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: Прикладная математика и информатика

**Отчет по лабораторной работе №1**

**«Сравнение эффективности алгоритмов сортировки»**

**Выполнил**:студент группы 381806-1

Маркин Иван Максимович

**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

Основа информатики – работа с данными, их получение, обработка и хранение. Сортировка – одна из самых базовых операций над данными. Однако, при больших размерах данных этот элементарный процесс начинает занимать много времени и тратить много ресурсов. Поэтому алгоритмы сортировки требуют дальнейшего изучения и оптимизации, для сокращения времени выполнения и затрат ресурсов. Уже создано и описано множество методов сортировки, но они лишь продолжают прибывать и дорабатываться, что лишь подтверждает актуальность вопроса. Анализ алгоритмов сортировки и исследование их работы на разных массивах очень важно для отбора наиболее эффективных методов сортировки и их улучшения.

Постановка задачи

Необходимо создать удобную программу, в которой можно было бы протестировать алгоритмы сортировки и получить подробную информацию об их работе, после чего, сравнив полученные данные, сделать вывод об их эффективности.

Руководство пользователя

При запуске пользователь попадает в главное меню (Рисунок 1).

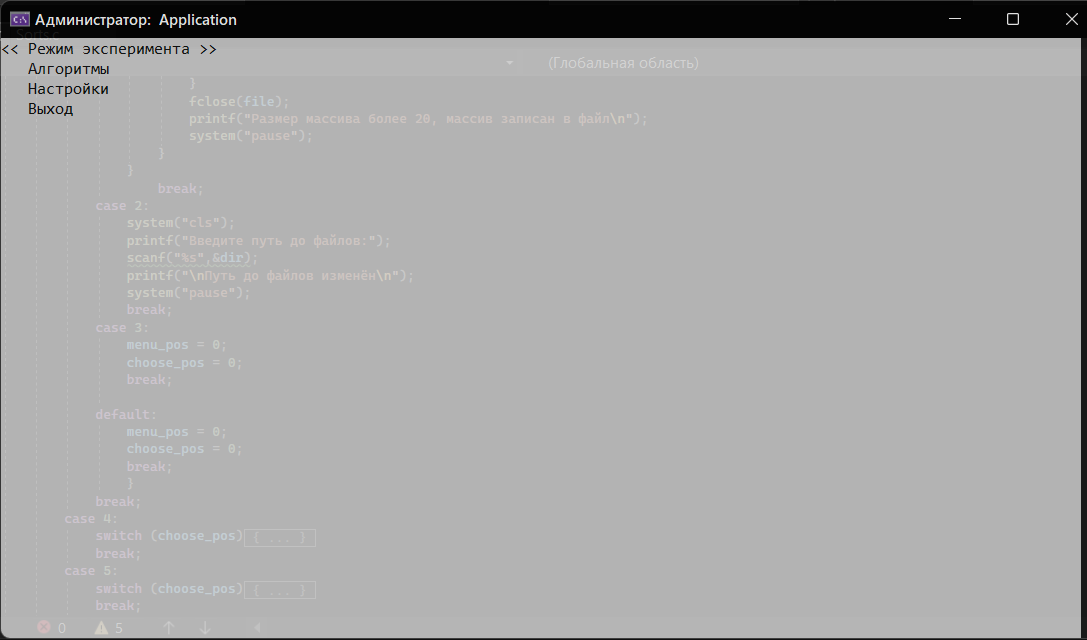


Рисунок . Главное меню

Первый пункт – режим эксперимента, позволит исследовать эффективность разных сортировок на массиве одного размера (пункт “Сравнение сортировок”) или же изучить работу одну сортировки на массивах разного размера(пункт “Эффективность”)(Рисунок 2).

