonstrate this statement up to a certain level of satisfaction and assurance; only a small subset of all possible (whole) exponent combinations are considered and, furthermore, all elements of one subset (in each binary partition) are constrained to never be raised above the power of 1. Beyond these imposed restrictions, as one makes use of the infinite possibilities afforded by various exponent combinations (assembled from the table of repeated multiplications), it becomes exceedingly impractical to confine the demonstration to a small paper. To be sure and thoroughly exhaustive, one would need to allow every prime number or immeasurable magnitude to be raised to every imaginable whole power, *i.e.* consider the entire infinite table of repeated multiplications (and compute the results of every combination), which is obviously (physically) unfeasible and beyond the computational power of a personal machine.

Furthermore, it can readily be seen that, as one raises or varies the exponents (thus resulting in more combinations), the same primes may occur more than once (some more often than others). It may therefore happen that, depending on the choice of exponents, columns, or cells, a prime that was not found in a previous

[13]Due to the long list of computations, most have been omitted.

iteration appears later down the program in subsequent computations. In point of fact, this is exactly what one observes for the next two iterations.

Letting the program proceed to the sixth iteration I_6, the "seed" set to use is {1, 2, 3, 5, 7, 11, 13}. Then, at the end of all computations (with the same carried-over constraints on the exponents, *i.e.* by assuming 5 as the maximum exponent of the smallest subset), the set $\mathbb{P}$ will be {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 145, 151, 163, 167}, which are all prime if one ignores the unit, although not all primes less than $C = 13^2 = 169$ (the greatest in the "seed" set for the iteration squared) are in the set; 157 is not present.

Notwithstanding, imposing the same constraints on the exponents, 157 is the found in the next iteration, the seventh (I_7); and $\mathbb{P} = \{1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 181, 193, 197, 199, 227, 241, 251, 257, 263, 271, 281, 283\}$.

At this point, the rationale for requiring that the given set form a complete "prime continuum" (in the sense of containing all primes up to the greatest in the set) should become more apparent; for, otherwise, if there exists a gap in the given "seed" set (*i.e.* a prime is missing), our algorithm—if sufficiently exhaustive—may yield a composite magnitude that, although not divisible (measurable) by any element of the given ("seed") set, is divisible by the missing prime nonetheless. The choice of C, forming the upper bound of a window, then guarantees that no prime greater than the greatest in the given set can divide any of the magnitudes that will be obtained.

Naturally, in light of what has been shown so far, one would undoubtedly be compelled to come to the conclusion that prime numbers, albeit appearing unpredictably chaotic among composites, evidently obey certain (comprehensible) exact, deterministic laws that are intrinsic (to prime numbers) and, hence, entirely independent of their derivatives, *i.e.* the composites; thus, primes ought to be treated separately, divorced from the necessity of apprehending them relatively to the contrasting background of composites, which only further obfuscates their true nature or meaning; one tends to speak of distribution of primes only in relation to other objects such as the composites.

Unless one is more inclined to attribute the numerical results and their precision or exactness to pure chance, the picture that emerges from this small experiment appear to suggest that prime numbers are simply a particular case of *immeasurable magnitudes* (which includes negative primes). In contrast to the common definition or description of primes, they can be aptly described instead, in a broad sense, as those entities which do not share any common measure (other than the unit). Thus, in this sense, words like "immeasurable" (borrowed from Euclid), "indivisible," "prime," *etc.* essentially convey the same idea—that of multiplicatively asymmetric entities.

Recapitulating, it can be restated that *given a set of n unique immeasurable magnitudes or numbers, $p_1, \cdots, p_n$, that contains all prime numbers up to the greatest in the set, p_g, then all numbers of the form*

$$\left| \prod_{i=1}^{m} p_i^{s_{i\cdot j}} \pm \prod_{i=m+1}^{n} p_i^{s_{i\cdot j}} \right|$$

that are less than $C = {p_g}^2$ (i.e. the greatest in the set squared) and greater than the unit are prime, where $s_{1\cdot j}, \cdots, s_{n\cdot j}$ are whole numbers and m is a whole number less than n ($n \geq 2$).[14]

This statement can be sanitized into a more concise form by omitting the (arguably) superfluous subscripts in the exponents, which were introduced primarily for illustrative purposes: *Given a set of n unique immeasurable magnitudes, $p_1, \cdots, p_n$, that contains all primes up to the greatest in the set, p_g, then all numbers of the form*

$$\left| \prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k} \right|$$

that are less than $C = {p_g}^2$ and greater than the unit are prime, where s_j and s_k denote (any) whole numbers (from the infinite set $\mathbb{N}_\infty$) and m is a whole number less than n ($n \geq 2$).

Now from a more careful reflection on the above statement, it can be further remarked that one needs only be concerned with the greatest element that marks the upper bound of a "prime continuum" in a given set of immeasurable magnitudes in order to determine C. Furthermore, since all primes below p_g (*i.e.* the greatest element marking the upper bound of the "prime continuum") are assumed to be present in the set, the window or bounds can be updated to simply say that all computed magnitudes occurring between p_g and $C = p_g^2$ are prime.

Thus, it can be said that *given n unique immeasurable magnitudes, $p_1, \cdots, p_n$, that contains a complete prime continuum, $p_1, \cdots, p_g$, where p_g, marking the upper bound (greatest) of this prime continuum, is less than or equal to p_n, then all numbers of the form*

$$\left| \prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k} \right|$$

that are less than $C = {p_g}^2$ and greater than p_g are prime, where s_j and s_k denote (any) whole number and m is a whole number less than n ($n \geq 2$).

For example, if the given ("seed") set is {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113}, then 47 marks the upper bound of the complete prime continuum because all primes below it are present in the set (as opposed to any element greater than 47 since 53 is missing, which breaks the continuum). Therefore, $C = 47^2 = 2209$ for the *whole* set.

As one should suspect, the formulated statements above are by no means exhaustive; more insights can undoubtedly be deduced such that the algorithm may perhaps be fine-tuned to be more efficient.[15] The final statement—as it applies primes only—can then be reformulated in more modern terminology as below.

