Faster Min-Plus Product for Monotone Instances

Shucheng Chi ∗ Ran Duan † Tianle Xie † Tianyi Zhang §

Abstract

In this paper, we show that the time complexity of monotone min-plus product of two $n \times n$ matrices is $\tilde{O}(n^{(3+\omega)/2}) = \tilde{O}(n^{2.687})$, where $\omega < 2.373$ is the fast matrix multiplication exponent [Alman and Vassilevska Williams 2021]. That is, when $A$ is an arbitrary integer matrix and $B$ is either row-monotone or column-monotone with integer elements bounded by $O(n)$, computing the min-plus product $C$ where $C_{i,j} = \min_k \{A_{i,k} + B_{k,j}\}$ takes $\tilde{O}(n^{(3+\omega)/2})$ time, which greatly improves the previous time bound of $\tilde{O}(n^{(12+\omega)/5}) = \tilde{O}(n^{2.875})$ [Gu, Polak, Vassilevska Williams and Xu 2021]. Then by simple reductions, this means the following problems also have $\tilde{O}(n^{(3+\omega)/2})$ time algorithms:

• $A$ and $B$ are both bounded-difference, that is, the difference between any two adjacent entries is a constant. The previous results give time complexities of $\tilde{O}(n^{2.824})$ [Bringmann, Grandoni, Saha and Vassilevska Williams 2016] and $\tilde{O}(n^{2.779})$ [Chi, Duan and Xie 2022].

• $A$ is arbitrary and the columns or rows of $B$ are bounded-difference. Previous result gives time complexity of $\tilde{O}(n^{2.922})$ [Bringmann, Grandoni, Saha and Vassilevska Williams 2016].

• The problems reducible to these problems, such as language edit distance, RNA-folding, scored parsing problem on BD grammars. [Bringmann, Grandoni, Saha and Vassilevska Williams 2016].

Finally, we also consider the problem of min-plus convolution between two integral sequences which are monotone and bounded by $O(n)$, and achieve a running time upper bound of $\tilde{O}(n^{1.5})$. Previously, this task requires running time $\tilde{O}(n^{(9+\sqrt{177})/12}) = \tilde{O}(n^{1.859})$ [Chan and Lewenstein 2015].

∗Tsinghua University, chisc21@mails.tsinghua.edu.cn
†Tsinghua University, duanran@mail.tsinghua.edu.cn
‡Tsinghua University, xtl21@mails.tsinghua.edu.cn
§Tel Aviv University, tianyiz21@tauex.tau.ac.il
1 Introduction

The min-plus product $C = A \ast B$ between two $n \times n$ matrices $A, B$ is defined as $C_{i,j} = \min_{1 \leq k \leq n} \{A_{i,k} + B_{k,j}\}$. The straightforward algorithm for min-plus product runs in $O(n^3)$ time, and a long line of research has been dedicated to breaking this cubic barrier. The currently fastest algorithm by Williams [Wil18a] for min-plus product runs in time $n^3/2^{\Theta(\sqrt{\log n})}$, and it remains a major open question whether a truly sub-cubic running time of $O(n^{3-\epsilon})$ can be achieved for some constant $\epsilon > 0$. In fact, it is widely believed that truly sub-cubic time algorithms do not exist according to the famous APSP hardness conjecture from the literature of fine-grained complexity [Wil18b].

Although min-plus product is hard in general cases, when the input matrices have certain structures, truly sub-cubic time algorithms are known. For example, when all matrix entries are bounded in absolute value by $W$, min-plus product can be computed in time $\tilde{O}(Wn^\omega)$ [AGM97]. Matrices with more general structural properties are studied in recent years. In paper [BGSW19], the authors introduced the notion of bounded-difference matrices.

**Definition 1.1.** An integral matrix is called bounded-difference, if each pair of adjacent elements differ by at most a constant $\delta$. Formally, a bounded-difference $n \times n$ matrix $X$ satisfies that for any pair of indices $1 \leq i, j \leq n$, we have:

$$|X_{i,j} - X_{i,j+1}| \leq \delta$$

$$|X_{i,j} - X_{i+1,j}| \leq \delta$$

The importance of this special type of min-plus product between bounded-difference matrices is demonstrated by its connection to sub-cubic algorithms for other problems (for example, language edit distance [BGSW19], RNA folding [BGSW19], and tree edit distance [Mao21]). As their main technical result, the authors of [BGSW19] gave the first sub-cubic time algorithm for computing min-plus product between two $n \times n$ bounded-difference matrices in time $\tilde{O}(n^{\omega/3})$. This upper bound was improved significantly to $\tilde{O}(n^{2+\omega/3})$ by a very recent work [CDX22]; here $\omega$ refers to the fast matrix multiplication exponent [AW21].

Following [BGSW19], less restricted types of matrices are studied in [WX20, GWX21]. In their work [WX20], Williams and Xu considered the case where one of the input matrices is monotone.

**Definition 1.2.** An $n \times n$ integral matrix is called row-monotone, or simply monotone, if all entries are nonnegative integers bounded by $O(n)$ and each row of this matrix is non-decreasing, that is, if $X$ is monotone, then for $i, j$, $0 \leq X_{i,j} = O(n)$, $X_{i,j} \leq X_{i,j+1}$. Similarly we can define column-monotone matrix.

It was shown in [GWX21] that min-plus product in the bounded-difference setting can be reduced to the monotone setting in quadratic time, so this monotone setting is at least as hard in general. With this definition, Williams and Xu [WX20] studied the monotone min-plus product problem where $A$ is an arbitrary integral matrix and $B$ is monotone, which has an application in the batch range mode problem, and they presented a sub-cubic algorithm with running time $\tilde{O}(n^{(15+\omega)/6})$. This upper bound was later improved to $\tilde{O}(n^{(12+\omega)/5})$ in a recent work [GWX21].

Other than matrix pairs, the concept of min-plus also applies to sequence pairs. Given two sequences $A, B$ with $n$ entries, their min-plus convolution $C = A \diamond B$ can be defined as $C_k = \min_{0 \leq k < n} \{A_i + B_{k-i}\}$, $\forall 2 \leq k \leq 2n$. Chan and Lewenstein [CL15] studied fast algorithms for min-plus convolution when input sequences $A, B$ are monotone.

**Definition 1.3.** An integral sequence of length $n$ is called monotone, if this sequence is monotonically increasing, plus that all entries are nonnegative and bounded by $O(n)$. 


When both sequences \( A, B \) are monotone, Chan and Lewenstein [CL15] showed that min-plus convolution can be computed in sub-quadratic time \( \tilde{O}(n^{(9+\sqrt{177})/12}) = O(n^{1.859}) \). This problem is important due to its connections with other problems like histogram indexing and necklace alignment [ACLL14, CL15, BCD+06].

1.1 Our results

The main result of this paper is a faster algorithm for min-plus matrix product in the monotone setting.

**Theorem 1.1.** There is a randomized algorithm that computes min-plus product \( A \star B \) with expected running time \( \tilde{O}(n^{(3+\omega)/2}) \), where \( A \) is an \( n \times n \) integral matrix while \( B \) is an \( n \times n \) monotone matrix.

This improves on the previous upper bound of \( \tilde{O}(n^{(12+\omega)/5}) \) [GWX21]; as a corollary, by a reduction from the bounded-difference setting to the monotone setting, this also implies that min-plus matrix product between two bounded-difference matrices can be computed in time \( \tilde{O}(n^{(3+\omega)/2}) \), which improves upon the recent upper bound of \( \tilde{O}(n^{2+\omega/3}) \) [CDX22].

By adapting our techniques to the monotone min-plus convolution problem, we can achieve the following result:

**Theorem 1.2.** There is a randomized algorithm that computes min-plus convolution between two monotonically increasing integral sequences \( A, B \), where entries of \( A, B \) are nonnegative integers bounded by \( O(n) \), and the expected running time of this algorithm is \( \tilde{O}(n^{1.5}) \).

In the appendix, we also generalize Theorem 1.1 to column-monotone \( B \):

**Theorem 1.3.** There is a randomized algorithm that computes min-plus product \( A \star B \) with expected running time \( \tilde{O}(n^{(3+\omega)/2}) \), where \( A \) is an \( n \times n \) integral matrix while \( B \) is an \( n \times n \) column-monotone matrix.

Since \((A \star B)^T = B^T \star A^T\), these also solve the case that \( A \) is row-monotone or column-monotone and \( B \) is arbitrary.

1.2 Technical overview

In this subsection, we take an overview of our algorithm for monotone matrix min-plus product. The basic algorithmic framework follows the main idea of the previous work [CDX22] but with some important modification so that it can achieve a running time of \( \tilde{O}(n^{2+\omega/3}) \) for monotone min-plus product instead of bounded-difference min-plus product. To push it down to \( \tilde{O}(n^{3+\omega/2}) \) as stated in Theorem 1.1, we need to follow a certain recursive paradigm. For simplicity, let us assume for now that \( \omega = 2 \).

**The basic algorithm.** Similar to [GWX21], as the first step we take the approximation matrices \( \tilde{A}, \tilde{B} \) of the input \( A, B \), which are defined as \( \tilde{A}_{i,j} = \lfloor A_{i,j}/n^{1/3} \rfloor \) and \( \tilde{B}_{i,j} = \lfloor B_{i,j}/n^{1/3} \rfloor \), respectively, and then compute \( \tilde{C} = \tilde{A} \star \tilde{B} \) using an elementary combinatorial method which takes time \( \tilde{O}(n^{8/3}) \). (See Section 3.1.)

The approximation matrix \( \tilde{C} \) gives a necessary condition for witness indices \( k \) such that \( A_{i,k} + B_{k,j} = C_{i,j} \); if the equality holds, then it must be the case that \( \tilde{A}_{i,k} + \tilde{B}_{k,j} - \tilde{C}_{i,j} = O(1) \). Using this fact, build the following two polynomial matrices \( A(x,y), B(x,y) \) on variables \( x, y \):

\[
A_{i,k}(x,y) = x^{A_{i,k}-n^{1/3}} \tilde{A}_{i,k} \cdot y^{\tilde{A}_{i,k}}
\]
\[ B_{k,j}(x,y) = x^{B_{k,j} - n^{1/3}} B_{k,j} . y^{\hat{B}_{k,j}} \]

Suppose we can directly compute \( C(x, y) = A(x, y) \cdot B(x, y) \) under the standard notion of \((+, \times)\) of matrix product. Then, to search for the true value \( C_{i,j} = \text{min}_k \{ A_{i,k} + B_{k,j} \} \), we only need to look at terms \( x^c y^d \) of polynomial \( C_{i,j}(x, y) \) such that \( |d - \hat{C}_{i,j}| = O(1) \), and determine \( C_{i,j} \) to be the minimum over all values of \( c + n^{1/3} d \).

Unfortunately, computing \( C(x, y) = A(x, y) \cdot B(x, y) \) is very costly in general since the degrees of \( y \) can be very large. To reduce the \( y \)-degrees, the idea is to take \( p \)-modulo on the exponent of \( y \), where \( p = \Theta(n^{1/3}) \) is a random prime number. Formally, construct two polynomial matrices \( A^p(x, y), B^p(x, y) \) as following:

\[
A^p_{i,k}(x, y) = x^{A_{i,k} - n^{1/3}} \tilde{A}_{i,k} . y^{\tilde{A}_{i,k}} \mod p
\]
\[
B^p_{k,j}(x, y) = x^{B_{k,j} - n^{1/3}} \hat{B}_{k,j} . y^{\hat{B}_{k,j}} \mod p
\]

In this way, matrix product \( C^p = A^p \cdot B^p \) only requires running time \( \tilde{O}(n^{8/3}) \). The problem with this approach is that, when we go over all the terms \( x^c y^d \) of polynomial \( C_{i,j}(x, y) \) such that \( |d - \hat{C}_{i,j} \mod p| = O(1) \), \( c + n^{1/3} d \) might be an underestimate of \( C_{i,j} \); in fact, it could be the case that for some index \( k \), we have:

\[
c = A_{i,k} - n^{1/3} \cdot \tilde{A}_{i,k} + B_{k,j} - n^{1/3} \cdot \hat{B}_{k,j}
\]

\[
d \equiv \tilde{A}_{i,k} + \hat{B}_{k,j} \mod p
\]

\[
d \neq \tilde{A}_{i,k} + \hat{B}_{k,j}
\]

To resolve this issue, we should first enumerate all triples \( i, j, k \) such that \( d \equiv \tilde{A}_{i,k} + \hat{B}_{k,j} \mod p \) and \( d \neq \tilde{A}_{i,k} + \hat{B}_{k,j} \), and then subtract the erroneous terms \( x^c y^d \) from \( C_{i,j}(x, y) \). To upper bound the total running time, the key point is that when \( p \) is a random prime, the probability that \( d \equiv \tilde{A}_{i,k} + \hat{B}_{k,j} \mod p \) is at most \( \tilde{O}(1/p) \) when \( d \neq \tilde{A}_{i,k} + \hat{B}_{k,j} \), and therefore the expected number of erroneous terms is bounded by \( \tilde{O}(n^{8/3}/p) = \tilde{O}(n^{8/3}) \).

**Improvement by recursion.** To push the upper bound exponent from 8/3 to 2.5, we again follow the idea in [CDX22] of using recursions. Roughly speaking, we will apply a numerical scaling technique on the input matrices \( A, B \), and the key technical point is that throughout different numerical scales we need to carefully maintain all erroneous terms.

