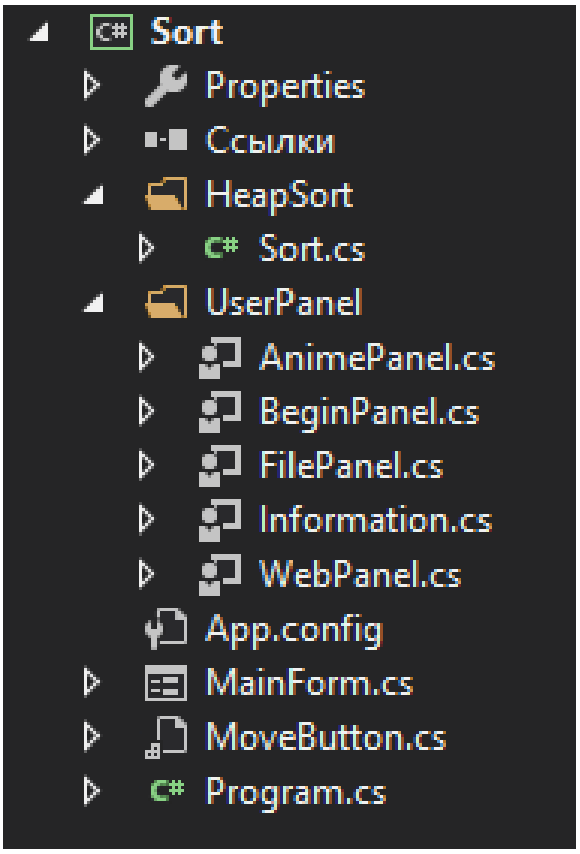
Руководство программиста

HeapSort

Оглавление

Краткое описание всех фалов и особенности приложения.



Все файлы приложения

- Класс *Sort*, содержит в себе реализацию алгоритма пирамидальной сортировки, а также заполнение и считывание текса с RichTextBox.
- Класс *MoveButton*, это наследованный класс от Button. Реализовано продвижение кнопок по панели для создания анимации.
- В bin\Icon хранятся все иконки приложения.
- В bin\Debug есть файл grafPoint.txt откуда считывается все точки и их значения графика практического времени выполнения алгоритма. (Также в основной папке есть тоже файл grafPoint.txt, однако из-за бага github.com без него программа оказывается запускаться).

Пользовательский интерфейсы:

- *BeginPanel*, начальная панель, которая загружается при старте программы. Реализовано считывание данных из RichTextBox и дальнейшая сортировка, а также случайное заполнение и вывод кол-ва затраченного времени и кол-во элементов в последовательности, как сгенерированной, так и в вручную введенной.

- *FilePanel*, панель, имеющая функцию ввода и вывода с файла, а также реализация графика теоретического и практического времени работы алгоритма.

