

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное*

*учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

# РТУ МИРЭА

Отчет по выполнению практического задания № 1.1

Вариант 5

**Тема:** «Оценка вычислительной сложности алгоритма»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Тринеев П.С.

ИКБО-32-22



Номер группы



Фамилия И.О.



Группа:

1. **ОТЧЕТ ПО ЗАДАНИЮ 1**
   1. **Формулировка задачи.**

**Цель:** приобретение практических навыков:

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

**Задание 1.** Выбрать эффективный алгоритм вычислительной задачи из двух предложенных, используя теоретическую и практическую оценку вычислительной сложности каждого из алгоритмов, а также его ёмкостную сложность.

* 1. **Математическая модель решения задачи.**

**a)** Описание алгоритма блок-схемой

Ниже представлены блок схемы двух алгоритмов (рис. 1, рис. 2)

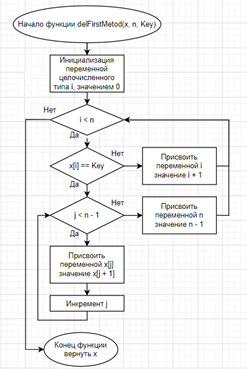
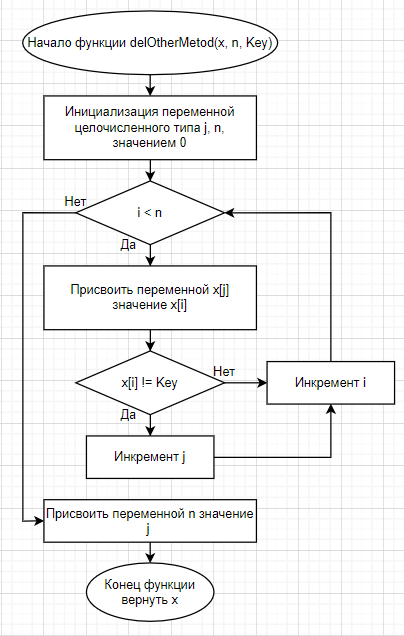
 ****

Рисунок 1. Рисунок 2.

Алгоритм 1.

На вход функции подается переменные: x, n, Key. Инициализация переменной целочисленного типа i значением 0. (1)Проверка если i <n. Если Нет, то конец функции вернуть переменную x. (2)Если Да, то проверка равенства элемента x[i] и переменной Key. Если Нет, то присвоить переменной i значение i + 1 вернуться к проверке 1. (3)Если Да, то проверка j <n -1. Если Нет, то присвоить переменной n значение n – 1 вернуться к проверке 1. Если Да, то присвоить переменной x[j] значение x[j + 1]. Инкремент j, вернуться к проверке 3.

Алгоритм 2.

На вход функции подается переменные: x, n, Key. Инициализация переменной целочисленного типа j значением 0. (1)Проверка i < n. Если Нет, то присвоить переменной n значение j, конец функции вернуть x. Если Да, то присвоить переменной x[j] значением x[i]. (2)Проверка неравенства x[i] и Key. Если Нет, то инкремент i, вернуться к проверке 1. Если Да, то инкремент j, инкремент i, вернуться к проверке 1.

**b)** Определить вычислительную сложность алгоритма

Все данные первого алгоритма, для вывода функции роста представлены в таблице 1.

|  |  |  |
| --- | --- | --- |
| Номер оператора | Оператор | Кол-во выполнений оператора в строке |
| 1 | int i = 0; | 1 |
| 2 | while (i < n) | n + 1 |
|  | { |  |
| 3 | if (x[i] == key) | n |
|  | { |  |
| 4 | for (int j = i; j <= n – 1; j++) | n + 1 |
|  | { |  |
| 5 | x[j] = x[j + 1] | n |
|  | } |  |
| 6 | n = n - 1 | n |
|  | } |  |
| 7 | else | n |
|  | { |  |
| 8 | i = i + 1 | n |
|  | } |  |
|  | } |  |
| 9 | return x; |  |

Таблица 1.

Оператор 1. Выполняется 1 раз.

Оператор 2. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится 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(if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 4. Это оператор – блок оператора if. Выполняется столько раз, сколько и if – n раз. А также; определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится. 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 раз за время работы.

Оператор 5. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 6. Это оператор – блок оператора if. Выполняется столько раз, сколько и if – n раз.

