Эффективная длинная арифметика

Студент Ли Дмитрий Сенунович
Б9121-09.03.03пикд
Руководитель доцент ИМКТ Кленин Александ Сергеевич

Навигация.


Предисловие.
...Предисловие...

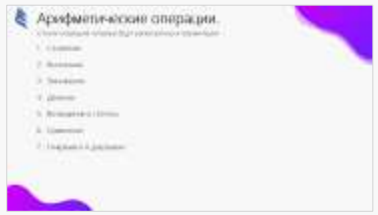

Введение.


Структура.


Дополнительные функции.


Ввод числа.
23472881
_digits


Вывод числа.
_digits 1 0 8 2 7 4 3 2
_positive: False


Арифметические операции.


Сложение.


Вычитание.


Умножение.


Деление.


Возведение в степень.


Сравнение.


Инкремент и декремент.


Анализ производительности.


Заключение.


Состав репозитория GitHub.

Предисловие.

Обозначения

В данной презентации созданы обозначения.

Обозначения в презентации:

- Заголовок «Предисловие.» – основная тема раздела презентации.
- Поставленный вопрос или тема «*Обозначения*»

Обозначения в представленном коде:

- Тип данных `int` – желтый
- Используемое из встроенных библиотек `vector` – зеленый
- Внутренние переменные `_digits` – красный
- Внутренние функции `_isZero()` – синий
- Внешние переменные `anyBigInt` – розовый
- Числа `423` – светло синий

Введение.

Что такое длинная арифметика?

Длинная арифметика - выполняемые с помощью вычислительной машины арифметические операции (сложение, вычитание, умножение, деление, возведение в степень, элементарные функции) над числами, разрядность которых превышает длину машинного слова данной вычислительной машины.

Эти операции реализуются не аппаратно, а программно, с использованием базовых аппаратных средств работы с числами меньших порядков.

Частный случай — арифметика произвольной точности — относится к арифметике, в которой длина чисел ограничена только объёмом доступной памяти.

Введение.

Для чего она применяется?

Длинная арифметика применяется в следующих областях:

- Криптография. Большинство систем подписывания и шифрования данных используют целочисленную арифметику по модулю m, где m — очень большое натуральное число, не обязательно простое.

- Математическое и финансовое ПО. Результат вычисления на бумаге должен совпадать с результатом работы компьютера с точностью до последнего разряда.

- Стандартная тема в спортивном программировании.

Структура.

Что нам нужно хранить, чтобы реализовать длинную арифметику?

1. Число, представленное в виде динамического массива.
   Как пример – `vector<int> _digits`

2. Знак числа. Как пример – `bool _positive`

Дополнительные функции.

Что нужно, чтобы код не только корректно работал, но и был лаконичен?

1. `void _removeLeftZeros()` – убрать (левые) лидирующие нули.
   Пример: `anyNumber._removeLeftZeros(); //00243 >> 243`

2. `void _afterOperation()` – выполняется после каждой операции.
   Алгоритм: убирает лидирующие нули; проверяет на нуль.

3. `void _doCarryOver(int start = 0)` – перенос переполненных разрядов.
   Используется в `_afterOperation()`

4. `bool _fitsInLongLong()` const – проверяет вмещается число в тип данных `long long` или нет.
   Пример: `anyNumber._fitsInLongLong(); //5345 >> true`

5. `long long _asLongLong() const` – преобразует длинное число в тип данных `long long`.
   Пример: `anyLongLong = anyBigInt._asLongLong();`

Дополнительные функции.

Что нужно, чтобы код не только корректно работал, но и был лаконичен?

6. `size_t _lenght() const` – возвращает длина числа.
   Пример: `anyBigInt._lenght()`

7. `bool _isOdd() const` – проверка на нечетное.
   Пример: `anyBigInt._isOdd() //23 >> true`

8. `bool _isEven() const` – проверка на четное.
   Пример: `anyBigInt._isEven() //23 >> false`

9. `bool _isZero() const` – проверка на нуль.
   Пример: `anyBigInt._isZero() //124 >> false`

10. `bool _isOne() const` – проверка на единицу.
    Пример: `anyBigInt._isOne() //1 >> true`

11. `bool _isPositive() const` – проверка на знак.
    Пример: `anyBigInt._isPositive() //-14253 >> false`

Дополнительные функции.

Что нужно, чтобы код не только корректно работал, но и был лаконичен?

