

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Утвержден на заседании кафедры

«Вычислительная техника» \_\_\_\_\_

" \_\_\_\_ " \_\_\_\_\_ 20 \_\_\_\_ г.

Заведующий кафедрой

\_\_\_\_\_ М.А. Митрохин

**ОТЧЕТ ПО УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКЕ**  
(2022/2023 учебный год)

\_\_\_\_\_ Горбунов Данила Алексеевич \_\_\_\_\_

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное обеспечение средств  
вычислительной техники и автоматизированных систем»

Форма обучения – очная    Срок обучения в соответствии с ФГОС – 4 года

Год обучения \_\_\_\_\_ 1 \_\_\_\_\_ семестр \_\_\_\_\_ 2 \_\_\_\_\_

Период прохождения практики с 29.06.2023 по 12.07.2023

Кафедра «Вычислительная техника» \_\_\_\_\_

Заведующий кафедрой д.т.н., профессор, Митрохин М.А. \_\_\_\_\_

(должность, ученая степень, ученое звание, Ф.И.О.)

Руководитель практики д.т.н., профессор, Зинкин С.А.

(должность, ученая степень, ученое звание)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Утвержден на заседании кафедры

«Вычислительная техника»

" \_\_\_\_ " \_\_\_\_\_ 20 \_\_\_\_ г.

Заведующий кафедрой

\_\_\_\_\_ М.А. Митрохин

**ИНДИВИДУАЛЬНЫЙ ПЛАН ПРОХОЖДЕНИЯ УЧЕБНОЙ  
(ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ**

(2022/2023 учебный год)

\_\_\_\_\_ Горбунов Данила Алексеевич

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное обеспечение средств  
вычислительной техники и автоматизированных систем»

Форма обучения – очная      Срок обучения в соответствии с ФГОС – 4 года

Год обучения \_\_\_\_\_ 1 \_\_\_\_\_ семестр \_\_\_\_\_ 2 \_\_\_\_\_

Период прохождения практики с 29.06.2023 по 12.07.2023

Кафедра «Вычислительная техника»

Заведующий кафедрой д.т.н., профессор, Митрохин М.А.

*(должность, ученая степень, ученое звание, Ф.И.О.)*

Руководитель практики д.т.н., профессор, Зинкин С.А.

*(должность, ученая степень, ученое звание)*

№ п/п	Планируемая форма работы во время практики	Количество часов	Календарные сроки проведения работы	Подпись руководителя практики от вуза
1	Выбор темы и разработка индивидуального плана проведения работ	2	29.06.2023 - 29.06.2023	
2	Подбор и изучение материала по теме работы	15	30.06.2023 – 02.07.23	
3	Разработка алгоритма	43	02.07.23 – 06.07.23	
4	Описание алгоритма и программы	18	6.07.23 – 08.07.23	
5	Тестирование	5	08.07.23 – 08.07.23	
6	Получение и анализ результатов	10	08.07.23 – 10.07.23	
7	Оформление отчёта	15	10.07.23 – 12.07.2023	
	<b>Общий объём часов</b>	108		

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ОТЧЁТ

О ПРОХОЖДЕНИИ УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ

(2022/2023 учебный год)

Горбунов Данила Алексеевич

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное обеспечение средств  
вычислительной техники и автоматизированных систем»

Форма обучения – очная Срок обучения в соответствии с ФГОС – 4 года

Год обучения 1 семестр 2

Период прохождения практики с 29.06.2023 по 12.07.2023

Кафедра «Вычислительная техника»

Горбунов Д.А. выполнял практическое задание «Пирамидальная сортировка». На первоначальном этапе были изучен и проанализирован алгоритм пирамидальной сортировки, был выбран метод решения и язык программирования С, на котором была написана программа сортировки массива методом пузырька. Также, осуществил работу с файлами и интерфейс программы. Протестировал и отладил программу. Оформил отчёт.

Бакалавр Горбунов Д.А. \_\_\_\_\_ "\_\_\_" \_\_\_\_\_ 2023 г.

Руководитель Зинкин С.А. \_\_\_\_\_ "\_\_\_" \_\_\_\_\_ 2023 г.  
практики

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**ОТЗЫВ**

**О ПРОХОЖДЕНИИ УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ**

(2022/2023 учебный год)

Горбунов Данила Алексеевич

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Наименование профиля подготовки «Программное обеспечение средств  
вычислительной техники и автоматизированных систем»

Форма обучения – очная Срок обучения в соответствии с ФГОС – 4 года

Год обучения 1 семестр 2

Период прохождения практики с 29.06.2023 по 12.07.2023

Кафедра «Вычислительная техника»

В процессе выполнения практики Горбунов Д.А. решал следующие задачи: создание алгоритма пирамидальной сортировки, анализ работы алгоритма, создание интуитивно понятного меню, организация работы с файлами, тестирование программы.

За период выполнения практики были освоены основные понятия и технологии пирамидальной сортировки, реализован метод работы с файлами и консольный интерфейс, протестирована программа на разных наборах данных. Во время выполнения работы Горбунов Д.А. показал себя ответственным, добросовестным учеником, знающим свой предмет, имеющим представление о современном состоянии науки, владеющим современными общенаучными знаниями по информатике и вычислительной технике, программированию и сортировке.

За выполнение работы Горбунов Д. А. заслуживает оценки «      ».

Руководитель практики д.т.н., профессор, Зинкин С.А. «    » 2023 г.

## Содержание

Введение .....	7
1. Постановка задачи .....	8
1.1 Достоинства алгоритма .....	8
1.2 Недостатки алгоритма .....	9
2. Выбор решения .....	9
3. Описание программы .....	10
4. Схемы программы .....	14
4.1 Блок-схема программы .....	14
4.2 Блок-схема алгоритма .....	15
5. Тестирование программы .....	16
6. Отладка .....	17
7. Работа с системой контроля версий .....	18
Заключение .....	19
Список используемых источников .....	20
Приложение А .....	21
Листинги программы .....	21
Приложение В .....	24

## **Введение**

На сегодняшний день сортировка данных является одним из наиболее востребованных и распространённых процессов обработки данных. Задачи на сортировку данных встречаются очень часто в различных профессиональных сферах деятельности.

Алгоритмы сортировки очень широко распространяются практически во всех задачах обработки информации. При этом они настолько тесно связаны друг с другом, что образуют отдельный класс алгоритмов. Алгоритмы сортировки, как правило, применяются с целью дальнейшего упрощённого пользования информацией и доступа к ней.

Важность сортировки основана на том факте, что на её примере можно показать многие основные фундаментальные приёмы и методы построения алгоритмов. Сортировка является хорошим примером огромного разнообразия алгоритмов, которые выполняют одну и ту же задачу. Кроме того, многие из них имеют определённые преимущества друг перед другом. За счёт усложнения алгоритма можно добиться существенного увеличения эффективности и быстродействия алгоритма по сравнению с более простыми методами. Как правило, термин «сортировка» понимают, как процесс перестановки объектов некоторого множества в определённом порядке. Цель сортировки – облегчить дальнейший поиск элементов в отсортированном множестве.

Пирамидальная сортировка – сортирующее дерево, в котором любой родитель больше, чем его потомки. Если потомок больше родителя, то они меняются местами. Таким образом в «корне» дерева появляется наибольший элемент массива. Максимум обменивается с последним ключом неотсортированного подмассива. Структура прекращает быть сортирующим деревом, а его неотсортированная часть становится меньше на один узел. К этой части заново применяется вся процедура – преобразование в сортирующее дерево с последующей перестановкой найденного максимума в

конец. Действие повторяется до тех пор, пока неотсортированной часть не уменьшится до единственного элемента.

## **1. Постановка задачи**

Необходимо реализовать и продемонстрировать работу алгоритма пирамидальной сортировки. Программа должна реализовывать алгоритм сортировки, выполнять считывание входных данных с файла и запись результатов в файл и отображать все данные в консольном интерфейсе.

Программа должна обрабатывать данные, предоставленные в корректно сформированных входных файлах.

Конечная программа не должна использовать реализацию алгоритмов сортировки, предоставляемые стандартными библиотеками, включая системные.

Решение задачи пирамидальной сортировки разделяется на 4 этапа:

- Из сортируемого массива строится дерево, в котором любой родительский элемент больше потомка
- Максимум обменивается с последним ключом неотсортированного подмассива
- Снова строится дерево с перестановкой найденного максимума в конец.
- Повторение 3 пункта до тех пор, пока не отсортируется массив.

### **1.1 Достоинства алгоритма**

- Худшее время работы –  $O(n \log n)$
- Требуется  $O(1)$  дополнительной памяти(сортирует на месте)



## **1.2 Недостатки алгоритма**

- На одном шаге выборку приходится делать хаотично по всей длине массива — поэтому алгоритм плохо сочетается с кэшированием и подкачкой памяти
- На почти отсортированных данных работает столь же долго, как и на хаотических данных
- Методу требуется мгновенный прямой доступ, не работает на связанных списках и других структурах памяти последовательного доступа

## **1.3 Типичные сценарии применения**

Пирамидальная сортировка поддерживает быстрое извлечение минимума/максимума из набора неупорядоченных элементов. Поэтому данный метод сортировки применяется к структурам данных, к элементам которых имеется прямой доступ, например, к массивам.

Данный алгоритм сортировки используется в системах, связанных с безопасностью, и встроенных системах, таких как ядро Linux.

## **2. Выбор решения**

Для написания программы был выбран язык программирования C — универсальный язык программирования, который имеет ряд преимуществ, таких как богатый набор операторов, экономичность выражения, современный поток управления и структуры данных. Отсутствие ограничений и общность языка делают его более удобным и эффективным для многих задач, чем языки более высокого уровня.

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Microsoft Visual Studio — это программная среда по разработке приложений для ОС Windows, как консольных, так и с графическим интерфейсом. Данная среда разработки удобна для написания,

редактирования, отладки и сборки кода, а затем для развертывания приложения. Помимо редактирования и отладки кода, Visual Studio включает компиляторы, средства завершения кода, систему управления версиями, расширения и многие другие функции для улучшения каждого этапа процесса разработки программного обеспечения.

Сортируемый массив создан динамическим с целью изменения его размера и тестирования времени сортировки массивов разных размеров.

Сортирующее дерево выполнено таким образом, что любой родительский элемент будет больше потомственного, что упрощает возможность нахождения наибольшего элемента массива. Наибольший элемент после каждой «пересборки» сортирующего дерева будет находиться в «корне» дерева и перемещаться в отсортированный массив.

### 3. Описание программы

Программа «Пирамидальная сортировка» основана на основе консольного меню и включает в себя следующие возможности:

- Задать размер массива и начать сортировку с последующей записью отсортированных данных в текстовый файл.
- Выход из программы.

При запуске программы открывается меню программы

(см. Приложение В)

```
printf("||=====||\n");
printf("||      HeapSort      ||\n");
printf("||                      ||\n");
printf("||  1. Начать сортировку  ||\n");
printf("||  2. Выход              ||\n");
printf("||=====||\n");
```

Главное меню состоит из двух пунктов:

- 1) Сортировка. После выбора данного пункта и задания корректного размера массива запускается процесс сортировки.

2) Выход. После выбора данного пункта осуществляется выход из программы

```
while (!_kbhit());
    switch (_getch()) {
        case 49: hsort();
            break;
        case 50: _Exit(0);
            break;
        default:
            goto label1;
    }
    return 0;
```

Навигация в меню осуществляется с помощью клавиатуры, нажатием клавиш 1 или 2. Отсутствует необходимость нажатия клавиши Enter. Нажатие иных клавиш ни к чему не приводит.

Описание состояний программы выполнено в таблице ниже.

Таблица 1 – Работа функции меню

Клавиши, вызывающие событие	Действие пользователя	Действие программы
1	Выбран пункт «Начать сортировку»	Пользователю предлагается ввести размер массива, после этого запускается сортировка массива данного размера
2	Выбран пункт «Выход»	Осуществляется выход из программы

После выполнения сортировки пользователь увидит время выполнения сортировки, а так же новое меню, которое является полным аналогом главного и имеет такие же функции. (см. Приложение В)

Таблица 2 – Работа функции restart()

Клавиши, вызывающие событие	Действие пользователя	Действие программы
1	Выбран пункт «Сортировать заново»	Пользователю предлагается ввести размер массива, после этого запускается сортировка массива данного размера
2	Выбран пункт «Выход»	Осуществляется выход из программы

Для определения размера будущего массива объявляется переменная n целого типа:

```
int n;
```

Затем у пользователя запрашивается размер массива, размер считывается и записывается в переменную n. Также осуществляется проверка на корректность введенного размера массива.

```
label:
system("cls");
printf("||=====||\n");
printf(" Введите размер массива:");
scanf("%d", &n);
if (n <= 0) goto label;
```

После того, как данные были введены, генерируется массив из случайных чисел.

```
arr = (int*)malloc(n * sizeof(int));
for (i = 0; i < n; i++) {
arr[i] = rand() % 10000;
}
```

Эти числа записываются в файл nosort.txt. Также выполняется проверка на создание файла.

```
if ((nosort = fopen("nosort.txt", "w")) == NULL) error();
```

```

for (i = 0; i < n; i++) {
    fprintf(nosort, "%d ", arr[i]);
}
fclose(nosort);

```

Далее над этими данными выполняется пирамидальная сортировка. Элемент-потомок сравнивается с родительским элементом и, если он больше, то меняется с ним местами. Так происходит до тех пор, пока не получится сортирующее дерево. Затем последний элемент меняется местами с корнем дерева, а корень дерева встаёт в конец массива. После этого сортирующее дерево восстанавливается. Так повторяется до тех пор, пока в дереве не останется 1 элемент.

Алгоритм сортировки:

```

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

```

После этого отсортированный массив записывается в файл sort.txt.

Программа также осуществляет подсчёт времени, которое заняла сортировка.

## 4. Схемы программы

### 4.1 Блок-схема программы

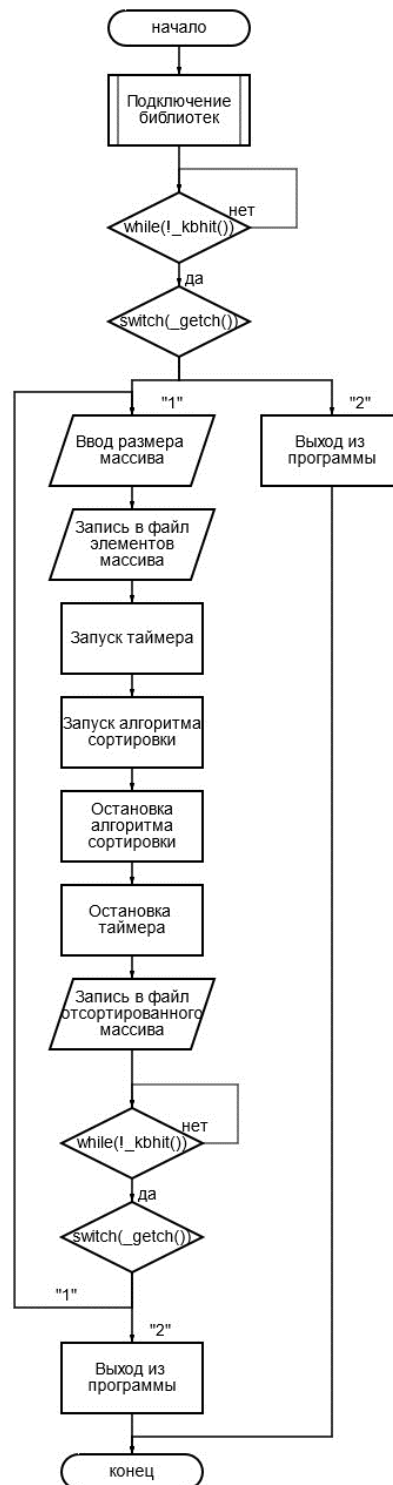


Рисунок 1 – Блок-схема программы

## 4.2 Блок-схема алгоритма

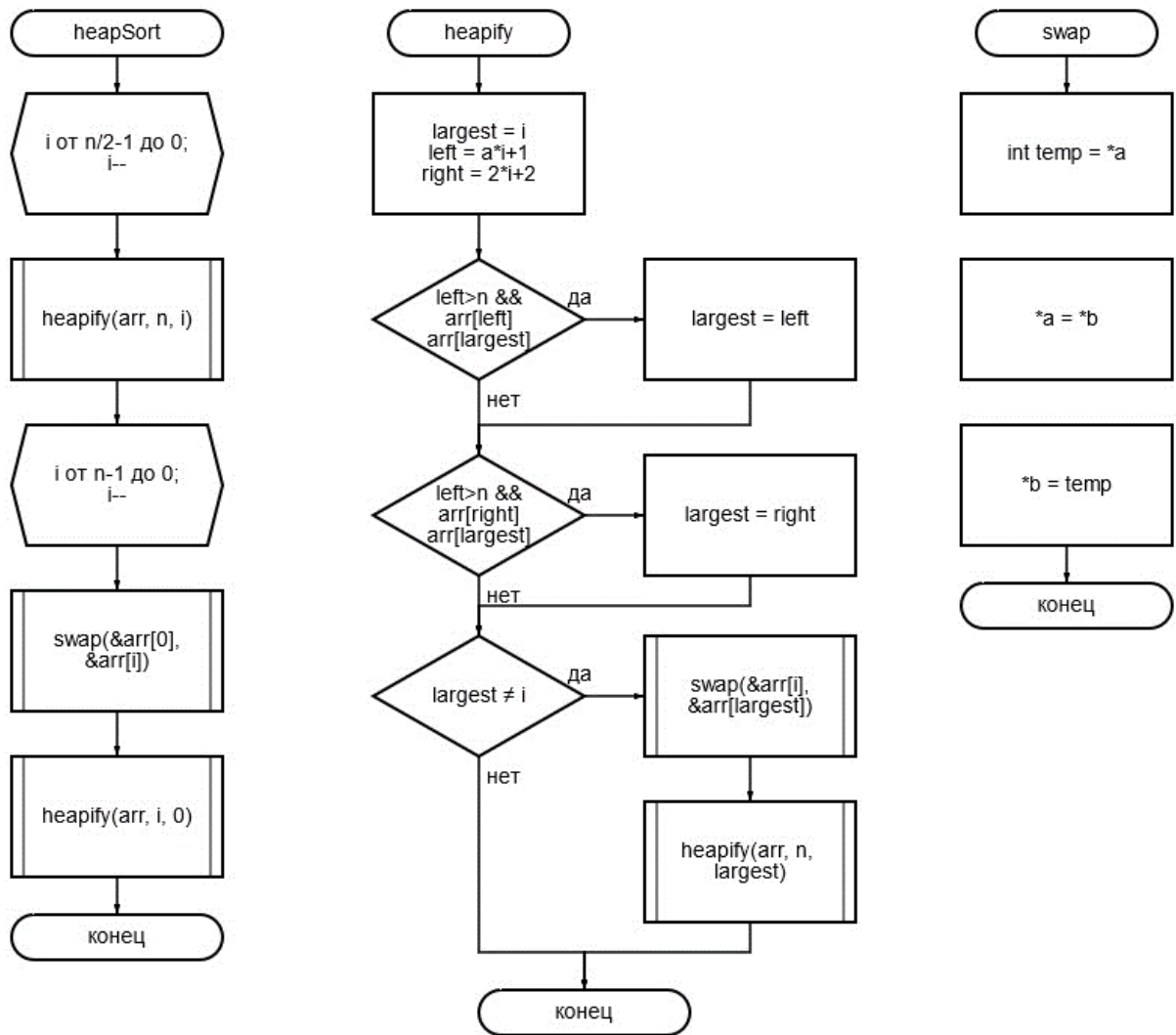


Рисунок 2 – Блок-схема алгоритма

## 5. Тестирование программы

Тестовый набор данных представлен в таблице 3.

Таблица 3 – Тестовый набор данных

№ теста	Размер массива	Время выполнения сортировки в секундах
1	10000	0.002
2	20000	0.003
3	30000	0.005
4	40000	0.007
5	50000	0.009
6	60000	0.012
7	70000	0.014
8	80000	0.016
9	90000	0.018
10	100000	0.02
11	110000	0.022

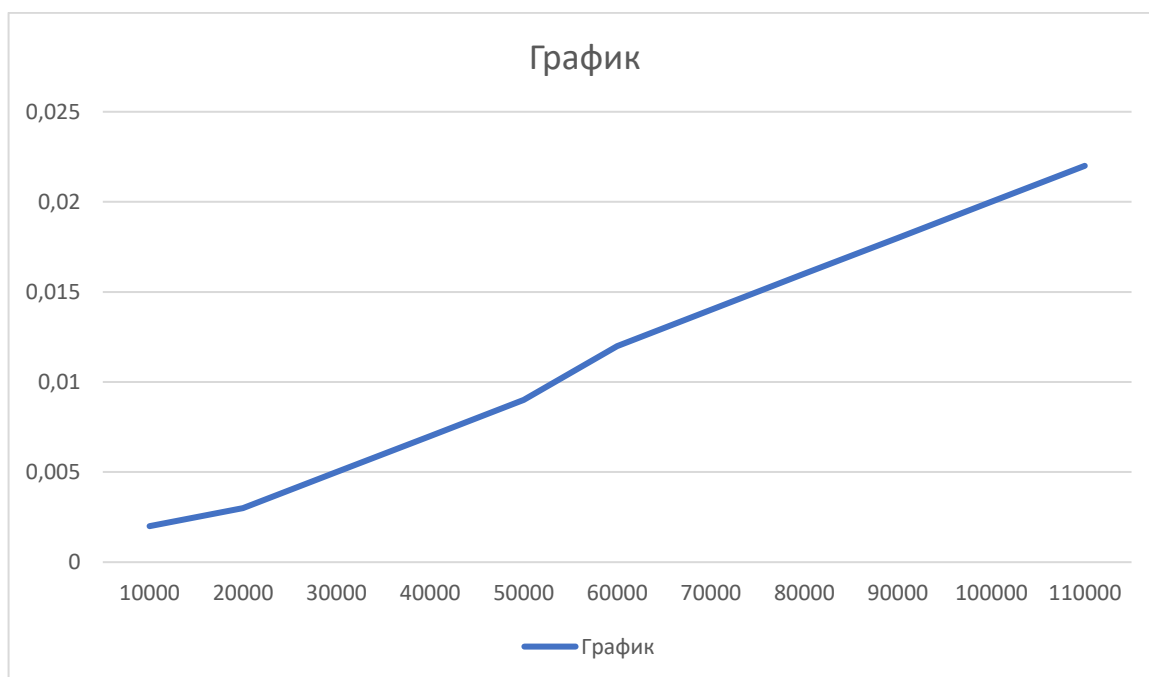


Рисунок 3 – Результаты тестирования



## **6. Отладка**

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Программа обладает всеми средствами, необходимыми при разработке и отладке программы. Для отладки использовалось несколько возможностей Visual Studio: точка останова, трассировка, анализ содержимого переменных.

Точка останова указывает программе, где следует приостановить выполнение кода, чтобы проверить значения переменных или поведение памяти. Точку останова можно задать, щелкнув в поле слева от строки кода.

С помощью клавиши F11 можно выполнить шаг с заходом при выполнении отладки. При запуске приложения с помощью клавиши F11 отладчик останавливается на первом выполняемом операторе, чтобы мы могли проверить содержимое переменных. Если оператор содержит вызов функции, то при шаге с заходом программа приостанавливает работу на первой строчке этой функции.

Клавиша F10 позволяет выполнять отладку без захода в функции в ходе.

Проверка содержимого переменных при проведении отладки осуществлялась с помощью окна «Видимые», в котором отображаются переменные, используемые вокруг текущей точки останова, и окна «Локальные», где показываются переменные, которые находятся в текущей области видимости.

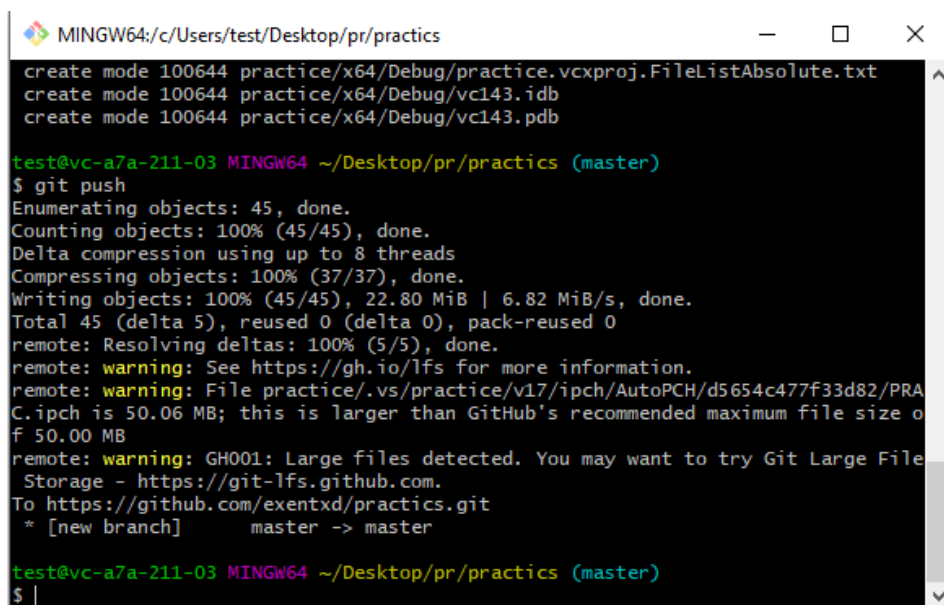
## 7. Работа с системой контроля версий

Во время работы над практикой была осуществлена работа в GitHub.

Мною был создан удалённый репозиторий, в который по мере выполнения практики загружались файлы проекта, отчёта и прочее.

Для загрузки данных на локальный репозиторий, а также отправки данных на удалённый репозиторий были использованы основные команды Git Bash: `git push`, `git pull`, `git clone` и другие.

Ссылка на удалённый репозиторий: <https://github.com/exentxd/practics>



```
MINGW64: c:/Users/test/Desktop/pr/practics
create mode 100644 practice/x64/Debug/practice.vcxproj.FileListAbsolute.txt
create mode 100644 practice/x64/Debug/vc143.idb
create mode 100644 practice/x64/Debug/vc143.pdb

test@vc-a7a-211-03 MINGW64 ~/Desktop/pr/practics (master)
$ git push
Enumerating objects: 45, done.
Counting objects: 100% (45/45), done.
Delta compression using up to 8 threads
Compressing objects: 100% (37/37), done.
Writing objects: 100% (45/45), 22.80 MiB | 6.82 MiB/s, done.
Total 45 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote: warning: See https://gh.io/lfs for more information.
remote: warning: File practice/.vs/practice/v17/ipch/AutoPCH/d5654c477f33d82/PRA
C.ipch is 50.06 MB; this is larger than GitHub's recommended maximum file size o
f 50.00 MB
remote: warning: GH001: Large files detected. You may want to try Git Large File
Storage - https://git-lfs.github.com.
To https://github.com/exentxd/practics.git
 * [new branch]      master -> master

test@vc-a7a-211-03 MINGW64 ~/Desktop/pr/practics (master)
$
```

Рисунок 4 – Загрузка данных на репозиторий

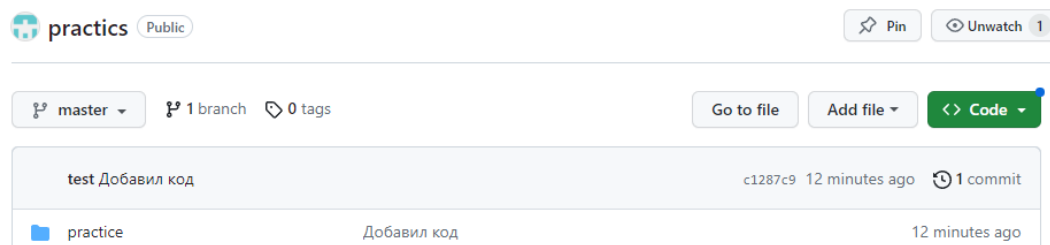


Рисунок 5 – Данные загружены на репозиторий.

## **Заключение**

В ходе выполнения данной практической работы были улучшены базовые знания программирования на языке C. Улучшены навыки отладки, тестирования программ и работы со сложными типами данных.

Данная практическая работа была выполнена в соответствии поставленной задачи в среде Microsoft Visual Studio 2022. Было проведено исследование компонентов программной среды Microsoft Visual Studio 2022, которые использовались при создании программы, также использовалось множество функций.

В дальнейшем программу можно улучшить путём добавления восходящей пирамидальной сортировки, а также добавлением графического интерфейса.

