«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий

ОТЧЕТ

По Лабораторной работе № 2

по дисциплине «Теория и технология программирования»

**Выполнил:**

студент группы з3530902/20001  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.Л. Зыкин

подпись, дата

**Проверил**

Доцент, кандидат технических наук                                  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.В. Хлопин

подпись, дата

Санкт-Петербург 2023г.

**Задание**

**Цель работы:** изучение алгоритмов сортировки. Применение указателей, ссылок при передаче параметров в функции. Изучение методик анализа данных, построения графиков абсолютных и нормированных значений. Изучение методик и правил построения блок схем алгоритмов.

В соответствии с индивидуальным заданием необходимо:

1. Создать программу для сортировки массива случайных данных заданными методом сортировки.
2. Провести серию экспериментов с разным размером сортируемых данных. Провести анализ изменения зависимости числа сравнений и перестановок элементов при сортировках от размера сортируемого массива. Построить графики изменения числа сравнений и перестановок от размеров массивов сортировки. Сравнение производить по абсолютным значениям, по нормированным (приведение к одному знаменателю – деление числа сравнений и перестановок на размер массива) и приведенным к логарифмическому виду величинам (вычисление логарифма от числа сравнений и перестановок).
3. Построить блок схему алгоритма сортировки, соответствующую алгоритму кода программы.

Вариант № 21:

|  |  |  |
| --- | --- | --- |
| Сортировки | Размеры массивов для экспериментов | Блок схема |
| Вставка, отбор | 12;24;48;96;192;384;768 | Вставка |

1. **Программа по сортировке случайных данных заданным методом:**

**Пример работы программы:**

1. (рис. 1) на старте программы, выбираем необходимый способ сортировки.

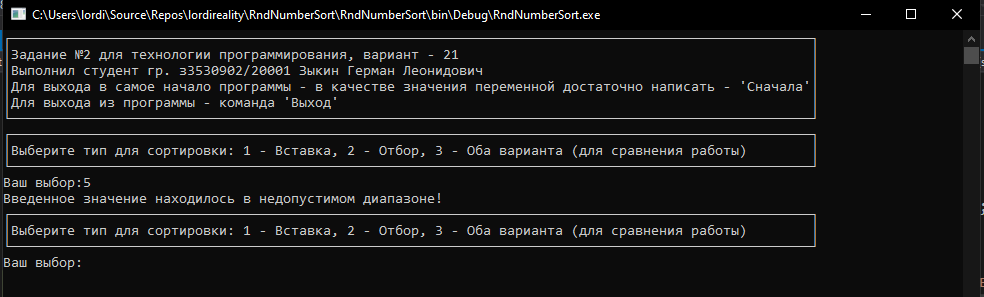


Рисунок 1 – Выбор необходимого способа сортировки

1. (рис. 2). Указываем размер массива, в соответствии с которым, будет создана n-ая последовательность чисел и произведена сортировка

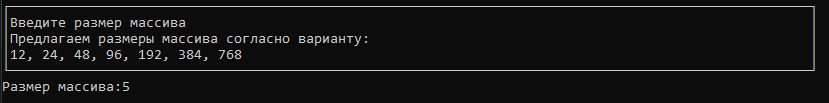


Рисунок 2 – Указание размера массива

1. (рис. 3) по итогу получаем случайно сгенерированный массив указанной размерности, отсортированный массив и сведения о кол-ве перестановок и сравнений при выполнении программы.

