

МИНОБРНАУКИ РОССИИ

*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 Хитров Н.С.

группа

## Москва 2022

ИКБО-02-21

# Цель работы:

Приобретение практических навыков по определению:

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

Разработка собственного алгоритма решения задачи и оценка его эффективности.

# Ход работы:

## Задание 1

**Алгоритм 1**

**Постановка задачи:**

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.

**Модель решения поставленной задачи:**

* + 1. Алгоритм перебирает все элементы массива и проверяет их на соответствие ключевому значению. Если элемент равен ключевому значению, то при помощи цикла for все элементы от i-того до n-1 присваивают себе значения справа стоящего от них элемента. С каждым подобным n будет уменьшаться на единицу. Таким образом, элементы от 0 до n получившегося в результате работы функции массива будут не равными ключевому значению.

Блок-схема алгоритма:

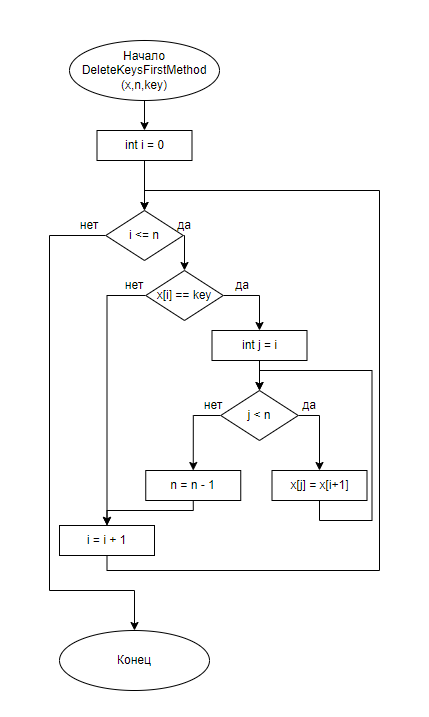


Рис. 1 – Блок-схема Алгоритма 1

* + 1. Доказательство корректности цикла

Инвариант внешнего цикла: n хранит в себе актуальное значение длины массива. До выполнения цикла оно не меняет своего значения. После каждой итерации цикла в случае равенства элемента массива значение длины уменьшается на 1 (так как из массива удаляется 1 элемент). По завершению цикла его длина будет определяться количеством элементов, не равных ключевому значению key.

* + 1. Вычислительная сложность алгоритма:

Общая вычислительная сложность алгоритма в худшем случае определяется функцией 𝑇(n)=n2+2n+3. То есть алгоритм имеет квадратичный порядок роста времени вычисления.

В лучшем случае, когда все нужно удалять, сложность определяется функцией 𝑇(n)=3𝑛+2.

Табл. 1 – Алгоритм 1, лучший случай

|  |  |
| --- | --- |
| Оператор | Кол-во выполнений оператора |
| 1 delFirstMetod(x,n,key){ |  |
| 2 i←1 | 1 |
| 3 while (i<=n) do | n+1 |
| 4 if x[i]=key then | n |
| 5 //удаление |  |
| 6 for j←i to n-1 do | 0 |
| 7 x[j] ←x[j+1] | 0 |
| 8 od |  |
| 9 n←n-1 | 0 |
| 10 else |  |
| 11 i←i+1 | n |
| 12 endif |  |
| 13 od |  |
| 14} |  |

Табл. 2 – Алгоритм 1, худший случай

|  |  |
| --- | --- |
| Оператор | Кол-во выполнений оператора |
| 1 delFirstMetod(x,n,key){ |  |
| 2 i←1 | 1 |
| 3 while (i<=n) do | n+1 |
| 4 if x[i]=key then | n |
| 5 //удаление |  |
| 6 for j←i to n-1 do | n\*(n - 1)/2 + 1 |
| 7 x[j] ←x[j+1] | n\*(n - 1)/2 |
| 8 od |  |
| 9 n←n-1 | n |
| 10 else |  |
| 11 i←i+1 | 0 |
| 12 endif |  |
| 13 od |  |
| 14} |  |

