МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Кафедра Автоматизированных систем управления

**ОТЧЕТ О ВЫПОЛНЕНИИ**

**РАСЧЁТНО-ГРАФИЧЕСКОГО РАБОТЫ**

по теме «*Разработка программы-визуализатора процесса сортировки*»

Дисциплина «*Технологии и методы программирования*»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнили:** |  | **Проверил:** |  | |
| Студенты | *Бузмаков А.И.,*  *Шестаков К.Д.* | Преподаватель | *Астапчук В. А., Пустовских Д. А.* | |
| Факультет | *АВТФ* |  |  | |
| Направление (специальность) подготовки | *09.03.01 – Информатика и вычислительная техника* | Балл: \_\_\_\_\_\_\_\_\_ | |  |
| Группа | *АП-227* | Оценка *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | | |
| Шифр |  |  |  | |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| подпись | | подпись | | |
| Дата сдачи: | «29» мая 2024г. | Дата защиты: | «\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_. | |

Новосибирск 2024

**Оглавление**

[**Введение** 3](#_Toc167818807)

[Актуальность 3](#_Toc167818808)

[Цель работы 3](#_Toc167818809)

[Задачи 3](#_Toc167818810)

[Методы исследования 4](#_Toc167818811)

[**Анализ предметной области** 4](#_Toc167818812)

[**Проектирование архитектуры приложения** 4](#_Toc167818813)

[**Виды сортировок** 4](#_Toc167818814)

[**Проектирование архитектуры приложения** 8](#_Toc167818815)

[**Работа разработанного приложения** 8](#_Toc167818816)

[**Вывод** 12](#_Toc167818817)

[**Приложения** 13](#_Toc167818818)

[**Источники** 35](#_Toc167818819)

**Введение**

Сортировки – способ упорядочивание элементов в определенном порядке (в нашем случае от меньшего к большему). Сортировки очень полезный способ более детально рассмотреть определённую выборку значений. Это очень полезно для статистики и работы с большим количеством данных.

Давайте далее чуть подробнее поговорим о наиболее распространенных видах сортировок.

## Актуальность

Актуальность упорядочивания элементов в определённом порядке никогда не пропадёт. Все больше и больше информации о пользователях хранится в различных сервисах. Компаниям важно осматривать информацию, и сортировка элементов очень сильно помогает в этом. Более того, сортировка, позволяет хранить информацию так, чтобы она занимала меньше времени.

## Цель работы

Продемонстрировать выполнение работы различных видов сортировок. На основания увиденных и полученных данных обосновать скорость выполнения сортировок.

## Задачи

1. Разобрать схему работы таких сортировок, как: сортировка вставкой, сортировка пузырьком, сортировка выбором, сортировка слиянием.
2. Описать алгоритмы, опираясь на псевдокоды
3. Разработать программу для расчета и демонстрации сортировок.
4. Сравнить скорость выполнения сортировок.
5. Сделать выводы.

## Методы исследования

Для проведения исследования, мы разработали программу, используя Windows Forms. Так мы сможем показать, как, те или иные сортировки работают, и посмотреть скорость выполнения сортировок.

**Анализ предметной области**

Для пользования программой, пользователь не обязательно должен обладать информацией о том, как работают сортировки. Ему достаточно понимать, для чего они нужны.

Пользователь перемещается по программе, путем нажатия кнопок, которые либо выполняют определенную сортировку, либо перемещают пользователя в другие окна, для более подробной работы с сортировками.

**Проектирование архитектуры приложения**

Для того, чтобы понять, как стоит реализовывать анимации различных видов сортировок, стоит более подробнее разобраться, в том, как они работают. Для этого выберем часть сортировок, и рассмотрим их работы более подробно

**Виды сортировок**

**Сортировка вставкой**

Сортировка вставкой (Insertion Sort) — это простой алгоритм сортировки, который строит окончательную отсортированную последовательность поэлементно. Работает аналогично тому, как люди сортируют карты в руках.

1. Начинаем с первого элемента, он считается отсортированным.
2. Берём следующий элемент и вставляем его в нужное место в уже отсортированной части.
3. Повторяем до тех пор, пока все элементы не будут отсортированы.

Псевдокод:

|  |
| --- |
| insertionSort(arr)  for i from 1 to length(arr) - 1  key = arr[i]  j = i - 1  while j >= 0 and arr[j] > key  arr[j + 1] = arr[j]  j = j - 1  arr[j + 1] = key |

**Сортировка пузырьком**

Сортировка пузырьком (Bubble Sort) — это простой алгоритм сортировки, который многократно проходит по списку, сравнивая соседние элементы и меняя их местами, если они стоят в неправильном порядке.

1. Проходим по массиву.
2. Сравниваем каждый элемент с соседним и меняем местами, если они в неправильном порядке.
3. Повторяем, пока массив не будет отсортирован.

Псевдокод:

|  |
| --- |
| bubbleSort(arr)  n = length(arr)  for i from 0 to n - 1  for j from 0 to n - i - 1  if arr[j] > arr[j + 1]  swap(arr[j], arr[j + 1]) |

**Сортировка выбором**

Сортировка выбором (Selection Sort) — это алгоритм сортировки, который делит массив на две части: отсортированную и неотсортированную. На каждом шаге выбирается минимальный элемент из неотсортированной части и меняется местами с первым элементом этой части.