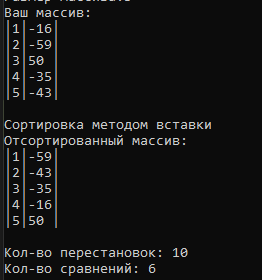


Рисунок 3 – Пример результата исполнения программы

**Код программы:**

|  |
| --- |
| using System;  using System.Collections;  using System.Collections.Generic;  using System.Data;  using System.Linq;  using System.Net;  using System.Net.Http.Headers;  using System.Runtime.InteropServices;  using System.Security.Cryptography;  using System.Text;  using System.Threading.Tasks;  using System.Web;  namespace RndNumberSort  {      internal class Program      {          /// <summary>          /// Основной метод работы программы          /// </summary>          /// <param name="args">Аргументы для запуска</param>          static void Main(string[] args)          {              var \_consoleHelper = new ConsoleHelper();              var \_magicStuffClass = new MagicStuffClass();              while (true)              {                  Console.Clear();                  Console.WriteLine(\_consoleHelper.getCredentials());                  var sortTypeChoice = new KeyBoardInput\_DTO() { succed = false, value = "" };                  while (!sortTypeChoice.succed)                  {                      Console.WriteLine(\_consoleHelper.getPrettyBox("Выберите тип для сортировки: 1 - Вставка, 2 - Отбор, 3 - Оба варианта (для сравнения работы)"));                      sortTypeChoice = InputController("Ваш выбор:", new Dictionary<string, string>() { { "typeof", "int" }, { "minVal", "0" }, { "maxVal", "4" } });                  }                  var arraySizeChoice = new KeyBoardInput\_DTO() { succed = false, value = "" };                  Console.WriteLine(\_consoleHelper.getPrettyBox3Rows(new string[] { "Введите размер массива ", "Предлагаем размеры массива согласно варианту: ", string.Join(", ", \_magicStuffClass.\_possibleSizes.Select(i => i.ToString()).ToArray()) }));                  while (!arraySizeChoice.succed)                  {                      arraySizeChoice = InputController("Размер массива:", new Dictionary<string, string>() { { "typeof", "int" }, { "minVal", "0" }, { "maxVal", int.MaxValue.ToString() } });                  }                  var arrayToSort = \_magicStuffClass.getArray(Int32.Parse(arraySizeChoice.value));                  //arrayToSort.CopyTo(arrayToSort2, 0);                  Console.WriteLine("Ваш массив:");                  Console.WriteLine(\_consoleHelper.buildTwoColOutputTable(arrayToSort));                  dynamic sortManagerInsertion = null;                  dynamic sortManagerSelection = null;                  switch (sortTypeChoice.value)                  {                      case "1": sortManagerInsertion = new SortManager.InsertionSort(arrayToSort); break;                      case "2": sortManagerSelection = new SortManager.SelectionSort(arrayToSort); break;                      case "3": sortManagerInsertion = new SortManager.InsertionSort(arrayToSort); sortManagerSelection = new SortManager.SelectionSort(arrayToSort);  break;                      default: throw new Exception();                  }                  if(sortManagerInsertion != null)                  {                      sortManagerInsertion.Run();                      Console.WriteLine(sortManagerInsertion.sortName);                      Console.WriteLine("Отсортированный массив:");                      Console.WriteLine(\_consoleHelper.buildTwoColOutputTable(sortManagerInsertion.getOutputArray()));                      Console.WriteLine("Кол-во перестановок: " + sortManagerInsertion.swapCount.ToString());                      Console.WriteLine("Кол-во сравнений: " + sortManagerInsertion.compareCount.ToString());                      Console.WriteLine("\n\n");                  }                  if (sortManagerSelection != null)                  {                      sortManagerSelection.Run();                      Console.WriteLine(sortManagerSelection.sortName);                      Console.WriteLine("Отсортированный массив:");                      Console.WriteLine(\_consoleHelper.buildTwoColOutputTable(sortManagerSelection.getOutputArray()));                      Console.WriteLine("Кол-во перестановок: " + sortManagerSelection.swapCount.ToString());                      Console.WriteLine("Кол-во сравнений: " + sortManagerSelection.compareCount.ToString());                      Console.WriteLine("\n\n");                  }                  Console.ReadLine();                }          }          /// <summary>          /// Контроллер ввода значений          /// </summary>          /// <param name="promt">Текст подсказки</param>          /// <param name="validationRules">Правила валидации</param>          /// <returns></returns>          public static KeyBoardInput\_DTO InputController(string promt, Dictionary<string, string> validationRules)          {              Console.Write(promt);              string input = Console.ReadLine();              var validatorInstance = new Validator(input, validationRules, true);              if (!validatorInstance.isValid())              {                  foreach (var message in validatorInstance.messages)                  {                      Console.WriteLine(message);                  }              }              var keyBoardInput\_DTO = new KeyBoardInput\_DTO()              {                  succed = validatorInstance.isValid(),                  value = input              };              return keyBoardInput\_DTO;          }      }      public class ConsoleHelper      {          /// <summary>          /// Строим вертикальную таблицу с двумя колонками          /// </summary>          /// <param name="values">Массив значений INT</param>          /// <returns>Колонка для отображения</returns>          public string buildTwoColOutputTable(int[] values)          {              int maxSymbolsFirst = values.Length.ToString().Length;              int maxSymbolsSecon = (values.Max().ToString().Length > values.Min().ToString().Length ? values.Max().ToString().Length : values.Min().ToString().Length);              string output = "";              for (int i = 0; i < values.Length; i++)              {                  int firstColumnSymbols = maxSymbolsFirst - (i + 1).ToString().Length;                  int secondColumnSymbols = maxSymbolsSecon - values[i].ToString().Length;                  output += "│" + (i + 1).ToString() + (firstColumnSymbols != 0 ? new String(' ', firstColumnSymbols) : "") + "│" + values[i].ToString() + (secondColumnSymbols != 0 ? new String(' ', secondColumnSymbols) : "") + "│" + "\n";              }              return output;          }          /// <summary>          /// Копирайт работы в CMD          /// </summary>          /// <returns>текст</returns>          public string getCredentials()          {              return              "┌────────────────────────────────────────────────────────────────────────────────────────────────────┐" + "\n" +              "│Задание №2 для технологии программирования, вариант - 21                                            │" + "\n" +              "│Выполнил студент гр. з3530902/20001 Зыкин Герман Леонидович                                         │" + "\n" +              "│Для выхода в самое начало программы - в качестве значения переменной