Оглавление

[1. Введение 2](#_Toc138949249)

[2. Постановка задачи 5](#_Toc138949250)

[3. Выбор решения 8](#_Toc138949251)

[4. Описание программы 9](#_Toc138949252)

[5. Схемы программы 13](#_Toc138949253)

[6. Тестирование программы 15](#_Toc138949254)

[7. Отладка 16](#_Toc138949255)

[8. Совместная разработка 17](#_Toc138949256)

[9. Заключение 18](#_Toc138949257)

[Список используемой литературы 19](#_Toc138949258)

[Приложение А 20](#_Toc138949259)

[Приложение Б “Листинг” 22](#_Toc138949260)

# 1. Введение

Microsoft Visual Studio — это программная среда по разработке приложений для ОС Windows, как консольных, так и с графическим интерфейсом.

Функциональная структура среды включает в себя:

1) редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего [рефакторинга кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3);

2) отладчик кода;

3) редактор форм, предназначенный для упрощённого конструирования графических интерфейсов;

4) веб-редактор;

5) дизайнер классов;

6) дизайнер схем баз данных.

Visual Studio также позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода, добавление новых наборов инструментов (для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования или инструментов для прочих аспектов процесса разработки программного обеспечения).

В комплект входят следующие основные компоненты:

1. Visual Basic.NET – для разработки приложений на VisualBasic;
2. Visual C++ – на традиционном языке C++;
3. Visual C# – на языке C# (Microsoft);
4. Visual F# – на F# (Microsoft Developer Division).

Интегрированная среда разработки (IntegratedDevelopmentEnvironment – IDE) Visual Studio предлагает ряд высокоуровневых функциональных возможностей, которые выходят за рамки базового управления кодом.

Ниже перечислены основные преимущества IDE-среды Visual Studio:

1. Встроенный Web-сервер;
2. Поддержка множества языков при разработке;
3. Интуитивный стиль кодирования;
4. Более высокая скорость разработки;
5. Возможности отладки.

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

Своими корнями он уходит в язык Си, который был разработан в 1969—1973 годах в компании Bell Labs программистом Деннисом Ритчи. В начале 1980-х годов датский программист Бьерн Страуструп, который в то время работал в компании Bell Labs, разработал С++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

С++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко С++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на С++. Но только системным программированием применение данного языка не ограничивается. С++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ. Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход мобильное направление, где С++ тоже нашел свое применение. И даже в веб-разработке также можно использовать С++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем С++ – язык широкого пользования, на котором можно создавать практически любые виды программ.

# 2. Постановка задачи

По программе: Необходимо заполнить массив из n-ого количества элементов случайными числами, записать данные элементы в отдельный файл. После этого выполнить шейкерную сортировку над данными, находящимися в массиве, записать отсортированные данные в другой файл.

Использовать сервис GitHub для совместной работы. Создать ветки и выложить коммиты, характеризующие действия, выполненные участником бригады.

Оформить отчет по проведенной практике.

Достоинства данного алгоритма сортировки:

* Простота: Сортировка шейкер относительно проста для понимания и реализации по сравнению с более сложными алгоритмами сортировки. Она не требует сложных структур данных или рекурсивных операций, что делает ее доступной для программистов с базовыми знаниями о сортировке;
* Сортировка на месте: Сортировка "на месте" работает непосредственно на исходном списке, не требуя дополнительного пространства в памяти. Она выполняет обмен между соседними элементами, сводя к минимуму необходимость в дополнительных структурах данных и сокращая использование памяти;
* Стабильность: Сортировка Шейкера является стабильным алгоритмом сортировки, что означает, что он сохраняет относительный порядок одинаковых элементов. Если в списке есть дублирующиеся элементы, они останутся в том же порядке после сортировки;
* Адаптивное поведение: Алгоритм сортировки шейкер обладает способностью адаптироваться к частично отсортированным или почти отсортированным спискам. В таких случаях он может значительно сократить количество проходов, необходимых для сортировки списка, что делает его эффективным в определенных сценариях;
* Минимальная дополнительная сложность: Алгоритм сортировки шейкером является вариацией алгоритма сортировки пузырьком, только с небольшими изменениями в направлении перестановки. Поэтому дополнительная сложность, вносимая шейкерной сортировкой, минимальна по сравнению с алгоритмом пузырьковой сортировки.

