

TOOM-COOK’s

The Fastest Multiplication exist



October 17, 2023

VigneshReddy.sols

-Vignesh Reddy Lingala

INTRODUCTION:

First Imagine a random 30 digit number and imagine another random same digit number. Multiply those two numbers on a book using old school methos it will take a long time and even if one miscalculation it may go wrong because of lots of multiplications .To reduce the complexity of the Multiplications some Analysts had developed algorithm to reduce the complexity of sub-multiplications

By in creasing addition and subtractions etc. For example

Karatsuba, Toom-cook, Strassen.

In the previous article we have read about the Karatsuba algorithm .As far as we consider Karatsuba it generates the time complexity of O(n1.585).But there is a developed against the Karatsuba that it can be done even faster than O(n1.585) . This paved a path for developing another algorithm i.e. TOOM-COOK algorithm.

Generally Toom-cook algo is considered when there exists a large number of digit s were required like in the previous example of 30 digit X 30 digit.

Let us Discuss A-bit more about the Toom-cook and where did the idea come from and what is the process of achieving the solution within a short-Time………………………..;

Toom-cook is a way of finding the algorithms for the multiplications by using the Divide and conquer approach

By dividing the subproblems into smaller ones and combining the solutions.

We have discussed about this approach in the earlier

Article now let us focus on the Toom’s way of finding algorithms.

Any Toom’s algorithm takes the time complexity by the formulae i.e.

|  |
| --- |
| **O( n log k (2k-1))** |

* Consider Toom -1 the complexity becomes O(n2)—

Which is considered as the Old school multiplication

* Consider Toom-2 the complexity becomes O(n1.585)

Which is considered as the Karatsuba algorithm.

* Consider Toom-3 algorithm complexity becomes O(n1.46) That is Toom-Cook algorithm.
* Generally these k values based on the interpolation of the data. This is the way of finding the unknown data form the known this is implemented in the Toom algorithm .This is one of the efficient Data mining Technique.

Generally Toom-3 used for Large digits:

Consider 2 digits:

M= 1234567890123456789012

N= 987654321987654321098

STEP:1 SPLITTING

* The first step in the Toom-3 algorithm was Splitting the numbers:
* Toom-3 algo states that the number must be Split in to 3 parts then ‘ k=3’.

The for Finding out the base we have a general formulae

i = max { [ log b  m / k] , [ log b  n/ k] }

* For the above numbers we get Base as 108 so we are going to divide the numbers in to \* digit numbers starting from LSD and grouping them .
* By following the number integration: 10^8

X(i) = Number % 108 ;

Number= Number / 108 ;

Until number <=108 – 1;

After splitting the numbers They become :

m2 = 123456

m1 = 78901234

m0 = 56789012

n2 = 98765

n1 = 43219876

n0 = 54321098

Converting the values in to polynomial equations By the degree of (k - 1) i.e. 2;

P(x) = m2 . x^2 + m1 . x +m0

q(x) = n2 . x2 + n1 . x +n0

p(x) = 123456. x2 +78901234 . x +56789012 ;

q(x) = 98765 . x2 + 43219876 . x +54321098 ;

NOTE : If the integer’s of different size’s are given then we take the splitting values of both are different Km  , Kn

Respectively. If we take TOOM-2.5 then Km =3 , Kn =2;

i = max { [ log b  m / km] , [ log b  n/ kn] }

STEP:2 EVALUATION

* The main reason of converting the numbers in to polynomials is to initialize the points foe the numbers through a (k-1) degree polynomial equations
* So the advantage of representing points for the number is we can solve the instances of the equations .
* By analysing the behaviour of the equations at various instances we can get the charecteristics of algorithm.
* The idea is to evaluate *p*(·) and *q*(·) at various points. Then multiply their values at these get points on the product polynomial. Finally interpolate to find its coefficients.

Since ;

Let TOOM\_2.5 then : Km =3 , Kn =2;

Deg(p,q) = Deg(p) + Deg(q) ;

Deg(p)+Deg(q)= Km +Kn - 1;

* Let us consider the value of d=5; for Toom-3
* This method of solving equations and representing in the form of matrix for finding is done for any values of the x So, for our ease range between -2,-1,0,1, ∞.
* The Above values are made for the following 5 Equations.