> *If a set of n unique prime numbers, $p_1, \cdots, p_n$, contains a complete continuum of primes such that p_g marks the (inclusive) upper bound of this*

[14] By *immeasurable magnitudes* in this statement, it is meant the unit 1 adjoined to a (complete) set of prime numbers forming a continuum in the sense of not having gaps up to the greatest in the set.

[15] No effort has been made to improve the efficiency of the program whatsoever.

continuum and all primes below p_g are present in the set, then all numbers of the form

$$\left| \prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k} \right|$$

that are greater than p_g and less than ${p_g}^2$ (that is, $p_g < \left|\prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k}\right| < p_g^2$) are also prime, where m is less than n ($n \geq 2$) and s_j and s_k denote (any) whole numbers.[a]

[a]It must not be misunderstood by s_j and s_k that only two whole numbers are applied as exponents to the expression $\left|\prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k}\right|$; but rather, it must be interpreted that, by expanding any of the two products, each prime (in the expression) can be raised to any whole number (independently of the others).

4. Crude programs

During the composition of this work, small programs were written in Kotlin to ease the tedious labor required for its verification and completion. Some (annotated with comments) are reproduced below—essentially unaltered from their original forms—as proof of correctness so that the reader may also verify the demonstration in the previous section and not feel like a victim of a well-performed mathematical trick of smoke and mirrors. Beyond this assurance, the programs may offer the reader an opportunity to experiment at his or her leisure and perhaps reveal what was not obvious to the author.

```
/* prime_number_algorithm_1.kt */

import kotlin.math.absoluteValue
import kotlin.math.pow

val MAX_ITERATIONS = 7
val MAX_EXPONENT = 5 // This maximum exponent is applied only to one subset.
val MAX_BOUND = 1000
var currentIterationIndex = 0
var currentBatchOfImmeasurables = mutableListOf<Long>() // The current list of immeasurables, which gets collected and cleared after every iteration.
var allImmeasurables = mutableListOf<Long>() // All immeasurables found.
var currentLadderStep = 0L // The biggest element used to calculate  C (the upper bound below which calculated immeasurables are prime).
val countMap = HashMap<Long, Int>() // A map used to count the number of times an immeasurable is found.
var exponentsMap = HashMap<Int, MutableList<MutableList<Int>>>() // A map that stores previously computed exponent variations.

fun main(args: Array<String>) {
    val seedImmeasurables = mutableListOf(1L, 2L)
    val seedImmeasurableString = getSetString(seedImmeasurables)
    println("Given the seed set $seedImmeasurableString")
    allImmeasurables.addAll(seedImmeasurables)
    currentLadderStep = seedImmeasurables.max()!!
```

```
    yieldImmeasurables(seedImmeasurables = seedImmeasurables,
    splitSwapIndex = 0)
}

fun getSetString(list: List<Any>) = list.toString().replace("[", "{").
    replace("]", "}")

// A function that generates exponents in an incremental manner for a
    specified size of magnitudes that need to be raised to a power.
fun generateExponents(numberOfImmeasurableToBeRaised: Int):
    MutableList<MutableList<Int>> {
    val variations = mutableListOf<MutableList<Int>>()
    val currentVariation = MutableList(numberOfImmeasurableToBeRaised)
     { 1 }
    val lastVariationAccumulatedProduct = List(
    numberOfImmeasurableToBeRaised) { MAX_EXPONENT }.reduce {
    accumulatedProduct, i -> accumulatedProduct * i }
    var index = 0
    variations.add(currentVariation.toMutableList())
    while (currentVariation.reduce { accumulatedProduct, i ->
    accumulatedProduct * i } < lastVariationAccumulatedProduct) {
        while (currentVariation[numberOfImmeasurableToBeRaised - 1 -
    index] + 1 > MAX_EXPONENT) {
            if (numberOfImmeasurableToBeRaised - 1 - index > 0) {
                currentVariation[numberOfImmeasurableToBeRaised - 1 -
    index] = 1
            }
            ++index
        }
        if (index > numberOfImmeasurableToBeRaised - 1) {
            index = 0
        }
        currentVariation[numberOfImmeasurableToBeRaised - 1 - index]
    += 1
        index = 0
        variations.add(currentVariation.toMutableList())
    }
    return variations
}

fun isWithinBound(found: Long, upperBound: Long) = found < upperBound

fun addFoundImmeasurableToMap(found: Long) {
    if (countMap.containsKey(found)) {
        countMap[found] = countMap[found]!! + 1
    } else {
        countMap[found] = 1
    }
}

// The main function which recursively yields immeasurables by using a
     split index as a swap pivot to generate the two sets or lists of
    immeasurables.
fun yieldImmeasurables(seedImmeasurables: MutableList<Long>,
    splitSwapIndex: Int) {
    val sizeOfCurrentSeedImmeasurables = seedImmeasurables.size
    if (sizeOfCurrentSeedImmeasurables < 2 || currentIterationIndex >=
     MAX_ITERATIONS) {
```

```
        return
    }
    var swapCursor = splitSwapIndex + 1
    while (swapCursor < sizeOfCurrentSeedImmeasurables + 1) {
        val leftCollection = seedImmeasurables.subList(0,
    splitSwapIndex + 1)
        val rightCollection = seedImmeasurables.subList(splitSwapIndex
     + 1, sizeOfCurrentSeedImmeasurables)
        // primeFulcrum is the successive product of the set that is
    constrained to never be raised above the power of 1.
        val primeFulcrum = rightCollection.reduce { acc, i -> acc * i
    }
        // This generates exponents if not previously stored.
        val listOfExponents = if (exponentsMap.containsKey(
    leftCollection.size)) {
            exponentsMap[leftCollection.size]
        } else {
            exponentsMap[leftCollection.size] = generateExponents(
    numberOfImmeasurableToBeRaised = leftCollection.size)
            exponentsMap[leftCollection.size]
        }
        listOfExponents!!.forEach { exponentsList ->
            val raisedLeftCollection = leftCollection.mapIndexed {
    index, prime -> (prime.toDouble().pow(exponentsList[index].
    toDouble())).toInt() }.toMutableList()
            val leftRepeatedMultiplications = raisedLeftCollection.
    reduce { acc, i -> acc * i }
            if (leftRepeatedMultiplications < 0 || primeFulcrum < 0) {
                return@forEach
            }
            val firstPrime = leftRepeatedMultiplications +
    primeFulcrum
            val secondPrime = (leftRepeatedMultiplications -
    primeFulcrum).absoluteValue
            val currentUpperBound = currentLadderStep *
    currentLadderStep
            if (isWithinBound(firstPrime, currentUpperBound)) {
                currentBatchOfImmeasurables.add(firstPrime)
                addFoundImmeasurableToMap(firstPrime)
            }
            if (isWithinBound(secondPrime, currentUpperBound)) {
                currentBatchOfImmeasurables.add(secondPrime)
                addFoundImmeasurableToMap(secondPrime)
            }
        }
        val tempImmeasurables = mutableListOf<Long>()
        tempImmeasurables.addAll(seedImmeasurables)
        if (splitSwapIndex + 1 < sizeOfCurrentSeedImmeasurables / 2) {
            yieldImmeasurables(tempImmeasurables, splitSwapIndex + 1)
        }
        if (swapCursor < sizeOfCurrentSeedImmeasurables) {
            seedImmeasurables[splitSwapIndex] = seedImmeasurables[
    splitSwapIndex] xor seedImmeasurables[swapCursor]
            seedImmeasurables[swapCursor] = seedImmeasurables[
    splitSwapIndex] xor seedImmeasurables[swapCursor]
            seedImmeasurables[splitSwapIndex] = seedImmeasurables[
    splitSwapIndex] xor seedImmeasurables[swapCursor]
        } else if (swapCursor == sizeOfCurrentSeedImmeasurables) {
```

```
            if (splitSwapIndex == 0 && currentLadderStep < MAX_BOUND) {
                currentIterationIndex++
                val foundImmeasurables = currentBatchOfImmeasurables.distinct().filter { it !in allImmeasurables }
                allImmeasurables.addAll(foundImmeasurables)
                currentLadderStep = allImmeasurables.filter { it > currentLadderStep }.min()!!
                currentBatchOfImmeasurables.clear()
                val newSeedImmeasurables = seedImmeasurables.toMutableList()
                newSeedImmeasurables.add(currentLadderStep)
                println("At iteration $currentIterationIndex this program found the following immeasurables: ${getSetString(allImmeasurables.sorted())},")
                var primeCountString = allImmeasurables.sorted().filter { it != 2L }.map { prime ->
                    "$prime was found ${countMap[prime]} times"
                }.toString().replace("[", "").replace("]", "")
                val lastIndexOfComma = primeCountString.lastIndexOf(",")
                primeCountString = primeCountString.replaceRange(lastIndexOfComma + 1, lastIndexOfComma + 2, " and ")
                println("where $primeCountString.")
                yieldImmeasurables(newSeedImmeasurables, 0)
            }
        }
        swapCursor++
    }
}
```