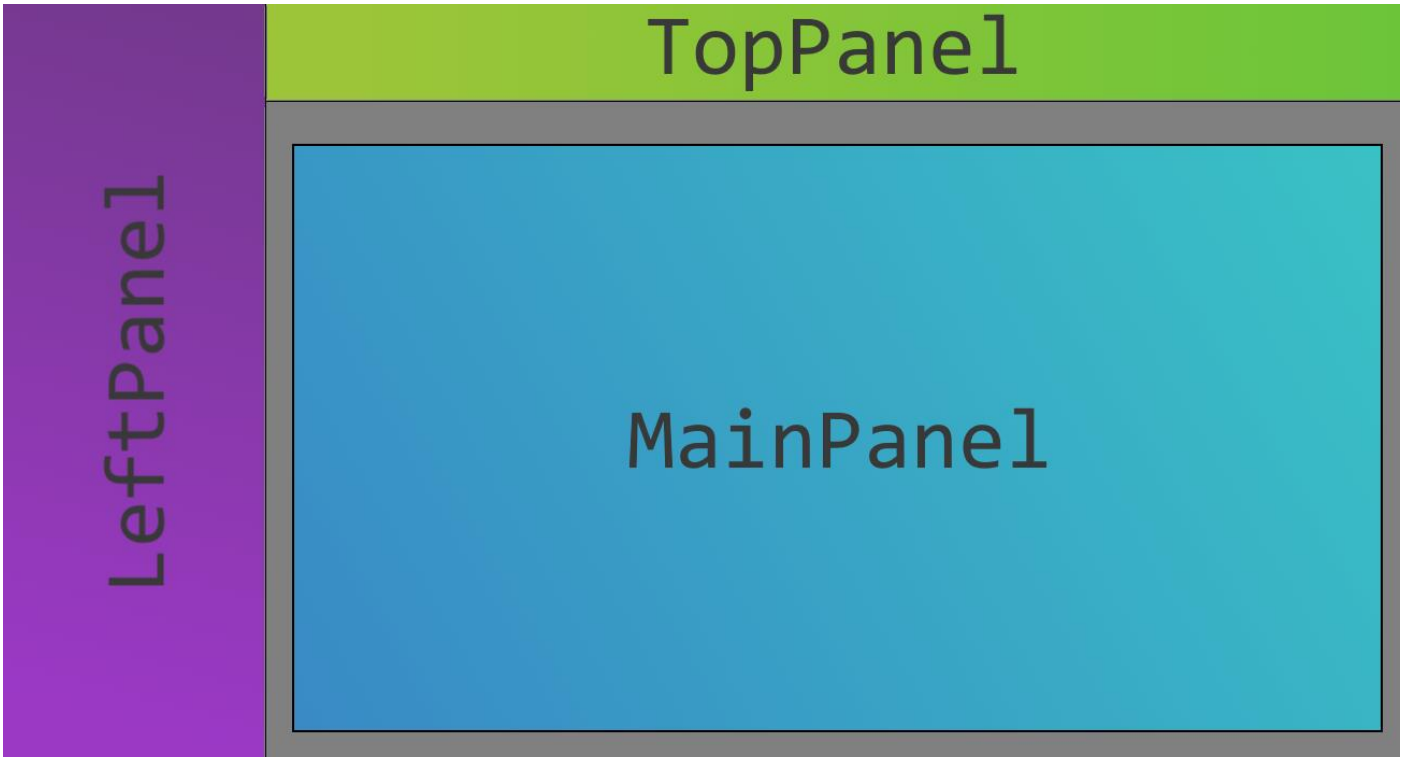
- *AnimePanel*, панель с реализованной пошаговой анимацией алгоритма сортировки кучей.

- *WebPanel*, панель с браузером и погружаемой странней о пирамидальные сортировки с «neerc.ifmo.ru».

- *Information*, панель с информацией о приложении и создателях.

MainForm

Для создания приложения Windows была использован стандартный Form (Net Framework). Был измене параметр FromBorderStyle = None. В дальнейшем окно было разделено на 3 основные панели при помощи элемента Panel.



MainPanel служит для отображения пользвоательских интерфейсов.

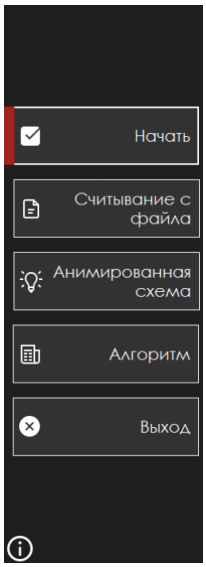
На LeftPanel были размещены кнокпи переключения между пользовательскими интерфейсами. Также было додавлено событие MouseDown и MouseMove для перемещение окна по экрану.

```
private int x;
private int y;

private void LeftPanel_MouseDown(object sender, MouseEventArgs e)
{
    x = e.X; y = e.Y;
}

private void LeftPanel_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        this.Left = (this.Left + e.X) - x;
        this.Top = (this.Top + e.Y) - y;
    }
}
```

TopPanel как и в LeftPanel аналагично реализованно перемещение формы по экрану.



Кнопки LeftPanel:

- Кнопка «Начать» вызывает пользовательский интерес (ПИ) BeginPanel.
- Кнопка «Считывание с файла» вызывает пользовательский интерес (ПИ) FilePanel.
- Кнопка «Анимированная схема» вызывает пользовательский интерес (ПИ) AnimePanel.
- Кнопка «Алгоритм» вызывает пользовательский интерес (ПИ) WebPanel.
- Кнопка «Выход» останавливает работу приложения.
- Кнопка «Info» вызывает пользовательский интерес (ПИ) Information.

Так же есть элемент SidePanel это красная черта, которая изменяет положение после нажатия на кнопку для однозначного определения где пользователь находить.

```
SidePanel.Height = BeginButton.Height;
SidePanel.Top = BeginButton.Top;
```

Реализация кода на примере перемещения ко кнопке «Начать».

ПИ вызываются при помощи метода BringToFront().

