|  |
| --- |
|  |
| 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: Лелюхин Н.С. |
|  |  |
|  | Группа: ИКБО-74-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 2](#_2b1t3l6yprld)

[2 ЗАДАНИЕ 1 3](#_d69tp1vb55ju)

1.1 Первый алгоритм 3

[1.1.1 Описание математической модели 3](#_30j0zll)

[1.1.2 Блок-схема алгоритма, доказательство корректности циклов 4](#_k9o5vamw1764)

[1.1.3 Определение вычислительной сложности алгоритма 5](#_pgx5m88s4c9y)

[1.1.4 Реализация алгоритма на C++ 5](#_5xmtq9z8o9)

[1.1.5 Тестирование 7](#_w81ngib56404)

[1.2 Второй алгоритм 8](#_bk04rbrz217j)

[1.2.1 Описание математической модели 8](#_185teyizt2vd)

[1.2.2 Блок-схема алгоритма, доказательство корректности циклов 8](#_pfc2c335ot1y)

[1.2.3 Определение вычислительной сложности алгоритма 10](#_rtoml2jvht9i)

[1.2.4 Реализация алгоритма на языке С++ 10](#_4ku3v2ac7ral)

[1.2.5 Тестирование 11](#_st9d3aq5ai79)

[1.3 Вывод по первому заданию 13](#_20qy4pcciox)

[3 ЗАДАНИЕ 2 14](#_gq2irmnms3s9)

[3.1 Описание математической модели 14](#_8yezh43z9a2d)

[3.2 Блок-схема алгоритма, доказательство корректности циклов 14](#_xlj8yf6ekgwi)

[3.3 Определение вычислительной сложности алгоритма 16](#_81o7lgh1j5ou)

[3.4 Реализация алгоритма на языке C++ 17](#_fstx8331pk38)

[3.5 Тестирование 18](#_4z1kmq7qq4z3)

[3.6 Выводы по заданию 2 19](#_wlor992wub1q)

[4 ВЫВОДЫ 20](#_8fm6fdhpl081)

[5 ЛИТЕРАТУРА 21](#_f0tca8900ew1)

# 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

**Формулировка задания:** 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.

## 

## 1.1 Первый алгоритм

### 1.1.1 Описание математической модели

С помощью цикла идем по массиву с первого элемента до n-ного, где n – размер массива. Если текущий элемент равен заданному значению, то смещаем все следующие значения в массиве на 1 позицию влево, тем самым заменяя и удаляя требуемый элемент. Переменную n, отвечающую за размер массива, уменьшаем на 1.

### 1.1.2 Блок-схема алгоритма, доказательство корректности циклов

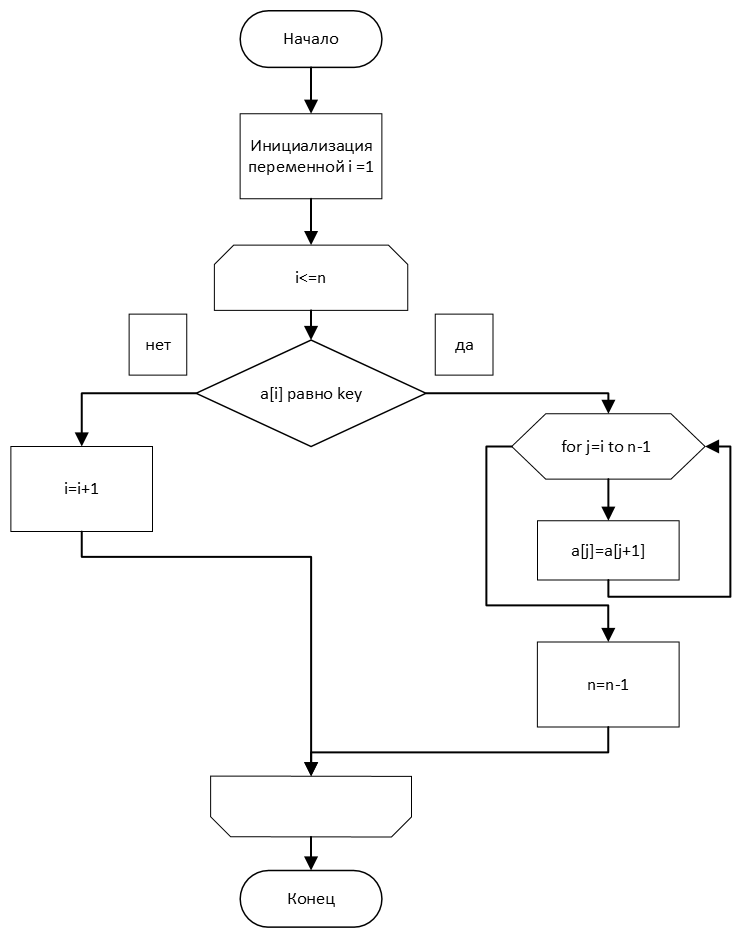


Рисунок 1 – Блок-схема для первого алгоритма

Определим инвариант для внешнего цикла: ни одно значение, индекс которого меньше i, не равно удаляемому значению key

Определим инвариант для внутреннего цикла: j всегда не больше n

Доказательство конечности цикла: при каждой итерации по переменной i область неопределённости сужается на 1 элемент. До начала цикла не просмотрено n элементов, после первой итерации n-1, после второй n-2 и так далее. После n-ной итерации будет не просмотрено n-n=0 элементов, следовательно цикл завершится.

Таким образом, все циклы алгоритма корректны, а значит и сам алгоритм, корректен.

### 1.1.3 Определение вычислительной сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Номер строки | Алгоритм, записанный на псевдокоде | Количество повторений действия в  зависимости от объема входных  данных n |
| 1 | delFirstMetod(x,n,key){ |  |
| 2 | i←1 | 1 |
| 3 | while (i<=n) **do** | n+1 |
| 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 |  |
| 11 | endif |  |
| 12 | **od** |  |
| 13 | } |  |

Количество повторений действия в строке 6 представляет собой арифметическую прогрессию. Найдем ее сумму  *.*

Тогда общая вычислительная сложность алгоритма в худшем случае определяется функцией**.** То есть алгоритм имеет квадратичный порядок роста времени вычисления.

