

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**институт математики и компьютерных технологий**

**Департамент информационных и компьютерных систем**

**ОТЧЕТ**

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

по теме «Внутренние сортировки»

**направление подготовки**

**09.03.03 Прикладная информатика**

**Прикладная информатика в экономике**

|  |  |
| --- | --- |
| Выполнил студент гр. Б9121 / 4 | |
|  |  |
|  | |
| Проверил ст. преподаватель | |
|  | Елсукова Е. А. |
|  | |
| не зачтено/зачтено | |

г. Владивосток

2022г.

Оглавление

[Условие задачи 3](#_Toc125459412)

[Функциональное описание 4](#_Toc125459413)

[Входные данные 4](#_Toc125459414)

[Структура программы 5](#_Toc125459415)

[Интерфейс программы 8](#_Toc125459416)

[Тестовые примеры 8](#_Toc125459417)

[Полученные результаты 11](#_Toc125459418)

[Приложение 12](#_Toc125459419)

Условие задачи

Освоение основных методов упорядочивания данных, расположенных в оперативной памяти.

1. Составить программу, состоящую из следующих пунктов:
2. Сортировка простыми вставками;
3. Сортировка простым обменом;
4. Сортировка простым выбором;
5. Сортировка простым обменом – Расческа;
6. Быстрая сортировка.
7. Характеристики сортировок;
8. Выход.

ВХОДНЫЕ ДАННЫЕ: чтение данных осуществляется из файла.

Выбор типа данных:

1. массив записей; (количество 10, 25) структура:   
   Дикий или Домашний, Кличка, Вид животного.
2. массив чисел (количество 15000).

ВЫХОДНЫЕ ДАННЫЕ: В пп. 1–5 результат выводится на экран.

В п. 6 результат записывается в файл и считывается из него на экран в виде таблицы 1.

Таблица1 – Пример результирующей таблицы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **вид сортировки** | **количество элементов** | **количество сравнений** | **количество перестановок** | **время выполнения** |

Функциональное описание

Входные данные

Для массива чисел. С помощью созданной функции (см. рисунок 1) заполняем файл случайными числовыми значениями. С помощью созданной функции (см. рисунок 2) заполняем массив числами из файла.

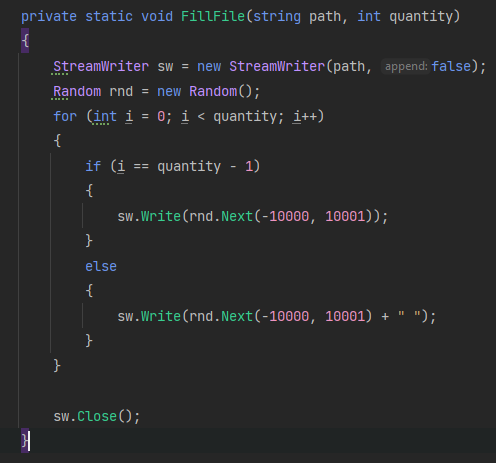


Рисунок 1 – Метод для заполнения файла

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Метод создания массива

Для массива записей.



Рисунок 3 – Инициализация массива записей

Структура программы

Программа реализован на c#, все сортировки в виде функций и находяться в отдельном абстрактном классе Sortings:

1. Сортировка простыми вставками

internal static void InsertionSort(int[] array)

1. Сортировка простым обменом

internal static void BubbleSort(int[] array)

1. Сортировка просты выбором

internal static void Selection Sort(int[] array)

1. Сортировка простым обменом – Гномья

internal static void GnomeSort(int[] array)

Данная сортировка имеет основу от сортировки простым выбором. Алгоритм заключается в том, что сравнивается текущий и предыдущий элементы: если они в правильном порядке, то происходит переход на один элемент вперед, иначе элементы меняются местами и происходит переход на один элемент назад (см. рисунок 4, 5). В сортировки простым выбором сравниваются рядом стоящие элементы и в случае неправильного расположения меняются.

