Содержание

[Введение 7](#_Toc139392803)

[1 Постановка задачи 9](#_Toc139392804)

[1.1 Достоинства алгоритма 9](#_Toc139392805)

[1.2 Недостатки алгоритма 9](#_Toc139392806)

[1.3 Типичные сценарии применения 10](#_Toc139392807)

[2 Выбор решения 11](#_Toc139392808)

[3 Описание программы 12](#_Toc139392809)

[4 Схема программы 16](#_Toc139392810)

[5 Тестирование программы 17](#_Toc139392811)

[5.1 Тестирование на разных наборах данных 17](#_Toc139392812)

[5.2 Анализ полученных результатов тестирования 17](#_Toc139392813)

[6 Отладка 19](#_Toc139392814)

[7 Совместная разработка 21](#_Toc139392815)

[Заключение 22](#_Toc139392816)

[Список используемой литературы 23](#_Toc139392817)

[Приложение А 24](#_Toc139392818)

[Приложение Б Листинг 28](#_Toc139392819)

# **Введение**

В настоящее время сортировка данных является одним из наиболее распространенных процессов обработки информации при современном развитии компьютерных технологий. Задачи на сортировку данных возникают в различных профессиональных сферах деятельности и требуют эффективных алгоритмов.

Алгоритмы сортировки образуют отдельный класс алгоритмов и используются практически во всех задачах обработки информации. Они настолько тесно связаны друг с другом, что образуют отдельный класс алгоритмов. Алгоритмы сортировки обычно применяются для упорядочивания данных с целью более эффективного последующего поиска. Например, в словарях слова упорядочены по алфавиту для удобства использования.

Важность сортировки заключается в том, что на ее примере можно показать многие основные фундаментальные приемы и методы построения алгоритмов. Сортировка является примером огромного разнообразия алгоритмов, которые выполняют одну и ту же задачу, и многие из них имеют определенные преимущества перед другими. Усложнение алгоритма может дать значительное увеличение эффективности и быстродействия по сравнению с более простыми методами. Термин "сортировка" обычно понимается как процесс перестановки объектов некоторого множества в определенном порядке.

Сортировка вставками — это простой и эффективный алгоритм, используемый для сортировки заданного списка элементов в порядке возрастания или убывания. Он подпадает под категорию алгоритмов сортировки на основе сравнения.

Основная идея сортировки вставками состоит в том, чтобы разделить список на два подсписка: отсортированный подсписок и несортированный подсписок. Элементы входной последовательности рассматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.[[1]](file:///C:\Users\Денис\Downloads\gavin_22vvp2.docx#2jxsxqh)

Алгоритм сортировки получает на вход последовательность из n чисел: a1, a2, ..., an, которые также называются ключами. Эти числа представлены в виде массива с n элементами. После сортировки алгоритм должен вернуть перестановку исходной последовательности a1', a2', ..., an', где выполняется следующее условие: a1' <= a2' <= ... <= an'.[[2]](file:///C:\Users\Денис\Downloads\gavin_22vvp2.docx#z337ya)

Алгоритм сортировки вставками начинает работу с пустой отсортированной последовательности. На каждом шаге алгоритма выбирается один элемент из входных данных и помещается на свою позицию в отсортированной последовательности. Это продолжается до тех пор, пока все элементы входных данных не будут отсортированы.

Для сортировки вставками не требуется использования дополнительной памяти, кроме постоянной величины для хранения одного элемента, так как перестановка выполняется в пределах входного массива. В результате работы алгоритма входной массив преобразуется в требуемую выходную последовательность элементов. [[3]](file:///C:\Users\Денис\Downloads\gavin_22vvp2.docx#3j2qqm3)

# **1 Постановка задачи**

Поставленная задача: необходимо заполнить массив из n-ого количества элементов псевдослучайными числами, если у пользователя нет файла, а также записать данные элементы в отдельный файл, иначе заполнить массив числами из пользовательского файла. После этого выполнить сортировку вставками над данными, находящимися в массиве, записать отсортированные данные в другой файл, посчитать время выполнения сортировки. Использовать сервис GitHub для совместной работы. Оформить отчет по проделанной практике.

## **1.1 Достоинства алгоритма**

а) Эффективность для небольших наборов данных;

б) Простота алгоритма;

в) Сортировка на месте: сортировка вставками работает с входным массивом, не требуя дополнительного места в памяти;

г) Онлайн-сортировка. Сортировка вставками хорошо подходит для сценариев онлайн-сортировки, когда элементы постоянно добавляются в массив в режиме реального времени;

д) Простота реализации на разных языках программирования.

## **1.2 Недостатки алгоритма**

а) Сортировка вставками имеет временную сложность O(n^2), где n представляет количество элементов в списке. В худшем случае алгоритму необходимо сравнить каждый элемент в не отсортированном подмножестве со всеми элементами в отсортированном;

б) Не подходит для распараллеливания. Природа сортировки вставками, когда элементы сравниваются и вставляются по одному, затрудняет распараллеливание процесса сортировки;

в) Отсутствие адаптивности: сортировка вставками не использует преимущества любого существующего порядка в наборе данных.

## **1.3 Типичные сценарии применения**

В целом, сортировка вставками лучше всего подходит для небольших наборов данных, частично отсортированных данных, сценариев онлайн-сортировки, сортировки связанных списков и в образовательных целях.

# **2 Выбор решения**

В рамках разработки программы будет использована среда Microsoft Visual Studio 2022 и язык программирования C.

