
UNSW
AUSTRALIA

# Searching for a counterexample to Kurepa's conjecture in average polynomial time

Ramanan Rajkumar

Supervisor: Dr. David Harvey

School of Mathematics and Statistics

UNSW Sydney

April 2019

Submitted in partial fulfilment of the requirements of the degree of

Masters by Research

UNSW SYDNEY | Australia's Global University

# Thesis/Dissertation Sheet

| | | |
|---|---|---|
| Surname/Family Name | : | Rajkumar |
| Given Name/s | : | Ramanan |
| Abbreviation for degree as give in the University calendar | : | MRes |
| Faculty | : | Science |
| School | : | Mathematics and Statistics |
| Thesis Title | : | Searching for a counterexample to Kurepa's conjecture in average polynomial time |

The left factorial of $n$ is defined to be $0!+1!+\dots+(n-1)!$ and is denoted by $!n$. Kurepa conjectured that $!n$ is not divisible by $n$ for $n>2$ and showed that it was sufficient to check the conjecture for odd primes $p$. We provide a survey of articles written on the search for a counterexample to Kurepa's conjecture and analyse the complexity of the algorithms used. These algorithms are all linear in $p$; that is, exponential in $\log p$. We develop the first known algorithm whose complexity is polynomial in $\log p$ when averaged over primes.

**COPYRIGHT STATEMENT**

'I hereby grant the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstract International (this is applicable to doctoral theses only).

I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.'

Signed .....

Date 04/21/19

**AUTHENTICITY STATEMENT**

'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Signed

Date 04/21/19

**ORIGINALITY STATEMENT**

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed .....

Date 04/21/19 .....

## INCLUSION OF PUBLICATIONS STATEMENT

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

**Publications can be used in their thesis in lieu of a Chapter if:**

- The student contributed greater than 50% of the content in the publication and is the "primary author", ie. the student was responsible primarily for the planning, execution and preparation of the work for publication
- The student has approval to include the publication in their thesis in lieu of a Chapter from their supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis

Please indicate whether this thesis contains published material or not.

☑ *This thesis contains no publications, either published or submitted for publication*

☐ *Some of the work described in this thesis has been published and it has been documented in the relevant Chapters with acknowledgement*

☐ *This thesis has publications (either published or submitted for publication) incorporated into it in lieu of a chapter and the details are presented below*

**CANDIDATE'S DECLARATION**

I declare that:

- I have complied with the Thesis Examination Procedure
- where I have used a publication in lieu of a Chapter, the listed publication(s) below meet(s) the requirements to be included in the thesis.

| Name | Signature | Date (dd/mm/yy) |
|---|---|---|
| Ramanan Rajkumar | | 04/21/19 |

# Plagiarism statement

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

By signing this declaration I am agreeing to the statements and conditions above.

Signed: _____ Date: _____

# Acknowledgements

First and foremost, I would like to thank my supervisor, David. Simply put, without him, this thesis would not exist.

I would like to thank the School of Mathematics and Statistics for their unbounded support and assistance. In particular, I would like to thank Professor Warton and Dr Potapov.

Special praise must be given to Raveen, who proofread my thesis and checked the maths line by line. Also thanks to Jeffrey, Patrick, Prosha and Melinda for your help with R.

Thanks must be given to all my family and friends. There are too many to name individually, but I am deeply indebted to you.

Ramanan Rajkumar, 31 August 2018.

# Abstract

The left factorial of $n$ is defined to be $0! + 1! + \cdots + (n-1)!$ and is denoted by $!n$. Kurepa conjectured that $!n$ is not divisible by $n$ for $n > 2$ and showed that it was sufficient to check the conjecture for odd primes $p$. We provide a survey of articles written on the search for a counterexample to Kurepa's conjecture and analyse the complexity of the algorithms used. These algorithms are all linear in $p$; that is, exponential in $\log p$. We develop the first known algorithm whose complexity is polynomial in $\log p$ when averaged over primes.

# Contents


Chapter 1 Previous results and methods 1

1.1 Kurepa's hypothesis . . . . . 1

1.2 Equivalent statements . . . . . 2

1.3 Attempted proofs . . . . . 3

1.4 The search for a counterexample . . . . . 4

Chapter 2 A class of recurrence problems 6

2.1 Definitions . . . . . 6

2.2 Examples . . . . . 7

2.2.1 Wilson primes . . . . . 7

2.2.2 Wolstenholme primes . . . . . 7

2.2.3 Kurepa's conjecture . . . . . 8

2.2.4 Central trinomial coefficient conjecture . . . . . 9

Chapter 3 Basic complexity results 10

3.1 Complexity of arithmetic operations for integers . . . . . 10

3.2 Complexity of arithmetic operations for integer matrices . . . . . 12

3.3 Complexity for calculating each example naively . . . . . 14

3.4 Naive complexity of calculating Wilson primes . . . . . 16

Chapter 4 Accumulating remainder tree 18

4.1 Product tree algorithm . . . . . 18

4.2 Complexity of the product tree algorithm . . . . . 19

4.3 Remainder tree algorithm . . . . . 24

4.4 Complexity of the remainder tree algorithm . . . . . 27
4.5 Accumulating remainder tree algorithm . . . . . 30
4.6 Complexity analysis of the accumulating remainder tree algorithm . 34
4.7 History of the accumulating remainder tree algorithm . . . . . 41

Chapter 5 A toy implementation 43

Chapter 6 Conclusion 45

# Chapter 1

# Previous results and methods

## 1.1 Kurepa's hypothesis

Kurepa defined the *left factorial* of $n$ as $\sum_{m=0}^{n-1} m!$ for non-negative integers $n$ in [Kur1971], where $0! := 1$. He denoted the left factorial of $n$ by $!n$. Other papers refer to $!p$ as $\kappa_p$ such as in [Ham1999] and $!n$ as $K(n)$ when considering the analytic continuation of $!n$ such as in [Mal2003]. We shall denote the smallest non-negative integer congruent to $!n$ mod $n$ by $r_n$.

The left factorial has uses in combinatorics, such as the number of permutations on $[n]$ that avoid the patterns $2n1$ and $n12$ where a $2n1$ pattern is a (scattered) subsequence $a - n - b$ with $a > b$. Further uses include counting the number of vertices of a certain sequence of recursively defined trees [PŽ1999]. More uses can be found in [OEIS, Sequence A003422].

Andrejić and Tatarevic defined the *generalised left factorial function* where $!^k n := \sum_{m=0}^{n-1} (m!)^k$ in [AT2014]. We note that $!^1 n = !n$.

The left factorial $!n$ satisfies the functional equation $f(n+1) - f(n) = \Gamma(n+1)$ for all $n$. This leads to the analytic continuation of $!n$ to the whole complex plane excluding $-1, -3, -4, -5, \ldots$ and is denoted by $K(z)$. Further information can be found in [Mal2003] but these results are outside the scope of this thesis. Studying $K(z)$ leads to a different generalisation in the complex plane; further details were provided by Milovanović in [Mil1996], Milovanović and Petojević [MP2002] and Petojević in [Pet2002]. These results are outside of the scope of this thesis.

It is shown in [Kur1971] that $!n \not\equiv 0 \bmod n$ for infinitely many $n$. The major question asked is if this is true for all $n$.

**Conjecture 1.1.1** (Kurepa's left factorial hypothesis)**.** *Let $n$ be a positive integer such that $n > 2$. Then $!n \not\equiv 0 \bmod n$.*

This conjecture is sometimes referred to as Kurepa's hypothesis in the literature. Kurepa then gives two equivalent statements to his conjecture in [Kur1971]. He shows that to prove $!n \not\equiv 0 \bmod n$ for every positive integer $n > 2$, it is necessary and sufficient to prove that $!p \not\equiv 0 \bmod p$ for every odd prime $p$. The greatest common divisor of $!n$ and $n!$ is denoted by $M_n$; that is, $M_n := \gcd(!n, n!)$. Then the statement that $M_n = 2$ for all $n > 1$ is also equivalent to Kurepa's hypothesis (Conjecture 1.1.1) as shown by Kurepa in [Kur1971].

Andrejić and Tatarevic asked whether the natural analogue of Kurepa's hypothesis (Conjecture 1.1.1) holds for the generalised left factorial function. They showed that for every $k = 2, 3, \ldots, 99$, there is an odd prime $p$ such that $p$ divides $!^k p$ so the natural generalisation of Kurepa's hypothesis is false [AT2014].

## 1.2 Equivalent statements

There are many equivalent statements of Kurepa's hypothesis (Conjecture 1.1.1); Mijajlović showed in [Mij1990] that

$$!p \equiv \sum_{k=0}^{p-1} (-1)^{k+1}/k! \equiv \sum_{k=0}^{p-1} (-1)^k (k+1)(k+2)\cdots(p-1) \bmod p.$$

We note that

$$D_n := n! \sum_{k=0}^{n-1} (-1)^{k+1}/k!$$

is the sequence of *derangement numbers*. For more details on derangement numbers, we direct the reader to [Has2003]. Hence Kurepa's hypothesis is equivalent to

$$D_{p-1} \not\equiv 0 \bmod p$$

for all odd primes $p$ [Meš2015].

Mijajlović also gave a recursive formula in [Mij1990]. Let $p$ be a prime and let $s_{p-1} := 0$ in $\mathbb{F}_p$. Define the recursion $s_i := 1 + is_{i+1}$ in $\mathbb{F}_p$ for $i = p-2, p-3, \ldots, 1$. Then $r_p = s_1$ and this recurrence was used to verify Kurepa's conjecture for all primes up to 311009. Meštrović gathered a summary of equivalent statements of Kurepa's conjecture in [Meš2015]; see Kellner in [Kel2004], Ivić and Mijajlović in [IM2004], Petojević, Žižović and Stana D. Cvejić in [PŽC1999], Šami in [Šam1974] and Živković in [Živ1999], Stanković in [Sta1973] and Stanković and Žižović in [SŽ1974]. Most of these equivalent statements involve recurrence relations or summation formulae. Other equivalent statements can be found in [GR2014] using Bell numbers and linear algebra. Some of the statements equivalent to Kurepa's conjecture involve recurrence relations, such as in [Mij1990], and are used to search for counterexamples using a computer such as in [AT2014].

## 1.3 Attempted proofs

We define the $n$-th *Bell number* as the number of partitions of a set of size $n$ and denote the $n$-th Bell number by $B_n$; it is denoted by $D_n$ in [Ham1999] and $P_n$ in [BB2004]. Bell numbers have many uses in combinatorics: for more information, we refer the reader to [Bel1934]. Hamadene proved that $!p \equiv B_{p-1} - 1 \bmod p$ for primes $p$ [Ham1999]. Barsky and Benzaghou gave a formula for $B_n \bmod p$ in terms of the $n$-th power of the trace of a fixed element in the Artin-Schreier extension of $\mathbb{F}_p$. Hence they reduced Kurepa's conjecture to a linear algebra problem and claimed to have solved Kurepa's conjecture [BB2004]. However it was pointed out that there was a mistake in the calculations and the proof of Kurepa's hypothesis (Conjecture 1.1.1) was retracted due to 'irreparable calculation errors' in solving the linear algebra problem; the equivalence of Kurepa's conjecture and the linear algebra problem was still valid [BB2011].

Reg. Bond proposed a proof which was unpublished but is mentioned by [Guy]. Živković mentions that Bond communicated that there was an error in a personal email [Živ1999] which we have not personally verified.

Currently, Kurepa's hypothesis (Conjecture 1.1.1) has not been disproven. However it is heuristically conjectured to be false; under certain assumptions regarding the uniform distribution of $r_p$, the 'probability' of Kurepa's conjecture holding is 0. Furthermore, it is suggested that under these assumptions there are infinitely many counterexamples and the probability of finding a counterexample in the interval $(x, x^\alpha]$ is approximately $1 - 1/\alpha$. Hence the probability of finding a counterexample in $(x, x^2]$ is $1/2$. The probability of finding a counterexample in $(2^n, 2^{n+1}]$ is $1/(n+1)$. Two types of chi-square tests were carried out on the uniformity of $r_p$ for primes in $(2^h, 2^{h+1})$ where $h = 10, 11, \ldots, 23$. These tests did not contradict the assumptions of the uniformity of $r_p$ which provides evidence that the assumptions are valid. An exposition of the heuristics is presented in [Živ1999].