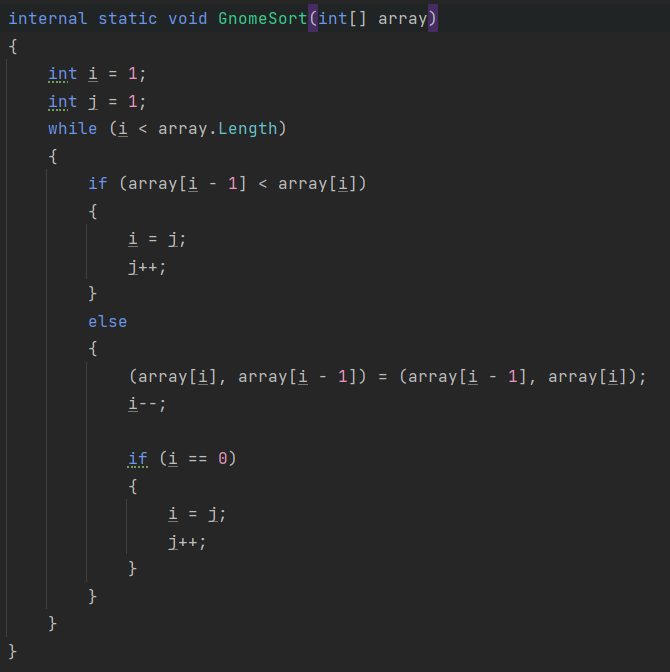


Рисунок 4 – Реализация Гномьей сортировки (массив чисел).

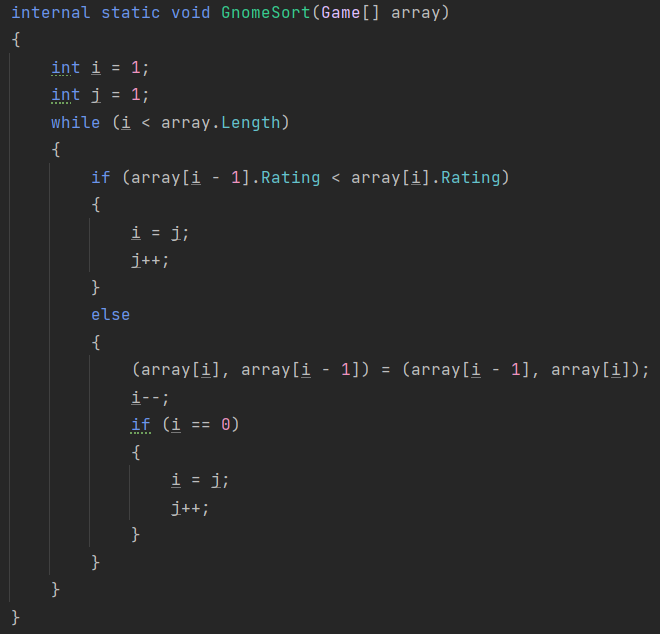


Рисунок 5 –Реализация Гномьей сортировки (массив записей)

1. Быстрая сортировка

internal static void QuickSort(int[] array)

1. Характеристики

Считается все характеристики и записываются в файл, после чего значения считываются из файла и выводятся на экран в виде таблицы (см. рисунок 6).

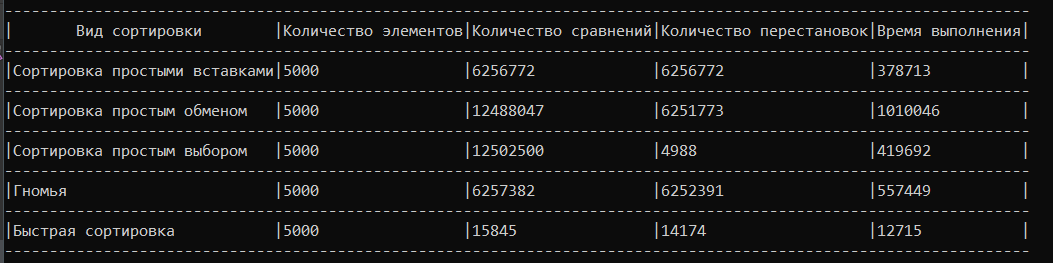


Рисунок 6 –Характеристики после сортировки 5000 элементов

Интерфейс программы

Интерфейс состоит из двух уровней на первом можно выбрать тип и размер массива (см. рисунок 7). На втором вид сортировки (см. рисунок 8).