The program above should require little to no explication. In Essence, it (more or less) retraces the demonstration described in the previous section. Unfortunately, from an efficiency standpoint, there is much to be desired; nothing is done to effectively reduce the (very) rapidly increasing number of computations or duplicate results (other than imposing arbitrary constraints on the exponents); furthermore, little is done to mitigate computations that exceeds the range of the type that represents primes (`Long`). As one might imagine, once such out-of-bound computations occur, the program greatly suffers in its predictive power, and hence, is entirely unreliable beyond certain limits. This is of course a problem inherited from the limitations in data representations in computers. If such limitations did not exist, the program or algorithm should with certainty always yield primes (at least in theory).

Presuming Kotlin is installed on a modern machine,[16] running the following terminal commands should yield the correct results:

```
kotlinc prime_number_algorithm_1.kt -include-runtime -d prime_number_algorithm_1.jar
java -jar prime_number_algorithm_1.jar
Given the seed set {1, 2}
At iteration 1 this program found the following immeasurables: {1, 2, 3},
where 1 was found 6 times, and 3 was found 7 times.
```

[16]All programs were written and tested on a (64-bit) MacBook Pro (running Mojave) with an Intel Core i9 processor.

```
At iteration 2 this program found the following immeasurables: {1, 2, 3, 5, 7},
where 1 was found 9 times, 3 was found 7 times, 5 was found 8 times, and 7 was found 7 times.
At iteration 3 this program found the following immeasurables: {1, 2, 3, 5, 7, 11, 13, 17, 19, 23},
where 1 was found 44 times, 3 was found 7 times, 5 was found 8 times, 7 was found 33 times, 11 was found 24 times, 13 was found 28 times, 17 was found 39 times, 19 was found 35 times, and 23 was found 15 times.
At iteration 4 this program found the following immeasurables: {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47},
where 1 was found 52 times, 3 was found 7 times, 5 was found 8 times, 7 was found 33 times, 11 was found 41 times, 13 was found 32 times, 17 was found 52 times, 19 was found 50 times, 23 was found 28 times, 29 was found 10 times, 31 was found 6 times, 37 was found 24 times, 41 was found 8 times, 43 was found 13 times, and 47 was found 13 times.
At iteration 5 this program found the following immeasurables: {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113},
where 1 was found 52 times, 3 was found 7 times, 5 was found 8 times, 7 was found 33 times, 11 was found 41 times, 13 was found 87 times, 17 was found 90 times, 19 was found 114 times, 23 was found 28 times, 29 was found 48 times, 31 was found 108 times, 37 was found 94 times, 41 was found 8 times, 43 was found 19 times, 47 was found 115 times, 53 was found 32 times, 59 was found 32 times, 61 was found 64 times, 67 was found 32 times, 71 was found 102 times, 73 was found 6 times, 79 was found 64 times, 83 was found 38 times, 89 was found 49 times, 97 was found 76 times, 101 was found 70 times, 103 was found 44 times, 107 was found 44 times, 109 was found 96 times, and 113 was found 6 times.
At iteration 6 this program found the following immeasurables: {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 163, 167},
where 1 was found 52 times, 3 was found 7 times, 5 was found 8 times, 7 was found 33 times, 11 was found 41 times, 13 was found 87 times, 17 was found 134 times, 19 was found 146 times, 23 was found 60 times, 29 was found 60 times, 31 was found 108 times, 37 was found 94 times, 41 was found 26 times, 43 was found 25 times, 47 was found 115 times, 53 was found 38 times, 59 was found 70 times, 61 was found 70 times, 67 was found 64 times, 71 was found 102 times, 73 was found 18 times, 79 was found 76 times, 83 was found 44 times, 89 was found 49 times, 97 was found 76 times, 101 was found 88 times, 103 was found 44 times, 107 was found 76 times, 109 was found 96 times, 113 was found 23 times, 127 was found 38 times, 131 was found 6 times, 137 was found 32 times, 139 was found 6 times, 149 was found 38 times, 151 was found 6 times, 163 was found 12 times, and 167 was found 12 times.
At iteration 7 this program found the following immeasurables: {1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 181, 193, 197, 199, 227, 241, 251, 257, 263, 271, 281, 283},
```

```
where 1 was found 202 times, 3 was found 7 times, 5 was found 8 times,
     7 was found 33 times, 11 was found 41 times, 13 was found 87
    times, 17 was found 134 times, 19 was found 170 times, 23 was
    found 60 times, 29 was found 60 times, 31 was found 108 times, 37
    was found 94 times, 41 was found 152 times, 43 was found 25 times,
     47 was found 115 times, 53 was found 38 times, 59 was found 70
    times, 61 was found 70 times, 67 was found 64 times, 71 was found
    102 times, 73 was found 144 times, 79 was found 76 times, 83 was
    found 170 times, 89 was found 73 times, 97 was found 202 times,
    101 was found 88 times, 103 was found 68 times, 107 was found 352
    times, 109 was found 222 times, 113 was found 149 times, 127 was
    found 38 times, 131 was found 258 times, 137 was found 158 times,
    139 was found 30 times, 149 was found 38 times, 151 was found 6
    times, 157 was found 24 times, 163 was found 12 times, 167 was
    found 12 times, 181 was found 24 times, 193 was found 126 times,
    197 was found 126 times, 199 was found 126 times, 227 was found
    150 times, 241 was found 252 times, 251 was found 150 times, 257
    was found 126 times, 263 was found 150 times, 271 was found 158
    times, 281 was found 24 times, and 283 was found 252 times.
```