Оператор 7(else). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 8. Это оператор – блок оператора else. Выполняется столько раз, сколько и else – n раз.

Оператор 9. Этот оператор вне цикла он будет выполняться 1раз.

Обозначим время выполнения алгоритма T(n) – функция, зависящая от n. Определим время выполнения алгоритма – как сумму времени выполнения каждого оператора:  
T(n) = 1 + (n + 1) + n + (n + 1) + n + n + n + n  
Раскроем скобки и приведем подобные:  
T(n) = 7\*n + 3, т.е. имеем линейную зависимость от размера выходных данных.

Так как требуется определить порядок роста времени от n, то константами можно пренебречь.

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

Рассмотрим лучший случай: значение равное key – это значение первого элемента массива. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в лучшем случае линейно зависит от n.

Рассмотрим средний случай: значение равное key – это значение элемента в позиции i=n/2 (целое значение) массива. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в среднем случае линейно зависит от n. Вывод. Порядок роста – линейный.

Все данные второго алгоритма, для вывода функции роста представлены в таблице 2.

|  |  |  |
| --- | --- | --- |
| Номер оператора | Оператор | Кол-во выполнений оператора в строке |
| 1 | int j = 0; | 1 |
| 2 | for (int I = 0; I < n; i++) | n + 1 |
|  | { |  |
| 3 | x[j] = x[i] | n |
| 4 | if (x[i] != key) | n |
|  | { |  |
| 5 | j++ | n |
|  | } |  |
|  | } |  |
| 6 | n = j; | 1 |
| 7 | return x |  |

Таблица 2.

Оператор 1. Выполняется один раз.

Оператор 2. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится 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: обеспечивался вход в цикл и один раз при выходе из цикла, таким образом, всего n + 1 раз за время работы.

Оператор 3. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

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 – n раз.

Оператор 6. Этот оператор вне цикла ион выполняться будет 1раз.

Оператор 7. Этот оператор вне цикла ион выполняться будет 1раз.

Обозначим время выполнения алгоритма T(n) – функция, зависящая от n. Определим время выполнения алгоритма - как сумму времени выполнения каждого оператора:  
T(n) = 1 + (n + 1) + n +n + n + 1.  
Раскроем скобки и приведем подобные:  
T(n) = 4\*n + 3, т.е. имеем линейную зависимость от размера выходных данных.

В худшем случае: цикл выполняется полное число раз – n+1.

В наилучшем случае: значение, которое ищем, в первом элементе массива. Условие цикла выполнится два раза, не зависит от n.

В среднем случае: значение, которое ищем, где-то в середине массива в позиции n/2. Подставим n/2 в формулу T(n) = 4\*n + 3 вместо n:

T(n)= 2\*n + 3, т.е. тоже линейная зависимость.

Вывод. Худший и средний случай дают одинаковый порядок роста вымени от n. Значит, в целом, скорость роста – линейная.

**1.3 Реализовать алгоритм в виде функции и отладить**Ниже представленный фрагменты кода с подсчетом сравнений, передвижения элементов, удаления элементов в алгоритме (рис. 3, 4).

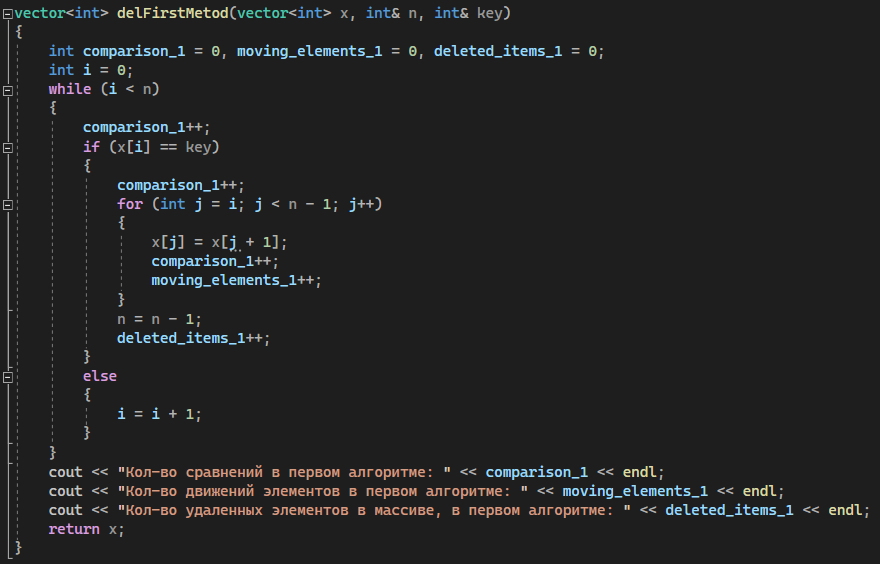


Рисунок 3. Первый алгоритмом.

