|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

| **Отчет по выполнению практического задания № 1** | |
| --- | --- |
| **Тема:** | |
| **«Оценка вычислительной сложности алгоритма»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Жаворонкова А.А. |
|  |  |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 3](#_30j0zll)

[2 ЗАДАНИЕ №1 4](#_1fob9te)

[2.1 Формулировка задачи 4](#_3znysh7)

[2.2 Математическая модель решения алгоритмов 4](#_2et92p0)

[2.2.1 Описание выполнения и блок-схема первого алгоритма 4](#_tyjcwt)

[2.2.2 Доказательство корректности циклов первого алгоритма 5](#_3dy6vkm)

[2.2.3 Определение вычислительной сложности первого алгоритма 6](#_1t3h5sf)

[2.2.4 Описание выполнения и блок-схема второго алгоритма 7](#_4d34og8)

[2.2.5 Доказательство корректности циклов второго алгоритма 8](#_2s8eyo1)

[2.2.6 Определение вычислительной сложности второго алгоритма 9](#_17dp8vu)

[2.3 Реализация алгоритмов на языке C++ 9](#_3rdcrjn)

[2.4 Тестирование 12](#_26in1rg)

[2.4.1 Тестирование алгоритмов в ситуации случайного заполнения массива 12](#_lnxbz9)

[2.4.2 Тестирование алгоритмов в ситуации все элементы массива должны быть удалены 13](#_35nkun2)

[2.4.3 Тестирование алгоритмов в ситуации ни один элемент не удаляется 15](#_1ksv4uv)

[2.5 Вывод по заданию №1 16](#_44sinio)

[3 ЗАДАНИЕ №2 17](#_2jxsxqh)

[3.1 Описание математической модели 17](#_z337ya)

[3.2 Описание математической модели 17](#_3j2qqm3)

[3.2.1 Описание выполнения и блок-схема первого алгоритма 17](#_1y810tw)

[3.3 Доказательство корректности циклов 17](#_2xcytpi)

[3.4 Определение вычислительной сложности алгоритма 18](#_1ci93xb)

[3.5 Реализация алгоритма на языке С++ 18](#_3whwml4)

[3.6 Тестирование 20](#_2bn6wsx)

[3.7 Выводы по заданию 7 22](#_qsh70q)

[4 ВЫВОДЫ 23](#_3as4poj)

[5 ЛИТЕРАТУРА 24](#_qcm90y5wxcpc)

# 1 ЦЕЛЬ

Приобретение практических навыков:

* Эмпирическому определению вычислительной сложности алгоритмов на теоретическом и практическом уровнях;
* Выбору эффективного алгоритма решения вычислительной задачи из нескольких.
* Разработка собственного алгоритма в соответствии с задачей.

# 2 ЗАДАНИЕ №1

## **2.1 Формулировка задачи**

Выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его емкостную сложность. Пусть имеется вычислительная задача:

– дан массив х из n элементов целого типа; удалить из этого массива все значения равные заданному (ключевому) key.

Удаление состоит в уменьшении размера массива с сохранением порядка следования всех элементов, как до, так и следующих после удаляемого.

## **2.2 Математическая модель решения алгоритмов**

### **2.2.1 Описание выполнения и блок-схема первого алгоритма**

Алгоритм выполняется с помощью цикла, который идёт по массиву с первого элемента до элемента n (размер цикла). Если элемент равен заданному значению для удаления, то все следующие значения в массиве смещаются на 1 позицию влево. Таким образом удаляется элемент, который равен заданному значению. Так как элемент удалился, то и размер массива стал меньше, а следовательно, переменную n уменьшаем на 1. Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.1).

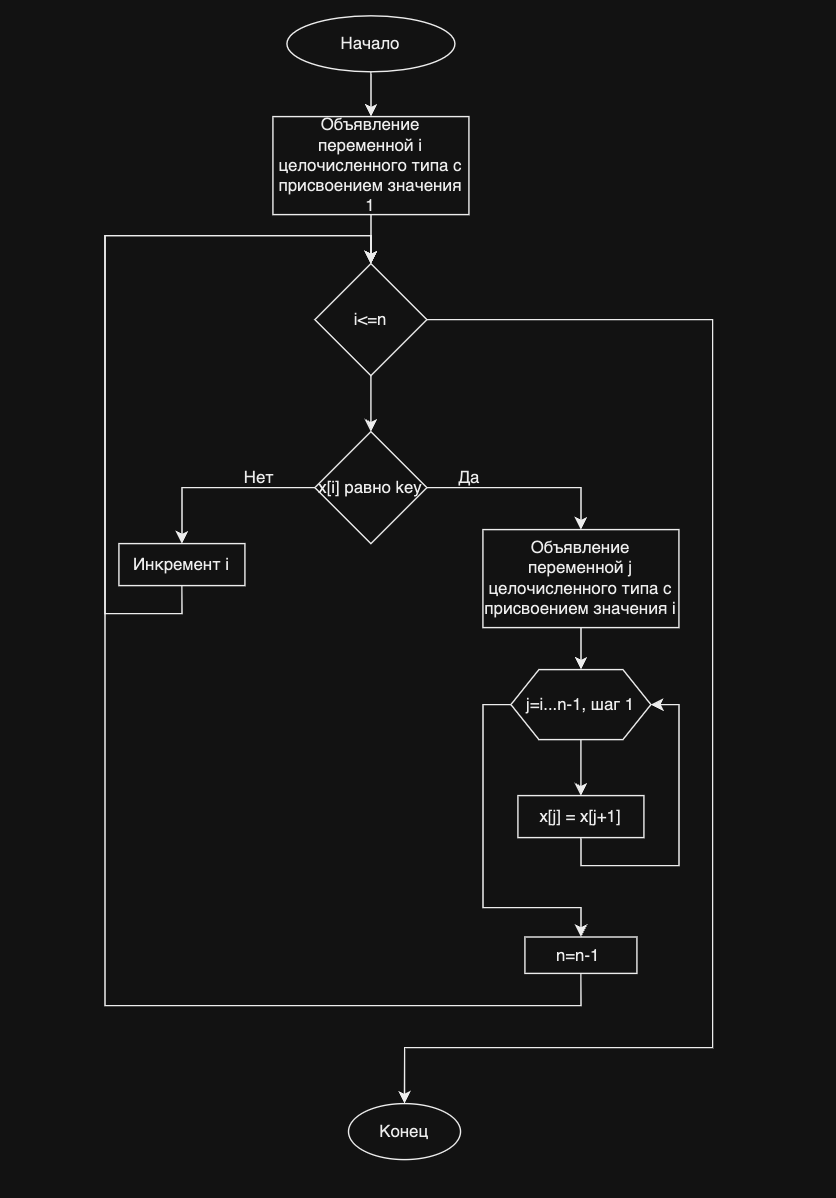


Рисунок 1 – Блок-схема первого алгоритма

### **2.2.2 Доказательство корректности циклов первого алгоритма**

Инвариант для внешнего цикла: значение переменной i всегда неотрицательно и меньше или равно n.