1. Проходим по массиву, находим минимальный элемент.
2. Меняем его местами с первым элементом неотсортированной части.
3. Повторяем для оставшейся неотсортированной части.

Псевдокод:

|  |
| --- |
| selectionSort(arr)  n = length(arr)  for i from 0 to n - 1  minIndex = i  for j from i + 1 to n - 1  if arr[j] < arr[minIndex]  minIndex = j  swap(arr[i], arr[minIndex]) |

**Сортировка слиянием**

Сортировка слиянием (Merge Sort) — это эффективный, но более сложный алгоритм сортировки. Он использует принцип "разделяй и властвуй": массив делится на две части, каждая из которых сортируется рекурсивно, а затем две отсортированные части сливаются.

1. Делим массив пополам.
2. Рекурсивно сортируем обе части.
3. Сливаем две отсортированные части в один.

Псевдокод:

|  |
| --- |
| mergeSort(arr)  if length(arr) > 1  mid = length(arr) / 2  leftHalf = arr[0 : mid]  rightHalf = arr[mid : length(arr)]  mergeSort(leftHalf)  mergeSort(rightHalf)  i = 0  j = 0  k = 0  while i < length(leftHalf) and j < length(rightHalf)  if leftHalf[i] < rightHalf[j]  arr[k] = leftHalf[i]  i = i + 1  else  arr[k] = rightHalf[j]  j = j + 1  k = k + 1  while i < length(leftHalf)  arr[k] = leftHalf[i]  i = i + 1  k = k + 1  while j < length(rightHalf)  arr[k] = rightHalf[j]  j = j + 1  k = k + 1 |

**Проектирование архитектуры приложения**

С# - очень удобный способ демонстрации работы сортировок. Возможность быстро и просто создавать работающие кнопки очень ускорит процесс создания работающего приложения. Более того, сортировки очень часто используют одни и те же функции, и удобность в исполнении ООП на C#, тоже окажет положительное влияние на реализацию приложения.

**Работа разработанного приложения**

При открытии приложения пользователь видит главное окно, на котором изображен случайно сгенерированный массив. На этом окне можно выбрать вид сортировки и количество элементов в массиве от 5 до 150. При нажатии кнопки «Сортировать массив» массив начнет сортироваться выбранным способом, помимо этого на окне будет изображено время сортировки и количество итераций.

