

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего

образования

**«МИРЭА - Российский технологический университет»**

РТУ МИРЭА

Институт информационных технологий

Кафедра математического обеспечения и стандартизации информационных технологий

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1**

***«*Поразрядные операции и их применение*»***

по дисциплине

**«**Структуры и алгоритмы обработки данных**»**

Выполнил студент группы *ИКБО-03-22 Хохлинов Д.И.*

Принял *Сорокин А.В.*

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | « » 2023 г. |  |
|  |  |  |
| «Зачтено» | « » 2023 г. |  |

Москва 2023

**СОДЕРЖАНИЕ**

[1 ЗАДАНИЕ 1 3](#_Toc145539495)

[1.1 Постановка задачи 3](#_Toc145539496)

[1.2 Описание алгоритмов 4](#_Toc145539497)

[1.3 Блок-схемы алгоритмов 5](#_Toc145539498)

[1.4 Код программы 8](#_Toc145539499)

[1.5 Тестирование 10](#_Toc145539500)

[2 ЗАДАНИЕ 2 12](#_Toc145539501)

[2.1 Постановка задачи 12](#_Toc145539502)

[2.2 Описание структуры хранения битового массива в оперативной памяти 12](#_Toc145539503)

[2.3 Описание алгоритмов 13](#_Toc145539504)

[2.4 Блок-схемы алгоритмов 15](#_Toc145539505)

[2.4 Код программы 17](#_Toc145539506)

[2.5 Тестирование 20](#_Toc145539507)

[3 ЗАКЛЮЧЕНИЕ 24](#_Toc145539508)

[4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc145539509)

# **1 ЗАДАНИЕ 1**

## **Постановка задачи**

В соответствии с [2] постановка задачи такова:

Разработать программу, которая продемонстрирует выполнение упражнений варианта. Результаты выполнения упражнения выводить на монитор. Требования к упражнениям:

Упражнение 1 - Определить переменную целого типа, присвоить ей значение, используя константу в шестнадцатеричной системе счисления. Разработать оператор присваивания и его выражение, которое установит седьмой, пятый и тринадцатый биты исходного значения переменной в значение 1, используя соответствующую маску и поразрядную операцию.

Упражнение 2 - Определить переменную целого типа. Разработать оператор присваивания и его выражение, которое обнуляет четыре старших бита исходного значения переменной, используя соответствующую маску и поразрядную операцию. Значение в переменную вводится с клавиатуры.

Упражнение 3 - Определить переменную целого типа. Разработать оператор присваивания и выражение, которое умножает значение переменной на 512, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.

Упражнение 4 - Определить переменную целого типа. Разработать оператор присваивания и выражение, которое делит значение переменной на 128, используя соответствующую поразрядную операцию. Изменяемое число вводится с клавиатуры.

Упражнение 5 - Определить переменную целого типа. Разработать оператор присваивания и выражение, которое устанавливает n-ый бит в 1, используя маску. Маска инициализируется единицей в старшем разряде. Изменяемое число вводится с клавиатуры.

## **1.2 Описание алгоритмов**

В вышеперечисленных упражнениях будем использовать тип переменной unsigned long (32 бита на переменную). Для решения поставленной задачи будем использовать следующие алгоритмы:

Упражнение 1. Установим для х некоторое константное значение в 16-ичной СС, например, 89ABCDEF. Затем инициализируем переменную mask типа unsigned long выражением (1 << 5) | (1 << 7) | (1 << 13). Оно представляет собой побитовую дизъюнкцию значений, полученных побитовым сдвигом числа 1 на 5, 7 и 13 разрядов влево. Для получения необходимого результата необходимо вычислить побитовую дизъюнкцию переменных x и mask и затем вывести ее на экран.

Упражнение 2. Введем число х с клавиатуры. Затем инициализируем переменную mask, установив все биты в 1 и выполнив побитовый сдвиг полученного значения на 4 разряда вправо. После этого необходимо вычислить побитовую конъюнкцию переменных х и mask и вывести ее на экран.

Например, пусть введено число х = 2124975845 (в двоичной СС число будет иметь вид 11111101010100010001110111001012). Маска имеет вид 000011111111111111111111111111112. После побитовой конъюнкции этих значений имеем результат 00001101010100010001110111001012 = 111709925.

Упражнение 3. Введем число х с клавиатуры. Затем выполним побитовый сдвиг числа х на 9 разрядов влево (т.к. ) и выведем результат на экран.

Например, пусть введено число 17 = 100012. Тогда результатом будет являться число 100010000000002 = 8704.