12. `BigInt _times10(int times = 1) const` – добавление нулей (для деления).
    Пример: `anyBigInt._times10(3) //15._times10(3) >> 15000`

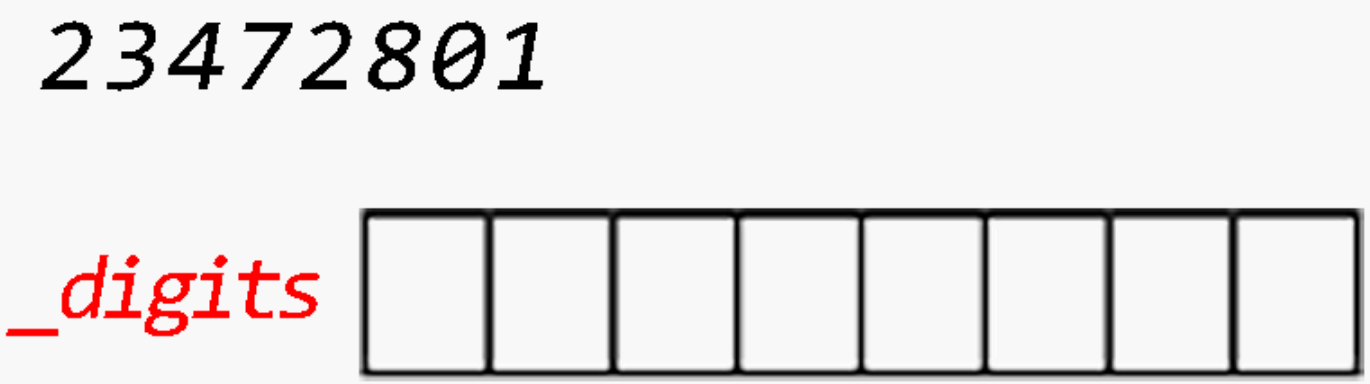
13. `BigInt _absoluteValue() const` – возвращает модуль числа.
    Пример: `anyBigInt._absoluteValue() //-124 >> 124`

Ввод числа.

Как число вписывается в переменную BigInt?

Как пример сохраним число 23 472 801. Число будет сохранено в массиве в обратном порядке. Для того, чтобы эффективнее создавать ячейки для разрядов выше.

В случае, если число отрицательное, то сначала мы внесем значение `false` в переменную `_positive`



В случае, если просмотр производится через PDF файл, анимация работать не будет.

Ввод числа.

Программный код.

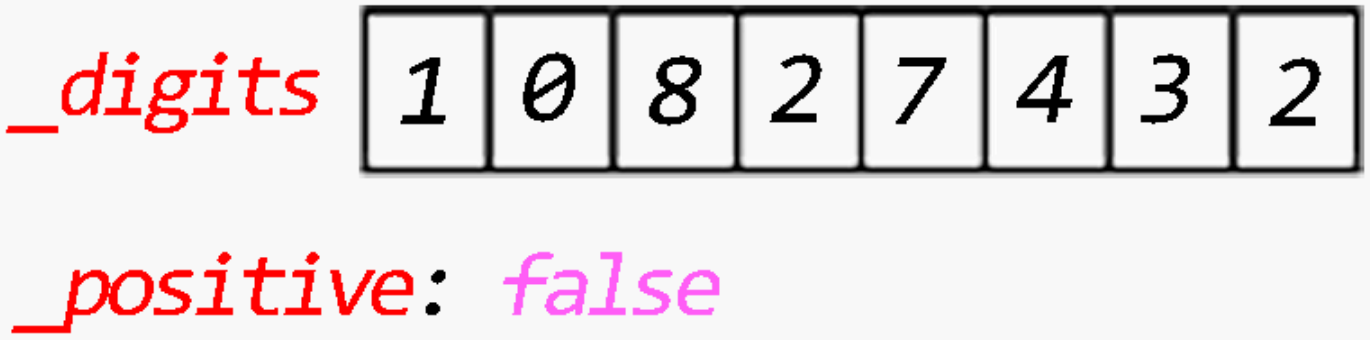
```
BigInt(std::string string) {
        if (string.length() > 0) {
                if (string.at(0) == '-' || string.at(0) == '+') {
                        this->_positive = string.at(0) == '+';
                        string = string.substr(1);
                }
                this->_digits.reserve(string.length());
                for (int i = string.size() - 1; i >= 0; i--) {
                        this->_digits.push_back(_charToInt(string[i]));
                }
        }
        else {
                this->_digits.push_back(0);
        }
        this->_removeLeftZeros();
}
```

Так же есть еще несколько функций ввода числа, в том числе и перегрузка оператора **cin>>**.