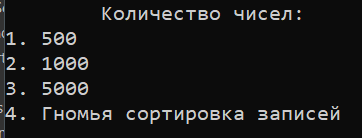


Рисунок 7 – Первый уровень интерфейса

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – Второй уровень интерфейса

Тестовые примеры

Приведены результаты функционала приложения.

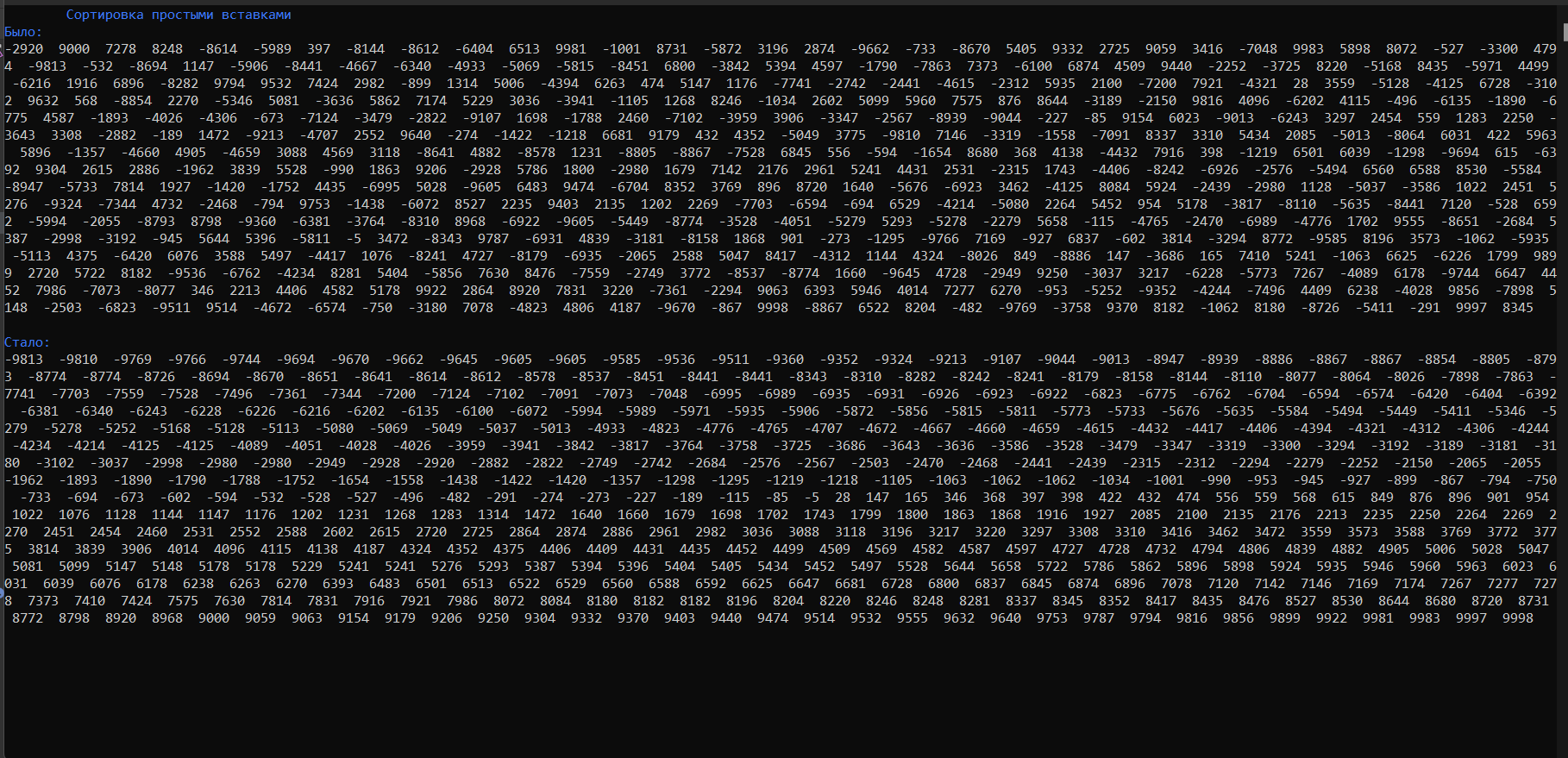