P(x)= m2 x^2 +m1 x +m0

Substitute the above values in the equation:

P(0) = m2 (0)^2 +m1  (0)+m0 = m0 .

P(1) = m2 (1)^2+m1  (1)+m0 = m2 +m1 +m0 .

P(-1) = m2 (-1)^2+m1  (-1)+m0 = m2 - m1 +m0 .

P(-2) = m2 (-2)^2+m1  (-2)+m0 = 4m2 - 2m1 +m0

P(∞) = m2 .

For solving same for Q(x)we get values in terms of n;

Q(0) = n2 (0)^2 +n1  (0)+n0 = n0 .

Q(1) = n2 (1)^2+n1  (1)+n0 = n2 +n1 +n0 .

Q(-1) = n2 (-1)^2+n1  (-1)+n0 = n2 - n1 +n0 .

Q(-2) = n2 (-2)^2+n1  (-2)+n0 = 4n2 - 2n1 +n0

Q(∞) = n2 .

After solving the equations in terms of numbers we get

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *p*(0) | = | *m*0 | = | 56789012 | = | 56789012 |
| *p*(1) | = | *m*0 + *m*1 + *m*2 | = | 56789012 + 78901234 + 123456 | = | 135813702 |
| *p*(−1) | = | *m*0 − *m*1 + *m*2 | = | 56789012 − 78901234 + 123456 | = | −21988766 |
| *p*(−2) | = | *m*0 − 2*m*1 + 4*m*2 | = | 56789012 − 2 × 78901234 + 4 × 123456 | = | −100519632 |
| *p*(∞) | = | *m*2 | = | 123456 | = | 123456 |
| *q*(0) | = | *n*0 | = | 54321098 | = | 54321098 |
| *q*(1) | = | *n*0 + *n*1 + *n*2 | = | 54321098 + 43219876 + 98765 | = | 97639739 |
| *q*(−1) | = | *n*0 − *n*1 + *n*2 | = | 54321098 − 43219876 + 98765 | = | 11199987 |
| *q*(−2) | = | *n*0 − 2*n*1 + 4*n*2 | = | 54321098 − 2 × 43219876 + 4 × 98765 | = | −31723594 |
| *q*(∞) | = | *n*2 | = | 98765 | = | 98765. |

The above data can be represented in the form of matrix-vector multiplication which makes the algorithm works faster

* By the above representation of the matrix-vector we can easily find out the p(x) values at any instances.
* This is done by ‘ Winograd multiplication of matrix’

Which reduces the numbers of additions need to be

done by the Algorithm

For Faster Evaluation process in generic method we can apply the following equations:

Let us consider that a variable i.e. p0=m0 +m2

* *p*0←*m*0 + *m*2=56789012 + 123456=56912468
* *p*(0)=*m*0=56789012=56789012
* *p*(1)=*p*0 + *m*1=56912468 + 78901234=135813702
* *p*(−1)=*p*0 − *m*1=56912468 − 78901234=−21988766
* *p*(−2)=(*p*(−1) + *m*2) × 2 − *m*0=(− 21988766 + 123456 ) × 2 − 56789012=− 100519632
* *p*(∞)=*m*2=123456=123456.
* The above equations are obtained from

p0=m0 +m2  ;

p(0)= m0 ; p(1)= m0 +m2 + m1 = p0 + m1 ;

p(-1)= m0 - m1 +m2  = p0 - m1 ;

p(-2)= 4m2 - 2m1 +m0

= ((p0-m1) + m2) \* 2;

= ((p1)+m2) \*2

= (m0+m2+-m1+m2) \*2

= 4m2 -2m1+m0

P(∞)=m2 .

* Multipoint evaluation can be obtained faster than with the above formulas. The number of elementary operations (addition/subtraction) can be reduced. The sequence given by “Bodrato” for Toom-3
* This sequence requires five addition/subtraction operations, one less than the straightforward evaluation. Moreover the multiplication by 4 in the calculation of *p*(−2) was saved.
* This reduces the time of calculations done in each step of the divide and conquer approach.

