МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

**по дисциплине  
 «АЛГОРИТМЫ И АНАЛИЗ СЛОЖНОСТИ»**

Выполнил студент группы 25/2                                       А.А. Козин

Направление подготовки  02.03.03  Математическое обеспечение и администрирование информационных систем

Курс    2

Отчет принял доктор физико-математических наук, профессор                                                                                       А.И. Миков

Краснодар

2021 г.

**Задание**: Найти функцию сложности для алгоритма Карацубы.

**Решение**.

Для анализа сложности алгоритма Карацубы мы приведём его в буквальном виде, поставив счётчик в операции умножении.

**Текст программы:**

#include <iostream>

#include <vector>

#include <ctime>

using namespace std;

int counter = 0, bz = 0;

vector<int> sub\_items(const vector<int>& v, const int from, const int to)

{

return vector<int>(v.begin() + from, v.begin() + to);

}

vector<int> add\_trailing\_zeros(const vector<int>& a, const int count)

{

vector<int> res = a;

for (int i = 0; i < count; ++i) res.insert(res.begin(), 0);

return res;

}

vector<int> remove\_leading\_zeros(const vector<int>& a)

{

vector<int> res = a;

while (res.size() > 1 && res.back() == 0) res.pop\_back();

return res;

}

vector<int> big\_sum(const vector<int>& a, const vector<int>& b)

{

vector<int> res;

int carry = 0;

for (int i = 0; i < max(a.size(), b.size()); ++i)

{

if (i < a.size()) carry += a[i];

if (i < b.size()) carry += b[i];

res.push\_back(carry % bz);

carry /= bz;

}

if (carry > 0) res.push\_back(carry);

return res;

}

vector<int> big\_difference(const vector<int>& a, const vector<int>& b)

{

vector<int> res;

int carry = 0;

for (int i = 0; i < a.size(); ++i)

{

carry += a[i] - (i < b.size() ? b[i] : 0);

res.push\_back(carry < 0 ? carry + bz : carry);

carry = carry < 0 ? -1 : 0;

}

return remove\_leading\_zeros(res);

}

vector<int> big\_product(const vector<int>& a, const int digit)

{

vector<int> res;

int carry = 0;

for (const int item : a)

{

carry += item \* digit;

res.push\_back(carry % bz);

carry /= bz;

}

if (carry > 0) res.push\_back(carry);

return remove\_leading\_zeros(res);

}

vector<int> karats(const vector<int>& a, const vector<int>& b)

{

if (a.size() == 1) { counter++; return big\_product(b, a[0]); }

if (b.size() == 1) { counter++; return big\_product(a, b[0]); }

const int m = min(a.size() / 2, b.size() / 2);

const vector<int> a1 = sub\_items(a, 0, m);

const vector<int> a2 = sub\_items(a, m, a.size());

const vector<int> b1 = sub\_items(b, 0, m);

const vector<int> b2 = sub\_items(b, m, b.size());

const vector<int> z0 = karats(a1, b1);

const vector<int> z1 = karats(big\_sum(a1, a2), big\_sum(b1, b2));

const vector<int> z2 = karats(a2, b2);

const vector<int> step1 = add\_trailing\_zeros(z2, m \* 2);

const vector<int> step2 = big\_difference(z1, big\_sum(z0, z2));

const vector<int> step3 = add\_trailing\_zeros(step2, m);

const vector<int> step4 = big\_sum(step1, step3);

return big\_sum(step4, z0);

}

vector<int> getVector(int n)

{

vector<int> res;

for (int i = 0; i < n; i++) res.push\_back(rand() % bz);

return res;

}

int main()

{

srand(time(0));

setlocale(LC\_ALL, "RU");

cout << "Введите bz:";

cin >> bz;

cout << "i " << " " << "counter" << endl;

for (int i = 1; i <= 100; i++)

{

const vector<int> a = getVector(i);

const vector<int> b = getVector(i);

const vector<int> res = karats(a, b);

cout << i << " " << counter << endl;

counter = 0;

}

return 0;

}

Программа выдает на консоль результаты в следующем виде:

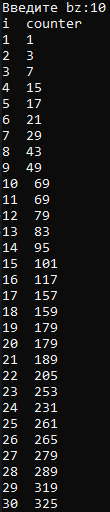


График роста количества операций:

**Анализ зависимости:**

Высчитывая формулу сложности алгоритма Карацубы применим теорему Мастера:

Т.к , где с - сложность каждого этапа рекурсии(и везде у нас только сложение и вычитание, а значит сложность линейная), а – количество рекурсивных подзадач в алгоритме(три умножения, значит, а=3), размерность исходной задачи каждый раз уменьшается в b раз(в 2 раза), тогда, подставив значения, получим:

А значит, сложность алгоритма примерно равна .

**Вывод**: Сложность алгоритма Карацубы: