МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский Нижегородский

государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки: Фундаментальная информатика и информационные технологии.

**Отчет по лабораторной работе №1**

**«Сравнение эффективности алгоритмов сортировки»**

**Выполнил**: студент группы 3822Б1ФИ3

Дудченко Олеся Викторовна

**Проверила**:

ассистент кафедры МОСТ,

Усова М.А.

Нижний Новгород

2022

**Содержание**

[Введение 3](#_Toc117849017)

[Постановка задачи 4](#_Toc117849018)

[Руководство пользователя 5](#_Toc117849019)

[Руководство программиста 5](#_Toc117849020)

[Описание структуры программы 5](#_Toc117849021)

[Описание алгоритмов 6](#_Toc117849022)

[Бинарный поиск 6](#_Toc117849023)

[Сортировка выбором 8](#_Toc117849024)

[Сортировка пузырьком 8](#_Toc117849025)

[Сортировка вставками 8](#_Toc117849026)

[Сортировка Хоара 8](#_Toc117849027)

[Вычислительный эксперимент 8](#_Toc117849028)

[Описание вычислительного эксперимента 8](#_Toc117849029)

[Результаты вычислительного эксперимента 8](#_Toc117849030)

[Анализ эффективности алгоритмов 9](#_Toc117849031)

[Заключение 9](#_Toc117849032)

[Литература 9](#_Toc117849033)

[Приложения 10](#_Toc117849034)

[Приложение 1. Функция бинарного поиска 10](#_Toc117849035)

Введение

Цель данной работы – написать программу, на вход которому будет подаваться массив чисел, с которым можно будет работать. Массивом является тип или структура данных в виде набора элементов, расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью индексации. Программа должна быть написана на языке Си. Для откладки использовался Visual Studio 2022. Для меня эта работа важна, как для программиста, потому что в ней используются самые базовые знания языка, которые можно использовать в данном коде (функции для работы с файлами, сложное меню, различные библиотеки, структуры, работа с массивами). Тем самым я смогу закрепить полученные знания в первом семестре. Данная программа может помочь при определенных обстоятельствах в различных сферах, когда нужна будет работа с числами.

Постановка задачи

Мне надо написать программу, которая будет выдавать меню. На вход программе подается массив чисел. Его можно ввести вручную, случайно сгенерировать и прочитать из файла. Нужно, чтобы массив в любое время можно было вывести. Должны присутствовать два вида поиска элементов: бинарный и наивный. Мы вводим элемент массива, индекс которого хотим узнать, и программа нам его печатает, если введенный элемент существует. В коде должны быть реализованы 4 вида сортировок массива: сортировка пузырьком, сортировка выбором, сортировка подсчетом, сортировка слиянием. К тому же программа должна засекать время работы каждой сортировки. После сортировки массив должен быть записан в файл. Также в программе должен присутствовать пункт смены директории. Нужно реализовать сравнение сортировок. Выводится таблица со столбцами: сортировка, время сортировки, замедление, количество перестановок, увеличение количества перестановок, количество сравнений, увеличение числа сравнений. В программе должно присутствовать «Эффективность сортировки», с помощью которой мы можем узнать, какая сортировка эффективней при определенном размере массива. Для усложнения задачи можно было реализовать смену языка.

Руководство пользователя

Программный комплекс начинает свою работу с того, что выдает нам главное меню, в котором мы можем выбрать один из пунктов Рис. 1:



Рис. 1 Главное меню

Навигация происходит при помощи следующих кнопок:

«Arrow\_up» - перемещение вверх.

«Arrow\_down» - перемещение вниз.

«Enter» - выбор нужного пункта.

Настройки.

Если перейти в пункт << Настройки >>, то у нас появится следующее меню Рис. 2:

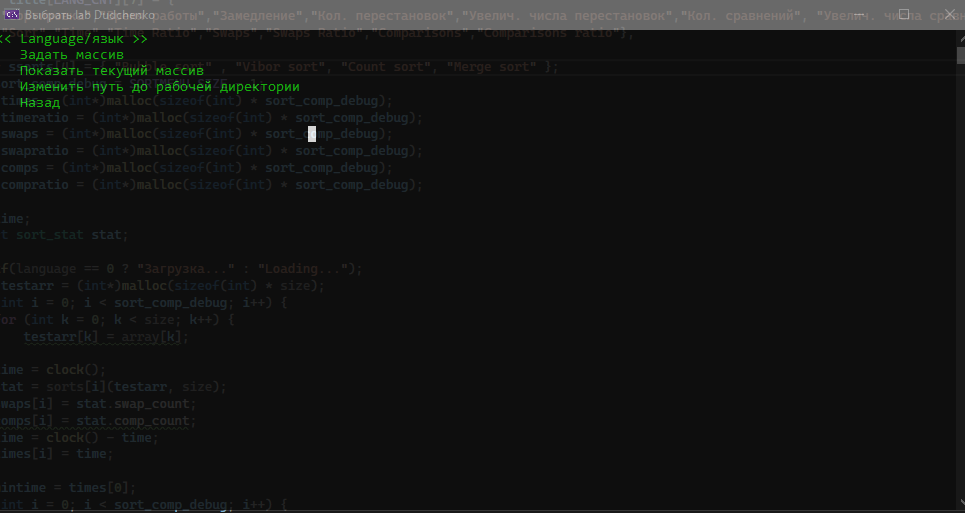


Рис. 2 Настройки

Язык.

Один из самых важных пунктов для пользователя это выбор языка. Если выбрать пункт << Language/Язык >>, то у нас появится следующий выбор Рис.3:

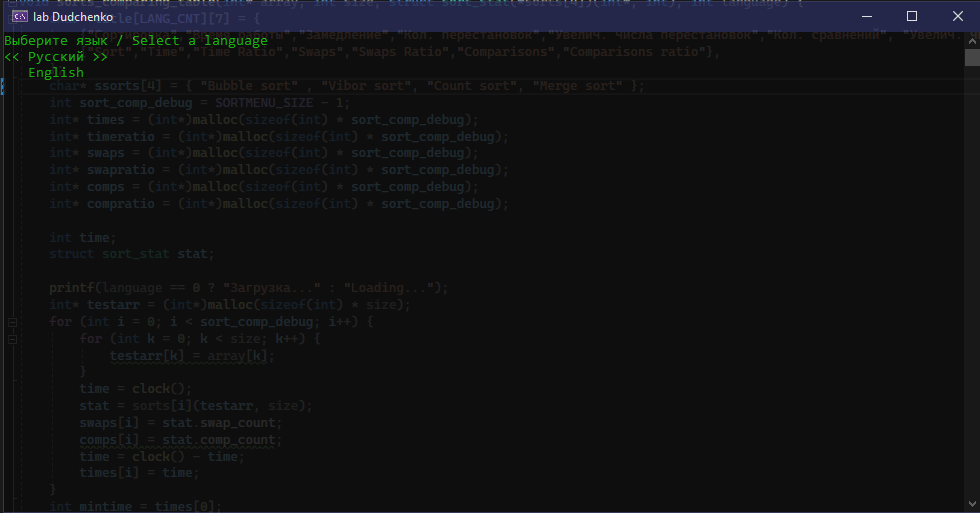


Рис.3 Языки

Соответственно, пользователь, может выбрать язык, на котором ему удобней пользоваться. Вся программа локализирована под оба языка.

Объявление массива.