Упражнение 4. Введем число х с клавиатуры. Затем выполним побитовый сдвиг числа х на 7 разрядов вправо (т.к. ) и выведем результат на экран.

Например, пусть введено число 548 = 10001001002. Тогда результатом будет являться число 1002 = 4.

Упражнение 5. Введем число х с клавиатуры. Затем введем число n – номер разряда, который необходимо установить в 1. Число n должно находиться в диапазоне [0; 31]; если оно выходит за границы диапазона слева или справа, то принимаем его равным 0 или 31 соответственно. Инициализируем маску константой 8000000016 (что эквивалентно 32-разрядному двоичному числу, старший разряд которого равен 1, остальные разряды – 0). Затем выполним побитовый сдвиг маски вправо на разрядов и вычислим побитовую дизъюнкцию переменных х и mask, после чего выведем ее на экран.

Например, пусть введено число х = 4051 = 1111110100112 и число n = 17. Итоговая маска будет иметь вид 0000 0000 0000 0010 0000 0000 0000 00002. Результатом побитовой дизъюнкции будет число 0000 0000 0000 0010 0000 1111 1101 00112 = 135123.

## **1.3 Блок-схемы алгоритмов**

Приведем блок-схемы описанных алгоритмов для упражнений 1-5 (рисунки 1-5):

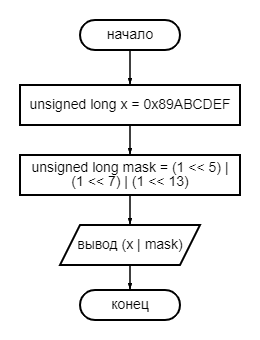


Рисунок 1 – Блок-схема алгоритма упражнения 1

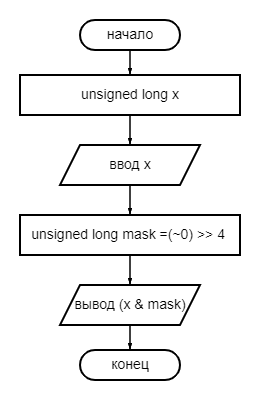


Рисунок 2 – Блок-схема алгоритма упражнения 2

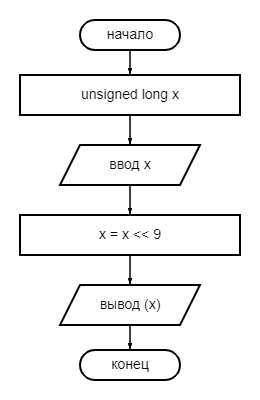


Рисунок 3 – Блок-схема алгоритма упражнения 3

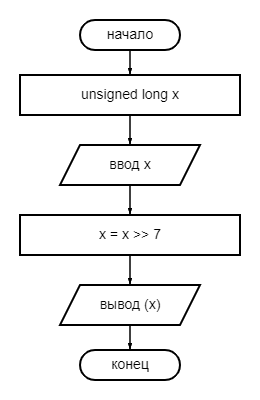


Рисунок 4 – Блок-схема алгоритма упражнения 4

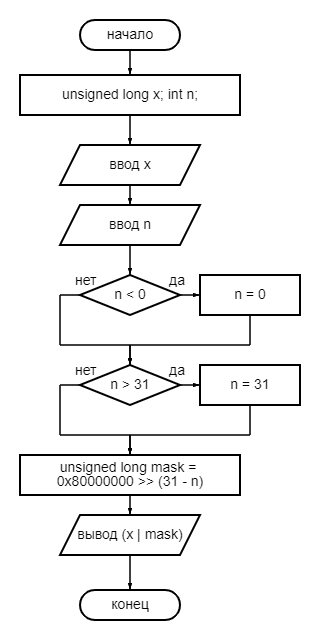


Рисунок 5 – Блок-схема алгоритма упражнения 5

## **1.4 Код программы**

Реализуем алгоритмы, описанные выше, на языке С++. Также для удобства поместим код всех алгоритмов в одну функцию и добавим отображение значения переменной х в двоичной и десятичной СС до и после выполнения алгоритма (листинг 1):