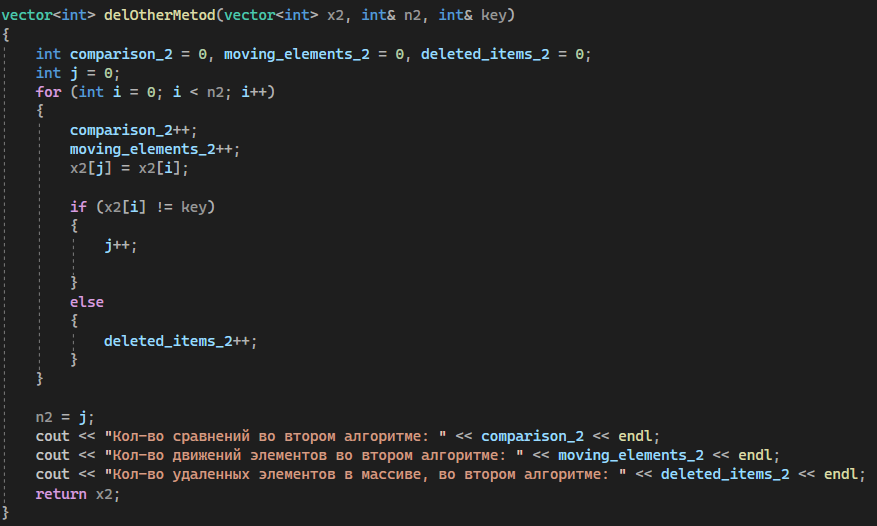


Рисунок 4. Второй алгоритм.

Вывод в консоль массивов с параметрами n = 10, n = 100 (рис. 5, 6)

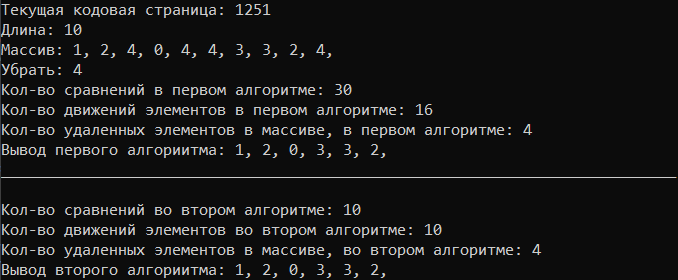


Рисунок 5. При n = 10.

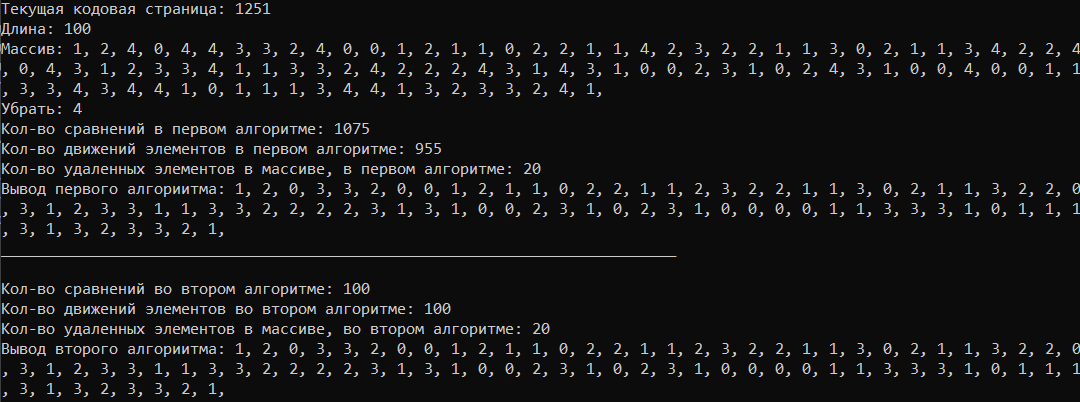


Рисунок 6. При n = 100.

