|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №1.1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9124-09.03.04прогин | | | |
|  |  |  | | И.Л. Барсуков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Преподаватель | | |  |
|  | | | | | | |  |  | А. А. Шулятьев | | | |
|  | | | |  |  | |  |  |  |  | И.Л. Барсуков | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2025 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Реализовать алгоритмы сортировки и сравнить их.

Входные данные:

1. текстовый файл, каждая из строк которого содержит 2 ключа в табличном виде (первый столбец - 1 ключ - дата, второй столбец - 2 ключ - ФИО.
2. целое число n - количество строк входного файла для обработки

Выходные данные:

1. текстовых файла, строки которых содержат отсортированные данные входного файла (первый файл - первым алгоритмом сортировки, второй файл - вторым алгоритмом сортировки), последняя строка - время, затраченное на сортировку.
2. Отсортировать данные входного файла в соответствии с заданным порядком сортировки ключей (см. свой вариант задания). Данные в выходном файле должны быть представлены в табличном виде, первый и второй столбец должны содержать значения ключей в соответствии с заданным порядком сортировки, третий столбец - номер строки входного файла.
3. Проверить на устойчивость.
4. Сравнить алгоритмы сортировки по времени.

**2 Описание алгоритмов сортировки**

**Сортировка вставками (InsertionSort)**

Алгоритм проходит по массиву, начиная со второго элемента, и «вставляет» каждый элемент в уже отсортированную часть слева. Алгоритм не использует дополнительные структуры данных для сортировки.  
Реализация использует обобщённый тип T, реализующий интерфейс IComparable<T>, что позволяет сортировать любые сравнимые объекты.

**Естественная сортировка слиянием (NaturalMergeSort)**

Алгоритм автоматически определяет уже отсортированные «пробеги» в массиве и сливает их попарно, пока весь массив не станет отсортированным.

Для хранения отсортированных частей используется дополнительный массив

Реализация использует обобщённый тип T, реализующий интерфейс IComparable<T>, что позволяет сортировать любые сравнимые объекты.

**3 Текст программы**

**Sortings:**

namespace Sorting;  
  
public static class Sorting  
{  
 public static void InsertionSort<T>(T[] array, int length) where T : IComparable<T>  
 {  
 for (var i = 1; i < length; i++)  
 {  
 var key = array[i];  
 var j = i - 1;  
   
 for (;j >= 0 && array[j].CompareTo(key) > 0; j--)  
 array[j + 1] = array[j];  
   
 array[j + 1] = key;  
 }  
 }  
 public static void NaturalMergeSort<T>(T[] array, int length) where T : IComparable<T>  
 {  
 if (length <= 1)  
 return;  
   
 var temp = new T[length];  
 bool merged;  
  
 do  
 {  
 merged = false;  
 var read = 0;  
  
 while (read < length)  
 {  
 var start1 = read;  
 var end1 = FindRunEnd(array, length, start1);  
 read = end1 + 1;  
  
 if (read >= length)  
 {  
 Array.Copy(array, start1, temp, start1, end1 - start1 + 1);  
 break;  
 }  
  
 var end2 = FindRunEnd(array, length, read);  
 read = end2 + 1;  
  
 Merge(array, temp, start1, end1, end2);  
 merged = true;  
 }  
   
 if (merged)  
 Array.Copy(temp, array, length);  
   