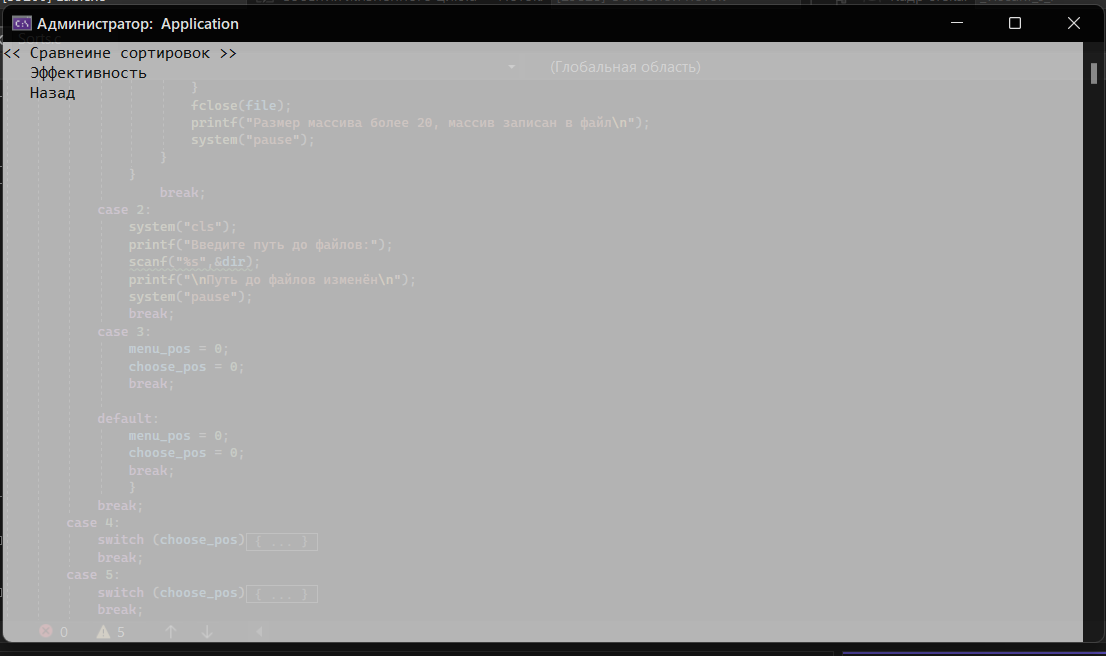


Рисунок . Меню эксперимента

Второй пункт – примение алгоритмов сортировки(пункт “Сортировка”, Рисунок 4) и поиска(пункт “Поиск”, Рисунок 5) к текущему массиву(Рисунок 3).