Недостатки данного алгоритма сортировки:

* Неэффективность: Шейкерная сортировка имеет временную сложность O(n^2), где n - количество элементов в списке. Это делает ее неэффективной для больших списков, так как количество сравнений и обменов растет экспоненциально с размером входных данных;
* Отсутствие оптимальности: Шейкерная сортировка не является оптимальным алгоритмом сортировки. Он может потребовать большого количества проходов по списку, даже если список уже отсортирован. В худшем случае, когда список сортируется в обратном порядке, шейкерная сортировка работает аналогично алгоритму пузырьковой сортировки;
* Ограниченный прирост производительности: Хотя шейкерная сортировка смягчает проблему медленного "всплывания" больших элементов на свои правильные позиции за счет чередования направления сортировки, она все равно демонстрирует неоптимальное поведение. Прирост производительности по сравнению с пузырьковой сортировкой незначителен, и для задач сортировки существуют более эффективные алгоритмы;
* Недостаточная универсальность: Встряхивающая сортировка в первую очередь подходит для небольших или частично отсортированных списков. Она не очень хорошо подходит для сортировки больших наборов данных или в сценариях, где производительность имеет решающее значение. Более продвинутые алгоритмы сортировки, такие как quicksort или mergesort, предлагают лучшие характеристики производительности в этих случаях;
* Сортировка только на основе сравнения: Сортировка шейкеров полагается на попарные сравнения между элементами для определения их порядка. Это делает ее непригодной для сортировки сложных структур данных, которые могут потребовать более сложных методов сравнения или извлечения ключей.

Типичные сценарии применения:

1. Малые массивы данных: Шейкерная сортировка хорошо работает для небольших массивов данных, где сложность сортировки не является критическим фактором. Она проста в реализации и понимании, поэтому может быть удобна для обработки небольших объемов данных.  
  
2. Частично отсортированные массивы: Если в массиве уже существует частичный порядок, то шейкерная сортировка может быть эффективной, потому что она может обнаружить отсортированные участки и свести количество операций к минимуму. Это может быть полезно, когда данные поступают постепенно или требуются частые обновления.  
  
3. Сортировка почти отсортированных массивов: Если массив почти отсортирован, то шейкерная сортировка может показать хорошую производительность. Она имеет лучший случай временной сложности O(n) для таких массивов, так как может пропустить множество итераций.  
  
4. Визуализация сортировки: Шейкерная сортировка является простым и наглядным алгоритмом сортировки, что делает ее популярной для визуализации. Она может использоваться в учебных целях или для демонстрации процесса сортировки на экране.

# 3. Выбор решения

Нашей бригадой было выбрано вести разработку в среде Microsoft Visual Studio на языке C/C++.

Все действия, совершаемые программой, происходят в одном файле.

Массив данных заполняется случайными элементами с использованием цикла for.

После заполнения массива, данные переписываются в файл “rannum.txt”, а массив сортируется с помощью функции Scheik.

После того, как массив будет отсортирован, данные из массива переписываются в другой файл “result.txt”.

Программа завершает свою работу.

Если файл не был найден или не может быть создан – программа выдает ошибку.

# 4. Описание программы

В программе для шейкерной сортировки подключены следующие заголовочные файлы: *stdio.h* – заголовочный файл со стандартными функциями для организации ввода-вывода; *Stdlib.h* – [заголовочный файл](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB) [стандартной библиотеки](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0_%D0%A1%D0%B8) языка [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), который содержит в себе функции, занимающиеся выделением памяти, контролем процесса выполнения программы, преобразованием типов и другие; *time.h*– заголовочный файл [стандартной библиотеки языка программирования C](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0_%D0%A1%D0%B8)и, содержащий типы и функции для работы с датой и временем. Макрос *#define \_CRT\_SECURE\_NO\_WARNINGS* необходим для корректной работы функций.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

При запуске программы происходит вывод сообщения “Enter the size of the array:”, после чего Пользователю необходимо вписать нужное количество сортируемых данных.

Далее инициализируется и вводится переменная *size*, отвечающая за количество занесенных в файл чисел и размерность массива.

После этого выводится сообщение “Enter the size of the array:”, после чего Пользователю необходимо вписать нужное количество сортируемых данных.