Листинг 1. Код программы, используемой для решения задания 1

#include <iostream>

#include <string>

#include <bitset>

#include <locale.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "ru");

cout << "Упражнение 1" << endl;

unsigned long x = 0x89ABCDEF;

unsigned long mask = 0;

mask = (1 << 5) | (1 << 7) | (1 << 13);

cout << "Значение х до применения маски: " << x << ", в двоичном виде: " << bitset<sizeof(x)\*8>(x) << endl;

cout << "Значение х после применения маски: " << (x | mask) << ", в двоичном виде: " << bitset<sizeof(x)\*8>(x|mask) << endl;

system("pause");

system("cls");

cout << "Упражнение 2" << endl;

cout << "Введите х: ";

cin >> x;

mask = (~0);

mask = mask >> 4;

cout << "Значение х до применения маски: " << x << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x) << endl;

cout << "Значение х после применения маски: " << (x & mask) << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x & mask) << endl;

system("pause");

system("cls");

cout << "Упражнение 3" << endl;

cout << "Введите х: ";

cin >> x;

cout << "Значение х до применения операции: " << x << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x) << endl;

x = x << 9;

cout << "Значение х после применения операции: " << x << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x) << endl;

system("pause");

system("cls");

cout << "Упражнение 4" << endl;

cout << "Введите х: ";

cin >> x;

cout << "Значение х до применения операции: " << x << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x) << endl;

x = x >> 7;

cout << "Значение х после применения операции: " << x << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x) << endl;

system("pause");

system("cls");

Продолжение листинга 1

cout << "Упражнение 5" << endl;

cout << "Введите х: ";

cin >> x;

cout << "Введите n (0-31): ";

int n;

cin >> n;

if (n < 0) n = 0;

if (n > 31) n = 31;

mask = 0x80000000;

mask = mask >> (31 - n);

cout << "Значение х до применения операции: " << x << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x) << endl;

cout << "Значение х после применения операции: " << (x | mask) << ", в двоичном виде: " << bitset<sizeof(x) \* 8>(x | mask) << endl;

}

## **1.5 Тестирование**

Проведем тестирование кода, изложенного в листинге 1. Будем использовать тестовые данные, описанные в пункте 1.2 (рисунки 6-10):

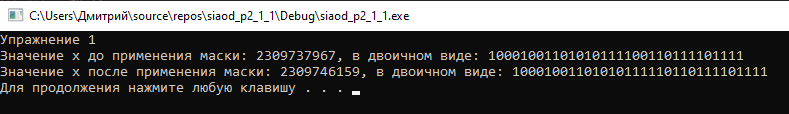


Рисунок 6 – Тестирование упражнения 1

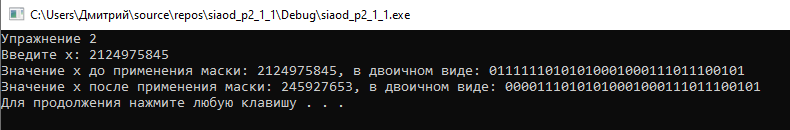


Рисунок 7 – Тестирование упражнения 2

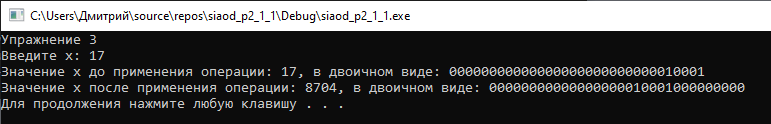


Рисунок 8 – Тестирование упражнения 3

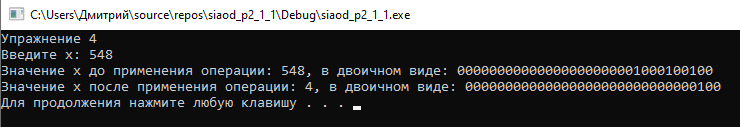


Рисунок 9 – Тестирование упражнения 4

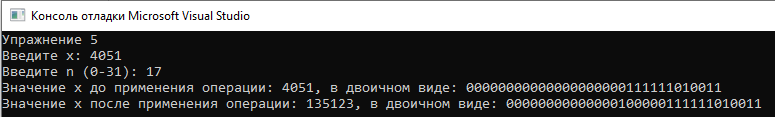


Рисунок 10 – Тестирование упражнения 5

По результатам тестирования программы фактические выходные данные совпадают с ожидаемыми, значит, можно говорить о правильности составленных алгоритмов.

# **2 ЗАДАНИЕ 2**

## **2.1 Постановка задачи**

В соответствии с [1]:

***Входные данные***: файл, содержащий не более n=107 неотрицательных целых чисел, среди них нет повторяющихся.

***Результат***: упорядоченная по возрастанию последовательность исходных чисел в выходном файле.

***Время работы программы***: ~10 с (до 1 мин. для систем малой вычислительной мощности).

***Максимально допустимый объём ОЗУ для хранения данных:*** 1 МБ.

## **2.2 Описание структуры хранения битового массива в оперативной памяти**

Очевидно, что максимально допустимого объема ОЗУ не хватит для решения этой задачи (для представления битового массива длины 107 требуется 107 / 8 / 1024 / 1024 1,192 МБ), поэтому возникают сомнения о правильности постановки задачи.