One observation worth pointing out here is the repeated occurrence of the unit at every iteration while primes less than or equal to $|\sqrt{C}|$ never reappear.

In the spirit of experimentation, few variants of this program were written. The next one is slightly different in the following way: Given an initial "seed" set of unique (relatively) immeasurable magnitudes that forms a complete "prime continuum" (which, by default, is the initial set $\mathbb{P}$), every newly found magnitude (*i.e* "branch") is united to the "seed" set of the previous iteration that forms a "prime continuum" to form multiple ("seed") sets; the upper bound C is calculated by squaring either the greatest magnitude of the initial (first) set (for the first iteration) or the smallest magnitude of the set $\mathbb{P} \setminus \{\textit{the seed set of the previous iteration that forms a prime continuum}\}$; then, for every formed set, all computed magnitudes (in the form of $\left|\prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k}\right|$) under C are regarded as immeasurable magnitudes (*i.e.* primes if the unit is ignored).

For example, suppose that from the "seed" set of immeasurable magnitudes {1, 2, 3, 5}, one obtains the following immeasurable magnitudes {7, 11, 13, 17, 19, 23}; then every newly found magnitudes is united to the previous "seed" set (that forms a "prime continuum") to obtain the sets below:

$$\{1, 2, 3, 5, 7\}$$
$$\{1, 2, 3, 5, 11\}$$
$$\{1, 2, 3, 5, 13\}$$
$$\{1, 2, 3, 5, 17\}$$
$$\{1, 2, 3, 5, 19\}$$
$$\{1, 2, 3, 5, 23\}$$

Then, for every set above, immeasurable magnitudes are computed (using the expression $\left|\prod_{i=1}^{m} p_i^{s_j} \pm \prod_{i=m+1}^{n} p_i^{s_k}\right|$) as before where the upper bound C here is determined to be $7^2 = 49$.