STEP:3

POINT-WISE MULTIPLICATION

* Point wise multiplication refers to the multiplying the instances obtained for both p(x) and q(x) with each other and storing it in the resultant R(x), This multiplication is done by using the TOOM-1 because after the splitting of the numbers the numbers becomes ATOMIC (may be single or less digits). If we apply the TOOM-3 algo to it then the process becomes complex.
* As it is easy to operate at integer level rather than the Floating-point level of numbers.
* This makes the Algorithm more efficient.

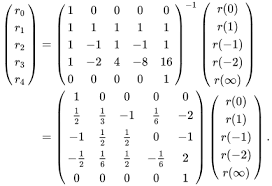
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *r*(0) | = | *p*(0)*q*(0) | = | 56789012 × 54321098 | = | 3084841486175176 |
| *r*(1) | = | *p*(1)*q*(1) | = | 135813702 × 97639739 | = | 13260814415903778 |
| *r*(−1) | = | *p*(−1)*q*(−1) | = | −21988766 × 11199987 | = | −246273893346042 |
| *r*(−2) | = | *p*(−2)*q*(−2) | = | −100519632 × −31723594 | = | 3188843994597408 |
| *r*(∞) | = | *p*(∞)*q*(∞) | = | 123456 × 98765 | = | 12193131840. |

STEP:4

INTERPOLATION

* Interpolation is the efficient way of finding unknown data from the available data .It is the most used “DATA MINING” technique .
* In mathematical approach interpolation refers to the finding out / estimation of the values between two or more Known data points.
* Hence we are dealing with the Polynomials we use Polynomial-interpolation.
* Polynomial Interpolation: Polynomial interpolation fits a polynomial function to the given data points. One of the most popular polynomial interpolation methods is Lagrange interpolation and Newton's divided difference interpolation.

* For more Accurate values we use Spline Interpolation especially we use “CUBIC SPLINES”.
* Spline Interpolation: Spline interpolation uses piecewise-defined functions (typically cubic splines) to approximate the curve between data points. This method provides smooth and continuous interpolants.
* Cubic Splines: Cubic spline interpolation is a specific type of spline interpolation that uses cubic polynomials for each segment. It provides continuous first and second derivatives at the junctions between segments.
* For solving the equation we have to multiply the resultant vector which we have found in the last step to the powered matrix of the chosen points .The below image shows the matrix multiplication



* All that remains is to compute this matrix-vector product. Although the matrix contains fractions, the resulting coefficients will be integers — so this can all be done with integer arithmetic, just additions, subtractions, and multiplication/division by small constants. A difficult design challenge in Toom–Cook is to find an efficient sequence of operations to compute this product; one sequence given by Bodrato.
* The fastest sequence of finding the resultant vector values is as follows .

r0 ← r(0) = 3084841486175176

r4 ← r(∞) = 12193131840

The above mentioned are directly obtained hence no need of computation.

r3 ← (r(−2) − r(1))/3

= (3188843994597408 − 13260814415903778)/3

= −3357323473768790

r1 ← (r(1) − r(−1))/2

= (13260814415903778 − (−246273893346042))/2

= 6753544154624910

r2 ← r(−1) − r(0) =

= (−246273893346042 )− 3084841486175176

= −3331115379521218

r3 ← (r2 − r3)/2 + 2r(∞)

=(−3331115379521218 − (−3357323473768790))/2

+ 2 × 12193131840

= 13128433387466

r2 ← r2 + r1 − r4

= −3331115379521218 + 6753544154624910

– 12193131840

= 3422416581971852

r1 ← r1 − r3

= 6753544154624910 − 13128433387466

= 6740415721237444.

* Hence the resultant vector values are obtained we represent those in the form of polynomial.

r(x) = r0 + r1 x^1 + r2 x^2 +r3 x^3 +r4 x^4

r(x)= 3084841486175176

+ 6740415721237444 \* x^1

+ 3422416581971852 \* x^2

+ 13128433387466 \* x^3

+ 12193131840\* x^4

* If we were using different km, kn, or evaluation points, the matrix and so our interpolation strategy would change; but it does not depend on the inputs and so can be hard-coded for any given set of parameters.