Для хранения массива битов в оперативной памяти используется массив типа unsigned char и длины 10 000 000 / 8 = 1 250 000. Каждому целому числу от 0 до 9999999 соответствует свой бит в массиве, как показано на рисунке 11:

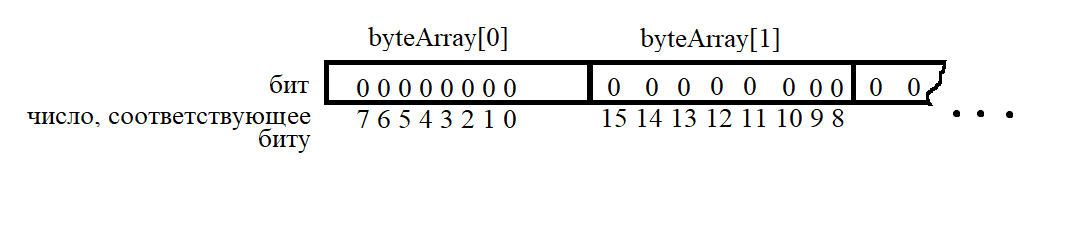


Рисунок 11 – Соответствие чисел битам в массиве

## **2.3 Описание алгоритмов**

Для решения задания будем использовать следующий алгоритм.

Выделим память под битовый массив, реализованный с помощью массива типа unsigned char длиной 1 250 000, и обнулим его. Затем предложим пользователю способ ввода: ручной или автоматический – из файла.

В случае ручного ввода пользователь вводит неповторяющиеся числа в диапазоне [0; 9 999 999], которые сразу же после ввода будем пытаться записать в битовый массив. Если бит, соответствующий очередному числу, равен 1, значит, данное число уже вводилось, и пользователь должен повторить ввод. Сигналом окончания ввода будет служить ввод числа -1.

В случае автоматического ввода пользователь вводит путь к файлу, в котором записаны целые числа в диапазоне [0; 9 999 999], каждое – на своей строке. Эти числа будут считываться в массив, при этом проверка на уникальность числа не требуется, исходя из постановки задачи.

После сортировки каждый бит битового массива считывается: если он равен 1, то на экран выводится соответствующее число.

Дадим более подробное описание алгоритмов занесения числа в массив битов, сортировки массива, заданного файлом, и считывания данных из этого массива.

Для занесения числа в массив битов используем функцию setNumber с аргументами n – int и data – указатель на массив типа unsigned char:

Инициализируем переменную int k = n; создаем маску типа unsigned char выражением 1 << (8 - (k % 8) - 1). Затем, если побитовая конъюнкция элемента data с индексом k / 8 и маски не равна 0, значит, вводимое число уже было записано в массив – тогда выполнение функции заканчивается и возвращается значение «ложь». В противном случае элементу data с индексом k / 8 присваивается значение побитовой дизъюнкции элемента data с индексом k / 8 и маски и возвращается значение «истина».

Для вывода битового массива используем функцию printNumbers с аргументом data – указателем на массив типа unsigned char:

Инициализируем переменную int n = 0. Далее в цикле for перебираем значения переменной i от 0 до 1 250 000 (длина массива). Для каждого прохода цикла создаем объект класса bitset<8> temp, используя в качестве аргумента для конструктора i-ый элемент массива data. Затем объявляем цикл for для переменной j от 7 до 0 включительно; в теле этого цикла выводим переменную n, если j-ый бит объекта temp равен 1, и увеличиваем n на 1.

Для заполнения битового массива из файла используем функцию insert с аргументами byte\_massive – указатель на массив типа unsigned char и path – строку, содержащую путь к файлу с числами:

Открываем файл с помощью объекта fstream data\_source. Затем считываем каждую строку из файла data\_source в переменную temp и вызываем функцию setNumber с аргументами temp и byte\_massive. После достижения конца файла закрываем его.