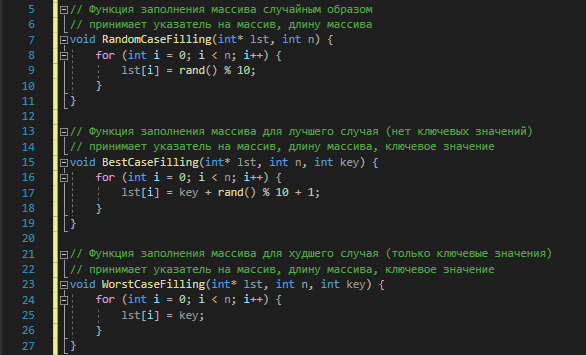
**Код Алгоритма 1:**

Рис. 2 – Функции, заполнения массива

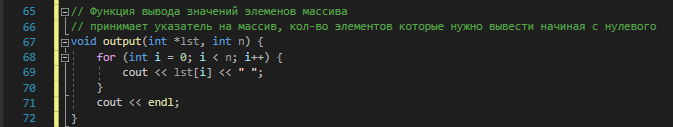


Рис. 3 – Функция вывода массива на экран

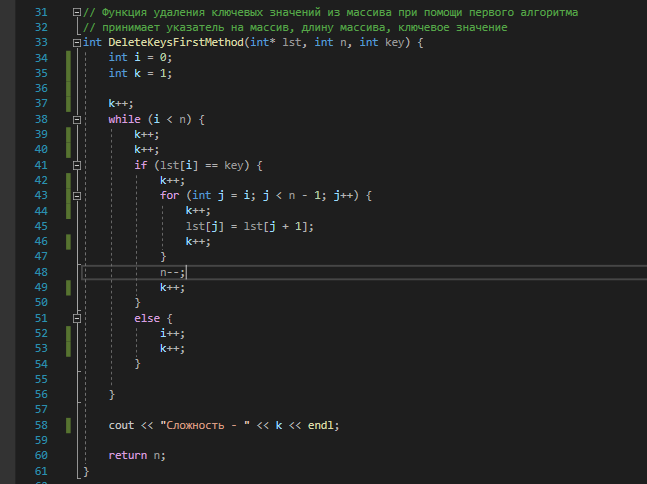


Рис. 4 – Функция, удаления ключевых элементов

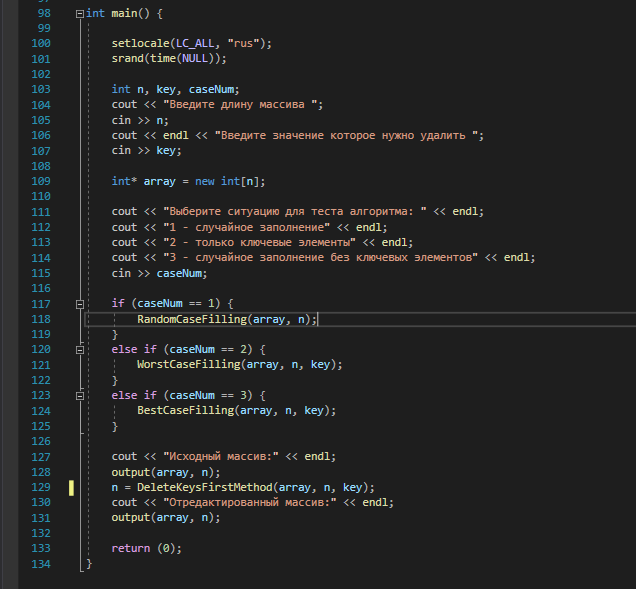


Рис. 5 – Функция main()