Для корректной работы программы, в начале работы нужно задать массив, иначе программа может работать некорректно. Если выбрать пункт << Задать массив >>, то перед нами появится следующее меню:

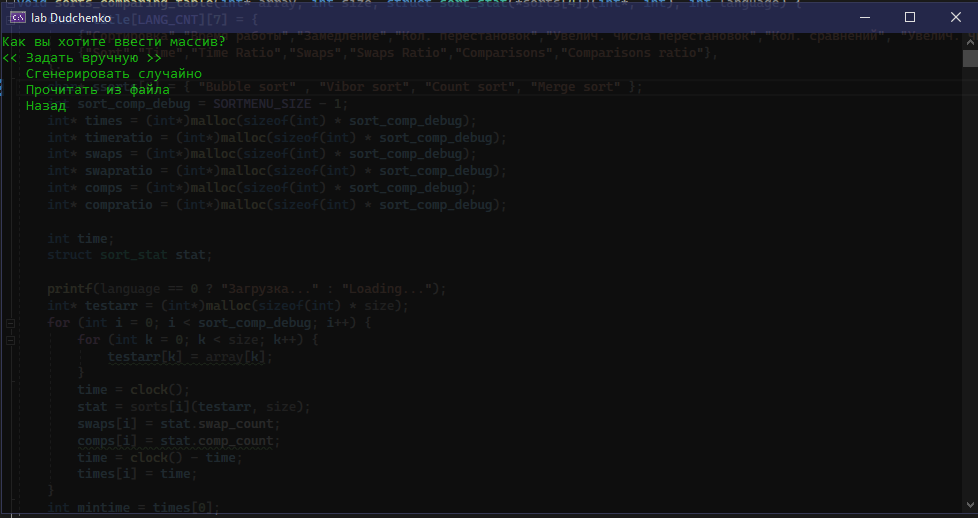


Рис.4 Меню массива

Соответственно если выбрать << Задать вручную >>, то программа попросит нас ввести размер массива, а после все элементы массива. Этот способ является одним из самых долгих и подходит для массивов маленького размера.

При выборе << Сгенерировать случайно >> программа опять же попросит нас ввести размер массива, а после сгенерирует случайные числа от 0 до 1000 и заполнит наш массив.

<< Прочитать из файла >> - один из самых сложных способов получения массива. Для его использования вам перед использованием данного пункта нужно перейти в << Изменить путь до рабочей директории >> и ввести путь и название файла, из которого хотите получить массив. О том, как вводить данные, читайте в инструкции по данному пункту. Далее вам надо проверить, корректны ли данные в файле, который вы указали. Первое число, которое у вас записано, программа запомнит, как размер массива, а остальные числа попадут в сам массив.

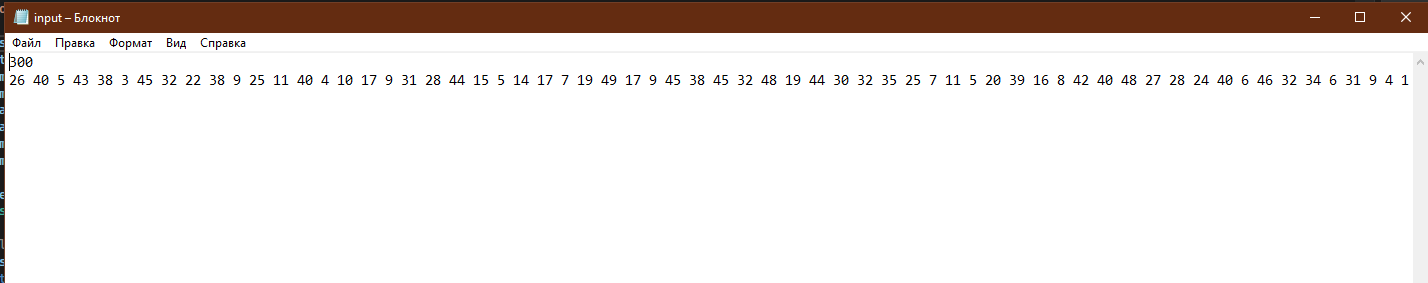


Рис. 5 Корректный ввод данных

После того, как вы введете путь и название файла и проверите данные в файле, вы можете смело переходить в << Прочитать из файла >>. Программа возьмет данные из файла и выведет полученный массив.

Соответственно, пункт << Назад >> возвратит нас обратно в главное меню.

Посмотреть текущий массив.

Если выбрать пункт << Показать текущий массив >>, то программа нам покажет массив, который был задан ранее. Если же массив не был задан, то нас попросят его ввести.

Сменить путь до рабочей директории.

Как уже ранее говорилось, в пункте << Сменить путь до рабочей директории >>, нам нужно будет вводить путь и название файлов. Один из файлов используется для получения массива, а другой для того, чтобы массив после сортировки был записан в него.

Пример ввода пути до файла:

D:\\files\_for\_lab\\input.txt

Далее вас попросят ввести путь до файла вывода. Нужно будет ввести до него путь в том же виде, как и для файла ввода.

Алгоритмы.

<< Алгоритмы >> - пункт, в котором можно выполнять различные действия над массивом. Если мы перейдем в данный пункт, то перед нами появится следующее меню Рис. 6:

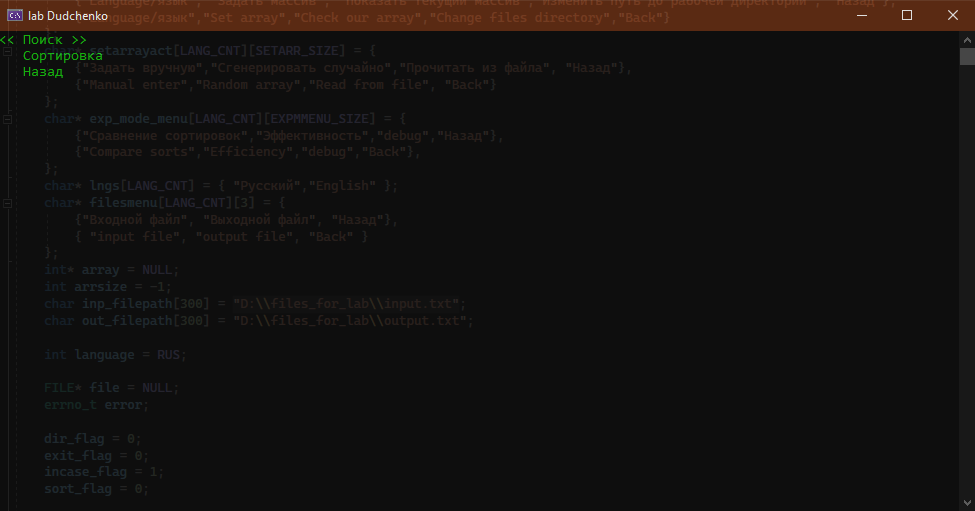


Рис. 6 Алгоритмы

Поиск.