Затем объявляется динамический массив *arr*. Далее располагается текст функции для создания и открытия файла, в которой устанавливается связь файловой переменной с файлом, открытие файла и запись в него и в массив случайных чисел, при помощи цикла со счетчиком и закрытие файла:

int size = 0; //инцилизация размера массива

printf("Enter the size of the array:");

scanf("%d", &size); //ввод размерности массива

int\* arr = (int\*)malloc(size \* sizeof(int)); //динамическое выделение памяти массива

FILE\* file = fopen("rannum.txt", "w"); //открытие файла для записи

srand(time\_t(NULL));

for (int i = 0; i < size; i++) //цикл для заполнения массива и файла случайными числами

{

int randomNumber = rand() % 100;

fprintf(file, "%d\n", randomNumber);

arr[i] = randomNumber;

}

fclose(file); //закрытие файла

Затем идет замер времени работы функции *Sheik,* а также ее вызов:

time\_t start = clock();//начало отсчета времени

Sheik(arr, size); //запуск функции сортировки

Далее идет текст функции шейкерной сортировки *Sheik*. Выбирается адрес массива, и его размер. А также инициализируются переменный обозначающие самое левое число, самое правое число, перестановку и 2 вспомогательные переменные:

void Sheik(int\* arr, int size)

{

int left = 0, right = size - 1, flag = 1;

int t = 0, t1 = 0;

Далее в цикле с предусловием, пока границы не сомкнутся и пока происходит перестановка, осуществляется перестановка элементов в массиве и сдвиг границ: меньшие элементы располагаются левее, а элементы, больше или равные – правее:

while ((left < right) && flag > 0)

{

flag = 0;

for (int i = left; i < right; i++) //двигаемся слева направо

{

if (arr[i] > arr[i + 1]) // если следующий элемент меньше текущего,

{ // меняем их местами

t = arr[i];

arr[i] = arr[i + 1];

arr[i + 1] = t;

flag = 1; // перемещения в этом цикле были

}

}

right--; // сдвигаем правую границу на предыдущий элемент

for (int i = right; i > left; i--) //двигаемся справа налево

{

if (arr[i - 1] > arr[i]) // если предыдущий элемент больше текущего,

{ // меняем их местами

t1 = arr[i];

arr[i] = arr[i - 1];

arr[i - 1] = t1;

flag = 1; // перемещения в этом цикле были

}

}

left++; // сдвигаем левую границу на следующий элемент

}

}

На этом текст функции шейкерной сортировки заканчивается.

Затем время работы функции засекается:

time\_t stop = clock();//конец отсчета времени

Потом выполняется создание и запись файла с результатом работы программы, а также запись в него отсортированного массива, закрытие файла:

FILE\* outputFile = fopen("result.txt", "w"); //открытие файла, для записи, для вывода результата

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

{

fprintf(outputFile, "%d\n", arr[i]); //запись в файл каждого элемента массива

}

fclose(outputFile);//закрытиек файла

Далее осуществляется вывод времени работы функции шейкерной сортировки, освобождение выделенной для массива *arr* памяти, вывод информации об исходном и отсортированном файлах:

double time = ((stop - start) / 1000.0); //время сортировки

printf("working time: %.3lf \n", time);//вывод времени сортировки

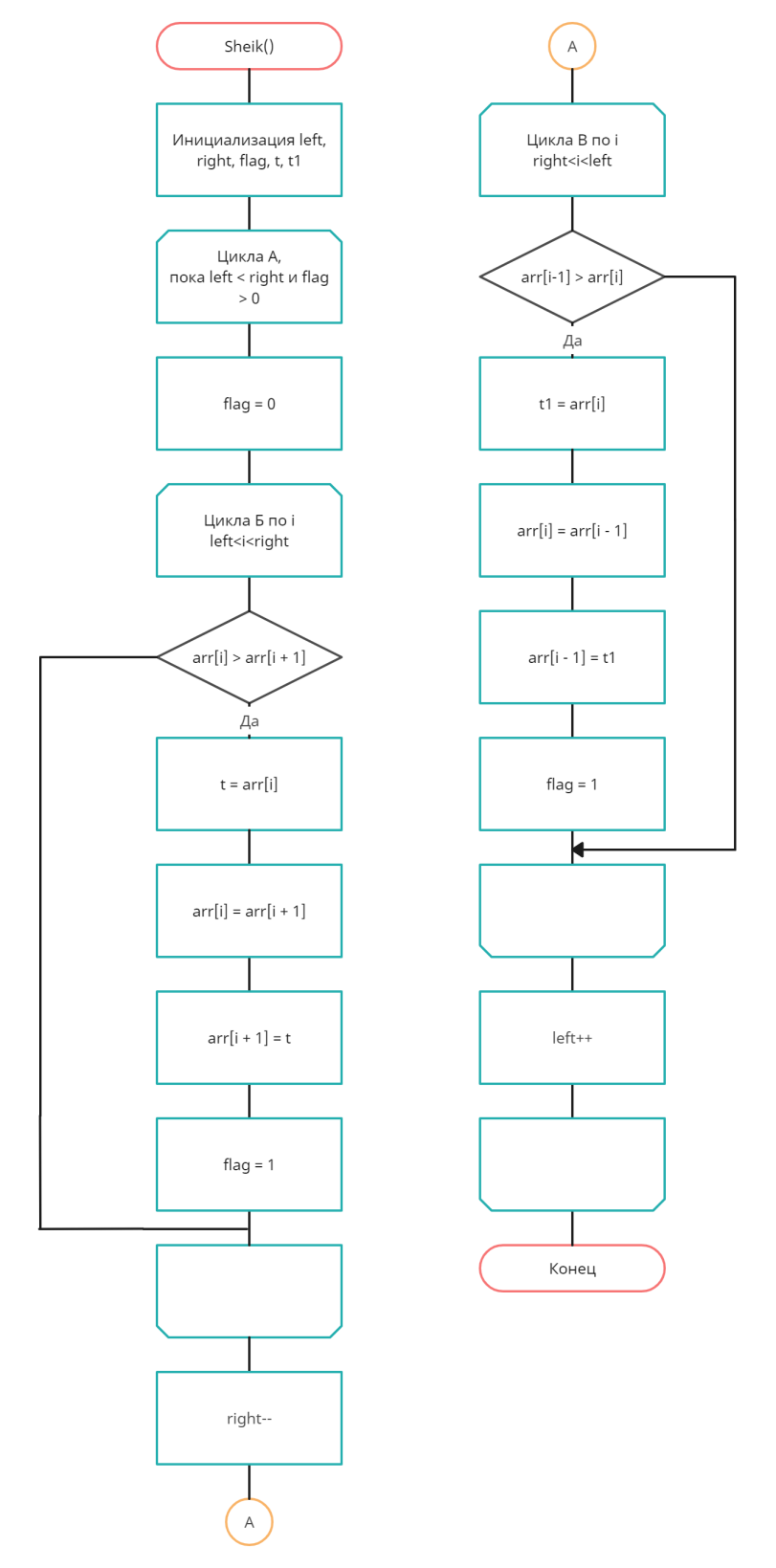
free(arr);//освобождение памяти

printf("Generated array: rannum.txt\n");

printf("Sorted array: result.txt");

Подробный алгоритм работы программы и функции сортировки представлен в пункте 5 на рисунках 1 “Блок-схема алгоритма” и рисунке 2 “Блок-схема программы”.

# 5. Схемы программы



Цикл А

Цикл Б

Нет

Нет

1

2

3

4

5

6

7

8

9

10

111

12

13

14

15

16

17

18

19

20

21

22

23

241

Рисунок 1 “Блок-схема алгоритма”



Цикл А по i

Цикл Б по i

1

2

3

4

5

6

7

8

9

10

111

12

13

14

15

16

17

18

19

20

21

22

23

241

25

26

Рисунок 2 “Блок-схема программы”

# 6. Тестирование программы

Тестовый набор данных представлен в таблице 1.

Снизу представлены диаграмма. (Диаграмма 1, в которой представлена линейная зависимость количества данных от времени, затраченного на выполнение сортировки).

Результаты тестирования приведены в Приложении А на рисунках

А.6 - А.16.

Таблица 1 – Тестовый набор данных на диапазоне от 10000 до 110000

|  |  |  |
| --- | --- | --- |
| № теста | Размер массива size | Время выполнения сортировки в секундах |
| 1 | 10000 | 0.168 |
| 2 | 20000 | 0.728 |
| 3 | 30000 | 1.775 |
| 4 | 40000 | 3.126 |
| 5 | 50000 | 5.159 |
| 6 | 60000 | 7.125 |
| 7 | 70000 | 9.701 |
| 8 | 80000 | 12.724 |
| 9 | 90000 | 16.049 |
| 10 | 100000 | 19.789 |
| 11 | 110000 | 23.948 |

Диаграмма 1 “Тестирование программы на диапазоне от 10000 до 110000”

В результате выполнения тестирования было выявлено время работы данного алгоритма и получен конечный результат, описанный в таблице.