## 1.4 The search for a counterexample

In Problem B44 of [Guy], Guy claims that Slavic checked up to 1000 using a computer and Wagstaff checked up to 50000 though we are unable to verify this ourselves. Mijajlovic checked Kurepa's conjecture for all primes $p \leq 311009 \approx 3.1 \cdot 10^5$ in [Mij1990]. The number of arithmetic operations required to verify Kurepa's hypothesis (Conjecture 1.1.1) for each $p$ is $4p$ and is therefore exponential in $\log p$. Hence the time complexity is exponential in $\log p$. Gogić checked all primes up to $p \leq 10^6$ in [Gog1991] but we were unable to obtain a copy of [Gog1991]. Živković checked for all primes less than $2^{23} \approx 8.4 \cdot 10^6$ in [Živ1999]. It is claimed that Malešivić verified Kurepa's conjecture for all primes $p \leq 3 \cdot 10^6$ in a personal communication [Živ1999] but we have been unable to verify this claim. Gallot checked up to $2^{26} \approx 6.7 \cdot 10^7$ in [Gal2000] using an algorithm with a running time which is exponential in $\log p$. Jobling claimed to have checked up to $2^{27} \approx 1.4 \cdot 10^8$ in [OEIS, Sequence A049782] and this is quoted in [AT2014] but we have been unable to verify this claim. Ilijašević verified Kurepa's conjecture up to $2^{31} \approx 2 \cdot 10^9$ in [Ili2015] using an algorithm with time complexity which is exponential in $\log p$. Andrejić and Tatarevic checked up to $p < 2^{34} \approx 1.7 \cdot 10^{10}$ and the time complexity is exponential in $\log p$ [AT2014]. They also claim that the fastest known

algorithm to verify Kurepa's hypothesis (Conjecture 1.1.1) for all primes up to $N$ is $O(N^2/\log N)$. It follows that the fastest known algorithm has an amortised average running time which is exponential in $\log N$. This means that for large $N$, verifying Kurepa's conjecture for all primes $p$ less than $N$ is not feasible using current algorithms. In Chapter 4, we will give an algorithm which has an amortised average running time which is polynomial in $\log N$.

# Chapter 2

# A class of recurrence problems

In this section, we show how a certain class of problems can be expressed in terms of recurrence relations. These problems all involve calculating some integer matrices $C_1, C_2, \ldots, C_N$.

## 2.1 Definitions

In this subsection, we give the framework required to define the recurrence relations.

Let $A = (a_{ij})_{ij}$ be a matrix with integer entries and let $m$ be a positive integer. Then we define

$$A \bmod m := (a_{ij} \bmod m)_{ij}.$$

When we write $a \bmod m$, we shall always choose the representative of the equivalence class which is between 0 and $m-1$ unless stated otherwise.

Suppose that $A_1, \ldots, A_{N-1}$ is a sequence of $d \times d$ matrices with integer entries and that $V \in \mathbb{Z}^d$ . Furthermore, we also assume that $m_n = n^\lambda$, for a fixed $\lambda$, if $n$ is prime and 1 otherwise. We define $A_0$ to be the $d \times d$ identity matrix and $m_0$ to be 1 to simplify calculations in Section 4.2. Our goal is to calculate the sequence of matrices $C_1, \ldots, C_N$ where

$$
\begin{aligned}
C_1 &:= A_0 V \bmod 1, \\
C_2 &:= A_1 A_0 V \bmod 2^\lambda, \\
C_3 &:= A_2 A_1 A_0 V \bmod 3^\lambda, \\
C_4 &:= A_3 A_2 A_1 A_0 V \bmod 1, \\
C_5 &:= A_4 A_3 A_2 A_1 A_0 V \bmod 5^\lambda, \\
&\vdots \\
C_N &:= A_{N-1} \cdots A_1 A_0 V \bmod m_N.
\end{aligned}
$$

**Remark 2.1.1.** We remark that $A_n$ can be generalised to be any object with an associative multiplication structure and $m_n$ can be any object such that division with remainder of $A_{n-1} \cdots A_0 V$ by $m_n$ is defined.

## 2.2 Examples

In this subsection, we give examples of some problems which can be written in the above formulation.

### *2.2.1 Wilson primes*

Our first example is to compute the set of Wilson primes up to some bound $N$. We remind the reader that Wilson's theorem states that $(p-1)! + 1 \equiv 0 \bmod p$ for all primes $p$ [Lag1771]. A Wilson prime is defined to be a prime $p$ that satisfies the equation $(p-1)! + 1 \equiv 0 \bmod p^2$. The only known Wilson primes less than $2 \times 10^{13}$ are 5, 13, and 563 [CGH2014] but it is conjectured that there are infinitely many Wilson primes [CGH2014]. Our object of interest is $(p-1)! \bmod p^2$ for every prime $p < N$. Hence we choose $d := 1$, $A_n := n$, $V := 1$, and $\lambda := 2$.

### *2.2.2 Wolstenholme primes*

A similar example to the search for Wilson primes is to compute the set of Wolstenholme primes up to some bound $N$. Wolstenholme's theorem states that $\binom{2p-1}{p-1} \equiv 1 \bmod p^3$ for all primes $p > 3$ [Wol1862]. A Wolstenholme prime is a prime $p$ such that $\binom{2p-1}{p-1} \equiv 1 \bmod p^4$ [McI1995]. The only known Wolstenholme primes less than $10^9$ are 16843 and 2124679 [MR2007] but it is conjectured that there are infinitely

many Wolstenholme primes [McI1995]. Hence our goal is to compute $\binom{2p-1}{p-1} \bmod p^4$. We note that

$$\binom{2n-1}{n-1} = \frac{2(2n-1)}{n}\binom{2n-3}{n-2}$$

for $n > 1$ but this is not an integer recurrence. To avoid this issue, we calculate the numerator and denominator separately. For the numerator, we let $d := 1$, $A_n := 2(2n+1)$, $V := 1$, and $\lambda := 5$. Then we calculate $A_{p-1} \cdots A_1 A_0 V \bmod m_p$. Now we calculate the denominator. We let $d' := 1$, $A'_n := n+1$, $V' = 1$ and $\lambda' := 5$. Then we calculate $A'_{p-1} \cdots A'_0 V' \bmod m_p$. We write $p \parallel n$ if $p$ divides $n$ and $p^2$ does not divide $n$. We note that $p \parallel A'_{p-1} \cdots A'_0 V'$ so $A'_{p-1} \cdots A'_0 V'/p$ is invertible in $\mathbb{Z}/p^4\mathbb{Z}$. Thus we may calculate

$$\begin{aligned}
\binom{2p-1}{p-1} \bmod p^4 &= C_p/C'_p \bmod p^4 \\
&= A_{p-1} \cdots A_0 V / A'_{p-1} \cdots A'_0 V' \bmod p^4 \\
&= \left(A_{p-1} \cdots A_0 V/p\right) / \left(A'_{p-1} \cdots A'_0 V'/p\right) \bmod p^4
\end{aligned}$$

as claimed.

### *2.2.3 Kurepa's conjecture*

A less trivial example is the verification of Kurepa's conjecture up to some bound $N$. As shown by Kurepa in [Kur1971], an equivalent formulation of Kurepa's hypothesis (Conjecture 1.1.1) is that $!p \not\equiv 0 \bmod p$ for all odd primes $p$. Thus our object of interest is $!n \bmod n$ which requires $2 \times 2$ matrices to form a recurrence relationship. For this recurrence relation, we let

$$d := 2, A_0 := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, A_n := \begin{pmatrix} n & 0 \\ n & 1 \end{pmatrix}, V := \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \lambda := 1$$

for $n \geq 1$. Thus

$$A_{p-1} A_{p-2} \cdots A_1 A_0 V \equiv \begin{pmatrix} (p-1)! \\ !p \end{pmatrix} \bmod p$$

and from this, we can read off $!p \bmod p$.

### *2.2.4 Central trinomial coefficient conjecture*

Another example is the verification of the central trinomial coefficient conjecture up to some bound $N$. We define $T(n)$ as the coefficient of $x^n$ in the expansion of $(1+x+x^2)^n$. It is conjectured that an integer $n > 3$ is prime if and only if $T(n) \equiv 1 \bmod n^2$ [Sun]. We would like to verify that if $n$ is prime, then $T(n) \equiv 1 \bmod n^2$. Therefore our object of interest is $T(p) \bmod p^2$. It is known that $T(n)$ satisfies the recurrence

$$T(n) = \frac{(2n-1)T(n-1) + 3(n-1)T(n-2)}{n}$$

as proven in [Eul1765]. This can be written using matrices as

$$\begin{pmatrix} T(n) \\ T(n-1) \end{pmatrix} = \frac{1}{n} \begin{pmatrix} 2n-1 & 3(n-1) \\ n & 0 \end{pmatrix} \begin{pmatrix} T(n-1) \\ T(n-2) \end{pmatrix}.$$

Similar to the search for Wolstenholme primes in Section 2.2.2, we do not have an integer recurrence. Thus we will calculate the numerators and denominators separately.

For the numerator, we let $d := 2$, $A_n := \begin{pmatrix} 2n+1 & 3n \\ n+1 & 0 \end{pmatrix}$, $V := \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $\lambda := 3$. Then we calculate $A_{p-1} \cdots A_1 A_0 V \bmod m_p$. Now we calculate the denominator. We let $d' := 1$, $A'_n := n+1$, $V' = 1$ and $\lambda' := 3$. Then we calculate $A'_{p-1} \cdots A'_1 A'_0 V' \bmod m_p$. We note that $p \parallel A'_{p-1} \cdots A'_1 A'_0 V'$ so $A'_{p-1} \cdots A'_1 A'_0 V'/p$ is invertible in $\mathbb{Z}/p^2\mathbb{Z}$. Thus we may calculate

$$\begin{aligned} \begin{pmatrix} T(p) \\ T(p-1) \end{pmatrix} \bmod p^2 &= C_p / C'_p \bmod p^2 \\ &= A_{p-1} \cdots A_1 A_0 V / A'_{p-1} \cdots A'_1 A'_0 V' \bmod p^2 \\ &= \left(A_{p-1} \cdots A_1 A_0 V/p\right) / \left(A'_{p-1} \cdots A'_1 A'_0 V'/p\right) \bmod p^2 \end{aligned}$$

and read off $T(p) \bmod p^2$ as required.

# Chapter 3

# Basic complexity results

## 3.1 Complexity of arithmetic operations for integers

In this section, we give basic results on the complexity of binary operations using integers. Let $B$ be a positive real number such that $B \geq 1$. Throughout this section, we assume that $m$ and $n$ are integers which are at most $\lceil B \rceil$ bits. Then the time complexity of adding $m$ and $n$ (or subtracting $m$ from $n$) is $O(B)$ and the space complexity is $O(B)$ [vZGG1999].

We denote $\mathsf{M}(B)$ as an upper bound for the time complexity of multiplying two integers which are at most $\lceil B \rceil$ bits. We assume that $\mathsf{M}(B)/B$ is increasing and $\mathsf{M}$ is superlinear [vZGG1999, Ch.8]; that is, $\mathsf{M}(A+B) \geq \mathsf{M}(A) + \mathsf{M}(B)$ for $A, B \geq 1$. These hypotheses guarantee that the running time of division with remainder of two $\lceil B \rceil$-bit integers is $O(\mathsf{M}(B))$ [vZGG1999, Ch.9]. We note that although multiplication and division with remainder have the same asymptotic growth, the constant in the division algorithm is larger. For further details, we refer the reader to [vZGG1999, Ch.8,9].

We give some examples of the time complexity of multiplication algorithms. Classical multiplication has time complexity $O(B^2)$. The Schönhage–Strassen algorithm has time complexity $O(B \log(B) \log\log(B))$ [vZGG1999, Thm 8.24]. The fastest known method runs in $O(B \log(B) 4^{\log^*(B)})$ time [HvdH2018], where $\log^*$ is the iterated logarithm function. The space complexity of multiplying $m$ and $n$ and dividing $m$ by $n$ with remainder is $O(B)$ for all of these algorithms, so throughout

this thesis, we shall assume that the space complexity of multiplying $m$ and $n$ is $O(B)$.

Throughout this thesis, we define $\lg(n)$ as $\log_2(n)$ for positive integers $n$ and 0 if $n = 0$. The space required to store the positive integer $n$ is $\lceil \lg(n) \rceil + 1 + C$ where $C$ is the overhead used to store pointers and memory allocation data. Now we introduce a function $\beta$ which approximates the space required to store different objects. This is a purely theoretical measure and is independent of the computer and the language. We define $\beta(n)$ as $\lg(|n|)$ if $n$ is a non-zero integer such that $|n| > 1$ and 1 if $n = -1, 0, 1$.