достаточно написать - 'Сначала'│" + "\n" +              "│Для выхода из программы - команда 'Выход'                                                           │" + "\n" +              "└────────────────────────────────────────────────────────────────────────────────────────────────────┘";          }          /// <summary>          /// Красивая коробочка для текста - 1 строка          /// </summary>          /// <param name="promt">Текст вписываемый в коробочку</param>          /// <returns></returns>          public string getPrettyBox(string promt)          {              return              "┌────────────────────────────────────────────────────────────────────────────────────────────────────┐" + "\n" +              "│" + promt + new String(' ', 100 - promt.Length) + "│" + "\n" +              "└────────────────────────────────────────────────────────────────────────────────────────────────────┘";          }          /// <summary>          /// Красивая коробочка для текста - 2 строки          /// </summary>          /// <param name="promt">Текст вписываемый в коробочку</param>          /// <returns></returns>          public string getPrettyBox2Rows(string[] promt)          {              return              "┌────────────────────────────────────────────────────────────────────────────────────────────────────┐" + "\n" +              "│" + promt[0] + new String(' ', 100 - promt[0].Length) + "│" + "\n" +              "│" + promt[1] + new String(' ', 100 - promt[1].Length) + "│" + "\n" +              "└────────────────────────────────────────────────────────────────────────────────────────────────────┘";          }          /// <summary>          /// Красивая коробочка для текста - 3 строки          /// </summary>          /// <param name="promt">Текст вписываемый в коробочку</param>          /// <returns></returns>          public string getPrettyBox3Rows(string[] promt)          {              return              "┌────────────────────────────────────────────────────────────────────────────────────────────────────┐" + "\n" +              "│" + promt[0] + new String(' ', 100 - promt[0].Length) + "│" + "\n" +              "│" + promt[1] + new String(' ', 100 - promt[1].Length) + "│" + "\n" +              "│" + promt[2] + new String(' ', 100 - promt[2].Length) + "│" + "\n" +              "└────────────────────────────────────────────────────────────────────────────────────────────────────┘";          }      }      /// <summary>      /// Класс магических значений      /// </summary>      public class MagicStuffClass      {          /// <summary>          /// "магические" размеры массивов из задания          /// </summary>          public int[] \_possibleSizes = new int[] { 12, 24, 48, 96, 192, 384, 768 };          /// <summary>          /// Получить массив необходимого размера          /// </summary>          /// <param name="sizeOfArray">Размер массива - (0 < размер)</param>          /// <param name="floor">Мин. значение генерируемых чисел</param>          /// <param name="limit">Макс. значение генерируемых чисел</param>          /// <returns>Сгенерированных массив</returns>          public int[] getArray(int sizeOfArray, int floor = -100, int limit = 100)          {              var \_rndGenerator = new Random();              var buff = new int[sizeOfArray];              for (int i = 0; i < sizeOfArray; i++)              {                  buff[i] = \_rndGenerator.Next(floor, limit);              }              return buff;          }      }      /// <summary>      /// Класс для хранения и передачи информации о вводе с клавиатуры и его успешности      /// </summary>      public class KeyBoardInput\_DTO      {          /// <summary>          /// Успешность ввода          /// </summary>          public bool succed = false;          /// <summary>          /// Введенное значение          /// </summary>          public string value = "";      }      /// <summary>      /// Класс - валидатор для вводимых значений      /// </summary>      public class Validator      {          /// <summary>          /// Строка для валидации          /// </summary>          private string strToValidate;          /// <summary>          /// Правила валидации          /// </summary>          private Dictionary<string, string> validationRules;          /// <summary>          /// Сообщения об ошибках валидации, если такие имеются          /// </summary>          public List<string> messages = new List<string>();          /// <summary>          /// Является ли значение валидным          /// </summary>          private bool valid = true;          /// <summary>          /// Класс конструктор валидатора          /// </summary>          /// <param name="strToValidate">Строка, которую необходимо проверить</param>          /// <param name="validationRules">Правила валидации</param>          /// <param name="strictValidate">Метка - моментальный выход из валидации, если проверяемое значение не валидно</param>          public Validator(string strToValidate, Dictionary<string, string> validationRules, bool strictValidate = true)          {              this.strToValidate = strToValidate;              this.validationRules = validationRules;              this.Validate();          }          /// <summary>          /// Возвращает значение валидации          /// </summary>          /// <returns>True/False В зависимости от валидности значений</returns>          public bool isValid()          {              return this.valid;          }          /// <summary>          /// Исполнение валидации          /// </summary>          /// <param name="strictValidate">Метка - моментальный выход из валидации, если проверяемое значение не валидно</param>          private void Validate(bool strictValidate = true)          {              foreach (var rule in this.validationRules)              {                  switch (rule.Key)                  {                      case "typeof": valid = ((typeOfCast(rule.Value) == false) ? false : valid); break;                      case "minVal": valid = ((minValCheck(rule.Value) == false) ? false : valid); break;                      case "maxVal": valid = ((maxValCheck(rule.Value) == false) ? false : valid); break;                      default: Console.Write("[Ошибка!]