Рисунок 9 – Результат Сортировки простыми вставками

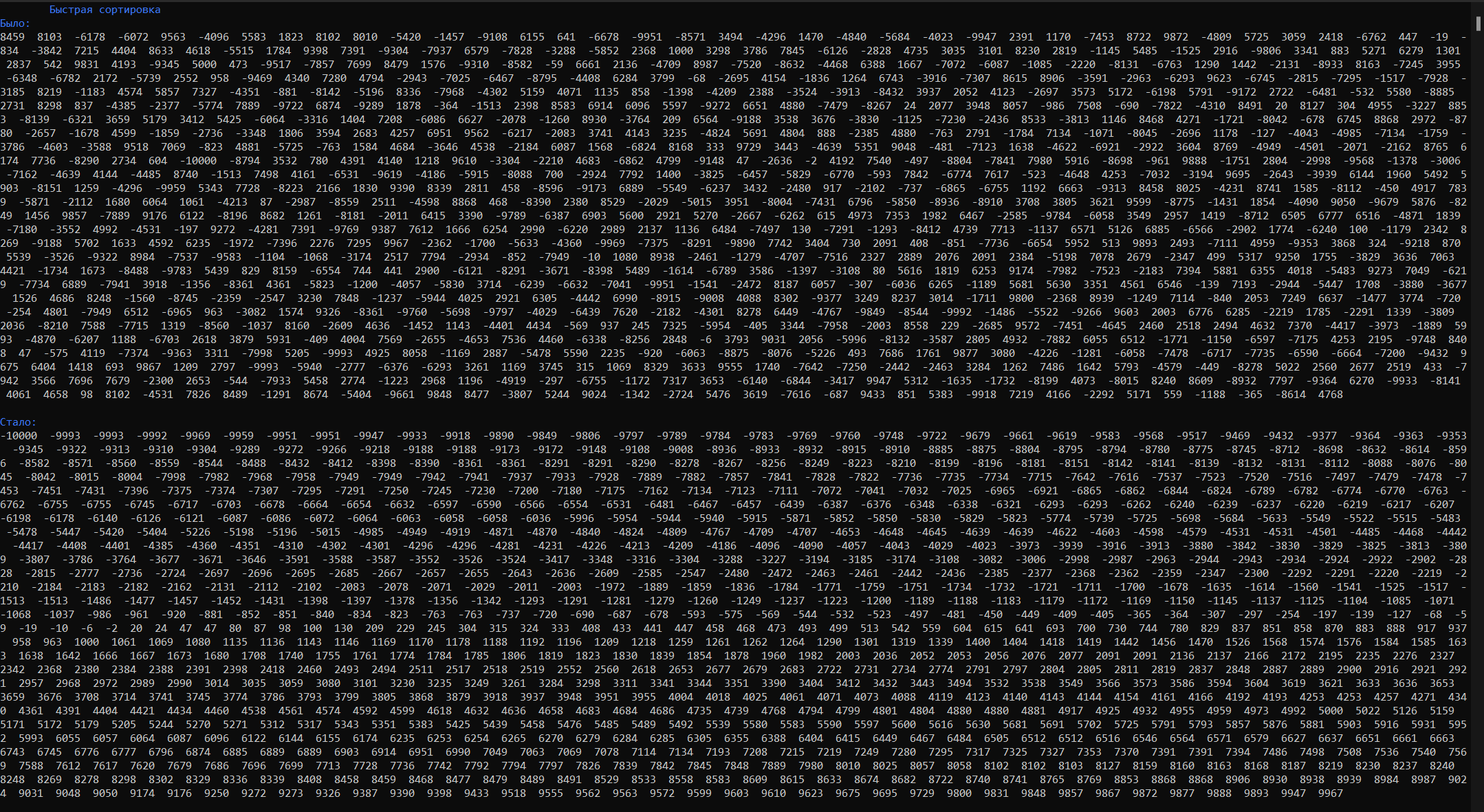


Рисунок 10 – Результат Быстрой сортировки

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Результат сортировки записей Гномьей сортировкой