Now $\beta$ has some useful properties, such as the fact that it is *sublogarithmic*; that is, $\beta(mn) \leq \beta(m) + \beta(n)$. Inductively, this implies that $\beta(m^k) \leq k\beta(m)$ which provides a very useful bound which will be utilised throughout this thesis.

We can consider the integers to be $1 \times 1$ matrices. We would like to define $\beta$ for arbitrarily sized $d \times d$ matrices with integer entries such that $\beta$ is still sublogarithmic. We first suppose that $X$ is a $d \times d$ matrix with integer coefficients. Then we define

$$\beta(X) := \lg(\max_{1 \leq j \leq d}(\sum_{i=1}^{d} |a_{ij}|)),$$

where $\max_{1 \leq j \leq d}(\sum_{i=1}^{d} |a_{ij}|)$ is the maximum column sum. This agrees with the definition of $\beta$ for integers if $d = 1$.

We remark that $\beta(X) = \lg(\|X\|_1)$ where $\|\cdot\|_1$ is the operator 1 norm [FRR1987, Prop. 2.1.i].

Now we show that $\beta$ is still sublogarithmic.

**Lemma 3.1.1.** *Let $X$ and $Y$ be $d \times d$ matrices with integer entries. Then $\beta(XY) \leq \beta(X) + \beta(Y)$.*

*Proof.* Let $X$ and $Y$ be $d \times d$ matrices with integer coefficients. It follows that

$$\begin{aligned}\beta(XY) &= \lg(\|XY\|_1) \\ &\leq \lg(\|X\|_1\|Y\|_1) \\ &\leq \lg(\|X\|_1) + \lg(\|Y\|_1) \\ &\leq \beta(X) + \beta(Y)\end{aligned}$$

where the first inequality holds since $\|\cdot\|_1$ is submultiplicative. Thus $\beta(XY) \leq \beta(X) + \beta(Y)$ as required. □

## 3.2 Complexity of arithmetic operations for integer matrices

We begin this subsection with some introductory theorems about the time complexity of arithmetic operations for integer matrices.

Throughout this thesis, we will use the definition of matrix multiplication to multiply matrices instead of using faster algorithms such as Strassen's algorithm [Str1969], the Coppersmith–Winograd algorithm [CW1990] or Le Gall's algorithm [LG2014]. This is purely to simplify the analysis of the time and space complexity bounds since the benefits from using faster matrix multiplication algorithms are minimal.

Note that throughout this thesis, we shall ignore the dependency on $d$ as the applications that follow only deal with $d \leq 3$.

**Theorem 3.2.1.** *Suppose $X$ and $Y$ are $d \times d$ matrices with integer coefficients such that $\beta(X), \beta(Y) \leq B$. Then the time complexity of computing $XY$ is $O(\mathsf{M}(B))$ and the space complexity of computing $XY$ is $O(B)$.*

*Proof.* Let $X = (x_{ij})_{ij}$ and $Y = (y_{ij})_{ij}$. Denote $XY$ as $Z = (z_{ij})_{ij}$. We observe that $\lg(|x_{ij}|), \lg(|y_{ij}|) \leq B$ since $\beta(X), \beta(Y) \leq B$. Hence $x_{ij}, y_{ij}$ are $\lceil B \rceil$-bit integers.

By definition, $z_{ij} = \sum_{k=1}^{d} x_{ik}y_{kj}$. We note that computing each $x_{ik}y_{kj}$ requires $O(\mathsf{M}(B))$ time and $O(B)$ bits. The time complexity of computing $z_{ij}$ is $O(\mathsf{M}(B))$ since addition of integers which are at most $dB$ bits is $O(dB) = O(\mathsf{M}(B))$. Likewise

the memory required is $O(dB)$. The running time is $O(d^2\mathsf{M}(B))$ and the total memory required is $O(d^3B)$ as there are $d^2$ entries in $Z$. Hence the time and space complexity of computing $XY$ is $O(\mathsf{M}(B))$ and $O(B)$ respectively as $d$ is fixed.

□

Theorem 3.2.1 links bounds on the memory required to store matrices $X, Y$ to an upper bound on the time required to compute $XY$ and is therefore a very powerful result for time complexity analysis.

**Proposition 3.2.2.** *Suppose $X$ is a $d \times d$ matrix with integer entries and $n$ is a positive integer. Then the time complexity of computing $X \bmod n$ is*

$$O(\mathsf{M}(\max(\beta(X), \beta(n)))).$$

*Proof.* Let $X = (x_{ij})_{ij}$. We note that $x_{ij}$ is a $\beta(x_{ij})$-bit integer and that $n$ is a $\beta(n)$-bit integer. The time complexity of computing $x_{ij} \bmod n$ is

$$O(\mathsf{M}(\max(\beta(x_{ij}), \beta(n)))).$$

But each $\beta(x_{ij})$ is bounded above by $\beta(X)$ so it follows that

$$O(\mathsf{M}(\max(\beta(x_{ij}), \beta(n)))) = O(\mathsf{M}(\max(\beta(X), \beta(n)))).$$

Thus the total time complexity is

$$O(d^2\mathsf{M}(\max(\beta(X), \beta(n)))) = O(\mathsf{M}(\max(\beta(X), \beta(n))))$$

as claimed. □

**Proposition 3.2.3.** *Suppose $X$ is a $d \times k$ matrix with integer entries and that $n$ is a positive integer. Then $\beta(X \bmod n) \leq \lg d + \beta(n)$.*

*Proof.* We let $X = (x_{ij})_{ij}$ and note that $x_{ij} \bmod n$ is some non-negative integer less than $n$ for every $i$ and $j$. Thus the maximum column sum is at most $dn$ so it follows that

$$\beta(X \bmod n) \leq \lg d + \lg n \leq \lg d + \beta(n).$$

□

We remark that this bound is independent of $k$ since we are taking the maximum column sum. We note that the $\beta(n)$ term dominates for large $n$ since $d$ is fixed.

## 3.3 Complexity for calculating each example naively

We recall from Section 2.1 that $C_n := A_{n-1}A_{n-2}\cdots A_1A_0V \bmod m_n$ where each $A_k$ is a $d \times d$ matrix with positive integer entries and each $m_k = k^\lambda$ if $k$ is prime and 1 otherwise. We now define a naive algorithm for calculating each $C_p$ individually and analyse its complexity.

We remind the reader that the dependence on $d$ will not be studied and $d$ is assumed to be fixed. Then we let $A_0, \ldots, A_{N-1}, V$ and $m_0, m_1, \ldots, m_{N-1}$ be as in Section 2.1 and give a naive algorithm for calculating $C_p$ for every prime $p < N$. We calculate the time complexity of this algorithm under some general assumptions. Finally, we calculate the time complexity of applying this algorithm to the Wilson prime problem in Section 2.2.1.

**Algorithm 1** Naïve Algorithm for calculating $C_p$

**Input:** $V, p^\lambda, A_1, \cdots, A_{p-1}$

**Output:** $C_p$

1: $X \leftarrow V \bmod p^\lambda$
2: **for** $k = 0$ to $p - 1$ **do**
3: $\quad X \leftarrow A_kX$
4: $\quad X \leftarrow X \bmod p^\lambda$
5: **end for**
6: **return** $X$

We define $\tau$ as the maximum of $\beta(A_1), \ldots, \beta(A_N)$. From this point onwards in the thesis, we shall assume that $\tau = O(\log N)$. Similarly, we define $\sigma$ as the maximum of $\beta(m_1), \ldots, \beta(m_N)$. It follows that $\sigma = O(\log N)$. The examples in Section 2.2 all satisfy these assumptions.

We now remind the reader of the Prime Number Theorem and one of its corollaries. This corollary will be used to simplify some of the bounds obtained.

**Theorem 3.3.1** (Prime Number Theorem)**.** *Suppose that $\pi(x)$ counts the number of primes $p \leq x$. Then*

$$\pi(x) \sim x/\log x$$

*where $f \sim g$ if $\lim_{x\to\infty} f(x)/g(x) = 1$.*

*Proof.* A short exposition of the proof can be found in [New1980] and [Zag1997]. □

**Corollary 3.3.2.** *It follows that $\pi(x) = O(x/\log x)$.*

Now we bound the running time of calculating each $C_p$ naïvely using Algorithm 1.

**Theorem 3.3.3.** *The running time of computing $C_p$ for each prime $p$ up to some $N$ using Algorithm 1 is $O(N^{2+\varepsilon})$ where $\varepsilon \to 0$ as $N \to \infty$.*

*Proof.* First we calculate the time complexity of the multiplication in Line 3. We note that $X$ has been reduced modulo $p^\lambda$ so

$$\beta(X) = O(\beta(p^\lambda)) = O(\log N).$$

Also, we recall that $\beta(A_k) = O(\log N)$. Thus the time complexity of computing $A_k X$ is $O(\mathsf{M}(\log N))$.

Now we calculate the time complexity of the modular arithmetic in Line 4. We note that

$$\begin{aligned}\beta(A_k X) &\leq \beta(A_k) + \beta(X) \\ &\leq \tau + \lg d + \sigma\end{aligned}$$

where the second inequality follows from Theorem 3.2.3. It follows that

$$\beta(A_k X), \beta(p^\lambda) \leq \tau + \lg d + \sigma.$$

Thus the time complexity of the modular arithmetic is

$$O(\mathsf{M}(\tau + \lg d + \sigma)) = O(\mathsf{M}(\log N))$$

since $\beta(p^\lambda) \leq \sigma$. Hence the total time to complete each iteration of the loop is $O(\mathsf{M}(\log N))$ and there are $p = O(N)$ iterations so each $C_p$ takes $O(N\mathsf{M}(\log N))$ time.

Furthermore, the time complexity of calculating $C_p$ for all prime $p$ up to some $N$ using the naive algorithm above is $O(N\pi(N)\mathsf{M}(\log N)) = O(N^{2+\varepsilon})$ using Corollary 3.3.2.

□

The time complexity of calculating each $C_p$ is $O(N^{1+\varepsilon})$. Thus the time complexity of this algorithm is exponential in $\log N$. Therefore this algorithm is too slow for large $N$ to be useful for practical purposes. Hence we need a more efficient algorithm which will be introduced in Section 4.5.

## 3.4 Naive complexity of calculating Wilson primes

We now give an example to illustrate the time complexity of using the naive algorithm to compute the set of Wilson primes less than $N$ as in Example 2.2.1. We note that this algorithm does not use the fastest subroutine for calculating $n!$ but it is one of the simplest; other methods involving calculating the prime factors such as in [Bor1985] are superior. We recall that $d := 1$, $A_n := n$, $V := 1$, and $\lambda := 2$. We note that $\beta(A_p) = \lg p$, $\beta(V) = 1$, and that $\beta(m_p) = 2\lg p$.

First we calculate the time complexity of the multiplication. We note that $\beta(k) \leq \lg p$ and that $\beta((k-1)! \bmod p^2) \leq 2\lg p$. Hence the time complexity of calculating $k((k-1)! \bmod p^2)$ is $O(\mathsf{M}(2\lg p)) = O(\mathsf{M}(\log N))$.

Next, we calculate the time complexity of the modular arithmetic. From the analysis of the time complexity of the multiplication, we see that

$$\begin{aligned}\beta(k((k-1)! \bmod p^2)) &\leq \beta(k) + \beta((k-1)! \bmod p^2) \\ &\leq 3 \lg p.\end{aligned}$$

We note that $\beta(p^2) \leq 3 \lg p$ so $3 \lg p$ is an upper bound for $\beta(p^2)$, $\beta(k(k-1)! \bmod p)$. Hence the time complexity of the modular arithmetic is

$$O(\mathsf{M}(3 \lg p)) = O(\mathsf{M}(\log N))).$$

Thus the time complexity of each iteration of the loop is $O(\mathsf{M}(\log N))$. There are $p$ loops and $p \leq N$ so the time complexity of $p$ loop iterations is $O(N\mathsf{M}(\log N))$. Hence the time complexity of the entire loop is $O(N\mathsf{M}(\log N))$. Thus the time required to calculate $C_p$ is $O(N\mathsf{M}(\log N))$. Therefore the time complexity of calculating $C_p$ for all primes $p$ less than $N$ is

$$O(N\pi(N)\mathsf{M}(\log N)) = O(N^{2+\varepsilon})$$

as claimed.

# Chapter 4

# Accumulating remainder tree

We recall from Section 2.1 that $C_n := A_{n-1}A_{n-2}\cdots A_1A_0V \bmod m_n$ where each $A_k$ is a $d \times d$ matrix with positive integer entries and each $m_k = k^\lambda$ if $k$ is prime and 1 otherwise.