Если выбрать пункт меню, то перед нами появится следующее меню:

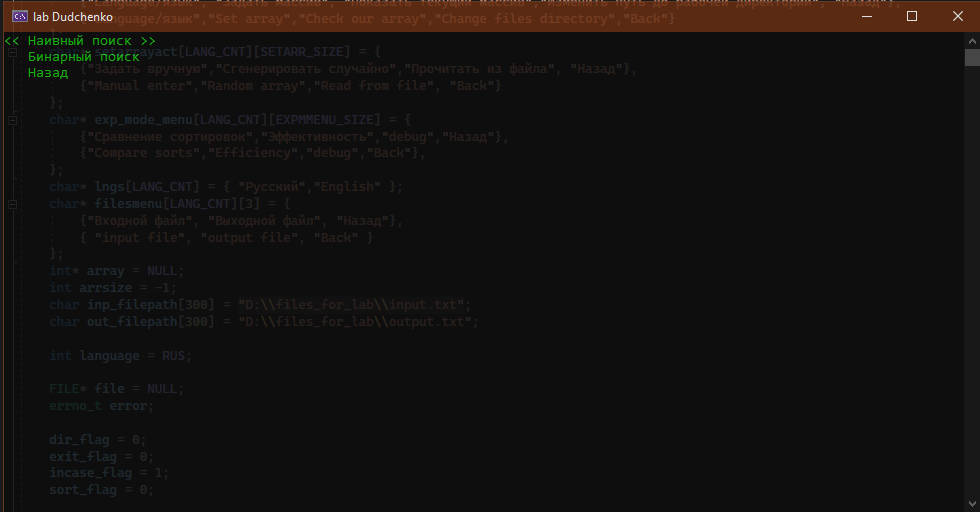


Рис.7 Поиски

Реализовано два вида поиска. Первый из них – наивный поиск. Данный способ хоть и является медленным, но весьма практичен. Мы вводим элемент массива, индексы которого хотим узнать, и программа выводит индексы. Если же данного элемента нет в массиве, то программа нам напишет об этом.

Бинарный поиск намного быстрее, чем наивный. Перед тем, как начать поиск, программа попросит нас отсортировать массив. Мы сможем выбрать, хотим ли мы его отсортировать или нет. После того, как мы отсортируем массив, нам нужно будет ввести элемент массива, и получаем индекс. Но минус данного вида поиска в том, что программа нам выведет самый первый индекс найденного элемента и остановится на этом. Поэтом если мы хотим узнать, если мы хотим узнать все индексы, на которых находится элемент, то надо использовать Наивный поиск.

Пример использования Рис.8:

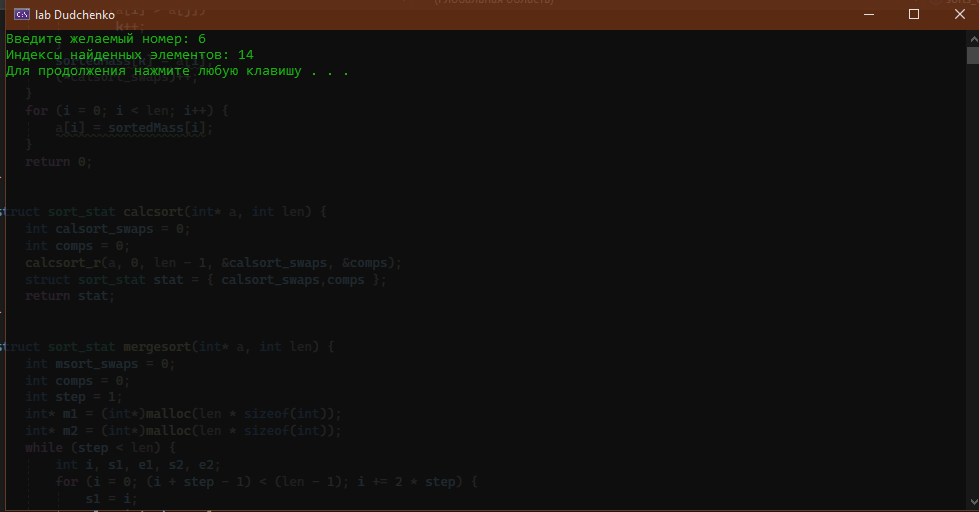


Рис. 8 Пример использования поиска

Следовательно, в нашем массива элемент «6» находится на 14 месте в неотсортированном массиве.

Сортировка массива.

Перейдя в << Сортировки >>, перед нами появится 4 вида сортировки:

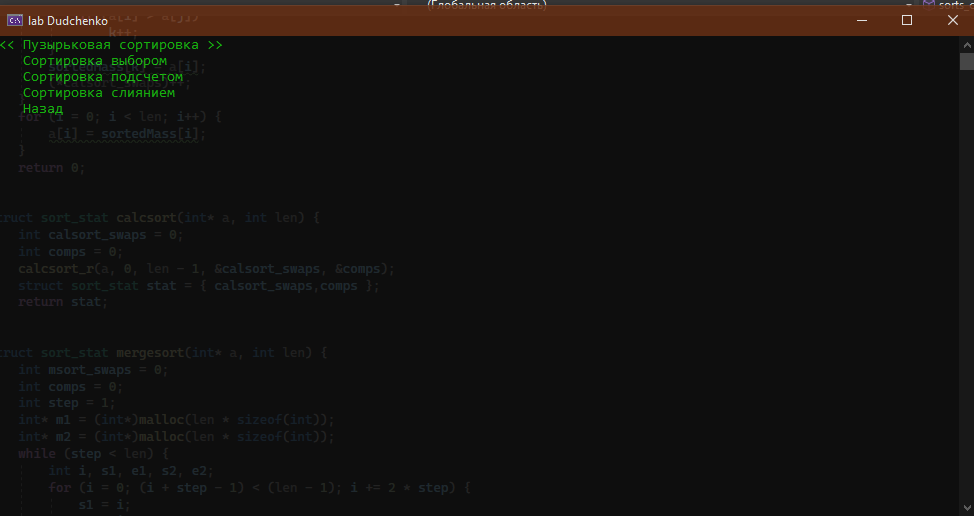


Рис.9 Сортировки

Каждая сортировка по-своему особенная. Например, пузырьковая сортировка довольно простая для понимания, но ее желательно использовать, если ваш массив не превышает 1000 элементов, иначе скорость данной сортировки будет довольно мала. Для таких случаев есть другие виды сортировок, которые сделают это быстрее. После каждой сортировки массив будет записан в файл, если вы указали путь до него в << Сменить путь до рабочей директории >>. Программа выведет отсортированный массив и файл, путь до которого вы указывали Рис. 10.

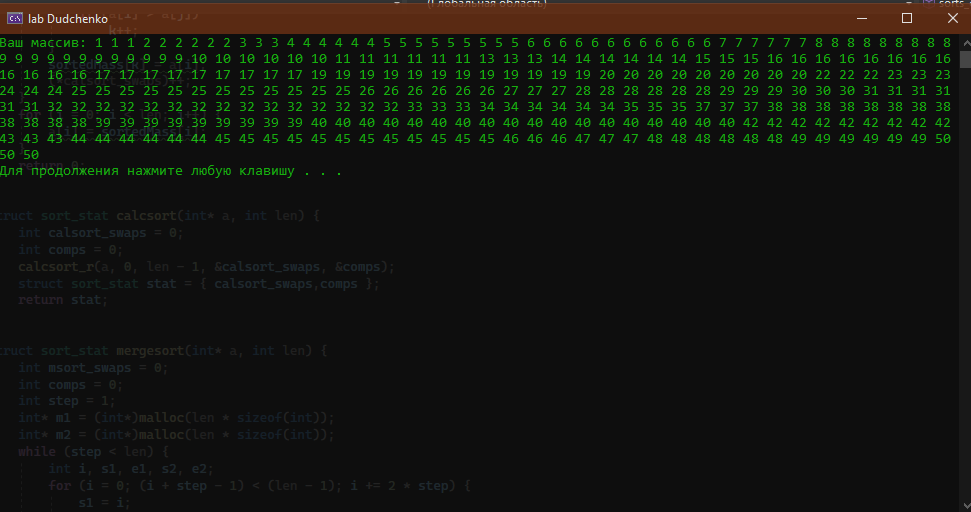


Рис. 10 Отсортированный массив

Режим эксперимента.