В лучшем случае, когда ни один элемент удалять не нужно, сложность определяется функцией**.**

### 1.1.4 Реализация алгоритма на C++

Напишем программу на языке C++(рис. 2,3). Используем библиотеки random и iostream.

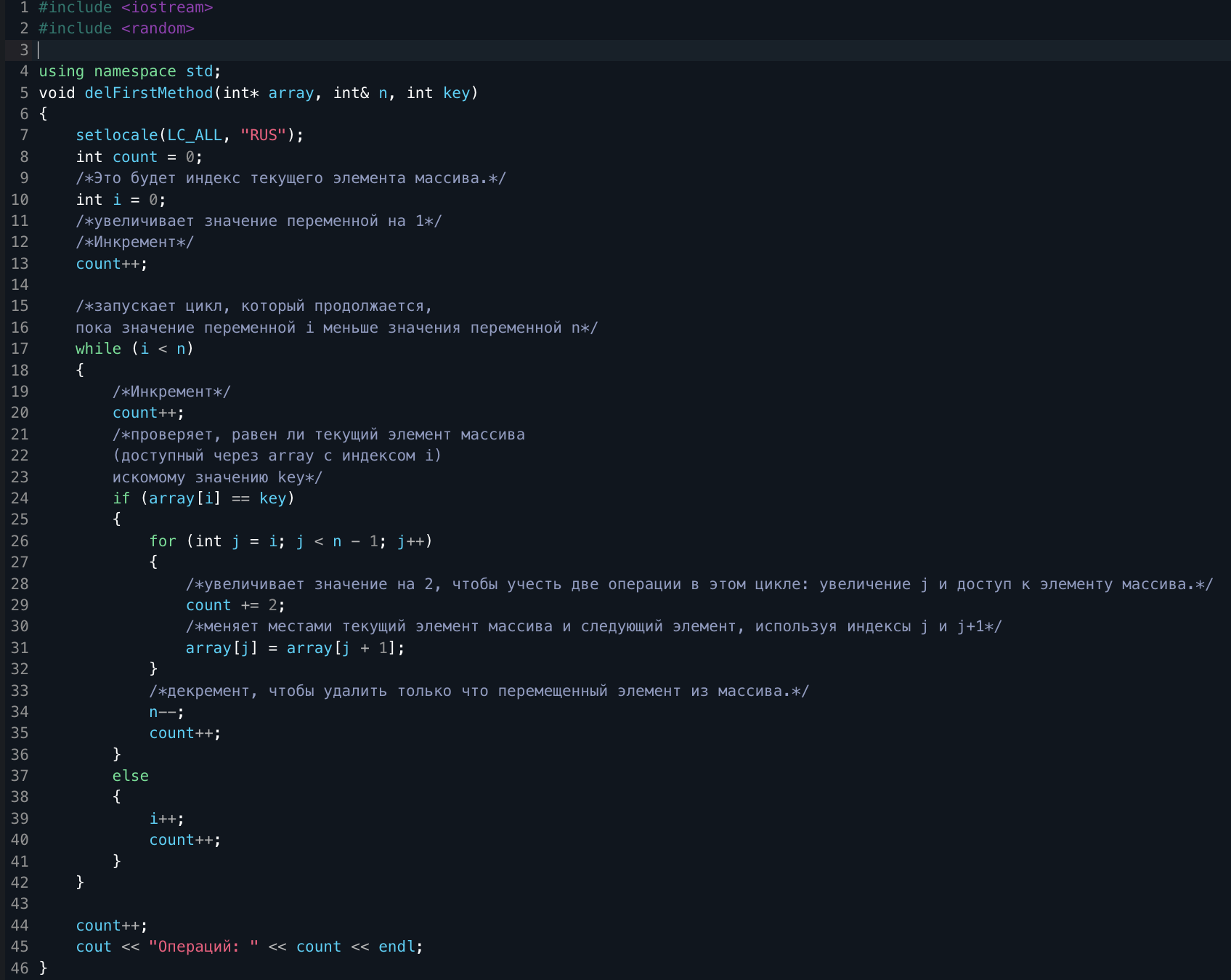


Рисунок 2 – Первый алгоритм

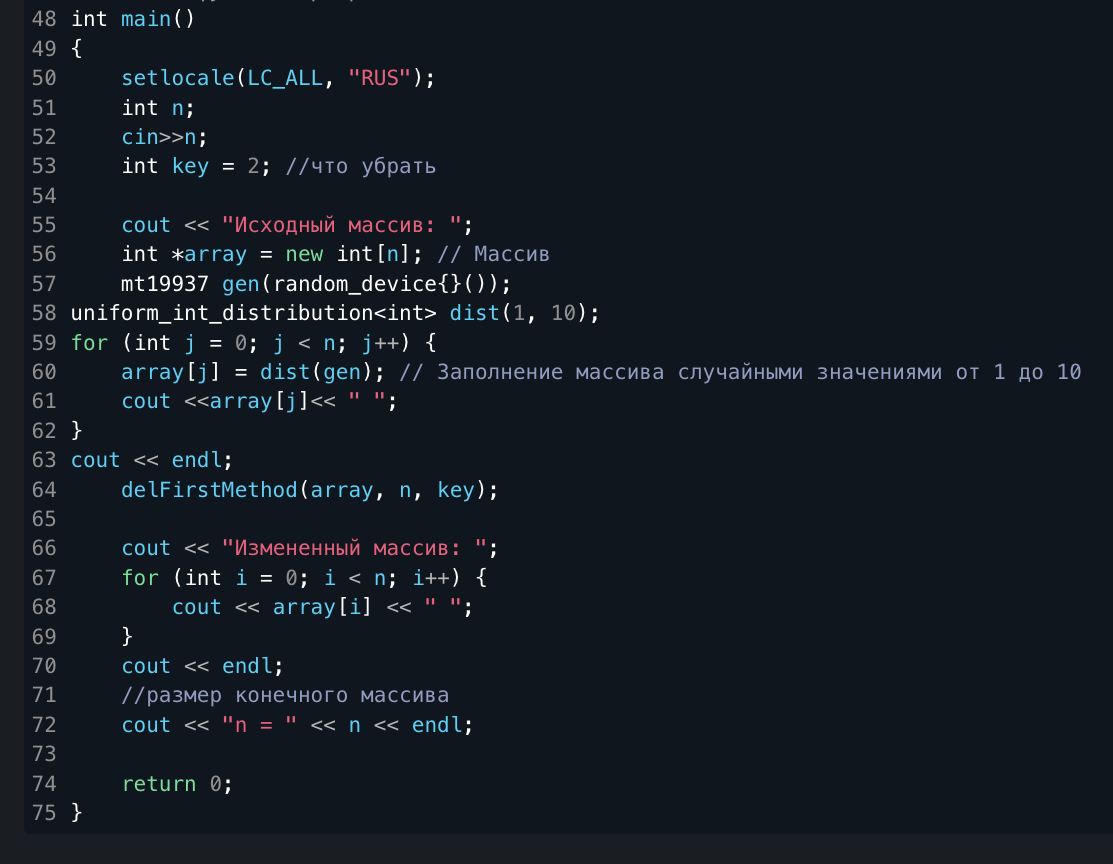


Рисунок 3 – Первый алгоритм

### 1.1.5 Тестирование

Проведем тестирование данной программы и продемонстрируем результаты(рис.4-8)

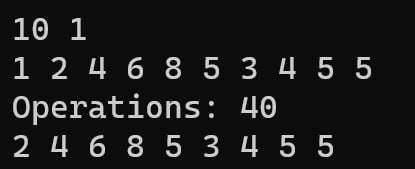


Рисунок 4 – Тестирование при 10 элементах. Случайная ситуация.

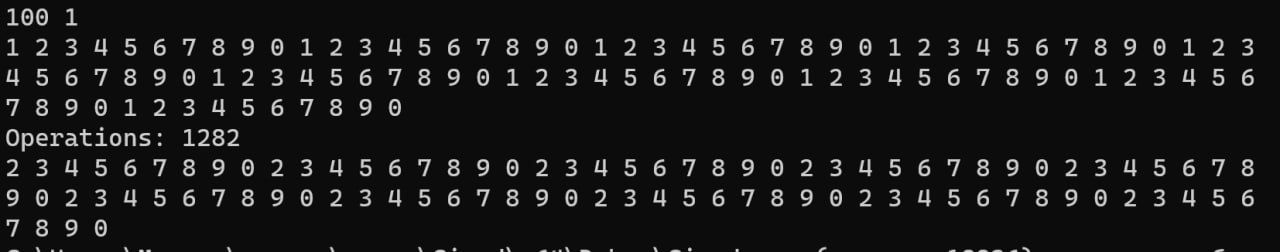


Рисунок 4 – Тестирование при 100 элементах. Случайная ситуация.

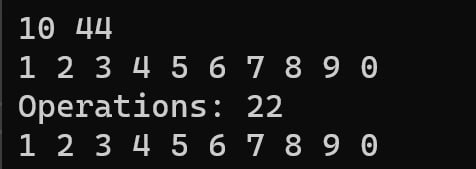


Рисунок 5 – Тестирование при 10 элементах. Ни одного элемента не нужно удалять.