Инвариант для внутреннего цикла: значение переменной j всегда меньше n.

Докажем конечность цикла. При каждом повторении цикла значение переменной i либо увеличивается на 1, либо уменьшается (когда удаляется элемент). Также во внутреннем цикле происходит удаление элемента и уменьшение переменной n на 1, следовательно размер массива уменьшается после каждого удаления элемента. Следовательно, цикл в данном алгоритме является конечным, так как каждое повторение цикла либо уменьшает значение переменной n, либо увеличивает значение переменной i, и процесс не может продолжаться бесконечно.

Из доказательства можно сделать вывод, что все циклы данного алгоритма корректны.

### **2.2.3 Определение вычислительной сложности первого алгоритма**

| Номер строки | Алгоритм, записанный на псевдокоде | Количество выполнений оператора в худшем случае | Количество выполнений оператора в лучшем случае |
| --- | --- | --- | --- |
| 1 | delFirstMetod(x,n,key){ |  |  |
| 2 | i←1 | 1 | 1 |
| 3 | while (i<=n) **do** | n+1 | n |
| 4 | if x[i]=key then | n |  |
| 5 | for j←i to n-1 do |  |  |
| 6 | x[j] ←x[j+1] |  |  |
| 7 | оd |  |  |
| 8 | n←n-1 | n |  |
| 9 | else |  |  |
| 10 | i←i+1 |  | n |
| 11 | endif |  |  |
| 12 | **od** |  |  |
| 13 | } |  |  |

В строке 6 количество повторений действия является арифметической прогрессией. Сумма арифметической прогрессии (формула 1).

*(1)*

Вычислительную сложность алгоритма в худшем случае можно определить функцией. Это квадратичная функция, что означает, что алгоритм имеет квадратичный порядок роста времени вычисления. Средний случай соответствует худшему случаю.

Вычислительную сложность алгоритма в лучшем случае можно определить функцией T(n)=2n+2. В данном случае ни один элемент удалять не нужно.

### **2.2.4 Описание выполнения и блок-схема второго алгоритма**

Алгоритм выполняется с помощью цикла, который идёт по массиву с первого элемента до элемента n (размер цикла). Переменная i – номер рассматриваемого элемента исходного массива. Переменная j – номер текущего элемента нового массива. Элементу массива с позицией j присваивается значение элемента массива с позицией i. Если рассматриваемый элемент i не равен искомому элементу удаления, то элемент j увеличивается на 1. Реализация данного описания выполнения алгоритма представлена в виде блок-схемы (рис.2).

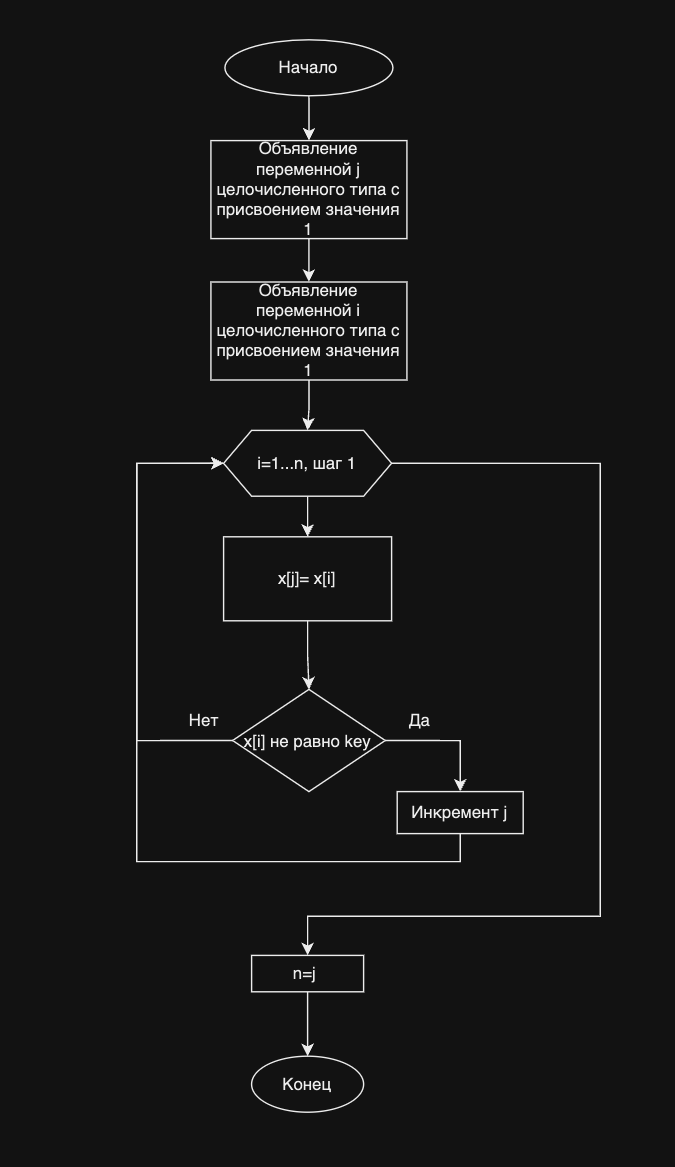


Рисунок 2 – Блок-схема второго алгоритма

### **2.2.5 Доказательство корректности циклов второго алгоритма**

Инвариант для цикла: значение переменной j всегда не больше i.

Докажем конечность цикла. Этот цикл будет выполнен n раз, так как он выполняется, пока перебираются числа от 1 до n. Цикл завершится после n повторений, так как количество повторений цикла зависит от значения переменной n. Следовательно, после n повторений цикл завершится.

Из доказательства можно сделать вывод, что цикл данного алгоритма корректен.

### **2.2.6 Определение вычислительной сложности второго алгоритма**

| № | Алгоритм, записанный на псевдокоде | Количество выполнений оператора в худшем случае | Количество выполнений оператора в лучшем случае |
| --- | --- | --- | --- |
| 1 | delOtherMetod(x,n,key){ |  |  |
| 2 | j←1 | 1 | 1 |
| 3 | for i←1 to n **do** | n+1 | n+1 |
| 4 | x[j]=x[i]; | n | n |
| 5 | if x[i]!=key then | n | n |
| 6 | j++ | n |  |
| 7 | endif |  |  |
| 8 | **od** |  |  |
| 9 | **n**←j | 1 | 1 |
| 10 | } |  |  |

Вычислительную сложность алгоритма в худшем случае можно определить функцией. Это линейная функция, а следовательно алгоритм обладает линейным порядком роста времени вычисления. Средний случай соответствует худшему случаю.

