Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Задание 4 - Усовершенствованные алгоритмы сортировки массивов

тема

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | Р.Ю. Царев |
|  | подпись, дата | инициалы, фамилия |
| Студент КИ18-17/1б 031831229 |  | В.А. Прекель |
| номер группы, зачетной книжки | подпись, дата | инициалы, фамилия |

Красноярск 2020

# 1 Цель работы с постановкой задачи

## 1.1 Цель работы

Усовершенствованные алгоритмы сортировки массивов.

## 1.2 Задача работы

Реализовать в программе два алгоритма (по выбору студента) из указанных ниже:

а) шейкерная сортировка,

б) сортировка Шелла,

в) быстрая сортировка.

Сравнить эффективность реализованных алгоритмов.

Требования к выполнению лабораторной работы:

1. Строгое соответствие программы и результатов ее работы с полученным заданием.

2. Самостоятельные тестирование и отладка программы.

3. Устойчивость работы программы при любых воздействиях, задаваемых пользователем через интерфейс программы.

4. Предоставление демонстрационного примера и исходного текста программы для защиты.

5. Предоставление отчета по практическому заданию, содержащего описание реализованного алгоритма, программы, результатов работы программы (отчет необходимо загрузить на сайт курса).

# 2 Описание реализованного алгоритма

Реализован алгоритм шейкерной сортировки и быстрой сортировки. Подсчитывается число сравнений и число присваиваний. Реализовано на языке C# и поддерживает различные сравнимые типы данных, а также выбор в каком порядке сортировать. Написаны юнит-тесты для различных типов данных используя фреймворк NUnit.

# 3 Описание программы (листинги кода)

Листинг 1 – Alg\_04/Alg\_04.Core/AbstractSort.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_04.Core  
{  
 public abstract class AbstractSort<T>  
 where T : IComparable  
 {  
 public enum SortOrder  
 {  
 Ascending = 1,  
 Descending = -1  
 }  
  
 public int AssignmentCount { get; protected set; }  
 public int CompareCount { get; protected set; }  
  
 public SortOrder Order { get; set; } = SortOrder.Ascending;  
  
 protected int Compare(T a, T b)  
 {  
 CompareCount++;  
 return (int) Order \* a.CompareTo(b);  
 }  
  
 public virtual void Sort(IList<T> list)  
 {  
 AssignmentCount = 0;  
 CompareCount = 0;  
 }  
 }  
}

Листинг 2 – Alg\_04/Alg\_04.Core/ShakerSort.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_04.Core  
{  
 public class ShakerSort<T> : AbstractSort<T>  
 where T : IComparable  
 {  
 public override void Sort(IList<T> list)  
 {  
 base.Sort(list);  
  
 var left = 0;  
  
 var right = list.Count - 1;  
 var flag = 0;  
  
 for (var i = 0; i < list.Count; i++)  
 {  
 flag = 0;  
 if (i % 2 == 0)  
 {  
 for (var j = right; j > left; j--)  
 {  
 if (Compare(list[j], list[j - 1]) < 0)  
 {  
 AssignmentCount += 2;  
 flag++;  
 var temp = list[j];  
 list[j] = list[j - 1];  
 list[j - 1] = temp;  
 }  
 }  
  
 left++;  
 if (flag == 0)  
 {  
 break;  
 }  
 }  
 else  
 {  
 for (var j = left; j < right; j++)  
 {  
 if (Compare(list[j], list[j + 1]) > 0)  
 {  
 AssignmentCount += 2;  
 flag++;  
 var temp = list[j];  
 list[j] = list[j + 1];  
 list[j + 1] = temp;  
 }  
 }  
  
 right--;  
 }  
  
 if (flag == 0)  
 {  
 break;  
 }  
 }  
 }  
 }  
}

Листинг 3 – Alg\_04/Alg\_04.Core/FastSort.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_04.Core  
{  
 public class FastSort<T> : AbstractSort<T>  
 where T : IComparable  
 {  
 public override void Sort(IList<T> list)  
 {  
 base.Sort(list);  
 Sort(list, 0, list.Count - 1);  
 }  
  
 private void Sort(IList<T> list, int left, int right)  
 {  
 if (left >= right)  
 {  
 return;  
 }  
  
 var p = Partition(list, left, right);  
  
 var s1 = new FastSort<T> {Order = Order};  
 s1.Sort(list, left, p);  
 CompareCount += s1.CompareCount;  
 AssignmentCount += s1.AssignmentCount;  
  
 var s2 = new FastSort<T> {Order = Order};  
 s2.Sort(list, p + 1, right);  
 CompareCount += s2.CompareCount;  
 AssignmentCount += s2.AssignmentCount;  
 }  
  
 private int Partition(IList<T> list, int left, int right)  
 {  
 var temp = list[(left + right) / 2];  
  
 var i = left;  
 var j = right;  
  
 while (i <= j)  
 {  
 while (Compare(list[i], temp) < 0)  
 {  
 i++;  
 }  
  
 while (Compare(list[j], temp) > 0)  
 {  
 j--;  
 }  
  
 if (i >= j)  
 {  
 break;  
 }  
  
 AssignmentCount += 2;  
 var temp1 = list[i];  
 list[i] = list[j];  
 list[j] = temp1;  
 i++;  
 j--;  
 }  
  
 return j;  
 }  
 }  
}

Листинг 4 – Alg\_04/Alg\_04.Core.Tests/TheoryGenericSortTests.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
  
using NUnit.Framework;  
  
namespace Alg\_04.Core.Tests  
{  
 internal class Comparable : IComparable  
 {  
 private static readonly Random Random = new Random();  
 private int Value { get; } = Random.Next();  
  
 public int CompareTo(object? obj) => obj == null ? 1 : Value.CompareTo(((Comparable) obj).Value);  
 }  
  