STEP:5

RECOMPOSITION

* Finally, we evaluate r(B) to obtain our final answer. This is straightforward since B is a power of b and so the multiplications by powers of B are all shifts by a whole number of digits in base b.
* We will multiply the number with the ‘x’ i.e.

X=108.

We get;

r(x)= 3084841486175176

+ 6740415721237444 \* 108^1

+ 3422416581971852 \* 108^2

+ 13128433387466 \* 108^3

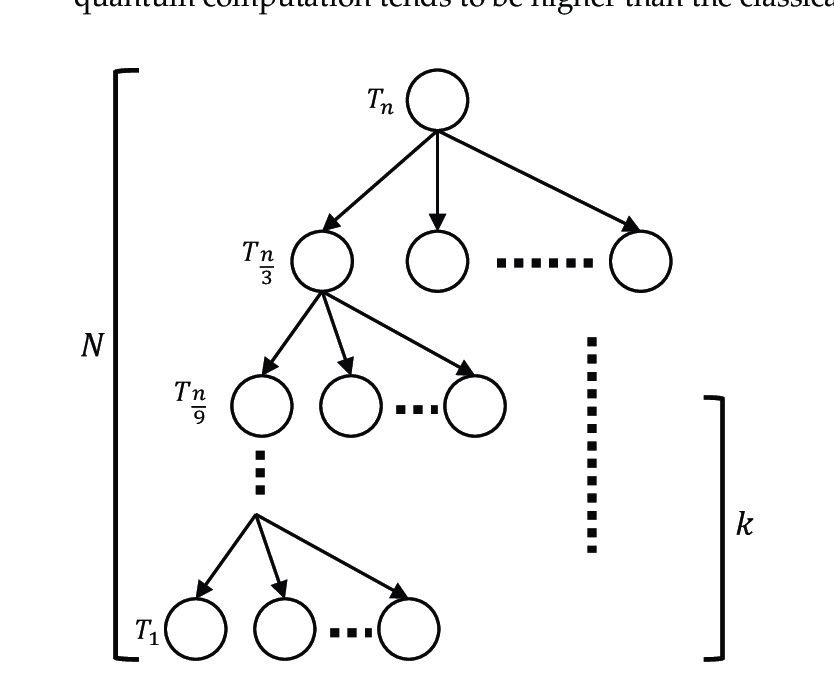
+ 12193131840\* 108^4

* Hence by adding all numbers we get;

121932631246761163249376009520858588617 5176

* The above number is the product of both ‘n’ and ‘m’.

RECURSION TREE FOR TOOM-3 MULTIPLICATION



Implementation of TOOM-3 Algorithm using

C++

#include <cstdlib> // for rand()

#include <iostream> // for cout

#include <math.h> // for pow()

#include <stdio.h> // for printf()

//#define TEST

#define D\_MAX 729

#define D 729

using namespace std;

class Calc

{

int A[D];

int B[D];

#ifdef TEST

int cnt\_mul;

clock\_t t1, t2;

double tt;

#endif

public:

Calc();

void calcToomCook();

private:

void multiplyNormal(int \*, int \*, int, int \*);

void multiplyToomCook3(int \*, int \*, int , int \*);

void doCarry(int \*, int);

void display(int \*, int \*, int \*);

};

Calc::Calc()

{

int i;

for (i = 0; i < D; i++)

{

A[i] = rand() % 10;

B[i] = rand() % 10;

}

}

void Calc::calcToomCook()

{

int a[D\_MAX];

int b[D\_MAX];

int z[D\_MAX \* 2];

int i;

#ifdef TEST

t1 = clock();

for (int l = 0; l < 1000; l++) {

cnt\_mul = 0;

#endif

for (i = 0; i < D; i++)

{

a[i] = A[i];

b[i] = B[i];

}

for (i = D; i < D\_MAX; i++)

{

a[i] = 0;

b[i] = 0;

}

multiplyToomCook3(a, b, D\_MAX, z);

doCarry(z, D\_MAX \* 2);

#ifdef TEST

}

t2 = clock();

tt = (double)(t2 - t1) / CLOCKS\_PER\_SEC;