 } while(merged);  
 }  
  
 private static int FindRunEnd<T>(T[] array, int length, int start) where T : IComparable<T>  
 {  
 var i = start;  
 while (i + 1 < length && array[i].CompareTo(array[i + 1]) <= 0)  
 i++;  
   
 return i;  
 }  
   
 private static void Merge<T>(T[] source, T[] destination, int left, int mid, int right) where T : IComparable<T>  
 {  
 var i = left;  
 var j = mid + 1;  
 var k = left;  
  
 while (i <= mid && j <= right)  
 {  
 if (source[i].CompareTo(source[j]) <= 0) {  
 destination[k++] = source[i++];  
 continue;  
 }  
   
 destination[k++] = source[j++];  
 }  
  
 while (i <= mid)  
 destination[k++] = source[i++];  
  
 while (j <= right)  
 destination[k++] = source[j++];  
 }  
}

**Main:**

using System.Diagnostics;  
using System.Globalization;  
using System.Numerics;  
using System.Text;  
using Record;  
  
namespace Lab2;  
  
public static class Program  
{  
 private const string InputFile = "input.txt";  
  
 private static List<UserRecord> ReadRecords(int countRecords)  
 {  
 var records = new List<UserRecord>();  
 using var reader = new StreamReader(InputFile);  
 var index = 1;  
  
 while (index <= countRecords && reader.ReadLine() is { } line)  
 {  
 var parts = line.Split('\t', StringSplitOptions.RemoveEmptyEntries);  
 var date = new CustomMyDate(parts[0]);  
  
 var names = parts[1].Split(' ');  
   
 records.Add(new UserRecord { Date = date, FullName = new FullName()  
 {  
 LastName = names[0],  
 FirstName = names[1],  
 MiddleName = names[2],  
 }, Index = index });  
 index++;  
 }  
   
 return records;  
 }  
   
 private static string GetProjectRoot()  
 {  
 var dirInfo = new DirectoryInfo(AppContext.BaseDirectory);  
   
 while (dirInfo is not null)  
 {  
 if (dirInfo.GetFiles("\*.csproj").Length != 0)  
 return dirInfo.FullName;  
  
 dirInfo = dirInfo.Parent;  
 }  
   
 return Directory.GetCurrentDirectory();  
 }  
   
 private static StringBuilder ToLines(UserRecord[] arr, TimeSpan time)  
 {  
 var lines = new StringBuilder();  
   
 foreach (var r in arr)  
 lines.AppendLine(r.ToString());  
   
 lines.Append($"Время сортировки: {time.TotalSeconds:F3} сек");  
 return lines;  
 }  
  
 private static TimeSpan RunSort(UserRecord[] array, int length, Action<UserRecord[], int> sort)  
 {  
 var sw = Stopwatch.StartNew();  
 sort(array, length);  
 sw.Stop();  
   
 return sw.Elapsed;  
 }  
   
 public static void Main()  
 {  
 Console.Write("Введите количество строк n: ");  
 var n = int.Parse(Console.ReadLine() ?? string.Empty);  
   
 var records = ReadRecords(n);  
 var dataMerge = records.ToArray();  
 var dataInsert = records.ToArray();  
   
 Console.WriteLine("Старт сортировки: Слияние (естественное, итеративная реализация)");  
 var mergeTime = RunSort(dataMerge, n, Sorting.Sorting.NaturalMergeSort);  
   
 Console.WriteLine("Старт сортировки: Простые вставки");  
 var insertTime = RunSort(dataInsert, n, Sorting.Sorting.InsertionSort);  
   
 Console.WriteLine("Сортировки завершены");  
   
 File.WriteAllText(Path.Combine(GetProjectRoot(), "output\_merge.txt"), ToLines(dataMerge, mergeTime).ToString());  
 File.WriteAllText(Path.Combine(GetProjectRoot(), "output\_insertion.txt"), ToLines(dataInsert, insertTime).ToString());  
 }  
}

**Record:**

namespace Record;  
  
public struct UserRecord : IComparable<UserRecord>  
{  
 public CustomMyDate Date;  
 public FullName FullName;  
 public int Index;  
  
 public override string ToString()  
 {  
 return $"{Date}\t{FullName}\t{Index}";  
 }  
   
 public string ToStringWithoutIndex()  
 {  
 return $"{Date}\t{FullName}";  
 }  
  
 public int CompareTo(UserRecord other)  
 {  
 var compare = Date.CompareTo(other.Date);  
 return compare != 0 ? compare : other.FullName.CompareTo(FullName);  
 }  
}