 [TestFixture(typeof(ShakerSort<int>), typeof(int))]  
 [TestFixture(typeof(FastSort<int>), typeof(int))]  
 [TestFixture(typeof(ShakerSort<double>), typeof(double))]  
 [TestFixture(typeof(FastSort<double>), typeof(double))]  
 [TestFixture(typeof(ShakerSort<string>), typeof(string))]  
 [TestFixture(typeof(FastSort<string>), typeof(string))]  
 [TestFixture(typeof(ShakerSort<DateTime>), typeof(DateTime))]  
 [TestFixture(typeof(FastSort<DateTime>), typeof(DateTime))]  
 [TestFixture(typeof(ShakerSort<Guid>), typeof(Guid))]  
 [TestFixture(typeof(FastSort<Guid>), typeof(Guid))]  
 [TestFixture(typeof(ShakerSort<Comparable>), typeof(Comparable))]  
 [TestFixture(typeof(FastSort<Comparable>), typeof(Comparable))]  
 public class TheoryGenericSortTests<TSort, T>  
 where TSort : AbstractSort<T>, new()  
 where T : IComparable  
 {  
 private TSort Sort { get; } = new TSort();  
  
 [Datapoint]  
 private List<double> \_arrayDouble1 = new List<double>(new[] {1.2, 3.4, 1.2, 3.4});  
  
 [Datapoint]  
 private List<double> \_arrayDouble2 = new List<double>(new[] {5.6, 7.8, 1.2, 3.4});  
  
 [Datapoint]  
 private List<int> \_arrayInt = new List<int>(new[] {0, 1, 5, 3});  
  
 [Datapoint]  
 private List<int> \_arrayInt1 = new List<int>(new[] {1, 3, 4, 34, 5, 6, 2, 33, 2});  
  
 [Datapoint]  
 private List<string> \_arrayString1 =  
 new List<string>(new[] {"gj", "hjhk", "ukft", "re", "aaa", "zzz", "hj", "fthf", "abcde"});  
  
 [Datapoint]  
 private List<string> \_arrayString2 =  
 new List<string>(new[] {"z", "x", "c"});  
  
 [Datapoint]  
 private List<DateTime> \_arrayDateTime1 =  
 new List<DateTime>(new[] {DateTime.Now, DateTime.Today, DateTime.MaxValue});  
  
 [Datapoint]  
 private List<DateTime> \_arrayDateTime2 =  
 new List<DateTime>(new[]  
 {  
 new DateTime(123, DateTimeKind.Utc),  
 new DateTime(214324, DateTimeKind.Utc),  
 new DateTime(325235235235, DateTimeKind.Utc),  
 new DateTime(433344, DateTimeKind.Utc),  
 new DateTime(0, DateTimeKind.Utc)  
 });  
  
 [Datapoint]  
 private List<DateTime> \_arrayDateTime3 =  
 new List<DateTime>(new[]  
 {  
 DateTime.Now,  
 new DateTime(2020, 05, 14),  
 new DateTime(2025, 05, 14)  
 });  
  
 [Datapoint]  
 private List<Guid> \_listGuid1 = new List<Guid>(new[]  
 {  
 Guid.NewGuid(), Guid.NewGuid(), Guid.NewGuid(), Guid.NewGuid(), Guid.NewGuid()  
 });  
  
 [Datapoint]  
 private List<Comparable> \_listComparable1 =  
 new List<Comparable>(Enumerable.Range(0, 100)  
 .Select(p => new Comparable())  
 );  
  
 [Theory]  
 public void ListSortTest(List<T> list)  
 {  
 Sort.Sort(list);  
 Assert.That(list.OrderByDescending(p => p).SequenceEqual(list), Is.False);  
 Assert.That(list.OrderBy(p => p).SequenceEqual(list), Is.True);  
 }  
  
 [Theory]  
 public void ListSortTestDescending(List<T> list)  
 {  
 Sort.Order = AbstractSort<T>.SortOrder.Descending;  
 Sort.Sort(list);  
 Assert.That(list.OrderByDescending(p => p).SequenceEqual(list), Is.True);  
 Assert.That(list.OrderBy(p => p).SequenceEqual(list), Is.False);  
 }  
 }  
}