В режиме эксперимента можно узнать, какая сортировка эффективнее, сравнить их. Если перейти в << Режим эксперимента >>, то перед нами появится следующее меню выбора Рис. 11:

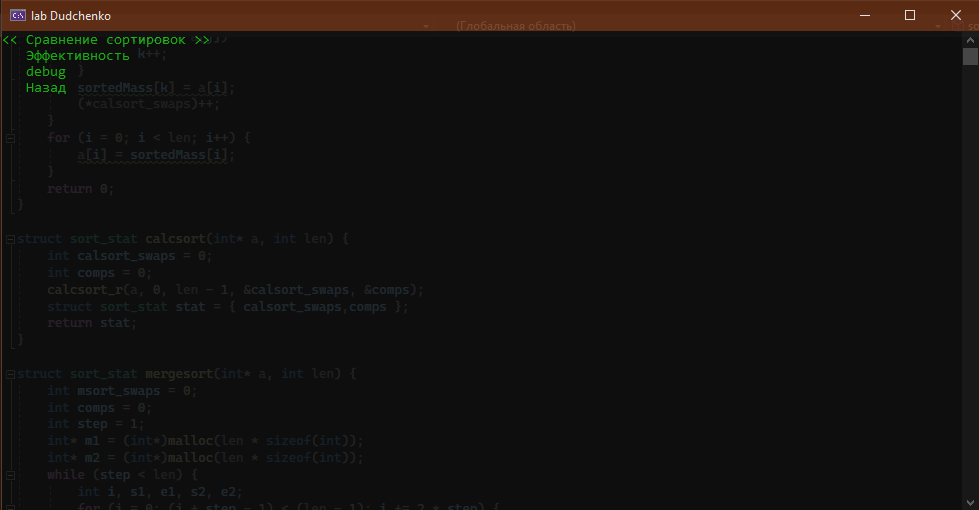


Рис. 11 Режим эксперимента

Сравнение сортировок.

Если перейти в << Сравнение сортировок >>, то перед нами появится таблица, в которой будут все виды сортировок. Если проанализировать данную таблицу, то можно будет понять, какая сортировка более эффективна. Мы узнаем количество перестановок, время работы каждой сортировки и замедление других видов сортировок.

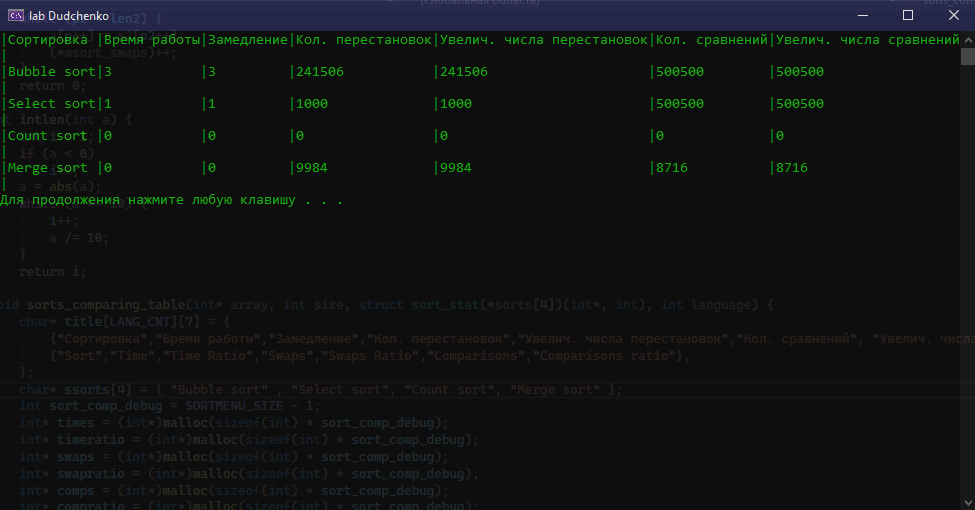


Рис.12 Сравнение сортировок

Был задан массив, в котором находилось 1000 элементов. По данным таблицы Рис.8 мы можем посмотреть, какая сортировка была более эффективна. Как мы видим, сортировка пузырьком оказалась менее эффективна, потому что показала наибольшее время.

Эффективность.

В пункте << Эффективность >> мы можем рассмотреть каждую сортировку отдельно. Нас попросят ввести цифру, которая будет соответствовать нужной нам сортировке.

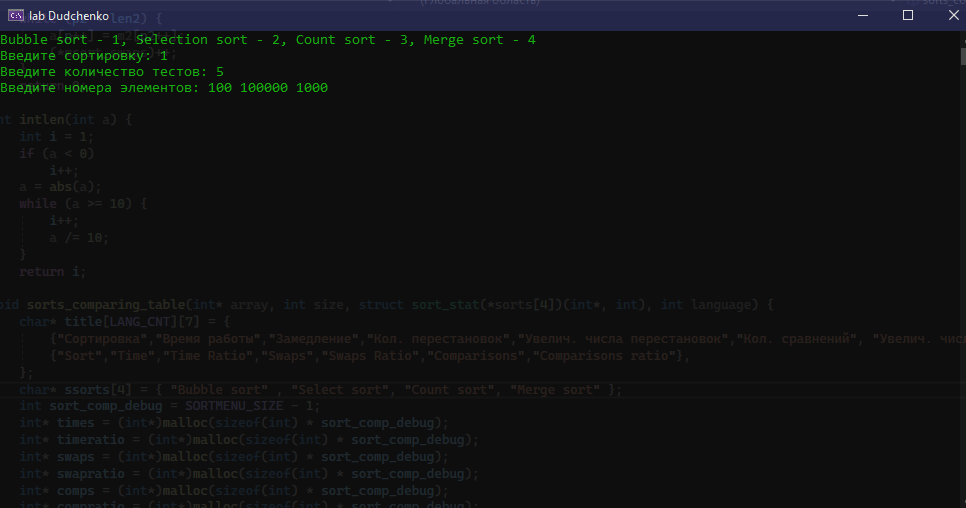


Рис. 13 Эффективность сортировки пузырьком

Мы выбрали сортировку пузырьком, а потом сортировку слиянием. Далее нас попросят ввести количество проверок.

Далее нам нужно ввести сами проверки (размеры данных).

Далее появляется таблица, в которой указано время работы и ускорение. Мы сможем узнать, как сильно размер массива влияет на каждую сортировку Рис. 15, 14.

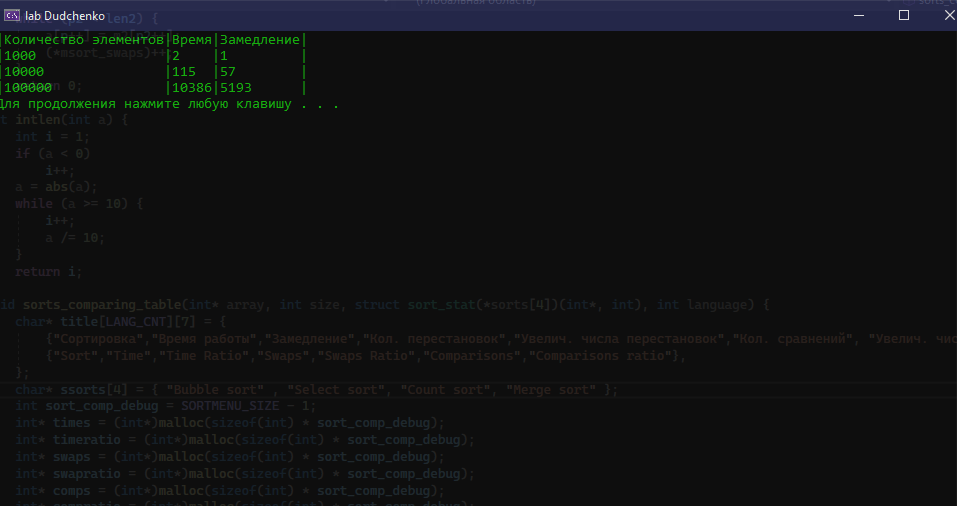


Рис. 14 Эффективность сортировки пузырьком

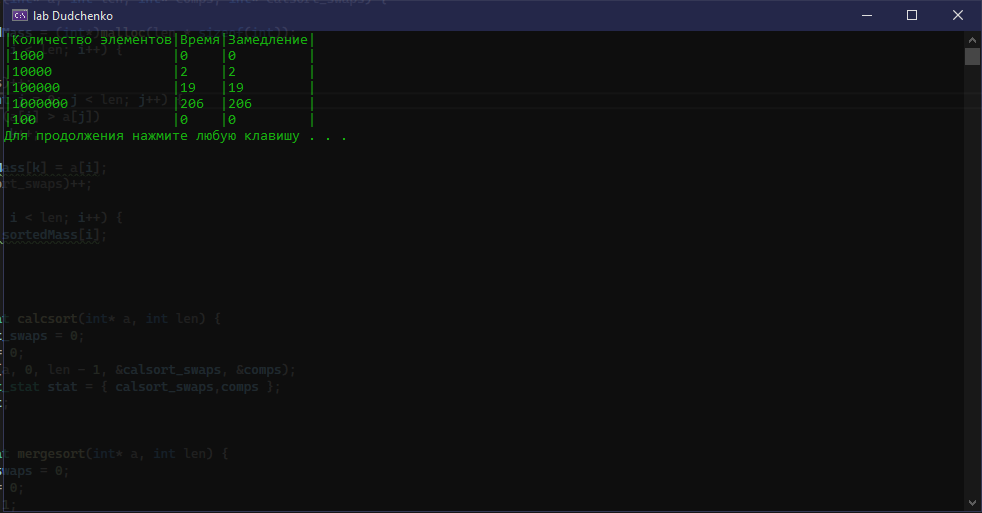


Рис. 15 Эффективность сортировки слиянием

Выход.

Соответственно, если мы выбираем пункт << Выход >>, то наша программа закроется Рис. 10.

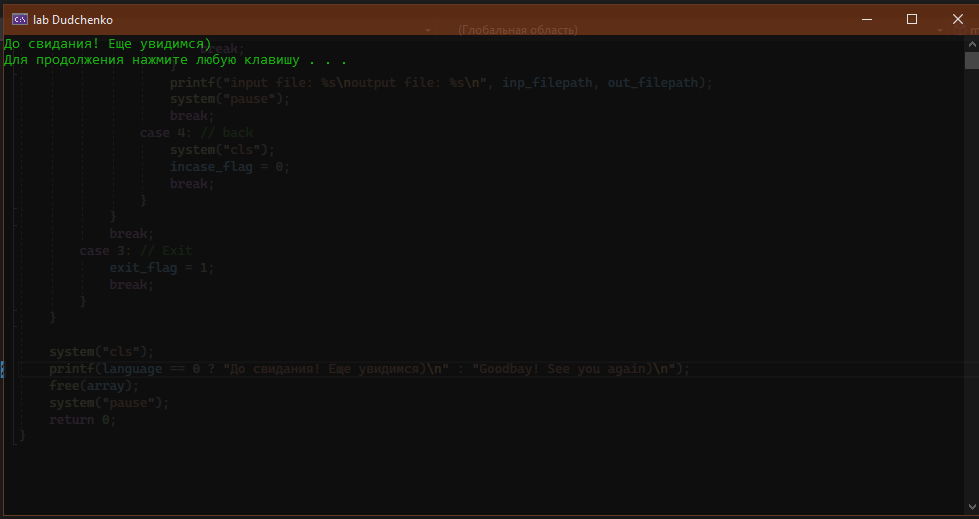


Рис. 10 Выход

**Руководство программиста**

Структура программы.

Структура программы довольно проста. В начале программы объявлены нужные библиотеки, написаны нужные функции. После идет main, в котором мы объявляем переменные и инициализируем их. После идет главные цикл while, который отвечает за главное меню, в нем другие циклы. После идет выход из программы. Рис. 11.

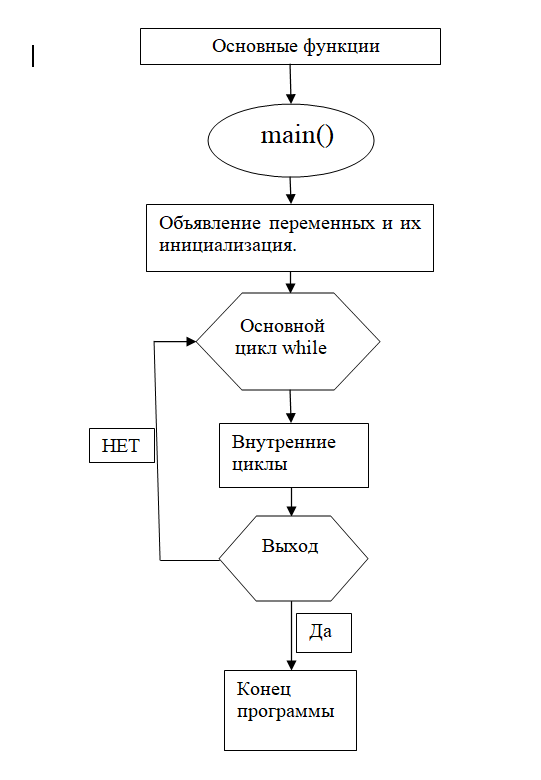


Рис. 11 Структура программы

Подключение и объявление констант.

С помощь #include подключаем нужные нам библиотеки для работы с ними [Приложение 1](#Приложение_1). С помощью #define объявляем нужные нам константы [Приложение 2](#Приложение_2).

Функции.

В данной программе всего 18 функций, 2 которые отвечают за поиск, 6 за сортировки. Остальные функции отвечают за то, чтобы задать массив, сгенерировать меню и.т.д. Также в программе присутствует структура.

Структура.

struct sorts\_stats {…}

Структура, которая содержит в себе два параметра.

**Парамерты:**

int swap\_count – количество перестановок в сортировке

int comp\_count – количество сравнений в сортировке

Описание функций.

void GenerateMenu(int\* choose\_pos, char\*\* menu, COORD\* cursorPos, HANDLE hStdOut, int menu\_size){…}

Данная функция отвечает за меню. Она отвечает за пункты меню, курсор, консоль.

**Парамерты:**

int\* choose\_pos – указатель на позицию курсора в меню

char\*\* menu – массив пунктов в меню

COORD\* cursorPos - координата курсора

HANDLE hStdOut – консоль

int menu\_size - размер меню

**Исключения:**

Для того, чтобы функция работала корректно, проверяйте параметры функции.

void initarray(int\*\* array, int\* size, int lang) {…}

Функция, которая определяет размер массива и выделяет под него память.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

int lang – язык программы

**Исключения:**

Может неверно выделить память, если ошибиться в разыменовании указателей. Если массив и размер не будут инициализированы в main(), то размер массива автоматически будет равен отрицательному числу, следовательно, программа будет работать некорректно.

void fillmass(int\* array, int size, int lang) {…}

Функция, которая отвечает за ручной ввода массива. Нам нужно будет ввести то количество элементов, которое мы ранее указали в функции initarray().

