

# BIG INT

Aarush Dua (2022EEB1145)  
Arnav Bansal (2022EEB1160)  
Aryan Jain (2022EEB1162)  
Astha Singhal (2022EEB1164)



November 9, 2024

## 1. Introduction

**Summary:** The primary objective of our project is to find the exact value of  $\pi$  up to 10000 decimal places. We are computing this by using C language. But the range of integers in C language is bounded, So we need to create a data structure which can hold huge integers; we call it - BIG INT. We are implementing all major arithmetic operations in BIG INT to achieve our final objective successfully. Also, We have implemented functions that can deal with substantial fractions. But finding  $\pi$  is itself a mathematical complexity. We are first using Newton-Raphson Method for root convergence and then using Chudnovsky Algorithm to find  $\pi$ .

This project is divided into three parts.

- i) The implementation of our self-designed data structure - BIGINT.
- ii) Making of utility functions & implementing fractions.
- iii) Use of BIGINT to deduce the value of  $\pi$  up to 10000 decimal places.

### I) IMPLEMENTATION OF BIGINT :

To implement BIGINT, we are making functions such as BIGINT Addition, BIGINT Subtraction, BIGINT Multiplication, BIGINT Division, BIGINT Decimal Division, BIGINT Remainder (Modulo), BIGINT GCD, BIGINT Power, BIGINT Factorial, BIGINT Square Root. These functions can be called and accessed using the switch case command. These functions will be further internally used in the process of calculation of  $\pi$ .

**II) MAKING OF UTILITY FUNCTION (FRACTIONS):** To add more worth to our program. We are adding some utility operations and implementing Fractions. To implement Fractions, we are making functions such as Fraction Addition, Fraction Subtraction, Fraction Multiplication, Fraction Division, and Fraction-Reduced to Simplest Form.

### III) USING BIGINT TO FIND $\pi$ :

To find the value of  $\pi$ , we are using Chudnovsky Algorithm. The Chudnovsky algorithm is a fast method for calculating the digits of  $\pi$ , based on Ramanujan's  $\pi$  formulae. To use Chudnovsky Algorithm, we need to use the Newton-Raphson Method of convergence to find the exact value of roots coming in Chudnovsky's formula. We are using all the functions we have made in implementing BIGINT to compute Chudnovsky's Formula & Hence finding the value of  $\pi$  exactly up to 10000 decimal places.

## 2. Equations

We have used two major equations

- i) Newton-Raphson Method
- ii) Chudnovsky Algorithm

### I) NEWTON-RAPHSON METHOD

The Newton–Raphson method, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a single-variable function  $f$  defined for a real variable  $x$ , the function's derivative  $f'$ , and an initial guess  $x_0$  for a root of  $f$ . If the function satisfies sufficient assumptions and the initial guess is close, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

is a better approximation of the root than  $x_0$ . Geometrically,  $(x_1, 0)$  is the intersection of the  $x$ -axis and the tangent of the graph of  $f$  at  $(x_0, f(x_0))$ : that is, the improved guess is the unique root of the linear approximation at the initial point. The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently precise value is reached.

### II) CHUDNOVSKY'S ALGORITHM

The Chudnovsky algorithm is a fast method for calculating the digits of  $\pi$ , based on Ramanujan's  $\pi$  formulae. The algorithm is based on the negated Heegner number  $d = -163$ , the  $j$ -function  $j\left(\frac{1+i\sqrt{163}}{2}\right) = -640320^3$ , and on the following rapidly convergent generalized hypergeometric series.

$$\frac{1}{\pi} = 12 \sum_{q=0}^{\infty} \frac{(-1)^q (6q)! (545140134q + 13591409)}{(3q)! (q!)^3 (640320)^{3q + \frac{3}{2}}}$$

For a high-performance iterative implementation, this can be simplified to

$$\frac{(640320)^{\frac{3}{2}}}{12\pi} = \frac{426880\sqrt{10005}}{\pi} = \sum_{q=0}^{\infty} \frac{(6q)! (545140134q + 13591409)}{(3q)! (q!)^3 (-262537412640768000)^q}$$