Листинг 5 – Alg\_04/Alg\_04.Console/Program.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
using Alg\_04.Core;  
  
namespace Alg\_04.Console  
{  
 public class Program  
 {  
 private static void SortAndOut(AbstractSort<int> sort, IList<int> ar)  
 {  
 sort.Sort(ar);  
 System.Console.WriteLine(String.Join(" ", ar));  
 System.Console.WriteLine(  
 $"Кол-во сравнений: {sort.CompareCount}, присваиваний: {sort.AssignmentCount}");  
 }  
  
 public static void Main(string[] args)  
 {  
 System.Console.InputEncoding = Encoding.UTF8;  
 System.Console.OutputEncoding = Encoding.UTF8;  
  
 while (true)  
 {  
 try  
 {  
 System.Console.WriteLine("Введите элементы через пробел: ");  
 var a = System.Console.ReadLine()  
 .Split(new[] {" "}, StringSplitOptions.RemoveEmptyEntries)  
 .Select(Int32.Parse)  
 .ToList();  
 var b = a.ToList();  
  
 var s1 = new ShakerSort<int>();  
 var s2 = new FastSort<int>();  
  
 System.Console.WriteLine("По возрастанию? [Y(Д)/n(н)]: ");  
 var ans = System.Console.ReadLine();  
 if (ans != "" && ans != "Y" && ans != "Д")  
 {  
 s1.Order = AbstractSort<int>.SortOrder.Descending;  
 s2.Order = AbstractSort<int>.SortOrder.Descending;  
 }  
  
 System.Console.WriteLine("Шейкерная сортирвка: ");  
 SortAndOut(s1, a);  
  
 System.Console.WriteLine("Быстрая сортировка: ");  
 SortAndOut(s2, b);  
  
 System.Console.ReadKey();  
  
 break;  
 }  
 catch (Exception e)  
 {  
 System.Console.Error.WriteLine($"Ошибка: {e.Message}\n");  
 }  
 }  
 }  
 }  
}

# 4 Результаты работы программы

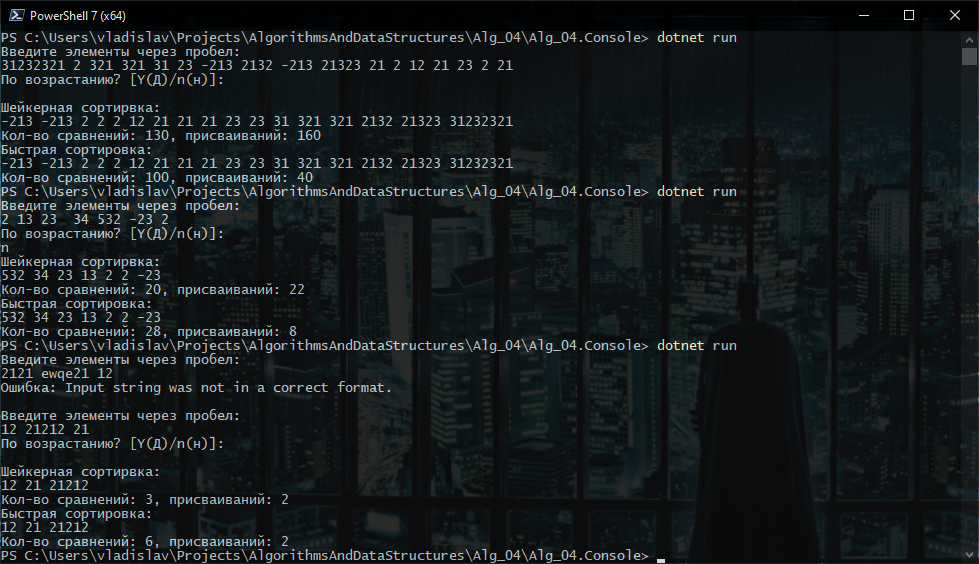


Рисунок 1 – Запуск программы

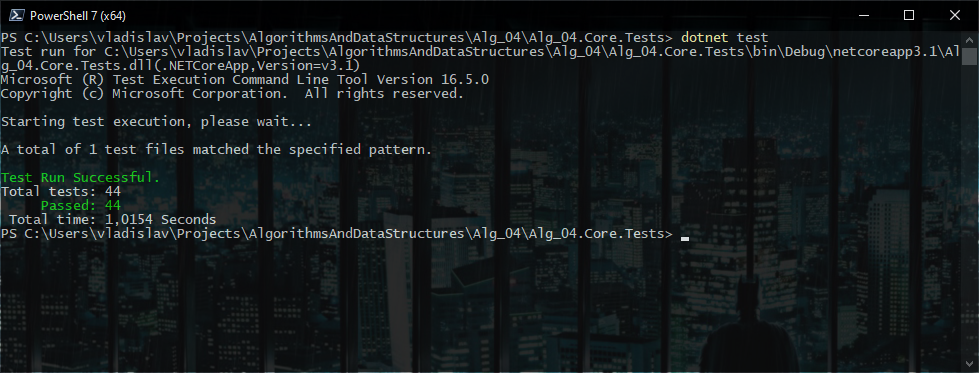


Рисунок 2 – Рисунок 2 – Запуск тестов