# 7. Отладка

В качестве среды разработки была выбрана программа Microsoft Visual Studio , которая содержит в себе все необходимые средства для разработки и отладки модулей и программ.

Для отладки программы использовались точки остановки и пошаговое выполнение кода программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. После завершения написания программы, человеком, выполнявшим тестирование программы, были выявлены и исправлены ошибки.

# 8. Совместная разработка

Во время работы над данной практикой, нашей бригадой осуществлялась совместная работа в GitHub.

Данная программа была написана владельцем репозитория – Тихоновым Д.А.

После написания программы, она была выгружена на удаленный репозиторий Github, на отдельную ветку под названием Praktica.

(См. Рисунок 3)

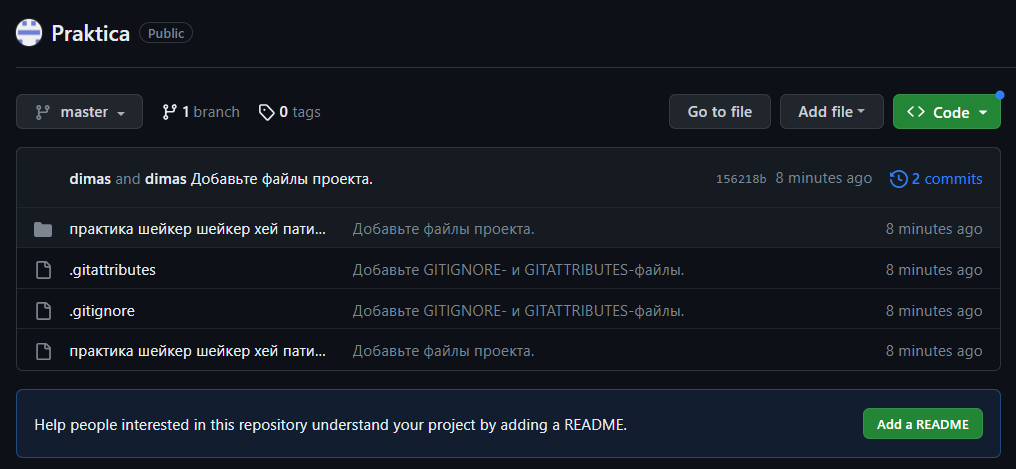


Рисунок 3

После этого, второй участник – Хворов Ф.М. загрузил данную программу себе на компьютер и добавил в нее алгоритм, позволяющий функции шейкерной сортировки работать с файлами, а также прописал интерфейс.

После выполнения данных изменений, Адов А.А. протестировал программу на различных данных, и, сверив результаты, сделал вывод о то, что «программа работает исправно, багов не имеет».

Ссылка на удаленный репозиторий:

https://github.com/dmeetro/Praktica.git

# 9. Заключение

Нашей бригадой были получены навыки совместной работы с помощью сервиса GitHub, навыки использования программы Git Bash. Нами так же был изучен алгоритм шейкерной сортировки. Тихонов Д.А. написал функцию, выполняющую данную сортировку над массивом случайно сгенерированных чисел, и оформил отчет по данной практике. Адов А.А. выполнил тестирование и отладку данной программы, Хворов Ф.М. написал программу, позволяющую функции шейкерной сортировки работать с файлами.

Так же при выполнении практической работы были улучшены наши базовые навыки программирования на языках С/C++. Улучшены навыки отладки, тестирования программ.

В дальнейшем программу можно улучшить путем подключения упрощающих реализацию данной сортировки библиотек и улучшения графического интерфейса.

# Список используемой литературы

1. Электронный ресурс. wikipedia.org Официальный сайт. – URL:

https://en.wikipedia.org/wiki/Cocktail\_sort (дата обращения 31.06.2023).

2. Электронный ресурс. geeksforgeeks.org Официальный сайт. – URL: https://www.geeksforgeeks.org/cocktail-sort/ (дата обращения 31.06.2023).

3. Электронный ресурс. toptal.com Официальный сайт. – URL: https://www.toptal.com/developers/sorting-algorithms/bubble-sort (дата обращения 01.07.2023).