**Тестирование Алгоритма 1**

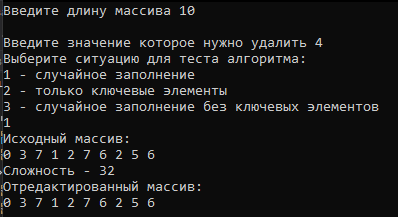


Рис. 6 – массив заполнен случайными числами, N = 10

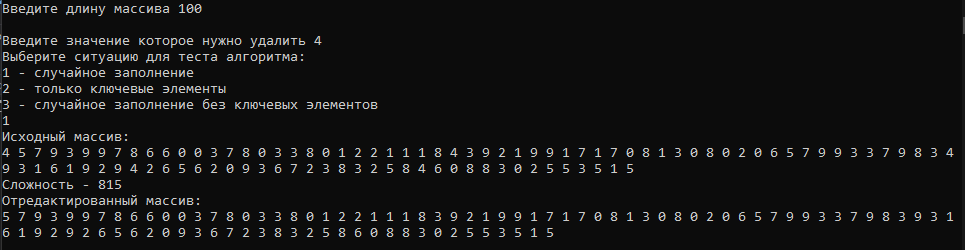


Рис. 7 – массив заполнен случайными числами, N = 100

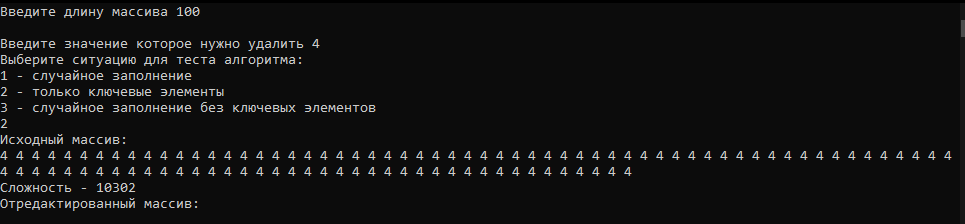


Рис. 8 – проверка худшего случая при N = 100

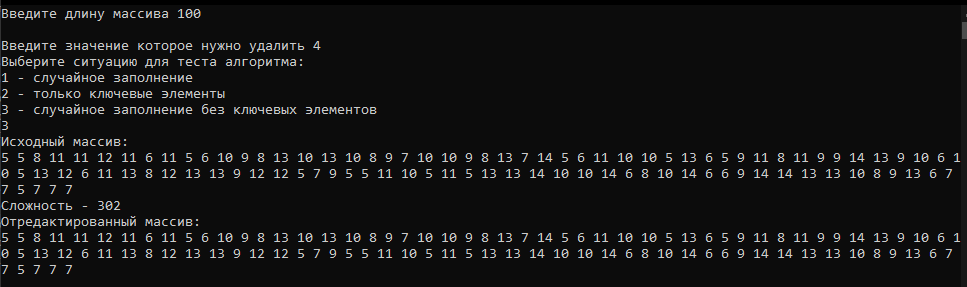


Рис. 9 – проверка лучшего случая при N = 100

## Алгоритм 2

**Постановка задачи:**

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.

**Модель решения поставленной задачи:**

* + 1. Алгоритм проходит по индексам массива от 1 до n. Если элемент массива с индексом i не равен значению, введенному пользователем – key – значение индекса j увеличивается на единицу, благодаря чему происходит движение по элементам массива. В противном случае, значение j не увеличивается и происходит сдвиг массива влево, удаляющий ключевой элемент.

Блок-схема алгоритма:

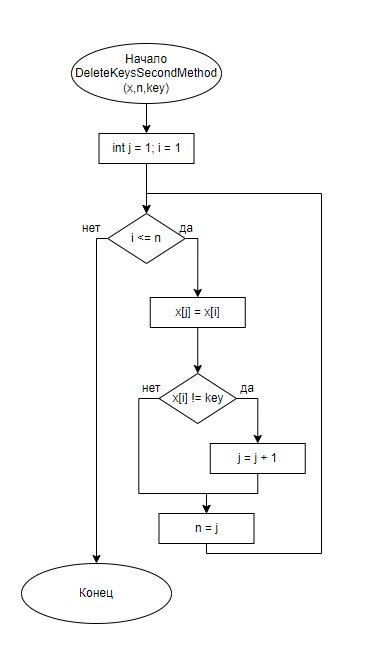


Рис. 10 – Блок-схема Алгоритма 2

* + 1. Доказательство корректности цикла

Инвариант цикла: элементы массива с индексом меньше чем j не равны удаляемому значению key. До начала выполнения цикла, значений до j, которое равно 1, вовсе нет. После каждой итерации цикла, при удалении элемента значение j не меняется (новые элементы не добавляются), либо добавляется корректное значение и значение j увеличивается на 1. Таким образом, j будет хранить в себе количество элементов массива, не равных ключевому значению key. В исключительных случаях j = 0, если все элементы были подвержены удалению, либо j=n, если все элементы не были удалены.