Теоретическая сложность вычисления при 10 элементах, когда ничего не нужно удалять, T(10)=2\*10+2=22

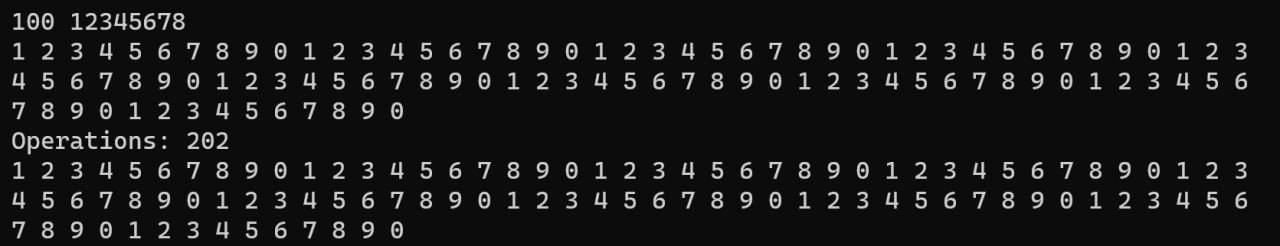


Рисунок 6 – Тестирование при 100 элементах. Ни одного элемента не нужно удалять.

Теоретическая сложность вычисления при 100 элементах, когда ничего не нужно удалять, T(100)=2\*100+2=202

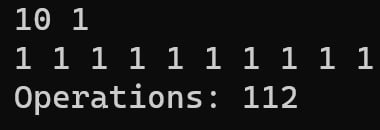


Рисунок 7 – Тестирование при 10 элементах. Все нужно удалить

Теоретическая сложность вычисления при 10 элементах, когда все нужно удалять, T(10)=10\*10+10+2= 112

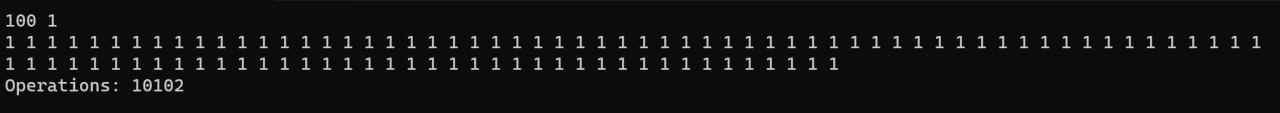


Рисунок 8 – Тестирование при 100 элементах. Все нужно удалить

Теоретическая сложность вычисления при 100 элементах, когда все нужно удалять, T(10)= 100\*100+100+2= 10102

Как мы видим, реальные результаты близки к теоретическим, квадратичная зависимость прослеживается, значит теоретический расчет верный.