Так же в кнопке выхода ралезованно сохранение из оперативной памяти всех точек теоричического времнеи работы алгоритма

```
StreamWriter sw = new StreamWriter("grafPoint.txt");
sw.WriteLine(UserPanel.FilePanel.Num);
foreach (var item in UserPanel.FilePanel.Graf)
    sw.WriteLine("{0} {1}", item.Key, item.Value);
sw.Close();
Application.Exit();
```

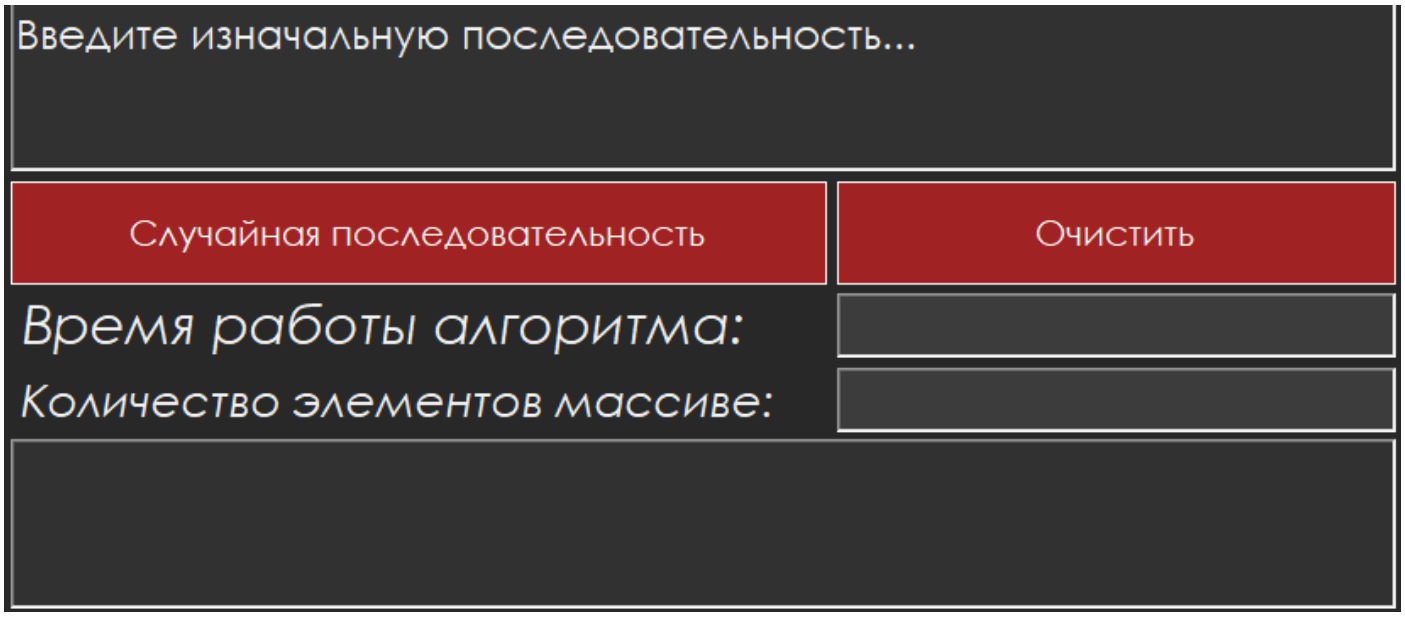
Так же при запуски приложения из файла grafPoint.txt считыаютсчя все значения для построение графика. Подробнее дальше.

ПИ BeginPanel

ПИ разделен на 5 зон (панелей). В самой центральной зоне находиться для RichTextBox. На ввод последовательности и на вывод уже отсортированной последовательности.

В самом верхнем RichTextBox написано «Введите изначальную последовательность...». После нажатия на панель это фраза исчезает. Реализовано это с помощью глобального флага FirstClick если на панель ввода нажил первый раз, то она очищается и флаг становиться false. После этого при повторном нажатии текст не исчезнет. Однако ели нажать кнопку «Очистить» то флаг восстановить значение на true и вернет надпись: «Введите изначальную последовательность...».

Также есть еще два поменьше RichTextBox которые отвечают за вывод кол-ва элементов и времени работы алгоритма в тиках.