Вывод числа.

Как число BigInt выводится?

Как пример выведем то же число, но отрицательное.



В случае, если просмотр производится через PDF файл, анимация работать не будет.

Вывод числа.

Программный код.

```
friend std::ostream& operator<<(std::ostream& stream, BigInt const& number) {
        stream << number._asString();
        return stream;
}
std::string _asString() const {
        std::stringstream ss;
        if (!this->_isPositive()) {
                ss << '-';
        }
        for (int i = this->_lenght() - 1; i >= 0; i--) {
                ss << this->_digits[i];
        }
        return ss.str();
}
```

Арифметические операции.

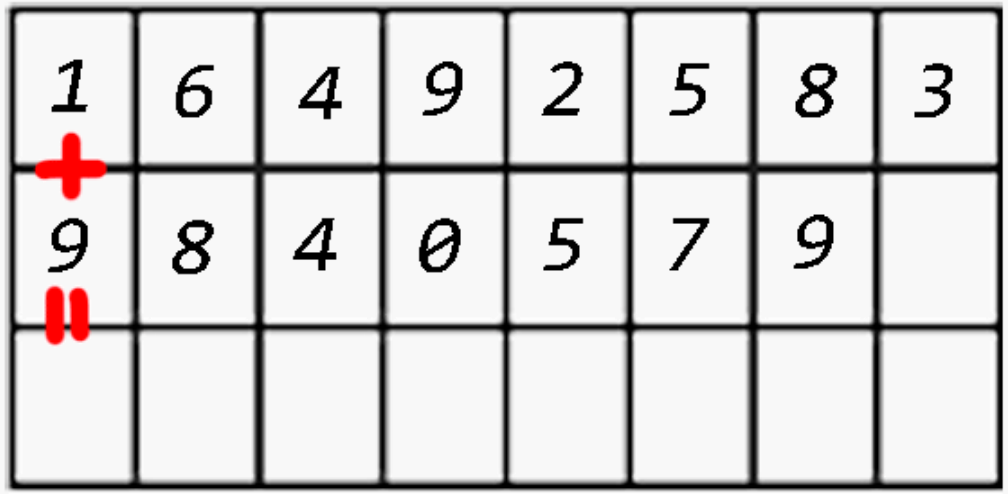
Список операций, которые будут расмотренны в презентации

1. Сложение
2. Вычитание
3. Умножение
4. Деление
5. Возведение в степень
6. Сравнение
7. Инкремент и декремент

Сложение.

Как происходит сложение двух чисел типа данных BigInt?

Как пример сложим числа 38 529 461 и 9 750 489.
В результате чего мы получим 05997284, что есть 48 279 950.



Так же существуют условия:

1. Выбирается число большей длинны.
2. Если одно из чисел отрицательное, то производится вычитание.
3. Если одно из чисел равно нулю, то возвращается другое число

Можем заметить, что реализовано сложение в столбик.

В случае, если просмотр производится через PDF файл, анимация работать не будет.

Сложение.

Программный код.

```
BigInt operator+(const BigInt& number) const {
            if (this->_isPositive() && !number._isPositive()) {
                        return *this - number._absoluteValue();
            }
            else if (!this->_isPositive() && number._isPositive()) {
                        return -(number - this->_absoluteValue());
            }
            if (number._isZero()) {
                        return *this;
            }
            if (this->_isZero()) {
                        return number;
            }

            BigInt result = *this;

            for (int i = 0; i < number._lenght(); i++) {
                        int digit = number._digits[i];
                        if (i < result._lenght()) {
                                    result._digits[i] += digit;
                        }
                        else {
                                    result._digits.push_back(digit);
                        }
            }
            result._doCarryOver();
            result._afterOperation();
            return result;
}
```

Вычитание.

Как происходит вычитание двух чисел типа данных BigInt?

Вычитание происходит аналогично сложению. То есть тоже по принципу в столбик. Так же сначала вычитается каждый разряд, затем, если число меньше 0, то прибавляется 10, при этом из следующего разряда вычитается 1.

Так же существуют условия аналогичные условиям сложения.

Вычитание.