## **2.4 Блок-схемы алгоритмов**

Приведем блок-схемы для описанных алгоритмов (рисунки 12-14):

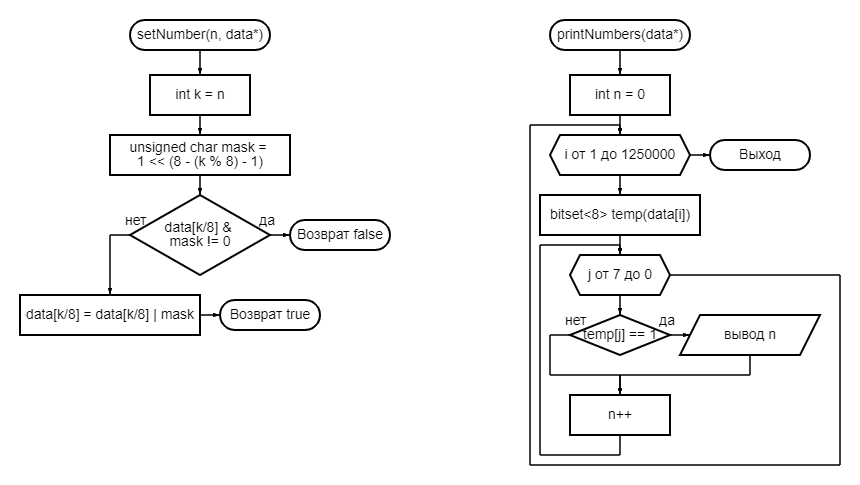


Рисунок 12 – Блок-схема алгоритмов функций setNumber и printNumbers

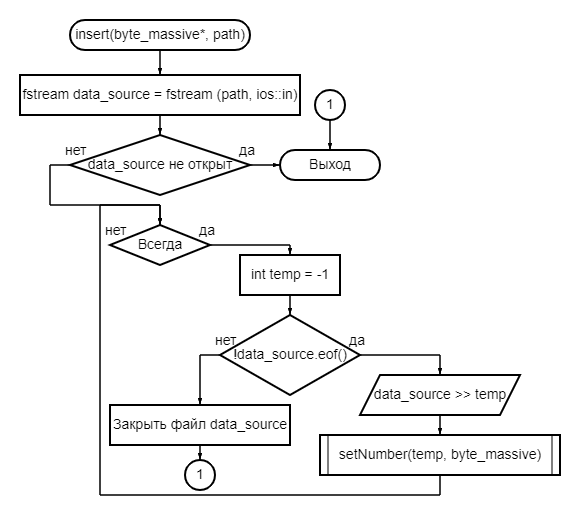


Рисунок 13 – Блок-схема алгоритма функции insert

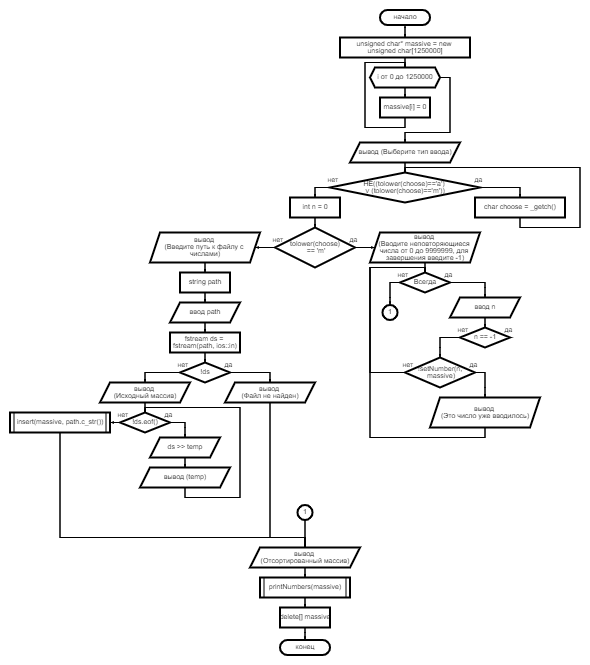


Рисунок 14 – Блок-схема алгоритма основной функции

## **2.4 Код программы**

Реализуем алгоритмы, описанные выше, на языке С++. Также добавим счетчик времени, затраченного на сортировку, в функции insert (листинг 2):

Листинг 2. Код программы, используемой для решения задания 2

#include <iostream>

#include <locale.h>

#include <conio.h>

#include <ctime>

#include <cmath>

#include <bitset>

#include <fstream>

#include <chrono>

using namespace std;

const int massive\_length = 10000000 / 8; //делим на 8 - количество чисел в одном элементе char,

//чтобы получить необходимую длину массива для сортировки целых чисел от 0 до 10^7

bool setNumber(int n, unsigned char\* data)

{

int k = n;

unsigned char mask = 1 << (8 - (k % 8) - 1);

if ((data[k / 8] & mask) != 0)