Microsoft Visual Studio 2022, последняя версия популярной интегрированной среды разработки (IDE), представляет собой значительный шаг вперед в средствах разработки программного обеспечения.

Одним из важных аспектов Visual Studio 2022 является расширенная поддержка кроссплатформенной разработки. Эта универсальность позволяет разработчикам охватить более широкую пользовательскую базу и максимизировать потенциальное влияние своих программных решений.

Что касается производительности, Visual Studio 2022 использует множество оптимизаций для улучшения общего опыта разработки. Время запуска IDE значительно сократилось, что позволяет разработчикам быстро приступить к написанию кода без лишних задержек.

Visual Studio 2022 также представляет 64-разрядную версию IDE, которая представляет собой существенное улучшение по сравнению с 32-разрядной предшественницей. Используя преимущества 64-разрядных вычислений, такие как увеличенный объем памяти и повышенная производительность, разработчики могут с легкостью выполнять более сложные проекты и работать с большими кодовыми базами.

Кроме того, Microsoft приложила значительные усилия для расширения возможностей отладки Visual Studio 2022. Обновленный отладчик обеспечивает повышенную надежность, улучшенную диагностику и расширенную поддержку различных языков программирования. Эти усовершенствования позволяют разработчикам быстро выявлять и устранять проблемы, обеспечивая поставку высококачественных программных решений.

В результате разработчики могут использовать Visual Studio 2022 для создания передовых приложений, отвечающих требованиям современного программного обеспечения.

Язык программирования C является универсальным и широко используемым языком, который широко используется в различных областях компьютерных наук и разработки программного обеспечения.

Одной из ключевых характеристик C является его тесная связь с лежащей в его основе аппаратной архитектурой. C обеспечивает прямой доступ к памяти, позволяя программистам писать эффективный и оптимизированный код, который может в полной мере использовать аппаратные ресурсы. Этот низкоуровневый элемент управления обеспечивает точную настройку производительности, что делает язык C идеальным выбором для сред с ограниченными ресурсами.

Еще одной примечательной особенностью языка C является мощная поддержка модульного программирования. Язык предоставляет механизмы, такие как функции и структуры, которые упрощают организацию кода и его повторное использование. Благодаря использованию библиотек и файлов заголовков разработчики могут создавать повторно используемые компоненты, продвигая модульность кода и облегчая сотрудничество между программистами.

Портативность — еще одна сильная сторона языка программирования Си. Компиляторы C доступны для широкого спектра платформ, включая различные операционные системы и аппаратные архитектуры. Эта межплатформенная совместимость позволяет разработчикам писать код, который можно легко портировать и запускать в разных системах без существенных модификаций.

Кроме того, C повлиял на развитие многих других языков программирования, включая C++, Objective-C и Java.

В заключение, язык программирования C предлагает универсальный и мощный набор инструментов для разработки эффективного и переносимого программного обеспечения. [4]

# **3 Описание программы**

Первая функция print\_name() отвечает за вывод названия программы в виде таблицы на экран. Эта функция не принимает аргументов и не возвращает значений. Она использует функцию printf() для вывода символьных строк, которые формируют таблицу с названием программы.

// Функция, выводящая таблицу с названием программы

void print\_name()

{

printf("\t\t\t\t\t\t +-------------------------+\n");

printf("\t\t\t\t\t\t | Сортировка вставкой |\n");

printf("\t\t\t\t\t\t +-------------------------+\n");

printf("\n\n\n\n");

}

Функция last\_table() отвечает за вывод результата сортировки и статистических данных на экран в виде таблицы. Эта функция принимает четыре аргумента: size\_mas - размер множества, max\_num - максимальное число в множестве, min\_num - минимальное число в множестве и time\_end - время выполнения сортировки. Функция использует функцию printf() для форматированного вывода символьных строк, которые представляют таблицу с результатами сортировки и статистическими данными. Функция также использует константу CLOCKS\_PER\_SEC для преобразования времени выполнения сортировки в секунды.

Обе эти функции используют функцию printf() для форматированного вывода символьных строк на экран в виде таблицы. Они повышают читабельность и эстетический вид программы, делая ее более понятной и удобной для пользователя.

void last\_table(int size\_mas, int max\_num, int min\_num, int time\_end)

{

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Размер множества | %14d |\n", size\_mas);

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Минимальное число | %13d |\n", min\_num);

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Максимальное число | %13d |\n", max\_num);

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Время сортировки | %14lf(c) |\n", (double)time\_end / CLOCKS\_PER\_SEC);

printf("\t\t\t\t +------------------------+------------------------+\n");

}

Затем идет текст главной программы. Выполняется инициализация

генератора случайных чисел rand, затем русификация и вывод таблицы с названием программы:

// Подключение случайных чисел

srand(time(NULL));

// Подключение русского языка

setlocale(LC\_ALL, "RUS");

// Вывод первой таблицы

print\_name();

// Ввод размера множества

printf("Размер множества: ");

scanf("%d", &size\_mas);

printf("\n");

// Ввод нижнего предела множества

printf("Минимальное число: ");

scanf("%d", &min\_num);

printf("\n");

// Ввод верхнего предела множества

printf("Максимальное число: ");

scanf("%d", &max\_num);

Далее происходит очистка консоли, для вывода таблицы с названием

программы и таблицы со следующими данными: размер множества, нижний

и верхний предел множества, время сортировки.