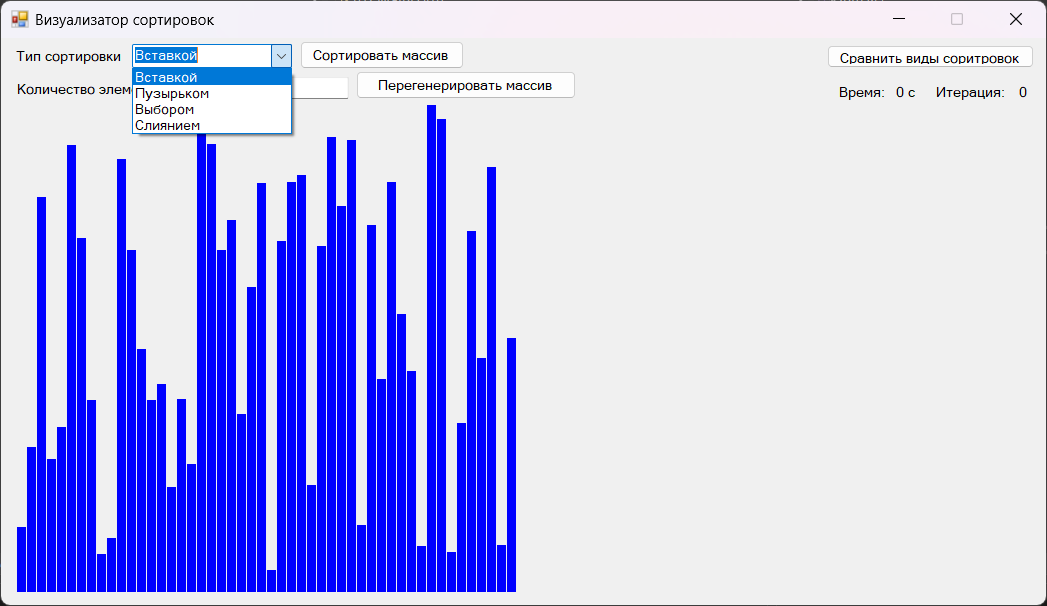


Рис. 1 - Главное окно

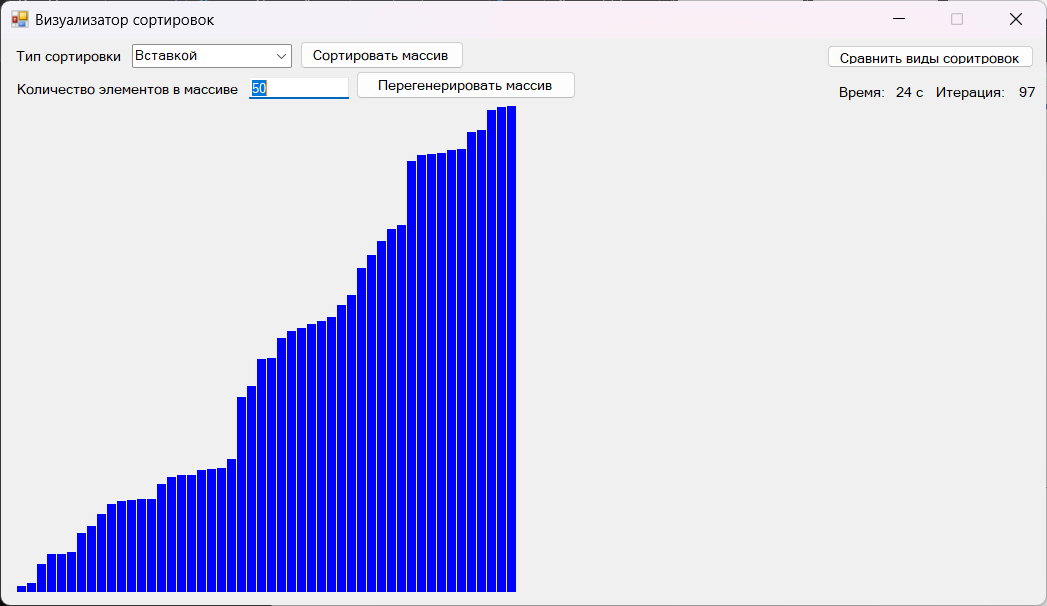


Рис. 2 - Массив после сортировки

При нажатии кнопки «Сравнить виды сортировок» откроется новое окно. На окне изображены случайно сгенерированные массивы для разных видов сортировок. При нажатии кнопки «Сравнить» массивы будут отсортированы. Также можно ввести количество элементов в массиве. На окне также изображены время и количество итераций для каждого вида сортировок.

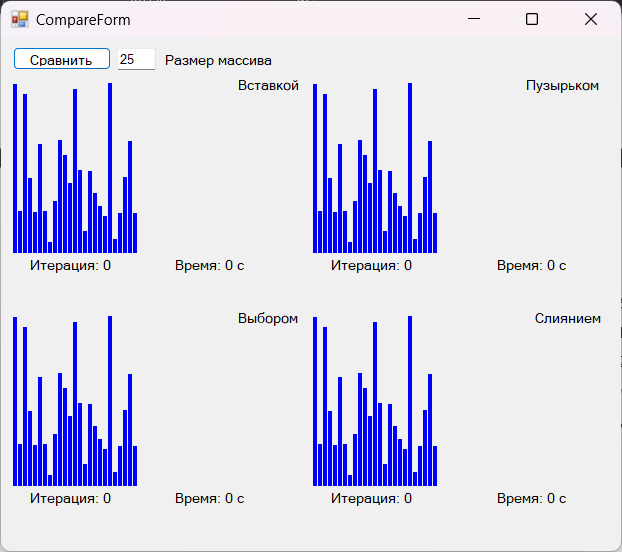


Рис. 3 - Окно сравнения видов сортировок

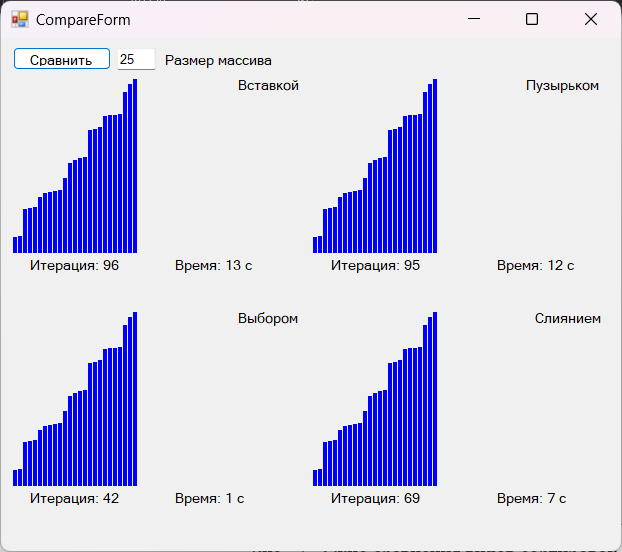


Рис. 4 - Отсортированные массивы

**Вывод**

В ходе выполнения данной расчётно-графической работы, мы более глубже познакомились с различными видами сортировок. Реализовали демонстрацию их работы в Windows Forms, для более наглядной картины происходящего.

Мы вновь убедились в актуальности знаний работы с сортировками. Наши демонстрационные модели позволяют сравнить между собой различные виды сортировок. Также на моделях наглядно видны различия в методах сортировок.

Наш проект еще есть куда усовершенствовать. Сортировок существует огромное количество, и мы до сих пор не разобрались даже с 10% от них. Стоит добавить таймер, который будет точно считать, сколько времени уходит на работу с массивами. Стоит добавить более красивую анимацию, чтобы пользователь (если не понимает, что происходит), мог разобраться в работе того или иного метода.