Программный код.

```
BigInt operator-(const BigInt& number) const {
            BigInt result;
            const BigInt* smaller;
            if (this->_lenght() >= number._lenght()) {
                        result = *this;
                        smaller = &number;
            }
            else {
                        result = number;
                        smaller = this;
            }
            for (int i = 0; i < smaller->_lenght(); i++) {
                        int dif = result._digits[i] - smaller->_digits[i];
                        if (dif < 0) {
                                    for (int j = i + 1; j < result._lenght(); j++) {
                                                if (result._digits[j] == 0) {
                                                            result._digits[j] = 9;
                                                }
                                                else {
                                                            dif += 10;
                                                            result._digits[j]--;
                                                            break;
                                                }
                                    }
                        }
                        result._digits[i] = dif;
            }
            result._positive = *this >= number;
            result._afterOperation();
            return result;
}
```

Здесь не представлены условия.
С полным кодом можно ознакомиться на GitHub.

Умножение.

Как происходит умножение двух чисел типа данных BigInt?

Умножение происходит с использованием алгоритма Карацубы, который позволяет перемножать длинные числа со сложностью O(n^1.58).

Алгоритм происходит по принципу «Разделяй и властвуй».
То есть рекурсивно разделяет массив на подмассивы до тех пор, пока не получится с легкостью произвести умножение.
Затем, поднимаясь по дереву рекурсии, по специальному алгоритму соединяет части.

Так же существуют условия:
1. Если одно из чисел равно нулю, то результат равен нулю.
2. Если одно из чисел равно единицу, то результат равен другому числу.

Умножение.

Алгоритм Карацубы.

1. Проверка условий.
2. Проверка чисел по длине. (Если числа помещаются в тип данных long long, то производится обычное умножение. Нужно для ускорения алгоритма карацубы, потому что с маленькими числами алгоритм работает дольше по сравнению с обычным умножением).
3. Вычисляется максимальная длинна из двух чисел.
4. Числа делятся на 2 части: длиной половины от пункта 3
5. Производится рекурсивное умножение соответствующих частей двух чисел (До срабатывания условия в пункте 2 – числа будут продолжать делиться).
6. Производится рекурсивное умножение сумм разрядов двух чисел (Остановка рекурсии соответственно пункту 5).
7. Далее с помощью рекурсии восстанавливается уровень разрядов 3 частей из пунктов 5 (2 части) и 6.
8. Затем части соединяются обычным суммированием.

Умножение.

Программный код.

```
BigInt operator*(const BigInt& number) const {
        if (this->_isZero() || number._isZero()) {
                return BigInt(0);
        }
        if (this->_isOne()) {
                return number;
        }
        if (number._isOne()) {
                return *this;
        }
        if (this->_lenght() < 10 && number._lenght() < 10) {
                return BigInt(this->_asLongLong() * number._asLongLong());
        }
        int maxLenght = std::max(this->_lenght(), number._lenght());
        int splitPoint = maxLenght / 2;
        std::pair<BigInt, BigInt> splitThis = this->_splitAt(splitPoint);
        std::pair<BigInt, BigInt> splitNumber = number._splitAt(splitPoint);
        BigInt secondProduct = splitThis.second * splitNumber.second;
        BigInt firstProduct = splitThis.first * splitNumber.first;
        BigInt sumProduct = (splitThis.second + splitThis.first) * (splitNumber.second + splitNumber.first);
        BigInt firstPadded = firstProduct._times10(splitPoint * 2);
        BigInt deltaPadded = (sumProduct - firstProduct - secondProduct)._times10(splitPoint);
        return firstPadded + deltaPadded + secondProduct;
}
```

Деление.

Как происходит деление двух чисел типа данных BigInt?

Изначально функция для деления высчитывает сразу и целочисленное деление, и остаток. В последствии результаты будут хранится в `pair <BigInt, BigInt>`. Где первое и второе число соответственно равны результату целочисленного деления и остатку.

Цифры, которые возможно преобразовать в тип данных `long long`: мы будем сразу преобразовывать и делить встроенными методами.

Само деление будет происходить по принципу деления в столбик.

Деление.

Алгоритм деления.