Вычислительную сложность алгоритма в лучшем случае можно определить функцией T(n)=3n+3. В данном случае все элементы нужно удалить.

## **2.3 Реализация алгоритмов на языке C++**

Реализуем первый алгоритм на языке С++ (рис.3). Для реализации понадобятся такие библиотеки как iostream. Iostream — это заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++. В программе алгоритма был реализован подсчет суммарного количества выполненных сравнений и перемещений элементов.

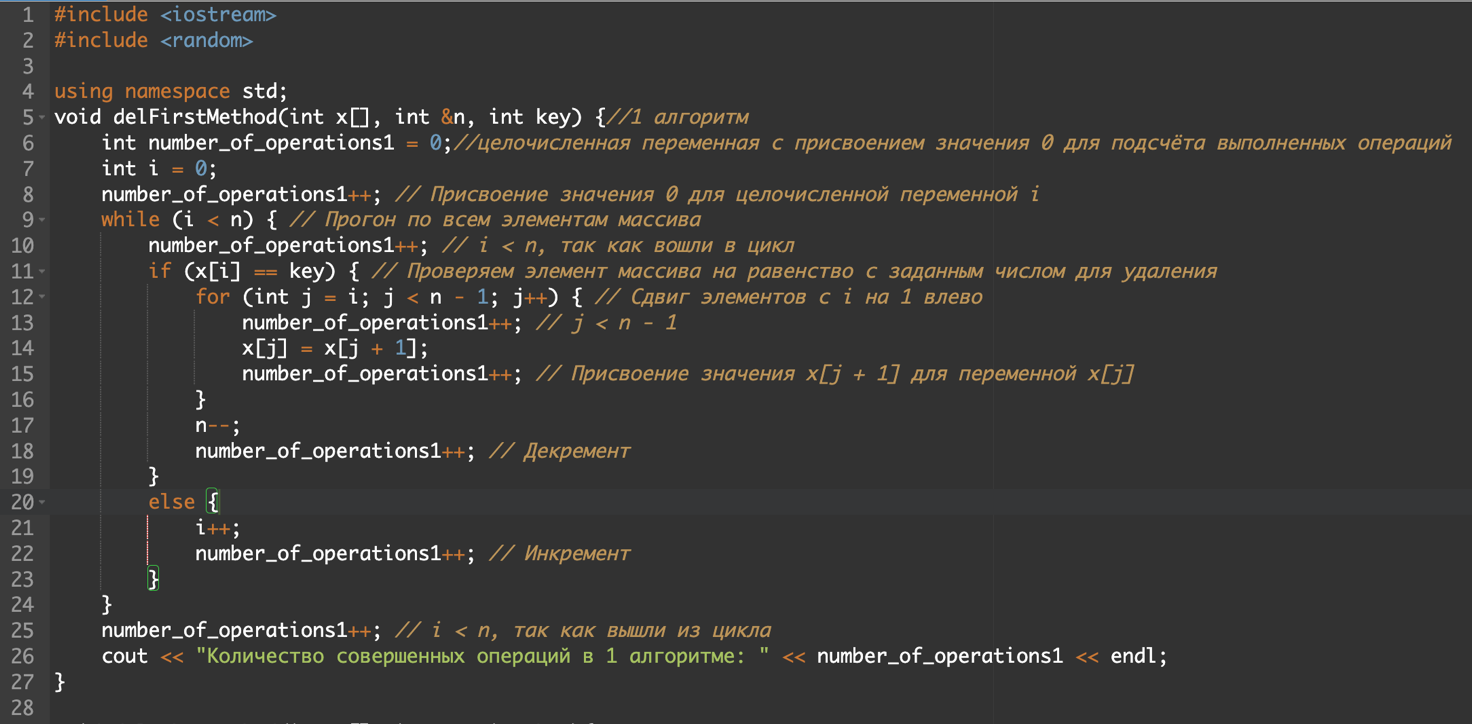


Рисунок 3 – Первый алгоритм удаления

Для реализации второго алгоритма (рис.4) потребуется библиотека iostream. Реализован подсчёт суммарного количества выполненных сравнений и перемещений элементов.