**Приложения**

Приложение 1 – Код основного окна

|  |
| --- |
| MainForm.cs |
| using System;  using System.Diagnostics;  using System.Drawing;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace rgz  {  public partial class MainForm : Form  {  private int[] dataArray;  private int arraySize;  private const int barWidth = 10;  private int iterCount = 0;  Stopwatch sw;  public MainForm()  {  InitializeComponent();  InitUI();  }  // Инициализация интерфейса пользователя  private void InitUI()  {  comboBoxSortType.Items.AddRange(new string[]  {  "Вставкой",  "Пузырьком",  "Выбором",  "Слиянием"  });  comboBoxSortType.SelectedIndex = 0;  buttonSort.Click += ButtonSort\_Click;  // Подписываемся на событие загрузки формы  this.Load += Form1\_Load;  }  // Обработчик события загрузки формы  private void Form1\_Load(object sender, EventArgs e)  {  // Генерируем массив данных после загрузки формы  GenerateRandomArray();  }  // Генерация случайного массива данных  private void GenerateRandomArray()  {  if (Int32.TryParse(textBoxArrElCnt.Text, out int arrSize))  {  if (arrSize <= 150 && arrSize >= 5)  {  arraySize = Int32.Parse(textBoxArrElCnt.Text);  dataArray = new int[arraySize];  if (panelVisualize == null)  {  return; // панель еще не инициализирована  }  Random rand = new Random();  for (int i = 0; i < arraySize; i++)  {  dataArray[i] = rand.Next(5, panelVisualize.Height);  }  panelVisualize.Invalidate();  }  else  {  MessageBox.Show("Массив должен иметь размер от 5 до 150!");  }  }  else  {  MessageBox.Show("Размером массива должно быть число!");  }  }  // Обработчик события нажатия кнопки "Сортировать"  private async void ButtonSort\_Click(object sender, EventArgs e)  {  iterCount = 0;  labelIterCnt.Text = iterCount.ToString();  timer1.Equals(0);  timer1.Start();  buttonSort.Enabled = false;  buttonRegenArr.Enabled = false;  sw = new Stopwatch();  sw.Start();  switch (comboBoxSortType.SelectedItem.ToString())  {  case "Вставкой":  await InsertionSort();  break;  case "Пузырьком":  await BubbleSort();  break;  case "Выбором":  await SelectionSort();  break;  case "Слиянием":  await MergeSort(0, dataArray.Length - 1);  break;  }  buttonSort.Enabled = true;  buttonRegenArr.Enabled = true;  timer1.Stop();  sw.Stop();  }  // Метод сортировки вставками  private async Task InsertionSort()  {  for (int i = 1; i < dataArray.Length; i++)  {  iterCount++;  labelIterCnt.Text = iterCount.ToString();  int key = dataArray[i];  int j = i - 1;  while (j >= 0 && dataArray[j] > key)  {  dataArray[j + 1] = dataArray[j];  j--;  panelVisualize.Invalidate();  await Task.Delay(50);  }  dataArray[j + 1] = key;  panelVisualize.Invalidate();  await Task.Delay(50);  }  }  // Метод сортировки пузырьком  private async Task BubbleSort()  {  for (int i = 0; i < dataArray.Length - 1; i++)  {  iterCount++;  labelIterCnt.Text = iterCount.ToString();  for (int j = 0; j < dataArray.Length - i - 1; j++)  {  if (dataArray[j] > dataArray[j + 1])  {  int temp = dataArray[j];  dataArray[j] = dataArray[j + 1];  dataArray[j + 1] = temp;  panelVisualize.Invalidate();  await Task.Delay(50);  }  }  }  }  // Метод сортировки выбором  private async Task SelectionSort()  {  for (int i = 0; i < dataArray.Length - 1; i++)  {  iterCount++;  labelIterCnt.Text = iterCount.ToString();  int minIndex = i;  for (int j = i + 1; j < dataArray.Length; j++)  {  if (dataArray[j] < dataArray[minIndex])  {  minIndex = j;  }  }  int temp = dataArray[minIndex];  dataArray[minIndex] = dataArray[i];  dataArray[i] = temp;  panelVisualize.Invalidate();  await Task.Delay(50);  }  }  // Метод сортировки слиянием  private async Task MergeSort(int left, int right)  {  iterCount++;  labelIterCnt.Text = iterCount.ToString();  if (left < right)  {  int middle = (left + right) / 2;  await MergeSort(left, middle);  await MergeSort(middle + 1, right);  await Merge(left, middle, right);  }  }  // Метод слияния массивов для сортировки слиянием  private async Task Merge(int left, int middle, int right)  {  int leftArraySize = middle - left + 1;  int rightArraySize = right - middle;  int[] leftArray = new int[leftArraySize];  int[] rightArray = new int[rightArraySize];  Array.Copy(dataArray, left, leftArray, 0, leftArraySize);  Array.Copy(dataArray, middle + 1, rightArray, 0, rightArraySize);  int i = 0, j = 0, k = left;  while (i < leftArraySize && j < rightArraySize)  {  if (leftArray[i] <= rightArray[j])  {  dataArray[k++] = leftArray[i++];  }  else  {  dataArray[k++] = rightArray[j++];  }  panelVisualize.Invalidate();  await Task.Delay(50);  }  while (i < leftArraySize)  {  dataArray[k++] = leftArray[i++];  panelVisualize.Invalidate();  await Task.Delay(50);  }  while (j < rightArraySize)  {  dataArray[k++] = rightArray[j++];  panelVisualize.Invalidate();  await Task.Delay(50);  }  }  // Обработчик события рисования панели визуализации  private void panelVisualize\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics;  for (int i = 0; i < dataArray.Length; i++)  {  g.FillRectangle(Brushes.Blue, i \* barWidth, panelVisualize.Height - dataArray[i], barWidth - 1, dataArray[i]);  }  }  // Обработчик события нажатия кнопки "Регенерировать массив"  private void buttonRegenArr\_Click(object sender, EventArgs e)  {  if (Int32.TryParse(textBoxArrElCnt.Text, out int arrSize))  {  if (arrSize <= 150 && arrSize >= 5)  {  arraySize = Int32.Parse(textBoxArrElCnt.Text);  GenerateRandomArray();  }  else  {  MessageBox.Show("Массив должен иметь размер от 5 до 150!");  }  }  else  {  MessageBox.Show("Размером массива должно быть число!");  }  }  // Обработчик события изменения значения в выпадающем списке  private void comboBoxSortType\_SelectedValueChanged(object sender, EventArgs e)  {  GenerateRandomArray();  }  // Обработчик события тика таймера  private void timer1\_Tick(object sender, EventArgs e)  {  labelTimer.Text = sw.Elapsed.Seconds.ToString() + " с";  }  // Обработчик события нажатия кнопки "Сравнить сортировки"  private void buttonCompareSorts\_Click(object sender, EventArgs e)  {  CompareForm compareForm = new CompareForm();  compareForm.Show();  this.Hide();  }  }  } |