#endif

display(a, b, z);

}

/\*

\* Multiplication Standard

\*/

void Calc::multiplyNormal(int \*a, int \*b, int tLen, int \*z)

{

int i, j;

for(i = 0; i < tLen \* 2; i++) z[i] = 0;

for (j = 0; j < tLen; j++) {

for (i = 0; i < tLen; i++) {

z[j + i] += a[i] \* b[j];

#ifdef TEST

cnt\_mul++;

#endif

}

}

}

void Calc::multiplyToomCook3(int \*a, int \*b, int tLen, int \*z)

{

int \*a0 = &a[0]; // Multiplicand / right side array pointer

int \*a1 = &a[tLen / 3]; // Multiplicand / central array pointer

int \*a2 = &a[tLen \* 2/ 3]; // Multiplicand / left side array pointer

int \*b0 = &b[0]; // Multiplier / right side array pointer

int \*b1 = &b[tLen / 3]; // Multiplier / central array pointer

int \*b2 = &b[tLen \* 2/ 3]; // Multiplier / left side array pointer

int \*c0 = &z[(tLen / 3) \* 0]; // c0

int \*c2 = &z[(tLen / 3) \* 2]; // c2

int \*c4 = &z[(tLen / 3) \* 4]; // c4

int c1 [(tLen / 3) \* 2]; // c1

int c3 [(tLen / 3) \* 2]; // c3

int a\_m2 [tLen / 3]; // a( -2)

int a\_m1 [tLen / 3]; // a( -1)

int a\_0 [tLen / 3]; // a( 0)

int a\_1 [tLen / 3]; // a( 1)

int a\_inf [tLen / 3]; // a(inf)

int b\_m2 [tLen / 3]; // b( -2)

int b\_m1 [tLen / 3]; // b( -1)

int b\_0 [tLen / 3]; // b( 0)

int b\_1 [tLen / 3]; // b( 1)

int b\_inf [tLen / 3]; // b(inf)

int c\_m2 [(tLen / 3) \* 2]; // c( -2)

int c\_m1 [(tLen / 3) \* 2]; // c( -1)

int c\_0 [(tLen / 3) \* 2]; // c( 0)

int c\_1 [(tLen / 3) \* 2]; // c( 1)

int c\_inf [(tLen / 3) \* 2]; // c(inf)

int i;

if (tLen <= 9)

{

multiplyNormal(a, b, tLen, z);

return;

}

// ==== a(-2) = 4 \* a2 - 2 \* a1 + a0, b(1) = 4 \* b2 - 2 \* b1 + b0

for(i = 0; i < tLen / 3; i++)

{

a\_m2[i] = (a2[i] << 2) - (a1[i] << 1) + a0[i];

b\_m2[i] = (b2[i] << 2) - (b1[i] << 1) + b0[i];

}

// ==== a(-1) = a2 - a1 + a0, b(1) = b2 - b1 + b0

for(i = 0; i < tLen / 3; i++)

{

a\_m1[i] = a2[i] - a1[i] + a0[i];

b\_m1[i] = b2[i] - b1[i] + b0[i];

}

// ==== a(0) = a0, b(0) = b0

for(i = 0; i < tLen / 3; i++) {

a\_0[i] = a0[i];

b\_0[i] = b0[i];

}

// ==== a(1) = a2 + a1 + a0, b(1) = b2 + b1 + b0

for(i = 0; i < tLen / 3; i++) {

a\_1[i] = a2[i] + a1[i] + a0[i];

b\_1[i] = b2[i] + b1[i] + b0[i];

}

// ==== a(inf) = a2, b(inf) = b2

for(i = 0; i < tLen / 3; i++) {

a\_inf[i] = a2[i];

b\_inf[i] = b2[i];

}

// ==== c(-2) = a(-2) \* b(-2)

multiplyToomCook3(a\_m2, b\_m2, tLen / 3, c\_m2 );

// ==== c(-1) = a(-1) \* b(-1)

multiplyToomCook3(a\_m1, b\_m1, tLen / 3, c\_m1 );

// ==== c(0) = a(0) \* b(0)

multiplyToomCook3(a\_0, b\_0, tLen / 3, c\_0 );