More specifically, take a random prime \( p \) in \([n^{0.5}, 2n^{0.5}]\), and define \( A^{(l)}_{i,j} = [(A_{i,j} \mod p)/2^l] \), \( B^{(l)}_{i,j} = [(B_{i,j} \mod p)/2^l] \), \( C^{(l)} = [(C_{i,j} \mod p)/2^l] \), then we will iteratively compute all \( C^{(l)} \) with \( l = h, h - 1, h - 2, \cdots, 0 \), for some parameter \( h \); note that in general \( C^{(l)} \neq A^{(l)} \cdot B^{(l)} \), so computing \( C^{(l)} \) would also require information from the original input matrices \( A, B \). Once we have \( C^{(0)} = C \mod p \), we can deduce the true value of \( C \) from the approximation matrix \( C^* = A^* \cdot B^* \), where \( A^*_{i,j} = [A^{(l)}_{i,j}/p] \) and \( B^*_{i,j} = [B^{(l)}_{i,j}/p] \); note that computing \( C^* \) takes time \( \tilde{O}(n^{2.5}) \).

To compute \( C^{(l)} \), the algorithm uses \( C^{(l+1)} \) as the approximation matrix. Namely, similar to the basic algorithm, let us construct two \( n \times n \) polynomial matrices \( A^p, B^p \) on variables \( x, y \) in the following way:

\[
A^p_{i,k} = x^{A^{(l)}_{i,k} - 2A^{(l+1)}_{i,k}} \cdot y^{A^{(l+1)}_{i,k}}
\]
\[
B^p_{k,j} = x^{B^{(l)}_{k,j} - 2B^{(l+1)}_{k,j}} \cdot y^{B^{(l+1)}_{k,j}}
\]

Then, compute the standard \((+, \times)\) matrix multiplication \( C^p = A^p \cdot B^p \) using fast matrix multiplication. The advantage of numerical scaling is that the degree of \( x \) is 0 or 1, so polynomial matrix multiplication only takes time \( \tilde{O}(n^{2.5}) \).
To efficiently enumerate these triples, the key idea is to maintain them iteratively for all \( d \) and then take 10 \( d \) Distribution of primes. Let \( \pi(x) \) be the prime-counting function that gives the number of primes less than or equal to \( x \). According to the famous prime number theorem [Jam03], \( \pi(x) \sim x/\ln(x) \).

### 2 Preliminaries

**Notations.** For any integers \( a, m \), let \((a \mod m)\) refer to the unique value \( b \in \{0, 1, 2, \ldots, m-1\} \) such that \( a \equiv b \mod m \). For any positive integer \( x \), \([x]\) refers to the set \( \{1, 2, 3, \ldots, x\} \). For a matrix \( A \) and a real number \( x \), \( A + x \) means adding \( x \) to every element of \( A \).

**Segment trees.** Let \( X = \{x_1, x_2, \ldots, x_N\} \) be an integral sequence of \( N \) elements which undergoes updates and queries. Each update operation specifies an interval \([i, j]\) and an integer value \( u \), then for each \( i \leq l \leq j \), \( x_l \) is updated as \( x_l \leftarrow \min\{x_l, u\} \). Each query operation inspects the current value of an arbitrary element \( x_i \). Using standard segment tree data structures [dBvKOS00], both update and query operations are supported in \( O(\log N) \) deterministic worst-case time.

**Matrix multiplication.** We denote with \( O(n^\omega) \) the arithmetic complexity of multiplying two \( n \times n \) matrices. Currently the best bound is \( \omega < 2.37286 \) [AW21, LG14, Wil12].

**Polynomial matrices.** Our algorithm will work with multivariate polynomials. For bivariate polynomials on variables \( x, y \), suppose the maximum degrees of \( x, y \) are bounded in absolute value by \( d_1, d_2 \), respectively (we allow their degrees to be negative). Given two polynomials \( p, q \in \mathbb{Z}[x, y] \), we can add and subtract \( p, q \) in \( O(d_1d_2) \) time, and multiply \( p, q \) in \( \tilde{O}(d_1d_2) \) time using fast-Fourier transformations [SS71]. Similar bounds hold for polynomials on three variables \( x, y, z \) as well.

We will also work with polynomial matrices from \( (\mathbb{Z}[x, y])^{n \times n} \). Products between two matrices in \( (\mathbb{Z}[x, y])^{n \times n} \) can be performed as usual, but since each arithmetic operation takes time \( \tilde{O}(d_1d_2) \), the cost of matrix multiplication takes time \( \tilde{O}(d_1d_2n^{\omega}) \). To do this, we can reduce it to multiplication of polynomial univariate matrices: replace \( y = x^{10d_1} \) and multiply the two univariate matrices, and then take 10\( d_1 \)-modulo on the degrees to recover the original degrees of \( x, y \) of each element.

**Distribution of primes.** Let \( \pi(x) \) be the prime-counting function that gives the number of primes less than or equal to \( x \). According to the famous prime number theorem [Jam03], \( \pi(x) \sim x/\ln(x) \).
As a corollary, for any large enough integer $N$, the number of primes in the range $[N, 2N]$ is at least $\Omega(N/\log N)$.

**Assumptions and Reductions.** When computing the min-plus product of $A$ and $B$, it is easy to see the following operations will not affect the complexity of computation:

(a) We can add the same value to all elements in a row of $A$ or add the same value to all elements in a column of $B$. To recover the original result $A \star B$ from the new result $C$, simply subtract the same value in the corresponding row of $C$ or subtract the same value in the corresponding column of $C$, resp.

(b) We can add the same value $\delta$ to all elements in $i$-th column of $A$ and subtract $\delta$ from all elements in $i$-th row of $B$. The min-plus product remain unchanged.

(c) If $B$ is column-monotone, we can make $A$ row-monotone (reverse order), since when $B_{k,j} \leq B_{k+1,j}$, if $A_{i,k} < A_{i,k+1}$, then $A_{i,k+1} + B_{k+1,j}$ cannot be a candidate of $C_{i,j}$, so we can make $A_{i,k+1} \leftarrow A_{i,k}$.

(d) [GWX21] If $B$ is $\delta$-row-bounded-difference, that is, $|B_{i,j} - B_{i,j+1}| \leq \delta$, then we can add $j \cdot \delta$ to the $j$-th column of $B$ to make $B$ row-monotone, so row-bounded-difference can be reduced to row-monotone. Similarly, if $B$ is $\delta$-column-bounded-difference, it can be reduced to column-monotone (with the change of $A$ by [b]).

(e) If all elements in $B$ are between $0$ and $c \cdot n$ for some constant $c$, by [a] we can adjust rows of $A$ so that the first column of $A$ are all set to $c \cdot n$, then all elements of $A$ can be made in the range $[0, 2c \cdot n]$.

Also, it is easy to get the following fact:

**Fact 1.** In $C = A \star B$, if $B$ is row-monotone, then $C$ is also row-monotone.

From [d] we can reduce $A \star B$ for any $A, B$ to the case that $B$ is row-monotone or column-monotone without $O(n)$-bound, so the general case of $B$ monotone is APSP-hard [WW10]. Thus, from [e] we only consider the case that $B$ is row-monotone or column-monotone and all elements in $A$ and $B$ are nonnegative integers bounded by $O(n)$. In this paper, monotone matrices are defined to have this element bound of $O(n)$ as in Definition 1.2.

### 3 Monotone min-plus product

#### 3.1 Basic Algorithm

In this section we prove Theorem 1.1 that is, $B$ is row-monotone. Take a constant parameter $\alpha \in (0, 1)$ which is to be determined in the end; for convenience let us assume $n^\alpha$ is an integer. The algorithm consists of three phases.

**Approximation.** Define two $n \times n$ integer matrices $\tilde{A}, \tilde{B}$ such that $\tilde{A}_{i,j} = \lfloor A_{i,j}/n^\alpha \rfloor$, $\tilde{B}_{i,j} = \lfloor B_{i,j}/n^\alpha \rfloor$. Therefore, $\tilde{B}$ is an integer matrix whose entries are bounded by $O(n^{1-\alpha})$, and each row of $\tilde{B}$ is non-decreasing.

Next, compute the approximation matrix $\tilde{C} = \tilde{A} \star \tilde{B}$ in the following way. Initialize each entry of $\tilde{C}$ to be $\infty$, and maintain each row of $\tilde{C}$ using a segment tree that supports interval updates. Then, for every pair of indices $i, k \in [n]$, run the following iterative procedure that scans the $k$-th row of $\tilde{B}$. 

5
Starting with index \( j = 1 \), find the largest index \( j \leq j_1 \leq n \) such that \( \tilde{B}_{k,j} = \tilde{B}_{k,j+1} = \cdots = \tilde{B}_{k,j_1} \) using binary search. Then, update all elements \( \tilde{C}_{i,l} \leftarrow \min \{ \tilde{C}_{i,l}, \tilde{A}_{i,k} + \tilde{B}_{k,l} \} \) for all \( j \leq l \leq j_1 \) using the segment tree data structure; notice that this operation is legal since all \( \tilde{B}_{k,l} \) are equal when \( j \leq l \leq j_1 \). After that, set \( j \leftarrow j_1 + 1 \) and repeat until \( j > n \).

**Polynomial matrix multiplication.** Uniformly sample a random prime number \( p \) in the range \([n^\alpha, 2n^\alpha] \). Construct two polynomial matrices \( A^p \) and \( B^p \) on variables \( x, y \) in the following way:

\[
A^p_{i,k} = x^{\tilde{A}_{i,k} - n^\alpha} \tilde{A}_{i,k} \mod p
\]

\[
B^p_{k,j} = x^{\tilde{B}_{k,j} - n^\alpha} \tilde{B}_{k,j} \mod p
\]

Then, compute the standard (+, \times) matrix multiplication \( C^p = A^p \cdot B^p \) using fast matrix multiplication algorithms.

**Subtracting erroneous terms.** The last phase is to extract the true values \( C_{i,j} \)'s from \( \tilde{C} \) and \( C^p \). The algorithm iterates over all offsets \( b \in \{0, 1, 2\} \), and computes the set \( T_b \subseteq [n]^3 \) of all triples of indices \((i, j, k)\) such that \( A_{i,k} + \tilde{B}_{k,j} \neq \tilde{C}_{i,j} + b \) but \( \tilde{A}_{i,k} + \tilde{B}_{k,j} \equiv \tilde{C}_{i,j} + b \mod p \); in the running time analysis, we will show that \( T_b \) can be computed in time \( \tilde{O}(|T_b| + n^{3-\alpha}) \).

For each pair of indices \( i, j \in [n] \), collect all the non-zero monomials \( \lambda x^c y^d \) (for some integer \( \lambda \)) of \( C^p_{i,j} \) such that

\[
d \equiv \tilde{C}_{i,j} + b \mod p
\]

and let \( C^p_{i,j,b}(x) \) be the sum of all such terms \( \lambda x^c \). Next, compute a polynomial

\[
R^p_{i,j,b}(x) = \sum_{(i,j,k) \in T_b} x^{\tilde{A}_{i,k} - n^\alpha \tilde{A}_{i,k} + \tilde{B}_{k,j} - n^\alpha \tilde{B}_{k,j} + n^\alpha C_{i,j} - n^\alpha \tilde{C}_{i,j} + \tilde{B}_{k,j} - \tilde{C}_{i,j}}
\]

Finally, let \( s_{i,j,b} \) be the minimum degree of \( x \) of the polynomial \( C^p_{i,j,b}(x) - R^p_{i,j,b}(x) \), and compute a candidate value \( c_{i,j,b} = n^\alpha (\tilde{C}_{i,j} + b) + s_{i,j,b} \). Ranging over all integer offsets \( b \in \{0, 1, 2\} \), take the minimum of all candidate values and output as \( C_{i,j} = \min_{0 \leq b \leq 2} \{ c_{i,j,b} \} \).

### 3.1.1 Proof of correctness

**Lemma 3.1.** For any triple \((i, j, k) \in [n]^3\) such that \( A_{i,k} + B_{k,j} = C_{i,j} \), we have

\[
0 \leq \tilde{A}_{i,k} + \tilde{B}_{k,j} - \tilde{C}_{i,j} \leq 2
\]

**Proof.** Clearly \( \tilde{A}_{i,k} + \tilde{B}_{k,j} - \tilde{C}_{i,j} \geq 0 \), so we only need to focus on the second inequality.