## 1.2 Второй алгоритм

### 1.2.1 Описание математической модели

С помощью цикла идем по массиву с первого элемента до n-ного, где n – размер массива. В переменной i хранится номер рассматриваемого элемента исходного массива, в переменной j хранится номер размещаемого в данный момент элемента в конечном массиве. В j тый элемент постоянно помещаем i тый, но увеличиваем j на 1 ( то есть размещаем теперь в следующий элемент конечного массива) только если текущий не равен искомому значению.

### 1.2.2 Блок-схема алгоритма, доказательство корректности циклов

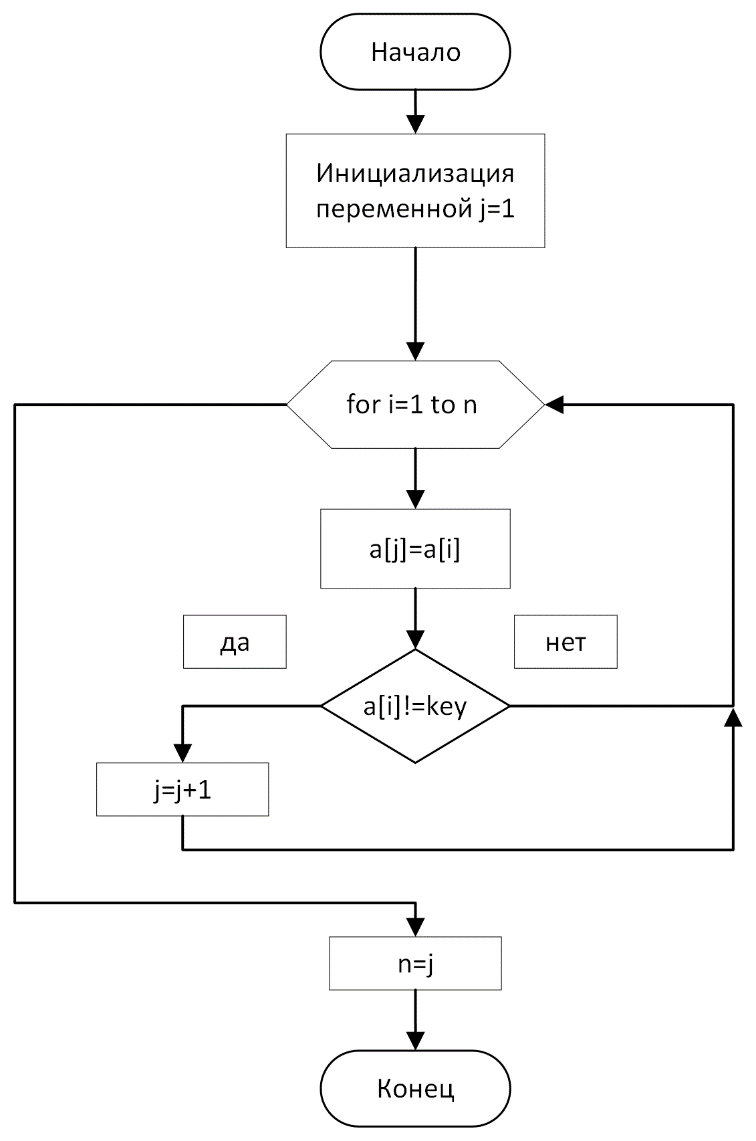


Рисунок 9 – Блок-схема второго алгоритма

Определим инвариант для цикла: j всегда не больше i и элементы с номерами меньшими j не содержат значения key.

Доказательство конечности цикла: при каждой итерации область неопределённости сужается на 1 элемент. До начала цикла не просмотрено n элементов, после первой итерации n-1, после второй n-2 и так далее. После n-ной итерации будет не просмотрено n-n=0 элементов, следовательно цикл завершится.

Таким образом, цикл алгоритма корректен, а значит и сам алгоритм, корректен.

### 1.2.3 Определение вычислительной сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Номер строки | Алгоритм, записанный на псевдокоде | Количество повторений действия в  зависимости от объема входных  данных n |
| 1 | delOtherMetod(x,n,key){ |  |
| 2 | j←1 | 1 |
| 3 | for i←1 to n **do** | n+1 |
| 4 | x[j]=x[i]; | n |
| 5 | if x[i]!=key then | n |
| 6 | j++ | n |
| 7 | endif |  |
| 8 | **od** |  |
| 9 | **n**←j | 1 |
| 10 | } |  |

Общая вычислительная сложность алгоритма в худшем случае определяется функцией. То есть алгоритм имеет линейный порядок роста времени вычисления.

В лучшем случае, когда все не нужно удалять, сложность определяется функцией

### 1.2.4 Реализация алгоритма на языке С++

Напишем программу на языке C++(рис. 10,11). Используем библиотеки random и iostream.