// ==== c(1) = a(1) \* b(1)

multiplyToomCook3(a\_1, b\_1, tLen / 3, c\_1 );

// ==== c(inf) = a(inf) \* b(inf)

multiplyToomCook3(a\_inf, b\_inf, tLen / 3, c\_inf);

// ==== c4 = 6 \* c(inf) / 6

for(i = 0; i < (tLen / 3) \* 2; i++)

c4[i] = c\_inf[i];

// ==== c3 = -c(-2) + 3 \* c(-1) - 3 \* c(0) + c(1) + 12 \* c(inf) / 6

for(i = 0; i < (tLen / 3) \* 2; i++)

{

c3[i] = -c\_m2[i];

c3[i] += (c\_m1[i] << 1) + c\_m1[i];

c3[i] -= (c\_0[i] << 1) + c\_0[i];

c3[i] += c\_1[i];

c3[i] += (c\_inf[i] << 3) + (c\_inf[i] << 2);

c3[i] /= 6;

}

// ==== c2 = 3 \* c(-1) - 6 \* c(0) + 3 \* c(1) - 6 \* c(inf) / 6

for(i = 0; i < (tLen / 3) \* 2; i++) {

c2[i] = (c\_m1[i] << 1) + c\_m1[i];

c2[i] -= (c\_0[i] << 2) + (c\_0[i] << 1);

c2[i] += (c\_1[i] << 1) + c\_1[i];

c2[i] -= (c\_inf[i] << 2) + (c\_inf[i] << 1);

c2[i] /= 6;

}

// ==== c1 = c(-2) - 6 \* c(-1) + 3 \* c(0) + 2 \* c(1) - 12 \* c(inf) / 6

for(i = 0; i < (tLen / 3) \* 2; i++) {

c1[i] = c\_m2[i];

c1[i] -= (c\_m1[i] << 2) + (c\_m1[i] << 1);

c1[i] += (c\_0[i] << 1) + c\_0[i];

c1[i] += (c\_1[i] << 1);

c1[i] -= (c\_inf[i] << 3) + (c\_inf[i] << 2);

c1[i] /= 6;

}

// ==== c0 = 6 \* c(0) / 6

for(i = 0; i < (tLen / 3) \* 2; i++)

c0[i] = c\_0[i];

// ==== z = c4 \* x^4 + c3 \* x^3 + c2 \* x^2 + c1 \* x + c0

for(i = 0; i < (tLen / 3) \* 2; i++) z[i + tLen / 3] += c1[i];

for(i = 0; i < (tLen / 3) \* 2; i++) z[i + (tLen / 3) \* 3] += c3[i];

}

void Calc::doCarry(int \*a, int tLen) {

int cr;

int i;

cr = 0;

for(i = 0; i < tLen; i++) {

a[i] += cr;

if(a[i] < 0) {

cr = -(-(a[i] + 1) / 10 + 1);

} else {

cr = a[i] / 10;

}

a[i] -= cr \* 10;

}

// Overflow

if (cr != 0) printf("[ OVERFLOW!! ] %d\n", cr);

}

/\*

\* Result output

\*/

void Calc::display(int \*a, int \*b, int \*z)

{

int i;

int aLen = D\_MAX, bLen = D\_MAX, zLen = D\_MAX \* 2;

while (a[aLen - 1] == 0) if (a[aLen - 1] == 0) aLen--;

while (b[bLen - 1] == 0) if (b[bLen - 1] == 0) bLen--;

while (z[zLen - 1] == 0) if (z[zLen - 1] == 0) zLen--;

// a

printf("a =\n");

for (i = aLen - 1; i >= 0; i--) {

printf("%d", a[i]);

if ((aLen - i) % 10 == 0) printf(" ");

if ((aLen - i) % 50 == 0) printf("\n");

}

printf("\n");

// b

printf("b =\n");

for (i = bLen - 1; i >= 0; i--) {

printf("%d", b[i]);

if ((bLen - i) % 10 == 0) printf(" ");

if ((bLen - i) % 50 == 0) printf("\n");

}

printf("\n");