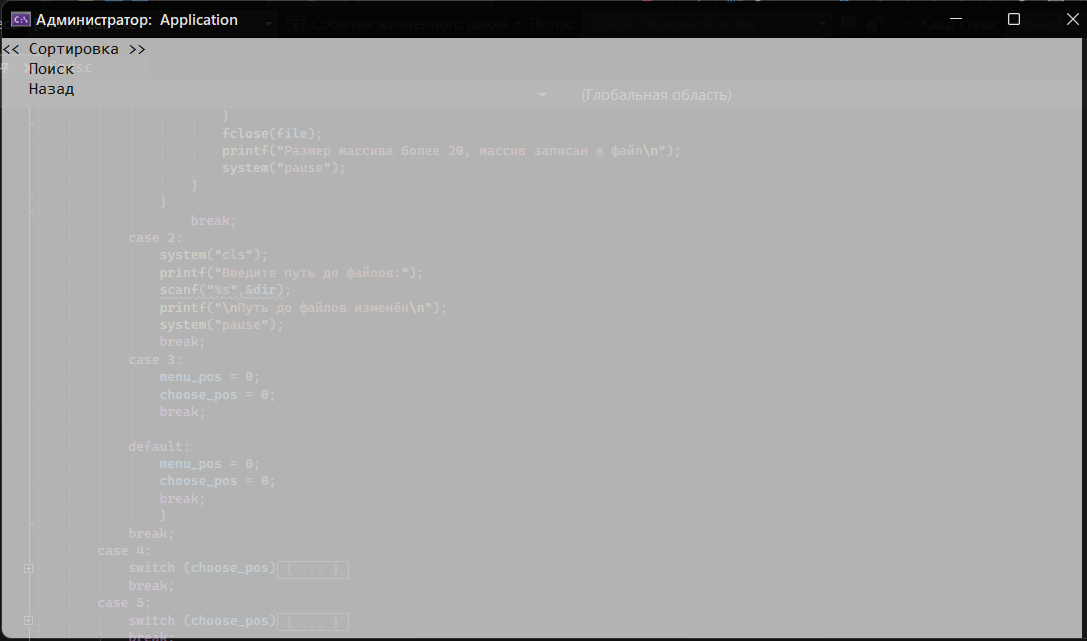


Рисунок . Меню алгоритмов

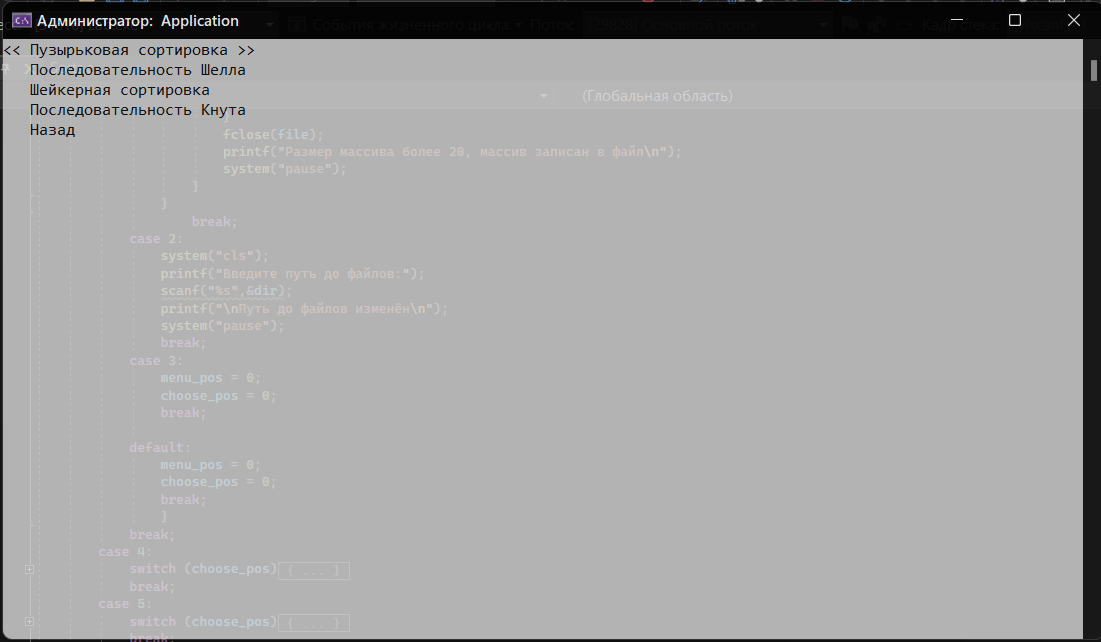


Рисунок . Меню сортировок

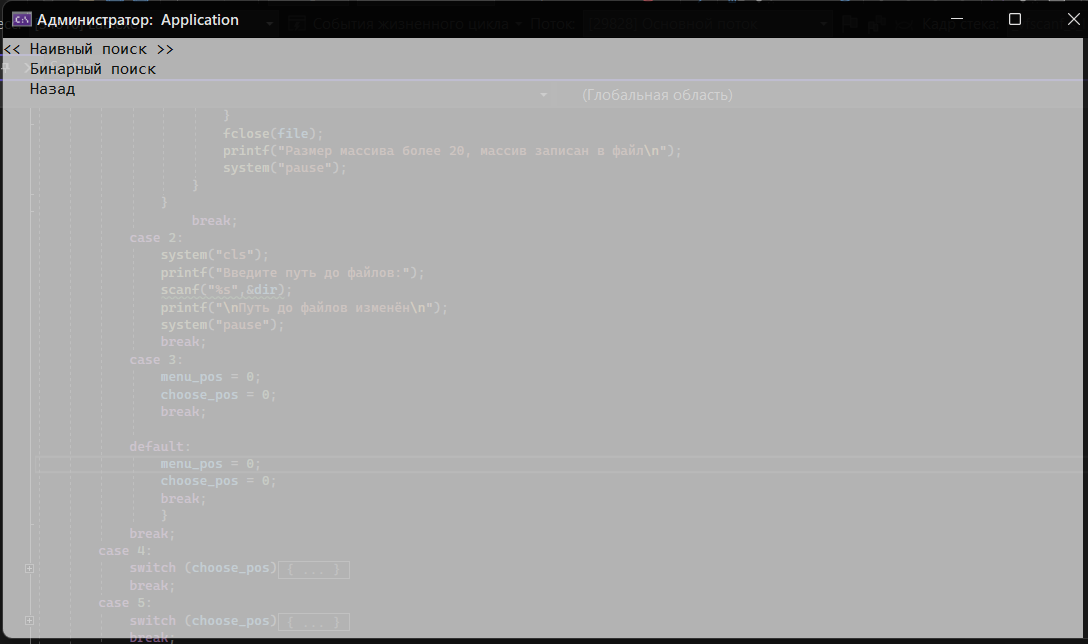


Рисунок . Меню поиска

Третий пункт – меню настроек(Рисунок 6), здесь есть возможность задать контрольный массив(пункт “Задать массив”), посмотреть содержимое актуального массива(пункт “Показать текущий массив”), или изменить директорию с файлами ввода и вывода(пункт “Изменить путь до рабочей директории”). Задать массив можно вручную, случайно или считав из файла (Рисунок 7). Вывод массива, по возможности, осуществляется на экран, но при большом количестве элементов значения записываются в файл(Рисунок 8).