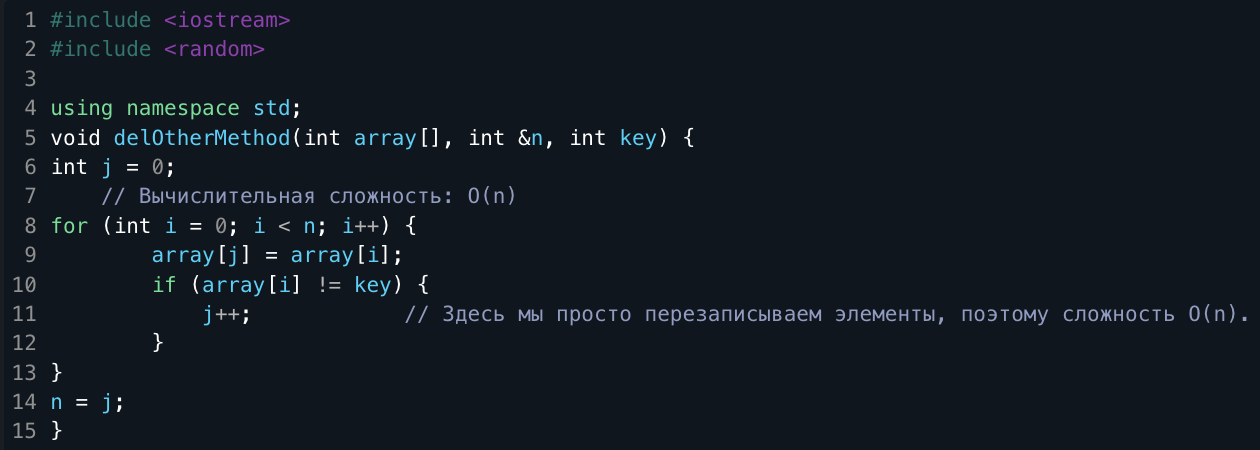
****

Рисунок 10 – Второй алгоритм

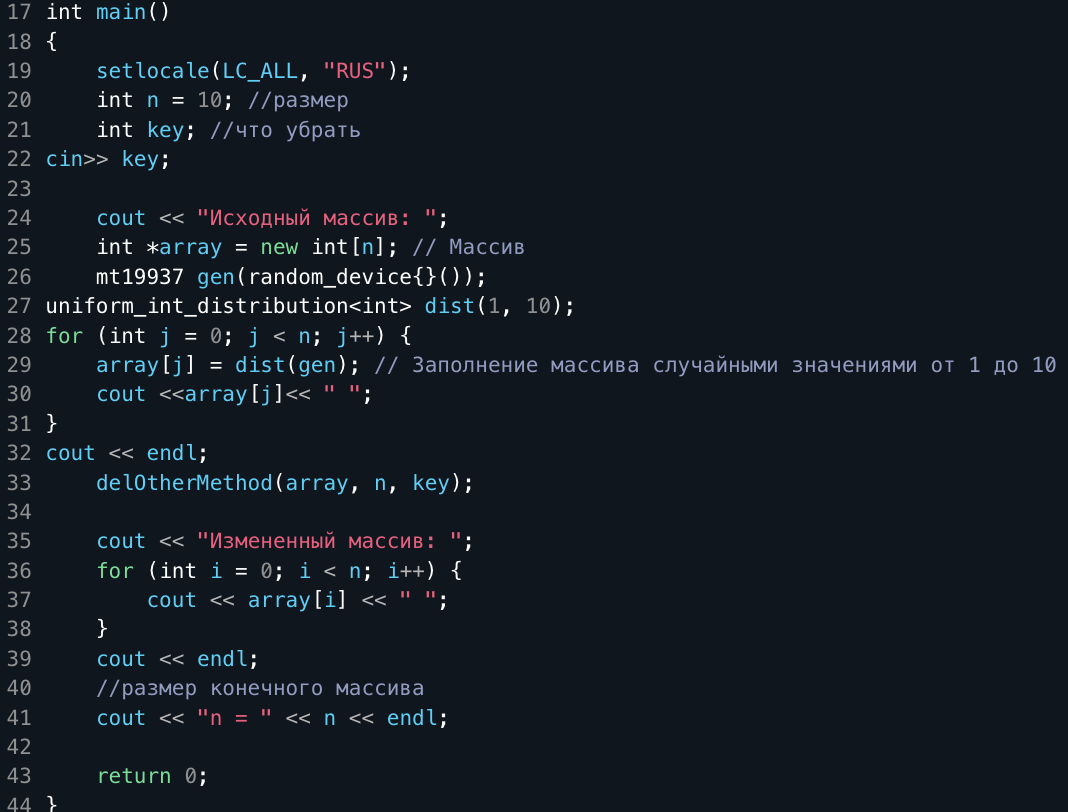


Рисунок 11 – Второй алгоритм

### 1.2.5 Тестирование

Проведем тестирование данной программы и продемонстрируем результаты(рис.12-17)

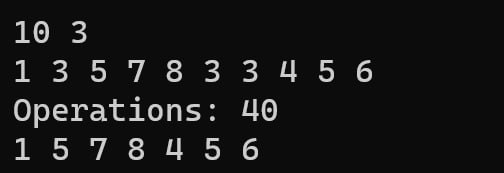


Рисунок 12 – Тестирование при 10 элементах. Случайная ситуация.

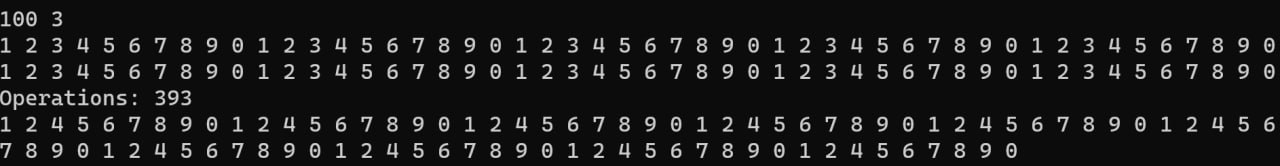


Рисунок 13 – Тестирование при 100 элементах. Случайная ситуация.

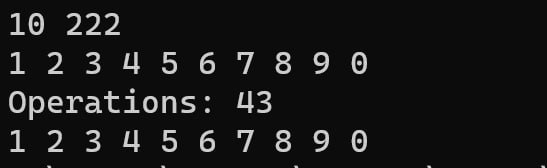


Рисунок 14 – Тестирование при 10 элементах. Ни одного элемента не нужно удалять.