Suppose \( \tilde{C}_{i,j} = \tilde{A}_{i,l} + \tilde{B}_{l,j} \) for some \( l \). Then, by definition of \( \tilde{A}, \tilde{B} \), we have:

\[
n^\alpha \tilde{C}_{i,j} = n^\alpha \tilde{A}_{i,l} + n^\alpha \tilde{B}_{l,j} \geq A_{i,l} + B_{l,j} - 2n^\alpha \geq C_{i,j} - 2n^\alpha
\]

\[
= A_{i,k} + B_{k,j} - 2n^\alpha \geq n^\alpha \tilde{A}_{i,k} + n^\alpha \tilde{B}_{k,j} - 2n^\alpha
\]

Hence, \( \tilde{A}_{i,k} + \tilde{B}_{k,j} - \tilde{C}_{i,j} \leq 2 \). \( \square \)

Next we argue that our algorithm correctly computes all entries \( C_{i,j} \). Let \( l \) be the index such that \( C_{i,j} = A_{i,l} + B_{l,j} \). By the above lemma, there exists an integer offset \( b \in \{0, 1, 2\} \) such that
\[ \tilde{A}_{i,l} + \tilde{B}_{l,j} = \tilde{C}_{i,j} + b. \] Therefore, by construction of polynomial matrices \( A^p, B^p \), we have:

\[
C^p_{i,j,b}(x) = \sum_{k|\tilde{A}_{i,k} + B_{k,j} = \tilde{C}_{i,j} + b} x^{A_{i,k} - n^\alpha \tilde{A}_{i,k} + B_{k,j} - n^\alpha \tilde{B}_{k,j}}
+ \sum_{k|(\tilde{A}_{i,k} + B_{k,j} \neq \tilde{C}_{i,j} + b) \land (\tilde{A}_{i,k} + B_{k,j} \equiv \tilde{C}_{i,j} + b \mod p)} x^{A_{i,k} - n^\alpha \tilde{A}_{i,k} + B_{k,j} - n^\alpha \tilde{B}_{k,j}}
= \sum_{k|\tilde{A}_{i,k} + B_{k,j} = \tilde{C}_{i,j} + b} x^{A_{i,k} - n^\alpha \tilde{A}_{i,k} + B_{k,j} - n^\alpha \tilde{B}_{k,j}} + \sum_{(i,j,k) \in T_b} x^{A_{i,k} - n^\alpha \tilde{A}_{i,k} + B_{k,j} - n^\alpha \tilde{B}_{k,j}}
= x^{-n^\alpha(\tilde{C}_{i,j} + b)} \cdot \sum_{k|\tilde{A}_{i,k} + B_{k,j} = \tilde{C}_{i,j} + b} x^{A_{i,k} + B_{k,j}} + R^p_{i,j,b}(x)
\]

Therefore,

\[ x^{-n^\alpha(\tilde{C}_{i,j} + b)} \sum_{k|\tilde{A}_{i,k} + B_{k,j} = \tilde{C}_{i,j} + b} x^{A_{i,k} + B_{k,j}} = C^p_{i,j,b}(x) - R^p_{i,j,b}(x) \]

Since \( \tilde{A}_{i,l} + \tilde{B}_{l,j} = \tilde{C}_{i,j} + b \), we can extract \( C_{i,j} \) from terms of \( C^p_{i,j,b}(x) - R^p_{i,j,b}(x) \). In the other way, every nonzero term \( C^p_{i,j,b}(x) - R^p_{i,j,b}(x) \) corresponds to a sum of \( \tilde{A}_{i,k} + \tilde{B}_{k,j} \), which is at least \( C_{i,j} \).

### 3.1.2 Running time analysis

**Lemma 3.2.** Computing the approximation matrix \( \tilde{C} \) takes time \( \tilde{O}(n^{3-\alpha}) \).

**Proof.** For any pair of \( i, k \), the algorithm iteratively increases index \( j \) and apply update operations on the segment tree data structure. Since elements of \( B \) are bounded by \( O(n) \), the total number of different values on the \( k \)-th row of \( \tilde{B} \) is at most \( O(n^{1-\alpha}) \). Therefore, the number of iterations over \( j \) is at most \( O(n^{1-\alpha}) \) as well. Hence, the running time of this phase is \( \tilde{O}(n^{3-\alpha}) \). \( \square \)

As for polynomial matrix multiplication, by definition the \( x \)-degree and \( y \)-degree of \( A^p, B^p \) are both bounded by \( O(n^\alpha) \) in absolute value, so the matrix multiplication takes time \( O(n^{\omega + 2\alpha}) \).

**Lemma 3.3.** The triple set \( T_b \) can be computed in time \( \tilde{O}(|T_b| + n^{3-\alpha}) \).

**Proof.** Fix any pair of \( i, k \), we try to find all \( j \) such that \( (i, j, k) \in T_b \). By Fact [M] \( \tilde{B} \) and \( \tilde{C} \) are both row-monotone, so we can divide the \( k \)-th row of \( \tilde{B} \) and \( i \)-th row of \( \tilde{C} \) into at most \( O(n^{1-\alpha}) \) consecutive intervals, such that entries in each interval are all equal. So there are \( O(n^{1-\alpha}) \) intervals \([j_0, j_1]\) such that for all \( j \in [j_0, j_1] \), \( \tilde{A}_{i,k} + \tilde{B}_{j,k} \) and \( \tilde{C}_{i,j} \) are fixed. Therefore, as the total number of such row intervals is bounded by \( O(n^{3-\alpha}) \), the total running time becomes \( \tilde{O}(|T_b| + n^{3-\alpha}) \). \( \square \)

By the above lemma, the subtraction phase takes time \( \tilde{O}(|T_b| + n^{3-\alpha}) \) as well. So it suffices to bound the size of \( T_b \). For any \( (i, j, k) \in [n]^3 \) such that \( \tilde{A}_{i,k} + \tilde{B}_{j,k} \neq \tilde{C}_{i,j} + b \) since \( |\tilde{A}_{i,k} + \tilde{B}_{j,k} - \tilde{C}_{i,j} - b| \) is bounded by \( O(n) \), there are at most \( O(1/\alpha) = O(1) \) different primes in \([n^{\alpha}, 2n^{\alpha}]\) that divides \( \tilde{A}_{i,k} + \tilde{B}_{j,k} - \tilde{C}_{i,j} - b \). Since \( p \) is a uniformly random prime in the range \([n^{\alpha}, 2n^{\alpha}]\), the probability that \( \tilde{A}_{i,k} + \tilde{B}_{j,k} - \tilde{C}_{i,j} - b \) can be divided by \( p \) is bounded by \( \tilde{O}(n^{-\alpha}) \). Hence, by linearity of expectation, we have \( \mathbb{E}_p[|T_b|] \leq \tilde{O}(n^{3-\alpha}) \).

Throughout all three phases, the expected running time of our algorithm is bounded by \( \tilde{O}(n^{3-\alpha} + n^{\omega + 2\alpha}) \). Taking \( \alpha = 1 - \omega/3 \), the running time becomes \( \tilde{O}(n^{2+\omega/3}) \).
### 3.2 Recursive Algorithm

Let \( \alpha \in (0, 1) \) be a constant parameter to be determined later, and pick a uniformly random prime number \( p \) in the range of \([40n^\alpha, 80n^\alpha]\). Without loss of generality, let us assume that \( n \) is a power of 2. Next we make the following assumption about elements in \( A \) and \( B \):

**Assumption 3.1.** For every \( i, j \), either \((A_{i,j} \mod p) < p/3 \) or \( A_{i,j} = +\infty \). For every \( B_{i,j} \), \((B_{i,j} \mod p) < p/3 \). And each row of \( B \) is monotone.

**Lemma 3.4.** The general computation of \( A \ast B \) where \( B \) is row-monotone can be reduced to a constant number of computations of \( A^t \ast B^t \), where all of \( A^t, B^t \)’s satisfy Assumption 3.1.

**Proof.** The idea is very simple: for every element \( A_{i,j} \),

- if \((A_{i,j} \mod p) < p/3 \), \( A'_{i,j} = A_{i,j}, A''_{i,j} = A'''_{i,j} = +\infty \)
- if \( p/3 < (A_{i,j} \mod p) < 2p/3 \), \( A''_{i,j} = A_{i,j}, A'_{i,j} = A'''_{i,j} = +\infty \)
- if \((A_{i,j} \mod p) > 2p/3 \), \( A'''_{i,j} = A_{i,j}, A'_{i,j} = A''_{i,j} = +\infty \)

When we try to define \( B' \), \( B'' \) and \( B''' \) similarly, to make them still row-monotone, we need to fill the “blanks” with appropriate numbers.

- if \((B_{i,j} \mod p) < p/3 \), let \( B'_{i,j} = B_{i,j} \) and \( B''_{i,j} = p \cdot (B_{i,j} / p) \) if \( B_{i,j} / p \) is finite, \( B'''_{i,j} = B_{i,j} / p + \lceil p/3 \rceil \) if \( B_{i,j} / p \) is not finite,
- if \( p/3 < (B_{i,j} \mod p) < 2p/3 \), let \( B''_{i,j} = B_{i,j} \) and \( B'_{i,j} = p \cdot (B_{i,j} / p + 1) \), \( B'''_{i,j} = p \cdot (B_{i,j} / p + 1) + \lceil p/3 \rceil \)
- if \((B_{i,j} \mod p) > 2p/3 \), let \( B'''_{i,j} = B_{i,j} \) and \( B'_{i,j} = p \cdot (B_{i,j} / p + 1) \), \( B''_{i,j} = p \cdot (B_{i,j} / p + 1) + \lceil p/3 \rceil \)

We can see each pair of \( A^* \) and \( B^* \), where \( A^* \in \{A', A'' - \lceil p/3 \rceil, A''' - \lceil 2p/3 \rceil \} \), \( B^* \in \{B', B'' - \lceil p/3 \rceil, B''' - \lceil 2p/3 \rceil \} \), all satisfy Assumption 3.1, so we compute \( C^* = \min_{B^* \in \{B', B'', B''' \}} \{A^* \ast B^* \} \) (element-wise minimum). Since elements in \( B', B'', B''' \) become no smaller than the corresponding ones in \( B \), similarly for \( A', A'', A''' \), so \( C^* \geq C_i \). But for the \( k \) satisfying \( A_{i,k} + B_{k,j} = C_{i,j} \), \( A_{i,j} \) and \( B_{i,j} \) must be in one of the 9 pairs, so \( C^*_{i,j} = C_{i,j} \).

Define integer \( h \) such that \((2^h - 1) / 2^h \leq p < 2^h \). For each integer \( 0 \leq l \leq h \), let \( A^{(l)} \) be the \( n \times n \) matrix defined as \( A^{(l)}_{i,j} = \left\lfloor \frac{A_{i,j} \mod p}{2^l} \right\rfloor \) if \( A_{i,j} \) is finite, otherwise \( A^{(l)}_{i,j} = +\infty \), similarly define matrix \( B^{(l)} = \left\lfloor \frac{B_{i,j} \mod p}{2^l} \right\rfloor \).

Define \( A^* \) and \( B^* \) as \( A^*_{i,j} = \left\lfloor A_{i,j} / p \right\rfloor \) and \( B^*_{i,j} = \left\lfloor B_{i,j} / p \right\rfloor \). We use the segment tree structure to calculate \( C^* = A^* \ast B^* \) in \( \tilde{O}(n^{3-\alpha}) \) time. By Assumption 3.1 \( C^*_{i,j} = \left\lfloor C_{i,j} / p \right\rfloor \) if \( C_{i,j} \) is finite.

We will recursively calculate \( C^{(l)} \) for \( l = h, h-1, \ldots, 0 \). Intuitively, \( C^{(l)}_{i,j} \) is the approximate result obtained from \( A^{(l)}_{i,k} \) and \( B^{(l)}_{k,j} \) for those \( k \) satisfying \( C^*_{i,j} = A^*_{i,k} + B^*_{k,j} \). If \( C_{i,j} \) is finite, \( C^{(l)} \) will satisfy that

\[
(1) \quad \left( \frac{(C_{i,j} \mod p) - 2^l}{2^l} \right) \leq C^{(l)}_{i,j} \leq \left( \frac{(C_{i,j} \mod p) + 2^l}{2^l} \right)
\]

(2) If \( C^*_{i,j_0} = C^*_{i,j_1} \) for \( j_0 < j_1 \), the elements in \( C^{(l)}_{i,j_0}, \ldots, C^{(l)}_{i,j_1} \) are monotonically non-decreasing.
(Note that $C^{(l)}$ is not necessarily equal to $A^{(l)} \ast B^{(l)}$.) In the end when $l = 0$ we can get the matrix

$C^{(0)}_{i,j} = C_{i,j} \mod p$, by the procedure of recursion. Thus we can calculate the exact value of $C_{i,j}$

by the result of $C_{i,j} \mod p$.

We can see all elements in $A^{(l)}, B^{(l)}, C^{(l)}$ are non-negative integers at most $O(n^\alpha/2^l)$ or infinite.

From $B$ is row-monotone and property (2) of $C^{(l)}$, every row of $B^{(l)}, C^{(l)}$ composed of $O(n/2^l)$

intervals, where all elements in each interval are the same. Define a segment as:

**Definition 3.1.** A segment $(i, k, [j_0, j_1])$ w.r.t. $B^{(l)}$ and $C^{(l)}$, where $i, k, j_0, j_1 \in [n]$ and $j_0 \leq j_1$,

satisfies that for all $i, k, j_0, j_1 \in [n]$ and $j_0 \leq j_1$, satisfies that for all $j_0 \leq j \leq j_1$, $B^{(l)}_{k,j} = B^{(l)}_{k,j_0}$, $B^{*}_{k,j} = B^{*}_{k,j_0}$ and $C^{(l)}_{i,j} = C^{(l)}_{i,j_0}$, $C^{*}_{i,j} = C^{*}_{i,j_0}$.

Then each pair of rows of $B^{(l)}, C^{(l)}$ can be divided into $O(n/2^l)$ segments.

We maintain the auxiliary sets $T^{(l)}_b$ for $-10 \leq b \leq 10$ throughout the algorithm, where the set $T^{(l)}_b$

consists of all the segments $(i, k, [j_0, j_1])$ w.r.t. $B^{(l)}$ and $C^{(l)}$ satisfying: (So this holds for all

$j \in [j_0, j_1]$.)

$A_{i,k}$ is finite and $A_{i,k}^* + B_{k,j_0}^* \neq C_{i,j_0}^*$ and $A_{i,k}^{(l)} + B_{k,j_0}^{(l)} = C_{i,j_0}^{(l)} + b$

The algorithm proceeds as:

- In the first iteration $l = h$, we want to calculate $C^{(h)}_{i,j}$. However since $p < 2^h$, $A^{(h)}, B^{(h)}, C^{(h)}$

are zero matrices, so $T^{(h)}_0$ includes all segments $(i, k, [j_0, j_1])$ where $A_{i,k}$ finite and $A_{i,k}^* + B_{k,j_0}^* \neq C_{i,j_0}^*$.

And $T^{(h)}_b = \emptyset$ ($b \neq 0$). Since the number of segments in a row w.r.t. $B^{(h)}, C^{(h)}$ is

$O(n^{1-\alpha})$, $|T^{(h)}_b| = O(n^{3-\alpha})$.