{

return false;

}

data[k / 8] = data[k / 8] | mask;

return true;

}

void printNumbers(unsigned char\* data)

{

int n = 0;

for (int i = 0; i < massive\_length; i++)

{

bitset<8> temp(data[i]);

for (int j = 7; j >= 0; j--)

{

if (temp[j] == 1)

{

cout << n << " ";

}

n++;

}

}

}

void insert(unsigned char\* byte\_massive, const char\* path)

{

fstream data\_source = fstream(path, ios::in);

if (!data\_source) return;

auto start = chrono::steady\_clock::now();

while(1)

{

int temp = -1;

if (!data\_source.eof())

{

Продолжение листинга 2

data\_source >> temp;

setNumber(temp, byte\_massive);

}

else

{

break;

}

}

auto end = chrono::steady\_clock::now();

data\_source.close();

auto duration = chrono::duration\_cast<chrono::microseconds> (end - start);

cout << "Время сортировки данных - " << duration.count() << " мкс" << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

unsigned char\* massive = new unsigned char[massive\_length];

for (int i = 0; i < massive\_length; i++)

{

massive[i] = 0;

}

char choose = ' ';

cout << "Выберите тип ввода: [M] - ручной, [A] - автоматический из файла: " << endl;

while (!((tolower(choose) == 'm') || (tolower(choose) == 'a')))

{

choose = \_getch();

}

int n = 0;

switch(tolower(choose))

{

case 'm':

cout << "Вводите не повторяющиеся целые числа в диапазоне [0; 9999999], для завершения введите -1:" << endl;

while (true)

{

cin >> n;

if (n == -1) break;

if (!setNumber(n, massive))

cout << "Это число уже вводилось. Введите другое или введите -1 для завершения." << endl;

}

break;

case 'a':

cout << "Введите путь к файлу с числами: " << endl;

string path;

getline(cin, path);

fstream ds = fstream(path, ios::in);

if (!ds) {

cout << "Файл не найден" << endl;

}

else

Продолжение листинга 2

{

cout << "Исходный массив: [ ";

int temp;

while (!ds.eof()) {

ds >> temp;

cout << temp << " ";

}

cout << "]" << endl;

insert(massive, path.c\_str());

}

}

cout << "Отсортированный массив: [ ";

printNumbers(massive);

cout << "]" << endl;

system("pause");

delete[] massive;

}

## **2.5 Тестирование**

Проведем тестирование написанной программы (рисунки 15-19):