Теоретическая сложность вычисления при 10 элементах, когда ничего не нужно удалять, T(10)=4\*10+3=43

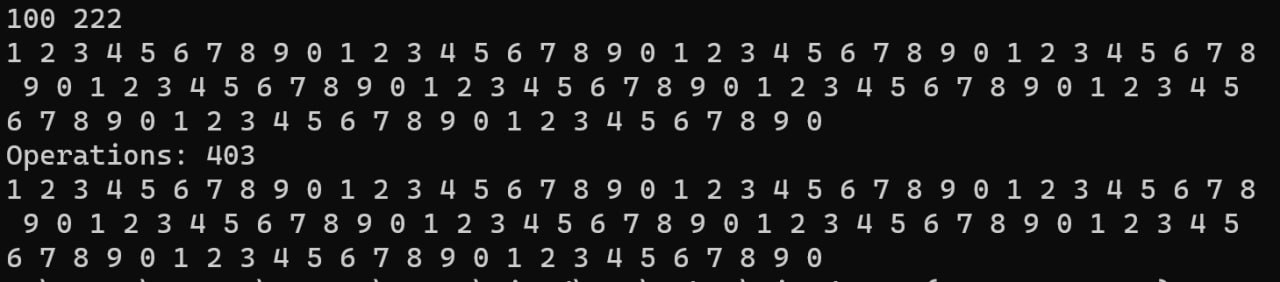


Рисунок 15 – Тестирование при 100 элементах. Ни одного элемента не нужно удалять.

Теоретическая сложность вычисления при 100 элементах, когда ничего не нужно удалять, T(100)=4\*100+3=403

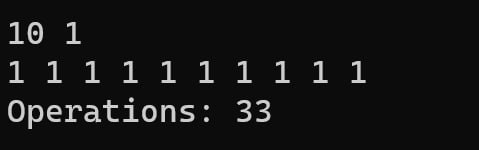


Рисунок 16 – Тестирование при 10 элементах. Все нужно удалить

Теоретическая сложность вычисления при 10 элементах, когда все нужно удалять, T(10)=3\*10+3= 33

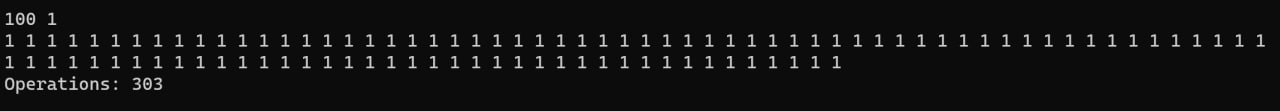


Рисунок 17 – Тестирование при 100 элементах. Все нужно удалить

Теоретическая сложность вычисления при 100 элементах, когда все нужно удалять, T(10)= 3\*100+3 = 303

Как мы видим, реальные результаты близки к теоретическим, линейная зависимость прослеживается, значит теоретический расчет верный.

## 1.3 Вывод по первому заданию

Основываясь на полученных результатах, можно сделать вывод, что второй алгоритм (с линейной зависимостью сложности, формула худшего случая ) более эффективен, чем первый (с квадратичной зависимостью сложности, формула худшего случая .) и в среднем и худшем случае требует намного меньше действий для выполнения.

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

# 

# 3 ЗАДАНИЕ 2

## 3.1 Описание математической модели

Для решения данной задачи можно использовать простую математическую модель в виде алгоритма:

1. Создать два указателя - left и right.

2. Установить left на самое начало массива (индекс 0) и right на самый конец массива (индекс n-1).

3. Пока left меньше right, выполнять следующие шаги:

- Если значение элемента, на который указывает left, равно 0, увеличить left на 1.

- Если значение элемента, на который указывает right, не равно 0, уменьшить right на 1.

- Если значение элемента, на который указывают left и right, соответствует условиям выше, поменять их местами и увеличить left на 1, а уменьшить right на 1.

4. Теперь все элементы равные 0 будут располагаться в начале массива, а все остальные - в оставшейся части массива.

Это простой алгоритм, который позволит решить задачу перестановки элементов массива так, чтобы сначала шли все элементы равные 0, а затем все остальные.

## 3.2 Блок-схема алгоритма, доказательство корректности циклов

Построим блок-схему данного алгоритма(рис.18).

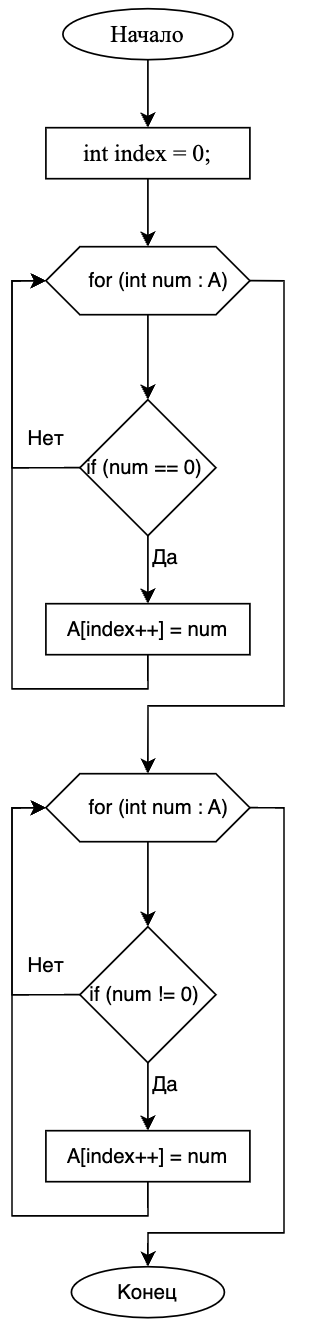


Рисунок 18 - Блок-схема второго алгоритма

Доказательство корректности алгоритма.

Первый цикл:

* Цикл проходит по всем элементам вектора A.
* Внутри цикла есть условие проверки, если элемент равен 0, то инкрементируется индекс index.
* Так как вектор A имеет конечное количество элементов, то цикл будет иметь конечное количество итераций.