**1.4. Реализовать функции: заполнение массива датчиком случайных чисел, вывод массива на экран монитора.**

Ниже представлена Фрагмент кода заполнения массива случайными числами и выводом массива на экран (рис. 7, 8, 9)

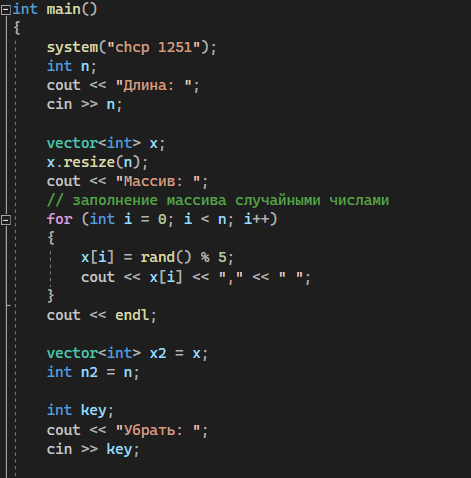


Рисунок 7. Алгоритм заполнения массива случайными числами.

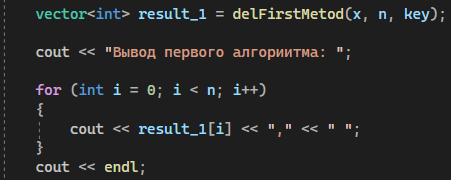


Рисунок 8. Вывод массива на экран первого алгоритма.

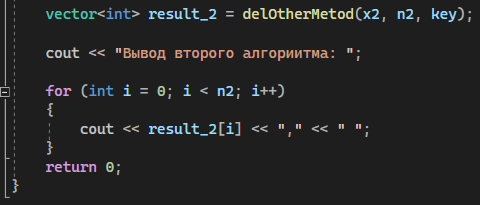


Рисунок 9. Вывод массива на экран второго алгоритма.

* 1. **Тестирование алгоритмов в различных ситуациях.**

1. **Случайное заполнение массива.**

Представлено ниже в рисунке 10.

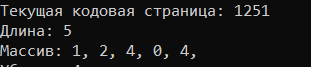


Рисунок 10. Случайное наполнение массива.

1. **Все элементы массива должны быть удалены.**

Выполнение решения представлен ниже в рисунке 11.

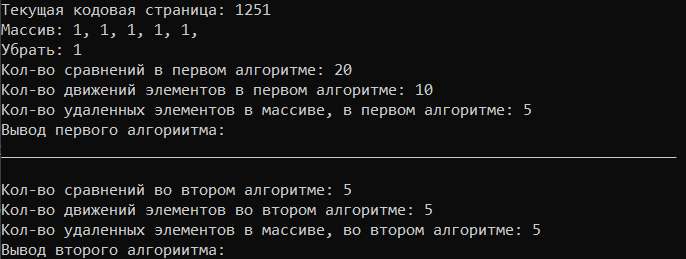
****

Рисунок 11. Удаление всех элементов массива.

1. **Ни один элемент не был удален.**

Выполнение решения представлен ниже в рисунке 12.

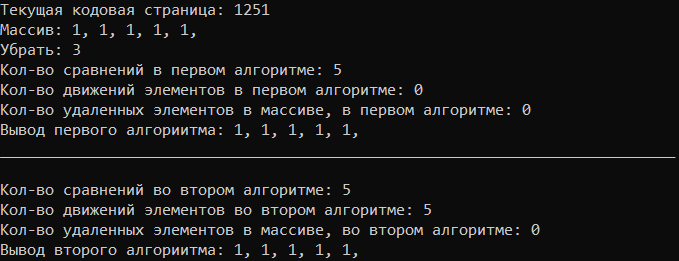


Рисунок 12. Ни один элемент не удален.