- For $l = h - 1, \cdots, 0$, we first compute $C^{(l)}$ with the help of $T^{(l+1)}_b$, then construct $T^{(l)}_b$

from $T^{(l+1)}_b$. By Lemma 3.6 that $\bigcup_{i=10}^{10} T^{(l+1)}_i \subseteq \bigcup_{i=10}^{10} T^{(l+1)}_i$, we can search the shorter segments

contained in $T^{(l+1)}_b$ to find $T^{(l)}_b$. By Lemma 3.7, $|T^{(l)}_b|$ is always bounded by $O(n^{3-\alpha})$.

Each iteration has three phases:

**Polynomial matrix multiplication.** Construct two polynomial matrices $A^p$ and $B^p$ on variables

$x, y$ in the following way: When $A_{i,k}$ finite,

$$A_{i,k}^p = x^{A_{i,k}^{(l)} - 2A_{i,k}^{(l+1)}} \cdot y^{A_{i,k}^{(l+1)}}$$

Otherwise $A_{i,k}^p = 0$, and:

$$B_{k,j}^p = x^{B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}} \cdot y^{B_{k,j}^{(l+1)}}$$

Then, compute the standard $(+, \times)$ matrix multiplication $C^p = A^p \cdot B^p$ using fast matrix multiplication

algorithms. Note that $A_{i,j}^{(l)} - 2A_{i,j}^{(l+1)}, B_{i,j}^{(l)} - 2B_{i,j}^{(l+1)}$ are 0 or 1, so the degree of $x$ terms are

0 or 1. This phase runs in time $O(n^{\omega+\alpha})$.

**Subtracting erroneous terms.** This phase is to extract the true values $C_{i,j}$’s from $C_{i,j}^{(l+1)}$. The

algorithm iterates over all offsets $-10 \leq b \leq 10$, and enumerates all the segments in $T^{(l+1)}_b$.

For each pair of indices $i, j \in [n]$, if $C_{i,j}^p = 0$ then $C_{i,j}^{(l)} = +\infty$, otherwise collect all the monomials

$\lambda x^d y^d$ of $C_{i,j}^p$ such that

$$d = C_{i,j}^{(l+1)} + b$$
Proof. For all \( \text{Lemma 3.5.} \) In each iteration \( C \) will show that the properties of \( \text{Lemma 3.6.} \) We have \( \text{Proof.} \) Finally, let \( s_{i,j,b} \) be the minimum degree of \( x \) in the polynomial \( C_{i,j,b}(x) - R_{i,j,b}^p(x) \), and compute a candidate value \( c_{i,j,b} = 2d + s_{i,j,b} \). (If \( s_{i,j,b} = 0 \) then \( c_{i,j,b} = +\infty \).) Ranging over all integer offsets \( -10 \leq b \leq 10 \), take the minimum of all candidate values and output as \( C_{i,j}^{(l)} = \min_{-10 \leq b \leq 10} \{c_{i,j,b}\} \). This phase runs in time \( \tilde{O}(n^{3-\alpha} + n^{2+\alpha}) \), since every segment \((i,k,[j_0,j_1]) \in T_b^{(l+1)} \) contains at most two different \( B_{k,j}^{(l)} \), thus also two different \( R_{i,j,b}^p(x) \), so we can use a segment tree to compute all of \( C_{i,j,b}(x) - R_{i,j,b}^p(x) \) in \( \tilde{O}(n^{2+\alpha} + |T_b^{(l+1)}|) \) time.

Computing Triples \( T_b^{(l)} \). Since \( B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)} \) and \( C_{i,j}^{(l)} - 2C_{i,j}^{(l+1)} \) are both between 0 and a constant (see Lemma 3.5), so each segment w.r.t. \( B^{(l+1)}, C^{(l+1)} \) can be split into at most \( O(1) \) segments w.r.t. \( B^{(l)}, C^{(l)} \). By Lemma 3.6 we know that \( \cup_{i=10} \cup_{i=-10} T_i^{(l)} \) is contained in \( \cup_{i=10} \cup_{i=-10} T_i^{(l+1)} \), so our work here is to check the sub-segments of each segment in \( \cup_{i=10} \cup_{i=-10} T_i^{(l+1)} \) and put it into the \( T_b^{(l+1)} \) it belongs to. Each segment in \( T_b^{(l+1)} \) breaks into at most \( O(1) \) sub-segments in the next iteration, and we can use binary search to find the breaking points. This phase runs in time \( \tilde{O}(n^{3-\alpha}) \).

The expected running time of the recursive algorithm is bounded by \( \tilde{O}(n^{3-\alpha} + n^{\omega+\alpha}) \). Taking \( \alpha = (3 - \omega)/2 \), the running time becomes \( \tilde{O}(n^{3+\omega}/2) \).

3.2.1 Proof of correctness

We first prove the lemmas needed to bound the running time and show the correctness, then we will show that the properties of \( C_{i,j}^{(l)} \) are maintained in the algorithm:

Lemma 3.5. In each iteration \( l = h - 1, \cdots, 0, -7 \leq C_{i,j}^{(l)} - 2C_{i,j}^{(l+1)} \leq 8 \).

Proof. For all \( l \), we can get:

\[
\frac{(C_{i,j} \mod p)}{2^l} - 3 \leq C_{i,j}^{(l)} \leq \frac{(C_{i,j} \mod p)}{2^l} + 2
\]

and

\[
2C_{i,j}^{(l+1)} - 7 \leq 2 \left(\frac{(C_{i,j} \mod p)}{2^{l+1}}\right) - 3 \leq C_{i,j}^{(l)} \leq 2 \left(\frac{(C_{i,j} \mod p)}{2^{l+1}}\right) + 2 \leq 2C_{i,j}^{(l+1)} + 8
\]

Lemma 3.6. We have \( \cup_{i=-10}^{10} T_i^{(l)} \subseteq \cup_{i=-10}^{10} T_i^{(l+1)} \), that is, the segments we consider in each iteration must be sub-segments of the segments in the last iteration.

Proof. Segments \((i,k,[j_0,j_1]) \in T_b^{(l)} \) and \( T_b^{(l+1)} \) must satisfy \( A_{i,k}^{*} \) is finite and \( A_{i,k}^{*} + B_{k,j_0}^{*} \neq C_{i,j_0}^{*} \). By definition, \( A_{i,k}^{(l)} - 2A_{i,k}^{(l+1)} = 0 \) or \( 1 \). Similar for \( B \), and by Lemma 3.5 we have
\[ A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - C_{i,j}^{(l+1)} \geq A_{i,k}^{(l)}/2 - 1/2 + B_{k,j}^{(l)}/2 - 1/2 - C_{i,j}^{(l)}/2 - 7/2 \]
\[ \geq \frac{1}{2} \left( A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} \right) - 9/2. \]
\[ A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - C_{i,j}^{(l+1)} \leq A_{i,k}^{(l)}/2 + B_{k,j}^{(l)}/2 - C_{i,j}^{(l)}/2 + 4 \]
\[ \leq \frac{1}{2} \left( A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} \right) + 4. \]

Therefore, when \(-10 \leq A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} \leq 10,\)
\[ -10 < -10/2 - 9/2 \leq A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - C_{i,j}^{(l+1)} \leq 10/2 + 4 < 10. \]

\[ \square \]

**Lemma 3.7.** The expected number of segments in \(T_b^{(l)}\) is \(\tilde{O}(n^{3-\alpha})\).

**Proof.** When \(2^l > p/100\), the total number of segments is bounded by \(O(n^{3-\alpha})\), so next we assume that \(2^l < p/100\).

For any segment \((i, k, [j_0, j_1])\), and arbitrarily pick a \(j \in [j_0, j_1]\) where \(A_{i,k}\) is finite and \(A_{i,k}^* + B_{k,j}^* \neq C_{i,j}^*\). By Assumption 3.1, \(\pmod{p} < 2p/3\), so if \(A_{i,k}^* + B_{k,j}^* \geq C_{i,j}^* + 1,\)
\[ A_{i,k}/p + B_{k,j}/p \geq [A_{i,k}/p] + [B_{k,j}/p] \geq [C_{i,j}/p] + 1 \geq C_{i,j}/p - 2/3 + 1 = C_{i,j}/p + 1/3 \]

So \(A_{i,k} + B_{k,j} \geq C_{i,j} + p/3\). Similarly, if \(A_{i,k}^* + B_{k,j}^* \leq C_{i,j}^* - 1,\)
\[ A_{i,k}/p - 1, B_{k,j}/p - 1/3 \leq [A_{i,k}/p] + [B_{k,j}/p] \leq [C_{i,j}/p] - 1 \leq C_{i,j}/p - 1 \]

Thus we get \(|A_{i,k} + B_{k,j} - C_{i,j}| \geq p/3\) in either case.

We want to bound the probability that \((i, k, [j_0, j_1])\) appears in \(T_b^{(l)}\). By definition, this is to say that
\[ \left\lfloor \frac{A_{i,k}}{2^l} \mod p \right\rfloor + \left\lfloor \frac{B_{k,j}}{2^l} \mod p \right\rfloor = C_{i,j}^{(l)} + b. \]

So
\[ -4 < \frac{A_{i,k}}{2^l} \mod p + \frac{B_{k,j}}{2^l} \mod p - \frac{C_{i,j}}{2^l} \mod p - b \leq 4 \]

Let \(C_{i,j} = A_{i,q} + B_{q,j}\), and
\[ (A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j}) \mod p \in [2^l(b - 4), 2^l(b + 4)]. \]

That is, \(A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j}\) should be congruent to one of the \(O(2^l)\) remainders. For each possible remainder \(r \in [2^l(b - 4), 2^l(b + 4)], (|b| \leq 10),\) we have
\[ |r| \leq 14 \cdot 2^l < p/6 \leq \frac{1}{2} |A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j}|. \]

If \(A_{i,k}, A_{i,q}\) are finite and \(B_{k,j}, B_{q,j}\) are from the original \(B\) (see Lemma 3.4), \(|(A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j}) - r|\) is a positive number bounded by \(O(n)\), the number of different primes \(p \in [40n^\alpha, 80n^\alpha]\) which divides \((A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j}) - r\) can not exceed \(1/\alpha = O(1)\). In our algorithm, when we uniformly choose a prime \(p\) from \([40n^\alpha, 80n^\alpha]\), the probability that \((A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j})\) \(\mod p \equiv r\) is \(\tilde{O}\left(\frac{1}{n^\alpha}\right)\).
However in Lemma 3.4, $B_{k,j}$ and $B_{q,j}$ may be set artificially to numbers which are congruent to 0, $[p/3]$ or $[2p/3]$ modulo $p$, but finite $A_{i,k}$ and $A_{i,q}$ must come from the original $A$. For example, if $B_{k,j}$ is made congruent to $[p/3]$ module $p$ and $B_{q,j}$ is from original $B$, we want that $p$ divides $A_{i,k} - A_{i,q} - B_{q,j} - r + [p/3]$. Since 3 does not divides $p$, $3[p/3]$ is $p + 1$ or $p + 2$, so $p$ divides $3(A_{i,k} - A_{i,q} - B_{q,j} - r) + 1$ or $3(A_{i,k} - A_{i,q} - B_{q,j} - r) + 2$. The probability is still $\tilde{O}\left(\frac{1}{m}\right)$. Other cases of $B_{k,j}$ and $B_{q,j}$ can be done similarly. Since on all cases of $B_{k,j}$ and $B_{q,j}$ the conditional probability that $p$ divides $(A_{i,k} + B_{k,j} - A_{i,q} - B_{q,j}) - r$ is bounded by $\tilde{O}\left(\frac{1}{m}\right)$, the total probability is also $\tilde{O}\left(\frac{1}{m}\right)$.

Since there are $O(2^l)$ such possible remainders $r$, in expectation we have $O(2^l) \cdot O\left(\frac{n^3}{2^l}\right) \cdot \tilde{O}\left(\frac{1}{m}\right) = \tilde{O}(n^{3 - \alpha})$ segments in $T_b^{(l)}$.

Lemma 3.8. If $A_{i,k} + B_{k,j} = C_{i,j}$, then $A_{i,k}^{(l)} + B_{k,j}^{(l)} = C_{i,j}^{(l)} + b$ for some $-10 \leq b \leq 10$.