Незивестное правило валидации - " + rule.Key); break;                  }                  if (strictValidate && !valid)                  {                      return;                  }              }          }          /// <summary>          /// Валидация типа данных          /// </summary>          /// <param name="type">Тип</param>          /// <returns>true/false в зависимости от успешности валидации</returns>          private bool typeOfCast(string type)          {              int intVal = 0; // костыль для TryParse              switch (type)              {                  case "int": return Int32.TryParse(this.strToValidate, out intVal);                  case "string": return true; //Емаё ввод с клавы и так стринг, костыль                  default: return false; //не знаем что за тип, возвращаем нет              }          }          /// <summary>          /// Валидация "пола" Int32 значений          /// </summary>          /// <param name="valToCheck">Сравниваемое значение</param>          /// <returns>true/false в зависимости от успешности валидации</returns>          private bool minValCheck(string valToCheck)          {              int intVal = 0;              int checkVal = 0;              Int32.TryParse(valToCheck, out checkVal);              bool casted = Int32.TryParse(this.strToValidate, out intVal);              if (casted)              {                  if (checkVal < intVal)                  {                      return true;                  }                  else                  {                      this.messages.Add("Введенное значение находилось в недопустимом диапазоне!");                      return false;                  }              }              else              {                  this.messages.Add("Введенное значение имеет недопустимый тип!");                  return false;              }          }          /// <summary>          /// Валидация "потолка" Int32 значений          /// </summary>          /// <param name="valToCheck">Сравниваемое значение</param>          /// <returns>true/false в зависимости от успешности валидации</returns>          private bool maxValCheck(string valToCheck)          {              int intVal = 0;              int checkVal = 0;              Int32.TryParse(valToCheck, out checkVal);              bool casted = Int32.TryParse(this.strToValidate, out intVal);              if (casted)              {                  if (checkVal > intVal)                  {                      return true;                  }                  else                  {                      this.messages.Add("Введенное значение находилось в недопустимом диапазоне!");                      return false;                  }              }              else              {                  this.messages.Add("Введенное значение имеет недопустимый тип!");                  return false;              }          }      }      /// <summary>      /// Класс сортировщик      /// </summary>      public class SortManager      {          /// <summary>          /// Базовый класс сортировщиков массивов          /// </summary>          public abstract class SortBase          {              internal int[] inputArray;              internal int[] tempArray;              internal int[] outputArray;              internal bool sortExecuted = false;              internal int compareCount = 0;              internal int swapCount = 0;              internal string SortName = "%название типа сортировки%";              public string sortName { get { return SortName; }}              /// <summary>              /// Класс конструктор              /// </summary>              /// <param name="inputArray">Массив для сортировки</param>              public SortBase()              {              }              /// <summary>              /// Получить вх. данные для данного экземпляра              /// </summary>              /// <returns></returns>              public int[] getInputArray()              {                  return this.inputArray;              }              /// <summary>              /// Получить выходные данные для данного экземпляра              /// Если не была произведена сортировка, будет выдано исключение              /// </summary>              /// <returns>Отсортированный массив</returns>              public int[] getOutputArray()              {                  if (sortExecuted) { return this.outputArray; }                  else                  {                      throw new NullReferenceException();                  }              }              /// <summary>              /// Выполнение сортировки              /// </summary>              /// <exception cref="NotImplementedException"></exception>              public void Run()              {                  throw new NotImplementedException();              }          }          /// <summary>          /// Класс сортировщика вставкой          /// </summary>          public class InsertionSort : SortBase          {              /// <summary>              /// Класс конструктор              /// </summary>              /// <param name="inputArray">Массив для сортировки</param>              public InsertionSort(int[] inputArray)              {                  this.SortName = "Сортировка методом вставки";                  this.inputArray = new int[inputArray.Length];                  inputArray.CopyTo(this.inputArray, 0);                  this.tempArray = new int[inputArray.Length];                  inputArray.CopyTo(this.tempArray, 0);              }              /// <summary>              /// Запустить сортировку              /// </summary>              public void Run()              {                  for (int i = 1; i < tempArray.Length; i++)                  {                      int key = tempArray[i];                      int j = i - 1;                      while (j >= 0 && tempArray[j] > key)                      {                          tempArray[j + 1] = tempArray[j];                          j--;                          compareCount++;                          swapCount++;                      }                      tempArray[j + 1] = key;                      swapCount++;                  }                  this.outputArray = new int[tempArray.Length];                  tempArray.CopyTo(outputArray, 0);                    this.sortExecuted = true;              }          }          /// <summary>          /// Класс сортировщика выбором          /// </summary>          public class SelectionSort : SortBase          {              /// <summary>              /// Класс конструктор              /// </summary>              /// <param name="inputArray">Массив для сортировки</param>              public SelectionSort(int[] inputArray)              {                  this.SortName = "Сортировка методом отбора";                  this.inputArray = new int[inputArray.Length];                  inputArray.CopyTo(this.inputArray, 0);                  this.tempArray = new int[inputArray.Length];                  inputArray.CopyTo(this.tempArray, 0);              }              /// <summary>              /// Запустить сортировку              /// </summary>              public void Run()              {                    for (int i = 0; i < tempArray.Length - 1; i++)                  {                      int minIdx = i;                      for (int j = i + 1; j < tempArray.Length; j++)                      {                          compareCount++;                          if (tempArray[j] < tempArray[minIdx])                          {                                minIdx = j;                            }                      }                        if (minIdx != i)                      {                          int temp = tempArray[i];                          tempArray[i] = tempArray[minIdx];                          tempArray[minIdx] = temp;                          swapCount++;                      }                  }                  this.outputArray = new int[tempArray.Length];                  tempArray.CopyTo(outputArray, 0);                  //this.outputArray = this.tempArray;                  this.sortExecuted = true;              }          }      }    } |