Throughout this section, we assume that $N$ is a power of 2. In order to calculate $C_n$ for all $n$ up to $N$ we introduce the product tree, remainder tree and accumulating remainder tree algorithms. We analyse the complexity of the accumulating remainder tree algorithm and compare the running time with the running time of the naive algorithm (Algorithm 1).

## 4.1 Product tree algorithm

The *product tree* takes as inputs two positive integers $a$ and $b$ such that $b - a = 2^h$ and a sequence of matrices $X_a, X_{a+1}, \ldots, X_{b-1}$.

Throughout this chapter, $X_{a,b}$ is defined as $X_{b-1}X_{b-2}\cdots X_a$.

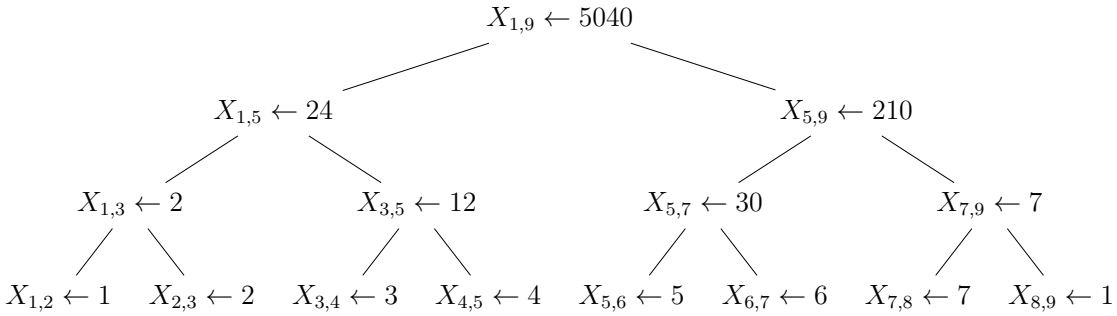
This algorithm recursively calculates $X_{a,b}$ by 'halving the interval'. If $b = a+1$, then $X_{a,b} = X_a$. Otherwise, we let $c := \lfloor (a+b)/2 \rfloor$ and calculate $X_{a,b}$ recursively using the relationship that $X_{a,b} = X_{c,b}X_{a,c}$. The outputs are all such $X_{a',b'}$ which were calculated in the product tree.

We now give a simple example of the product tree algorithm. Suppose we would like to compute 7! using the product tree algorithm. Let $X_n := n$ for $n = 1, \ldots, 7$. Thus $a := 1$. We need $b - a$ to be a power of 2 so we round up to the nearest power of 2 and let $b := 9$. Thus we define $X_8 := 1$. This is similar to

the technique of zeropadding. For further details about zeropadding, we refer the reader to [Ber2008]. Then

$$\begin{aligned} X_{1,9} &= X_{5,9}X_{1,5} \\ &= X_{7,9}X_{5,7}X_{3,5}X_{1,3} \\ &= X_{8,9}X_{7,8}X_{6,7}X_{5,6}X_{4,5}X_{3,4}X_{2,3}X_{1,2} \\ &= 1 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \\ &= 5040. \end{aligned}$$

This can be represented using a graph:




## 4.2 Complexity of the product tree algorithm

In this section we give general bounds for the time and space complexity of the product tree algorithm and then give specific bounds for the case where $X_n = A_n$ and $X_n = m_n$ where $A_n$ and $m_n$ are defined as in Section 2.1.

**Theorem 4.2.1.** *Suppose that $X_0, \ldots, X_{N-1}$ are $d \times d$ matrices with integer entries such that $\beta(X_n) \leq B_n$ for all $n$. Then the time complexity of computing the product tree for $X_0, X_1, \ldots, X_{N-1}$ is*

$$O\left(\mathsf{M}\left(\sum_{n=0}^{N-1} B_n\right) \log N\right).$$

*Proof.* We remind the reader that $N = 2^h$ for some non-negative integer $h$. First we consider the time required to compute the first level. To compute $X_{0,2^h}$, we first

compute $X_{0,2^{h-1}}$ and $X_{2^{h-1},2^h}$ and then multiply these to calculate $X_{0,2^h}$. We apply this recursively until we must calculate the pairs

$$X_0X_1, X_2X_3, \ldots, X_{2^h-2}X_{2^h-1}.$$

Now we consider the time complexity of calculating the $j$-th level. Define $w := 2^{h-j}$. This involves calculating

$$X_{0,w} \cdot X_{w,2w}, X_{2w,3w} \cdot X_{3w,4w}, \ldots, X_{(2^{j+1}-2)w,(2^{j+1}-1)w} \cdot X_{(2^{j+1}-1)w,2^{j+1}w}.$$

We bound the time complexity of calculating the product of $X_{0,w}$ and $X_{w,2w}$. First, we note that

$$\begin{aligned}\beta(X_{0,w}) &\leq \sum_{n=0}^{w-1} \beta(X_n) \\ &\leq \sum_{n=0}^{w-1} B_n\end{aligned}$$

using the fact that $\beta$ is sublogarithmic and the assumption that $\beta(X_n) \leq B_n$ for all $n$. Similarly,

$$\beta(X_{w,2w}) \leq \sum_{n=w}^{2w-1} B_n$$

so the product of $X_{0,w}$ and $X_{w,2w}$ can be computed in

$$\begin{aligned}&O\left(\mathsf{M}\left(\max\left(\sum_{n=0}^{w-1} B_n, \sum_{n=w}^{2w-1} B_n\right)\right)\right) \\ &= O\left(\mathsf{M}\left(\sum_{n=0}^{2w-1} B_n\right)\right)\end{aligned}$$

time using Theorem 3.2.1. Similar bounds can be found for the $2^j$ pairs of subproducts so the total time complexity of computing the $j$-th level is

$$\sum_{k=0}^{2^j-1} O\left(\mathsf{M}\left(\sum_{n=2kw}^{2(k+1)w-1} B_n\right)\right) = O\left(\mathsf{M}\left(\sum_{k=0}^{2^j-1}\sum_{n=2kw}^{2(k+1)w-1} B_n\right)\right)$$
$$= O\left(\mathsf{M}\left(\sum_{n=0}^{2^h-1} B_n\right)\right)$$

where the first equality holds due to the superlinearity of $\mathsf{M}$. Note that this bound is independent of $j$ and there are $h$ levels so the the total time complexity is

$$O\left(h\mathsf{M}\left(\sum_{n=0}^{2^h-1} B_n\right)\right) = O\left(\mathsf{M}\left(\sum_{n=0}^{N-1} B_n\right)\log N\right)$$

as claimed. □

**Remark 4.2.2.** We remark that Theorem 4.2.1 and its proof is a generalisation of [Bor1985, Prop. 1] and its proof from integers to arbitrarily sized matrices with integer entries.

**Corollary 4.2.3.** *The time required to compute $X_0 \cdots X_{N-1}$ is*

$$O\left(\mathsf{M}\left(\sum_{n=0}^{N-1} B_n\right)\log N\right)$$

*since calculating $X_0 \cdots X_{N-1}$ requires every level of the product tree.*

**Theorem 4.2.4.** *Suppose that $X_0, \ldots, X_{N-1}$ are $d \times d$ matrices with integer entries such that $\beta(X_n) \leq B_n$ for all $n$. Then the space complexity of computing the product tree for $X_0, X_1, \ldots, X_{N-1}$ is*

$$O\left(\left(\sum_{n=0}^{N-1} B_n\right)\log N\right).$$

*Proof.* We use the same method of proof as in the proof of Theorem 4.2.1. We consider the space complexity of calculating the $j$-th level. Define $w := 2^{h-j}$. This

involves calculating

$$X_{0,w} \cdot X_{w,2w}, X_{2w,3w} \cdot X_{3w,4w}, \ldots, X_{(2^{j+1}-2)w,(2^{j+1}-1)w} \cdot X_{(2^{j+1}-1)w,2^{j+1}w}.$$

We bound the space required to calculate the product of $X_{0,w}$ and $X_{w,2w}$. We note that

$$\beta(X_{0,w}) \leq \sum_{n=0}^{w-1} B_n, \ \beta(X_{w,2w}) \leq \sum_{n=w}^{2w-1} B_n$$

from the proof of Theorem 4.2.1. Hence the space required to compute the product of $X_{0,w}$ and $X_{w,2w}$ is

$$\begin{aligned} O\left(\max\left(\beta(X_{0,w}), \beta(X_{w,2w})\right)\right) &= O\left(\max\left(\sum_{n=0}^{w-1} B_n, \sum_{n=w}^{2w-1} B_n\right)\right) \\ &= O\left(\sum_{n=0}^{2w-1} B_n\right). \end{aligned}$$

Similar bounds can be found for the $2^j$ pairs of subproducts so the total space complexity of computing the $j$-th level is

$$\begin{aligned} \sum_{k=0}^{2^j-1} O\left(\sum_{n=2kw}^{2(k+1)w-1} B_n\right) &= O\left(\sum_{k=0}^{2^j-1} \sum_{n=2kw}^{2(k+1)w-1} B_n\right) \\ &= O\left(\sum_{n=0}^{2^h-1} B_n\right). \end{aligned}$$

This bound is independent of $j$ so the total space required is

$$O\left(h \sum_{n=0}^{2^h-1} B_n\right) = O\left(\left(\sum_{n=0}^{N-1} B_n\right) \log N\right)$$

as claimed. □

We can calculate both the time and space complexities of applying the algorithm in Section 4.1 to $A_0, \ldots, A_{N-1}$ and $m_0, \ldots, m_{N-1}$ using Theorems 4.2.1 and 4.2.4 respectively.

We assume that the list of primes less than $N$ have been precomputed; this can be done in $O(N \log N \log\log N)$ time and $O(N)$ memory [Pri1987].

**Theorem 4.2.5.** *The time complexity of using the product tree algorithm in Section 4.1 to compute the product tree of $m_0, \ldots, m_{N-1}$ is $O\left(\mathsf{M}(N)\log N\right)$.*

*Proof.* We recall that $\beta(m_n) = \lambda \lg(n)$ if $n$ is prime and 1 otherwise. Thus we let $B_n := \lambda \lg(n)$ if $n$ is prime and 1 otherwise. Then the time complexity of computing $m_1 \cdots m_N$ is

$$O\left(\mathsf{M}\left(\sum_{n=0}^{N-1} B_n\right)\log N\right)$$

using Theorem 4.2.1. This expression simplifies to

$$\begin{aligned} O\left(\mathsf{M}\left(\sum_{n=0}^{N-1} B_n\right)\log N\right) &= O\left(\mathsf{M}\left(N - \pi(N-1) + \lambda \sum_{p<N} \lg p\right)\log N\right) \\ &= O(\mathsf{M}(N - \pi(N-1) + \lambda\pi(N-1)\lg N)\log N) \\ &= O(\mathsf{M}(N)\log N) \end{aligned}$$

where $\pi(N)$ is approximated using Corollary 3.3.2. □

**Theorem 4.2.6.** *The space complexity of using the product tree algorithm in Section 4.1 to compute the product tree of $m_0, \ldots, m_{N-1}$ is $O(N \log N)$.*

*Proof.* The space complexity is $O\left(\left(\sum_{n=0}^{N-1} B_n\right)\log N\right)$ using Theorem 4.2.4. In a similar vein to the proof of Theorem 4.2.5, we define $B_n$ as $\lambda \lg(n)$ if $n$ is prime and 1 otherwise. We have shown that $\sum_{n=0}^{N-1} B_n = O(N)$ in the proof of Theorem 4.2.5 so it follows that the space required is $O(N \log N)$ as claimed.

□

**Theorem 4.2.7.** *The time complexity of computing the product tree of objects $A_0, \ldots, A_{N-1}$ using the algorithm in Section 4.1 is $O(\mathsf{M}(\tau N)\log N)$.*

*Proof.* We remind the reader that $N$ is a power of 2. We recall that $\beta(A_n) \leq \tau$ for all $n \leq N$. We define $B_n := \tau$ for all $n$. Thus the running time of computing

the product tree is $O\left(\mathsf{M}\left(\sum_{n=0}^{N-1} B_n\right)\log N\right)$ using Theorem 4.2.1. The complexity simplifies to $O(\mathsf{M}(\tau N)\log N)$.

