

|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

|  |  |
| --- | --- |
| Отчет по выполнению практического задания №1 | |
| **Тема:** Оценка сложности и определение эффективности алгоритма | |
| **Дисциплина: «**Структуры и алгоритмы обработки данных**»** | |
| Выполнил студент | Антонов А.Д. |
| группа | ИКБО-01-20 |

Москва 2021

# Содержание

[Задание 1 4](#_bookmark0)

1. [Постановка задачи 4](#_bookmark1)
2. [Модель решения поставленной задачи в первом алгоритме 4](#_bookmark2)
   1. [Описание алгоритма 4](#_bookmark3)
   2. [Инвариант цикла 4](#_bookmark4)
   3. [Определение вычислительной сложности алгоритма 5](#_bookmark5)
3. [Алгоритм в виде функции 7](#_bookmark6)
4. [Функции заполнения массива случайными числами и вывода на экран 7](#_bookmark7)
5. [Результаты тестирования 8](#_bookmark8)
6. [Результаты тестирования крайних случаев 8](#_bookmark9)
7. [Модель решения поставленной задачи в первом алгоритме 10](#_bookmark10)
   1. [Описание алгоритма 10](#_bookmark11)
   2. [Инвариант цикла 10](#_bookmark12)
   3. [Определение вычислительной сложности алгоритма 10](#_bookmark13)
8. [Алгоритм в виде функции 11](#_bookmark14)
9. [Функции заполнения массива случайными числами и вывода на экран 12](#_bookmark15) [10. Результаты тестирования 12](#_bookmark16)

[11. Результаты тестирования крайних случаев 12](#_bookmark17)

[ВЫВОД 14](#_bookmark18)

[Задание 2. Выполнение индивидуального задания в соответствии с вариантом](#_bookmark19)  [15](#_bookmark19)

[1. Постановка задачи 15](#_bookmark20)

[2. Модель решения 15](#_bookmark21)

1. [Разработка эффективного алгоритма 15](#_bookmark22)
   1. [Алгоритм 15](#_bookmark23)
   2. [Инварианты 15](#_bookmark24)
   3. [Определение вычислительной сложности алгоритма 16](#_bookmark25)
   4. [Реализация алгоритма в виде функции 18](#_bookmark26)
   5. [Тестирование алгоритма 18](#_bookmark27)
   6. [Практическая оценка сложности алгоритма 19](#_bookmark28)

[ВЫВОДЫ 21](#_bookmark29)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 21](#_bookmark30)

1. **Постановка задачи**

# Задание 1

Определить эффективный алгоритм из двух предложенных, используя оценку теоретической сложности каждого из алгоритмов и емкостную сложность, решения следующей задачи: дан массив из n элементов целого типа, удалить из массива все значения равные заданному.

Таблица 1: предложенные алгоритмы на псевдокоде

|  |  |
| --- | --- |
| x-массив, n – количество элементов в массиве, key – удаляемое значение | |
| Алгоритм 1. delFirstMetod(x,n,key){ i←1  while (i<=n) **do**  if x[i]=key then  //удаление  for j←i to n-1 do x[j] ←x[j+1]  od n←n-1  else  i←i+1  endif  **od**  } | Алгоритм 2 delOtherMetod(x,n,key){ j←1  for i←1 to n **do**  x[j]=x[i];  if x[i]!=key then j++  endif **od n**←j  } |

## Модель решения поставленной задачи в первом алгоритме

## Описание алгоритма:

С помощью цикла while находим элементы массива равные key, с помощью вложенного цикла от i до n-1 смещаем все элементы влево и уменьшаем значение n на единицу, если элемент массива не равен key, то просто переходим к следующему, увеличивая i.

## Инвариант цикла:

* + 1. i находится в промежутке от [0, n-1],
    2. j находится в промежутке от [i, n-2]

## Определение вычислительной сложности алгоритма

|  |  |  |  |
| --- | --- | --- | --- |
| Номер оператора | Оператор | Время выполнения одного оператора | Кол-во выполнений оператора в строке |
| Номер столбца | 1 | 2 | 3 |
| 1 | int i = 0; | C1 | 1 раз |
| 2 | while (i < n){ | C2 | n+1 раз |
| 3 | if (x[i] == key){ | C3 | n раз |
| 4 | for (int j = i; j < n - 1; j++){ | C4 | n\*(n+1) раз |
| 5 | x[j] = x[j + 1]; | C5 | n\*(n+1)-1 раз |
| 6 | n = n - 1;} | C6 | n раз |
| 7 | else{i++;}} | C7 | n-nC3 раз |

Таблица 2. Подсчет количества операторов алгоритма

Оператор 1. Выполняется один раз.