Кнопка «Очистить» просто очищает все RichTextBox о значений которые могу они в себе содержать.

```
FirstClick = true;
MasReadTextBox.Text = null;
MasReadTextBox.Text = "Введите изначальную последовательность...";
MasWriteTextBox.Text = null;
TimerSec.Text = null;
CountArray.Text = null;
```

Так же случайна генерация последовательности работает вот так.

```
FirstClick = false;
Random rnd = new Random();
int n = rnd.Next(1, 30);
string line = null;
for (int i = 0; i < n - 1; i++)
{
    line += rnd.Next(-1000, 1000);
    if (i % 10 == 0 && i != 0)
        line += '\n';
    else
        line += " ";
}
line += rnd.Next(-10000, 10000);
MasReadTextBox.Text = line;
```

Мы заводим генерируем число n которое отвечает за количество элементов в последовательности. Создаем line и путем генерации чисел от -1000 до 1000 помещаем в строчку. Если элемент, который мы помещаем имеет вид {10*n\ 0}. То есть является по номеру 10 в строке, на которой сейчас находится генерация, то алгоритм переносит его на новую. Алгоритм работает до n - 1 потому, что в конец возможен такой случай, что добавится или пробел, или символ переноса строки. Это было сделана чтобы в дальнейшем при вызове Split() программа не выдавала ошибку.

ПИ FilePanel

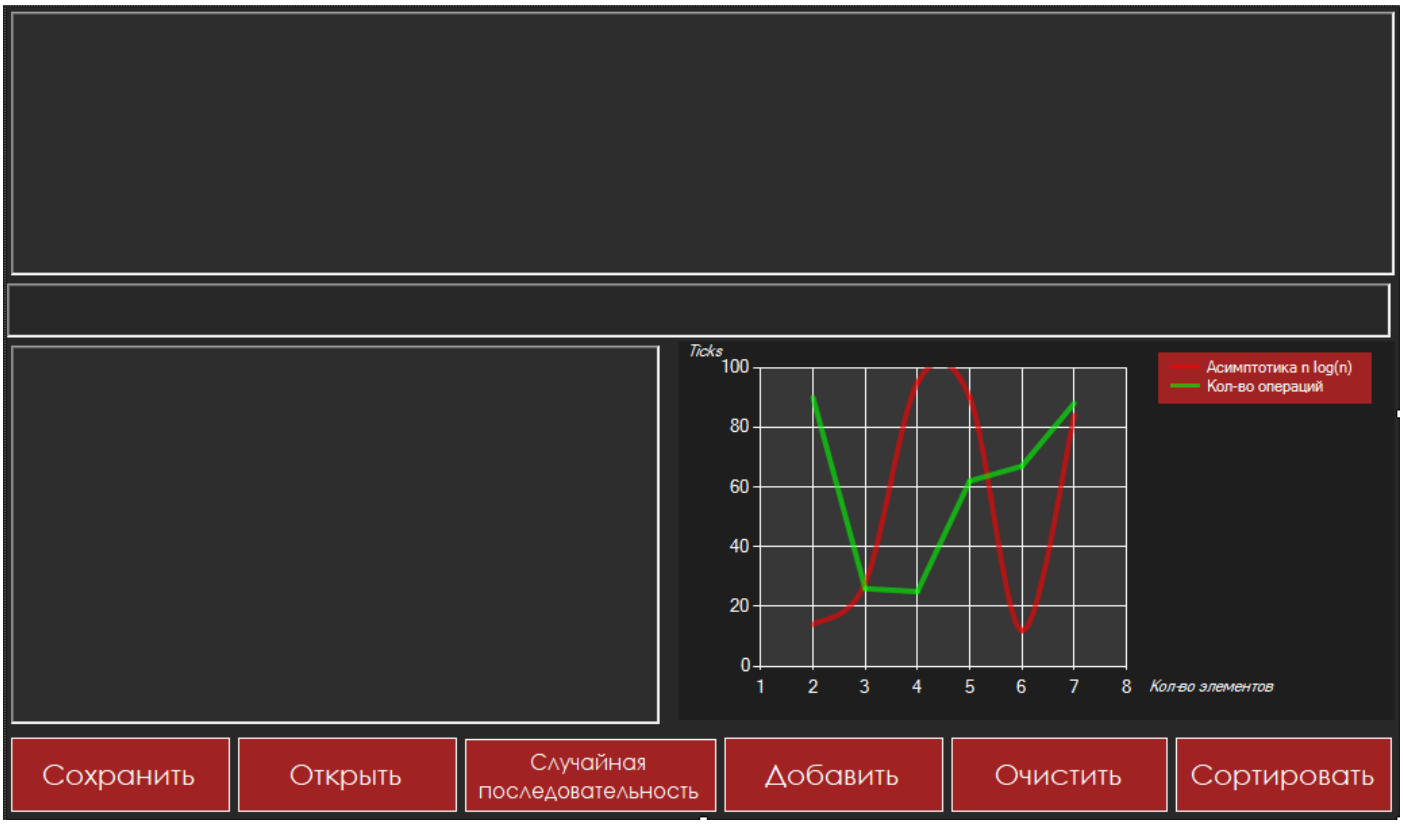
ПИ также, как и предыдущая разделан на зоны. В ней есть 3 RichTextBox. Самый верхний отвечает за ввод последовательности, маленький за кол-во тиков во время работы алгоритма, и левый снизу за отсортированный массив.