There are 3 big integer terms (the multinomial term  $M_q$ , the linear term  $L_q$ , and the exponential term  $X_q$ ) that make up the series, and  $\pi$  equals the constant  $C$  divided by the sum of the series, as below:

$$\pi = C \left( \sum_{q=0}^{\infty} \frac{M_q \cdot L_q}{X_q} \right)^{-1}, \text{ where :}$$

$$C = 426880\sqrt{10005},$$

$$M_q = \frac{(6q)!}{(3q)! (q!)^3},$$

$$L_q = 545140134q + 13591409,$$

$$X_q = (-262537412640768000)^q.$$

The terms  $M_q$ ,  $L_q$ , and  $X_q$  satisfy the following recurrences and can be computed as such:

$$\begin{aligned} L_{q+1} &= L_q + 545140134 & \text{where } L_0 &= 13591409 \\ X_{q+1} &= X_q \cdot (-262537412640768000) & \text{where } X_0 &= 1 \\ M_{q+1} &= M_q \cdot \left( \frac{(12q+2)(12q+6)(12q+10)}{(q+1)^3} \right) & \text{where } M_0 &= 1 \end{aligned}$$

where  $M_0 = 1$ .

### 3. Functions & Operations

#### 3.1. Function Prototypes

```

59 // ---- BigInt functions ----
60 BigInt new_BigInt(const unsigned int length);
61 void set_zero(BigInt b);
62 void free_BigInt(BigInt b);
63 void print_BigInt(BigInt b);
64 BigInt Add(const BigInt a, const BigInt b);
65 BigInt Subtract(const BigInt a, const BigInt b);
66 void _MUL(llu x, llu y, llu *carry, llu *result);
67 BigInt Multiply(const BigInt a, const BigInt b);
68 void Left_Shift(BigInt num, unsigned int shift);
69 int Compare(const BigInt a, const BigInt b);
70 BigInt Divide(const BigInt a, const BigInt b, BigInt *remainder);
71 char *Decimal_Division(BigInt a, BigInt b);
72 BigInt Remainder(BigInt a, BigInt b);
73 BigInt Power(BigInt num, llu p);
74 BigInt GCD(BigInt a, BigInt b);
75 BigInt Factorial(llu n);
76 void precompute_factorial();
77 void Increment(const BigInt a, const BigInt delta);
78 void increase_size(BigInt b, const unsigned int delta_len);
79 void remove_preceding_zeroes(BigInt a);
80 int isPrime(int n);
81 int gcd(int a, int b);
82

```

Figure 1: BIGINT FUNCTION PROTOTYPES.

```

83
84 // ---- Fraction functions ----
85 Fraction new_Fraction();
86 Fraction input_Fraction();
87 void print_Fraction(Fraction a);
88 void reduce_Fraction(Fraction a);
89 Fraction add_Fraction(Fraction a, Fraction b);
90 Fraction subtract_Fraction(Fraction a, Fraction b);
91 Fraction multiply_Fraction(Fraction a, Fraction b);
92 Fraction divide_Fraction(Fraction a, Fraction b);
93 void reciprocal_Fraction(Fraction a);
94 void free_Fraction(Fraction a);
95 void cancel_zeroes(Fraction a);
96
97 // Calculate the square root of a BigInt using Newton Rapson method
98 Fraction Square_Root(BigInt k, int n);
99
100 // Computes value of PI using Chudnovsky algorithm
101 void PI_Chudnovsky(int n);

```

Figure 2: FRACTION FUNCTION PROTOTYPES.

## 3.2. Operations

### I. Basic Operations on Big Integers

1. Addition : Takes two BIGINT and adds them
2. Subtraction : Takes two BIGINT and subtracts one from the other
3. Multiplication : Takes two BIGINT and multiplies them
4. Division : Takes two BIGINT and divides to give Quotient & Remainder
5. Decimal Division : Takes two BIGINT and divides to give the exact value with decimals
6. Remainder (Modulo) : Takes two BIGINT and gives the remainder
7. GCD : Takes two BIGINT & return GCD of them
8. Power : Takes base(x) as BIGINT & exponent(y) long long int and gives  $x^y$
9. Factorial : Takes a long long int and outputs its factorial

### II. Operations on Fractions

1. Addition : Takes fractions and adds them
2. Subtraction : Takes two fractions and subtracts them
3. Multiplication : Takes two fractions and multiplies them
4. Division: Takes fractions and divides one from the other
5. Reduce to Simplest Form : Takes a fraction and reduces it to its simplest form.