// Очистка консоли

system("cls");

// Вывод первой таблицы

print\_name();

// Вывод второй таблицы

last\_table(size\_mas, max\_num, min\_num, time\_end);

После происходит очистка консоли, для вывода таблицы с названием

программы и таблицы со следующими данными: размер множества, нижний

и верхний предел множества, время сортировки.

// Очистка консоли

system("cls");

// Вывод первой таблицы

print\_name();

Также будет выведена таблица со временем потраченным на сортировку

// Вывод времени потраченного на сортировку

printf("\t\t\t\t +------------------------+----------------

--------+\n");

printf("\t\t\t\t | Время сортировки | %14lf(c)

|\n", (double)time\_end / CLOCKS\_PER\_SEC);

printf("\t\t\t\t +------------------------+----------------

--------+\n");

Для проведения экспериментов и оценки производительности алгоритма, завел переменные начала и конца таймера и использовал сам таймер clock\_t, который позволяет измерить время выполнения определенных операций в тактах процессора. Для этого я определил переменную time\_start и присвоил ей значение системного времени с помощью функции clock(). Затем, после выполнения необходимых операций, я вычислил значение переменной time\_end как разность между текущим временем и временем начала отсчета, сохраненным в переменной time\_start.

// Начало отсчета времени

clock\_t time\_start = clock();

// Конец отсчета времени

clock\_t time\_end = clock() - time\_start;

# **4 Схема программы**



Рисунок 4.1 - Схема вывода таблицы Рисунок 4.2 - Схема вывода результата

# **5 Тестирование программы**

## **5.1 Тестирование на разных наборах данных**

Тестовый набор данных представлен в таблице 1. Результаты тестирования приведены в Приложении А на рисунках А.1 - А.11.

Таблица 1 – Тестовый набор данных

|  |  |  |
| --- | --- | --- |
| № теста | Размер массива size | Время выполнения сортировки в секундах |
| 1 | 10000 | 0.035 |
| 2 | 20000 | 0.083 |
| 3 | 30000 | 0.164 |
| 4 | 40000 | 0.272 |
| 5 | 50000 | 0.45 |
| 6 | 60000 | 0.617 |
| 7 | 70000 | 0.866 |
| 8 | 80000 | 1.014 |
| 9 | 90000 | 1.307 |
| 10 | 100000 | 1.607 |
| 11 | 110000 | 1.894 |

## **5.2 Анализ полученных результатов тестирования**

Анализ данных, полученных при тестировании алгоритма сортировки вставками на рисунке 5, позволяет сделать следующий вывод: время работы программы прямо пропорционально количеству элементов. Результаты тестирования показывают, что алгоритм сортировки демонстрирует линейную сложность. Это означает, что время работы программы в значительной степени зависит от размера входных данных. Чем больше входных элементов, тем дольше будет выполняться сортировка.

Также стоит отметить, что данный алгоритм не обладает лучшей производительностью в сравнении с другими алгоритмами сортировки, но может быть полезен при работе с уже частично отсортированными данными. В таких случаях он может показать себя быстрее в сравнении с другими алгоритмами. Однако, при работе с неупорядоченными данными или большими объемами данных, рекомендуется использовать другие методы сортировки.

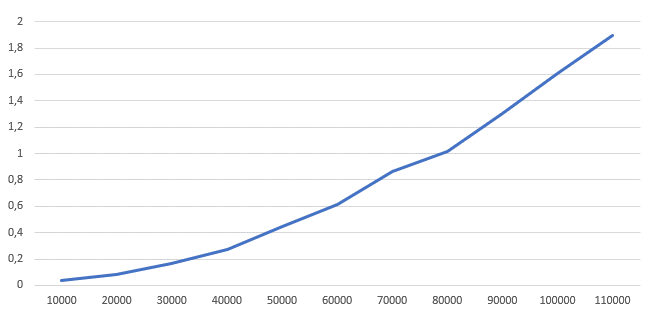


Рисунок 5 – Результаты тестирования

# **6 Отладка**

Отладка программы является важным этапом разработки и позволяет выявить и исправить ошибки в коде. В данном случае, отладка программы будет проводиться в трех фрагментах кода.

Для более детальной отладки программы можно использовать такие методы, как пошаговое выполнение кода, установка точек останова на нужных строках и просмотр значений переменных во время выполнения программы

Таким образом, проведение отладки программы на каждом этапе позволяет выявить и исправить ошибки.

Рисунок 6.1 отображает этап отладки кода перед запуском алгоритма. Проанализировав этот процесс, можно убедиться в правильном выполнении инициализации переменных, генерации чисел и их записи в файл.

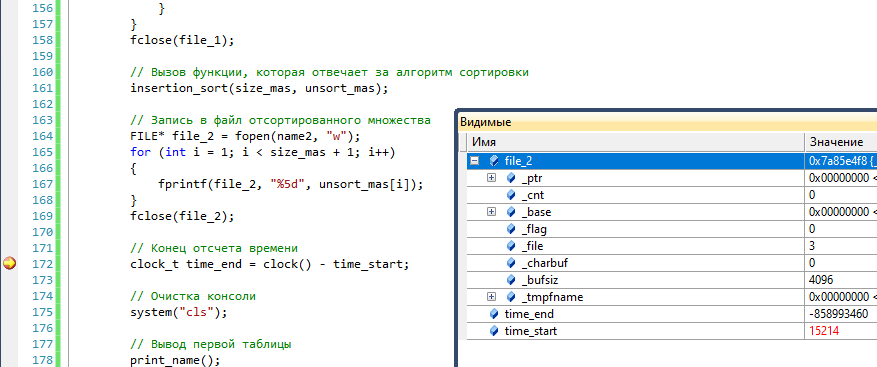


Рисунок 6.1 – Отладка до начала алгоритма

На рисунке 6.2 представлена точка остановки в алгоритме сортировки чисел из файла с использованием метода сортировки вставками. Эта точка останова указывает на успешное выполнение сортировки до данного момента, где алгоритм правильно размещает числа в отсортированной последовательности, вставляя каждое число на свое место в соответствии с правилами сортировки вставками.