□

**Remark 4.2.8.** We note that the time complexity of computing the product tree of $m_0, \ldots, m_{N-1}$ is smaller than the time complexity of computing the product tree of $A_0, \ldots A_{N-1}$ by a factor of $\log N$ as $\tau = O(\log N)$ by assumption. This is due to the fact that $B_n = \lambda \lg n$ if $n$ is prime and 1 otherwise for the $m_n$ case and $B_n = \tau$ for all $n$ in the $A_n$ case. Since primes are sparse in large intervals, $B_n$ is 1 for almost all $n$ in the moduli case if $N$ is large. However $B_n = \tau$ is a tight bound for the $A_n$ case.

**Theorem 4.2.9.** *The space complexity of using Algorithm 4.1 to compute the product tree of $A_0, \ldots, A_{N-1}$ is $O(\tau N \log N)$.*

*Proof.* The proof uses the same method of proof as Theorem 4.2.6. Define $B_n$ as $\tau$ for all $n$. Then the space complexity is

$$O\left(\left(\sum_{n=0}^{N-1} B_n\right)\log N\right) = O(\tau N \log N)$$

as claimed.

□

## 4.3 Remainder tree algorithm

The remainder tree algorithm takes as inputs two positive integers $a$ and $b$ where $b-a = 2^h$, a sequence of positive integers $n_a, n_{a+1}, \ldots, n_{b-1}$ and a vector $U$ of length $d$ with integer entries. The output is the sequence of vectors $U \bmod n_a, U \bmod n_{a+1}, \ldots, U \bmod n_{b-1}$.

In order to calculate $U \bmod n_a, U \bmod n_{a+1}, \ldots, U \bmod n_{b-1}$, we define the remainder tree recursively. First we compute the product tree for $n_a, \ldots, n_{b-1}$. We

define $U_{a,b} := U \bmod n_{a,b}$. If $b \neq a+1$, then let $c := \lfloor (a+b)/2 \rfloor$. Then we define $U_{a,c} := U_{a,b} \bmod n_{a,c}$ and $U_{c,b} := U_{a,b} \bmod n_{c,b}$ and calculate recursively (this notation should not be confused with the notation in Section 4.1).

**Theorem 4.3.1.** *Suppose that $n_a, \ldots, n_{b-1}, U$ are defined as in Section 4.3. Then $U_{a',b'} = U \bmod n_{a',b'}$.*

*Proof.* We prove this by induction on the levels of the tree. We remark that the base case is true by the definition of $U_{a,b}$. Suppose the parent $U_{a',b'}$ satisfies the inductive hypothesis; that is, $U_{a',b'} = U \bmod n_{a',b'}$. Let $c := \lfloor (a'+b')/2 \rfloor$. We consider its children $U_{a',c'}$ and $U_{c',b'}$. Now

$$\begin{aligned} U_{a',c'} &= U_{a',b'} \bmod n_{a',c'} \\ &= (U \bmod n_{a',b'}) \bmod n_{a',c'} \\ &= U \bmod n_{a',c'} \end{aligned}$$

where the last equality holds since $n_{a',c'}$ divides $n_{a',b'}$. Similarly, it can be shown that $U_{c',b'} = U \bmod n_{c',b'}$. Hence the formula holds for the children if it holds for the parent. This completes the proof.

□

**Corollary 4.3.2.** *Suppose that $n_a, \ldots, n_{b-1}, U$ are defined as in Section 4.3. Then $U_{k,k+1} = U \bmod n_k$.*

*Proof.* This follows from Theorem 4.3.1.

□

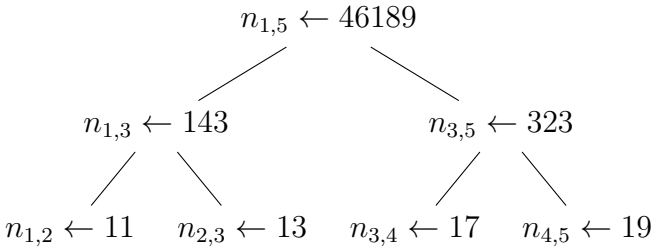
We note that throughout this thesis, we shall not actually use the remainder tree algorithm; it is included for completeness and to motivate the accumulating remainder tree algorithm.

Suppose we want to calculate the sequence of integers 10! modulo 11, 13, 17, 19, 23, 29, 31 and 37. Then we let $U := 10!$, $n_1 := 11$, $n_2 := 13$, $n_3 := 17$, $n_4 := 19$, $n_5 := 23$, $n_6 := 29$, $n_7 := 31$, and $n_8 := 37$.

The first step is to compute a product tree for the moduli.

$$\begin{aligned} n_{1,9} &= n_{5,9} \cdot n_{1,5} \\ &= n_{7,9} \cdot n_{5,7} \cdot n_{3,5} \cdot n_{1,3} \\ &= n_{8,9} \cdot n_{7,8} \cdot n_{6,7} \cdot n_{5,6} \cdot n_{4,5} \cdot n_{3,4} \cdot n_{2,3} \cdot n_{1,2} \\ &= 37 \cdot 31 \cdot 29 \cdot 23 \cdot 19 \cdot 17 \cdot 13 \cdot 11 \\ &= 35336848261. \end{aligned}$$

This can be represented using a graph:




Now we compute the remainder tree itself.

The base case is

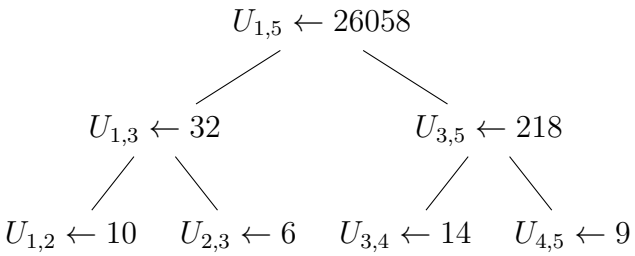
$$\begin{aligned} U_{1,5} &= U \bmod n_{1,5} \\ &= 10! \bmod 46189 \\ &= 26058. \end{aligned}$$

Thus

$$\begin{aligned} U_{1,3} &= U_{1,5} \bmod n_{1,3} \\ &= 26058 \bmod 143 \\ &= 32 \end{aligned}$$

and we continue recursively.

This can be represented using a graph:




Hence 10! is 10 mod 11, 6 mod 13, 14 mod 17, and 9 mod 19.

## 4.4 Complexity of the remainder tree algorithm

In this section, we calculate the time and space complexity of computing $U$ mod $n_0, \ldots, U$ mod $n_{N-1}$ using the remainder tree algorithm in Section 4.3.

**Theorem 4.4.1.** *Let $U \in \mathbb{Z}^d$ and let $n_0, \ldots, n_{N-1}$ be a sequence of positive integers such that $\beta(U) \leq \beta(n_k)$ for all $k$. Let $B_k$ be a sequence of real numbers such that $\beta(n_k) \leq B_k$ for all $k$. The time complexity of using the remainder tree algorithm in Section 4.3 to compute a remainder tree for $U, n_0, \ldots, n_{N-1}$ is*

$$O\left(\mathsf{M}\left(\sum_{k=0}^{N-1} B_k\right) \log N\right).$$

*Proof.* We remind the reader that $N = 2^h$ for a non-negative integer $h$. First we must calculate the product tree of $n_0, n_1, \ldots, n_{N-1}$. This has running time $O(\mathsf{M}(\sum_{k=0}^{N-1} B_k)) \log N)$ using Theorem 4.2.1.

We recall that $U_{a,b} = U \bmod n_{a,b} = U \bmod n_a \cdots n_{b-1}$.

Calculating $U_{0,2^h} = U \bmod n_{0,2^h}$ has time complexity

$$\begin{aligned} O\left(\mathsf{M}\left(\beta(n_0 \cdots n_{2^h-1})\right)\right) &= O\left(\mathsf{M}\left(\sum_{k=0}^{N-1} \beta(n_k)\right)\right) \\ &= O\left(\mathsf{M}\left(\sum_{k=0}^{N-1} B_k\right)\right) \end{aligned}$$

since $\beta(U) \leq \beta(n_k)$ for all $k$.

Now we calculate the time complexity of calculating the $j$-th level. Let $w := 2^{h-j}$. We know the $2^{j-1}$ parents which are

$$U_{0,2w}, U_{2w,4w}, \ldots, U_{(2^j-2)w,2^j w}.$$

Each of those parents has two children in the $j$-th level; the children of

$$U_{2iw,2(i+1)w}$$

are

$$\left(U_{2iw,2(i+1)w}\right)_{2iw,(2i+1)w}, \left(U_{2iw,2(i+1)w}\right)_{(2i+1)w,2(i+1)w}.$$

We observe that

$$\beta(U \bmod n_{2iw,2(i+1)w})$$

is an upper bound for

$$\beta(n_{2i)w} \cdots n_{(2i+1)w-1}), \beta(n_{(2i+1)w} \cdots n_{2(i+1)w-1}).$$

We note that

$$\beta(U \bmod n_{2iw,2(i+1)w}) \leq \lg d + \sum_{k=2iw}^{2(i+1)w-1} \beta(n_k)$$

using Theorem 3.2.3.

Hence the time complexity of computing the children is

$$O\left(\mathsf{M}\left(\lg d + \sum_{k=2iw}^{2(i+1)w-1} \beta(n_k)\right)\right) = O\left(\mathsf{M}\left(\sum_{k=2iw}^{2(i+1)w-1} \beta(n_k)\right)\right).$$

Summing over all of the $2^{j-1}$ parents, the total time complexity is

$$\begin{aligned}\sum_{i=0}^{2^{j-1}-1} O\left(\mathsf{M}\left(\sum_{k=2(i-1)w}^{2iw-1}\beta(n_k)\right)\right) &= O\left(\mathsf{M}\left(\sum_{k=0}^{2^h-1}\beta(n_k)\right)\right)\\ &= O\left(\mathsf{M}\left(\sum_{k=0}^{2^h-1}B_k\right)\right).\end{aligned}$$

This bound holds for every level and is independent of $j$ and there are $h$ levels so the total time complexity is

$$O\left(h\mathsf{M}\left(\sum_{k=0}^{2^h-1}B_k\right)\right) = O\left(\mathsf{M}\left(\sum_{k=0}^{N-1}B_k\right)\log N\right)$$

as required.

□

**Theorem 4.4.2.** *The space complexity of using the algorithm in Section 4.3 to compute a remainder tree for $U, n_0, \ldots, n_{N-1}$ is*

$$O\left(\left(\sum_{k=0}^{N-1}B_k\right)\log N\right).$$

*Proof.* We compute the space required to store the $j$-th level. Let $w := 2^{h-j}$. Then the space required to store the children of the parent $U_{2iw,2(i+1)w}$ is

$$\begin{aligned}\lg d + \beta(n_{2iw,(2i+1)w}) + \lg d + \beta(n_{(2i+1)w,2(i+1)w}) &\leq 2\lg d + \sum_{k=2iw}^{2(i+1)w-1}\beta(n_k)\\ &\leq 2\lg d + \sum_{k=2iw}^{2(i+1)w-1}B_k.\end{aligned}$$

Summing over all $i$ from 0 to $2^j - 1$, the space is

$$O\left(2^{j+1}\lg d + \sum_{k=0}^{N-1}B_k\right) = O\left(\sum_{k=0}^{N-1}B_k\right)$$

space. There are $\log N$ levels so the total space is

$$O\left(\left(\sum_{k=0}^{N-1} B_k\right)\log N\right)$$

as claimed. □

## 4.5 Accumulating remainder tree algorithm

We now have the tools to introduce the *accumulating remainder tree algorithm*. In the remainder tree algorithm, we are changing the moduli but are keeping the object we are reducing modulo constant so we calculate $V \bmod m_a, V \bmod m_{a+1}, \ldots, V \bmod m_{b-1}$. However, for the examples in Section 2.2, we would like to calculate $A_{n-1}A_{n-2}\cdots A_1A_0V \bmod m_n$ for all $n$ up to $N$. Thus a remainder tree is not applicable.

We define the accumulating remainder tree algorithm recursively. The algorithm take as inputs positive integers $a, b$ such that $b-a = 2^h$,a sequence of matrices with integer entries $M_a, \ldots, M_{b-1}$, a vector $Y$ with integer entries and a sequence of positive integers $n_a, \ldots, n_{b-1}$. First, we calculate product trees for $n_a, \ldots, n_{b-1}$ and $M_a, \ldots, M_{b-1}$. Then we define $D_{a,b} := Y_{a,b} = Y \bmod n_{a,b}$. If $b \neq a+1$, then we define $c := \lfloor (a+b)/2 \rfloor$. We define $D_{a,c} := D_{a,b} \bmod n_{a,c}$ and $D_{c,b} := M_{a,c}D_{a,b} \bmod n_{c,b}$ and calculate recursively (This notation should not be confused with the notation in Sections 4.1 or 4.3).