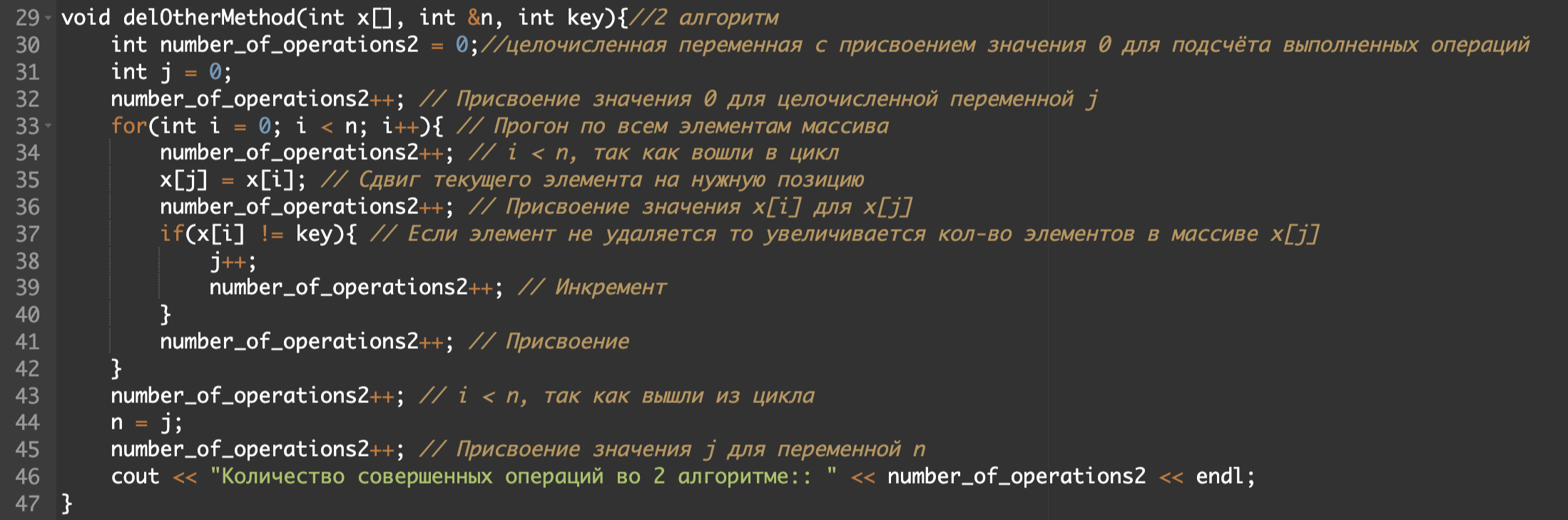


Рисунок 4 – Второй алгоритм удаления

Создадим функцию main для первого алгоритма (рис.5) и второго алгоритма (рис.6), которая обеспечивает работу программы. Чтобы реализовать эту функцию понадобятся библиотеки iostream и random. Random - позволяет генерировать случайные числа в диапазоне. Здесь задаются такие переменные как размер массива и удаляемое значение, которое нужны для выполнения алгоритмов. Для заполнения массива можно выбрать способ ввода: случайный или с клавиатуры. На выводе программы получаем конечный размер массива, конечный массив и сколько выполненных сравнений и перемещений элементов потребовалось для выполнения алгоритма.

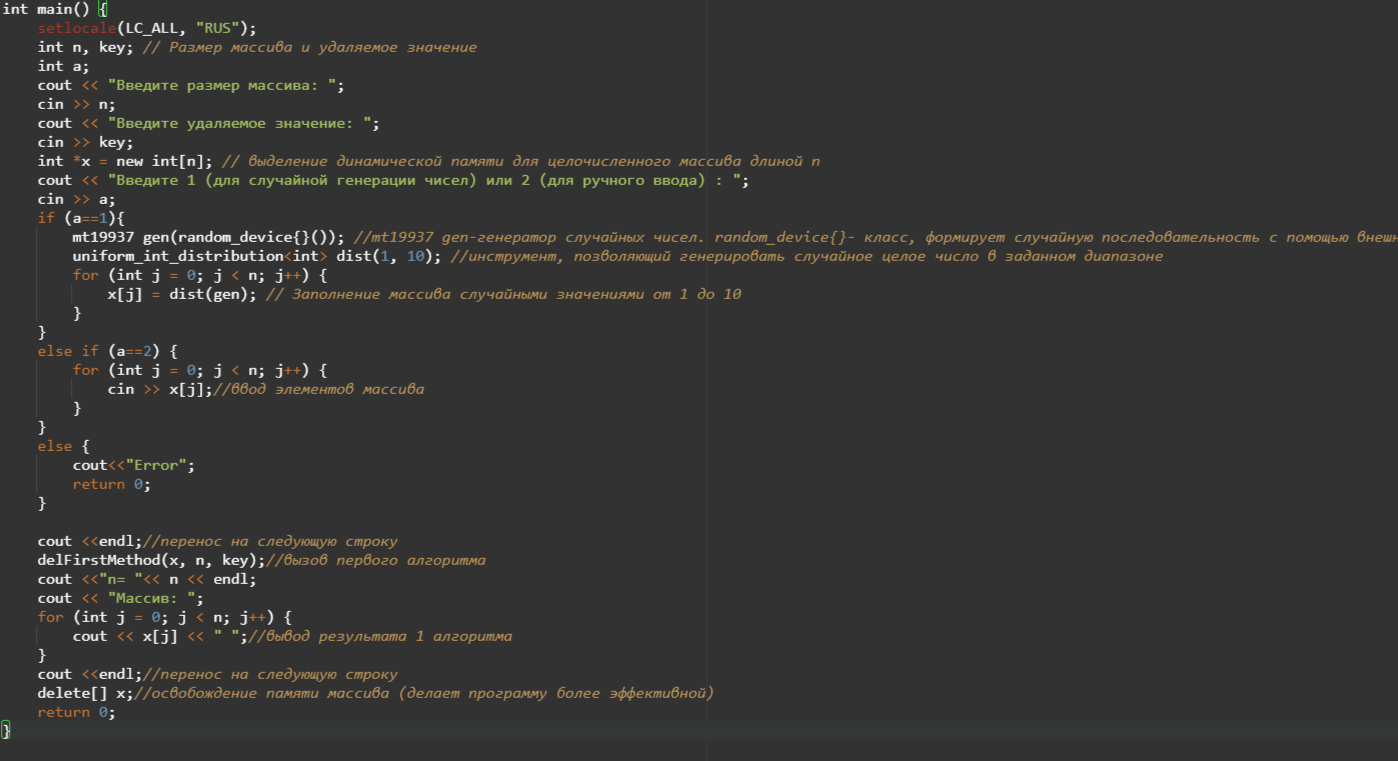


Рисунок 5 – Функция main для первого алгоритма



Рисунок 6 – Функция main для второго алгоритма

## **2.4 Тестирование**

### **2.4.1 Тестирование алгоритмов в ситуации случайного заполнения массива**

Проведём тест, выбрав случайную генерацию чисел для заполнения массива. Первый тест будет проведён с размером массива равным 10 для первого алгоритма (рис.7) и второго алгоритма (рис.8). Второй тест будет проведён с размером массива равным 100 для первого алгоритма (рис.9) и второго алгоритма (рис.10).