Оператор 2. В худшем случае – в массиве нет заданного значения, а значит n не уменьшается – выполнится n+1 раз, n раз с заходом внутрь цикла и 1 раз с проверкой, когда I = n.

Оператор 3 (if). Это оператор тела цикла, т.е. он выполняется n раз, сколько количество входов в тело цикла.

Оператор 4. Вложенный цикл. В худшем случае выполнится n\*(n+1) раз Оператор 5. Выполнится соответственно вложенному циклу, без одной

операции: n\*(n+1)-1 раз

Оператор 7 (else). Выполнится (n-nC3).

Определим функцию роста для времени выполнения алгоритма в худшем случае:

T(n)= C1 + C2\*(n + 1) + C3\*n + C4\*n\*(n + 1) + C5\*(n\*(n - 1) - 1)+ C6\*n+C7\*n

=(С4+С5)\*n2 +(C2+C3+C4+C5+C6+C7)\*n + (C1+C2+\*C4-\*C5) = An2 + Bn+C

Пренебрегаем константой С. Получаем T(n) = An2+Bn. Функция n2 имеет порядок роста выше, чем функция n. T(n)= An2+Bn, доминирующей функцией

является n2, и она определяет порядок роста для алгоритма в худшем случае. Т.е. T(n)=Ө(n2).

Выведем функцию роста для времени выполнения алгоритма в лучшем случае (введенного значения в массиве нет):

T(n) = С1\*1+С2(n+1)+C3\*n+C7\*n=An+B

Порядок роста времени в зависимости от n в наилучшем случае линейный, т.е. T(n)=Ө(n).

## Алгоритм в виде функции:

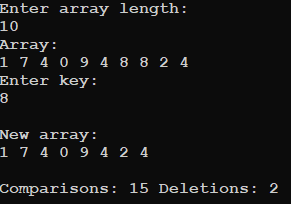
## 

## Функции заполнения массива случайными числами и вывода на экран

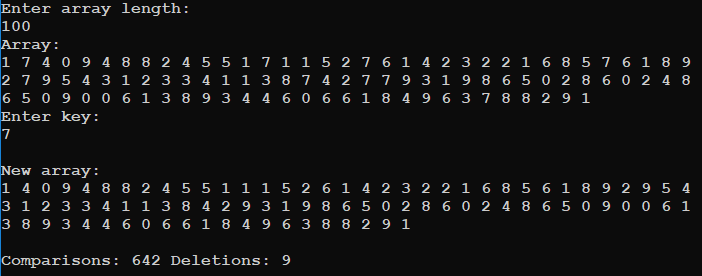
## 

## Результаты тестирования

Для n = 10:

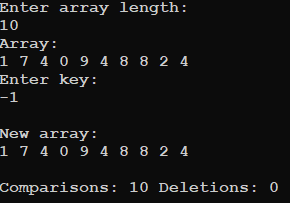


Для n = 100:

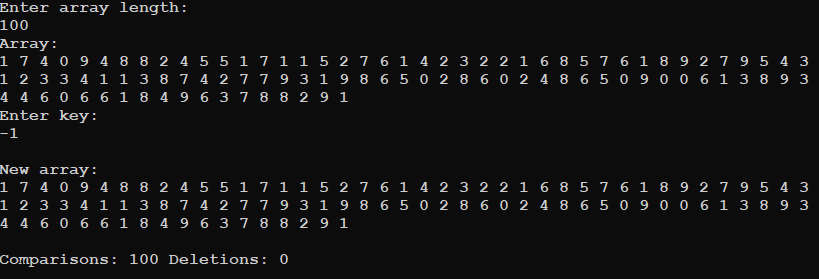


Как видно, при увеличении n на 1 порядок, кол-во удалений возрастает на два порядка, что подтверждает полученный порядок роста T(n)=O(n2)

1. **Результаты тестирования крайних случаев:** Лучший случай (нет чисел равных заданному) Для n = 10:

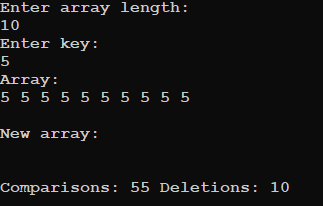


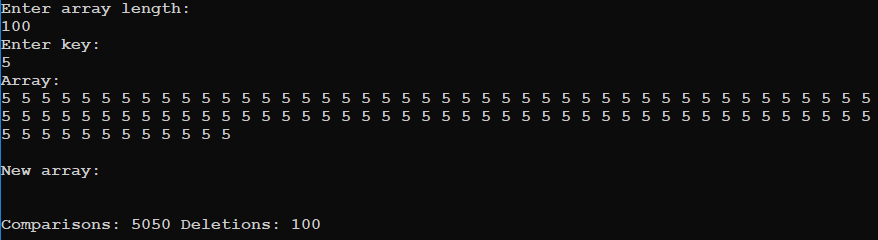
Для n =100:



Как видно, в лучшем случае кол-во операций линейно зависит от размера массива, что подтверждает полученный порядок роста для лучшего случая T(n) = Ө(n).

Худший случай (все числа равных заданному) Для n = 10:



Для n=100:

Как видно, в худшем случае кол-во операций квадратично зависит от размера массива, что подтверждает полученный порядок роста для худшего случая T(n) = O(n2).

## Модель решения поставленной задачи во втором алгоритме

## Описание алгоритма

С помощью цикла for проходимся по всем элементам массива, при входе в цикл мы записываем выбранный элемент в самого себя же, если до этого нам не встретились числа равные заданному, в противном случае мы перезаписываем выбранный элемент внутрь элемента равного заданному, т.е. производим удаление.

## Инвариант цикла

* + 1. i находится в промежутке от [0, n-1],

## Определение вычислительной сложности алгоритма

Таблица 3. Подсчет количества операторов алгоритма 2

|  |  |  |  |
| --- | --- | --- | --- |
| Номер оператора | Оператор | Время выполнения одного оператора | Кол-во выполнений оператора в строке |
| Номер столбца | 1 | 2 | 3 |
| 1 | int j = 1; | C1 | 1 раз |
| 2 | for (int i = 0; i < n; i++){ | C2 | n+1 раз |
| 3 | x[j] = x[i]; | C3 | n раз |
| 4 | if (x[i] != key){ | C4 | n раз |
| 5 | j++;}} | C5 | n раз |
| 6 | n = j;} | C6 | n раз |

Оператор 1 выполняется один раз.

Оператор 2. Согласно циклу с предусловием: первый вход в цикл при i=0; последний вход в цикл при i=n-1; после последнего входа i=n, т.е. ещё одна проверка и завершение цикла. Считаем сколько раз выполнялся оператор i <n: n раз обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n+1 раз за время работы.

Оператор 3. Выполнится n раз.

Оператор 4 (if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 5. Это оператор – блок оператора if. Выполняется столько раз, сколько и if (в худшем случае) – n раз.

Оператор 6. Выполнится n раз.

Определим время выполнения алгоритма - как сумму времени выполнения каждого оператора:

T(n)=С1\*1+С2\*(n+1)+C3\*n+C4\*n+C5\*n+C6\*n = (C2+C3+C4+C5+C6)n + (C1 + C2) = An + B.

В результате T(n) в худшем случае линейно зависит от n.

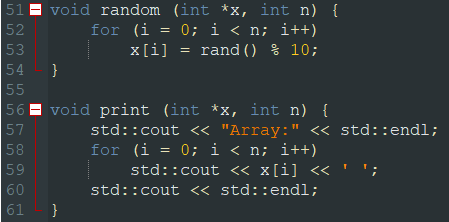
В лучшем и среднем случае порядок роста все равно будет линейно зависит от n, так как цикл будет все равно выполняться n+1 раз.

Вывод. Порядок роста T(n)=Ө(n).