Второй цикл:

* После первого цикла и инкрементирования index, он будет указывать на первый элемент, который не равен 0.
* Таким образом, второй цикл также проходит по всем элементам вектора A, но уже начиная с позиции, где закончился первый цикл.
* И также имеет конечное количество итераций, так как индекс вектора A ограничен его размером.

Таким образом, оба цикла в данном коде имеют конечное количество итераций, что доказывает конечность циклов в этом коде. Следовательно все циклы данного алгоритма корректны.

## 3.3 Определение вычислительной сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Номер строки | Оператор | Количество повторений действия в  зависимости от объема входных  данных n |
| 1 | void moveZeroes(vector<int>& A, long& o) { |  |
| 2 | int index = 0; | 1 |
| 3 | for (int num : A) { | n+1 |
| 4 | if (num == 0) { | n |
| 5 | A[index++] = num; | n |
| 6 | } |  |
| 7 | } |  |
| 8 | for (int num : A) { | n+1 |
| 9 | if (num != 0) { | n |
| 10 | A[index++] = num; | n |
| 11 | } |  |
| 12 | } |  |
| 13 | } |  |

Вычислительная сложность алгоритма в лучшем случае (когда не встречается ни одного нуля) будет равна T(n)=5n+3. Вычислительная сложность алгоритма в худшем случае (когда встречаются нули) будет равна T(n)=6n+3.

## 3.4 Реализация алгоритма на языке C++

Реализуем данный алгоритм на языке программирования C++(рис.19,20). Воспользуемся библиотеками iostream, random,algorithm и vector для выполнения данной программы.