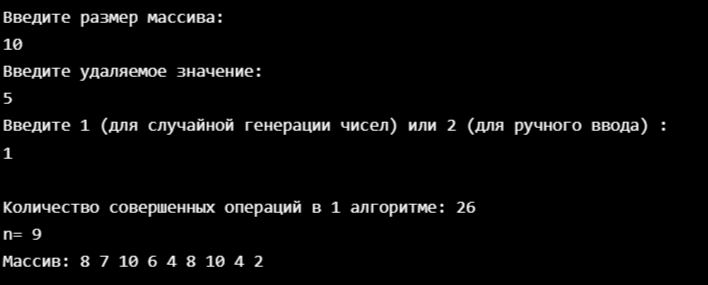


Рисунок 7 – Тестирование при 10 элементах для первого алгоритма

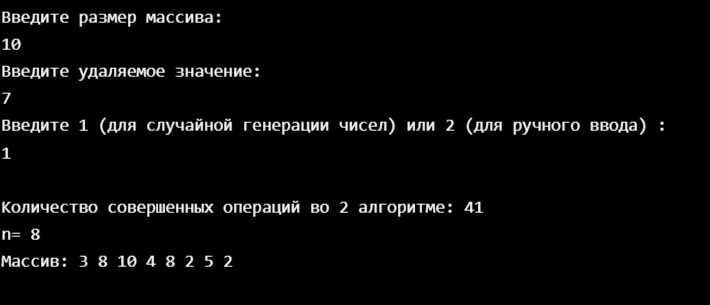


Рисунок 8 – Тестирование при 10 элементах для второго алгоритма

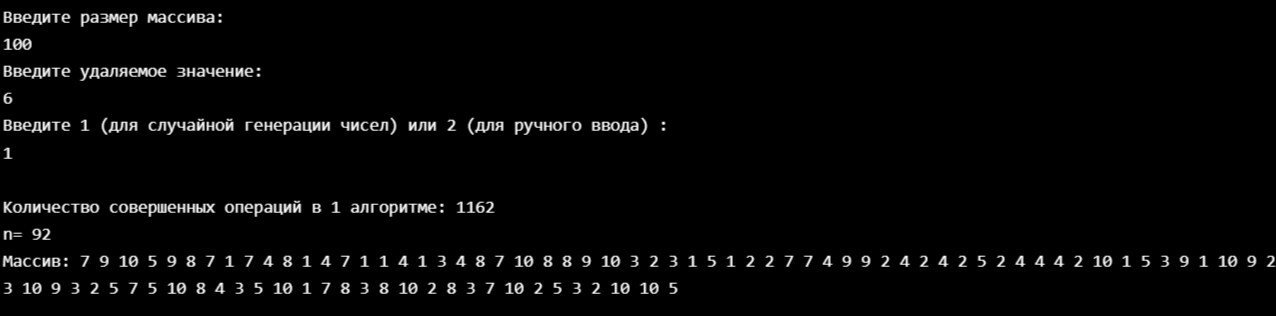
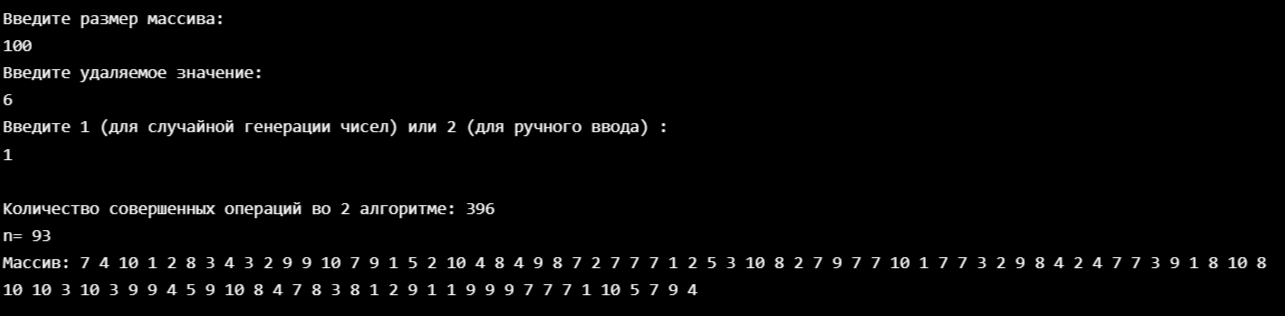


Рисунок 9 – Тестирование при 100 элементах для первого алгоритма

Рисунок 10 – Тестирование при 100 элементах для второго алгоритма

Анализируя результаты, можно заметить, что при 10 элементах более эффективным алгоритмом оказался первый, а при 100 элементах второй алгоритм.

### **2.4.2 Тестирование алгоритмов в ситуации все элементы массива должны быть удалены**

Проведём теоретические расчёты и тестирование при 10 элементах для первого алгоритма (рис.11) и второго алгоритма(рис.12) и при 100 элементах для первого алгоритма (рис. 13) и второго алгоритма(рис.14).

Все элементы массива первого алгоритма должны быть удалены, следовательно теоретическая сложность вычисления при 10 элементах равна T(10)=10\*10+10+2= 112.

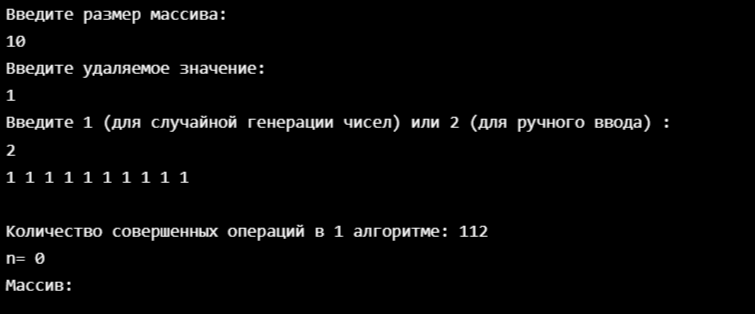


Рисунок 11 – Тестирование при 10 элементах для первого алгоритма

Все элементы массива второго алгоритма должны быть удалены, следовательно теоретическая сложность вычисления при 10 элементах равна T(10)=3\*10+3= 33.

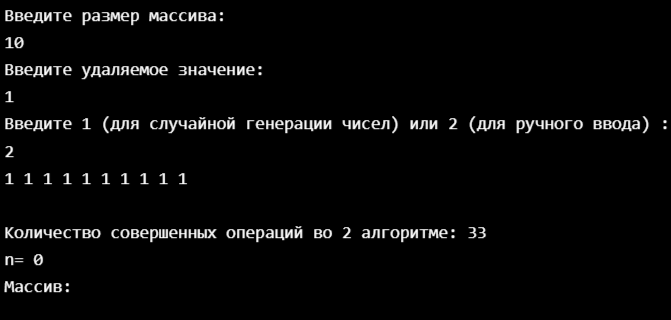


Рисунок 12 – Тестирование при 10 элементах для второго алгоритма

Все элементы массива первого алгоритма должны быть удалены, следовательно теоретическая сложность вычисления при 100 элементах равна T(10)= 100\*100+100+2= 10102.

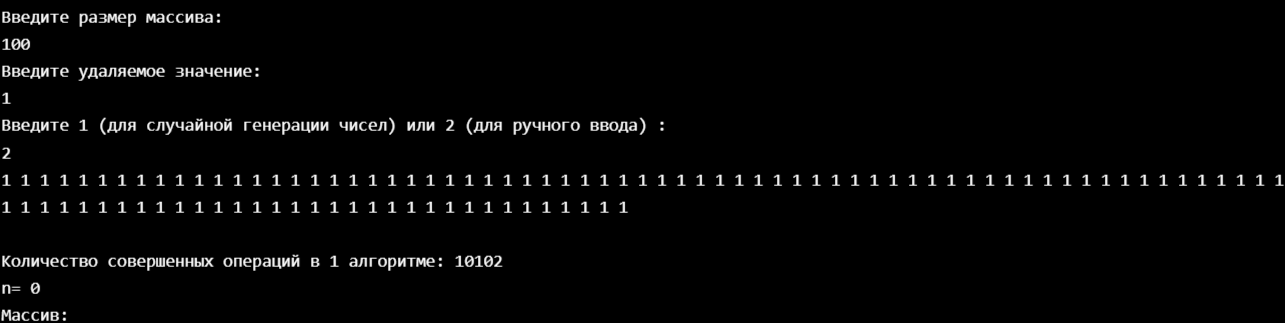


Рисунок 13 – Тестирование при 100 элементах для первого алгоритма

Все элементы массива второго алгоритма должны быть удалены, следовательно теоретическая сложность вычисления при 100 элементах равна T(10)= 3\*100+3 = 303.