Приложение 2 – Код окна сравнения сортировок

|  |
| --- |
| CompareForm.cs |
| using System;  using System.Diagnostics;  using System.Drawing;  using System.Threading.Tasks;  using System.Windows.Forms;  namespace rgz  {  public partial class CompareForm : Form  {  private int[] dataArray;  private const int barWidth = 5;  private int arraySize = 25;  private int iterCount = 0;  Stopwatch swIns;  Stopwatch swBub;  Stopwatch swSel;  Stopwatch swMer;  private int[] insArray;  private int[] bubArray;  private int[] selArray;  private int[] merArray;  public CompareForm()  {  InitializeComponent();  }  // Метод для генерации случайного массива данных  private void GenerateRandomArray()  {  dataArray = new int[arraySize];  if (panelInsertion == null || panelBubble == null || panelSelection == null || panelMerge == null)  {  return; // панель еще не инициализирована  }  Random rand = new Random();  for (int i = 0; i < arraySize; i++)  {  dataArray[i] = rand.Next(5, panelInsertion.Height);  }  panelInsertion.Invalidate();  panelBubble.Invalidate();  panelSelection.Invalidate();  panelMerge.Invalidate();  }  // Метод для копирования массива данных  private int[] CopyArray()  {  int[] copy = new int[dataArray.Length];  Array.Copy(dataArray, copy, dataArray.Length);  return copy;  }  // Метод для сортировки вставкой  private async Task InsertionSort(int[] array, Panel panel)  {  timerIns.Start();  swIns = new Stopwatch();  swIns.Start();  for (int i = 1; i < array.Length; i++)  {  iterCount++;  labelInsIter.Text = "Итерация: " + iterCount.ToString();  int key = array[i];  int j = i - 1;  while (j >= 0 && array[j] > key)  {  array[j + 1] = array[j];  j--;  panel.Invalidate();  await Task.Delay(50);  }  array[j + 1] = key;  panel.Invalidate();  await Task.Delay(50);  }  timerIns.Stop();  swIns.Stop();  }  // Метод для пузырьковой сортировки  private async Task BubbleSort(int[] array, Panel panel)  {  timerBub.Start();  swBub = new Stopwatch();  swBub.Start();  for (int i = 0; i < array.Length - 1; i++)  {  iterCount++;  labelBubIter.Text = "Итерация: " + iterCount.ToString();  for (int j = 0; j < array.Length - i - 1; j++)  {  if (array[j] > array[j + 1])  {  int temp = array[j];  array[j] = array[j + 1];  array[j + 1] = temp;  panel.Invalidate();  await Task.Delay(50);  }  }  }  timerBub.Stop();  swBub.Stop();  }  // Метод для сортировки выбором  private async Task SelectionSort(int[] array, Panel panel)  {  timerSel.Start();  swSel = new Stopwatch();  swSel.Start();  for (int i = 0; i < array.Length - 1; i++)  {  iterCount++;  labelSelIter.Text = "Итерация: " + iterCount.ToString();  int minIndex = i;  for (int j = i + 1; j < array.Length; j++)  {  if (array[j] < array[minIndex])  {  minIndex = j;  }  }  int temp = array[minIndex];  array[minIndex] = array[i];  array[i] = temp;  panel.Invalidate();  await Task.Delay(50);  }  timerSel.Stop();  swSel.Stop();  }  // Метод для сортировки слиянием  private async Task MergeSort(int[] array, int left, int right, Panel panel)  {  if (left == 0 && right == array.Length - 1)  {  timerMer.Start();  swMer = new Stopwatch();  swMer.Start();  }  if (left < right)  {  int middle = (left + right) / 2;  await MergeSort(array, left, middle, panel);  await MergeSort(array, middle + 1, right, panel);  iterCount++;  labelMerIter.Text = "Итерация: " + iterCount.ToString();  await Merge(array, left, middle, right, panel);  }  if (left == 0 && right == array.Length - 1)  {  timerMer.Stop();  swMer.Stop();  }  }  // Метод для слияния массивов  private async Task Merge(int[] array, int left, int middle, int right, Panel panel)  {  int leftArraySize = middle - left + 1;  int rightArraySize = right - middle;  int[] leftArray = new int[leftArraySize];  int[] rightArray = new int[rightArraySize];  