Proof. By Assumption 3.1,

$$A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} = \left\lfloor \frac{A_{i,k} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k,j} \mod p}{2^l} \right\rfloor - \left\lfloor \frac{C_{i,j} \mod p}{2^l} \right\rfloor$$

$$\leq \frac{A_{i,k} \mod p}{2^l} + \frac{B_{k,j} \mod p}{2^l} - \frac{C_{i,j} \mod p}{2^l} + 3$$

$$= (A_{i,k} + B_{k,j} - C_{i,j}) \mod p + 3 = 3,$$

$$A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} = \left\lfloor \frac{A_{i,k} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k,j} \mod p}{2^l} \right\rfloor - \left\lfloor \frac{C_{i,j} \mod p}{2^l} \right\rfloor$$

$$\geq \frac{A_{i,k} \mod p}{2^l} + \frac{B_{k,j} \mod p}{2^l} - \frac{C_{i,j} \mod p}{2^l} - 4$$

$$= (A_{i,k} + B_{k,j} - C_{i,j}) \mod p - 4 = -4.$$
polynomial matrices \( A^p, B^p \), we have:

\[
C_{i,j,b}^p(x) = \sum_{k \atop k|A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} = c_{i,j}^{(l+1)} + b} x^{|A_{i,k}^{(l)} + B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

\[
= \sum_{k \atop |(A_{i,k}^* + B_{k,j}^*) = c_{ij}^{(l+1)} \wedge (A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} = c_{i,j}^{(l+1)} + b)} x^{|A_{i,k}^{(l)} + B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

\[
+ \sum_{k \atop |(A_{i,k}^* + B_{k,j}^* \neq c_{ij}^{(l+1)} \wedge (A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} = c_{i,j}^{(l+1)} + b)} x^{|A_{i,k}^{(l)} + B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

\[
= \sum_{k \atop |(A_{i,k} + B_{k,j}^* = c_{ij}^{(l+1)} \wedge (A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} = c_{i,j}^{(l+1)} + b)} x^{|A_{i,k}^{(l)} + B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

\[
+ \sum_{k \atop (i,k,[j_0,j_1]) \in T_{i,j}^{(l+1)}} x^{|A_{i,k}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

\[
= x^{2(C_{i,j}^{(l+1)} + b)} + \sum_{k \atop |(A_{i,k} + B_{k,j}^* = c_{ij}^{(l+1)} \wedge (A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} = c_{i,j}^{(l+1)} + b)} x^{|A_{i,k}^{(l)} + B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

Since \( A_{i,q} + B_{q,j} = c_{i,j}^{(l+1)} \) and \( A_{i,q}^{(l+1)} + B_{q,j}^{(l+1)} = C_{i,j}^{(l+1)} + b \), when we extract \( A_{i,q} + B_{q,j} \) from terms of \( C_{i,j,b}^p(x) - R_{i,j,b}^p(x) \), it satisfies

\[
\frac{A_{i,q} \mod p}{2^l} + \frac{B_{q,j} \mod p}{2^l} \leq (A_{i,q} + B_{q,j}) \mod p = C_{i,j} \mod p \leq \frac{(C_{i,j} \mod p) + 2^l - 1}{2^l}
\]

\[
\frac{A_{i,q} \mod p}{2^l} + \frac{B_{q,j} \mod p}{2^l} \geq ((A_{i,q} + B_{q,j}) \mod p) - 2(2^l - 1) \geq \frac{(C_{i,j} \mod p) - 2(2^l - 1)}{2^l}
\]

Thus the term which gives \( A_{i,q} + B_{q,j} \) can give a valid \( C_{i,j} \). Also for every term which gives \( A_{i,k} + B_{k,j} \) satisfying \( A_{i,k}^* + B_{k,j}^* = c_{i,j}^{(l+1)} \) and \( A_{i,k} + B_{k,j} \geq c_{i,j} \),

\[
\frac{A_{i,k} \mod p}{2^l} + \frac{B_{k,j} \mod p}{2^l} \geq ((A_{i,k} + B_{k,j}) \mod p) - 2(2^l - 1) \geq \frac{(C_{i,j} \mod p) - 2(2^l - 1)}{2^l}
\]

So by choosing the minimum, we can get a valid \( C_{i,j}^{(l+1)} \) satisfying property (1).

To see that \( C^{(l)} \) satisfies property (2), consider \( C_{i,j_0}^* = C_{i,j_1}^* \) where \( j_0 < j_1 \). For all the \( k \) such that \( A_{i,k} + B_{k,j_1} = C_{i,j_1}^* \), \( A_{i,j_0} \leq A_{i,k} + B_{k,j_0} \leq A_{i,k}^* + B_{k,j_1}^* = C_{i,j_1}^* \), so \( B_{k,j_0}^* = B_{k,j_1}^* \) and \( B_{k,j_0} \leq B_{k,j_1} \). Thus, for term \( A_{i,k}^* + B_{k,j_1}^* \) which gives \( C_{i,j_1}^{(l+1)} \), the term with \( A_{i,k}^* + B_{k,j_0}^* \) also exist in \( C_{i,j_0}^* \) and cannot be subtracted since \( A_{i,k} + B_{k,j_0}^* = C_{i,j_0}^* \). If this result is not included in \( C_{i,j,b}^p(x) - R_{i,j,b}^p(x) \) for all \( -10 \leq b \leq 10 \), \( C_{i,j}^{(l+1)} + B_{k,j_0}^{(l+1)} - C_{i,j_0}^{(l+1)} \) is larger than 10 or less than

\[
10. A_{i,k}^{(l+1)} + B_{k,j_0}^{(l+1)} - C_{i,j_0}^{(l+1)} < -10,
\]

\[
10 > A_{i,k}^{(l+1)} + B_{k,j_0}^{(l+1)} - C_{i,j_0}^{(l+1)}
\]

\[
= \frac{A_{i,k} \mod p}{2^{l+1}} + \frac{B_{k,j_0} \mod p}{2^{l+1}} - C_{i,j_0}^{(l+1)}
\]

\[
\geq \frac{A_{i,k} \mod p}{2^{l+1}} + \frac{B_{k,j_0} \mod p}{2^{l+1}} - \frac{C_{i,j_0} \mod p}{2^{l+1}} - 4
\]
So \((A_{i,k} \mod p) + (B_{k,j_0} \mod p) - (C_{i,j_0} \mod p) < 0\), which is impossible since \(A^*_i + B^*_k = C^*_{i,j_0}\). Thus, it can only be that \(A^{(l+1)}_{i,k} + B^{(l+1)}_{k,j_1} - C^{(l+1)}_{i,j_0} > 10\), so by inductive assumption and Lemma 3.5,

\[
C^{(l)}_{i,j_1} = A^{(l)}_{i,k} + B^{(l)}_{k,j_1} > 2 \left( A^{(l+1)}_{i,k} + B^{(l+1)}_{k,j_1} \right) \geq 2 \left( A^{(l+1)}_{i,k} + B^{(l+1)}_{k,j_0} \right) > 2C^{(l+1)}_{i,j_0} + 20 \geq C^{(l)}_{i,j_0} + 12
\]

This proves property (2).

4 Monotone min-plus convolution

4.1 Basic Algorithm

In this section we prove Theorem 1.2 following the same algorithmic framework of Theorem 1.1.

The min-plus convolution \(C = A \circ B\) of two array \(A\) and \(B\) can be defined as \(C_k = \min_{i=1}^{k-1} \{A_i + B_{k-i}\}, \forall 2 \leq k \leq 2n\). Take two constant parameters \(\alpha, \beta \in (0, 1)\) which are to be determined in the end; for convenience let us assume \(n^\alpha\) is an integer.

**Approximation.** Define two integral arrays \(\tilde{A}, \tilde{B}\) such that \(\tilde{A}_i = \lfloor A_i/n^\alpha \rfloor\), \(\tilde{B}_i = \lfloor B_i/n^\alpha \rfloor\). Therefore, \(\tilde{A}, \tilde{B}\) is an integer array whose entries are bounded by \(O(n^{1-\alpha})\), and both \(\tilde{A}, \tilde{B}\) are non-decreasing.

Next, compute the approximate min-plus convolution \(\tilde{C} = \tilde{A} \circ \tilde{B}\) combinatorially. Initialize each entry of \(\tilde{C}\) to be \(\infty\), and maintain \(\tilde{C}\) using a segment tree that supports interval updates. Divide \(A\) and \(B\) into at most \(O(n^{1-\alpha})\) consecutive intervals:

\[
[n] = [1, a_2 - 1] \cup [a_2, a_3 - 1] \cup \cdots \cup [a_g, n] = [1, b_2 - 1] \cup [b_2, b_3 - 1] \cup \cdots \cup [b_h, n]
\]

such that for each \(i \in [a_i, a_{i+1} - 1]\), \(\tilde{A}_i\)'s are all equal, and for each \(j \in [b_k, b_{k+1} - 1]\), \(\tilde{B}_j\)'s are all equal (assume \(a_1 = b_1 = 1\) and \(a_{g+1} = b_{h+1} = n + 1\)). Then, to compute \(\tilde{C}\), take any pair of indices \(k, l \in [g] \times [h]\), and update \(\tilde{C}_i \leftarrow \min\{\tilde{C}_i, \tilde{A}_{a_i} + \tilde{B}_{b_l}\}\) for each index \(a_i + b_k \leq i \leq a_{i+1} + b_{k+1} - 2\) using the segment tree data structure maintained on array \(\tilde{C}\). The total time is \(\tilde{O}(n^{2-2\alpha})\).

**Polynomial multiplication.** Uniformly sample a random prime number \(p\) in the range \([n^\beta, 2n^\beta]\).

Construct two polynomial \(A^p\) and \(B^p\) on variables \(x, y, z\) in the following way:

\[
A^p(x, y, z) = \sum_{i=1}^{n} x^{A_i - n^\alpha \tilde{A}_i} y^{\tilde{A}_i} \mod p \cdot z^i
\]

\[
B^p(x, y, z) = \sum_{i=1}^{n} x^{B_i - n^\alpha \tilde{B}_i} y^{\tilde{B}_i} \mod p \cdot z^i
\]

Then, compute polynomial multiplication \(C^p(x, y, z) = A^p(x, y, z) \cdot B^p(x, y, z)\) using standard fast Fourier transform algorithms [SS71].

**Subtracting erroneous terms.** The last phase is to extract the true values \(C_i\)'s from \(\tilde{C}\) and \(C^p(x, y, z)\). The algorithm iterates over all offsets \(b \in \{0, 1, 2\}\), and computes the set \(T_b \subseteq [n]^2\) of all pairs of indices \((i, j)\) such that \(\tilde{A}_i + \tilde{B}_j \neq \tilde{C}_{i+j} + b\) but \(\tilde{A}_i + \tilde{B}_j \equiv \tilde{C}_{i+j} + b \mod p\); in the running time analysis, we will show that \(T_b\) can be computed in time \(\tilde{O}(|T_b| + n^{2-2\alpha})\).
For each index $1 \leq k \leq n$, consider the coefficient of $z^k$ in $C^p(x, y, z)$ denoted by $C_{k}^p(x, y, z)$. Enumerate all terms $\lambda x^c y^d$ of $C_{k}^p(x, y)$ such that $d \equiv \tilde{C}_k + b \mod p$, and define $C_{k, b}^p(x)$ to be the sum of all $\lambda x^c$. Next, compute a polynomial:

$$R_{k, b}^p(x) = \sum_{(i, k-i) \in T_b} x^{A_i - n^\alpha \tilde{A}_i + B_{k-i} - n^\alpha \tilde{B}_{k-i}}$$

Finally, let $s_{k, b}$ be the minimum degree of $x$ of the polynomial $C_{k, b}^p(x) - R_{k, b}^p(x)$, and compute a candidate value $n^\alpha (\tilde{C}_k + b) + s_{k, b}$ for $C_k$. Ranging over all integer offsets $b \in \{0, 1, 2\}$, take the minimum of all candidate values and output as $C_k = \min_{0 \leq b \leq 2} \{n^\alpha (\tilde{C}_k + b) + s_{k, b}\}$.

### 4.1.1 Proof of correctness

**Lemma 4.1.** For any pair of indices $1 \leq i, j \leq n$ such that $A_i + B_j = C_{i+j}$, we have:

$$0 \leq \tilde{A}_i + \tilde{B}_j - \tilde{C}_{i+j} \leq 2$$

**Proof.** Clearly $\tilde{A}_i + \tilde{B}_j - \tilde{C}_{i+j} \geq 0$ by definition of min-plus convolution. So let us only focus on the second inequality. Suppose $\tilde{C}_{i+j} = \tilde{A}_k + \tilde{B}_l$ for some indices $1 \leq k, l \leq n$ such that $k + l = i + j$. Then, by definition of $A, B$, we have:

$$n^\alpha \tilde{C}_{i+j} = n^\alpha \tilde{A}_k + n^\alpha \tilde{B}_l \geq A_k + B_l - 2n^\alpha \geq C_{i+j} - 2n^\alpha$$

$$= A_i + B_j - 2n^\alpha \geq n^\alpha \tilde{A}_i + n^\alpha \tilde{B}_j - 2n^\alpha$$

Hence, $\tilde{A}_i + \tilde{B}_j - \tilde{C}_{i+j} \leq 2$. \qed

Next, we argue that our algorithm correctly computes all entries $C_k$. Let $l$ be the index such that $C_k = A_l + B_{k-l}$. By the above lemma, there exists an integer offset $b \in [0, 2]$ such that $\tilde{A}_l + \tilde{B}_{k-l} = \tilde{C}_k + b$. Therefore, by construction of polynomial matrices $A^p, B^p$, we have:

$$C_{k, b}^p(x) = \sum_{i|\tilde{A}_i + B_{k-i} = \tilde{C}_k + b} x^{A_i - n^\alpha \tilde{A}_i + B_{k-i} - n^\alpha \tilde{B}_{k-i}}$$

$$+ \sum_{i|(\tilde{A}_i + B_{k-i} \neq \tilde{C}_k + b) \land (\tilde{A}_i + B_{k-i} \equiv \tilde{C}_k + b \mod p)} x^{A_i - n^\alpha \tilde{A}_i + B_{k-i} - n^\alpha \tilde{B}_{k-i}}$$

$$= \sum_{k|\tilde{A}_l + \tilde{B}_{k-l} = \tilde{C}_k + b} x^{A_l - n^\alpha \tilde{A}_l + B_{k-l} - n^\alpha \tilde{B}_{k-l}} + \sum_{(i, k-i) \in T_b} x^{A_i - n^\alpha \tilde{A}_i + B_{k-i} - n^\alpha \tilde{B}_{k-i}} + R_{k, b}^p(x)$$

$$= x^{-n^\alpha (\tilde{C}_k + b)} \cdot \sum_{k|\tilde{A}_l + \tilde{B}_{k-l} = \tilde{C}_k + b} x^{A_l + B_{k-l} - \tilde{C}_k + b} = C_{k, b}^p(x) - R_{k, b}^p(x).$$

Therefore, $x^{-n^\alpha (\tilde{C}_k + b)} \cdot \sum_{k|\tilde{A}_l + \tilde{B}_{k-l} = \tilde{C}_k + b} x^{A_l + B_{k-l}} = C_{k, b}^p(x) - R_{k, b}^p(x)$. Since $\tilde{A}_l + \tilde{B}_{k-l} = \tilde{C}_k + b$, we know $n^\alpha (\tilde{C}_k + b) + s_{k, b} = C_k$.