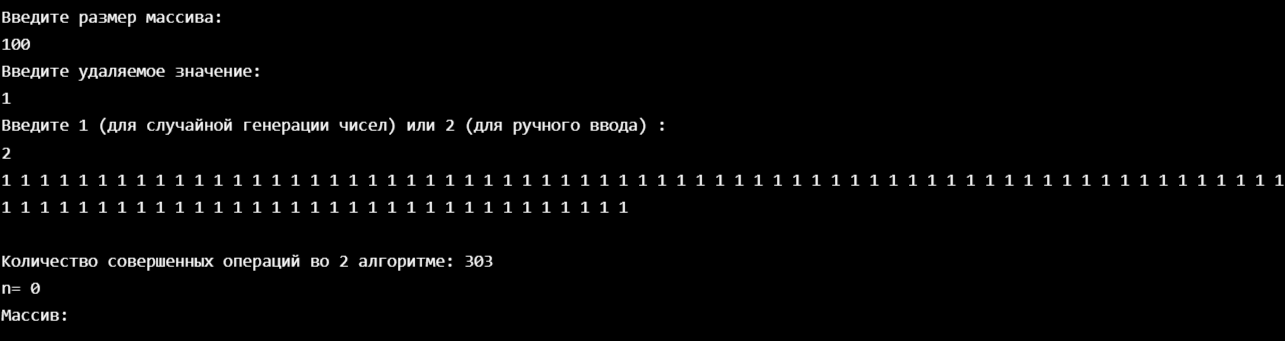


Рисунок 14 – Тестирование при 100 элементах для второго алгоритма

Анализируя результаты, можно заметить, что в обоих случаях более эффективным является второй алгоритм.

Результаты теоретического расчета сложности практически совпадают с экспериментально полученными, можно заявить, что расчеты выполнены верно.

### **2.4.3 Тестирование алгоритмов в ситуации ни один элемент не удаляется**

Проведём теоретические расчёты и тестирование при 10 элементах для первого алгоритма (рис. 15) и второго алгоритма(рис.16) и при 100 элементах для первого алгоритма (рис. 17) и второго алгоритма(рис.18).

Ничего не нужно удалять, следовательно теоретическая сложность вычисления первого алгоритма при 10 элементах равна T(10)=2\*10+2=22.

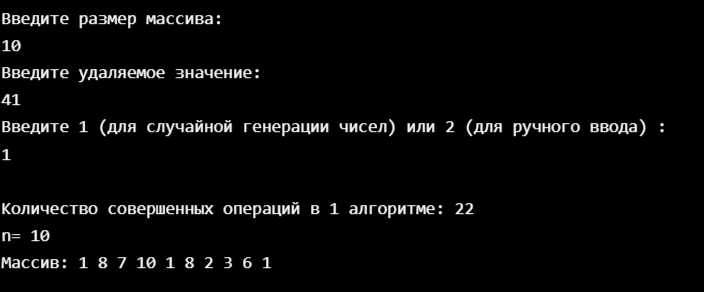


Рисунок 15 – Тестирование при 10 элементах для первого алгоритма

Ничего не нужно удалять, следовательно теоретическая сложность вычисления второго алгоритма при 10 элементах T(10)=4\*10+3=43.

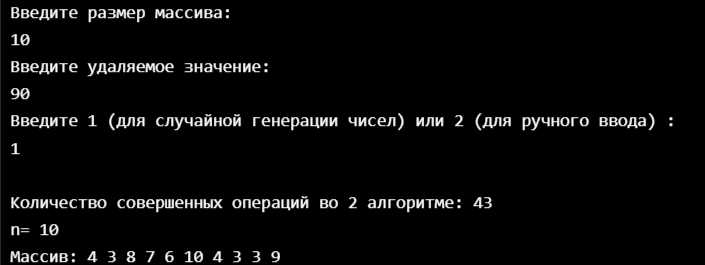


Рисунок 16 – Тестирование при 10 элементах для первого алгоритма

Ничего не нужно удалять, следовательно теоретическая сложность вычисления первого алгоритма при 100 элементах равна T(100)=2\*100+2=202.

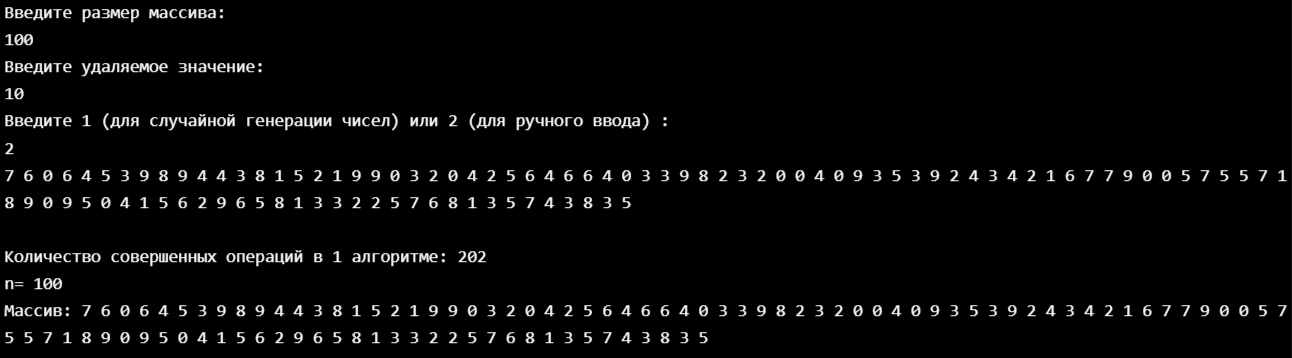


Рисунок 17 – Тестирование при 100 элементах для первого алгоритма

Ничего не нужно удалять, следовательно теоретическая сложность вычисления второго алгоритма при 100 элементах T(100)=4\*100+3=403.