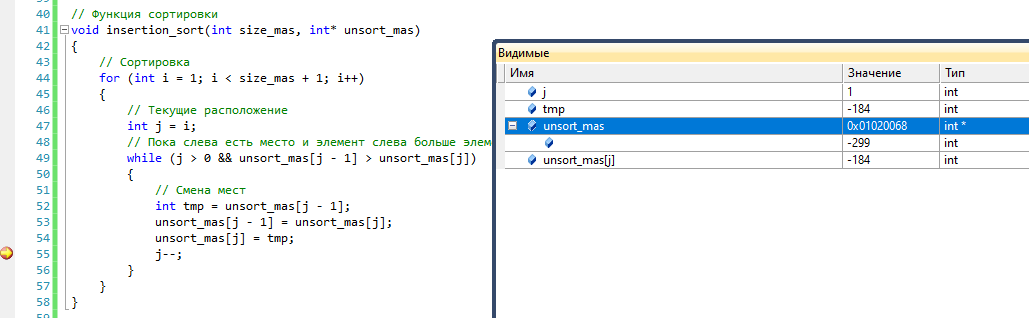


Рисунок 6.2 – Отладка алгоритма сортировки

На рисунке 6.3 изображен конец программы после успешного завершения выполнения алгоритма сортировки чисел методом сортировки вставками.

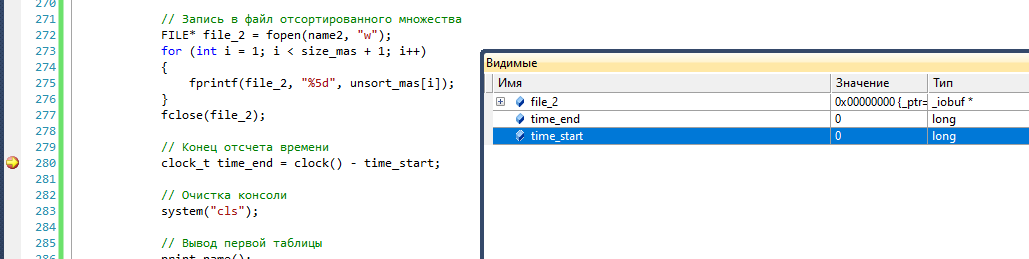


Рисунок 6.3 – Отладка в конце программы



Рисунок 6.4 - Структура не отсортированного файла



Рисунок 6.5 - Структура отсортированного файла

# **7 Совместная разработка**

При разработке программы использовалась среда GitHub для обменивания промежуточными данными. На рисунке 7 показана схема, которая получилась при совместной разработке.

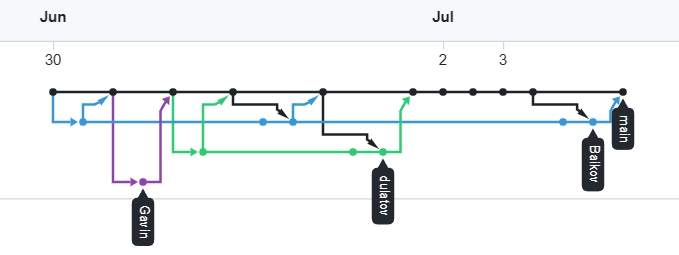


Рисунок 7 - Схема совместной разработки

# **Заключение**

В ходе выполнения практической работы удалось создать программу, использующую алгоритм сортировки вставками для эффективного упорядочения множества неопределенного размера. Это достижение стало возможным благодаря командной работе, чему способствовало использование платформы GitHub.

# **Список используемой литературы**

1. Microsoft Developer Blogs : https://devblogs.microsoft.com/visualstudio/visual-studio-2022/
2. Кнут Д. Э. 5.2 Внутренняя сортировка // Искусство программирования. Том 3 Сортировка и поиск, 2007. — Т. 3. — 832 с.
3. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. 2.1. Сортировка вставкой // Алгоритмы: построение и анализ, 2013. — С. 38—45.
4. Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д. Структуры данных и алгоритмы, 2000. — С. 231.