**FullName:**

namespace Record;  
  
public struct FullName : IComparable<FullName>  
{  
 public string LastName;  
 public string FirstName;  
 public string MiddleName;  
   
 public override string ToString()  
 {  
 return $"{LastName} {FirstName} {MiddleName}";  
 }  
   
 public int CompareTo(FullName other)  
 {  
 var lastNameCompare = string.Compare(LastName, other.LastName, StringComparison.OrdinalIgnoreCase);  
 if (lastNameCompare != 0)  
 return lastNameCompare;  
   
 var firstNameCompare = string.Compare(FirstName, other.FirstName, StringComparison.OrdinalIgnoreCase);  
 if (firstNameCompare != 0)  
 return firstNameCompare;  
   
 return string.Compare(MiddleName, other.MiddleName, StringComparison.OrdinalIgnoreCase);  
 }  
}

**CustomDateTime:**using System;  
  
namespace Record  
{  
 public readonly struct CustomMyDate : IComparable<CustomMyDate>  
 {  
 public readonly int Year;  
 public readonly int Month;  
 public readonly int Day;  
  
 private static int DaysInMonth(int year, int month)  
 {  
 return month switch  
 {  
 1 or 3 or 5 or 7 or 8 or 10 or 12 => 31,  
 4 or 6 or 9 or 11 => 30,  
 2 => IsLeapYear(year) ? 29 : 28,  
 \_ => throw new ArgumentException("Invalid month")  
 };  
 }  
  
 private static bool IsLeapYear(int year)  
 {  
 if ((year & 3) != 0) return false;  
 if ((year & 15) == 0) return true;  
 return (uint)year % 25 != 0;  
 }  
   
 public CustomMyDate(int year, int month, int day)  
 {  
 if (year < 1 || month < 1 || month > 12 || day < 1 || day > DaysInMonth(year, month))  
 throw new ArgumentException("Invalid date");  
   
 Year = year;  
 Month = month;  
 Day = day;  
 }  
   
 public CustomMyDate(string dateStr)  
 {  
 if (string.IsNullOrWhiteSpace(dateStr))  
 throw new ArgumentException("Date string cannot be null or empty");  
  
 var parts = dateStr.Split('-');  
 if (parts.Length != 3)  
 throw new ArgumentException("Date string must be in format yyyy-MM-dd");  
  
 if (!int.TryParse(parts[0], out var year) ||  
 !int.TryParse(parts[1], out var month) ||  
 !int.TryParse(parts[2], out var day))  
 throw new ArgumentException("Invalid date components");  
  
 if (year < 1 || month < 1 || month > 12 || day < 1 || day > DaysInMonth(year, month))  
 throw new ArgumentException("Invalid date");  
  
 Year = year;  
 Month = month;  
 Day = day;  
 }  
   
 public CustomMyDate AddDays(int daysToAdd)  
 {  
 var year = Year;  
 var month = Month;  
 var day = Day + daysToAdd;  
  
 while (true)  
 {  
 var daysInMonth = DaysInMonth(year, month);  
 if (day <= daysInMonth)  
 break;  
  
 day -= daysInMonth;  
 month++;  
 if (month <= 12) continue;  
   
 month = 1;  
 year++;  
 }  
  
 while (day < 1)  
 {  
 month--;  
 if (month < 1)  
 {  
 month = 12;  
 year--;  
 }  
 day += DaysInMonth(year, month);  
 }  
  
 return new CustomMyDate(year, month, day);  
 }  
   
 public int CompareTo(CustomMyDate other)  
 {  
 if (Year != other.Year) return Year.CompareTo(other.Year);  
 if (Month != other.Month) return Month.CompareTo(other.Month);  
 return Day.CompareTo(other.Day);  
 }  
   
 public override string ToString()  
 {  
 return $"{Year:0000}-{Month:00}-{Day:00}";  
 }  
 }  
}