## **Список используемых источников**

1. J-sort [Электронный ресурс] – URL: <https://habr.com/ru/articles/221095/>
2. Восходящая сортировка кучей [Электронный ресурс] – URL: <https://habr.com/ru/companies/edison/articles/509330/>

## Приложение А

### Листинги программы.

#### Файл prас.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <locale.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Heap sort
    for (int i = n - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i]);

        heapify(arr, i, 0);
    }
}

void restart();
void error();
```

```

void hsort() {
    FILE* nosort;
    FILE* sort;
    int n;
    int i = 0;
    int* arr;
    label:
    system("cls");
    printf("||=====||\n");
    printf(" Введите размер массива:");
    scanf("%d", &n);
    if (n <= 0) goto label;
    srand(time(0));
    if ((nosort = fopen("nosort.txt", "w")) == NULL) error();
    arr = (int*)malloc(n * sizeof(int));
    for (i = 0; i < n; i++) {
        arr[i] = rand() % 10000;
    }
    for (i = 0; i < n; i++) {
        fprintf(nosort, "%d ", arr[i]);
    }
    fclose(nosort);

    clock_t st = clock();
    heapSort(arr, n);
    clock_t end = clock();

    float sec = float(end) - float(st);
    float milsec = sec / 1000;
    if ((sort = fopen("sort.txt", "w")) == NULL) error();
    for (i = 0; i < n; i++) {
        fprintf(sort, "%d ", arr[i]);
    }
    fclose(sort);
    free(arr);
    system("cls");
    printf("||=====||\n");
    printf("|| Массив успешно отсортирован ||\n");
    printf("||      за %5.3f секунд      ||\n", milsec);
    printf("||      1. Сортировать заново      ||\n");
    printf("||      2. Выход                    ||\n");
    printf("||=====||\n");
    restart();
}

void error() {
    system("cls");
    printf("||=====||\n");
    printf("||                               ||\n");
    printf("||      Error!                    ||\n");
    printf("||      Не удалось открыть файл  ||\n");
    printf("||                               ||\n");
}

```

```

    printf("||=====||\n");
    _Exit(0);
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    printf("||=====||\n");
    printf("||          HeapSort          ||\n");
    printf("||          ||\n");
    printf("||          1. Начать сортировку      ||\n");
    printf("||          2. Выход                  ||\n");
    printf("||=====||\n");
label1:
    while (!_kbhit());
    switch (_getch()) {
    case 49: hsort();
        break;
    case 50: _Exit(0);
        break;
    default:
        goto label1;
    }
    return 0;
}

void restart() {
label2:
    while (!_kbhit());
    switch (_getch()) {
    case 49: hsort();
        break;
    case 50: _Exit(0);
        break;
    default:
        goto label2;
    }
}

```

## Приложение В

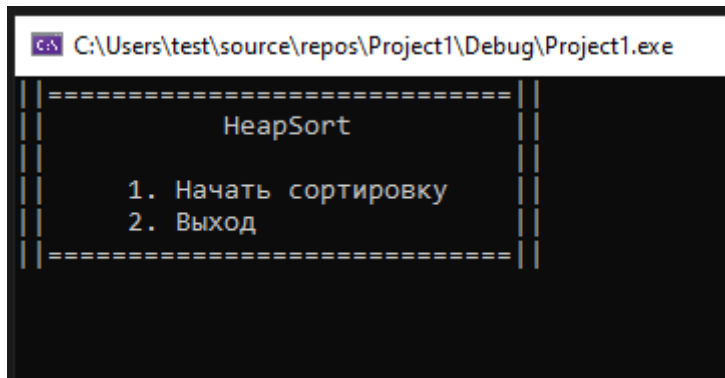


Рисунок 5 – Главное меню программы

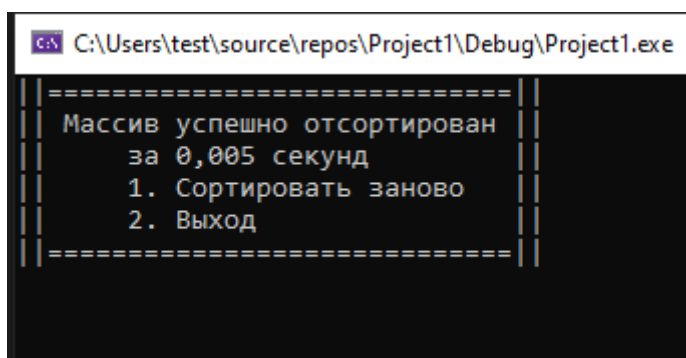


Рисунок 6 – Меню рестарта