## Алгоритм в виде функции

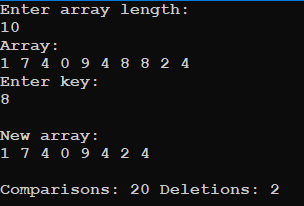
## 

## Функции заполнения массива случайными числами и вывода на экран

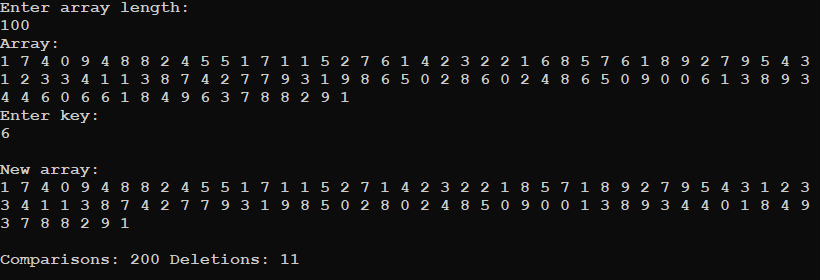


## Результаты тестирования

Для n = 10:

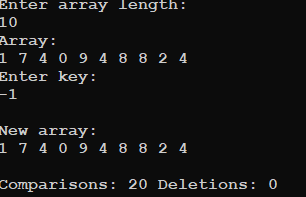


Для n = 100:

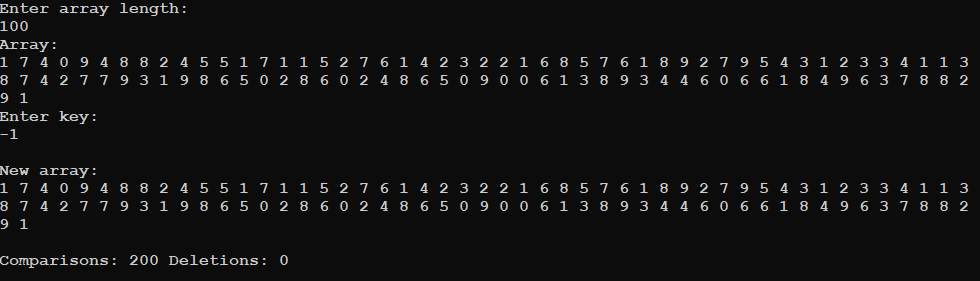


Как видно при увеличении n на 1 порядок, количество операций увеличивается также на 1 порядок, что подтверждает порядок роста T(n)=Ө(n).

1. **Результаты тестирования крайних случаев** Лучший случай (нет чисел равных заданному) Для n = 10:

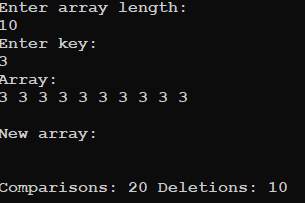


Для n =100:

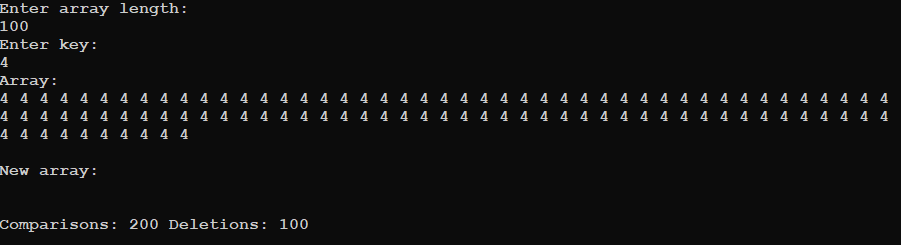


Как видно, в лучшем случае кол-во операций линейно зависит от размера массива, что подтверждает полученный порядок роста для лучшего случая T(n) = Ө(n).

Худший случай (все числа равны заданному) Для n = 10:



Для n =100:



Как видно, в худшем случае кол-во операций также линейно зависит от размера массива, что подтверждает полученный порядок роста для худшего случая T(n) = Ө(n).

# ВЫВОД

В ходе выполнения практического задания для данных алгоритмов были определены инварианты внешнего и внутренних циклов. Также была определена вычислительная сложность обоих алгоритмов теоретическим подходом. На основании оценки вычислительной сложности было выявлено, что второй алгоритм является более эффективным. Обе версии алгоритма были реализованы на C++ и протестированы. Теоретические оценкам вычислительной сложности совпадают с результатом практических тестировании представленного алгоритма.

# Задание 2. Выполнение индивидуального задания в соответствии с вариантом. Вариант 1.

## 1. Постановка задачи

Умножение квадратных матриц.