Сравнение:

1. В случае с удалением всех элементов вычислительная сложность первого алгоритма возрастает прямо пропорционально количеству элементов в массиве, а во втором алгоритме вычислительная сложность ровна количеству элементов в массиве.
2. В случае с неизменностью массива вычислительная сложность первого алгоритма равна количеству элементов в массиве, во втором же алгоритме остается прежней

**1.6 Представить в отчёте результаты тестирования**

* **При n = 10.**

Алгоритм 1:

Кол-во сравнений в первом алгоритме: 30

Кол-во движений элементов в первом алгоритме: 16

Кол-во удаленных элементов в массиве, в первом алгоритме: 4

Алгоритм 2;

Кол-во сравнений во втором алгоритме: 10

Кол-во движений элементов во втором алгоритме: 10

Кол-во удаленных элементов в массиве, во втором алгоритме: 4

* **При n = 100.**

Алгоритм 1;

Кол-во сравнений в первом алгоритме: 1075

Кол-во движений элементов в первом алгоритме: 955

Кол-во удаленных элементов в массиве, в первом алгоритме: 20

Алгоритм 2;

Кол-во сравнений во втором алгоритме: 100

Кол-во движений элементов во втором алгоритме: 100

Кол-во удаленных элементов в массиве, во втором алгоритме: 20

* **При удаление всего массива.**

Алгоритм 1;

Кол-во сравнений в первом алгоритме: 20

Кол-во движений элементов в первом алгоритме: 10

Кол-во удаленных элементов в массиве, в первом алгоритме: 5

Алгоритм 2;

Кол-во сравнений во втором алгоритме: 5

Кол-во движений элементов во втором алгоритме: 5

Кол-во удаленных элементов в массиве, во втором алгоритме: 5

* **При не изменение массива.**

Алгоритм 1;

Кол-во сравнений в первом алгоритме: 5

Кол-во движений элементов в первом алгоритме: 0

Кол-во удаленных элементов в массиве, в первом алгоритме: 0

Алгоритм 2;

Кол-во сравнений во втором алгоритме: 5

Кол-во движений элементов во втором алгоритме: 5

Кол-во удаленных элементов в массиве, во втором алгоритме: 0

**1.6 Оценить ёмкостную сложность алгоритма.**

* **При n = 10**

Время = 5,951 сек., Затрата памяти = 1МБ

* **При n = 100**

Время = 2,609 сек., Затрата памяти = 1МБ

* **При удаление всего массива.**

Время = 1,415 сек., Затрата памяти = 1МБ

* **При не изменение массива.**

Время = 5,731 сек., Затрата памяти = 1МБ

1. Более эффективным алгоритмом является 2. Этот вывод можно сделать по практический и теоретический способ проверки корректности алгоритма.
2. Задание 2.

Обход матрицы по спирали (по часовой стрелке: первая строка, последний столбец, нижняя строка, первый столбец)

3.2 Математическая модель решения задачи.

а) Описать, как выполняется алгоритм

Ниже представлена блок-схема программы (рис. 13)