Now, obviously, this variant adds more computations; however, one may then envisage lowering the maximum exponent and still obtain the expected prime numbers.

```
/* prime_number_algorithm_2.kt */

import kotlin.math.absoluteValue
import kotlin.math.pow

val MAX_ITERATIONS = 8
val MAX_EXPONENT = 5 // This maximum exponent is applied only to one subset.
val MAX_BOUND = 100000
var currentIterationIndex = 0
var currentBatchOfImmeasurables = mutableListOf<Long>() // The current list of immeasurables, which gets collected and cleared after every iteration.
var allImmeasurables = mutableListOf<Long>() // All immeasurables found.
var currentLadderStep = 0L // The biggest element used to calculate  C (the upper bound below which calculated immeasurables are prime).
val countMap = HashMap<Long, Int>() // A map used to count the number of times an immeasurable is found.
var exponentsMap = HashMap<Int, MutableList<MutableList<Int>>>() // A map that stores previously computed exponent variations.

fun main(args: Array<String>) {
    val seedImmeasurables = mutableListOf(1L, 2L)
    val seedImmeasurableString = getSetString(seedImmeasurables)
    println("Given the seed set $seedImmeasurableString")
    allImmeasurables.addAll(seedImmeasurables)
    currentLadderStep = seedImmeasurables.max()!!
    yieldImmeasurables(seedImmeasurables = seedImmeasurables, splitSwapIndex = 0)
}

fun getSetString(list: List<Any>) = list.toString().replace("[", "{").replace("]", "}")

// A function that generates exponents in an incremental manner for a specified size of magnitudes that need to be raised to a power.
fun generateExponents(numberOfImmeasurableToBeRaised: Int): MutableList<MutableList<Int>> {
    val variations = mutableListOf<MutableList<Int>>()
    val currentVariation = MutableList(numberOfImmeasurableToBeRaised) { 1 }
    val lastVariationAccumulatedProduct = List(numberOfImmeasurableToBeRaised) { MAX_EXPONENT }.reduce { accumulatedProduct, i -> accumulatedProduct * i }
    var index = 0
    variations.add(currentVariation.toMutableList())
    while (currentVariation.reduce { accumulatedProduct, i -> accumulatedProduct * i } < lastVariationAccumulatedProduct) {
        while (currentVariation[numberOfImmeasurableToBeRaised - 1 - index] + 1 > MAX_EXPONENT) {
            if (numberOfImmeasurableToBeRaised - 1 - index > 0) {
                currentVariation[numberOfImmeasurableToBeRaised - 1 - index] = 1
            }
            ++index
        }
        if (index > numberOfImmeasurableToBeRaised - 1) {
```

```
            index = 0
        }
        currentVariation[numberOfImmeasurableToBeRaised - 1 - index]
    += 1
        index = 0
        variations.add(currentVariation.toMutableList())
    }
    return variations
}

fun isWithinBound(found: Long, upperBound: Long) = found < upperBound

fun addFoundImmeasurableToMap(found: Long) {
    if (countMap.containsKey(found)) {
        countMap[found] = countMap[found]!! + 1
    } else {
        countMap[found] = 1
    }
}

// The main function which recursively yields immeasurables by using a
     split index as a swap pivot to generate the two sets or lists of
    immeasurables.
// isLeaf, when true, marks a recursion terminating node
fun yieldImmeasurables(seedImmeasurables: MutableList<Long>,
    splitSwapIndex: Int, isLeaf: Boolean = false) {
    val sizeOfCurrentSeedImmeasurables = seedImmeasurables.size
    if (sizeOfCurrentSeedImmeasurables < 2 || currentIterationIndex >=
     MAX_ITERATIONS) {
        return
    }
    var swapCursor = splitSwapIndex + 1
    while (swapCursor < sizeOfCurrentSeedImmeasurables + 1) {
        val leftCollection = seedImmeasurables.subList(0,
    splitSwapIndex + 1)
        val rightCollection = seedImmeasurables.subList(splitSwapIndex
     + 1, sizeOfCurrentSeedImmeasurables)
        // primeFulcrum is the successive product of the set that is
    constrained to never be raised above the power of 1.
        val primeFulcrum = rightCollection.reduce { acc, i -> acc * i
    }
        // This generates exponents if not previously stored.
        val listOfExponents = if (exponentsMap.containsKey(
    leftCollection.size)) {
            exponentsMap[leftCollection.size]
        } else {
            exponentsMap[leftCollection.size] = generateExponents(
    numberOfImmeasurableToBeRaised = leftCollection.size)
            exponentsMap[leftCollection.size]
        }
        listOfExponents!!.forEach { exponentsList ->
            val raisedLeftCollection = leftCollection.mapIndexed {
    index, prime -> (prime.toDouble().pow(exponentsList[index].
    toDouble())).toInt() }.toMutableList()
            val leftRepeatedMultiplications = raisedLeftCollection.
    reduce { acc, i -> acc * i }
            if (leftRepeatedMultiplications < 0 || primeFulcrum < 0) {
                return@forEach
```

```
            }
            val firstPrime = leftRepeatedMultiplications +
    primeFulcrum
            val secondPrime = (leftRepeatedMultiplications -
    primeFulcrum).absoluteValue
            val currentUpperBound = currentLadderStep *
    currentLadderStep
            if (isWithinBound(firstPrime, currentUpperBound)) {
                currentBatchOfImmeasurables.add(firstPrime)
                addFoundImmeasurableToMap(firstPrime)
            }
            if (isWithinBound(secondPrime, currentUpperBound)) {
                currentBatchOfImmeasurables.add(secondPrime)
                addFoundImmeasurableToMap(secondPrime)
            }
        }
        val tempImmeasurables = mutableListOf<Long>()
        tempImmeasurables.addAll(seedImmeasurables)
        if (splitSwapIndex + 1 < sizeOfCurrentSeedImmeasurables / 2) {
            yieldImmeasurables(tempImmeasurables, splitSwapIndex + 1)
        }
        if (swapCursor < sizeOfCurrentSeedImmeasurables) {
            seedImmeasurables[splitSwapIndex] = seedImmeasurables[
    splitSwapIndex] xor seedImmeasurables[swapCursor]
            seedImmeasurables[swapCursor] = seedImmeasurables[
    splitSwapIndex] xor seedImmeasurables[swapCursor]
            seedImmeasurables[splitSwapIndex] = seedImmeasurables[
    splitSwapIndex] xor seedImmeasurables[swapCursor]
        } else if (swapCursor == sizeOfCurrentSeedImmeasurables) {
            if (splitSwapIndex == 0 && currentLadderStep < MAX_BOUND
    && !isLeaf) {
                currentIterationIndex++
                val foundImmeasurables = currentBatchOfImmeasurables.
    distinct().filter { it !in allImmeasurables }
                allImmeasurables.addAll(foundImmeasurables)
                currentLadderStep = allImmeasurables.filter { it >
    currentLadderStep }.min()!!
                currentBatchOfImmeasurables.clear()
                foundImmeasurables.filter { it != currentLadderStep }.
    forEach {
                    val leafSeedImmeasurables = seedImmeasurables.
    distinct().toMutableList()
                    leafSeedImmeasurables.add(it)
                    yieldImmeasurables(leafSeedImmeasurables, 0, true)
                }
                val newSeedImmeasurables = seedImmeasurables.
    toMutableList()
                newSeedImmeasurables.add(currentLadderStep)
                println("At iteration $currentIterationIndex this
    program found the following immeasurables: ${getSetString(
    allImmeasurables.sorted())},")
                var primeCountString = allImmeasurables.sorted().
    filter { it != 2L }.map { prime ->
                    "$prime was found ${countMap[prime]} times"
                }.toString().replace("[", "").replace("]", "")
                val lastIndexOfComma = primeCountString.lastIndexOf
    (",")
```

```
                primeCountString = primeCountString.replaceRange(
    lastIndexOfComma + 1, lastIndexOfComma + 2, " and ")
                println("where $primeCountString.")
                yieldImmeasurables(newSeedImmeasurables, 0)
            }
        }
        swapCursor++
    }
}
```