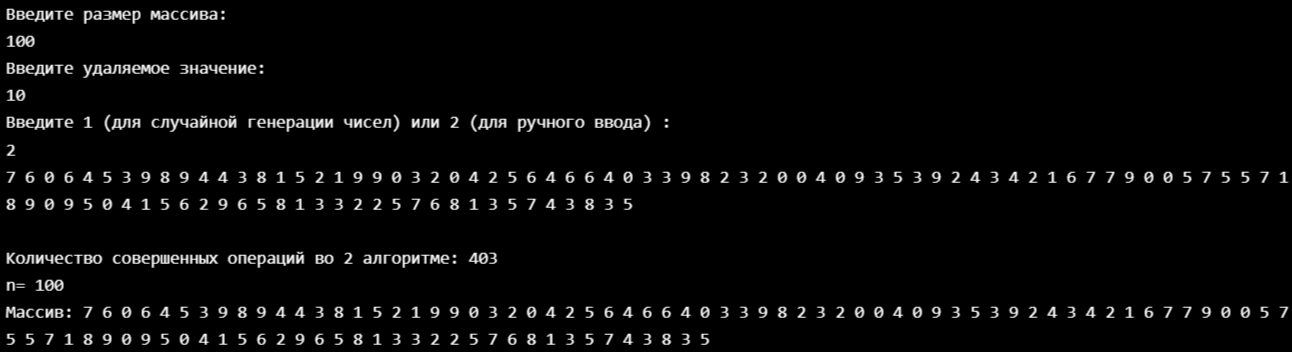


Рисунок 18 – Тестирование при 100 элементах для первого алгоритма

Анализируя результаты, можно заметить, что в обоих случаях более эффективным является первый алгоритм. Это доказывает, что первый алгоритм имеет квадратичную зависимость, что делает его более эффективным, чем второй, который имеет линейную зависимость, в ситуации, когда ни один элемент не удаляется.

Результаты теоретического расчета сложности практически совпадают с экспериментально полученными, можно заявить, что расчеты выполнены верно.

## **2.5 Вывод по заданию №1**

Основываясь на результатах тестирования и расчётов можно сделать вывод, что второй алгоритм более эффективный, чем первый алгоритм, так как при выполнении второго алгоритма в среднем и худшем случае требует намного меньше действий для выполнения, чем в первом алгоритме.

# 3 ЗАДАНИЕ №2

## **3.1 Описание математической модели**

**Формулировка задания (Вариант 10)**: выполнить транспонирование матрицы

## **3.2 Описание математической модели**

### **3.2.1 Описание выполнения и блок-схема первого алгоритма**

Создадим матрицу A размера n x m. Затем создаётся новая матрица В, которая будет являться транспонированной матрицей, размером m и n. Затем с помощью циклов создаются элементы матрицы. Затем с помощью циклов для каждого элемента в матрице A с индексами i и j присваиваем элементы матрице B с индексами j и i.

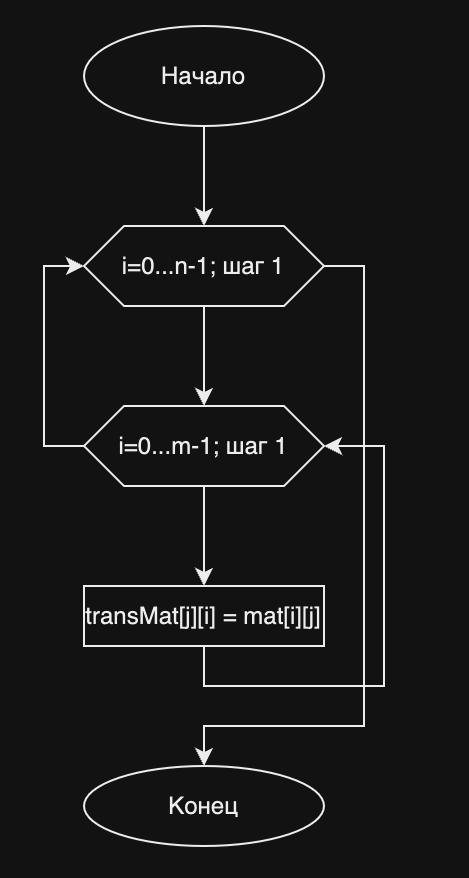


Рисунок 19 – Блок-схема алгоритма

## **3.3 Доказательство корректности циклов**

Определим инвариант для внешнего цикла: i всегда не больше n

Определим инвариант для внутреннего цикла: j всегда не больше m

Доказательство конечности цикла: при каждом повторении цикла переменная i принимает значения от 0 до n-1, это означает что цикл проходит по всем строкам матрицы mat. Также при каждом повторении цикла переменная j принимает значения от 0 до m-1, это означает что цикл проходит по всем строкам матрицы mat. При каждом прогоне цикла значения mat[i][j] помещаются в transMat[j][i].

Таким образом, после завершения выполнения цикла переменная transMat будет содержать транспонированную матрицу mat, где строки стали столбцами и наоборот. Следовательно, данный цикл конечен и корректно транспонирует матрицу mat и, следовательно, является корректным.

## **3.4 Определение вычислительной сложности алгоритма**

| Номер строки | Оператор | Количество повторений действия  в зависимости от объема  входных данных n |
| --- | --- | --- |
| 1 | for (int i = 0; i < n; i++) { | n+1 |
| 2 | for (int j = 0; j < m; j++) { | n(m+1) |
| 3 | transMat[j][i] = mat[i][j] | nm |
| 4 | } |  |
| 5 | } |  |

Общая вычислительная сложность алгоритма в худшем случае определяется функцией +1. Вычислительная сложность алгоритма в лучшем/среднем случае не будет отличаться от худшего, так как при любых значениях элементов матрицы можно совершить транспонирование. Из этого следует, что в любом случае будут выполняться все циклы алгоритма. Алгоритм имеет полиномиальный рост времени вычисления относительно размерности матрицы.

## **3.5 Реализация алгоритма на языке С++**

Реализуем алгоритм на языке C++ (рис.20,21). Для реализации понадобятся такие библиотеки как iostream и random. Iostream — это заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++. Random - позволяет генерировать случайные числа в диапазоне. Здесь задаются такие переменные как размер массива и удаляемое значение, которое нужны для выполнения алгоритмов. Для заполнения массива можно выбрать способ ввода: случайный или с клавиатуры. На выводе программы получаем конечный размер массива, конечный массив и сколько выполненных сравнений и перемещений элементов потребовалось для выполнения алгоритма.



Рисунок 20- Программа на C++



Рисунок 21- Программа на C++

## **3.6 Тестирование**

Проведем тестирование с различными величинами матриц с случайным заполнением элементов с помощью генератора чисел.