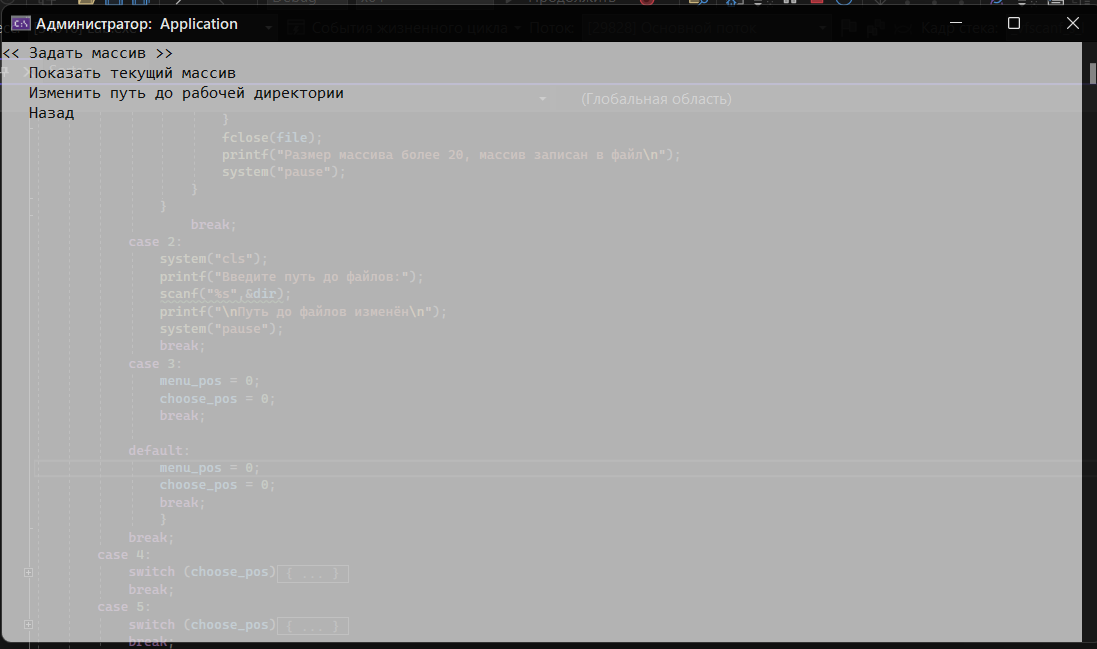


Рисунок . Меню настроек

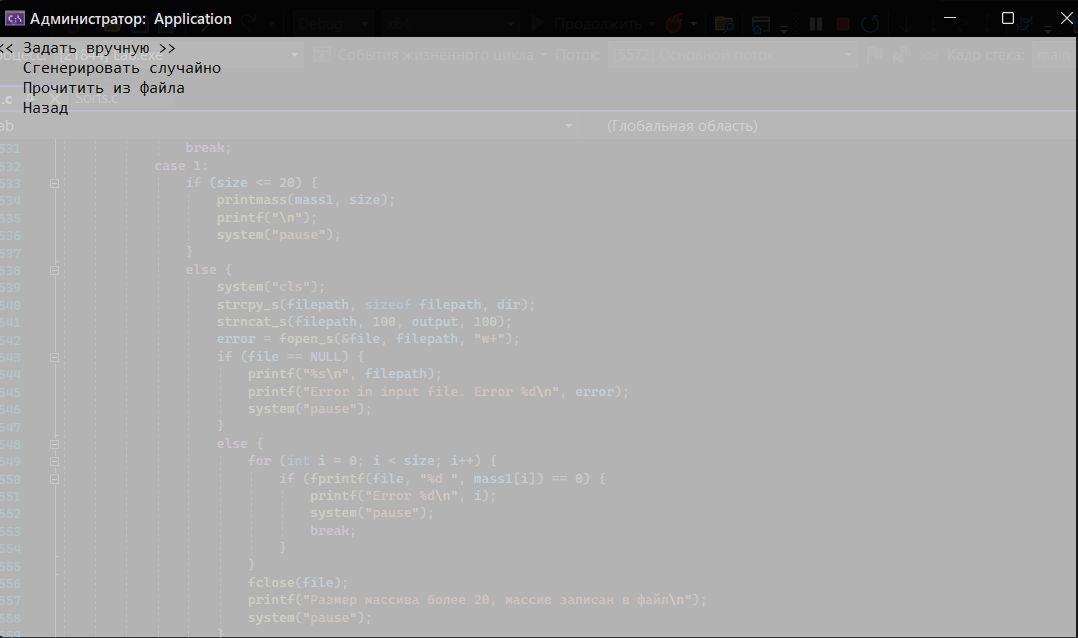


Рисунок . Меню задачи массива

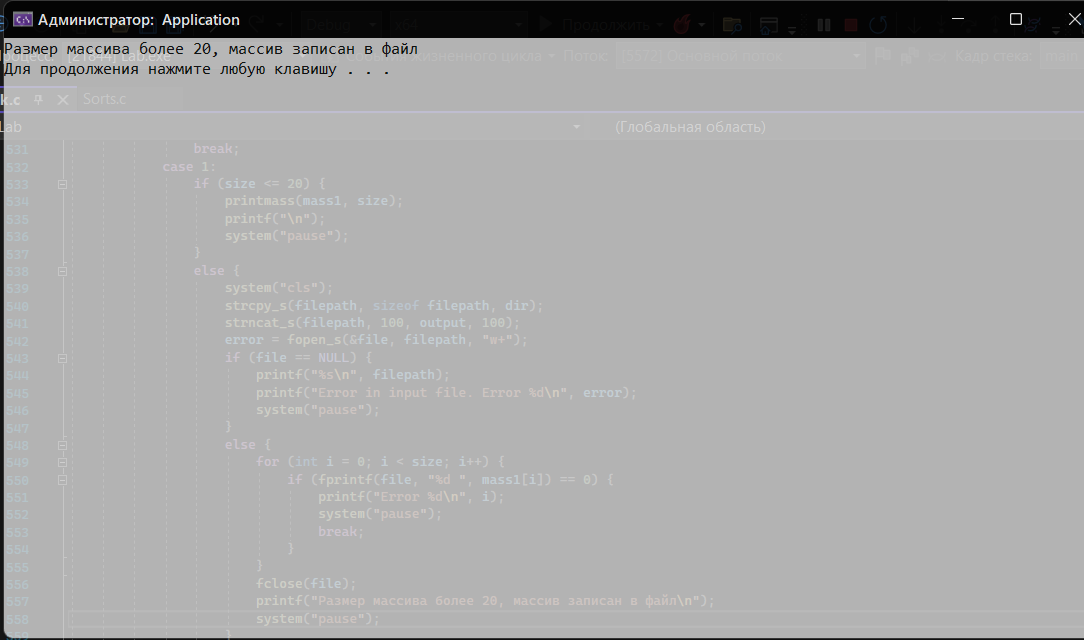


Рисунок . Вывод массива в файл

Руководство программиста

В начале проводятся какие-то общие факты о вашем программном модуле. Можно нарисовать блок-схему работы вашего программного комплекса.

Основная часть руководства программиста состоит из нескольких частей.

Описание структуры программы

В первой части приводится описание всех процедур и функций, реализованных в программе. Для функций описывается назначение, набор входящих и исходящих параметров. Рекомендуется в начале привести **граф вызовов процедур и функций**.

Пример описания функции:

void sorting(int\* arr, int\* result\_arr, int size, int\* time, int\* count\_comp, int\* count\_swap, void(\*sort)(int\*, int));

Функция, выполняющая сортировку массива.

**Параметры**

arr — указатель на исходный массив,

result\_arr — указатель на массив для сохранения отсортированного массива,

size — количество элементов массива,

time — время сортировки,

count\_comp — количество сравнений,

count\_swap — количество перестановок,

sort — указатель на используемую функцию сортировки.

Параметры time, count\_comp, count\_swap передаются по указателю для сохранения в них значений по выходу из функции.

**Исключения**

Может вызывать std::logic\_error при попытке отсортировать пустой массив.

**Заметки**

Передаваемые по указателю функции сортировки принимают на вход указатель на массив, который будет отсортирован на месте и размер массива и ничего не возвращают.

Описание алгоритмов

Во второй части приводится описание использованных или реализованных алгоритмов. Для описания алгоритма вначале приводится текстовое описание. Затем по необходимости приводятся блок схемы или псевдокод. В конце алгоритм разбирается на конкретном примере.

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. Приведём более строгий алгоритм.