* + 1. Вычислительная сложность алгоритма:

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

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

Табл. 3 – Алгоритм 2, лучший случай

|  |  |
| --- | --- |
| Оператор | Количество выполнений оператора |
| 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++ | 0 |
| 7 endif |  |
| 8 od |  |
| 9 n←j | 1 |
| 10} |  |

Табл. 4 – Алгоритм 2, худший случай

|  |  |
| --- | --- |
| Оператор | Количество выполнений оператора |
| 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} |  |

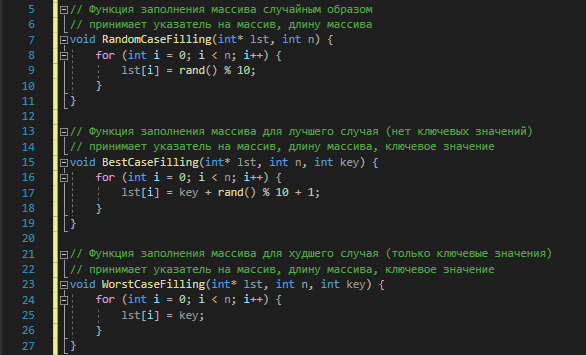
**Код Алгоритма 2:**

Рис. 11 – Функции, заполняющие массив

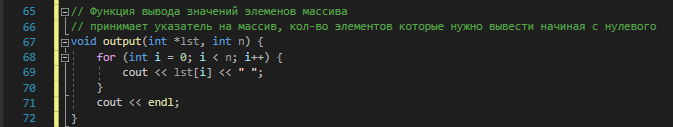


Рис. 12 – Функция для вывода массива на экран

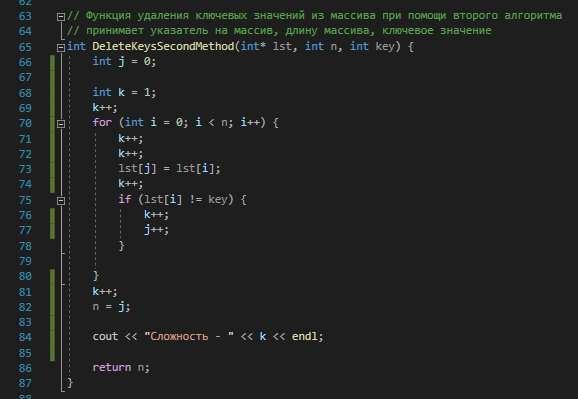


Рис. 13 – Функция, удаляющая ключевые элементы

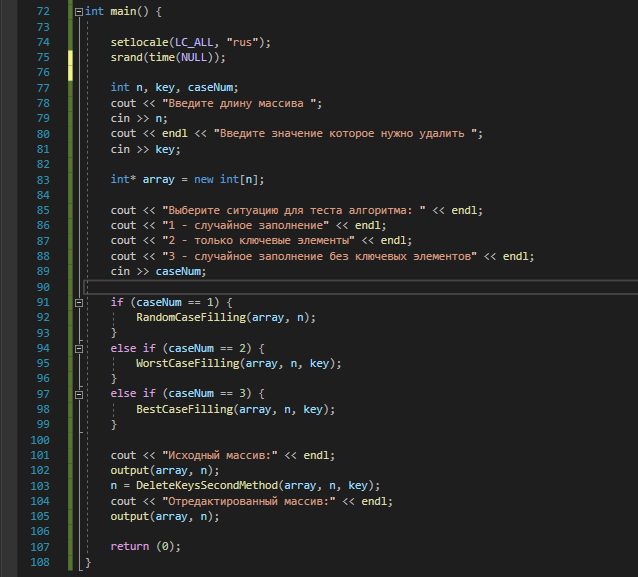


Рис. 14 – Функция main, часть 2

**Тестирование Алгоритма 2:**

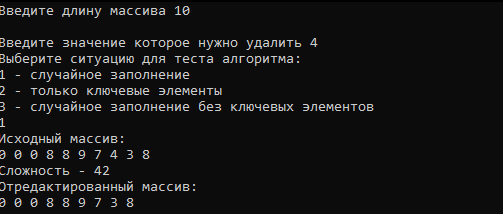


Рис. 15 – массив заполнен случайными числами, n = 10

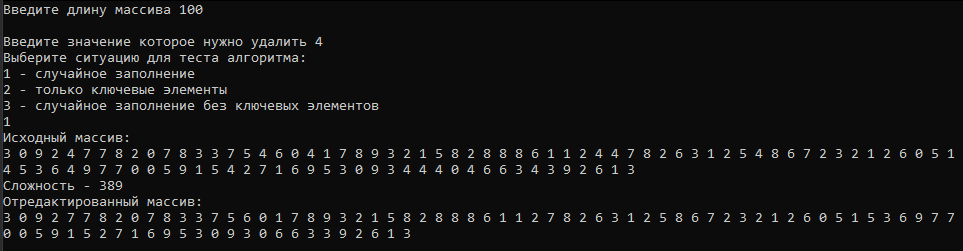


Рис. 16 – массив заполнен случайными числами, N = 100

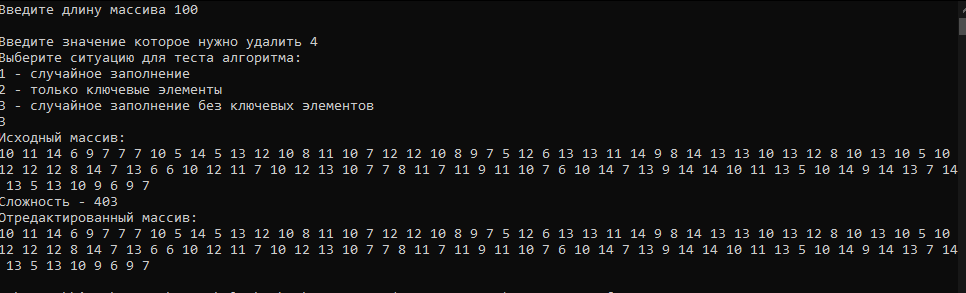


Рис. 17 – проверка худшего случая при N = 100

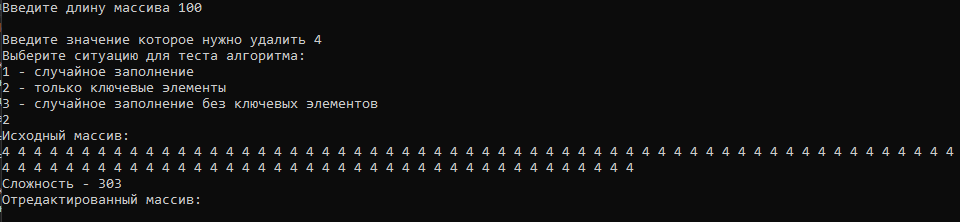


Рис. 18 – проверка лучшего случая при N = 100