Полученные результаты

Получены характеристики разных видов сортировок с разными видами данных. Сравнив характеристики можно оценить каждую сортировку. Сортировка простыми вставками занимает второе место по скорости работы, что доказывает ее эффективность. Сортировка простым обменом является самой медленной из-за очень большого количества сравнений и перестановок. Сортировка простым выбором имеет такое же количество сравнений, как и сортировка простым обменом, но из-за того, что количество перестановок в тысячу раз меньше, значительно уменьшает время, потраченное на сравнения. Гномья сортировка является модификацией сортировки простым выбором, а именно в ней оптимизировано количество сравнений, что делает ее быстрее чем сортировку простым обменом. Быстрая сортировка является самой быстрой.

Приложение

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using Microsoft.Office.Interop.Excel;

namespace DifferentSortings

{

internal class Program

{

public static int qunt = 0;

private static void FillFile(string path, int quantity)

{

StreamWriter sw = new StreamWriter(path, false);

Random rnd = new Random();

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

{

if (i == quantity - 1)

{

sw.Write(rnd.Next(-10000, 10001));

}

else

{

sw.Write(rnd.Next(-10000, 10001) + " ");

}

}

sw.Close();

}

private static int[] CreateArray(string path)

{

StreamReader sr = new StreamReader(path);

string numbers = sr.ReadLine();

sr.Close();

var array = numbers.Split(' ').Where(x => !string.IsNullOrWhiteSpace(x)).Select(x => int.Parse(x)).ToArray();

return array;

}

private static void WriteArray(int[] array)

{

foreach (var i in array)

{

Console.Write(i + " ");

}

Console.WriteLine();

}

private static int[] SelectSizeArrayOrStructs(ref int quant)

{

int[] originalArray = new int[]{};

Console.Clear();

Console.WriteLine("\tКоличество чисел:\n" +

"1. 500\n" +

"2. 1000\n" +

"3. 5000\n" +

"4. Гномья сортировка записей ");

switch (Console.ReadKey().Key)

{

case ConsoleKey.D1:

File.WriteAllText(@"..\..\Data\500num.txt", string.Empty);

FillFile(@"..\..\Data\500num.txt", 500);

quant = 500;

originalArray = CreateArray(@"..\..\Data\500num.txt");

break;

case ConsoleKey.D2:

File.WriteAllText(@"..\..\Data\1000num.txt", string.Empty);

FillFile(@"..\..\Data\1000num.txt", 1000);

quant = 1000;

originalArray = CreateArray(@"..\..\Data\1000num.txt");

break;

case ConsoleKey.D3:

File.WriteAllText(@"..\..\Data\5000num.txt", string.Empty);

FillFile(@"..\..\Data\5000num.txt", 5000);

quant = 5000;

originalArray = CreateArray(@"..\..\Data\5000num.txt");

break;

case ConsoleKey.D4:

int count = 0;

var sr = new StreamReader(@"..\..\Data\Datat.txt");

while ((sr.ReadLine()) != null)

{

count++;

}

sr.Close();

var textInArray = new string[count];

var structsArray = new Game[count / 3];

var sr2 = new StreamReader(@"..\..\Data\Datat.txt");

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

{

textInArray[i] = sr2.ReadLine();

}

sr2.Close();

for (int i = 0, j = 0; i < count; i += 3, j ++)

{

structsArray[j] = new Game(textInArray[i], textInArray[i+1], int.Parse(textInArray[i+2]));

}

Sortings.GnomeSort(structsArray);

Console.Clear();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(0, 0);

Console.Write("Название игры");

Console.SetCursorPosition(40, 0);

Console.Write("Жанр");

Console.SetCursorPosition(80, 0);

Console.Write("Рейтинг");

Console.ResetColor();

for (int i = count / 3; i > 0; i--)