### III. Computation of $\pi$

1. Compute Sqrt(10005) using Newton-Raphson Algorithm
2. Compute Value of PI using Chudnovsky Algorithm

### IV. Miscellaneous

1. Set Decimal Precision
2. Exit the program



## 4. Output

For the demonstration and showcase of the usability of our program, we are attaching the final output image.

```
Enter your choice: 16
Enter number of terms of Chudnovsky Algorithm: 215
Computing pi...
Chudnovsky formula requires square root of 10005 to be calculated first.
Computing sqrt(10005)...
Computing term: 3... Done!
Root 10005 calculated!
sqrt(10005) computed
Computing term 215... Done!
Calculating SUM of terms...
SUM of terms calculated
Rational Equivalent of pi computed
Execution time: 29.34 seconds
Do you want to convert it to decimal and write it to a file?
Note: Fraction to decimal conversion is very computationally intensive and takes a lot of time.
Your choice? (y/n): y
PI =
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651
3282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933446128
4756482337867831652712019091456485669234603486104543266482133936072602491412737245870066063155881748815209209628
2925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511
8548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190702179860943702
770539217176293176523846748184676694051320005681271452635608277857713427577896091736371787214684409012249534301
4654958537105079227968925892354201995611212902196086403441815981362977477130996051870721134999999837297804995105
9731732816096318595024459455346908302642522308253344685035261931188171010003137838752886587533208381420617177669
1473035982534904287554687311595628638823537875937519577818577805321712268066130019278766111959092164201989380952
5720106548586327886593615338182796823030195203530185296899577362259941389124972177528347913151557485724245415069
5950829533116861727855889075098381754637464939319255060400927701671139009848824012858361603563707660104710181942
9555961989467678374494482553797747268471040475346462080466842590694912933136770289891521047521620569660240580381
5019351125338243003558764024749647326391419927260426992279678235478163600934172164121992458631503028618297455570
6740838505404588586926995690927210797509302955321165344987202755960236480665499119881834797753566369807426542527
6255181841754767289097772793800081647060016145249192173217214772350141441973568548163161573525521334757418494
6843852332390739414333454776241686251898356948556209921922218427255025425688767179049460165346680498862723279178
6085784383827967976681454100953883786360950680064225125205117392984896084128488626945604241965285022210661186306
7442786220391949450471237137869609563643719172874677646575739624138908658326459958133904780275923931163323946717
9928016994445635829634329582568846664358368625861293549628789990802410824522007188352364268064667119107305838551
3182385647704759207791928674179108654771505367833084273543006490469586033244730588634665977892977434881191622994
3898777563926625247397881703809998192196823221883496930352512398031381888567936233512952698387902817611192662734
9773636544018001288714534116755714240799703995341113559926574007924027589910234366319727475763902125122723606239
6885624957507196910286000348744279212449880649370550116274926702246051917195921740343446528954643584144407163779
0444150649533462195694095963500359854029846094675131711767263357586500792019988641104221669002711741895747072966
707603785360839402772169174929242437473166202075793800239618891801235599658348059418935590938010063076540073492
6119763209369325321467408819155101770809322782115103198007005887344861252783000846264538169834880007997794509665
022333407081855184682560780791657279505260190295063421949957219527427215151436806909687541
Execution time: 566.05 seconds
Output Written to file
```

Fig 4: Output

## 5. Tables

On running the program for different levels of precision of  $\pi$  &  $\sqrt{10005}$ , we are tracking down the average time taken to achieve the required accuracy, and then we are plotting those data in the following table :-

### CALCULATION OF $\pi$

	No. of terms considered(Chudnovsky)	Accuracy in value of $\pi$	Time Taken
1).	10	141	0.002 <i>seconds</i>
2).	70	992	0.53 <i>seconds</i>
3).	210	2977	46.629 <i>seconds</i>
4).	430	6098	894.347 <i>seconds</i>
5).	710	10068	3741.891 <i>seconds</i>

Table 1: Calculation of  $\pi$  & Benchmarks

### CALCULATION OF $\sqrt{10005}$

	No. of terms cons.(Newton-Raphson)	Accuracy in value of $\sqrt{10005}$	Time Taken
1).	1	497	0.018 <i>seconds</i>
2).	3	1995	0.594 <i>seconds</i>
3).	4	3994	3.956 <i>seconds</i>
4).	6	15985	213.331 <i>seconds</i>
5).	7	31971	1682.48 <i>seconds</i>

Table 2: Calculation of  $\sqrt{10005}$  & Benchmarks

## 6. Conclusions

We have been able to store huge numbers that are even out of the range of integers stored in C language. We implemented BIGINT by making various functions to perform all major arithmetic operations. We made implemented Fractions in C. We constructed algorithms that would take minimum memory and give results in reasonable time complexity. After implementing BIGINT, we used the Newton Raphson convergence method and Chudnovsky Algorithm to deduce the value of  $\pi$  up to 10000 decimal places.