Array.Copy(array, left, leftArray, 0, leftArraySize);  Array.Copy(array, middle + 1, rightArray, 0, rightArraySize);  int i = 0, j = 0, k = left;  while (i < leftArraySize && j < rightArraySize)  {  if (leftArray[i] <= rightArray[j])  {  array[k++] = leftArray[i++];  }  else  {  array[k++] = rightArray[j++];  }  panel.Invalidate();  await Task.Delay(50);  }  while (i < leftArraySize)  {  array[k++] = leftArray[i++];  panel.Invalidate();  await Task.Delay(50);  }  while (j < rightArraySize)  {  array[k++] = rightArray[j++];  panel.Invalidate();  await Task.Delay(50);  }  }  // Отрисовка панели для сортировки вставкой  private void panelInsertion\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics;  for (int i = 0; i < insArray.Length; i++)  {  g.FillRectangle(Brushes.Blue, i \* barWidth, panelInsertion.Height - insArray[i], barWidth - 1, insArray[i]);  }  }  // Отрисовка панели для пузырьковой сортировки  private void panelBubble\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics;  for (int i = 0; i < bubArray.Length; i++)  {  g.FillRectangle(Brushes.Blue, i \* barWidth, panelBubble.Height - bubArray[i], barWidth - 1, bubArray[i]);  }  }  // Отрисовка панели для сортировки выбором  private void panelSelection\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics;  for (int i = 0; i < selArray.Length; i++)  {  g.FillRectangle(Brushes.Blue, i \* barWidth, panelSelection.Height - selArray[i], barWidth - 1, selArray[i]);  }  }  // Отрисовка панели для сортировки слиянием  private void panelMerge\_Paint(object sender, PaintEventArgs e)  {  Graphics g = e.Graphics;  for (int i = 0; i < merArray.Length; i++)  {  g.FillRectangle(Brushes.Blue, i \* barWidth, panelMerge.Height - merArray[i], barWidth - 1, merArray[i]);  }  }  // Обработка нажатия на кнопку для начала сортировки  private async void buttonShowSorts\_Click(object sender, EventArgs e)  {  if (Int32.TryParse(textBoxArrSize.Text, out int arrSize))  {  if (arrSize <= 25 && arrSize >= 5)  {  arraySize = Int32.Parse(textBoxArrSize.Text);  GenerateRandomArray();  iterCount = 0;  insArray = CopyArray();  bubArray = CopyArray();  selArray = CopyArray();  merArray = CopyArray();  var insTask = InsertionSort(insArray, panelInsertion);  var bubTask = BubbleSort(bubArray, panelBubble);  var selTask = SelectionSort(selArray, panelSelection);  var merTask = MergeSort(merArray, 0, merArray.Length - 1, panelMerge);  await Task.WhenAll(insTask, bubTask, selTask, merTask);  }  else  {  MessageBox.Show("Массив должен иметь размер от 5 до 25!");  }  }  else  {  MessageBox.Show("Размером массива должно быть число!");  }  }  // Метод, который выполняется при загрузке формы  private void CompareForm\_Load(object sender, EventArgs e)  {  GenerateRandomArray();  insArray = CopyArray();  bubArray = CopyArray();  selArray = CopyArray();  merArray = CopyArray();  }  // Метод, который выполняется при закрытии формы  private void CompareForm\_FormClosed(object sender, FormClosedEventArgs e)  {  MainForm mainForm = new MainForm();  mainForm.Show();  }  // Обновление времени для сортировки вставкой  private void timerIns\_Tick(object sender, EventArgs e)  {  labelInsTimer.Text = "Время: " + swIns.Elapsed.Seconds.ToString() + " с";  }  // Обновление времени для пузырьковой сортировки  private void timerBub\_Tick(object sender, EventArgs e)  {  labelBubTimer.Text = "Время: " + swBub.Elapsed.Seconds.ToString() + " с";  }  // Обновление времени для сортировки выбором  private void timerSel\_Tick(object sender, EventArgs e)  {  labelSelTimer.Text = "Время: " + swSel.Elapsed.Seconds.ToString() + " с";  }  // Обновление времени для сортировки слиянием  private void timerMer\_Tick(object sender, EventArgs e)  {  labelMergeTimer.Text = "Время: " + swMer.Elapsed.Seconds.ToString() + " с";  }  }  } |