## 2. Модель решения

Считаем значение каждого элемента результирующей матрицы - оно равно произведению строки первой матрицы на столбец второй матрицы. С помощью вложенного цикла проходим по каждой паре строка-столбец, и с помощью еще одного цикла считаем значение элемента.

## Разработка эффективного алгоритма

## Алгоритм

Проходимся по парам строк первого двумерного массива и столбцов второго двумерного массива, суммируем произведения элементов в них.

## Инварианты

Цикла i: i принадлежит от [0..n-1]

Цикла j: j принадлежит от [0..n-1]

Цикла k: k принадлежит от [0..n-1]

Циклы корректны, т.к. в любой момент времени рассматриваем элементы массивов с индексом от [0][0] до [n-1][n-1], т.е. все элементы массивов.

## Определение вычислительной сложности алгоритма

Таблица 4. Подсчет количества операторов в алгоритме

|  |  |  |  |
| --- | --- | --- | --- |
| Номер оператора | Оператор | Время выполнения одного оператора | Кол-во выполнений оператора в строке |
| Номер столбца | 1 | 2 | 3 |
| 1 | int temp; | C1 | 1 |
| 2 | for (i=0; i<n; i++) | C2 | n+1 |
| 3 | for (j=0; j<n; j++) | C3 | n\*(n+1) |
| 4 | temp = 0; | C4 | n2 |
| 5 | for (k=0; k<n; k++) | C5 | (n+1)\*n2 |
| 6 | temp += a[i][k] \* b[k][j] | C6 | n3 |
| 7 | std::cout << temp; | C7 | n2 |
| 8 | std::cout << std::endl; | C8 | n |

Оператор 1. Выполняется 1 раз.

Оператор 2. Согласно циклу с предусловием: первый вход в цикл при i=0; последний вход в цикл при i=n-1; после последнего входа i=n, т.е. ещё одна проверка и завершение цикла. Считаем сколько раз выполнялся оператор i<n: n раз обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n+1 за время работы.

Оператор 3. Согласно циклу с предусловием: первый вход в цикл при j=0; последний вход в цикл при j=n-1; после последнего входа j=n, т.е. ещё одна проверка и завершение цикла. Считаем сколько раз выполнялся оператор j<n: n раз обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n+1 за время работы, и, так как цикл внутри другого цикла, общее выполнение будет равно n\*(n+1)

Оператор 4. Выполняется внутри двух циклов, соответственно n2 раз

Оператор 5. Согласно циклу с предусловием: первый вход в цикл при k=0; последний вход в цикл при k=n-1; после последнего входа k=n, т.е. ещё одна проверка и завершение цикла. Считаем сколько раз выполнялся оператор k<n: n раз обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n+1 за время работы, и, так как цикл внутри другого цикла, общее выполнение будет равно (n+1)\*n2

Оператор 6. Выполняется внутри трех циклов, соответственно n3 раз

Оператор 7. Выполняется внутри двух циклов, соответственно n2 раз

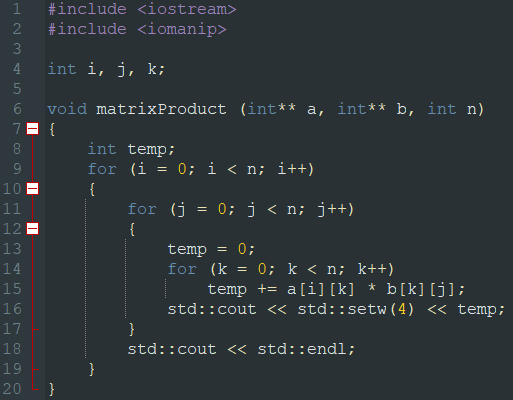
Оператор 8. Выполняется внутри цикла, соответственно n раз

Определим время выполнения алгоритма - как сумму времени выполнения каждого оператора:

T(n) = С1 + С2\*(n+1) + C3\*(n2+n) + C4\*n2 + C5\*(n3+n2) + C6\*n3 + C7\*n2 + C8\*n = (C5+C6)\*n3 + (C3+C4+C5+C7)\*n2 + (C2+C3+C8)\*n + (C1+C2) = An3 + Bn2 + Cn + D

Пренебрегаем константой D. Получаем T(n) = An3+Bn2+Cn. Функция n3 имеет порядок роста выше, чем функции n2 и n. T(n) = An3+Bn2+Cn, доминирующей функцией является n3, и она определяет порядок роста для алгоритма в худшем случае. Т.е. T(n)=Ө(n3).