1. Числа проверяются по условиям:
    1. Если делитель равен нулю, то выводит ошибку деления.
    2. Если делимое равно нулю, то возвращает нуль.
    3. Если числа равны, то целочисленное деление равно 1, а остаток равен 0.
    4. Если делитель больше делимого, то целочисленное деление равно 0, а остаток равен делимому.
2. Далее создаем переменные:
    1. Переменная **`mod`** – изначально хранит в себе модуль делимого (В последствии из которого будет вычитаться делитель, для выявления остатка).
    2. Переменная **`absoluteNumber`** – хранит в себе модуль делителя (Будет использоваться для того, чтобы производить деление в столбик).
    3. Пустая переменная **`div`** (В которую постепенно будет записываться результат деления.
3. Производим деление в цикле **`while`**: (Происходит обычное деление в столбик.

Деление.

Программный код.

```
std::pair<BigInt, BigInt> _divide(const BigInt& number) const {
        BigInt mod = this->_absoluteValue();
        const BigInt absoluteNumber = number._absoluteValue();
        BigInt div;
        int lenghDifference = mod._lenght() - absoluteNumber._lenght();
        while (lenghDifference-- >= 0) {
                BigInt toSubtract = absoluteNumber._times10(lenghDifference);
                while (mod >= toSubtract) {
                        div += BigInt(1)._times10(lenghDifference);
                        mod -= toSubtract;
                }
        }
        div._positive = this->_positive == number._positive;
        div._afterOperation();
        mod._afterOperation();
        return std::make_pair(div, mod);
}
```

Здесь не представлены условия.
С полным кодом можно ознакомиться на GitHub.

Возведение в степень.

Как происходит возведение в степень числа типа данных BigInt?

После создания операции умножения, создание операции возведения в степень становится одной из самых простых задач.

Достаточно проверить условия:

1. Если оба числа равны нулю, то вывести ошибку.
2. Если степень является отрицательным числом, то вывести ошибку.
3. Если число, возводимое в степень равняется нулю, то вернуть нуль.
4. Если степень равна единице, то вернуть число возводимое в степень.

Далее произвести рекусивное возведение в степень:

1. Проверить степень на четное или нечетное.
2. Если нечетная, то перемножить число 3 раза и возвести в степень (n-1)/2.
3. Если число четное, то перемножить число и возвести в степень n/2.

Возведение в степень.

Программный код.

```
BigInt pow(BigInt number) const {
        if (this->_isZero() && number._isZero()) {
                throw std::invalid_argument("Zero to the power of Zero is undefined.");
        }
        if (!number._isPositive()) {
                throw std::invalid_argument("Power cannot be negative.");
        }
        if (this->_isZero()) {
                return BigInt(0);
        }
        if (number._isZero()) {
                return BigInt(1);
        }
        if (number._isOdd()) {
                return *this * (*this * *this).pow((number - 1) / 2);
        }
        else {
                return (*this * *this).pow(number / 2);
        }
}
```

Сравнение.

Как происходит сравнение чисел типа данных BigInt?

Для написания всех функций сравнения достаточно написать две функции: обязательно функцию == и на выбор: > или <, затем встроенными операциями без проблем получится создать оставшиеся.

Сравнение происходит по следующему алгоритму:

1. Если второе число отрицательное, а первое положительное, то второе число меньше первого.
2. Если числа либо оба положительные, либо оба отрицательные, то если длина первого числа больше чем длина второго, то второе число меньше первого.
3. Если длинны равны, то сравнивается каждый разряд до тех пор, пока не найдется больший разряд.
4. В случае если каждый разряд равен, то числа равны.

Инкремент и декремент.

Как происходит инкрементирование и декрементирование типа данных BigInt?

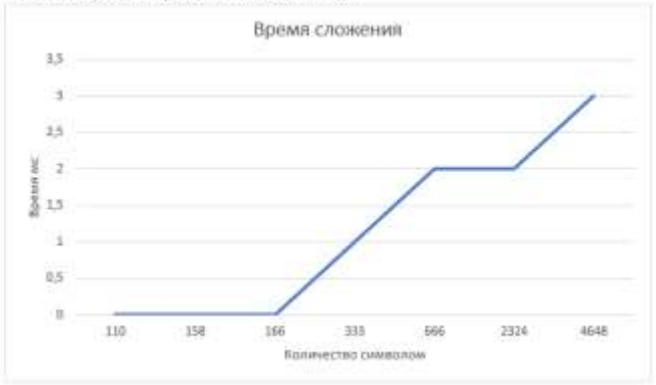
Учитывая написанные операции сложения и вычитания, то для написания функций инкрементирования и декрементирования достаточно либо добавить 1, либо отнять 1.