# **Приложение А**

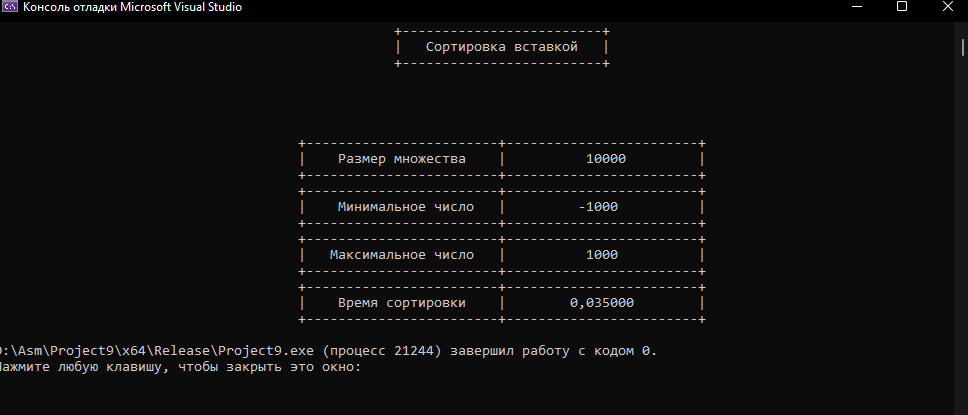


Рисунок А.1

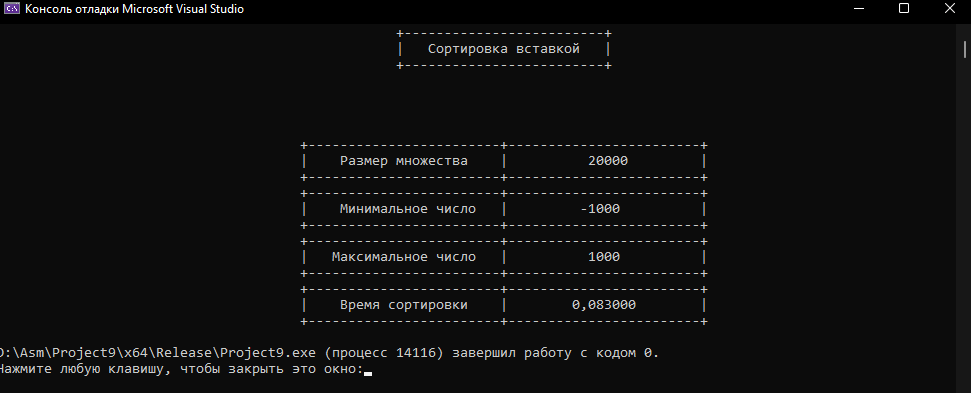


Рисунок А.2

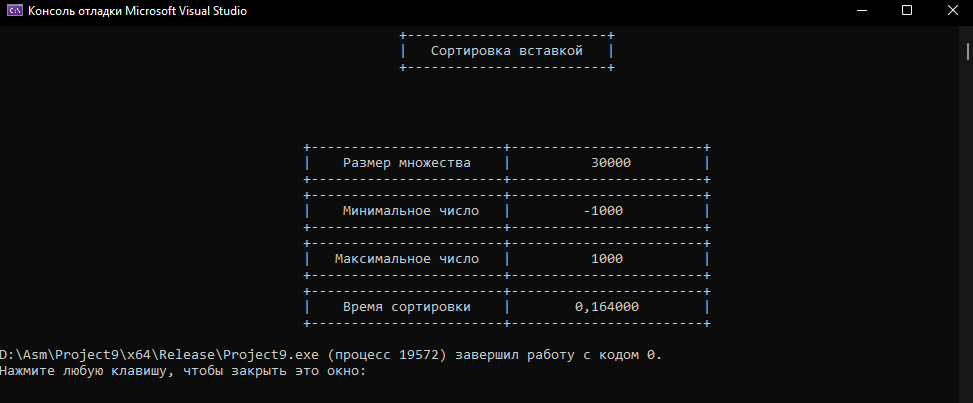


Рисунок А.3

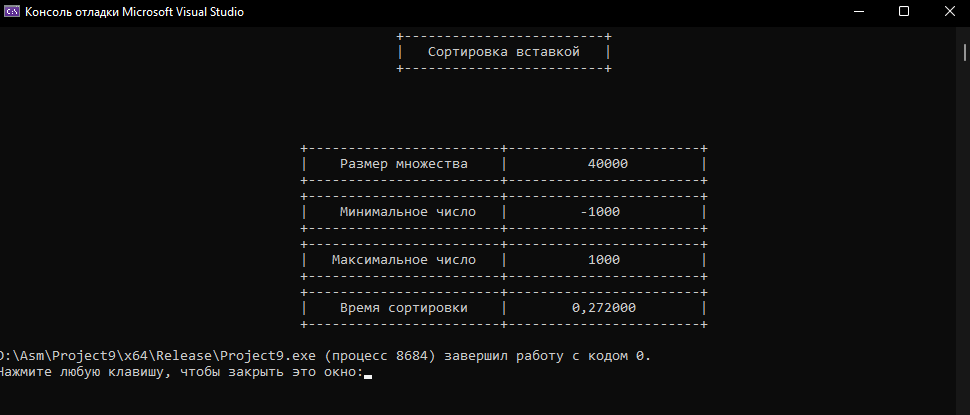


Рисунок А.4

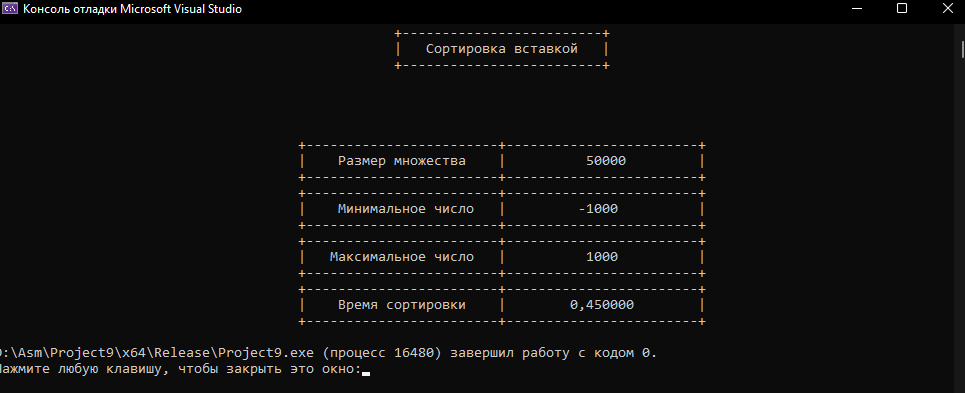


Рисунок А.5

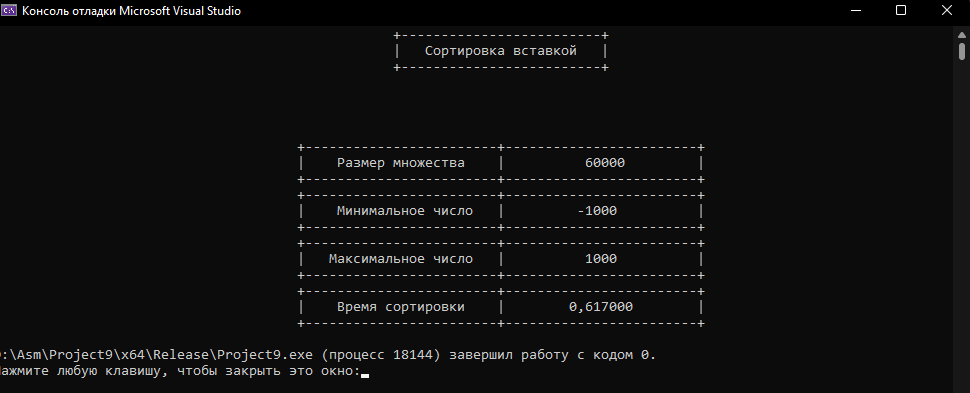


Рисунок А.6

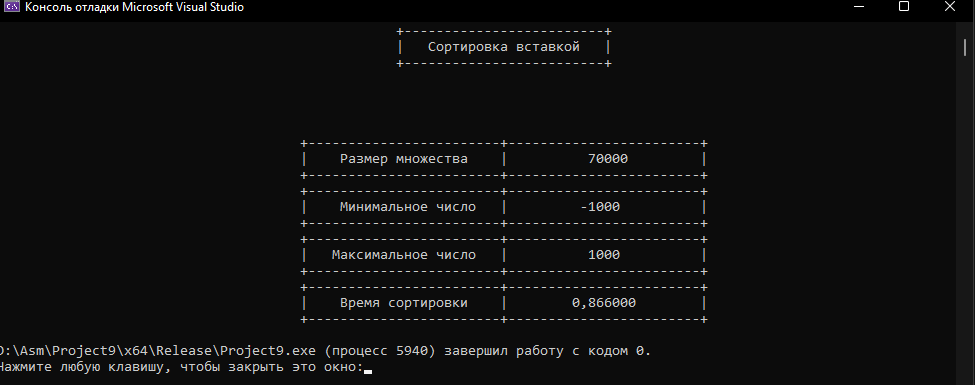


Рисунок А.7

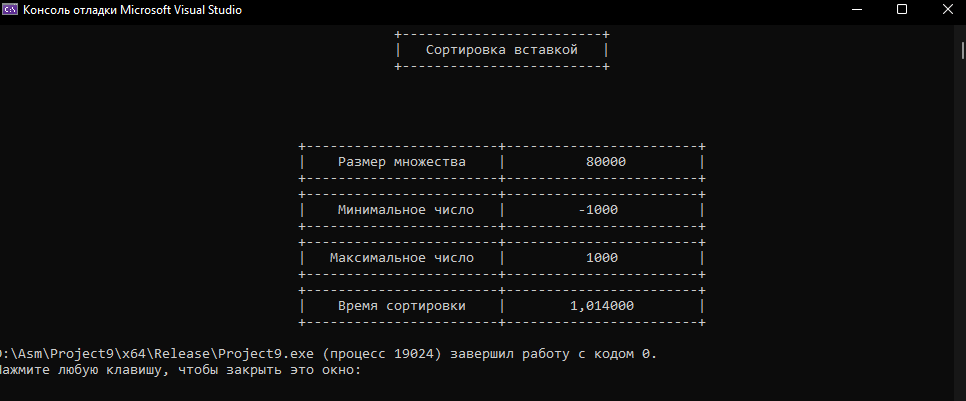


Рисунок А.8

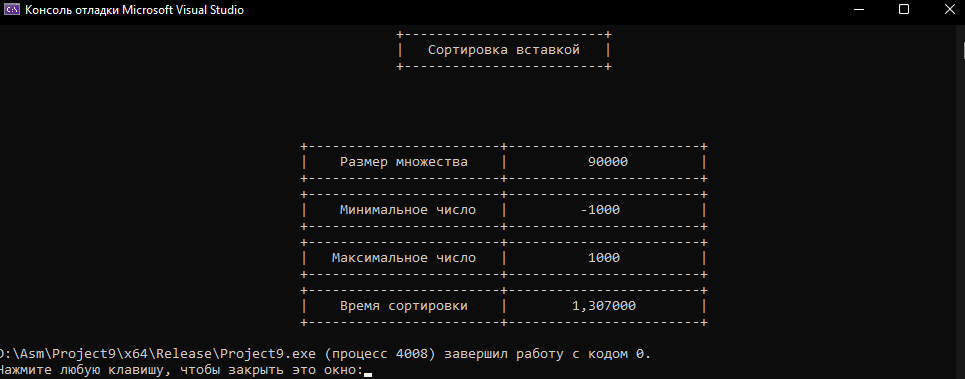


Рисунок А.9

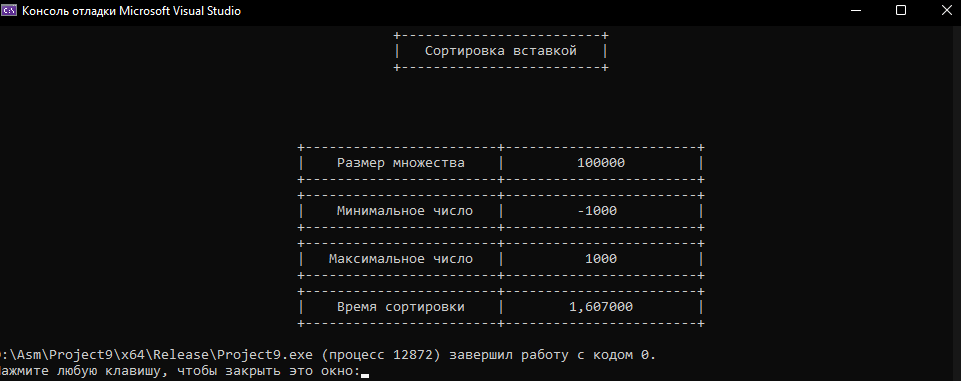


Рисунок А.10

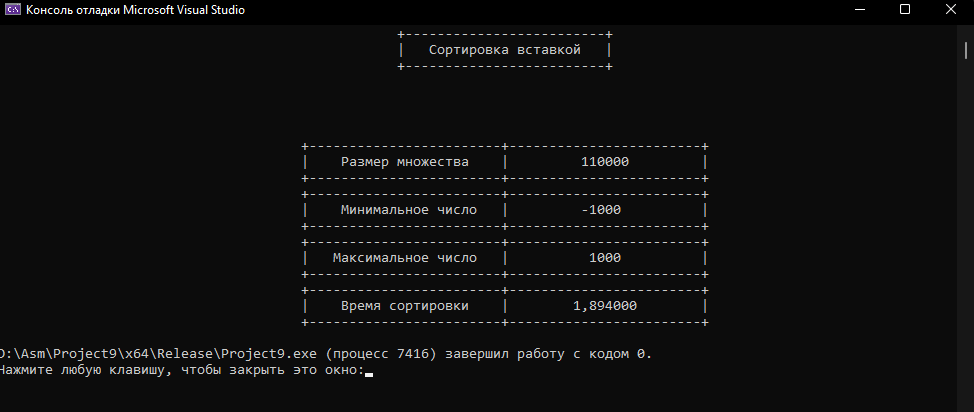


Рисунок А.11

# **Приложение Б Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <locale.h>

#include <malloc.h>

#include <stdlib.h>

// Функция, выводящая таблицу с названием программы

void print\_name()

{

printf("\t\t\t\t\t\t +-------------------------+\n");

printf("\t\t\t\t\t\t | Сортировка вставкой |\n");

printf("\t\t\t\t\t\t +-------------------------+\n");

printf("\n\n\n\n");

}

// Функция, выводящая таблицу с введенными данными пользователя и итоговое время сортировки

void last\_table(int size\_mas, int max\_num, int min\_num, int time\_end)

{

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Размер множества | %14d |\n", size\_mas);

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Минимальное число | %13d |\n", min\_num);

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Максимальное число | %13d |\n", max\_num);

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Время сортировки | %14lf(c) |\n", (double)time\_end / CLOCKS\_PER\_SEC);

printf("\t\t\t\t +------------------------+------------------------+\n");

}

// Функция сортировки

void insertion\_sort(int size\_mas, int\* unsort\_mas)

{

// Сортировка

for (int i = 1; i < size\_mas + 1; i++)

{

// Текущие расположение

int j = i;

// Пока слева есть место и элемент слева больше элемента справа

while (j > 0 && unsort\_mas[j - 1] > unsort\_mas[j])

{

// Смена мест

int tmp = unsort\_mas[j - 1];

unsort\_mas[j - 1] = unsort\_mas[j];

unsort\_mas[j] = tmp;

j--;

}

}

}

// Основная функция

int main()

{

// Подключение случайных чисел

srand(time(NULL));

// Подключение русского языка

setlocale(LC\_ALL, "RUS");

// Вывод первой таблицы

print\_name();

int size\_mas; // Размер множества

int\* unsort\_mas; // Указатель для расширение памяти

int max\_num; // Минимальное число

int min\_num; // Максимальное число

char have\_file; // Для проверки на наличие пользавотельского файла

char buf; // Буфер

char name1[20]; // Имя файла не отсортированного множества

char name2[20]; // Имя файла отсортированного множества

// Подтверждение или отрицание наличия файла

printf("Есть файл для сортировки? (Для подтверждения введите Y/y, если нет, то введите N/n, программа будет работать со случайными числами): ");

scanf("%c", &have\_file);

printf("\n");

// Если введено не "Y\yes", то заполнением файла псевдослучайными числами

if (have\_file != 'Y' && have\_file != 'y')