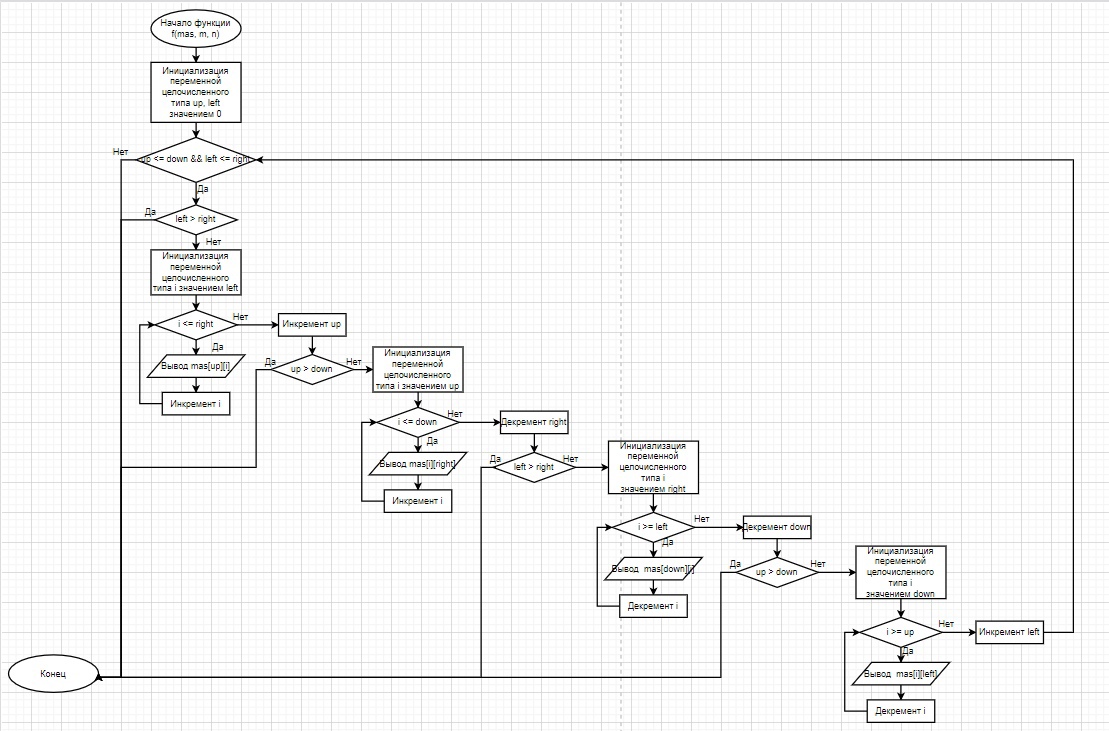


Рисунок 13.

На функцию подается переменные: mas, n, m. Инициализация переменной целочисленного типа up, left значением 0. (1)Проверка up <= down && left <= right. Если Нет, то конец цикла. Если Да, то (2)Проверка left > right. Если Да, то конец цикла. Если Нет, то инициализация переменной целочисленного типа i значением left. (3)Проверка i <= right. Если Да, то вывод mas[up][i], инкремент i вериться к проверке 3. Если Нет, то инкремент up. (4)Проверка up > down. Если да, то конец цикла. Если Нет, то инициализация переменной целочисленного типа i значением up. (5)Проверка i <= down. Если Да, то вывод mas[i][right], инкремент i, вернуться к проверке 5. Если Нет, Декремент right. (6)Проверка left > right. Если Да, то конец цикла. Если Нет, то инициализация переменной целочисленного типа i значением right. (7)Проверка i >= left. Если Да, то вывод mas[down][i], декремент i, вернуться к проверке 7. Ели Нет, то декремент down. (8)Проверка up > down. Если Да, то конец цикла. Если Нет, то инициализация переменной целочисленного типа i значением down. (9)Проверка i >= up. Если Да, то вывод mas[i][left], декремент i, вернуться к проверке 9. Если Нет, Инкремент left вернуться к проверке 1.

b) Определить вычислительную сложность алгоритма

Все данные алгоритма, для вывода функции роста представлены в таблице 3.

|  |  |  |
| --- | --- | --- |
| Номер оператора | Оператор | Кол-во выполнений оператора в строке |
| 1 | int up = 0, down = m - 1, right = n - 1, left = 0; | 1 |
| 2 | while (up <= down && left <= right) { | n + 1 |
| 3 | if (left > right) { | n |
| 4 | break; | 1 |
|  | } |  |
| 5 | for (int i = left; i <= right; i++){ | n + 1 |
| 6 | cout << mas[up][i] << " "; | n |
|  | } |  |
| 7 | up++; | n |
| 8 | if (up > down){ | n |
| 9 | break; | 1 |
|  | } |  |
| 10 | for (int i = up; i <= down; i++) { | n + 1 |
| 11 | cout << mas[i][right] << " "; | n |
|  | } |  |
| 12 | right--; | n |
| 13 | if (left > right) { | n |
| 14 | break; | 1 |
|  | } |  |
| 15 | for (int i = right; i >= left; i--) { | n + 1 |
| 16 | cout << mas[down][i] << " "; | n |
|  | } |  |
| 17 | down--; | n |
| 18 | if (up > down) { | n |
| 19 | break; | 1 |
|  | } |  |
| 20 | for (int i = down; i >= up; i--) { | n + 1 |
| 21 | cout << mas[i][left] << " "; | n |
|  | } |  |
| 22 | left++; | n |
|  | } |  |

Таблица 3

Оператор 1. Выполняется один раз.

Оператор 2. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится.

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

Оператор 4. Это оператор – блок оператора if. Выполняется 1 раз, т.к. это break выход из цикла.

Оператор 5. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится.

Оператор 6. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 7. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 8(if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 9. Это оператор – блок оператора if. Выполняется 1 раз, т.к. это break выход из цикла.