### 4.1.2 Running time analysis

By the algorithm, computing the approximation array $\tilde{C}$ takes time $\tilde{O}(n^{2-2\alpha})$. As for polynomial multiplication, by definition the $x$-degree and $y$-degree of $A^p, B^p$ are both bounded in absolute value by $O(n^{\alpha}), O(n^{\beta})$ respectively, so the polynomial multiplication takes time $O(n^{1+\alpha+\beta})$. 

15
Lemma 4.2. The set $T_b$ can be computed in time $\tilde{O}(|T_b| + n^{2-2\alpha})$.

Proof. Recall the partition of sequences $A, B$ into intervals in the first phase:

$$[n] = [1, a_2 - 1] \cup [a_2, a_3 - 1] \cup \cdots \cup [a_g, n] = [1, b_2 - 1] \cup [b_2, b_3 - 1] \cup \cdots \cup [b_h, n]$$

such that $A, B$ are all equal within each interval. Fix two intervals $[a_s, a_{s+1} - 1]$ and $[b_t, b_{t+1} - 1]$ where array $\tilde{A}$ and $\tilde{B}$ have the same value. Next, we try to find all $i \in [a_s, a_{s+1} - 1], j \in [b_t, b_{t+1} - 1]$ such that $(i, j) \in T_b$. The key advantage is that the value $\Delta = \tilde{A}_i + \tilde{B}_j$ is a fixed value for any $(i, j) \in [a_s, a_{s+1} - 1] \times [b_t, b_{t+1} - 1]$.

Now search for all indices $a_s + b_t \leq k \leq a_{s+1} + b_{t+1} - 2$ such that $\tilde{C}_k + b \neq \Delta$ while $\tilde{C}_k + b = \Delta$ mod $p$. Using standard binary search tree data structures, we can obtain the set of all such indices $K_b$ in time $\tilde{O}(|K_b|)$. Then, for each $k \in K_b$, enumerate all $(i, k-i)$ satisfying

$$\max\{a_s, k+1-b_t\} \leq i \leq \min\{a_{s+1} - 1, k-b_t\}$$

and add the pair $(i, k-i)$ to $T_b$. Ranging over all choices of intervals $[a_s, a_{s+1} - 1]$ and $[b_t, b_{t+1} - 1]$, the total time becomes $\tilde{O}(|T_b| + n^{2-2\alpha})$. □

By the above lemma, the subtraction phase takes time $\tilde{O}(|T_b| + n^{2-2\alpha})$ as well. So it suffices to bound the size of $T_b$. For any $(i, k-i)$ such that $\tilde{A}_i + \tilde{B}_{k-i} \neq \tilde{C}_k + b$, since $p$ is a uniformly random prime in the range $[n^\beta, 2n^\beta]$, the probability that $\tilde{A}_i + \tilde{B}_{k-i} - \tilde{C}_k - b$ can be divided by $p$ is bounded by $\tilde{O}(n^{-\beta})$. Hence, by linearity of expectation, $\mathbb{E}[|T_b|] \leq \tilde{O}(n^{2-\beta})$.

Throughout all three phases, the expected running time of our algorithm is bounded by $\tilde{O}(n^{2-2\alpha} + n^{1+\alpha+\beta} + n^{2-\beta})$. Taking $\alpha = 0.2, \beta = 0.4$, the running time becomes $\tilde{O}(n^{1.6})$.

4.2 Recursive Algorithm

Let $\alpha \in (0, 1)$ be a constant parameter to be determined later, and pick a uniformly random prime number $p$ in the range of $[40n^\alpha, 80n^\alpha]$. Without loss of generality, let us assume that $n$ is a power of 2. Like in Section 3.2 w.l.o.g. we make the following assumption about elements in $A$ and $B$:

Assumption 4.1. For every $i$, either $(A_i \mod p) < p/3$ or $A_i = +\infty$, and $A$ is monotone besides the infinite elements. Similar for $B$.

Lemma 4.3. The general computation of $A \circ B$ can be reduced to a constant number of computations of $A^l \circ B^l$ where all of $A^l, B^l$’s satisfy Assumption 4.1. The number of intervals of infinity in each $A^l$ and $B^l$ is bounded by $O(n^{1-\alpha})$.

Proof. We just arrange the elements of $A$ to $A', A'', A'''$ by their remainders module $p$, other elements becomes $+\infty$. It is easy to see that the number of intervals of infinity in each of $A', A'', A'''$ is bounded by $O(n^{1-\alpha})$. Similar for $B$. □

Define integer $h$ such that $2^{h-1} \leq p < 2^h$. For each integer $0 \leq l \leq h$, let $A^{(l)}$ be a sequence of length $n$ defined as $A^{(l)}_i = \lfloor \frac{A_i \mod p}{2^l} \rfloor$ if $A_i$ is finite, otherwise $A^{(l)}_i = +\infty$, similarly define sequence $B^{(l)} = \lfloor \frac{B_i \mod p}{2^l} \rfloor$.

We will recursively calculate $C^{(l)}$ for $l = h, h-1, \cdots, 0$, and if $C_i$ is finite, $C^{(l)}$ will satisfy

$$\left\lfloor \frac{(C_i \mod p) - 2(2^l - 1)}{2^l} \right\rfloor \leq C^{(l)}_i \leq \left\lfloor \frac{(C_i \mod p) + 2(2^l - 1)}{2^l} \right\rfloor$$
(Note that \(C^{(l)}\) is not necessarily equal to \(A^{(l)} \odot B^{(l)}\).) In the end when \(l = 0\) we can get the matrix \(C^{(l)}_1 = C_1 \mod p\) by the procedure of recursion. Define \(A^*\) and \(B^*\) as \(A^*_i = \lfloor A_i/p \rfloor\) and \(B^*_i = \lceil B_i/p \rceil\). We use the segment tree structure to calculate \(C^* = A^* \odot B^*\) in \(O(n^{2-2\alpha})\) time. By Assumption 4.1, \(C_i = \lfloor C_i/p \rfloor\) if \(C_i\) is finite. Thus we can calculate the exact value of \(C_i\) by the result of \(C_i \mod p\).

We can see all elements in \(A^{(l)}, B^{(l)}, C^{(l)}\) are non-negative integers at most \(O(n^{\alpha}/2^l)\) or infinite. Since \(A, B\) are monotone and by Lemma 4.3 \(A^{(l)}, B^{(l)}\) compose of \(O(n/2^l)\) intervals, where all elements in each interval are the same. Define a segment as:

**Definition 4.1.** A segment \(([i_0, i_1], k)\) w.r.t. \(A^{(l)}\) and \(B^{(l)}\), where \(i_0, i_1, k \in [n]\) and \(i_0 \leq i_1\), satisfies that for all \(i_0 \leq i \leq i_1\), \(A_i, B_{k-i}\) are finite, and \(A_i = A_{i_0}^{(l)}, A^*_i = A^*_{i_0}\) and \(B_{k-i} = B_{k-i_0}^{(l)}, B^*_{k-i} = B^*_{k-i_0}\).

Then \(A^{(l)}, B^{(l)}\) can be divided into \(O(n/2^l)\) segments for some \(k\).

We maintain the auxiliary sets \(T_0^{(l)}\) for \(-10 \leq b \leq 10\) throughout the algorithm, where the set \(T_b^{(l)}\) consists of all the segments \(([i_0, i_1], k)\) w.r.t. \(A^{(l)}\) and \(B^{(l)}\) satisfying:

\[
C_k \text{ is finite and } A^*_{i_0} + B^*_{k-i_0} \neq C^*_k \text{ and } A^{(l)}_{i_0} + B^{(l)}_{k-i_0} = C^{(l)}_{k} + b
\]

The algorithm proceeds as:

- In the first iteration \(l = h\), we want to calculate \(C^{(h)}\). However since \(p < 2^h\), \(A^{(h)}, B^{(h)}, C^{(h)}\) are zero sequences, so \(T_0^{(h)}\) includes all segments \(([i_0, i_1], k)\) where \(A^*_{i_0} + B^*_{k-i_0} \neq C^*_k\), and \(T_b^{(h)} = \emptyset\ (b \neq 0)\). Since the number of segments w.r.t. \(A^{(h)}, B^{(h)}\) for every \(k\) is \(O(n^{1-\alpha})\), \(|T_b^{(h)}| = O(n^{2-\alpha})\).

- For \(l = h - 1, \cdots, 0\), we first compute \(C^{(l)}\) with the help of \(T_1^{(l+1)}\), then construct \(T_b^{(l)}\) from \(T_0^{(l+1)}\) and \(T_1^{(l+1)}\). By Lemma 4.4 that \(\bigcup_{i=-10}^{10} T_i^{(l)} \subseteq \bigcup_{i=-10}^{10} T_i^{(l+1)}\), we can search the shorter segments contained in \(T_b^{(l+1)}\) to find \(T_b^{(l)}\). By Lemma 4.5 \(|T_b^{(l)}|\) is always bounded by \(O(n^{2-\alpha})\). Each iteration has three phases:

**Polynomial multiplication.** Construct two polynomial matrices \(A^p\) and \(B^p\) on variables \(x, y\) in the following way:

\[
A^p = \sum_{i=1}^{n} x^{A_i^{(l)} - 2A_i^{(l+1)}} \cdot y^{A_i^{(l+1)}} \cdot z^i.
\]

\[
B^p = \sum_{j=1}^{n} x^{B_j^{(l)} - 2B_j^{(l+1)}} \cdot y^{B_j^{(l+1)}} \cdot z^j.
\]

Then, compute the polynomial multiplication \(C^p = A^p \cdot B^p\) using standard FFT. Note that \(A_i^{(l)} - 2A_i^{(l+1)}, B_j^{(l)} - 2B_j^{(l+1)}\) are 0 or 1, so the degree of \(x\) terms are 0 or 1. This phase runs in time \(\tilde{O}(n^{1+\alpha})\).

**Subtracting erroneous terms.**

This phase is to extract the true values \(C^{(l)}_k\)’s from \(C^{(l+1)}_k\). The algorithm iterates over all offsets \(-10 \leq b \leq 10\), and enumerates all the segments in \(T_b^{(l+1)}\).
Lemma 4.4. Let \( \alpha \) and \( \lambda \) be real numbers. For each index \( 1 \leq k \leq n \), consider the coefficient of \( z^k \) in \( C^p_k \) denoted by \( C^p_k(x,y) \). Enumerate all terms \( \alpha x^e y^d \) of \( C^p_k(x,y) \) such that \( d = C^p_k(x,y) + 1 + b \), and define \( C^p_{k,\lambda}(x) \) to be the sum of all such \( \alpha x^e \). Next, compute a polynomial:

\[
P^k_{\lambda,b}(x) = \sum_{(i_0,i_1, k) \in T^{(l+1)}_b, j \in [i_0, i_1]} x^{A^{(l)}_i - 2A^{(l+1)}_i} + B^{(l)}_b^{(l+1)} - 2B^{(l+1)}_{k,i}.
\]

Finally, let \( s_{k,b} \) be the minimum degree of \( x \) of the polynomial \( C^p_{k,\lambda}(x) - R^p_{k,b}(x) \), and compute a candidate value \( s_{k,b} + 2d \) for \( C^{(l)}_k \). Ranging over all integer offsets \(-10 \leq b \leq 10\), take the minimum of all candidate values and output as \( C^{(l)}_k = \min_{-10 \leq b \leq 10} \{ s_{k,b} + 2d \} \).

Computing Triples \( T^{(l)}_b \).

To compute \( T^{(l)}_b \), initially set all \( T^{(l)}_b \leftarrow \emptyset \) for all \( |b| \leq 10 \). By Lemma 4.4, we know that \( \bigcup_{|i| = 10} T^{(l)}_i \) is contained in \( \bigcup_{|i| = 10} T^{(l+1)}_i \), so our work here is to check each segment in \( \bigcup_{|i| = 10} T^{(l+1)}_i \) and put it into the \( T^{(l)}_b \) it belongs to. Each segment in \( T^{(l+1)}_b \) breaks into at most 4 segments in the next iteration, and we can use binary search to find the breaking points. This phase runs in time \( \tilde{O}(n^{2-\alpha}) \) by Lemma 4.4.

The expected running time of the recursive algorithm is bounded by \( \tilde{O}(n^{1+\alpha} + n^{2-\alpha}) \). Taking \( \alpha = 0.5 \), the running time becomes \( \tilde{O}(n^{1.5}) \).

4.3 Proof of correctness

Lemma 4.4. We have \( \bigcup_{i = -10}^{10} T^{(l)}_i \subseteq \bigcup_{i = -10}^{10} T^{(l+1)}_i \).

Proof. By definition, \( A^{(l)}_i - 2A^{(l+1)}_i = 0 \) or 1, and \( B^{(l)}_i - 2B^{(l+1)}_i = 0 \) or 1. For \( C^{(l)} \), we can see similar result as Lemma 3.5 still holds.

\[
A^{(l+1)}_{i_0} + B^{(l+1)}_{k-i_0} - C^{(l+1)}_k \geq A^{(l)}_{i_0}/2 - 1/2 + B^{(l)}_{k-i_0}/2 - 1/2 - C^{(l)}_k/2 - 7/2 \\
\geq 1 \left( A^{(l)}_{i_0} + B^{(l)}_{k-i_0} - C^{(l)}_k \right) - 9/2.
\]

\[
A^{(l+1)}_{i_0} + B^{(l+1)}_{k-i_0} - C^{(l+1)}_k \leq A^{(l)}_{i_0}/2 + B^{(l)}_{k-i_0}/2 - C^{(l)}_k/2 + 8/2 \\
\leq 1 \left( A^{(l)}_{i_0} + B^{(l)}_{k-i_0} - C^{(l)}_k \right) + 4.
\]

Therefore, when \(-10 \leq A^{(l)}_{i_0} + B^{(l)}_{k-i_0} - C^{(l)}_k \leq 10 \),

\[-10 < -10/2 - 9/2 \leq A^{(l+1)}_{i_0} + B^{(l+1)}_{k-i_0} - C^{(l+1)}_k \leq 10/2 + 4 < 10.\]

Lemma 4.5. The expected number of segments in \( T^{(l)}_b \) is \( \tilde{O}(n^{2-\alpha}) \).