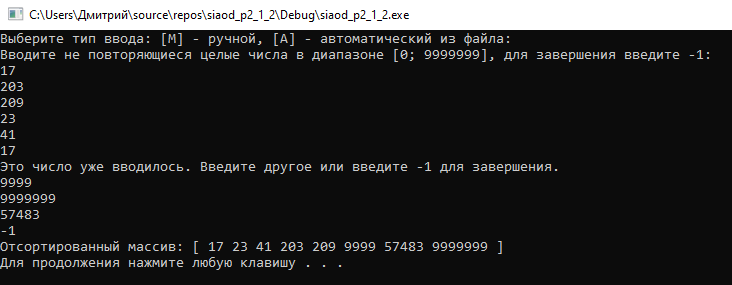


Рисунок 15 – Пример ручного ввода чисел

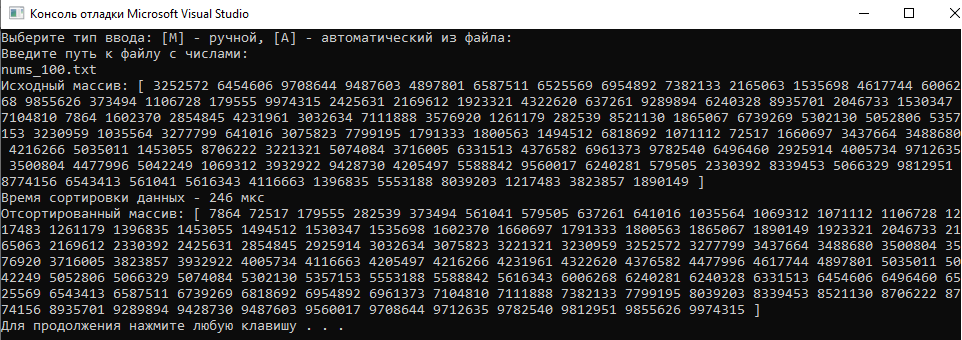


Рисунок 16 – Пример сортировки 100 чисел

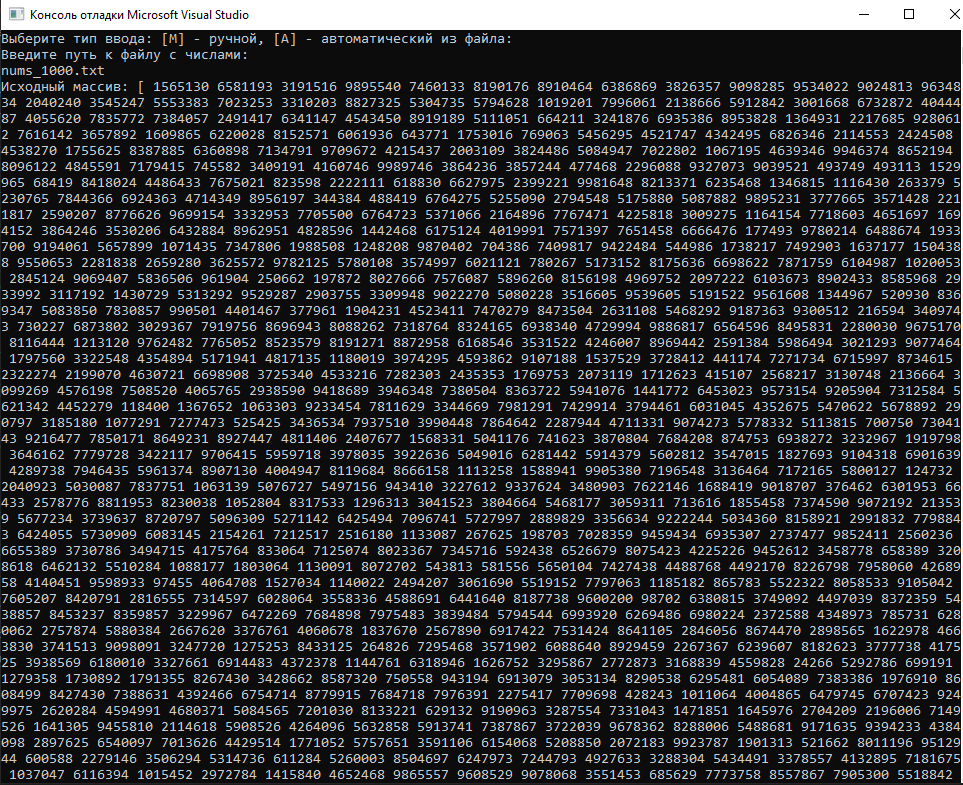


Рисунок 17 – Пример сортировки 1000 чисел, часть 1

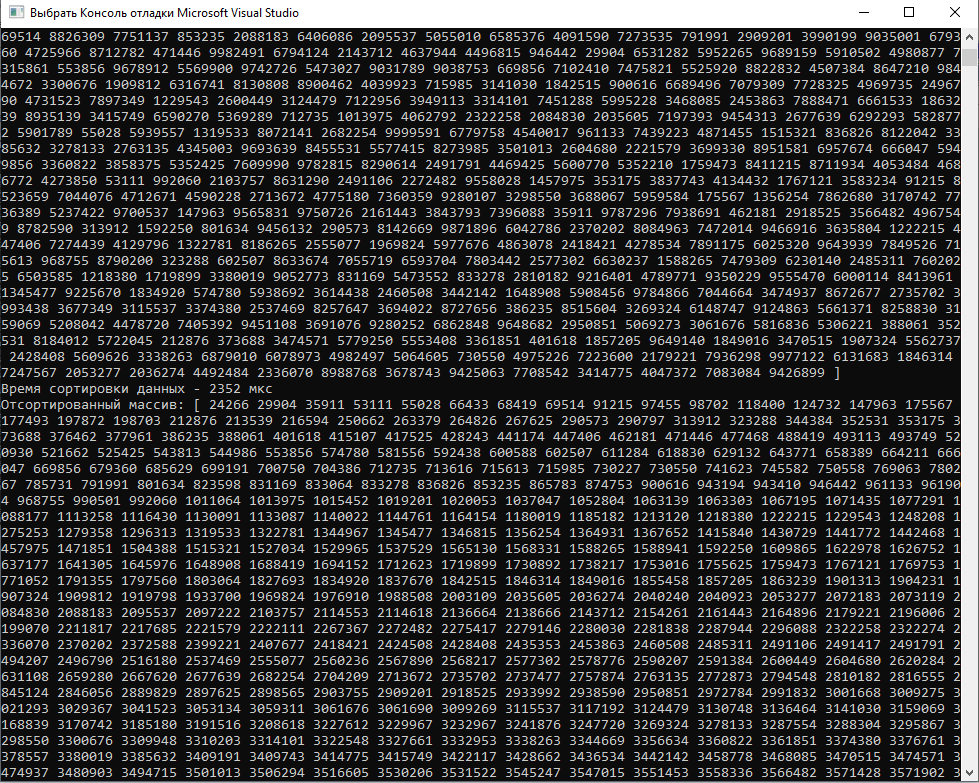


Рисунок 18 – Пример сортировки 1000 чисел, часть 2

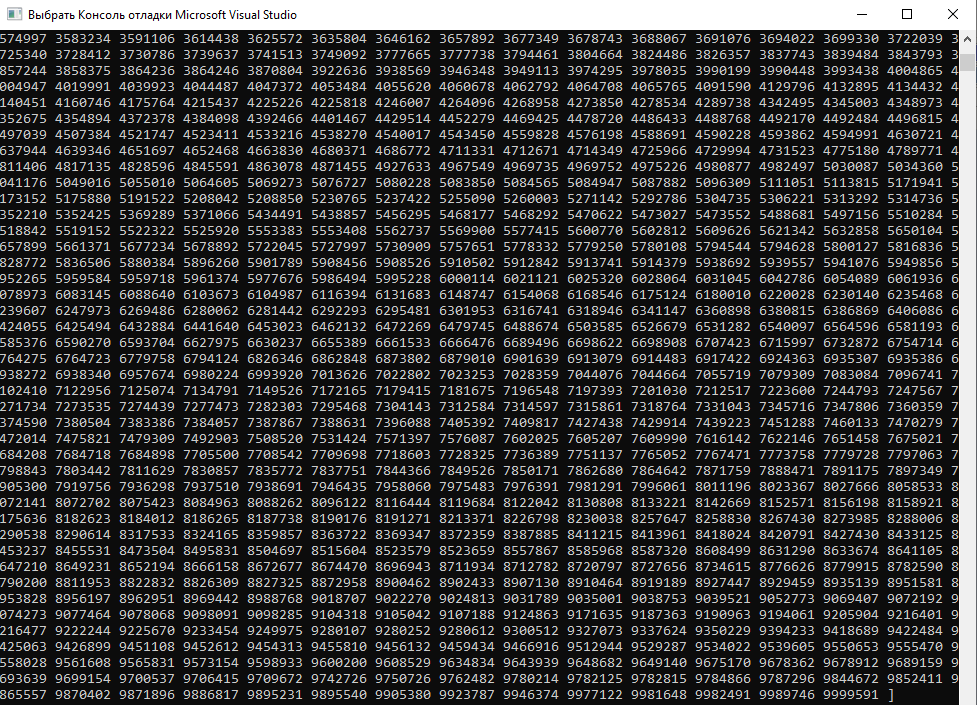


Рисунок 19 – Пример сортировки 1000 чисел, часть 3

По результатам тестирования программы сортировка с помощью битового массива выполнена корректно, значит, можно говорить о правильности составленных алгоритмов. На примере тестов с 100 и 1000 числами видно, что сложность сортировки с помощью битового массива равна O(n), n – число элементов.

# **3 ЗАКЛЮЧЕНИЕ**

В ходе практической работы были изучены битовые операции и способы их применения, в частности, сортировка чисел битовым массивом. Все написанные в рамках этой работы программы работают корректно, следовательно, можно говорить о правильности применения полученных знаний.

# **4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Методические указания к практической работе. URL: <https://online-edu.mirea.ru/pluginfile.php?file=%2F1126230%2Fmod_folder%2Fcontent%2F0%2F%D0%9F%D0%A0-2.1%20%28%D0%91%D0%B8%D1%82.%D0%BE%D0%BF%D0%B5%D1%80.%29.docx&forcedownload=1>, дата обращения: 13.09.23
2. Задания для самостоятельной работы №1. URL: <https://online-edu.mirea.ru/pluginfile.php?file=%2F1137386%2Fmod_assign%2Fintroattachment%2F0%2F%D0%A1%D0%B8%D0%90%D0%9E%D0%94%20%D0%A1%D0%B0%D0%BC%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%201%20%28%D0%BF%D0%BE%D1%80%D0%B0%D0%B7%D1%80%D1%8F%D0%B4%D0%BD%D1%8B%D0%B5%20%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8%29.pdf&amp;forcedownload=1>, дата обращения: 13.09.23