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

int lang – язык программы

**Исключения:**

Внимательно смотрите на то, какой размер массива вы указали. Т.к если вы размер указали 2, а введете 3 элемента массива, то последний элемент будет потерян.

void randarray(int\* array, int size, int lang){…}

Данная функция генерирует случайный массив. Мы вводим количество элементов, нас просят ввести минимальный и максимальный элемент, и массив заполняется числами из этого промежутка.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

int lang – язык программы

void printarray(int\* array, int size, int lang){…}

Данная функция печатает ранее объявленный массив. Простой цикл for, который выводит элементы массива.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

int lang – язык программы

**Исключения:**

Обязательно должен быть объявлен массив.

struct sorts\_stats BubbleSort(int\* array, int size){…}

Данная функция представляет собой сортировку пузырьком. Не самая быстрая, но одна из самых простых сортировок.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

**Исключения:**

Нужно следить за тем, чтобы массив был задан, иначе программа выведет ошибку.

struct sort\_stat viborsort(int\* array, int size{…}

Сортировка выбором. Проходим по массиву в поисках максимального элемента. Найденный максимум меняем местами с последним элементом.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

**Исключения:**

Нужно следить за тем, чтобы массив был ранее задан.

struct sorts\_stats ShellSort(int\* array, int size){…}

Сортировка Шелла. Данный алгоритм является усовершенствованной сортировкой вставками.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

**Исключения:**

Массив обязательно должен быть заранее задан.

struct sort\_stat mergesort(int\* a, int len){…}

Сортировка слиянием. Массив разделяется на несколько частей и рекурсивно сортируется каждая часть.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

**Исключения:**

Массив заранее должен быть заполнен.

void stupidsearch(int\* array, int size, int lang){…}

Наивный поиск. Вводим элемент массива, получаем его индекс или индексы, если их несколько. Довольно простой алгоритм.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

int lang – язык программы

**Исключения:**

Массив должен быть заранее задан.

void binsearch(int\* array, int size, int lang) {…}

Бинарный поиск. Алгоритм быстро работает, но минус в том, что он выводит только первый индекс элемента, который вы ввели.

Параметры:

int\* array – массив целых чисел

int size – размер массива

int lang – язык программы

**Исключения:**

Массив должен быть заранее объявление и отсортирован. Если массив не будет отсортирован, но данный вид поиска не будет работать.

void sorts\_comparing\_table(int\* array, int size, struct sort\_stat(\*sorts[4])(int\*, int), int language) {…}

Функция, которая сравнивает сортировки. 4 вида сортировки, в каждой считается время сортировки, количество перестановок и сравнений. После выводится таблица, в которой все указано.

**Параметры:**

int\* array – массив целых чисел

int size – размер массива

int (\*sorts[4])(int\*, int) – массив видов сортировок

int language – язык программы

void draw\_efficiency\_table(struct sort\_stat(\*sorts[4])(int\*, int), int language) {…}

Функция, которая определяет эффективность сортировки. Мы выбираем нужную нам сортировку, вводим размер данных и получаем таблицу, в которой показано время работы.

**Параметры:**

int (\*sorts[4])(int\*, int) – массив видов сортировок

int language – язык программы

int intlen(int a) {…}

Функция, которая считает длину целого числа.

Параметры:

int a – длина целого числа

Описание алгоритмов.

Наивный поиск.

**Основная идея алгоритма:** Наивный поиск заключается в том, что вы вводите тот элемент массива, который хотите найти, и через цикл программа каждый раз сравнивает элемент массива и ваше число.

**Описание алгоритма:** Значение **num** хранит в себе число, индекс которого мы хотим узнать. Значение **k** – количество элементов массива, которые равно **num**.

Изначально **k** = 0.

Запускается цикл for, который пробегает по всему массива. Внутреннее условие **if** следит за тем, равен элемент массива числу или нет. Если элемент равен число, то к переменной **k** прибавляется 1. И печатаем индексы. Если после того, как **for** пройдет по всему массиву и **k** будет равно 0, то программа напечатает, что элементы не найдены.

Реализация наивного поиска приведена в [Приложение 3](#Приложение_3).

**Пример работы алгоритма**: Найди индекс(ы) элемента, который равен 5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 5 | 6 | 5 | 6 | 9 | 0 | 3 |

I = 0 1 2 3 4 5 6 7 8 9

1. Цикл for проходит по массиву с индекса, который равен 0. **I** = 0.
2. Условие if сравнивает 1 и 5. Они не равны. **K** остается равной нулю.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 5 | 6 | 5 | 6 | 9 | 0 | 3 |

I = 0 1 2 3 4 5 6 7 8 9

1. Цикл доходит до элемента с индексом 3. Т.к 5 равно 5, следовательно к переменной **k** прибавляется 1 и печатается индекс 3. Дальше for доходит до индекса 5, элемент массива и число равно. Переменная **k** равна 2. В итоге мы получаем в ответе: 3 5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 5 | 6 | 5 | 6 | 9 | 0 | 3 |

I = 0 1 2 3 4 5 6 7 8 9

Бинарный поиск.

**Основная идея алгоритма:** Бинарный (двоичный) поиск заключается в том, что на каждом шаге массив делится на две части и в работе остаётся та часть массива, где находится искомый объект.

**Описание алгоритма:** Значение **left** хранит левый граничный индекс рассматриваемой части массива, **right** хранит правый индекс рассматриваемой части массива, **middle** – индекс текущего рассматриваемого элемента (середина отрезка [**left**, **right**]).

Изначально **left = 0, right = size - 1.**

Пока элемент не будет найден делаем следующие действия:

1. Вычислить индекс среднего элемента **middle = (left + right) / 2**.
2. Сравнить элемент массива с индексом middle и искомый элемент **num**:

* если элемент массива равен искомому элементу **num**, то мы нашли элемент,
* если искомый элемент больше элемента массива, то сдвигаем левую границу области поиска **left** в **middle + 1** (ищем в правом подмассиве).
* если искомый элемент меньше элемента массива, то сдвигаем правую границу области поиска **right** в **middle - 1** (ищем в левом подмассиве).

1. Если правая граница стала меньше левой – выход (элемента в массиве нет). Иначе - вернуться к пункту 1.

Если элемент был найден, то в печатаем переменную **middle** и **flag = 1,** если же элемент не найдет, то программа печатаем об этом. Реализация бинарного поиска представлена в [Приложение 4](#Приложение_4).

**Замечание:** перед тем, как использовать бинарный поиск, нужно отсортировать массив.

**Пример работы алгоритма:** Надо найти индекс элемента, который равен 5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 5 | 6 | 10 | 6 | 9 | 0 | 3 |

I = 0 1 2 3 4 5 6 7 8 9

1)Нужно отсортировать массив.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 | 3 | 4 | 5 | 6 | 6 | 9 | 10 |

I = 0 1 2 3 4 5 6 7 8 9

2) Находим значение middle = (9 + 0) / 2 = 4; Под индексом 4 находимся элемент массива, который не равен нашему числу. Сравниваем его с 5. Т.к 5 > 4, поэтому left + 1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 | 3 | 4 | 5 | 6 | 6 | 9 | 10 |

I = 0 1 2 3 4 5 6 7 8 9

1. middle = (1 + 9) / 2 = 5. Элемент с индексом 5 равен нашему числу. Печатаем переменную middle.