**Theorem 4.5.1.** *Suppose that $M_a, \ldots, M_{b-1}, Y$ and $n_a, \ldots, n_{b-1}$ are defined as in Section 4.5. Then $D_{a',b'} = M_{a,a'}Y \bmod n_{a',b'}$.*

*Proof.* We prove this statement using induction on levels of the tree. For the base case, we note that

$$\begin{aligned} M_{a,a}Y \bmod n_{a,b} &= Y \bmod n_{a,b} \\ &= D_{a,b} \bmod n_{a,b} \end{aligned}$$

so the base case is true.

Now suppose that the parent $D_{a',b'} = M_{a,a'}Y \bmod n_{a',b'}$. Let $c' := \lfloor (a'+b')/2 \rfloor$. We show that $D_{a',c'} = M_{a,a'}Y \bmod n_{a',c'}$ and $D_{c',b'} = M_{a,c'} \bmod n_{c',b'}$. We note that

$$\begin{aligned} D_{a',c'} &= D_{a',b'} \bmod n_{a',c'} \\ &= (M_{a,a'}Y \bmod n_{a',b'}) \bmod n_{a',c'} \\ &= M_{a,a'}Y \bmod n_{a',c'} \end{aligned}$$

where the last equality holds as $n_{a',c'}$ divides $n_{a',b'}$.

Also we observe that

$$\begin{aligned} C_{c',b'} &= (M_{a',c'}C_{a',b'}) \bmod n_{c',b'} \\ &= (M_{a',c'}\,(M_{a,a'}Y \bmod n_{a',b'})) \bmod n_{c',b'} \\ &= M_{a',c'} \bmod n_{c',b'} \cdot (M_{a,a'}Y \bmod n_{a',b'}) \bmod n_{c',b'} \\ &= (M_{a',c'} \bmod n_{c',b'})\,(M_{a,a'}Y \bmod n_{c',b'}) \\ &= M_{a,c'}Y \bmod n_{c',b'} \end{aligned}$$

as required. We note that the second-last equality holds as $n_{c',b'}$ divides $n_{a',b'}$. This completes the proof so $C_{a',b'} = M_{a,a'}Y \bmod n_{a',b'}$. □

**Corollary 4.5.2.** *Suppose that $A_0, \ldots, A_{N-1}, V$ and $m_0, \ldots, m_{N-1}$ are defined as in Section 2.1. Let $M_k = A_k$, $n_k = m_k$ and $D_k = C_k$ for all $k$ and let $Y = V$. Then $D_{k,k+1} = C_k$.*

*Proof.* From Theorem 4.5.1, it follows that

$$\begin{aligned} D_{k,k+1} &= M_{0,k}Y \bmod m_{k,k+1} \\ &\equiv A_{0,k}V \bmod m_{k,k+1} \\ &= A_{k-1} \cdots A_0 V \bmod m_k \end{aligned}$$

as required.

□

**Remark 4.5.3.** We note that the product tree, remainder tree and accumulating remainder tree algorithms are only defined when $b - a$ is a power of 2. They can actually be defined more generally but analysing the time and space complexity of the algorithms is more involved.

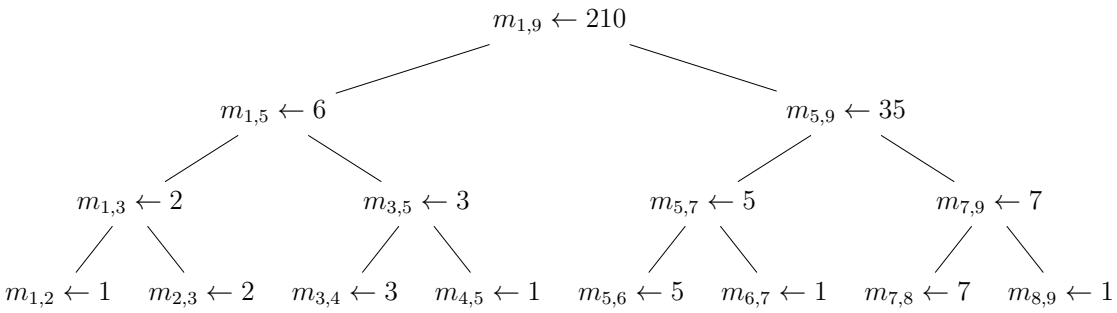
We give an example of the accumulating remainder tree algorithm. Suppose we would like to verify Wilson's theorem up to 7 as in Section 2.2.1.

We define $M_k = A_k := k$ for $k = 1, \ldots, 7$ and 1 for $n = 8$, $Y = V := 1$ and $n_k = m_k := k$ if $k$ is prime and 1 otherwise. The first step is to compute the product trees of $m_1, \ldots, m_9$ and $A_1, \ldots, A_9$.

We note that

$$\begin{aligned} m_{1,9} &= m_{5,9}m_{1,5} \\ &= m_{5,7}m_{7,9}m_{3,5}m_{1,3} \\ &= m_{8,9}m_{7,8}m_{6,7}m_{5,6}m_{4,5}m_{3,4}m_{2,3}m_{1,2} \\ &= 1 \cdot 7 \cdot 1 \cdot 5 \cdot 1 \cdot 3 \cdot 2 \cdot 1 \\ &= 210. \end{aligned}$$

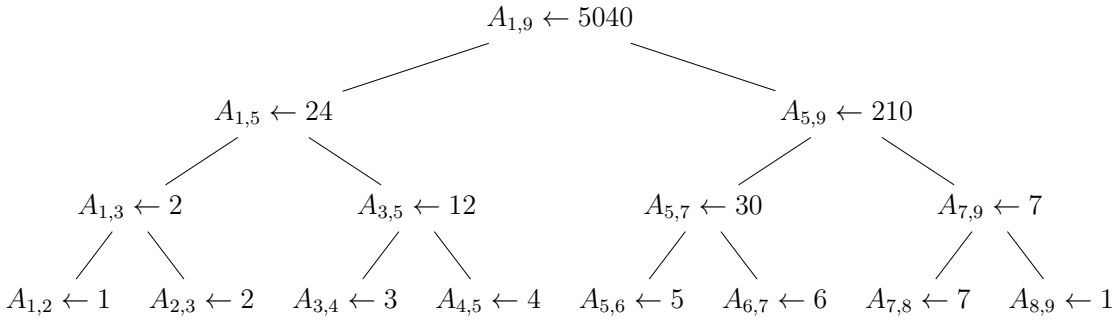
This can be represented using a graph:



Furthermore, we observe that

$$\begin{aligned} A_{1,9} &= A_{5,9}A_{1,5} \\ &= A_{7,9}A_{5,7}A_{3,5}A_{1,3} \\ &= A_{8,9}A_{7,8}A_{6,7}A_{5,6}A_{4,5}A_{3,4}A_{2,3}A_{1,2} \\ &= 1 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \\ &= 5040. \end{aligned}$$

This can be represented using a graph:




The second step is to compute the accumulating remainder tree itself.

The base case is

$$\begin{aligned} C_{1,9} &= V \bmod m_{1,9} \\ &= 1 \bmod 210 \\ &= 1. \end{aligned}$$

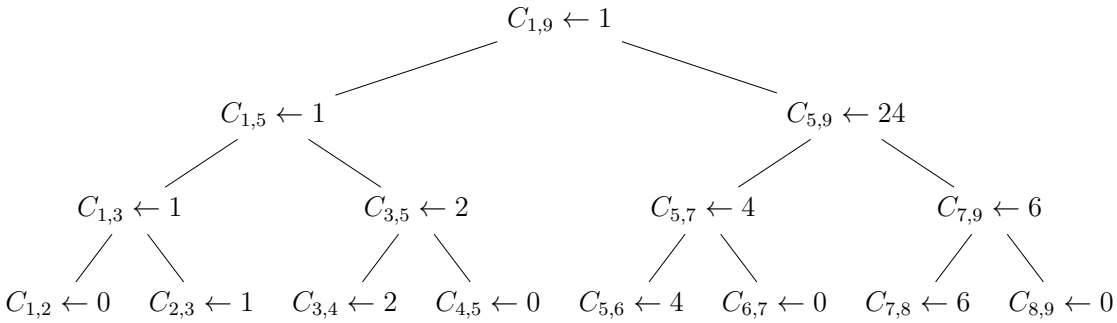
Recursively,

$$\begin{aligned} C_{1,5} &= C_{1,9} \bmod m_{1,5} \\ &= 1 \bmod 6 \\ &= 1. \end{aligned}$$

$$\begin{aligned} C_{5,9} &= A_{1,5}C_{1,9} \bmod m_{5,9} \\ &= 24 \cdot 1 \bmod 35 \\ &= 24 \end{aligned}$$

and we continue recursively.

This can be represented using a graph:




Thus

$$(p-1)! \bmod p = C_p = C_{p,p+1} = p - 1$$

for every prime between 1 and 9 as required by Wilson's Theorem.

## 4.6 Complexity analysis of the accumulating remainder tree algorithm

In this section, we bound the space and time complexity of the accumulating remainder tree algorithm in Section 4.5. Throughout this section, we will assume that $a := 0$, $b := N$, $M_k = A_k$, $n_k = m_k$ and $Y = V$ where $N$, $M_k$, $n_k$ and $V$ are defined as Section 2.1.

**Theorem 4.6.1.** *Suppose $A_0, \ldots, A_{N-1}, \tau$ and $m_0, \ldots, m_{N-1}$ are defined as in Sections 2.1 and 3.3. The time complexity of using the accumulating remainder tree algorithm in Section 4.5 to compute $C_0, \ldots, C_{N-1}$ is*

$$O(\mathsf{M}(\tau N) \log N).$$

*Proof.* Let $M_k = A_k$, $n_k = m_k$ and $D_k = C_k$ for all $k$ and let $Y = V$. The first step is to compute product trees for $m_n$ and $A_n$ which have running times of $O(\mathsf{M}(N)\log N)$ and $O(\mathsf{M}(\tau N)\log N)$ respectively using Theorems 4.2.5 and 4.2.7. It follows that the time complexity of computing both trees is $O(\mathsf{M}(\tau N)\log N)$ since $\tau = O(\log N)$ by assumption. We calculate the time required to compute each level.

The first level requires calculating $V$ mod $m_{0,2^h}$ which has time complexity

$$O(\mathsf{M}(\max(\beta(V), \beta(m_{0,2^h})))) = O(\mathsf{M}(2^h))$$

since $\beta(V) \leq \beta(m_{0,2^h})$ which is $O(2^h)$.

We now bound the time required to compute the $j$-th level for $j > 1$. We define $w := 2^{h-j}$. The parents in the $(j-1)$-th level are of the form

$$C_{0,2w}, \ldots, C_{(2^{j-1}-2)w,2^{j-1}w}.$$

Let $i = 0, \ldots, 2^{j-1} - 1$. We calculate the time taken to compute the children from the parent $C_{2iw,2(i+1)w}$.

The left child is

$$C_{2iw,(2i+1)w} = C_{2iw,2(i+1)w} \bmod m_{2iw,(2i+1)w}$$

so we are only required to use modular arithmetic. We remark that $C_{2iw,2(i+1)w}$ is a vector which has been reduced modulo $m_{2iw,2(i+1)w}$ so

$$\beta(C_{2iw,2(i+1)w}) \leq \lg d + \beta(m_{2iw,2(i+1)w})$$

using Proposition 3.2.3. We also observe that

$$\beta(m_{2iw,(2i+1)w}) \leq \lg d + \beta(m_{2iw,2(i+1)w}).$$

Hence the time complexity of this modular arithmetic is

$$\begin{aligned} O(\mathsf{M}(\max(\beta(C_{2iw,2(i+1)w}), \beta(m_{2iw,(2i+1)w})))) &= O(\mathsf{M}(\lg d + \beta(m_{2iw,2(i+1)w}))) \\ &= O(\mathsf{M}(\beta(m_{2iw,2(i+1)w}))) \\ &= O(\mathsf{M}(2(i+1)w - 2iw)) \\ &= O(\mathsf{M}(2w)). \end{aligned}$$

Now we bound the time complexity of computing the right child

$$C_{(2i+1)w,2(i+1)w} = (A_{2iw,(2i+1)w} C_{2iw,2(i+1)w}) \bmod m_{(2i+1)w,2(i+1)w}.$$

We first bound the time complexity of the multiplication. We remark that $\beta(A_{2iw,(2i+1)w}) \leq \tau w$ and $\beta(C_{2iw,2(i+1)w}) \leq \lg d + \beta(m_{2iw,2(i+1)w})$. Hence the time complexity is

$$\begin{aligned} O(\mathsf{M}(\max(\tau w, \lg d + \beta(m_{2iw,2(i+1)w})))) &= O(\mathsf{M}(\max(\tau w, \beta(m_{2iw,2(i+1)w})))) \\ &= O(\mathsf{M}(\max(\tau w, 2w))) \\ &= O(\mathsf{M}(\tau w)). \end{aligned}$$

using the fact that $\tau = O(\log N)$ by assumption.