**4 Тесты**

**Сравнение двух алгоритмов по времени их работы**

Для проведения тестов решено было провести 16 случаев, где на каждый случай количество элементов вычислялось по формуле: 2i где i – номер теста от 1 до 16.

|  |  |  |
| --- | --- | --- |
| N | Прямые вставки | Естественное слияние |
| 2 | 2 µs | 1 µs |
| 4 | 1 µs | 2 µs |
| 8 | 3 µs | 3 µs |
| 16 | 4 µs | 5 µs |
| 32 | 10 µs | 15 µs |
| 64 | 27 µs | 25 µs |
| 128 | 1452 µs | 70 µs |
| 256 | 386 µs | 108 µs |
| 512 | 1416 µs | 291 µs |
| 1024 | 5783 µs | 611 µs |
| 2048 | 24506 µs | 1480 µs |
| 4096 | 97878 µs | 3141 µs |
| 8192 | 374894 µs | 7171 µs |
| 16384 | 1622083 µs | 16445 µs |
| 32768 | 8293861 µs | 49901 µs |
| 65536 | 38176364 µs | 151632 µs |

**Наихудшее и наилучшее значение времени сортировок**

Условия:

N = 50000

Для наилучшего случая массив будет уже отсортирован

Для наихудшего случая массив будет отсортирован в обратном порядке

|  |  |  |
| --- | --- | --- |
| Случаи | Прямые вставки | Естественное слияние |
| Наилучший случай | 8002 | 7339 |
| Наихудший случай | 37427368 | 151686 |

Из таблицы видно что алгоритмы показывают себя практически одинаково по времени при сортировки отсортированного массива, что соответствует асимптотической сложности O(n). В обратном случае, когда массив отсортирован в обратном порядке, алгоритмы показывают совершенно разные результаты, что соответствует асимптотам O(n²) для прямхе вставок и O(n log n) для естественного слияния.

**Стабильность сортировок**

Для теста было взято 10 элементов с одинаковыми ключами и значения

|  |  |
| --- | --- |
| 2024-07-27 | Белякова Алёна Святославовна |
| 2021-12-19 | Лазарева Милана Богдановна |
| 2024-07-27 | Белякова Алёна Святославовна |
| 2021-12-19 | Лазарева Милана Богдановна |
| 2024-07-27 | Белякова Алёна Святославовна |
| 2021-12-19 | Лазарева Милана Богдановна |
| 2024-07-27 | Белякова Алёна Святославовна |
| 2021-12-19 | Лазарева Милана Богдановна |
| 2024-07-27 | Белякова Алёна Святославовна |
| 2021-12-19 | Лазарева Милана Богдановна |

Результаты обоих сортировок

|  |  |  |
| --- | --- | --- |
| 2021-12-19 | Лазарева Милана Богдановна | 2 |
| 2021-12-19 | Лазарева Милана Богдановна | 4 |
| 2021-12-19 | Лазарева Милана Богдановна | 6 |
| 2021-12-19 | Лазарева Милана Богдановна | 8 |
| 2021-12-19 | Лазарева Милана Богдановна | 10 |
| 2024-07-27 | Белякова Алёна Святославовна | 1 |
| 2024-07-27 | Белякова Алёна Святославовна | 3 |
| 2024-07-27 | Белякова Алёна Святославовна | 5 |
| 2024-07-27 | Белякова Алёна Святославовна | 7 |
| 2024-07-27 | Белякова Алёна Святославовна | 9 |

По результатам обе сортировки стабильны, т.е. сохраняют порядок индексов при одинаковых значениях.

**Вывод**

Сравнение алгоритмов сортировки прямыми вставками и естественным слиянием показало превосходство алгоритма “естественное слияние” с линейно-логарифмической сложностью O(n log n) над алгоритмом “ прямые вставки” с квадратичной сложностью O(n²) на больших объемах данных.