Для матрицы с количеством строк равным 10 и количеством столбцов равным 10, количество операций должно быть 2 \* 10 \* 10 + 2\* 10 + 1 = 221. Тестирование программы на С++ представлено на рисунке 22.

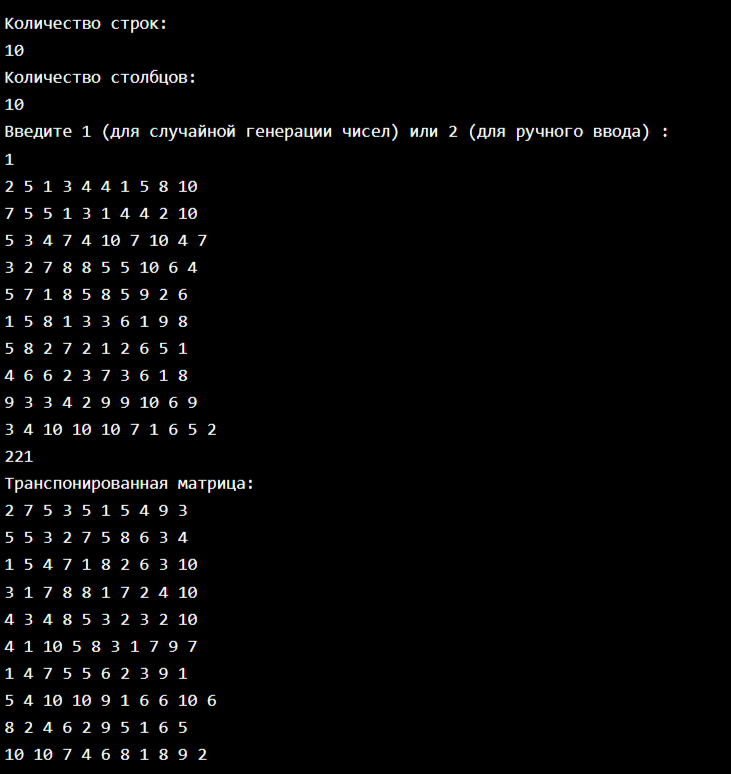


Рисунок 22 – Тест 1

Для матрицы с количеством строк равным 2 и количеством столбцов равным 7, количество операций должно быть 2 \* 2 \* 7 + 2\*2 +1 = 33. Тестирование программы на С++ представлено на рисунке 23.

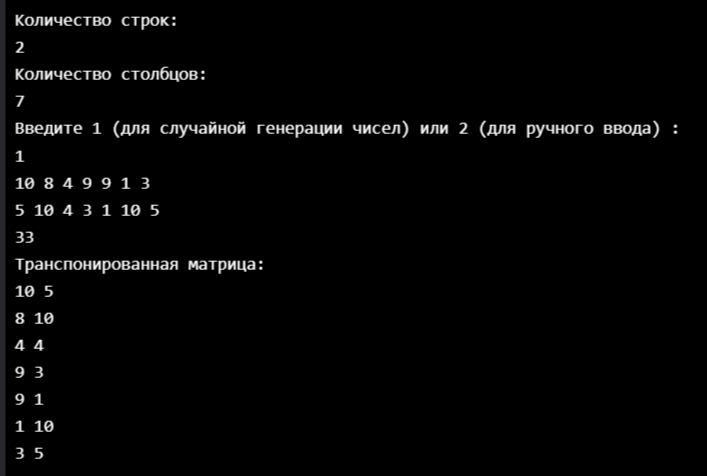


Рисунок 23 – Тест 2

Для матрицы с количеством строк равным 10 и количеством столбцов равным 5, количество операций должно быть 2 \* 10 \* 5 + 2\*10 +1 = 121.Тестирование программы на С++ представлено на рисунке 24.

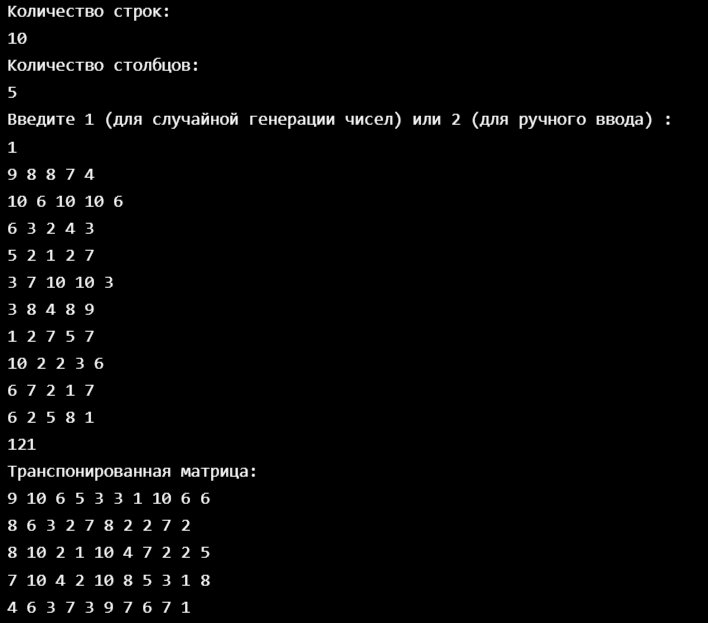


Рисунок 24 – Тест 3

## **3.7 Выводы по заданию 7**

В ходе работы был разработан алгоритм в соответствии с индивидуальным вариантом, оценена его сложность теоретическим и практическим методами. Основываясь на полученных данных, можно утверждать, что алгоритм транспонирования матрицы имеет линейную зависимость от количества элементов в матрице.

# 4 ВЫВОДЫ

В ходе работы отработаны навыки определению:

* сложности алгоритмов на теоретическом и практическом уровнях;
* эффективного алгоритма решения задачи из нескольких.

Разработан собственный алгоритм решения задачи и оценена его эффективность. Тестирование подтвердило правильность решения задачи алгоритмом, а также правильность теоретического расчета вычислительной сложности алгоритмов.

# 5 ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.

2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.

3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для про-граммистов и любопытствующих, 2017.

4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.

5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.

7. Скиена С. Алгоритмы. Руководство по разработке, 2011.

8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.

9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Инфор-матика и вычислительная биология, 2003.

По языку С++:

10. Страуструп Б. Программирование. Принципы и практика с использовани-ем C++. 2-е изд., 2016.

11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.

12. Прата С. Язык программирования С++. Лекции и упражнения. - 6-е изд., 2012.

13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002

14. Хортон А. Visual C++ 2010. Полный курс, 2011.

15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.