Clearly, this modified algorithm slightly departs from the assertion of the final prime number statement in the previous section, yet, curiously enough, it correctly produces primes (if the unit is ignored) as can be observed below.

```
kotlinc prime_number_algorithm_2.kt -include-runtime -d
    prime_number_algorithm_2.jar
java -jar prime_number_algorithm_2.jar
Given the seed set {1, 2}
At iteration 1 this program found the following immeasurables: {1, 2,
    3},
where 1 was found 6 times, and 3 was found 7 times.
At iteration 2 this program found the following immeasurables: {1, 2,
    3, 5, 7},
where 1 was found 22 times, 3 was found 7 times, 5 was found 32 times,
     and 7 was found 7 times.
At iteration 3 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23},
where 1 was found 67 times, 3 was found 7 times, 5 was found 32 times,
     7 was found 60 times, 11 was found 88 times, 13 was found 102
    times, 17 was found 84 times, 19 was found 75 times, and 23 was
    found 66 times.
At iteration 4 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47},
where 1 was found 215 times, 3 was found 7 times, 5 was found 32 times
    , 7 was found 60 times, 11 was found 251 times, 13 was found 182
    times, 17 was found 191 times, 19 was found 204 times, 23 was
    found 237 times, 29 was found 119 times, 31 was found 134 times,
    37 was found 94 times, 41 was found 124 times, 43 was found 170
    times, and 47 was found 141 times.
At iteration 5 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113},
where 1 was found 239 times, 3 was found 7 times, 5 was found 32 times
    , 7 was found 60 times, 11 was found 251 times, 13 was found 305
    times, 17 was found 297 times, 19 was found 344 times, 23 was
    found 307 times, 29 was found 181 times, 31 was found 316 times,
    37 was found 176 times, 41 was found 224 times, 43 was found 296
    times, 47 was found 331 times, 53 was found 150 times, 59 was
    found 228 times, 61 was found 120 times, 67 was found 138 times,
    71 was found 292 times, 73 was found 120 times, 79 was found 152
    times, 83 was found 260 times, 89 was found 129 times, 97 was
    found 252 times, 101 was found 182 times, 103 was found 196 times,
     107 was found 132 times, 109 was found 196 times, and 113 was
    found 170 times.
At iteration 6 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131,
    137, 139, 149, 151, 157, 163, 167},
```

```
15 where 1 was found 515 times, 3 was found 7 times, 5 was found 32 times
       , 7 was found 60 times, 11 was found 251 times, 13 was found 305
       times, 17 was found 665 times, 19 was found 400 times, 23 was
       found 591 times, 29 was found 241 times, 31 was found 490 times,
       37 was found 374 times, 41 was found 290 times, 43 was found 302
       times, 47 was found 355 times, 53 was found 480 times, 59 was
       found 464 times, 61 was found 150 times, 67 was found 194 times,
       71 was found 418 times, 73 was found 408 times, 79 was found 164
       times, 83 was found 416 times, 89 was found 287 times, 97 was
       found 426 times, 101 was found 224 times, 103 was found 196 times,
        107 was found 440 times, 109 was found 370 times, 113 was found
       235 times, 127 was found 76 times, 131 was found 234 times, 137
       was found 414 times, 139 was found 132 times, 149 was found 86
       times, 151 was found 6 times, 157 was found 82 times, 163 was
       found 72 times, and 167 was found 232 times.
16 At iteration 7 this program found the following immeasurables: {1, 2,
       3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
       67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131,
       137, 139, 149, 151, 157, 163, 167, 173, 181, 191, 193, 197, 199,
       211, 223, 227, 229, 233, 241, 251, 257, 263, 269, 271, 277, 281,
       283},
17 where 1 was found 665 times, 3 was found 7 times, 5 was found 32 times
       , 7 was found 60 times, 11 was found 251 times, 13 was found 305
       times, 17 was found 665 times, 19 was found 550 times, 23 was
       found 591 times, 29 was found 265 times, 31 was found 616 times,
       37 was found 548 times, 41 was found 440 times, 43 was found 428
       times, 47 was found 355 times, 53 was found 504 times, 59 was
       found 512 times, 61 was found 276 times, 67 was found 194 times,
       71 was found 592 times, 73 was found 534 times, 79 was found 362
       times, 83 was found 590 times, 89 was found 311 times, 97 was
       found 552 times, 101 was found 224 times, 103 was found 220 times,
        107 was found 748 times, 109 was found 520 times, 113 was found
       535 times, 127 was found 202 times, 131 was found 486 times, 137
       was found 564 times, 139 was found 282 times, 149 was found 362
       times, 151 was found 132 times, 157 was found 256 times, 163 was
       found 72 times, 167 was found 508 times, 173 was found 174 times,
       181 was found 48 times, 191 was found 32 times, 193 was found 426
       times, 197 was found 276 times, 199 was found 300 times, 211 was
       found 24 times, 223 was found 276 times, 227 was found 372 times,
       229 was found 174 times, 233 was found 450 times, 241 was found
       276 times, 251 was found 150 times, 257 was found 378 times, 263
       was found 552 times, 269 was found 388 times, 271 was found 206
       times, 277 was found 158 times, 281 was found 174 times, and 283
       was found 426 times.
18 At iteration 8 this program found the following immeasurables: {1, 2,
       3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
       67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131,
       137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
       199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271,
       277, 281, 283, 293, 307, 313, 331, 337, 349, 353},
```

```
where 1 was found 665 times, 3 was found 7 times, 5 was found 32 times
    , 7 was found 60 times, 11 was found 251 times, 13 was found 305
    times, 17 was found 665 times, 19 was found 550 times, 23 was
    found 591 times, 29 was found 265 times, 31 was found 640 times,
    37 was found 548 times, 41 was found 488 times, 43 was found 428
    times, 47 was found 355 times, 53 was found 504 times, 59 was
    found 512 times, 61 was found 276 times, 67 was found 194 times,
    71 was found 592 times, 73 was found 534 times, 79 was found 386
    times, 83 was found 614 times, 89 was found 437 times, 97 was
    found 552 times, 101 was found 224 times, 103 was found 220 times,
     107 was found 748 times, 109 was found 520 times, 113 was found
    535 times, 127 was found 202 times, 131 was found 486 times, 137
    was found 564 times, 139 was found 282 times, 149 was found 362
    times, 151 was found 156 times, 157 was found 256 times, 163 was
    found 72 times, 167 was found 508 times, 173 was found 174 times,
    179 was found 72 times, 181 was found 48 times, 191 was found 32
    times, 193 was found 426 times, 197 was found 276 times, 199 was
    found 300 times, 211 was found 48 times, 223 was found 276 times,
    227 was found 420 times, 229 was found 174 times, 233 was found
    450 times, 239 was found 24 times, 241 was found 276 times, 251
    was found 150 times, 257 was found 378 times, 263 was found 552
    times, 269 was found 388 times, 271 was found 206 times, 277 was
    found 182 times, 281 was found 198 times, 283 was found 426 times,
     293 was found 24 times, 307 was found 48 times, 313 was found 24
    times, 331 was found 72 times, 337 was found 24 times, 349 was
    found 150 times, and 353 was found 120 times.
```