4. Электронный ресурс. Brilliant.org Официальный сайт. – URL:

https://brilliant.org/wiki/shaker-sort/ (дата обращения 01.07.2023)

# Приложение А

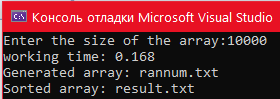


Рисунок А.4

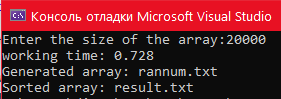


Рисунок А.5

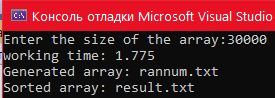


Рисунок А.6

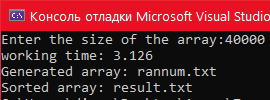


Рисунок А.7

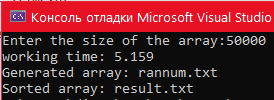


Рисунок А.8

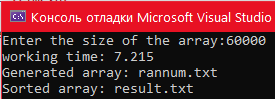


Рисунок А.9

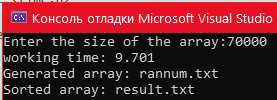


Рисунок А.10

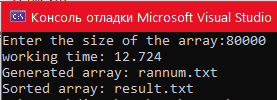


Рисунок А.11

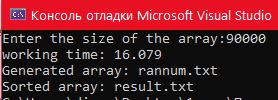


Рисунок А.12

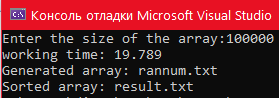


Рисунок А.13

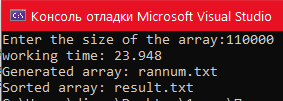


Рисунок А.14

# Приложение Б “Листинг”

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void Sheik(int\* arr, int size)

{

int left = 0, right = size - 1, flag = 1; // левая и правая границы сортируемой области массива// флаг наличия перемещений

int t = 0, t1 = 0;

// Выполнение цикла пока левая граница не сомкнётся с правой

// и пока в массиве имеются перемещения

while ((left < right) && flag > 0)

{

flag = 0;

for (int i = left; i < right; i++) //двигаемся слева направо

{

if (arr[i] > arr[i + 1]) // если следующий элемент меньше текущего,

{ // меняем их местами

t = arr[i];

arr[i] = arr[i + 1];

arr[i + 1] = t;

flag = 1; // перемещения в этом цикле были

}

}

right--; // сдвигаем правую границу на предыдущий элемент

for (int i = right; i > left; i--) //двигаемся справа налево

{

if (arr[i - 1] > arr[i]) // если предыдущий элемент больше текущего,

{ // меняем их местами

t1 = arr[i];

arr[i] = arr[i - 1];

arr[i - 1] = t1;

flag = 1; // перемещения в этом цикле были

}

}

left++; // сдвигаем левую границу на следующий элемент

}

}

int main()

{

int size = 0; //инцилизация размера массива

printf("Enter the size of the array:");

scanf("%d", &size); //ввод размерности массива

int\* arr = (int\*)malloc(size \* sizeof(int)); //динамическое выделение памяти массива

FILE\* file = fopen("rannum.txt", "w"); //открытие файла для записи

srand(time\_t(NULL));

if (file == NULL) {

printf("Error opening file.\n");

return 1;

}

for (int i = 0; i < size; i++) //цикл для заполнения массива и файла случайными числами

{

int randomNumber = rand() % 100;

fprintf(file, "%d\n", randomNumber);

arr[i] = randomNumber;

}

fclose(file); //закрытие файла

time\_t start = clock();//начало отсчета времени

Sheik(arr, size); //запуск функции сортировки

time\_t stop = clock();//конец отсчета времени

FILE\* outputFile = fopen("result.txt", "w"); //открытие файла, для записи, для вывода результата

if (file == NULL) {

printf("Error opening file.\n");

return 1;

}

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

{

fprintf(outputFile, "%d\n", arr[i]); //запись в файл каждого элемента массива

}

fclose(outputFile);//закрытиек файла

double time = ((stop - start) / 1000.0); //время сортировки

printf("working time: %.3lf \n", time);//вывод времени сортировки

free(arr);//освобождение памяти

printf("Generated array: rannum.txt\n");

printf("Sorted array: result.txt");

return 0;

}