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

Изначально left = 0, right = size.

Пока элемент не будет найден делаем следующие действия:

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

* если элемент массива равен искомому элементу value, то мы нашли элемент,
* если искомый элемент больше элемента массива, то сдвигаем левую границу области поиска left в base + 1 (ищем в правом подмассиве).
* если искомый элемент меньше элемента массива, то сдвигаем правую границу области поиска right в base - 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 переменную index сохраняем найденный индекс base, иначе значение index останется равным -1.

**Замечание.** Алгоритм будет работать правильно, только с отсортированным массивом. А если по случайности вы забыли отсортировать массив перед его использованием, то в большинстве случаев тот ответ, который подсчитал алгоритм, будет неверным.

Реализация алгоритма приведена в Приложение 1. Функция бинарного поиска.

**Пример работы алгоритма.** Необходимо найти элемент равный 54 в массиве

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 20 | 26 | 31 | 44 | 54 | 55 | 65 | 77 | 93 |
| left = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | right = 9 |

**1 шаг.** Выбираем базовый (опорный) элемент base = (9 + 0) / 2 = 4:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 20 | 26 | 31 | **44** | 54 | 55 | 65 | 77 | 93 |
| left = 0 | 1 | 2 | 3 | **base = 4** | 5 | 6 | 7 | 8 | right = 9 |

Сравниваем базовый элемент с искомым: 54 > 44, так как искомый больше – ищем в правом подмассиве.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 20 | 26 | 31 | 44 | 54 | 55 | 65 | 77 | 93 |
| 0 | 1 | 2 | 3 | 4 | left = 5 | 6 | 7 | 8 | right = 9 |

**2 шаг.** Выбираем опорный элемент:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 20 | 26 | 31 | 44 | 54 | 55 | **65** | 77 | 93 |
| 0 | 1 | 2 | 3 | 4 | left = 5 | 6 | **base = 7** | 8 | right = 9 |