Proof. When \( 2^l \geq p/100 \), the total number of segments is bounded by \( O(n^{2-\alpha}) \), so next we assume that \( 2^l < p/100 \).

For any segment \( ([i_0, i_1], k) \) of finite elements where \( A^{*}_{i_0} + B^{*}_{k-i_0} = C^{*}_k \). By Assumption 4.1 (\( C_k \) mod \( p \) < \( 2p/3 \)), so we can get \( |A_{i_0} + B_{k-i_0} - C_k| \geq p/3 \) as in Lemma 3.7.
We want to bound the probability that \((i_0, i_1, k)\) appears in \(T_b^{(l)}\). If it is in \(T_b^{(l)}\),
\[
\left\lfloor \frac{A_{i_0} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k-i_0} \mod p}{2^l} \right\rfloor = C_k^{(l)} + b.
\]
So
\[
-4 \leq A_{i_0} \mod p \frac{2^l} + B_{k-i_0} \mod p \frac{2^l} - C_k \mod p \leq 4
\]
Let \(C_k = A_q + B_{k-q}\), and
\[
(A_{i_0} + B_{k-i_0} - A_q - B_{k-q}) \mod p \in [2^l(b - 4), 2^l(b + 4)].
\]
That is, \(A_{i_0} + B_{k-i_0} - A_q - B_{k-q}\) should be congruent to one of the \(O(2^l)\) remainders. As the argument in Lemma 3.7, the probability that it falls into the range of length \(O(2^l)\) is \(O(2^l/n^\alpha)\).

Since the number of segments is \(O(n^2/2^l)\), the expected number of segments in \(T_b^{(l)}\) is \(\tilde{O}(n^2/2^l)\).

**Lemma 4.6.** If \(A_i + B_{k-i} = C_k\), then \(A_i^{(l)} + B_{k-i}^{(l)} = C_k^{(l)} + b\) for some \(-10 \leq b \leq 10\).

**Proof.** By Assumption 4.1,
\[
A_i^{(l)} + B_{k-i}^{(l)} - C_k^{(l)} = \left\lfloor \frac{A_i \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k-i} \mod p}{2^l} \right\rfloor - C_k^{(l)}
\]
\[
\leq \left\lfloor \frac{A_i \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k-i} \mod p}{2^l} \right\rfloor - C_k \frac{2^l} \mod p + 3
\]
\[
= \frac{(A_i + B_{k-i} - C_k) \mod p}{2^l} + 3 = 3.
\]
\[
A_i^{(l)} + B_{k-i}^{(l)} - C_k^{(l)} = \left\lfloor \frac{A_i \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k-i} \mod p}{2^l} \right\rfloor - C_k^{(l)}
\]
\[
\geq \left\lfloor \frac{A_i \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k-i} \mod p}{2^l} \right\rfloor - C_k \frac{2^l} \mod p - 4
\]
\[
= \frac{(A_i + B_{k-i} - C_k) \mod p}{2^l} - 4 = -4.
\]

Next we argue that our algorithm correctly computes all entries \(C_k^{(l)}\) from \(C_k^{(l+1)}\) and \(T_b^{(l+1)}\), for \(l = h - 1, \cdots, 0\). Let \(q\) be the index such that \(C_k = A_q + B_{k-q}\). By the above lemma, there exists an integer offset \(b \in [-10, 10]\) such that \(A_q^{(l+1)} + B_{k-q}^{(l+1)} = C_k^{(l+1)} + b\). Therefore, by construction of
polynomials $A^p, B^p$, we have:

$$C^p_{k,b}(x) = \sum_{i\in\{l_{i(k+1)}+B_{k-i}^p\}} x^{A^p_{i(k+1)} + B^p_{k-i} - 2B^p_{k-i}}$$

$$= \sum_{i\in\{l_{i(k+1)}+B_{k-i}^p\}} x^{A^p_{i(k+1)} - 2A^p_{i(k+1)} + B^p_{k-i}} - 2B^p_{k-i}$$

$$+ \sum_{i\in\{l_{i(k+1)}+B_{k-i}^p\}} x^{A^p_{i(k+1)} + B^p_{k-i} - 2B^p_{k-i}}$$

$$+ \sum_{i\in\{l_{i(k+1)}+B_{k-i}^p\}} x^{A^p_{i(k+1)} + 2B^p_{k-i} - 2B^p_{k-i}}$$

$$= x^{-2(C^p_{k,b} + b)} \sum_{i\in\{l_{i(k+1)}+B_{k-i}^p\}} x^{A^p_{i(k+1)} + B^p_{k-i} + R^p_{k,b}(x)}$$

Since $A^p_{q} + B^p_{k-q} = C^p_{k}$ and $A^p_{q} + B^p_{k+1} = C^p_{k} + b$, when we extract $A^p_{q} + B^p_{k-q}$ from terms of $C^p_{k,b}(x) - R^p_{k,b}(x)$, it satisfies

$$\left\lfloor \frac{A^p_{q} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B^p_{k-q} \mod p}{2^l} \right\rfloor \leq \left\lfloor \frac{(A^p_{q} + B^p_{k-q}) \mod p}{2^l} \right\rfloor = \left\lfloor \frac{C^p_{k} \mod p}{2^l} \right\rfloor \leq \left\lfloor \frac{(C^p_{k} \mod p) + 2^l - 1}{2^l} \right\rfloor$$

$$\left\lfloor \frac{A^p_{q} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B^p_{k-q} \mod p}{2^l} \right\rfloor \geq \left\lfloor \frac{(A^p_{q} + B^p_{k-q}) \mod p}{2^l} \right\rfloor - 2(2^l - 1) \geq \left\lfloor \frac{(C^p_{k} \mod p) - 2(2^l - 1)}{2^l} \right\rfloor$$

Thus the term which gives $A^p_{q} + B^p_{k-q}$ can give a valid $C^p_{k}$. Also for every term which gives $A^p_{l(i)} + B^p_{l(k-i)}$ which satisfies $A^p_{l(i)} + B^p_{l(k-i)} = C^p_{k}$ and $A^p_{l(i)} + B^p_{l(k-i)} = C^p_{k}$, we have

$$\left\lfloor \frac{A^p_{l(i)} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B^p_{l(k-i)} \mod p}{2^l} \right\rfloor \geq \left\lfloor \frac{(A^p_{l(i)} + B^p_{l(k-i)}) \mod p}{2^l} \right\rfloor - 2(2^l - 1) \geq \left\lfloor \frac{(C^p_{k} \mod p) - 2(2^l - 1)}{2^l} \right\rfloor$$

So by choosing the minimum, we can get a valid $C^p_{k}$.

\[\square\]

**Acknowledgment**

Tianyi Zhang is supported by funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

**References**

[ACLL14] Amihood Amir, Timothy M Chan, Moshe Lewenstein, and Noa Lewenstein. On hardness of jumbled indexing. In *International Colloquium on Automata, Languages, and Programming*, pages 114–125. Springer, 2014.
Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

David Bremner, Timothy M Chan, Erik D Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and $x+y$. In *European Symposium on Algorithms*, pages 160–171. Springer, 2006.

Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019.

Shucheng Chi, Ran Duan, and Tianle Xie. Faster algorithms for bounded-difference min-plus product. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2022.

Timothy M Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the forty-seventh annual ACM Symposium on Theory of Computing*, pages 31–40, 2015.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. More geometric data structures. In *Computational Geometry*, pages 211–233. Springer, 2000.

Yuzhou Gu, Adam Polak 0001, Virginia Vassilevska Williams, and Yinzhan Xu. Faster monotone min-plus product, range mode, and single source replacement paths. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

Graham James Oscar Jameson. *The prime number theorem*. Number 53. Cambridge University Press, 2003.

François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303, 2014.

Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. *arXiv preprint arXiv:2106.02026*, 2021.

Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.

Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898, 2012.

Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018.
A When $B$ is column monotone

In Section 3, we consider the restricted case that the rows of $B$ are monotone. Now we explain how to calculate the min-plus product with the same asymptotic time complexity when $B$ is column-monotone, via minor adjustments of the recursive algorithm.

We want to calculate $C = A \star B$, where $A, B$ are $n \times n$ matrices, and the columns of $B$ are monotonously non-decreasing. We can assume without loss of generality that the rows of $A$ are monotonously non-increasing: If there exists two entries $A_{i,k_1}$ and $A_{i,k_2}$ in the same row of $A$, with $k_1 < k_2$ and $A_{i,k_1} < A_{i,k_2}$, then for any entry $C_{i,j} = \min_k\{A_{i,k} + B_{k,j}\}$, we have $A_{i,k_1} + B_{k_1,j} < A_{i,k_2} + B_{k_2,j}$, so the value of $A_{i,k_2}$ is never considered in the calculation, thus in this case we can set $A_{i,k_2} \leftarrow A_{i,k_1}$. When $B$ is bounded by $O(n)$, we can make $A$ and $C$ also bounded by $O(n)$ by the method in Section 2.

Let $\alpha \in (0, 1)$ be a constant parameter to be determined later, and pick a uniformly random prime number $p$ in the range of $[40n^\alpha, 80n^\alpha]$. Without loss of generality, let us assume that $n$ is a power of 2. Next we make the following assumption about elements in $A$ and $B$:

Assumption A.1. For every $i, j$, either $(A_{i,j} \mod p) < p/3$ or $A_{i,j} = +\infty$, and each row of $A$ is monotone besides the infinite elements. Similar for $B$: either $(B_{i,j} \mod p) < p/3$ or $B_{i,j} = +\infty$, and each column of $B$ is monotone besides the infinite elements.

By the same method in Lemma 4.3, we can prove:

Lemma A.1. The general computation of $A \circ B$ can be reduced to a constant number of computations of $A^i \circ B^i$ where all of $A^i, B^i$’s satisfy Assumption A.1. The number of intervals of infinity in each row of $A^i$ and in each column of $B^i$ is bounded by $O(n^{1-\alpha})$.

Define integer $h$ such that $2^{h-1} \leq p < 2^h$. For each integer $0 \leq l \leq h$, let $A^{(l)}$ be the $n \times n$ matrix defined as $A_{i,j}^{(l)} = \lfloor A_{i,j} \mod p \rfloor / 2^l$ if $A_{i,j}$ is finite, otherwise $A_{i,j}^{(l)} = +\infty$, similarly define matrix $B^{(l)}$.

We will recursively calculate $C^{(l)}$ for $l = h, h-1, \cdots, 0$, and if $C_{i,j}$ is finite, $C^{(l)}$ will satisfy

$$\lfloor (C_{i,j} \mod p) - 2(2^l - 1) \rfloor \leq C_{i,j}^{(l)} \leq \lfloor (C_{i,j} \mod p) + 2(2^l - 1) \rfloor$$

(Note that $C^{(l)}$ is not necessarily equal to $A^{(l)} \star B^{(l)}$.) In the end when $l = 0$ we can get the matrix $C_{i,j}^* = C_{i,j} \mod p$, by the procedure of recursion. Define $A^*$ and $B^*$ as $A_{i,j}^* = \lfloor A_{i,j} / p \rfloor$ and $B_{i,j}^* = \lfloor B_{i,j} / p \rfloor$. We use the trivial method which checks each interval on $i$-th row of $A^*$ and $j$-th
column of \( B^* \) to calculate \( C^* = A^* \ast B^* \) in \( \tilde{O}(n^{3-\alpha}) \) time. By Assumption [A.1], \( C^*_{i,j} = \lfloor C_{i,j}/p \rfloor \) if \( C_{i,j} \) is finite. Thus we can calculate the exact value of \( C_{i,j} \) by the result of \( C_{i,j} \mod p \).

We can see all elements in \( A^{(l)}, B^{(l)}, C^{(l)} \) are non-negative integers at most \( O(n^\alpha/2^l) \) or infinite. Since \( A \) is row-monotone and \( B \) is column-monotone, every row of \( A^{(l)} \) and every column of \( B^{(l)} \) is composed of \( O(n/2^l) \) intervals, where all elements in each interval are the same. The change we should make on the recursive algorithm is the organization of segments: instead of fixing \( i, k \), we fix \( i, j \).

**Definition A.1.** A segment w.r.t. \( A^{(l)} \) and \( B^{(l)} \) as \((i, j, [k_0, k_1])\), where \( i, j, k_0, k_1 \in [n] \) satisfies that for all \( k_0 \leq k \leq k_1 \), \( A_{i,k_0} \) and \( B_{k_0,j} \) are finite, \( A^{(l)}_{i,k} = A^{(l)}_{i,k_0} \) and \( A^*_{i,k} = A^*_{i,k_0} \), \( B^{(l)}_{k,j} = B^{(l)}_{k_0,j} \) and \( B^*_{k,j} = B^*_{k_0,j} \).

Then for the \( i \)-th row of \( A^{(l)} \) and the \( j \)-th column of \( B^{(l)} \), \([n]\) can be divided into \( O(n/2^l) \) segments.

We maintain the auxiliary sets \( T^{(l)}_b \) for \(-10 \leq b \leq 10\) throughout the algorithm, where the set \( T^{(l)}_b \) consists of all the segments \((i, j, [k_0, k_1])\) w.r.t. \( A^{(l)} \) and \( B^{(l)} \) satisfying:

\[
A_{i,k_0} \text{ is finite and } A^*_{i,k_0} + B^*_{k_0,j} \neq C^*_{i,j} \text{ and } A^{(l)}_{i,k_0} + B^{(l)}_{k_0,j} = C^{(l)}_{i,j} + b
\]

The algorithm proceeds as:

- In the first iteration \( l = h \), \( A^{(h)}, B^{(h)}, C^{(h)} \) are zero matrices, and it is easy to see \( |T^{(h)}_b| = O(n^{3-\alpha}) \).