## Реализация алгоритма в виде функции



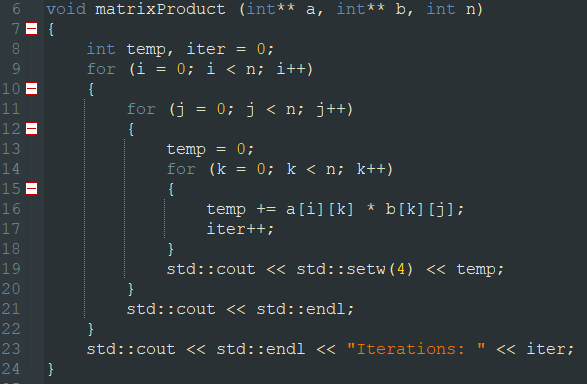
## Тестирование алгоритма.

|  |  |  |
| --- | --- | --- |
| Входные значения | Ожидаемый вывод | Вывод |
| 2  1 2  3 4  1 0  0 1 | 1 2  3 4 |  |
| 3  1 2 3  4 5 4  3 2 1  5 4 3  2 1 2  3 4 5 | 18 18 22  42 37 42  22 18 18 |  |

Из тестов видно, что алгоритм работает согласно описанию и модели.

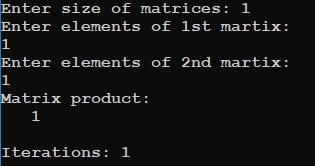
## Практическая оценка сложности алгоритма

Для оценки практической сложности будем считать количество операций внутри каждого цикла. Изменим код следующим образом:

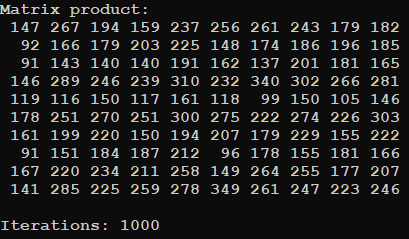


Добавим переменную iter, которая будут считать количество операций, совершенных внутренними циклами для оценки сложности.

Для n = 1 (лучший случай):

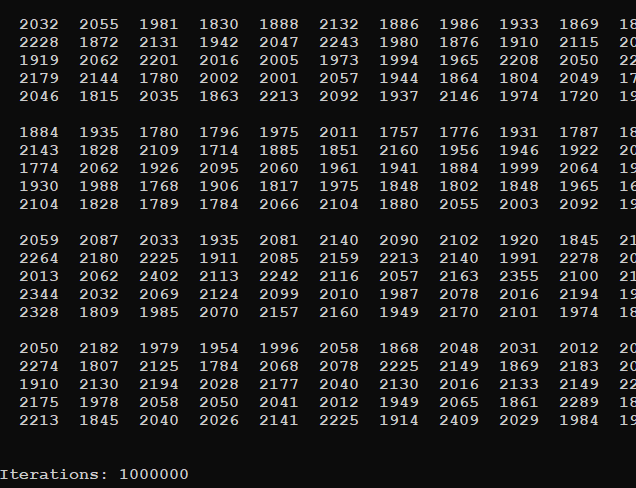


Для n = 10:



Можно заметить, что при увеличении значения n на порядок, количество операций увеличивается на три порядка, что подтверждается тем, что порядок роста для алгоритма T(n)=Ө(n3).

Прогон на n = 100 (не вмещается на экран):



# ВЫВОДЫ

В ходе данной практической работы мы изучили определение сложности алгоритма, научились теоретически определять порядок роста, разработали алгоритм в соответствии с требованием индивидуального задания, определили его порядок роста и на практике подтвердили полученный результат.

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

* + 1. Лекционный материал по структуре и алгоритмам обработки данных Гданского Н.И.
    2. Теоретический материал по структурам и алгоритмам обработки данных.
    3. Кораблин Ю.П., Сыромятников В.П., Скворцова Л.А. Учебно-методическое пособие Структуры и алгоритмы обработки данных, М.:МИРЭА, 2020
    4. Методичка с примерами определения функции роста.