Сравниваем базовый элемент с искомым: 54 < 64, так как искомый меньше – ищем в левом подмассиве.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 20 | 26 | 31 | 44 | 54 | 55 | 65 | 77 | 93 |
| 0 | 1 | 2 | 3 | 4 | left = 5 | right = 6 | 7 | 8 | 9 |

**3 шаг.** Выбираем опорный элемент. При сравнении получаем, что это искомый элемент.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 20 | 26 | 31 | 44 | **54** | 55 | 65 | 77 | 93 |
| 0 | 1 | 2 | 3 | 4 | **base = left = 5** | right = 6 | 7 | 8 | 9 |

Сортировка выбором

Сортировка пузырьком

Сортировка вставками

Сортировка Хоара

Вычислительный эксперимент

В этом разделе приводятся описание проводимых численных экспериментов и анализ полученных результатов.

Данная программа написана с помощью среды программирования Microsoft Visual Studio 2019 на языке C. Вычислительные эксперименты проводились на компьютере с процессором Intel Core i7 10750H CPU 2.6GHz.

Критериями оценки методов сортировки массивов будут являться:

1. число перестановок компонентов,
2. число сравнений компонентов,
3. затраченное время на сортировку массива.

Описание вычислительного эксперимента

Описание как конструировались тесовые наборы данных и проводились сами эксперименты.

Результаты вычислительного эксперимента

**Задача 1**

**Параметры задачи**

* длина массива: 5000,
* диапазон значений: [-5000, 5000].

**Результаты вычислительного эксперимента:**

Таблица 1 Сравнение количественных характеристик работы сортировок на массиве в 5000 элементов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Метод сортировки** | **Количество перестановок** | **Увеличение перестановок (раз)** | **Количество сравнений** | **Увеличение сравнений (раз)** | **Время (сек.)** | **Замедление (раз)** |
| Выбором | 4995 | 1.000 | 35735 | 175.011 | 0.042 | 35.482 |
| Пузырьком | 6254022 | 1252.100 | 6254022 | 6113.413 | 0.295 | 245.396 |
| Вставками | 6254022 | 1252.100 | 6254022 | 6113.413 | 0.035 | 29.722 |
| Хоара | 13919 | 2.800 | 1023 | 1.000 | 0.001 | 1.000 |

Провести короткий анализ промежуточных результатов

Анализ эффективности алгоритмов

На основе полученных результатов экспериментов сделать выводы о сложности и эффективности реализованных алгоритмов.

Заключение

В заключении обобщаются выводы, полученные в результате выполнения численных экспериментов, а также приводятся цели, которых удалось достичь в процессе выполнения лабораторной работы

Литература

Раздел отчета содержит список литературы, оформленный согласно рекомендациям ГОСТ.

Пример списка литературы:

1. Кнут Д. Э. Искусство программирования. Том 1. Основные алгоритмы = The Art of Computer Programming. Volume 1. Fundamental Algorithms / под ред. С. Г. Тригуб (гл. 1), Ю. Г. Гордиенко (гл. 2) и И. 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. Получисленные алгоритмы = The Art of Computer Programming. Volume 2. Seminumerical Algorithms / под ред. 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. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
4. Кнут Д. Э. Искусство программирования, том 4, A. Комбинаторные алгоритмы, часть 1 = The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1 / под ред. 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*». *© Гергель В. 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 studopedia. Сортировка вставками <https://studopedia.ru/5_159643_sortirovka-viborom.html>
10. Сайт intuit.ru. Сортировка Хоара <https://www.intuit.ru/studies/courses/648/504/lecture/11472?page=2>

Приложения

В качестве приложения приводится код. Как правило вставляется не весь код, а только существенно важные куски кода.

Приложение 1. Функция бинарного поиска

int binarySearch(int value, int size, int\* mass) {

int left = 0;

int right = size;

int index = -1;

while (left < right) {

int base = (left + right) / 2;

if (mass[base] < value) {

left = base + 1;

} else if (mass[base] > value) {

right = base - 1;

} else {

index = base;

break;

}

}

return index;

}