{

Console.SetCursorPosition(0, count / 3 - i + 1);

Console.Write(structsArray[i - 1].Name);

Console.SetCursorPosition(40, count / 3 - i + 1);

Console.Write(structsArray[i - 1].GameType);

Console.SetCursorPosition(80, count / 3 - i + 1);

Console.Write(structsArray[i - 1].Rating);

}

Console.ReadKey();

Environment.Exit(0);

SelectSizeArrayOrStructs(ref quant);

break;

default:

SelectSizeArrayOrStructs(ref quant);

break;

}

return originalArray;

}

static void WriteInFile(int swap, int comparison, object time)

{

StreamWriter sw = new StreamWriter(@"..\..\Data\characteristics.txt", true);

sw.Write($"{comparison} {swap} {time} \n");

sw.Close();

}

private static void Refresh(int[] originalArray, int[] array, ref int counterSwap, ref int counterComparison)

{

Array.Copy(originalArray, array, originalArray.Length);

counterComparison = 0;

counterSwap = 0;

}

private static void Refresh(int[] originalArray, int[] array)

{

Array.Copy(originalArray, array, originalArray.Length);

}

public static void Main(string[] args)

{

Console.CursorVisible = false;

int[] originalArray = SelectSizeArrayOrStructs(ref qunt);

int[] fakeArray = new int[originalArray.Length];

Array.Copy(originalArray, fakeArray, originalArray.Length);

int counterSwap = 0;

int counterСomparison = 0;

bool exit = true;

while(exit)

{

Console.Clear();

Console.WriteLine("\tОпции:\n" +

"1. Сортировка простыми вставками\n" +

"2. Сортировка простым обменом\n" +

"3. Сортировка простым выбором\n" +

"4. Гномья\n" +

"5. Быстрая сортировка\n" +

"6. Характеристики сортировок\n" +

"7. Выход\n");

switch (Console.ReadKey().Key)

{

case ConsoleKey.D1:

Console.Clear();

Refresh(originalArray, fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\tСортировка простыми вставками\nБыло:");

Console.ResetColor();

WriteArray(fakeArray);

Sortings.InsertionSort(fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\nСтало:");

Console.ResetColor();

WriteArray(fakeArray);

Console.ReadKey();

break;

case ConsoleKey.D2:

Console.Clear();

Refresh(originalArray, fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\tСортировка простыми обменом\nБыло:");

Console.ResetColor();

WriteArray(fakeArray);

Sortings.BubbleSort(fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\nСтало:");

Console.ResetColor();

WriteArray(fakeArray);

Console.ReadKey();

break;

case ConsoleKey.D3:

Console.Clear();

Refresh(originalArray, fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\tСортировка простым выбором\nБыло:");

Console.ResetColor();

WriteArray(fakeArray);

Sortings.SelectionSort(fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\nСтало:");

Console.ResetColor();

WriteArray(fakeArray);

Console.ReadKey();

break;

case ConsoleKey.D4:

Console.Clear();

Refresh(originalArray, fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\tГномья\nБыло:");

Console.ResetColor();

WriteArray(fakeArray);

Sortings.GnomeSort(fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\nСтало:");

Console.ResetColor();

WriteArray(fakeArray);

Console.ReadKey();

break;

case ConsoleKey.D5:

Console.Clear();

Refresh(originalArray, fakeArray);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\tБыстрая сортировка\nБыло:");

Console.ResetColor();

WriteArray(fakeArray);

Sortings.QuickSort(fakeArray, 0, fakeArray.Length - 1);

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("\nСтало:");

Console.ResetColor();

WriteArray(fakeArray);

Console.ReadKey();

break;

case ConsoleKey.D6:

Console.Clear();

File.WriteAllText(@"..\..\Data\characteristics.txt", string.Empty);

var time = new Stopwatch();

time.Start();

Sortings.InsertionSort(fakeArray, ref counterSwap, ref counterСomparison);

time.Stop();

WriteInFile(counterSwap, counterСomparison, time.ElapsedTicks);

Refresh(originalArray, fakeArray, ref counterSwap, ref counterСomparison);

time.Restart();

time.Start();