Another (exponentially inefficient) variant of this program consists of forming multiple sets by uniting the previous "seed" set that forms a "prime continuum" with every non-empty power set of the found magnitudes (instead of uniting the previous "seed" set that forms a "prime continuum" with every newly found magnitudes as in the previous program).

```
/* prime_number_algorithm_3.kt */

import kotlin.math.absoluteValue
import kotlin.math.pow
import java.math.BigInteger.valueOf

val MAX_ITERATIONS = 5
val MAX_EXPONENT = 3 // This maximum exponent is applied only to one
    subset.
val MAX_BOUND = 100000
var currentIterationIndex = 0
var currentBatchOfImmeasurables = mutableListOf<Long>() // The current
     list of immeasurables, which gets collected and cleared after
    every iteration.
var allImmeasurables = mutableListOf<Long>() // All immeasurables
    found.
var currentLadderStep = 0L // The biggest element used to calculate  C
     (the upper bound below which calculated immeasurables are prime).
val countMap = HashMap<Long, Int>() // A map used to count the number
    of times an immeasurable is found.
var exponentsMap = HashMap<Int, MutableList<MutableList<Int>>>() // A
    map that stores previously computed exponent variations.

fun main(args: Array<String>) {
    val seedImmeasurables = mutableListOf(1L, 2L)
```

```
    val seedImmeasurableString = getSetString(seedImmeasurables)
    println("Given the seed set $seedImmeasurableString")
    allImmeasurables.addAll(seedImmeasurables)
    currentLadderStep = seedImmeasurables.max()!!
    yieldImmeasurables(seedImmeasurables = seedImmeasurables,
    splitSwapIndex = 0)
}

fun getSetString(list: List<Any>) = list.toString().replace("[", "{").
    replace("]", "}")

fun <T> Collection<T>.powerset(): Set<Set<T>> {
    val one = valueOf(1L)
    var bigInt = valueOf(0L)
    val max = valueOf(2).pow(size)
    val powerset = mutableSetOf<Set<T>>()
    while (bigInt < max) {
        powerset.add(asSequence()
            .withIndex()
            .filter { (n, _) -> bigInt.testBit(n) }
            .map { it.value }
            .toSet())
        bigInt += one
    }
    return powerset
}

// A function that generates exponents in an incremental manner for a
    specified size of magnitudes that need to be raised to a power.
fun generateExponents(numberOfImmeasurableToBeRaised: Int):
    MutableList<MutableList<Int>> {
    val variations = mutableListOf<MutableList<Int>>()
    val currentVariation = MutableList(numberOfImmeasurableToBeRaised)
     { 1 }
    val lastVariationAccumulatedProduct = List(
    numberOfImmeasurableToBeRaised) { MAX_EXPONENT }.reduce {
    accumulatedProduct, i -> accumulatedProduct * i }
    var index = 0
    variations.add(currentVariation.toMutableList())
    while (currentVariation.reduce { accumulatedProduct, i ->
    accumulatedProduct * i } < lastVariationAccumulatedProduct) {
        while (currentVariation[numberOfImmeasurableToBeRaised - 1 -
    index] + 1 > MAX_EXPONENT) {
            if (numberOfImmeasurableToBeRaised - 1 - index > 0) {
                currentVariation[numberOfImmeasurableToBeRaised - 1 -
    index] = 1
            }
            ++index
        }
        if (index > numberOfImmeasurableToBeRaised - 1) {
            index = 0
        }
        currentVariation[numberOfImmeasurableToBeRaised - 1 - index]
    += 1
        index = 0
        variations.add(currentVariation.toMutableList())
    }
    return variations
```

```
}

fun isWithinBound(found: Long, upperBound: Long) = found < upperBound

fun addFoundImmeasurableToMap(found: Long) {
    if (countMap.containsKey(found)) {
        countMap[found] = countMap[found]!! + 1
    } else {
        countMap[found] = 1
    }
}

// The main function which recursively yields immeasurables by using a split index as a swap pivot to generate the two sets or lists of immeasurables.
// isLeaf, when true, marks a recursion terminating node
fun yieldImmeasurables(seedImmeasurables: MutableList<Long>, splitSwapIndex: Int, isLeaf: Boolean = false) {
    val sizeOfCurrentSeedImmeasurables = seedImmeasurables.size
    if (sizeOfCurrentSeedImmeasurables < 2 || currentIterationIndex >= MAX_ITERATIONS) {
        return
    }
    var swapCursor = splitSwapIndex + 1
    while (swapCursor < sizeOfCurrentSeedImmeasurables + 1) {
        val leftCollection = seedImmeasurables.subList(0, splitSwapIndex + 1)
        val rightCollection = seedImmeasurables.subList(splitSwapIndex + 1, sizeOfCurrentSeedImmeasurables)
        // primeFulcrum is the successive product of the set that is constrained to never be raised above the power of 1.
        val primeFulcrum = rightCollection.reduce { acc, i -> acc * i }
        // This generates exponents if not previously stored.
        val listOfExponents = if (exponentsMap.containsKey(leftCollection.size)) {
            exponentsMap[leftCollection.size]
        } else {
            exponentsMap[leftCollection.size] = generateExponents(numberOfImmeasurableToBeRaised = leftCollection.size)
            exponentsMap[leftCollection.size]
        }
        listOfExponents!!.forEach { exponentsList ->
            val raisedLeftCollection = leftCollection.mapIndexed { index, prime -> (prime.toDouble().pow(exponentsList[index].toDouble())).toInt() }.toMutableList()
            val leftRepeatedMultiplications = raisedLeftCollection.reduce { acc, i -> acc * i }
            if (leftRepeatedMultiplications < 0 || primeFulcrum < 0) {
                return@forEach
            }
            val firstPrime = leftRepeatedMultiplications + primeFulcrum
            val secondPrime = (leftRepeatedMultiplications - primeFulcrum).absoluteValue
            val currentUpperBound = currentLadderStep * currentLadderStep
            if (isWithinBound(firstPrime, currentUpperBound)) {
```

```
                currentBatchOfImmeasurables.add(firstPrime)
                addFoundImmeasurableToMap(firstPrime)
            }
            if (isWithinBound(secondPrime, currentUpperBound)) {
                currentBatchOfImmeasurables.add(secondPrime)
                addFoundImmeasurableToMap(secondPrime)
            }
        }
        val tempImmeasurables = mutableListOf<Long>()
        tempImmeasurables.addAll(seedImmeasurables)
        if (splitSwapIndex + 1 < sizeOfCurrentSeedImmeasurables / 2) {
            yieldImmeasurables(tempImmeasurables, splitSwapIndex + 1)
        }
        if (swapCursor < sizeOfCurrentSeedImmeasurables) {
            seedImmeasurables[splitSwapIndex] = seedImmeasurables[splitSwapIndex] xor seedImmeasurables[swapCursor]
            seedImmeasurables[swapCursor] = seedImmeasurables[splitSwapIndex] xor seedImmeasurables[swapCursor]
            seedImmeasurables[splitSwapIndex] = seedImmeasurables[splitSwapIndex] xor seedImmeasurables[swapCursor]
        } else if (swapCursor == sizeOfCurrentSeedImmeasurables) {
            if (splitSwapIndex == 0 && currentLadderStep < MAX_BOUND && !isLeaf) {
                currentIterationIndex++
                val foundImmeasurables = currentBatchOfImmeasurables.distinct().filter { it !in allImmeasurables }
                allImmeasurables.addAll(foundImmeasurables)
                currentLadderStep = allImmeasurables.filter { it > currentLadderStep }.min()!!
                currentBatchOfImmeasurables.clear()
                val powerSetOfFoundImmeasurables = foundImmeasurables.powerset()
                powerSetOfFoundImmeasurables.forEach {
                    val subPowerSetList = it.toMutableList()
                    val leafSeedImmeasurables = seedImmeasurables.distinct().toMutableList()
                    leafSeedImmeasurables.addAll(subPowerSetList)
                    yieldImmeasurables(leafSeedImmeasurables, 0, true)
                }
                val newSeedImmeasurables = seedImmeasurables.toMutableList()
                newSeedImmeasurables.add(currentLadderStep)
                println("At iteration $currentIterationIndex this program found the following immeasurables: ${getSetString(allImmeasurables.sorted())},")
                var primeCountString = allImmeasurables.sorted().filter { it != 2L }.map { prime ->
                    "$prime was found ${countMap[prime]} times"
                }.toString().replace("[", "").replace("]", "")
                val lastIndexOfComma = primeCountString.lastIndexOf(",")
                primeCountString = primeCountString.replaceRange(lastIndexOfComma + 1, lastIndexOfComma + 2, " and ")
                println("where $primeCountString.")
                yieldImmeasurables(newSeedImmeasurables, 0)
            }
        }
        swapCursor++
```

```
    }
}
```