Приложение 3 – Техническое задание

|  |
| --- |
| ТЕХНИЧЕСКОЕ ЗАДАНИЕ  для «Программы визуализатора процесса сортировки»  **1 Введение**  **1.1 Наименование программы**  Наименование программы – «Программа-визуализатор процесса сортировки».  **1.2 Краткая характеристика области применения**  Программа-визуализатор процесса сортировки может быть применена в следующих областях:   1. **Образование**: В учебных заведениях, от начальной школы до университета, визуализаторы сортировки используются для демонстрации и объяснения алгоритмов сортировки. Они помогают студентам лучше понять принципы работы различных методов сортировки, таких как сортировка пузырьком, сортировка вставками, быстрая сортировка и т. д. 2. **Научные исследования**: Визуализаторы могут быть использованы исследователями в области компьютерных наук для изучения производительности и поведения алгоритмов сортировки при различных условиях. Это может помочь в оптимизации алгоритмов для конкретных сценариев использования или улучшении их эффективности. 3. **Разработка программного обеспечения**: Программисты могут использовать визуализаторы для отладки и тестирования своих собственных реализаций алгоритмов сортировки. Визуализация шагов сортировки помогает им лучше понять, как именно работает их код, а также выявить и исправить ошибки. 4. **Обучение программированию**: Визуализаторы могут быть полезны для начинающих программистов, которые только начинают изучать алгоритмы сортировки. Они могут использовать визуализаторы для экспериментирования с различными алгоритмами и наблюдения за тем, как они работают в реальном времени. 5. **Популяризация науки и образование общественности**: Визуализаторы сортировки могут быть использованы в образовательных программах или мероприятиях для привлечения внимания к компьютерным наукам и алгоритмам. Они могут помочь людям без технического образования лучше понять сложные концепции через визуальное представление.   **2 Назначение разработки**  Программа будет использоваться в учебном заведении для объяснения обучающимся алгоритмов разработки.  **2.1 Функциональное назначение**  Предоставление пользователю инструмента для визуализации шагов алгоритмов сортировки, позволяющего легко понять и изучить процесс сортировки данных.  **2.2 Эксплуатационное назначение**  Использование в учебных целях для демонстрации алгоритмов сортировки, исследования и сравнения их производительности, отладки программного кода и образовательных мероприятий по популяризации компьютерных наук.  **3 Требования к программе или программному изделию**  **3.1 Требования к функциональным характеристикам**  **3.1.1 Требования к составу выполняемых функций**  Программа-визуализатор должна включать в себя следующие функции:   1. Программа должна отображать исходный набор данных, который будет сортироваться. 2. Способность демонстрировать шаги алгоритма сортировки визуально, позволяя пользователям наблюдать изменения в данных на каждом этапе. 3. Возможность выбора различных алгоритмов сортировки для визуализации, таких как сортировка пузырьком, сортировка вставками, быстрая сортировка и другие. 4. Возможность управления процессом визуализации, такая как пауза, продолжение, шаг вперед и назад, что позволит пользователям более детально изучить каждый шаг алгоритма. 5. Показ времени, затраченного на выполнение каждого шага алгоритма, чтобы пользователи могли сравнивать производительность различных методов сортировки. 6. После завершения сортировки программа должна отобразить отсортированный набор данных и, по возможности, вывести статистику о процессе сортировки. 7. Обеспечение интерактивного пользовательского интерфейса, который позволит пользователям легко взаимодействовать с программой и настраивать параметры визуализации. 8. Возможность визуализации сортировки различных типов данных, таких как числа, строки или пользовательские объекты.   **3.1.2 Требования к организации входных и выходных данных**   **Входные данные**:   * **Набор данных для сортировки**: Пользователь должен иметь возможность ввести или загрузить набор данных, который будет подвергнут сортировке. Этот набор данных может быть числовым, текстовым или состоять из пользовательских объектов. * **Выбор алгоритма сортировки**: Пользователь должен иметь возможность выбрать алгоритм сортировки, который будет визуализироваться над его входными данными.    **Выходные данные**:   * **Визуализация процесса сортировки**: Программа должна визуально отображать процесс сортировки, позволяя пользователю наблюдать изменения в данных на каждом этапе. * **Отсортированный набор данных**: После завершения сортировки программа должна отобразить отсортированный набор данных. * **Статистика выполнения**: При желании пользователь может получить статистику о процессе сортировки, такую как количество операций или время выполнения.   **3.2 Требования к надежности**  Программа должна быть стабильной и не подверженной критическим сбоям в процессе работы. Она должна корректно обрабатывать непредвиденные ситуации, такие как ошибки ввода данных или прерывания в работе.  **3.2.1 Требования к обеспечению надежного (устойчивого) функционирования программы**  Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:   **Обработка ошибок и исключений**: Программа должна корректно обрабатывать ошибки, исключения и непредвиденные ситуации, чтобы избежать краха и предоставить пользователю информацию о проблеме.   **Стабильность и отказоустойчивость**: Программа должна быть способна продолжать работу в случае возникновения ошибок или сбоев. Это может включать в себя автоматическое восстановление после сбоя или сохранение промежуточных результатов.   **Тестирование**: Программа должна быть тщательно протестирована на различных сценариях использования и наборах данных, чтобы обнаружить и устранить потенциальные проблемы и ошибки.  **3.3 Условия эксплуатации**  Программа (клиент) запускается на компьютере пользователя. Окно программы должно быть открыто на весь экран, должна быть возможность закрыть, свернуть приложение или запустить любое стороннее программное обеспечение.  **3.3.1 Климатические условия эксплуатации**  Специальные условия не требуются.  **3.3.2 Требования к видам обслуживания**  Программа не требует проведения каких-либо видов обслуживания.  **3.4 Требования к составу и параметрам технических средств**  Требования отсутствуют.  **3.5 Требования к информационной и программной совместимости**  Программа должна быть совместима с различными операционными системами, включая Windows, macOS и различные дистрибутивы Linux, чтобы пользователи могли использовать ее на своих предпочтительных платформах.  **3.6 Требование к маркировке и упаковке**  Программное изделие передается по сети Internet в виде архива — загружается с официального сайта производителя. Специальных требований к маркировке не предъявляется. Для проверки подлинности программного обеспечения рекомендуется проверять контрольные суммы загруженных файлов со значениями, указанными на официальном сайте.  **3.7 Требования к транспортированию и хранению**  Специальных требований не предъявляется.  **3.8 Специальные требования**  Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса, разработанного согласно рекомендациям компании-производителя операционной системы.  **4 Требования к программной документации**  Предварительный состав программной документации:   * техническое задание (включает описание применения); * программа и методика испытаний; * руководство программиста; * ведомость эксплуатационных документов; * формуляр.   **5 Технико-экономические показатели**  Потребность в программе может быть высокой, так как она будет распространяться бесплатно.  **6 Стадии и этапы разработки**  Разработка должна быть проведена в три стадии:   1. техническое задание; 2. технический (и рабочий) проекты; 3. внедрение.   На стадии «Техническое задание» должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.  На стадии «Технический (и рабочий) проект» должны быть выполнены перечисленные ниже этапы работ:   * разработка программы; * разработка программной документации; * испытания программы.   На стадии «Внедрение» должен быть выполнен этап разработки «Подготовка и передача программы».  Содержание работ по этапам: На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:   * постановка задачи; * определение и уточнение требований к техническим средствам; * определение требований к программе; * определение стадий, этапов и сроков разработки программы и документации на нее; * согласование и утверждение технического задания.   На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.  На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями ГОСТ 19.101-77.  На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:   * разработка, согласование и утверждение порядка и методики испытаний; * проведение приемо-сдаточных испытаний; * корректировка программы и программной документации по результатам испытаний.   На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах заказчика.  **7 Порядок контроля и приемки**  Приемосдаточные испытания программы должны проводиться согласно разработанной исполнителем и согласованной заказчиком «Программы и методики испытаний».  Ход проведения приемо-сдаточных испытаний заказчик и исполнитель документируют в протоколе испытаний. На основании протокола испытаний исполнитель совместно с заказчиком подписывают акт приемки-сдачи программы в эксплуатацию. |