{

// Имя файла для не отсортированного множества

printf("В каком файле сохранить не отсортированное множество?: ");

scanf("%s", &name1);

printf("\n");

// Имя файла для отсортированного множества

printf("В каком файле сохранить отсортированное множество?: ");

scanf("%s", &name2);

printf("\n");

// Ввод размера множества

printf("Размер множества: ");

scanf("%d", &size\_mas);

printf("\n");

// Ввод нижнего предела множества

printf("Минимальное число: ");

scanf("%d", &min\_num);

printf("\n");

// Ввод верхнего предела множества

printf("Максимальное число: ");

scanf("%d", &max\_num);

printf("\n");

// Расширение памяти

unsort\_mas = (int\*)malloc(size\_mas \* sizeof(int));

// Заполнение файла не отсортированным множеством

FILE\* file\_1 = fopen(name1, "w");

for (int i = 0; i < size\_mas; i++)

{

if (min\_num >= 0)

{

// Если минимальное число положительное

fprintf(file\_1, "%5d", unsort\_mas[i] = rand() % max\_num);

}

else

{

// Если минимальное число отрицательное

fprintf(file\_1, "%5d", unsort\_mas[i] = min\_num + rand() % max\_num);

}

}

fclose(file\_1);

// Начало отсчета времени работы сортировки

clock\_t time\_start = clock();

// Чтение из файла для сортировки

file\_1 = fopen(name1, "r");

// До конца файла

while ((buf = fgetc(file\_1)) != EOF)

{

for (int i = 0; i < size\_mas + 1; i++) {

if (i == ' ') {

// Без ' '

continue;

}

else {

fscanf(file\_1, "%5d", &unsort\_mas[i]);

}

}

}

fclose(file\_1);

// Вызов функции, которая отвечает за алгоритм сортировки

insertion\_sort(size\_mas, unsort\_mas);

// Запись в файл отсортированного множества

FILE\* file\_2 = fopen(name2, "w");

for (int i = 1; i < size\_mas + 1; i++)

{

fprintf(file\_2, "%5d", unsort\_mas[i]);

}

fclose(file\_2);

// Конец отсчета времени

clock\_t time\_end = clock() - time\_start;

// Очистка консоли

system("cls");

// Вывод первой таблицы

print\_name();

// Вывод второй таблицы

last\_table(size\_mas, max\_num, min\_num, time\_end);

}

// Второй сценарий, если у пользователя есть файл для сортировки

else

{

// Ввод имени пользовательского файла для сортировки

printf("Какой файл необходимо отсортировать?: ");

scanf("%s", &name1);

printf("\n");

// Проверка на наличие файла

FILE\* file\_1 = fopen(name1, "r");

while (!file\_1)

{

char next[4]; // Выход из программы

printf("Ошибка: файл с таким именем не найден!");

printf("\n\n");

// Если пользователь захочет прекратить попытки ввода корректного имени файла

printf("Продолжить? (Введите \"Yes/yes\", если хотите продолжить): ");

scanf("%s", &next);

printf("\n");

// Если пользователь введет не "Yes\yes", то выход из программы

if (next[0] != 'Y' && next[0] != 'y')

{

return 1;

}

// Если после неудачной попытки, введет корректное имя файла

else

{

printf("Какой файл необходимо отсортировать?: ");

scanf("%s", &name1);

printf("\n");

file\_1 = fopen(name1, "r");

}

fclose(file\_1);

}

fclose(file\_1);

// Сохранение длины файла

file\_1 = fopen(name1, "r");

fseek(file\_1, 0L, SEEK\_END);

size\_mas = ftell(file\_1);

fclose(file\_1);

// если файл пуст

if (size\_mas == 0)

{

printf("Ошибка: Файл пуст!");

printf("\n");

return 1;

}

else

{

// Ввод имени файла для сохранения отсортированного множества

printf("В каком файле сохранить отсортированное множество?: ");

scanf("%s", &name2);

// Расширение памяти

unsort\_mas = (int\*)malloc(size\_mas \* sizeof(int));

// Начало отсчета времени работы сортировки

clock\_t time\_start = clock();

// Чтение из файла для сортировки

file\_1 = fopen(name1, "r");

// До конца файла

while ((buf = fgetc(file\_1)) != EOF)

{

for (int i = 0; i < size\_mas + 1; i++) {

if (i == ' ') {

// Без ' '

continue;

}

else {

fscanf(file\_1, "%d", &unsort\_mas[i]);

}

}

}

fclose(file\_1);

// Вызов функции отвечающий за сортировку

insertion\_sort(size\_mas, unsort\_mas);

// Запись в файл отсортированного множества

FILE\* file\_2 = fopen(name2, "w");

for (int i = 1; i < size\_mas + 1; i++)

{

fprintf(file\_2, "%5d", unsort\_mas[i]);

}

fclose(file\_2);

// Конец отсчета времени

clock\_t time\_end = clock() - time\_start;

// Очистка консоли

system("cls");

// Вывод первой таблицы

print\_name();

// Вывод времени потраченного на сортировку

printf("\t\t\t\t +------------------------+------------------------+\n");

printf("\t\t\t\t | Время сортировки | %14lf(c) |\n", (double)time\_end / CLOCKS\_PER\_SEC);

printf("\t\t\t\t +------------------------+------------------------+\n");

}

}

// Освобождение памяти

free(unsort\_mas);

return 0;

}