It is worth noting that the maximum exponent is lowered to 3, yet the program exhaustively yields primes (up to the fifth iteration).

```
kotlinc prime_number_algorithm_3.kt -include-runtime -d
    prime_number_algorithm_3.jar
java -jar prime_number_algorithm_3.jar
Given the seed set {1, 2}
At iteration 1 this program found the following immeasurables: {1, 2,
    3},
where 1 was found 11 times, and 3 was found 10 times.
At iteration 2 this program found the following immeasurables: {1, 2,
    3, 5, 7},
where 1 was found 50 times, 3 was found 10 times, 5 was found 28 times
    , and 7 was found 34 times.
At iteration 3 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23},
where 1 was found 238 times, 3 was found 10 times, 5 was found 28
    times, 7 was found 305 times, 11 was found 145 times, 13 was found
     177 times, 17 was found 164 times, 19 was found 163 times, and 23
     was found 149 times.
At iteration 4 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47},
where 1 was found 402 times, 3 was found 10 times, 5 was found 28
    times, 7 was found 305 times, 11 was found 395 times, 13 was found
     355 times, 17 was found 306 times, 19 was found 293 times, 23 was
     found 479 times, 29 was found 406 times, 31 was found 403 times,
    37 was found 259 times, 41 was found 376 times, 43 was found 419
    times, and 47 was found 256 times.
At iteration 5 this program found the following immeasurables: {1, 2,
    3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
    67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113},
where 1 was found 402 times, 3 was found 10 times, 5 was found 28
    times, 7 was found 305 times, 11 was found 395 times, 13 was found
     394 times, 17 was found 332 times, 19 was found 333 times, 23 was
     found 479 times, 29 was found 432 times, 31 was found 469 times,
    37 was found 305 times, 41 was found 376 times, 43 was found 425
    times, 47 was found 302 times, 53 was found 168 times, 59 was
    found 162 times, 61 was found 97 times, 67 was found 206 times, 71
     was found 264 times, 73 was found 273 times, 79 was found 320
    times, 83 was found 117 times, 89 was found 61 times, 97 was found
     122 times, 101 was found 415 times, 103 was found 101 times, 107
    was found 405 times, 109 was found 283 times, and 113 was found
    157 times.
```

If concerned only with primes (in all three programs), one should use $\{2, 3\}$ as a starting set instead (`val seedImmeasurables = mutableListOf(2L, 3L)`) and update the function `isWithinBound` to exclude the unit.

References

[1] Thomas L. Heath, and Euclid. *The Thirteen Books of Euclid's Elements.*

Email address: bruno.ziki.kongawi@gmail.com