Сортировка пузырьком.

**Основная идея алгоритма:** идея пузырьковой сортировки заключается в том, чтобы с помощью двойного цикла for сравнивать 2 элемента массива и менять их местами.

**Описание алгоритма:** Создаем две переменных, которые будут отвечать за количество **swaps** и **comp**. Переменная temp будет хранить в себе значение, которому нужно поменять место. Пишем 2 цикла for:

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

2)for(int j = I; j < size; j++)

Если **mass[i]** > **mass[j],** то значение **mass[i]** сохраняем в temp, значение **mass[j]** сохраняем в **mass[i],** а значение, которое было ранее сохранено в temp помещаем в **mass[j].**

Реализация сортировки пузырьком представлена в [Приложение 5](#Приложение_5).

**Пример работы алгоритма:** Нужно отсортировать массив.

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 4 | 2 | 1 |

I=0 1 2 3

1)Сравниваем элементы с индексами 0 и 1. 3 > 4, поэтому все остается на местах. I++, j++

2)Смотрим элементы с индексами 1 и 2. 4 > 2, поэтому меняем их местами, а после и 4 с 1.

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 1 | 4 |

I=0 1 2 3

3)Циклы пойдут по второму кругу и так отсортируют наш массив.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |

I=0 1 2 3

Сортировка выбором.

**Основная идея алгоритма:** суть заключается в том, что мы найденный минимум меняем местами с первым элементом.

**Описание алгоритма:** Запускаем цикл **for** по всему массиву. В переменную **tempmin** сохраняем значение элемента массива **array[i].** Фиксируем **tminindex** минимальный индекс **i**. Пишем еще один цикл **for** начиная с **i**-ого элемента до **size** и если **tempmin** больше рассматриваемого элемента **array[k]**, где k – итератор второго цикла **for**, то меняем их местами и меняем **tminindex**. После, благодаря вспомогательной переменной меняем местами нынешний элемент массива **array[i]** с элементом массива с найденным **tminindex**.

Таким образом массив будет отсортирован. Реализация сортировки вставками реализована в [Приложение 6](#Приложение_6).

**Пример работы алгоритма:** Отсортировать массив.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 2 | 4 | 3 | 1 |

I = 0 1 2 3 4

1)Изначально мы фиксируем 5 как сравниваемый элемент, после доходим до 1 и меняем с 5 местами.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 3 | 5 |

I = 0 1 2 3 4

2) После, сравниваем 2 с последующими элементами и оставляем его на месте, т.к. он наименьший из последующих.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 3 | 5 |

I = 0 1 2 3 4

3)Следующим прогоном меняем 4 с 3 местами.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |

I = 0 1 2 3 4

Сортировка слиянием.

**Основная идея алгоритма:** идея в том, ищутся упорядоченные подмассивы.

и соединяются в общий упорядоченный подмассив.

**Описание алгоритма:** Делим массив пополам, каждый из них сортируем слиянием и потом соединяем оба массива. Каждый разделённый массив тоже нарезаем на два подмассива до тех пор, пока в каждом не окажется по одному элементу.

Здесь тоже используется рекурсия — то есть повторение алгоритма внутри самого алгоритма. Но это только один из элементов алгоритма.

Второй элемент — соединение отсортированных элементов между собой, причём тоже хитрым способом: раз оба массива уже отсортированы, то нам достаточно сравнивать элементы друг с другом по очереди и заносить в итоговый массив данные по порядку.

**Пример работы алгоритма:** Нужно отсортировать массив.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 1 | 9 | 2 | 3 | 10 | 6 | 8 | 11 | 12 |

1)Делим массив на два подмассива.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 1 | 9 | 2 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 6 | 8 | 11 | 12 |

2)Аналогично разбиваем подмассивы на меньшие подмассивы и сравниваем элементы в подмассивах.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 5 | 6 | 8 | 9 | 10 | 11 | 12 |

3) Когда меньший элемент из подмассива записывается обратно в массив, оставшийся элемент сравнивается с остальными, пока по итогу все элементы не запишутся.

Вычислительный эксперимент

Данная программа была написана с помощью среды программирования Visual Studio 2022 на языке C. Вычислительные эксперименты проводились на ПК с процессором Intel(R) Pentium(R) CPU G4560 3.50 GHz

Критериями оценки методов сортировки массивов будут являться:

* число перестановок
* число сравнений
* затраченное время

Описание вычислительного эксперимента.

Был сгенерирован случайный массив размером 10000, элементы которого находили в диапазоне [1, 10000]. С помощью каждой сортировки массив был отсортирован, посчитано время работы, количество перестановок и сравнений. В итоге вычислений мы получили таблицу, в которой показаны все полученные данные.

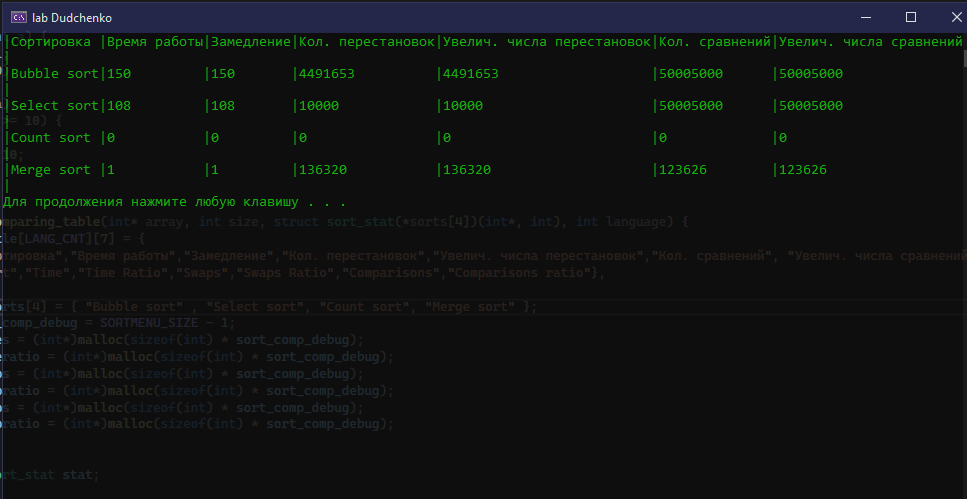


Рис. 16 Сравнение сортировок на массиве размером 5000

Короткий анализ: по данным таблицы мы можем увидеть, что более эффективной сортировкой оказалась сортировка слиянием, но сортировка выбором не сильно и по количеству сравнений и перестановок наравне. Пузырьковая сортировка оказалась самой неэффективной. Поэтом ее лучше использовать на массивах, размер которых меньше 100, т.к алгоритм достаточно простой.

Анализ эффективности алгоритмов.

По результатам вычислительного эксперимента можно сделать вывод, что самой эффективной сортировкой является сортировка слиянием. Она затратила наименьшее количество времени среди других видов сортировок. По итоговым данным видно, что количество перестановок у сортировки выбором намного меньше, чем у пузырьковой сортировки. Делаем вывод, что сортировка пузырьком самая неэффективная на массивах с большим размером.

Что, говоря о видах поиска, можно сказать, что бинарный поиск быстрее, чем наивный, но с помощью бинарного поиска мы можем узнать только первый индекс, в этом минус данного вида поиска. Если нам надо узнать все индексы, то наивный поиск в нашем случае будет очень полезен, т.к алгоритм достаточно простой.