Sortings.BubbleSort(fakeArray, ref counterSwap, ref counterСomparison);

time.Stop();

WriteInFile(counterSwap, counterСomparison, time.ElapsedTicks);

Refresh(originalArray, fakeArray, ref counterSwap, ref counterСomparison);

time.Restart();

time.Start();

Sortings.SelectionSort(fakeArray, ref counterSwap, ref counterСomparison);

time.Stop();

WriteInFile(counterSwap, counterСomparison, time.ElapsedTicks);

Refresh(originalArray, fakeArray, ref counterSwap, ref counterСomparison);

time.Restart();

time.Start();

Sortings.GnomeSort(fakeArray, ref counterSwap, ref counterСomparison);

time.Stop();

WriteInFile(counterSwap, counterСomparison, time.ElapsedTicks);

Refresh(originalArray, fakeArray, ref counterSwap, ref counterСomparison);

time.Restart();

time.Start();

Sortings.QuickSort(fakeArray, 0, fakeArray.Length - 1, ref counterSwap, ref counterСomparison);

time.Stop();

WriteInFile(counterSwap, counterСomparison, time.ElapsedTicks);

Refresh(originalArray, fakeArray, ref counterSwap, ref counterСomparison);

Table.Print();

Console.ReadLine();

break;

case ConsoleKey.D7:

Console.Clear();

exit = false;

break;

default:

break;

}

}

}

}

}

namespace DifferentSortings

{

public abstract class Sortings

{

internal static void InsertionSort(int[] array)

{

for (int i = 1; i < array.Length; i++)

{

int barrier = array[i];

int j = i - 1;

while (j >= 0 && array[j] > barrier)

{

array[j + 1] = array[j];

j--;

}

array[j + 1] = barrier;

}

}

internal static void InsertionSort(int[] array, ref int counterSwap, ref int counterСomparison)

{

for (int i = 1; i < array.Length; i++)

{

int barrier = array[i];

int j = i - 1;

counterСomparison++;

while (j >= 0 && array[j] > barrier)

{

counterСomparison++;

array[j + 1] = array[j];

counterSwap++;

j--;

}

array[j + 1] = barrier;

counterSwap++;

}

}

internal static void BubbleSort(int[] array)

{

bool t = true;

int j = array.Length - 1;

while (t)

{

t = false;

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

if (array[i] > array[i + 1])

{

t = true;

(array[i], array[i + 1]) = (array[i + 1], array[i]);

}

j--;

}

}

internal static void BubbleSort(int[] array, ref int counterSwap, ref int counterСomparison)

{

bool t = true;

int j = array.Length - 1;

while (t)

{

t = false;

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

{

counterСomparison++;

if (array[i] > array[i + 1])

{

t = true;

(array[i], array[i + 1]) = (array[i + 1], array[i]);

counterSwap++;

}

}

j--;

}

}

internal static void SelectionSort(int[] array)

{

for (int i = array.Length - 1; i >= 0; i --)

{

int max = array[0];

int k = 0;

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

{

if (max < array[j])

{

max = array[j];

k = j;

}

}

if (max > array[i])

(array[i], array[k]) = (array[k], array[i]);

}

}

internal static void SelectionSort(int[] array, ref int counterSwap, ref int counterComparison)

{

for (int i = array.Length - 1; i >= 0; i --)

{

int max = array[0];

int k = 0;

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

{

counterComparison++;

if (max < array[j])

{

max = array[j];

k = j;

}

}

counterComparison++;

if (max > array[i])

{

counterSwap++;

(array[i], array[k]) = (array[k], array[i]);

}

}

}

internal static void GnomeSort(int[] array)

{

int i = 1;

int j = 1;

while (i < array.Length)

{

if (array[i - 1] < array[i])

{

i = j;

j++;

}

else

{

(array[i], array[i - 1]) = (array[i - 1], array[i]);

i--;

if (i == 0)

{

i = j;

j++;

}

}

}

}

internal static void GnomeSort(Game[] array)

{

int i = 1;

int j = 1;

while (i < array.Length)