1. **Серия экспериментов с разным размером сортируемых данных:**
   1. **Сводная таблица (рис. 4)**



Рисунок 4 – Сводная таблица сравнений кол-ва выполняемых операций (сравнений, перестановок), в зависимости от выбранного типа сортировки.

* 1. **Графики изменения числа сравнений и перестановок от размеров массивов сортировки**

На основании данных из сводной таблицы (рис. 4), были построены графики:

* + 1. Сравнения (Абсолютные) (рис.5):

Рисунок

* + 1. Перестановки (Абсолютные) (рис.6):

Рисунок

* + 1. Сравнения (Нормированные) (рис.7):

Рисунок

* + 1. Перестановки (Нормированные) (рис.8):

Рисунок

* + 1. Сравнения (Логарифмические) (рис.9):

Рисунок

* + 1. Перестановки (Логарифмические) (рис.10):

Рисунок

* + 1. Сравнения (Логарифмические – десятичный логарифм) (рис.11):

Рисунок

* + 1. Перестановки (Логарифмические – десятичный логарифм) (рис.12):

Рисунок

1. **Алгоритм работы программы**

Блок схема отображающая общий алгоритм работы программы, приведен ниже (рис.13).



Рисунок 13 - Блок схема алгоритма работы программы при использовании метода сортировки - вставками

**Выводы**

В ходе выполнения работы, были изучены различные методы сортировок численных значений в массиве (метод вставки/отбора). Также была создана программа, реализующая данные типы сортировок и позволяющая получить информацию о кол-ве исполняемых операций (сравнений/перестановок) при выполнении данных типов сортировок в программе. Эксперименты в свою очередь показали, что метод сортировки отбором значительно эффективней, за счет меньшего кол-ва перестановок элементов в массиве в то время, как кол-во сравнений ненамного больше. Таким образом можно кол-во сравнений компенсировать кол-вом перестановок (в т.ч. в качестве быстродействия, можно применить тот факт, что при сравнении не требуется производить запись, а только чтение данных).