Now we bound the time complexity of the modular arithmetic for the right child. This involves reducing $A_{2iw,(2i+1)w} C_{2iw,2(i+1)w}$ modulo $m_{(2i+1)w,2(i+1)w}$. We note that

$$\begin{aligned} \beta(A_{2iw,(2i+1)w} C_{2iw,2(i+1)w}) &\leq \beta(A_{2iw,(2i+1)w}) + \beta(C_{2iw,2(i+1)w}) \\ &\leq \tau w + \lg d + \beta(m_{2iw,2(i+1)w}) \end{aligned}$$

from the proof of the bound of the time complexity. We observe that

$$\beta(m_{(2i+1)w,2(i+1)w}) \leq \tau w + \lg d + \beta(m_{2iw,2(i+1)w}).$$

Hence the time complexity of the modular arithmetic is

$$\begin{aligned} O(\mathsf{M}(\tau w + \lg d + \beta(m_{2iw,2(i+1)w}))) &= O(\mathsf{M}(\tau w + \lg d + 2w)) \\ &= O(\mathsf{M}(\tau w)). \end{aligned}$$

The total time complexity of the multiplication for computing these two children is $O(\mathsf{M}(\tau w))$. The total time complexity of the modular arithmetic for these two children is $O(\mathsf{M}(2w)) + O(\mathsf{M}(\tau w)) = O(\mathsf{M}(\tau w))$ since $\tau = O(\log N)$ by assumption. Thus the total running time of computing the two children is $O(\mathsf{M}(\tau w))$. This bound is independent of $i$. Hence the total time complexity of computing the $j$-th level is

$$2^{j-1} O(\mathsf{M}(\tau w)) = O(\mathsf{M}(\tau 2^h))$$

using the superlinearity of $\mathsf{M}$.

There are $h$ levels so the time complexity of computing the accumulating remainder tree is

$$h O(\mathsf{M}(\tau 2^h)) = O(\mathsf{M}(\tau N) \log N).$$

Thus the total complexity of the algorithm is

$$O(\mathsf{M}(\tau N) \log N) + O(\mathsf{M}(\tau N) \log N) = O(\mathsf{M}(\tau N) \log N)$$

as claimed.

□

**Theorem 4.6.2.** *The space complexity of the accumulating remainder tree algorithm in Section 4.5 is $O(\tau N \log N)$.*

*Proof.* Let $M_k = A_k$, $n_k = m_k$ and $D_k = C_k$ for all $k$ and let $Y = V$. The first step is to compute the product trees for $m_n$ and $A_n$. The space complexity of the product trees for $m_n$ and $A_n$ is $O(N \log N)$ and $O(\tau N \log N)$ respectively. It

follows that the total space complexity of storing both trees is $O(\tau N \log N)$ since $\tau = O(\log N)$ by assumption.

The first level requires calculating $V$ mod $m_{0,2^h}$. The space complexity of calculating $V$ mod $m_{0,2^h}$ is

$$O(\max(\beta(V), \beta(m_{0,2^h}))) = O(2^h)$$

since $\beta(V) \leq \beta(m_{0,2^h})$ which is $O(2^h)$.

We now bound the space required to compute the $j$-th level for $j > 1$. We define $w := 2^{h-j}$. The parents in the $(j-1)$-th level are of the form

$$C_{0,2w}, \ldots, C_{(2^{j-1}-2)w,2^{j-1}w}.$$

Let $i = 0, \ldots, 2^{j-1} - 1$. We calculate the memory required to compute the children from the parent $C_{2iw,2(i+1)w}$.

The left child is

$$C_{2iw,(2i+1)w} = C_{2iw,2(i+1)w} \bmod m_{2iw,(2i+1)w}$$

so we are only required to use modular arithmetic. We remark that $C_{2iw,2(i+1)w}$ is a vector which has been reduced modulo $m_{2iw,2(i+1)w}$ so

$$\beta(C_{2iw,2(i+1)w}) \leq \lg d + \beta(m_{2iw,2(i+1)w})$$

using Proposition 3.2.3. We also observe that

$$\beta(m_{2iw,(2i+1)w}) \leq \lg d + \beta(m_{2iw,2(i+1)w}).$$

Hence the space complexity of this modular arithmetic is

$$\begin{aligned} O(\max(\beta(C_{2iw,2(i+1)w}), \beta(m_{2iw,(2i+1)w}))) &= O((\lg d + \beta(m_{2iw,2(i+1)w})) \\ &= O(\beta(m_{2iw,2(i+1)w})) \\ &= O(2(i+1)w - 2iw) \\ &= O(2w). \end{aligned}$$

Now we bound the space complexity of computing the right child

$$C_{(2i+1)w,2(i+1)w} = (A_{2iw,(2i+1)w} C_{2iw,2(i+1)w}) \bmod m_{(2i+1)w,2(i+1)w}.$$

We first bound the memory required for the multiplication. We remark that $\beta(A_{2iw,(2i+1)w}) \leq \tau w$ and $\beta(C_{2iw,2(i+1)w}) \leq \lg d + \beta(m_{2iw,2(i+1)w})$. Hence the space complexity is

$$\begin{aligned} O(\max(\beta(A_{2iw,(2i+1)w}), \beta(C_{2iw,2(i+1)w}))) &= O(\max(\tau w, \lg d + \beta(m_{2iw,2(i+1)w})) \\ &= O(\max(\tau w, 2w)) \\ &= O(\tau w) \end{aligned}$$

using the fact that $\tau = O(\log N)$ by assumption.

Now we bound the space complexity of the modular arithmetic for the right child. This involves reducing $A_{2iw,(2i+1)w} C_{2iw,2(i+1)w}$ modulo $m_{(2i+1)w,2(i+1)w}$. We note that

$$\begin{aligned} \beta(A_{2iw,(2i+1)w} C_{2iw,2(i+1)w}) &\leq \beta(A_{2iw,(2i+1)w}) + \beta(C_{2iw,2(i+1)w}) \\ &\leq \tau w + \lg d + \beta(m_{2iw,2(i+1)w}) \end{aligned}$$

from the bounding of the space complexity. We observe that

$$\beta(m_{(2i+1)w,2(i+1)w}) \leq \tau w + \lg d + \beta(m_{2iw,2(i+1)w}).$$

Hence the memory required for the modular arithmetic is

$$\begin{aligned} O(\tau w + \lg d + \beta(m_{2iw,2(i+1)w})) &= O(\tau w + \lg d + 2w) \\ &= O(\tau w). \end{aligned}$$

The total space used for the multiplication for computing these two children is $O(\tau w)$. The total space complexity of the modular arithmetic for these two children is $O(2w) + O(\tau w) = O(\tau w)$ since $\tau = O(\log N)$ by assumption. Thus the total memory required to compute the two children is $O(\tau w)$. This bound is independent of $i$. Hence the total space complexity of computing the $j$-th level is

$$2^{j-1}O(\tau w) = O(\tau 2^h).$$

There are $h$ levels so the space complexity of computing the accumulating remainder tree is

$$hO(\tau 2^h) = O(\tau N \log N).$$

Thus the total space complexity of the algorithm is

$$O(\tau N \log N) + O(\tau N \log N) = O(\tau N \log N)$$

as claimed.

□

**Corollary 4.6.3.** *Kurepa's conjecture can be verified for every prime up to $N$ in $O(\mathsf{M}(N \log N) \log N)$ time using $O(N \log^2 N)$ space.*

*Proof.* We let

$$d := 2, A_0 := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, A_n := \begin{pmatrix} n & 0 \\ n & 1 \end{pmatrix}, V := \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \lambda := 1$$

for $n \geq 1$. Thus

$$A_{p-1}A_{p-2}\cdots A_1A_0V \equiv \begin{pmatrix} (p-1)! \\ !p \end{pmatrix} \bmod p$$

and from this, we can read off $!p \bmod p$. We note that $\beta(A_n) = O(\log n)$ so $\tau = O(\log N)$. Every $C_p$ can be computed for every prime less than $N$ using the accumulating remainder tree algorithm in 4.5. The time and space complexity of using this algorithm are $O(\mathsf{M}(\tau N)\log N)$ and $O(\tau N \log N)$ using Theorems 4.6.1 and 4.6.2 respectively. This simplifies to $O(\mathsf{M}(N\log N)\log N)$ time using $O(N\log^2 N)$ space since $\tau = O(\log N)$ which completes the proof. □

## 4.7 History of the accumulating remainder tree algorithm

The product tree has been invented independently by many authors; most of the early uses of product trees only considered specific applications and not in full generality. Some of these applications include calculating the sum of two non-negative integers [WS1958] and single variable polynomial evaluation [Est1960]. We were unable to obtain a copy of [WS1958]. Kogge and Stone recognised that H.R. Downs, H Lomax and Trout had independently discovered the product tree algorithm in full generality [KS1973]. Further information can be found in [Ber2008].

The remainder tree was first introduced by Moenck and Borodin and defined for elements of a Euclidean Domain [MB1972]. It is then applied to dividing polynomials and applying the Chinese remainder theorem for integers.

The accumulating remainder tree algorithm was first used by Gerbicz in 2011 [Ger] and was first published in 2014 to search for Wilson primes in [CGH2014]. It was subsequently used in [Har2014] to compute the zeta function of a hyperelliptic curve over $\mathbb{F}_p$ for all $p < N$ and generalised to all arithmetic schemes in [Har2015]. It was also applied in [HS2014] to compute the Hasse-Witt matrix of an arbitrary hyperelliptic curve for all primes of good reduction up to some upper bound $N$. This algorithm was subsequently improved in [HS2016]. Another use was in [HMS2016] to calculate the local zeta functions of a given curve of genus three. In essence, these applications are all very similar; rather than calculate some $C_p \bmod p$ individually

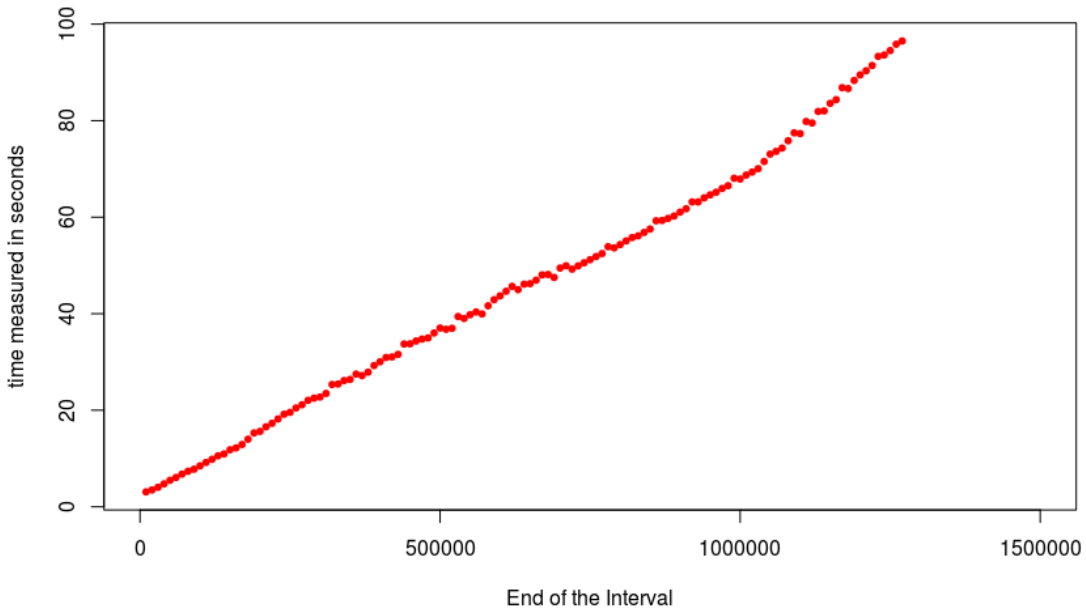
where $C_p$ is defined as in Section 2.1, a recurrence relation is sought and all $C_p$ are calculated simultaneously to minimise repeating the same calculations while using modular arithmetic to shorten computation times.