{

if (array[i - 1].Rating < array[i].Rating)

{

i = j;

j++;

}

else

{

(array[i], array[i - 1]) = (array[i - 1], array[i]);

i--;

if (i == 0)

{

i = j;

j++;

}

}

}

}

internal static void GnomeSort(int[] array, ref int counterSwap, ref int counterComparison)

{

int i = 1;

int j = 1;

while (i < array.Length)

{

counterComparison++;

if (array[i - 1] < array[i])

{

i = j;

j++;

}

else

{

counterSwap++;

(array[i], array[i - 1]) = (array[i - 1], array[i]);

i--;

if (i == 0)

{

i = j;

j++;

}

}

}

}

internal static void QuickSort(int[] array, int ilo, int ihi)

{

int lo = ilo;

int hi = ihi;

int mid = array[(ilo + ihi) / 2];

do

{

while (array[lo] < mid) lo++;

while (array[hi] > mid) hi--;

if (lo > hi)

{

break;

}

if (array[lo] > array[hi])

(array[lo], array[hi]) = (array[hi], array[lo]);

lo++;

hi--;

} while (lo <= hi);

if (lo < ihi)

QuickSort(array, lo, ihi);

if (ilo < hi)

QuickSort(array, ilo, hi);

}

internal static void QuickSort(int[] array, int ilo, int ihi, ref int counterSwap, ref int counterComparison)

{

int lo = ilo;

int hi = ihi;

int mid = array[(ilo + ihi) / 2];

do

{

while (array[lo] < mid) lo++;

while (array[hi] > mid) hi--;

if (lo > hi)

{

break;

}

counterComparison++;

if (array[lo] > array[hi])

{

counterSwap++;

(array[lo], array[hi]) = (array[hi], array[lo]);

}

lo++;

hi--;

} while (lo <= hi);

if (lo < ihi)

QuickSort(array, lo, ihi, ref counterSwap, ref counterComparison);

if (ilo < hi)

QuickSort(array, ilo, hi, ref counterSwap, ref counterComparison);

}

}

}

using System;

using System.IO;

namespace DifferentSortings

{

public abstract class Table

{

public static void Print()

{

StreamReader sr = new StreamReader(@"..\..\Data\characteristics.txt");

string[] data = (sr.ReadLine() + sr.ReadLine() + sr.ReadLine() + sr.ReadLine() + sr.ReadLine()).Split(' ');

sr.Close();

Title();

Line("Сортировка простыми вставками", Convert.ToString(DifferentSortings.Program.qunt), data[0], data[1], data[2]);

Line("Сортировка простым обменом", Convert.ToString(DifferentSortings.Program.qunt), data[3], data[4], data[5]);

Line("Сортировка простым выбором", Convert.ToString(DifferentSortings.Program.qunt), data[6], data[7], data[8]);

Line("Гномья", Convert.ToString(DifferentSortings.Program.qunt), data[9], data[10], data[11]);

Line("Быстрая сортировка", Convert.ToString(DifferentSortings.Program.qunt), data[12], data[13], data[14]);

}

private static void Title()

{

Console.Write(

$"{new string('-', 114)}\n" +

$"|{new string(' ', 7)}Вид сортировки{new string(' ', 8)}|Количество элементов|Количество сравнений|Количество перестановок|Время выполнения|\n" +

$"{new string('-', 114)}\n");

}

private static void Line(string name, string quantity, string comparison, string swap, string time)

{

Console.Write(

$"|{name}{new string(' ', 29 - name.Length)}" +

$"|{quantity}{new string(' ', 20 - quantity.Length)}" +

$"|{comparison}{new string(' ', 20 - comparison.Length)}" +

$"|{swap}{new string(' ', 23 - swap.Length)}" +

$"|{time}{new string(' ', 16 - time.Length)}|\n" +

$"{new string('-', 114)}\n"

);

}

}

}

namespace DifferentSortings

{

public struct Game

{

public string Name { get; set; }

public string GameType { get; set; }

public int Rating { get; set; }

public Game(string name, string gameType, int rating)

{

Name = name;

GameType = gameType;

Rating = rating;

}

}

}