Анализ производительности.

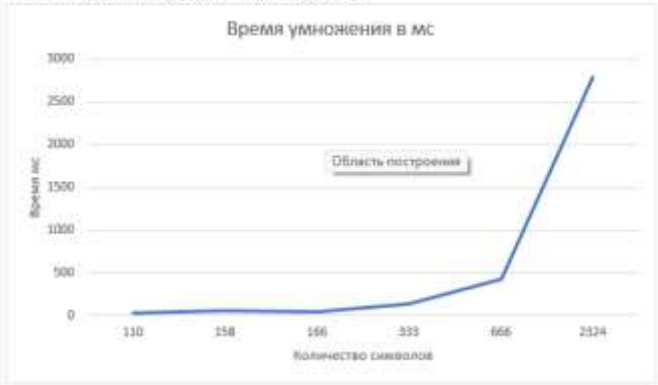
Тесты скорости работы.

Все тесты скорости работы предоставлены в реферате, который находится на GitHub. Ниже можно ознакомиться с графиками:

Тест скорости работы сложения

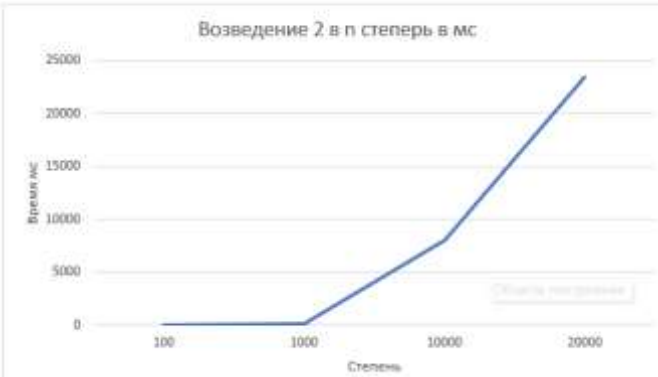


Тест скорости работы умножения

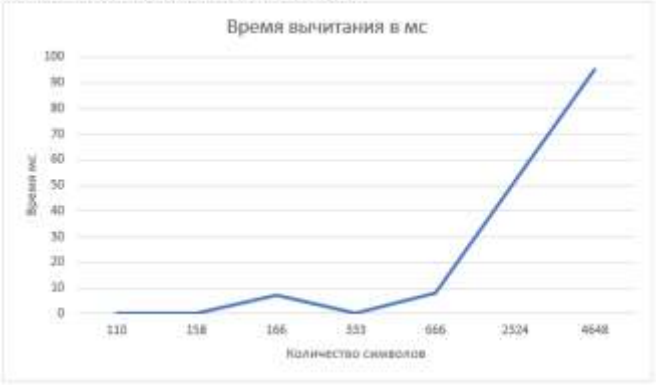


Тест скорости работы возведения в степень

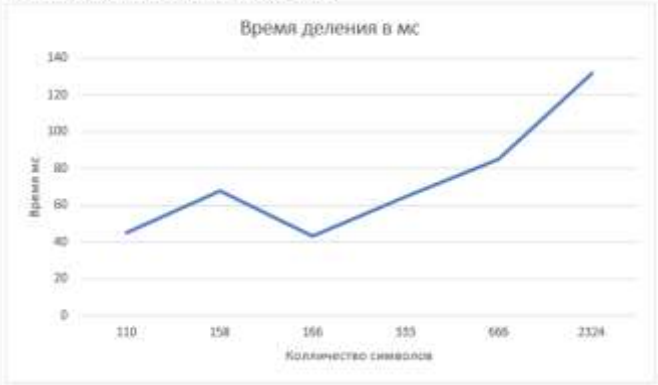
Изначально рассматривается возведение 2 в n степень.



Тест скорости работы вычитания



Тест скорости работы деления



Заключение.

В ходе изучения эффективной длинной арифметики были реализованы операции:

- Сложения
- Вычитания
- Умножения
- Деления (Целочисленное и получение остатка)
- Возведение в степень
- Сравнения

Был изучен материал в большом количестве.

А так же были произведены тесты скорости работы.

Состав репозитория GitHub.

В репозиторий было добавлено:

1. Реферат об алгоритме
2. Исходный код алгоритма
3. Тестирующая система
4. Презентация

Ссылка на GitHub: https://github.com/Elsium/ASD