// z

printf("z =\n");

for (i = zLen - 1; i >= 0; i--) {

printf("%d", z[i]);

if ((zLen - i) % 10 == 0) printf(" ");

if ((zLen - i) % 50 == 0) printf("\n");

}

printf("\n\n");

#ifdef TEST

printf("Counts of multiply / 1 loop = %d\n", cnt\_mul); // Multiplication count

printf("Total time of all loops = %f seconds\n", tt); // processing time

#endif

}

int main()

{

try

{

Calc objCalc;

objCalc.calcToomCook();

}

catch (...) {

cout << "Exception occurred!" << endl;

return -1;

}

return 0;

}

VALUE OF A:

a =

2591143596 2346775392 0937646531 6568615864 8398725493 4087130219 1596917334 0177683992 0813223745 9779754352 8944932316 9629947339 9445091835 0388960192 8955061632 3738412276 1433199925 0315713085 9378419329 3659409294

2480119500 7197101549 2212451932 2704588466 4812771903 8901977387 5802786582 6405523129 4325764724 67201430001277681909 6692814478 7319151423 1290842279 3038342092 1065637955 8549310037 0533764082 9436637253 9515438679

7766720886 1618471741 7221329703 7458017793 3575966087 0590224271 1768660254 4076013016 1293279873 59459597089946080467 5263339808 7033368282 4246141390 0394299488 0252206204 7509191362 6987956604 6039990313 8920242824

0815106928 5729454200 4798463368 3859845729 7374431491 6860803289 6677917104 5612777454 6962438673 4424366456 2613584191 3929722712 030296373

B=

5640936430 9010018506 2074049213 9225420531 7120391806 3626925169 5258572474 6042646617 5789516910 0220359949 3394237741 5504199300 2689317010 0173030204 0908343875 7276728921 5639684944 4001086363 5835539015 9644066815

4976237121 6087234637 2062179441 7355273885 5857697267 0889167023 0551830095 0630610627 7679851212 5513317149 5213794389 4091568806 9338677524 8328634542 4070207416 2850176096 3013514856 8183157295 7343340228 8454195001

4934592104 9265001117 5102743998 1388994837 8389707468 3149905673 0009246688 7647591722 9004589105 9497241813 2372874109 8652730769 6483888898 1629437542 5694561003 2950241948 0871495217 0922867240 6583613401 1096478656

6501028381 4742690593 3678808349 1837517816 5164400885 1539589914 3536751799 9106221809 6280914880 4755795751 0306047932 6782530986 669712556.

For multiplying the Above numbers The Output becomes very large you can copy the code and run it To get the output.

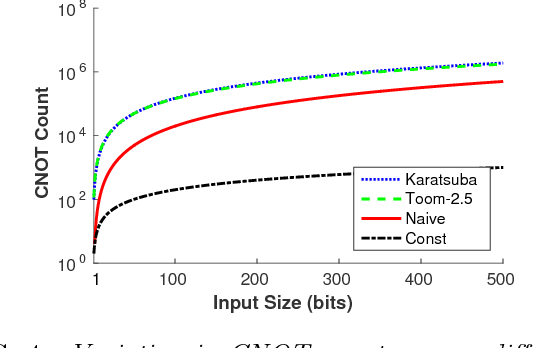
The execution Time:

CPU Time: 0.00 sec(s), Memory: 3456 kilobyte(s)

compiled and & executed in 1.182 sec(s) .

Just Imagine if this is done using Traditional Even by computer It will mess up the solution .

Time Complexity of traditional vs Karatsuba vs TOOM-cook



REFERENCES:

Sources: WIKKIPEDIA :

Link: [Toom–Cook multiplication - Wikipedia](https://en.wikipedia.org/wiki/Toom%E2%80%93Cook_multiplication)

IMAGES : RESEARCHGATE.COM

Link: [The number of bit operations of the classical, recursive Karatsuba, and... | Download Scientific Diagram (researchgate.net)](https://www.researchgate.net/figure/The-number-of-bit-operations-of-the-classical-recursive-Karatsuba-and-the-hybrid_fig2_4243622)

EXTRA-SOURCES :

Thanks To SREEDHAR

-Head Of The CET Program.