- For \( l = h - 1, \ldots, 0 \), we first compute \( C^{(l)} \) with the help of \( T^{(l+1)}_b \), then construct \( T^{(l)}_b \) from \( T^{(l+1)}_b \). By Lemma [A.3] that \( \bigcup_{i = -10}^{10} T^{(l+1)}_i \subseteq \bigcup_{i = -10}^{10} T^{(l+1)}_i \), we can search the shorter segments contained in \( T^{(l+1)}_b \) to find \( T^{(l)}_b \). By Lemma [A.4], \( |T^{(l)}_b| \) is always bounded by \( O(n^{(3-\alpha)}) \).

Each iteration has three phases:

**Polynomial matrix multiplication.** Construct two polynomial matrices \( A^p \) and \( B^p \) on variables \( x, y \) in the following way: When \( A_{i,k} \) is finite,

\[
A^p_{i,k} = x A^{(l)}_{i,k} - 2 A^{(l+1)}_{i,k} \ast y A^{(l+1)}_{i,k}
\]

Otherwise \( A^p_{i,k} = 0 \), and when \( B_{k,j} \) is finite,

\[
B^p_{k,j} = x B^{(l)}_{k,j} - 2 B^{(l+1)}_{k,j} \ast y B^{(l+1)}_{k,j}
\]

Otherwise \( B^p_{k,j} = 0 \). Then, compute the standard \((+, \times)\) matrix multiplication \( C^p = A^p \ast B^p \) using fast matrix multiplication algorithms. Note that \( A^{(l)}_{i,j} - 2 A^{(l+1)}_{i,j} \) and \( B^{(l)}_{i,j} - 2 B^{(l+1)}_{i,j} \) are 0 or 1, so the degree of \( x \) terms are 0 or 1. This phase runs in time \( \tilde{O}(n^{(3-\alpha)}) \).

**Subtracting erroneous terms.** This phase is to extract the true values \( C^{(l)}_{i,j} \)'s from \( C^{(l+1)}_{i,j} \). The algorithm iterates over all offsets \(-10 \leq b \leq 10\), and enumerates all the segments in \( T^{(l+1)}_b \).

For each pair of indices \( i, j \in [n] \), if \( C^p_{i,j} = 0 \) then \( C^{(l)}_{i,j} = +\infty \), otherwise collect all the monomials \( \lambda x^e y^d \) of \( C^p_{i,j} \) such that

\[
d = C^{(l+1)}_{i,j} + b
\]
and let \( C^p_{i,j,b}(x) \) be the sum of all such terms \( \lambda x^c \). Next, compute a polynomial

\[
R^p_{i,j,b}(x) = \sum_{(i,j,k_0,k_1) \in T^p_{b(l+1)}, k \in [k_0,k_1]} x^{A^{(l)}_{i,k} - 2A_{i,k}^{(l+1)} + B_{k,j}^{(l)} - 2B_{k,j}^{(l+1)}}
\]

Finally, let \( s_{i,j,b} \) be the minimum degree of \( x \) in the polynomial \( C^p_{i,j,b}(x) - R^p_{i,j,b}(x) \), and compute a candidate value \( c_{i,j,b} = 2d + s_{i,j,b} \). Ranging over all integer offsets \(-10 \leq b \leq 10\), take the minimum of all candidate values and output as \( C^p_{i,j,b}(x) = \min_{-10 \leq b \leq 10} \{ c_{i,j,b} \} \). This phase runs in time \( \tilde{O}(n^{2+\alpha} + n^{3-\alpha}) \) (see Lemma A.4), since every segment \( (i,j,[k_0,k_1]) \in T^p_{b(l+1)} \) contains at most two different \( A_{i,k}^{(l)} \) and two different \( B_{k,j}^{(l)} \), thus it is easy to compute all of \( C^p_{i,j,b}(x) - R^p_{i,j,b}(x) \) in \( O(n^{2+\alpha} + |T^p_{b(l+1)}|) \) time.

**Computing Triples** \( T^p_b \). Since \( A_{i,k}^{(l)} - 2A_{i,k}^{(l+1)} \) and \( B_{i,j}^{(l)} - 2B_{i,j}^{(l+1)} \) are both 0 or 1, so each segment w.r.t. \( A^{(l+1)}, B^{(l+1)} \) can be split into at most \( O(1) \) segments w.r.t. \( A^{(l)}, B^{(l)} \). By Lemma A.3 we know that \( \bigcup_{i=-10}^{10} T^p_i \) is contained in \( \bigcup_{i=-10}^{10} T^p_{i(l+1)} \), so our work here is to check the sub-segments of each segment in \( \bigcup_{i=-10}^{10} T^p_{i(l+1)} \) and put it into the \( T^p_b(l) \) it belongs to. This phase runs in time \( \tilde{O}(|T^p_{b(l+1)}|) \).

The expected running time of the recursive algorithm is bounded by \( \tilde{O}(n^{3-\alpha} + n^{\omega+\alpha}) \) by Lemma A.4. Taking \( \alpha = (3 - \omega)/2 \), the running time becomes \( \tilde{O}(n^{(3+\omega)/2}) \).

**A.1 Proof of correctness**

We can get a similar lemma as Lemma 3.5.

**Lemma A.2.** In each iteration \( l = h - 1, \cdots, 0 \), \(-7 \leq C^{(l)}_{i,j} - 2C^{(l+1)}_{i,j} \leq 8\).

**Lemma A.3.** We have \( \bigcup_{i=-10}^{10} T^p_i \subseteq \bigcup_{i=-10}^{10} T^{(l+1)}_i \), that is, the segments we consider in each iteration must be sub-segments of the segments in the last iteration.

**Proof.** Segments \((i,j,[k_0,k_1])\) in \( T^p_b(l) \) and \( T^{(l+1)}_b \) must satisfy \( A_{i,k_0}, B_{k_0,j} \) are finite and \( A_{i,k_0}^* + B_{k_0,j}^* \neq C_{i,j}^* \). By definition, \( A_{i,k_0}^{(l)} - 2A_{i,k_0}^{(l+1)} = 0 \) or 1, and similar for \( B \). By Lemma A.2 we have

\[
A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - C_{i,j}^{(l)} \geq \frac{1}{2} \big( A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} \big) - 9/2.
\]

\[
A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - C_{i,j}^{(l)} \leq \frac{1}{2} \big( A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} \big) + 4.
\]

Therefore, when \(-10 \leq A_{i,k}^{(l)} + B_{k,j}^{(l)} - C_{i,j}^{(l)} \leq 10\),

\[-10 < -10/2 - 9/2 \leq A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - C_{i,j}^{(l+1)} \leq 10/2 + 4 \leq 10 < 10/2 - 9/2.
\]

**Lemma A.4.** The expected number of segments in \( T^p_b(l) \) is \( \tilde{O}(n^{3-\alpha}) \).

**Proof.** As before we assume that \( 2^l < p/100 \). For any segment \((i,j,[k_0,k_1])\) and \( k \in [k_0,k_1] \) where \( A_{i,k}, B_{k,j} \) are finite and \( A_{i,k}^* + B_{k,j}^* \neq C_{i,j}^* \), similar to proof in Lemma 3.7 we get \( |A_{i,k} + B_{k,j} - C_{i,j}| \geq p/3 \).
We want to bound the probability that \( (i, j, [k_0, k_1]) \) appears in \( T_b^{(l)} \). If it is in \( T_b^{(l)} \),

\[
\left\lfloor \frac{A_{i,k} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k,j} \mod p}{2^l} \right\rfloor = C_{i,j}^{(l)} + b.
\]

So

\[
-4 \leq \frac{A_{i,k} \mod p}{2^l} + \frac{B_{k,j} \mod p}{2^l} - \frac{C_{i,j} \mod p}{2^l} - b \leq 4.
\]

\((A_{i,k} + B_{k,j} - C_{i,j}) \mod p \in [2^l(b - 4), 2^l(b + 4)].\)

That is, \( A_{i,k} + B_{k,j} - C_{i,j} \) should be congruent to one of the \( O(2^l) \) remainders. For each possible remainder \( r \in [2^l(b - 4), 2^l(b + 4)], |b| \leq 10 \), we have

\[|r| \leq 14 \cdot 2^l < p/6 \leq \frac{1}{2} |A_{i,k} + B_{k,j} - C_{i,j}|.\]

So \( |(A_{i,k} + B_{k,j} - C_{i,j}) - r| \) is a positive number bounded by \( O(n) \), and the number of different primes \( p \in [40n^\alpha, 80n^\alpha] \) that \( p | (A_{i,k} + B_{k,j} - C_{i,k}) - r \) can not exceed \( 1/\alpha = O(1) \). In our algorithm, when we uniformly choose a prime \( p \) from \([40n^\alpha, 80n^\alpha]\), the probability that \((A_{i,k} + B_{k,j} - C_{i,j}) \mod p = r\) is \( \tilde{O} \left( \frac{1}{n^\alpha} \right) \). Since there are \( O(2^l) \) such possible remainders, in expectation we have \( O(2^l) \cdot O \left( \frac{n^\alpha}{2^l} \right) \cdot \tilde{O} (\frac{1}{n^\alpha}) = \tilde{O}(n^{3-\alpha}) \) segments in \( T_b^{(l)} \).

\(\square\)

From the proof of Lemma 3.8 we can get:

**Lemma A.5.** If \( A_{i,k} + B_{k,j} = C_{i,j} \), then \( A_{i,k}^{(l)} + B_{k,j}^{(l)} = C_{i,j}^{(l)} + b \) for some \(-10 \leq b \leq 10\).

Next we argue that our algorithm correctly computes all entries \( C_{i,j}^{(l)} \) from \( C_{i,j}^{(l+1)} \) and \( T_b^{(l+1)} \), for \( l = h - 1, \ldots, 0 \). Let \( q \) be the index such that \( C_{i,j} = A_{i,q} + B_{q,j} \). By the above lemma, there exists an integer offset \( b \in [-10, 10] \) such that \( A_{i,q}^{(l+1)} + B_{q,j}^{(l+1)} = C_{i,j}^{(l+1)} + b \). Therefore, by construction of polynomial matrices \( A^p, B^p \), we have:

\[
C_{i,j,b}^p(x) = \sum_{k|A_{i,k}^* + B_{k,j}^* = C_{i,j}^{*}} x^{A_{i,k}^{(l+1)} - 2A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - 2B_{k,j}^{(l+1)}}
\]

\[
+ \sum_{k|A_{i,k}^* + B_{k,j}^* \neq C_{i,j}^{*}} x^{A_{i,k}^{(l+1)} - 2A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - 2B_{k,j}^{(l+1)}}
\]

\[
+ \sum_{k|(A_{i,k}^* + B_{k,j}^*) = C_{i,j}} x^{A_{i,k}^{(l+1)} - 2A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - 2B_{k,j}^{(l+1)}}
\]

\[
+ \sum_{(i,j, k|k_0, k_1) \in T_b^{(l)}, \ k \in [k_0, k_1]} x^{A_{i,k}^{(l+1)} - 2A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - 2B_{k,j}^{(l+1)}}
\]

\[
= x^{-2\left( C_{i,j}^{(l+1)} + b \right)} \cdot \sum_{k|A_{i,k}^* + B_{k,j}^* = C_{i,j}^{*}} x^{A_{i,k}^{(l+1)} + B_{k,j}^{(l+1)} - 2B_{k,j}^{(l+1)}}
\]

Since \( A_{i,q}^{*} + B_{q,j}^{*} = C_{i,j}^{*} \) and \( A_{i,q}^{(l+1)} + B_{q,j}^{(l+1)} = C_{i,j}^{(l+1)} + b \), when we extract \( A_{i,q}^{(l)} + B_{q,j}^{(l)} \) from terms of \( C_{i,j,b}^p(x) - R_{i,j,b}^p(x) \), it satisfies

\[
\left\lfloor \frac{A_{i,q} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{q,j} \mod p}{2^l} \right\rfloor \leq \left( A_{i,q} + B_{q,j} \mod p \right) \frac{p}{2^l} = C_{i,j} \mod p \leq \left( C_{i,j} \mod p \right) + 2^l - 1.
\]
\[ \left\lfloor \frac{A_{i,q} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{q,j} \mod p}{2^l} \right\rfloor \geq \frac{(A_{i,q} + B_{q,j}) \mod p - 2(2^l - 1)}{2^l} \geq \left\lfloor \frac{(C_{i,j} \mod p) - 2(2^l - 1)}{2^l} \right\rfloor \]

Thus the term which gives \( A_{i,q}^{(l)} + B_{q,j}^{(l)} \) can give a valid \( C_{i,j}^{(l)} \). Also for every term which gives \( A_{i,k}^{(l)} + B_{k,j}^{(l)} \) which satisfies \( A_{i,k}^{*} + B_{k,j}^{*} = C_{i,j}^{*} \) and \( A_{i,k} + B_{k,j} \geq C_{i,j} \),

\[ \left\lfloor \frac{A_{i,k} \mod p}{2^l} \right\rfloor + \left\lfloor \frac{B_{k,j} \mod p}{2^l} \right\rfloor \geq \frac{(A_{i,k} + B_{k,j}) \mod p - 2(2^l - 1)}{2^l} \geq \left\lfloor \frac{(C_{i,j} \mod p) - 2(2^l - 1)}{2^l} \right\rfloor \]

So by choosing the minimum, we can get a valid \( C_{i,j}^{(l)} \).