**Заключение.**

В итоге лабораторной работы мы получаем, что сортировка слиянием самая эффективная и быстрая на массивах большого размера. Но сортировка выбором не сильно ей уступает. Благодаря данной лабораторной работе я многое узнал, научился анализировать данные, писать функции, пользоваться структурами. Одной из главных целей для меня было научиться писать читабельный код, чтобы его можно было понять. Я смогу достичь этой цели. Теперь я могу делать над массивом различные действия: сортировать, искать элементы. Также я научился взаимодействовать с файлами в программировании.

Литература

1. Кнут Д. Э. Искусство программирования. Том 1. Основныеалгоритмы = TheArtofComputerProgramming. Volume 1. FundamentalAlgorithms / подред. С. Г. Тригуб (гл. 1), Ю. Г. Гордиенко (гл. 2) иИ. В. Красикова (разд. 2.5 и 2.6). — 3. — Москва: Вильямс, 2002. — Т. 1. — 720 с. — ISBN 5-8459-0080-8.
2. Кнут Д. Э. Искусство программирования. Том 2. Получисленныеалгоритмы = The Art of Computer Programming. Volume 2. Seminumerical Algorithms / подред. Л. Ф. Козаченко (гл. 3, разд. 4.6.4 и 4.7), В. Т. Тертышного (гл. 4) и И. В. Красикова (разд. 4.6). — 3. — Москва: Вильямс, 2001. — Т. 2. — 832 с. — ISBN 5-8459-0081-6.
3. Кнут Д. Э. Искусство программирования. Том 3. Сортировкаипоиск = The Art of Computer Programming. Volume 3. Sorting and Searching / подред. В. Т. Тертышного (гл. 5) иИ. В. Красикова (гл. 6). — 2-еизд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
4. КнутД. Э. Искусствопрограммирования, том 4, A. Комбинаторныеалгоритмы, часть 1 = TheArtofComputerProgramming, Volume 4A: CombinatorialAlgorithms, Part 1 / подред. Ю. В. Козаченко. — 1. — Москва: Вильямс, 2013. — Т. 4. — 960 с. — ISBN 978-5-8459-1744-7.
5. Учебный курс «*Методы программирование-2: алгоритмы и структуры данных*». *© ГергельВ. П.*
6. Учебное пособие «Программа общего курса и описание лабораторных работ» © Балло Л.В., Барышева И.В., Гергель В.П., Гришагин В.А., Долгов Г.А., Кулакова А.П, Стронгин Р.Г.
7. Сайт algolist.manual .Сортировка выбором <http://algolist.manual.ru/sort/select_sort.php>
8. Сайт algolist.manual .Сортировка пузырьком <http://algolist.manual.ru/sort/bubble_sort.php>
9. Сайт habr https://habr.com/ru/company/edison/blog/472466/
10. Сайт thecode.media. Сортировка слиянием https://thecode.media/merge-sort/

**Приложения**

Приложение 1 – библиотеки.

#include <math.h>

#include <locale.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <windows.h>

#include <conio.h>

#include <time.h>

Приложение 2 – объявление констант.

#define KEY\_EXIT 27

#define KEY\_ENTER 13

#define ARROW\_KEY 224

#define KEY\_SPACE 0

#define KEY\_BACKSPACE 8

#define KEY\_ARROW\_RIGHT 77

#define KEY\_ARROW\_LEFT 75

#define KEY\_ARROW\_UP 72

#define KEY\_ARROW\_DOWN 80

#define ONE\_SIZE 1

#define TWO\_SIZE 2

#define THREE\_SIZE 3

#define FOUR\_SIZE 4

#define FIVE\_SIZE 5

#define RUS 0

#define ENG 1

Приложение 3 – наивный поиск.

void StupidSearch(int\* array, int size, int lang) {

system("cls");

if (size != -1) {

int number, k = 0;

scanf\_s(" %d", &number);

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

if (array[i] == number) {

printf("%d ", i);

k++;

}

}

if (k == 0) {

printf("Элемент не найден");

}

}

else {

printf("Массив не определен.");

}

}

Приложение 4 – бинарный поиск.

void BinarySearch(int\* array, int size, int lang) {

if (size != -1) {

int num;

printf("Введите число, индекс которого вы хотите узнать:");

scanf\_s(" %d", &num);

system("cls");

int left = 0, right = size - 1;

int flag = 0;

while (left <= right) {

int middle = (left + right) / 2;

if (array[middle] == num) {

printf("Индекс найденного элемента: %d\n", middle);

flag = 1;

break;

}

if (array[middle] > num)

right = middle - 1;

else

left = middle + 1;

}

if (flag != 1) {

printf("Такого элемента нет\n");

}

else {

printf("Массив не определен. Перейдите в <<Настройки>> и задайте массив.");

}

Приложение 5 – сортировка пузырьком.

struct sort\_stat bubble\_sort(int\* array, int size) {

int temp = 0;

int swaps = 0;

int comp = 0;

if (size != -1) {

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

for (int j = i; j < size; j++) {

comp++;

if (array[i] > array[j]) {

temp = array[i];

array[i] = array[j];

array[j] = temp;

swaps++;

}

}

}

}

struct sort\_stat stat = { swaps,comp };

return stat;

}

Приложение 6 – сортировка выбором.

struct sort\_stat viborsort(int\* array, int size) {

int tempmin, tminindex, temp, swaps = 0, comp = 0;

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

tempmin = array[i];

tminindex = i;

for (int k = i; k < size; k++) {

comp++;

if (tempmin > array[k]) {

tempmin = array[k];

tminindex = k;

}

}

temp = array[i];

array[i] = tempmin;

array[tminindex] = temp;

swaps++;

}

struct sort\_stat stat = { swaps,comp };

return stat;

}

Приложение 7 – сортировка слиянием.

struct sort\_stat mergesort(int\* a, int len) {

int msort\_swaps = 0;

int comps = 0;

int step = 1;

int\* m1 = (int\*)malloc(len \* sizeof(int));

int\* m2 = (int\*)malloc(len \* sizeof(int));

while (step < len) {

int i, s1, e1, s2, e2;

for (i = 0; (i + step - 1) < (len - 1); i += 2 \* step) {

s1 = i;

e1 = i + step - 1;

s2 = e1 + 1;

(i + 2 \* step - 1) < (len - 1) ? (e2 = i + 2 \* step - 1) : (e2 = len - 1);

merge\_two(a, s1, e1, s2, e2, m1, m2, &msort\_swaps, &comps);

}

step = step << 1;

}

struct sort\_stat stat = { msort\_swaps, comps };

return stat;

}

int merge\_two(int\* a, int s1, int e1, int s2, int e2, int\* m1, int\* m2, int\* msort\_swaps, int\* comps) {

int len1 = e1 - s1 + 1, len2 = e2 - s2 + 1;

int p1 = 0, p2 = 0, p = s1;

memcpy(m1, a + s1, sizeof(int) \* len1);

memcpy(m2, a + s2, sizeof(int) \* len2);

while (p1 < len1 && p2 < len2) {

(\*comps)++;

if (m1[p1] < m2[p2]) {

a[p++] = m1[p1++];

}

else {

a[p++] = m2[p2++];

}

(\*msort\_swaps)++;

}

while (p1 < len1) {

a[p++] = m1[p1++];

(\*msort\_swaps)++;

}

while (p2 < len2) {

a[p++] = m2[p2++];

(\*msort\_swaps)++;

}

return 0;

}