Оператор 10. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится

Оператор 11. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 12. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 13(if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 14. Это оператор – блок оператора if. Выполняется 1 раз, т.к. это break выход из цикла.

Оператор 15. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится.

Оператор 16. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 17. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 18(if). Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 19. Это оператор – блок оператора if. Выполняется 1 раз, т.к. это break выход из цикла.

Оператор 20. Определяется количество выполнений условия цикла при просмотре всех значений (худший случай), т.е. сколько раз он применится.

Оператор 21. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Оператор 22. Это оператор тела цикла, т.е. он выполняется n раз – количество входов в тело цикла.

Обозначим время выполнения алгоритма T(n) – функция, зависящая от n.

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

T(n) = 1 + (n + 1) + n + 1 + (n + 1) + n + n + n + 1 + (n + 1) + n + n + n + 1 + (n + 1) + n + n + n + 1 + (n + 1) + n + n.

Раскроем скобки и приведем подобные:

T(n) = 17\*n + 10, т.е. имеем линейную зависимость времени от размера входных данных.

Так как требуется определить порядок роста времени от n, то константами можно пренебречь.

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

Рассмотрим лучший случай: значение равное m, n = 1, тогда мы получим матрицу с 1 элементом. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в лучшем случае линейно зависит от n.

Рассмотрим средний случай: значение равное m, n = 100, тогда мы получим матрицу с 100x100 элементами. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в лучшем случае линейно зависит от n. Цикл будет выполняться все равно n+1 раз. Тогда T(n) в среднем случае линейно зависит от n. Вывод. Порядок роста – линейный.

3.3 Реализовать алгоритм в виде функции и отладить

Ниже представленный фрагменты кода с подсчетом сравнений в алгоритме (рис. 14).

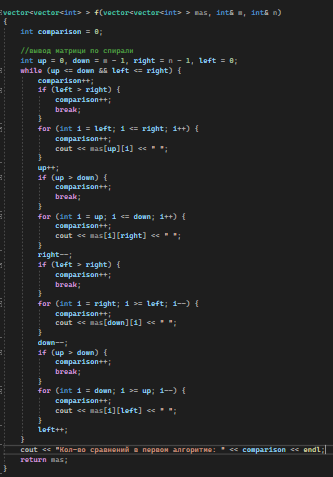


Рисунок 14.

3.4. Реализовать функции: заполнение массива датчиком случайных чисел и вывод матрицы на экран.

Ниже представлены реализации функции с датчиком случайных чисел и вывод матрицы на экран (рис. 15, 16)

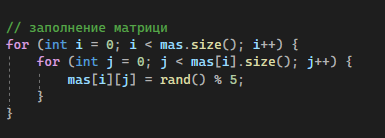


Рисунок 15.

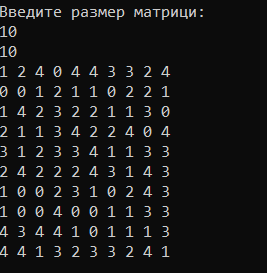


Рисунок 16.

* 1. Представить в отчёте результаты тестирования и оценить ёмкостную сложность алгоритма.

При m, n = 10:

Кол-во сравнений в первом алгоритме: 106.

Время = 3,052 сек., Затрата памяти = 1МБ

При m, n = 100:

Кол-во сравнений в первом алгоритме: 10051.

Время = 3,48 сек., Затрата памяти = 1МБ

При m, n = 1000:

Кол-во сравнений в первом алгоритме: 1000501.

Время = 1:31 мин., Затрата памяти = 10МБ

1. Используемые ресурсы.
2. Лекционный материал Макеевой О.В.
3. Примеры 1, 2 из задания <https://online-edu.mirea.ru/pluginfile.php?file=%2F1044518%2Fmod_assign%2Fintroattachment%2F0%2FПР_2.1%20%28Теоретическая%20сложность%20алгоритма%29.pdf&amp;forcedownload=1> 12/03/2023
4. Материалы: Введение в структуры и алгоритмы обработки данных. <https://online-edu.mirea.ru/pluginfile.php?file=%2F1044598%2Fmod_folder%2Fcontent%2F0%2FВведение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Ч.%201%20-%20учебное%20пособие%2C%202022.pdf&forcedownload=1> 12/03/2023