Согласно проведенному тестированию теоретические результаты равны практическим, что говорит о верности теоретического расчета.

# Задание 2

Вариант 11

**Постановка задачи:**

Дан одномерный массив из n элементов целого типа. Определить, сколько раз в массив входит максимальное значение.

**Модель решения поставленной задачи:**

Инициализируется статический массив длины 2, в котором с индексом нуль – значение максимального элемента массива, с индексом единица – кол-во к конкретному моменту найденных элементов с максимальным значением. В цикле от нуля до n на максимальное значение проверяются все элементы: если элемент больше максимального значения, то это значение обновляется, обновляется кол-во найденных макс элементов, иначе если элемент равен максимальному значению, то кол-во найденных максимальных значений увеличивается на единицу.

**Разработка эффективного алгоритма:**

* + 1. Блок-схема алгоритма

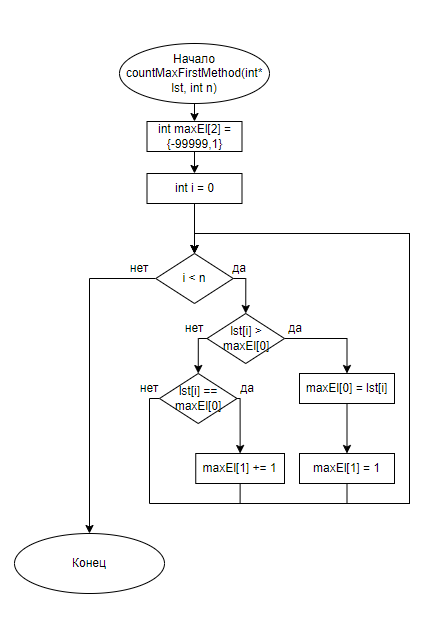


Рис. 19 – Блок-схема алгоритма

* + 1. Инвариант: в перебираемом массиве от нулевого до i-того элемента есть от одного до n максимальных элементов;
    2. Среди любого массива чисел, как минимум, одно будет больше или равно остальным, следовательно, при переборе всех элементов массива инвариант окажется верным как до, так и после выполнения цикла.
    3. Определение вычислительной сложности алгоритма:

Табл. 5 – худший случай (возрастающая последовательность)

|  |  |
| --- | --- |
| Оператор | Количество выполнений оператора |
| 1 countMax(int\* lst, int n) |  |
| 2 maxEl[2] ← {-99999,1} | 1 |
| 3 for i ← 0 to n do | n + 1 |
| 4 if lst[i] > maxEl[0] then | n |
| 5 maxEl[0] = lst[i] | n |
| 6 maxEl[1] ← 1 | n |
| 7 endif |  |
| 8 else if lst[i] == maxEl[0] then | 0 |
| 9 maxEl[1] ++ | 0 |
| 10 endif |  |
| 11 od |  |

Табл. 6 – лучший случай (убывающая последовательность)

|  |  |
| --- | --- |
| Оператор | Количество выполнений оператора |
| 1 countMax(int\* lst, int n) |  |
| 2 maxEl[2] ← {-99999,1} | 1 |
| 3 for i ← 0 to n do | n + 1 |
| 4 if lst[i] > maxEl[0] then | n |
| 5 maxEl[0] = lst[i] | 1 |
| 6 maxEl[1] ← 1 | 1 |
| 7 endif |  |
| 8 else if lst[i] == maxEl[0] then | n - 1 |
| 9 maxEl[1] ++ | 0 |
| 10 endif |  |
| 11 od |  |

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

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

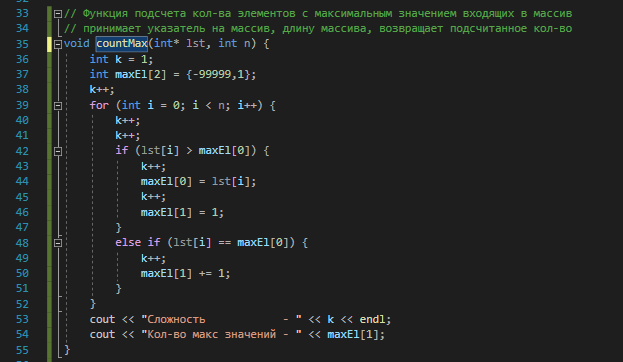
**Реализация алгоритма варианта в виде одной функции:** 

Рис. 20 – Функция подсчета кол-ва элементов с макс. значением

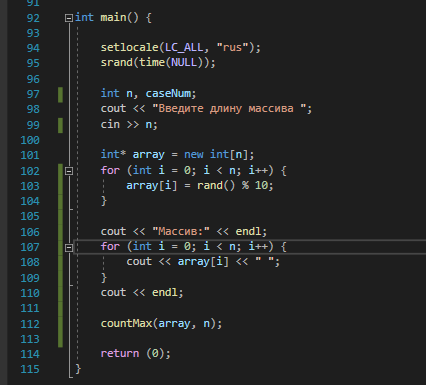


Рис. 21 – Функция main()

**Тестирование алгоритма на массиве из 10 чисел:**

Табл. 7 – тесты

|  |  |
| --- | --- |
| Случайное заполнение массива |  |
|  |  |
| Лучший случай заполнения:  массив – убывающая последовательность |  |
|  | Теоретическая сложность T(10) = 10 \* 4 + 2 = 42 |
| Худший случай заполнения:  массив – возрастающая последовательность |  |
|  | Теоретическая сложность T(10) = 10 \* 3 + 4 – 1 = 33 |

**Тестирование алгоритма на массиве из 100 чисел:**

Табл. 7 – тесты

|  |  |
| --- | --- |
| Случайное заполнение массива |  |
|  |  |
| Лучший случай заполнения:  массив – убывающая последовательность |  |
|  | Теоретическая сложность T(100) = 100 \* 4 + 2 = 402 |
| Худший случай заполнения:  массив – возрастающая последовательность |  |
|  | Теоретическая сложность T(100) = 100 \* 3 + 4 – 1 = 303 |

Согласно проведенному тестированию теоретические результаты равны практическим, что говорит о верности теоретического расчета.

# Вывод

В результате проделанной работы были приобретены навыки по определению сложности алгоритмов на теоретическом и практическом уровнях, а также по выбору эффективного алгоритма решения задачи из нескольких.