Further space and running time optimisations to the accumulating remainder tree are possible. One important optimisation is the accumulating remainder forest technique; the idea can be found in [HS2014] and [HS2016] for example. However this technique is outside the scope of this thesis.

# Chapter 5

# A toy implementation

We produced an implementation to verify Kurepa's conjecture up to some bound $N$. We remark that it was merely a proof of concept implementation and was not designed to improve the bounds that were found in [AT2014]. The implementation was written in Sage [Sag]. Sage uses the GMP library [Gra] for computations with large integers. We measured the time taken to verify Kurepa's conjecture up to $N = 10^4, 2 \cdot 10^4, \ldots, 120 \cdot 10^4$. The implementation ran using 1 core on a 4 core i-7-4790 processor at 3.60 GHz and using 16 GB RAM. This took 1 CPU-hour for all the calculations. The code can be accessed at [Raj].



A separate test was done to verify the correctness of the algorithm; a random prime $p < N$ was selected and $!p \bmod p$ was calculated using a for-loop and compared to the result from the accumulating remainder tree algorithm. This was repeated 1000 times and they agreed every time. The code can be accessed at [Raj].

The graph provides evidence that the algorithm is running in quasilinear time. As an example, the time taken to compute $N = 5 \cdot 10^6$ is roughly half the time required to compute $N = 10^7$.

It remains to scale up this implementation and write it using a language with lower overheads such as C++. Further possible optimisations include parallelising the implementation and using the accumulating remainder forest technique.

# Chapter 6

# Conclusion

The left factorial of $n$ is defined to be $0! + 1! + \cdots + (n-1)!$ and is denoted by $!n$. Kurepa conjectured that $!n$ is not divisible by $n$ for $n > 2$ and showed that it was sufficient to check for odd primes $p$. We provided a survey of known results on the search for a counterexample to Kurepa's conjecture and analysed the complexity of the algorithms used. These algorithms are all exponential in $\log p$. We developed the first known algorithm whose complexity is polynomial in $\log p$ when averaged over primes.

Further research could involve applying the accumulating remainder tree to similar problems in number theory. Other directions of interest could be applying the accumulating remainder forest technique from [HS2014] and [HS2016], writing the code in C++ and parallelisng it and optimising the algorithm to be more space efficient at the cost of increasing the time complexity such as in [CGH2014].

# References


[AT2014] V. Andrejić and M. Tatarevic, *Searching for a counterexample to Kurepa's Conjecture*, Math. Comp. **85** (September 2014), no. 302, 3061–3068.

[BB2004] D. Barsky and B. Benzaghou, *Nombres de Bell et somme de factorielles*, J. Théor Nombres Bordeaux **16** (2004), no. 1, 1–17 (French).

[BB2011] D. Barsky and B. Benzaghou, *Erratum à l àrticle Nombres de Bell et somme de factorielles*, J. Théor. Nombres Bordeaux **23** (2011), no. 2, 527 (French).

[Bel1934] E. T. Bell, *Exponential Numbers*, Amer. Math. Monthly **41** (1934), no. 7, 411–419.

[Ber2008] D. J. Bernstein, *Fast multiplication and its applications*, Algorithmic number theory: lattices, number fields, curves and cryptography, 2008, pp. 325–384.

[Bor1985] P. B. Borwein, *On the complexity of calculating factorials*, J. Algorithms **6** (1985), no. 3, 376–380.

[CGH2014] E. Costa, R. Gerbicz, and D. Harvey, *A search for Wilson primes*, Math. Comp. **83** (2014), no. 290, 3071–3091.

[CW1990] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput. **9** (1990), no. 3, 251–280.

[Est1960] G. Estrin, *Organization of Computer Systems: The Fixed Plus Variable Structure Computer*, Papers Presented at the May 3-5, 1960, Western Joint IRE-AIEE-ACM Computer Conference (San Francisco, California), 1960, pp. 33–40.

[Eul1765] L. Euler, *Observationes Analyticae*, Novi Comment. Acad. Sci. Imp. Petropol. **11** (1765), 129–130.

[FRR1987] C. K. Fong, H. Radjavi, and P. Rosenthal, *Norms for matrices and operators*, J. Operator Theory **18** (1987), no. 1, 99–113.

[GR2014] L. H. Gallardo and O. Rahavandrainy, *Bell Numbers Modulo a Prime Number, Traces and Trinomials*, Electron. J. Combin. **21** (2014), no. 4, P4.49.

[Gal2000] Y. Gallot, *Is the number of primes* $\frac{1}{2}\sum_{i=0}^{n-1} i!$ *finite?* (2000), http://yves.gallot.pagesperso-orange.fr/papers/lfact.html. Accessed June 21, 2018.

[Ger] R. Gerbicz, *Factorial modulo a prime*, http://ftp.mersenneforum.org/showthread.php?t=15948&page=7. Accessed June 21, 2018.

[Gog1991] G. Gogić, *Parallel algorithms in arithmetic*, Masters Thesis, Belgrade University, 1991.

[Gra] T. Granlund, *The GNU Multiple Precision Arithmetic Library (Version 6.1.2)*, https://gmplib.org. Accessed July 25, 2018.

[Guy] R. Guy, *Unsolved Problems in Number Theory*, 3rd ed., Vol. 1, Springer-Verlag.

[OEIS] OEIS Foundation Inc. (2018), *The On-Line Encyclopedia of Integer Sequences*, https://oeis.org/. Accessed June 28, 2017.

[Ham1999] A. G. Hamadene, *Congruences pour quelques suites classiques de nombres; sommes de factorielles et calcul ombral*, Ph.D. Thesis, Universite De Neuchatel, 1999, https://doc.rero.ch/record/4372/files/2_these_GertschHamadeneA.pdf (French).

[HvdH2018] D. Harvey and J. van der Hoeven, *Faster integer multiplication using short lattice vectors*, ArXiv e-prints (2018), available at 1802.07932.

[HMS2016] D. Harvey, M. Massierer, and A. V. Sutherland, *Computing L-series of geometrically hyperelliptic curves of genus three*, LMS J. Comput. Math. **19** (2016).

[HS2014] D. Harvey and A. V. Sutherland, *Computing Hasse–Witt matrices of hyperelliptic curves in average polynomial time*, LMS J. Comput. Math. **17** (2014).

[HS2016] D. Harvey and A. V. Sutherland, *Computing Hasse–Witt matrices of hyperelliptic curves in average polynomial time, II*, Frobenius distributions: Lang–Trotter and Sato–Tate conjectures, 2016.

[Har2014] D. Harvey, *Counting points on hyperelliptic curves in average polynomial time*, Ann. of Math. **179** (2014), no. 2, 783-803.

[Har2015] D. Harvey, *Computing zeta functions of arithmetic schemes*, Proc. Lond. Math. Soc. **111** (2015), no. 6, 1379–1401.

[Has2003] M. Hassani, *Derangements and Applications*, J. Integer Seq. **6** (2003), no. 1.

[Ili2015] I. Ilijašević, *Verification of Kurepa's Left Factorial Conjecture for Primes up to* $2^{31}$, Transactions on Internet Research (January 2015).

[IM2004] A. Ivić and Ž. Mijajlović, *On Kurepa's problems in number Theory*, Publ. Inst. Math. (Beograd) (N.S.) **57(71)** (January 2004), 19–28.

[Lag1771] J. L. Lagrange, *Demonstration dú n thÃľoréme nouveau concernant les nombres premiers*, Nouveaux Mémoires de l'Académie Royale des Sciences et Belles-Lettres **2** (1771), 125–137 (French).

[LG2014] F. Le Gall, *Powers of Tensors and Fast Matrix Multiplication*, Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (Kobe, Japan), 2014, pp. 296–303.

[Kel2004] B. C. Kellner, *Some remarks on Kurepa's left factorial* (November 2004), unpublished.

[KS1973] P. M. Kogge and H. S. Stone, *A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations*, IEEE Trans. Computers **C-22** (1973), no. 8, 786–793.

[Kur1971] Đ. Kurepa, *On The Left Factorial !n*, Math. Balkanica (N.S.) **1** (1971), 147–153.

[Mal2003] B. J. Malešević, *Some considerations in connection with Kurepa's function*, Univ. Beograd. Publ. Elektrotehn. Fak. Ser. Mat. **14** (January 2003), 26–36.

[Meš2015] R. Meštrović, *The Kurepa-Vandermonde matrices arising from Kurepa's left factorial hypothesis*, Filomat **29** (2015), no. 10, 2207–2215.

[Mij1990] Ž. Mijajlović, *On some formulas involving !n and the verification of the !n-hypothesis by use of computers*, Publ. Inst. Math. (Beograd) (N.S.) **47(61)** (January 1990), 24–32.

[Mil1996] G. V. Milovanović, *A sequence of Kurepa's functions*, Sci. Rev. Ser. Sci. Eng. **19–20** (January 1996), 137–146.

[MP2002] G. V. Milovanović and A. Petojević, *Generalized Factorial Functions, Numbers and Polynomials*, Math. Balkanica (N.S.) **16** (2002), no. 1–4, 113–130.

[McI1995] R. J. McIntosh, *On the converse of Wolstenholme's Theorem*, Acta Arith. **71** (1995), no. 4, 381–389.

[MR2007] R. J. McIntosh and E. Roettger, *A search for Fibonacci-Wieferich and Wolstenholme primes*, Math. Comp. **76** (2007), no. 260, 2087-2094.

[MB1972] R. Moenck and A. Borodin, *Fast Modular Transforms via Division*, Proceedings of the 13th Annual Symposium on Switching and Automata Theory (Swat 1972), 1972, pp. 90–96.

[New1980] D.J. Newman, *Simple Analytic Proof of the Prime Number Theorem*, Amer. Math. Monthly **87** (1980), no. 9, 693–696.

[Pet2002] A. Petojević, *The function ${}_vM_m(s:a,z)$ and some well-known sequences*, J. Integer Seq. **5** (2002), no. 1, 16.

[PŽ1999] A. Petojević and M. Žižović, *Trees and the Kurepa hypothesis for left factorial*, Filomat **13** (1999), 31–40.

[PŽC1999] A. Petojević, M. Žižović, and S. D. Cvejić, *Difference Equations and New Equivalents of the Kurepa Hypothesis*, Math. Morav. **3** (1999), 39–42.

[Pri1987] P. Pritchard, *Linear Prime-Number Sieves: A Family Tree.*, Sci. Comput. Programming, 1987, pp. 17–35.

[Raj] R. Rajkumar, *Accumulating remainder tree code*, https://github.com/ramananrajkumar92/accumulating-remainder-tree-algorithm-code. Accessed January 11, 2019.

[Sag] *SageMath (Version 8.3)*, https://www.sagemath.org. Accessed July 25, 2018.

[Šam1974] Z. Šami, *On the M-hypothesis of Dj. Kurepa*, Math. Balkanica (N.S.) **4** (1974), 530–532.

[Sta1973] J. Stanković, *Über einige Relationen zwischen Fakultäten und den linken Fakultäten*, Math Balkanica (N.S.) **3** (1973), 488–497.

[SŽ1974] J. Stanković and M. Žižović, *Noch einige Relationen zwischen den Fakultäten und den linken Fakultäten*, Math Balkanica (N.S.) **4** (1974), 555–559.

[Str1969] V. Strassen, *Gaussian Elimination is Not Optimal*, Numer. Math. **13** (1969), no. 4, 354–356.

[Sun] Z. Sun, *Characterizing primes via central trinomial coefficients*, https://listserv.nodak.edu/cgi-bin/wa.exe?A2=ind1612&L=NMBRTHRY&P=1301. Accessed December 7, 2016.

[vZGG1999] J. von Zur Gathen and J. Gerhard, *Modern Computer Algebra*, 1st ed., Cambridge University Press, 1999.

[WS1958] A. Weinberger and J.L. Smith, *A Logic for High-Speed Addition*, National Bureau of Standards Circulation **591** (1958), 3–12.

[Wol1862] J. Wolstenholme, *On Certain Properties of Prime Numbers*, The Quarterly Journal of Pure and Applied Mathematics (1862), 35–39.

[Zag1997] D. Zagier, *Newman's Short Proof of the Prime Number Theorem*, Amer. Math. Monthly **104** (1997), no. 8, 705–708.

[Živ1999] M. Živković, *The number of primes* $\sum_{i=1}^{n}(-1)^{n-i}i!$ *is finite*, Math. Comp. **225** (January 1999), no. 68, 403–409.