**Источники**

1. Левитин А. В. Глава 3. Метод грубой силы: Сортировка выбором // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 143—144. — 576 с. — ISBN 978-5-8459-0987-9
2. *Н. Вирт.* Алгоритмы и структуры данных. — М.: ДМК Пресс, 2010. — С. 74. — 272 с. — ISBN 987-5-94074-584-6.
3. Левитин А. В. Глава 7. Пространственно-временной компромисс: Сортировка подсчётом // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 331—339. — 576 с.
4. Кнут Д. Э. 5.2 Внутренняя сортировка // 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.
5. Вирт Н.Алгоритмы и структуры данных. Новая версия для Оберона / Н.Вирт - М.: ДМК Пресс, 2010.- 272 с.
6. Вирт Н.Алгоритмы и структуры данных. / Н.Вирт - М.: Мир, 1989.- 360 с.
7. Вирт Н. Алгоритмы + структуры данных = программы / Н.Вирт - М.: Мир, 1985.- 406 с.
8. Нобак, М. Принципы разработки программных пакетов: практическое пособие / М. Нобак; пер. с англ. Д. А. Беликова. - Москва: ДМК Пресс, 2020. - 274 с.
9. Керниган Брайан У. Практика программирования [Текст]/ Брайан У. Керниган, Роб Пайк – М.: Вильямс, 2019.- 228 с.
10. Маклин Холл Гэри Адаптивный код: гибкое кодирование с помощью паттернов проектирования и принципов SOLID [Текст]/ Холл Гэри Маклин - М.: Вильямс, 2017.- 448 с.
11. Константайн Л. Разработка программного обеспечения [Текст]/ Л.  [Константайн](https://www.ozon.ru/person/konstantayn-l-1663927/), Л. [Локвуд](https://www.ozon.ru/person/lokvud-l-1663931/) – СПб.: Питер, 2004.- 592 с.