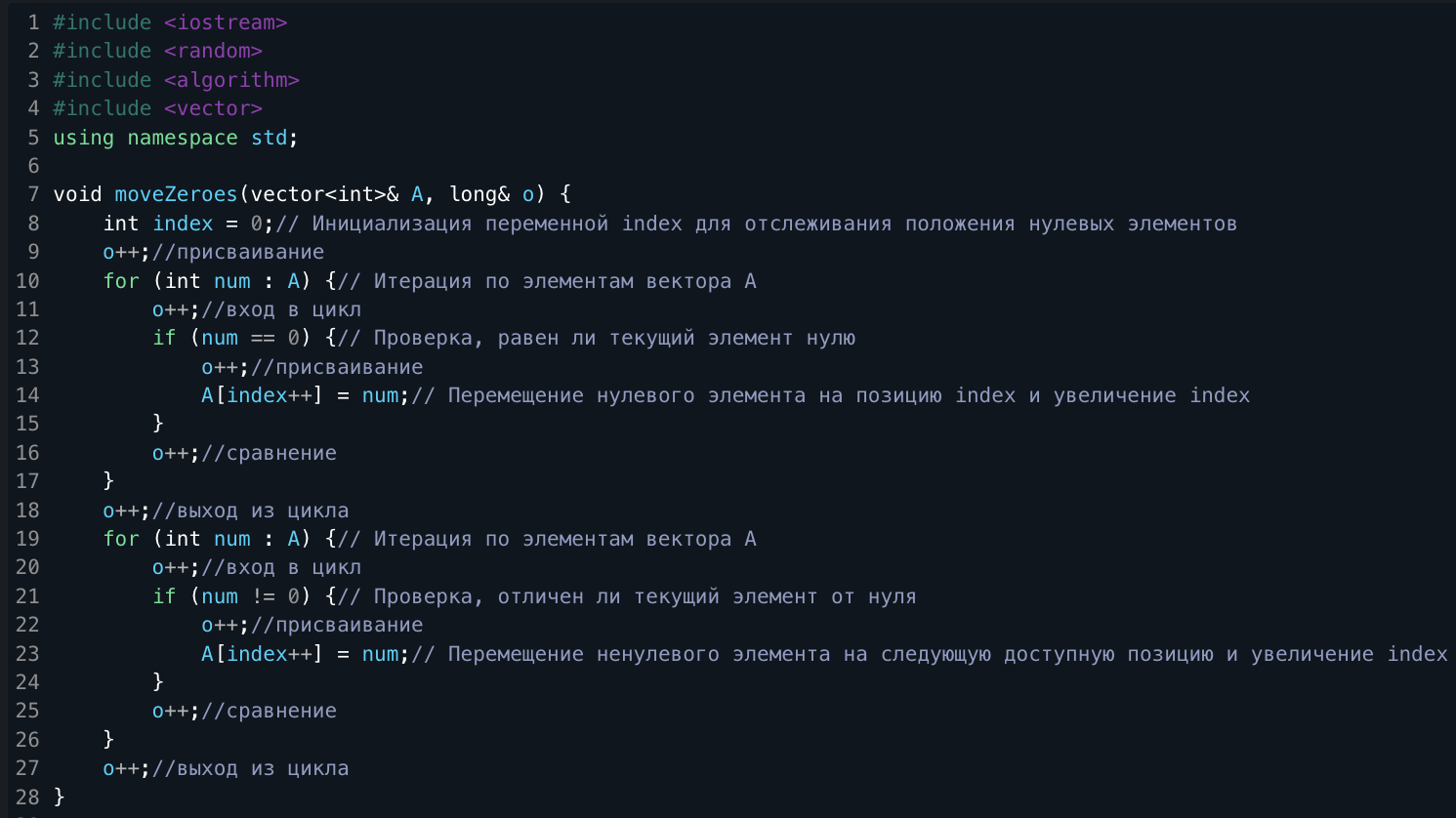


Рисунок 19 – Программа на С++

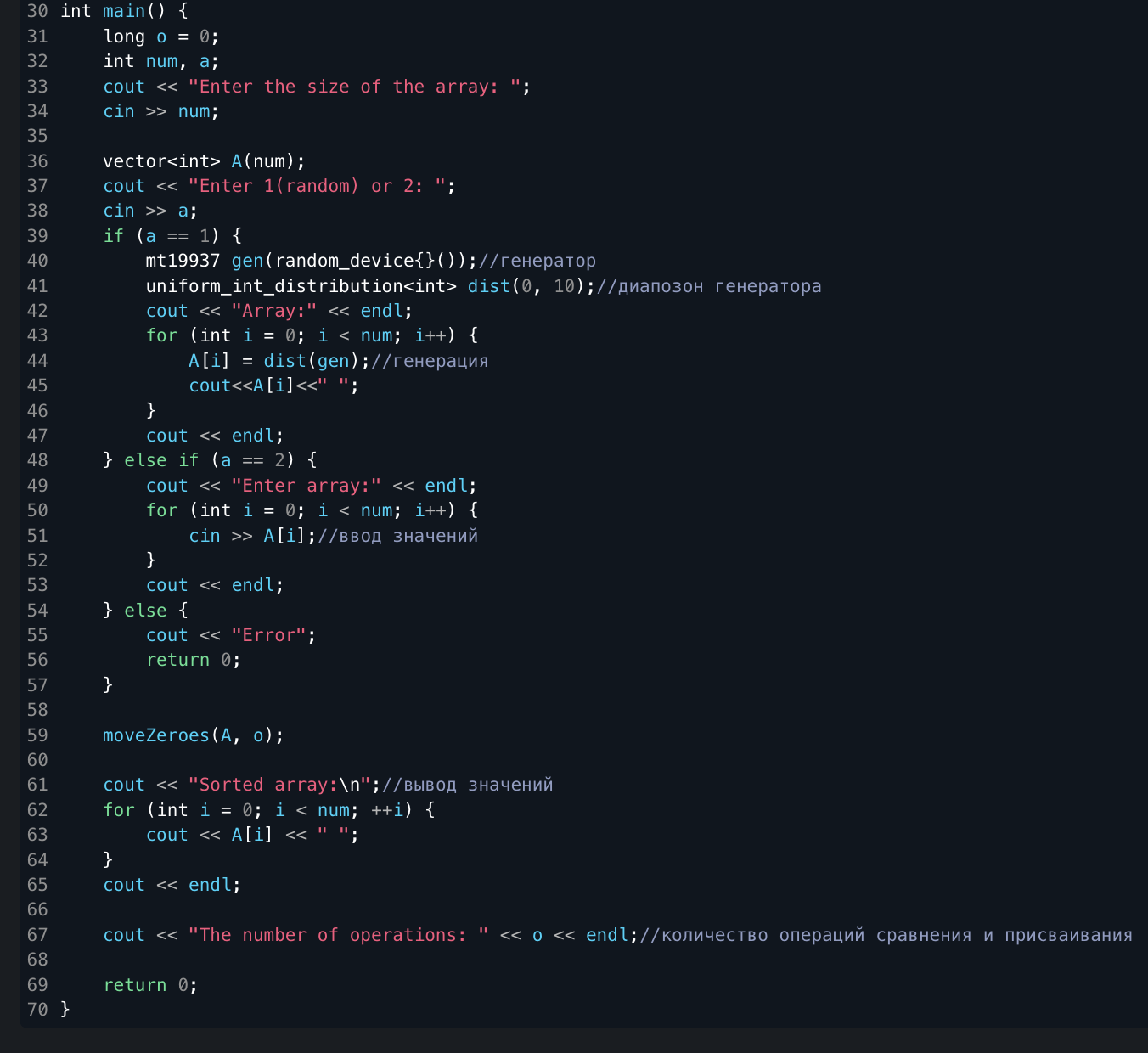


Рисунок 19 – Программа на C++

## 3.5 Тестирование

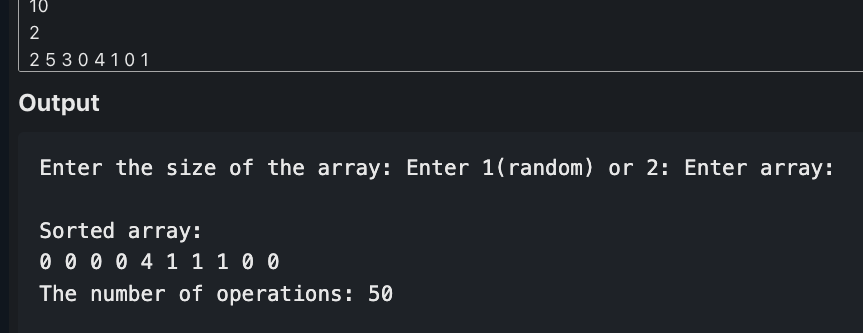


Рисунок 20 – Тест при размере 10

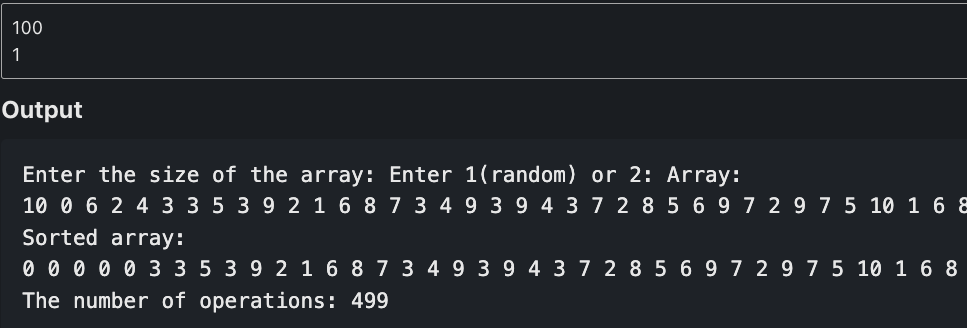


Рисунок 21 – Тест при размере 100

## 3.6 Выводы по заданию 2

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 преобразования массива в данном варианте зависит от входных данных в массив.

### 

### 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.