Так же на ПИ есть элемент Chart где рисуется график теоретической асимптоты $n\log(n)$, а также практического времени работы.

При запуски этого ПИ для начала отрисовывается график $n\log(n)$, а потом уже из файла grafPoint.txt в grafPoint.txt в bin\Debug берутся точки второго графика.

```
300
1 7
2 6
3 7
6 15
7 18
9 23
11 24
18 36
25 53
```

Файл grafPoint.txt содержит в себе на первой строке кол-во точек и на следующих их координаты. После загрузки приложения программа считвает весь файл grafPoint.txt в глобальный SortedDictionary<int, int> Graf. Так же после закрытие приложения файл grafPoint.txt удаляется и перезаписывается заного выгружая все значения, которые находятся в словаре Graf.



Так же парочку слов о расчёте практического времени. Используется библиотека System.Diagnostics, а если точнее, то класс Stopwatch.

```
NewMasTextBox.Clear();
Stopwatch time = new Stopwatch();
long[] array = HeapSort.Sort.arrayReadText(MainMasTextBox.Text);
time.Start();
HeapSort.Sort.heapSort(ref array);
time.Stop();
string temp = HeapSort.Sort.outputArray(ref array);
algoTime = time.Elapsed.Ticks.ToString();
```

Вызывается метод Start() до начала вызова фикции сортировки и поле вызывается метод Stop(). При первом запуски программы и рассчитывайте времени программа сперва покажет «космические» значения, однако если еще раз пересчитать кол-во тиков, то они изменяться до вполне вменяемого значения.

```
private void Open_Click(object sender, EventArgs e)
{
    OpenFileDialog OFD = new OpenFileDialog();
    OFD.Filter = "(Текстовый файл: )|*.txt";

    if (OFD.ShowDialog() == DialogResult.OK)
        MainMasTextBox.Text = File.ReadAllText(OFD.FileName);
}

private void Save_Click(object sender, EventArgs e)
{
    SaveFileDialog SFD = new SaveFileDialog();
    SFD.FileName = "Unnamed.txt";
    SFD.Filter = "(Текстовый файл: )|*.txt";
    if (SFD.ShowDialog() == DialogResult.OK)
        File.WriteAllText(SFD.FileName, NewMasTextBox.Text);
}
```

На нижней панели расположены кнопки «Открыть» и «Сохранить». Обычная и стандартная реализация кнопок сохранение и открытия.

Кнопка «Случайная последовательность» работает точно также, как и в предыдущем ПИ, за исключением пар мелочей.

```
private void RandomButton_Click(object sender, EventArgs e)
{
    NewMasTextBox.Clear();
    TimerSec.Clear();
    Random rnd = new Random();
    int n = rnd.Next(1, 1000);
    string line = null;
    for (int i = 0; i < n - 1; i++)
    {
        line += rnd.Next(-100000, 100000);
        if (i % 8 == 0 && i != 0)
            line += '\n';
        else
            line += " ";
    }
    line += rnd.Next(-10000, 10000);
    MainMasTextBox.Text = line;
}
```

Кнопка «Добавить» добавляет на график практической асимптотики новую точку или перезаписывает уже существующей в соответствии со значением кол-ва тиков для определённого кол-ва элементов. Так же реализована защита от добавления комических значений при помощи флага. Первое значение измерения кол-ва тиков не вноситься.

```
private void AddBottun_Click(object sender, EventArgs e)
{
    if (FirstClicl)
    {
        AddGraf(HeapSort.Count.size, int.Parse(algoTime));
        PaintGraf();
    }
    else
    {
        Clear_Click(sender, e);
        FirstClicl = true;
    }
}
```

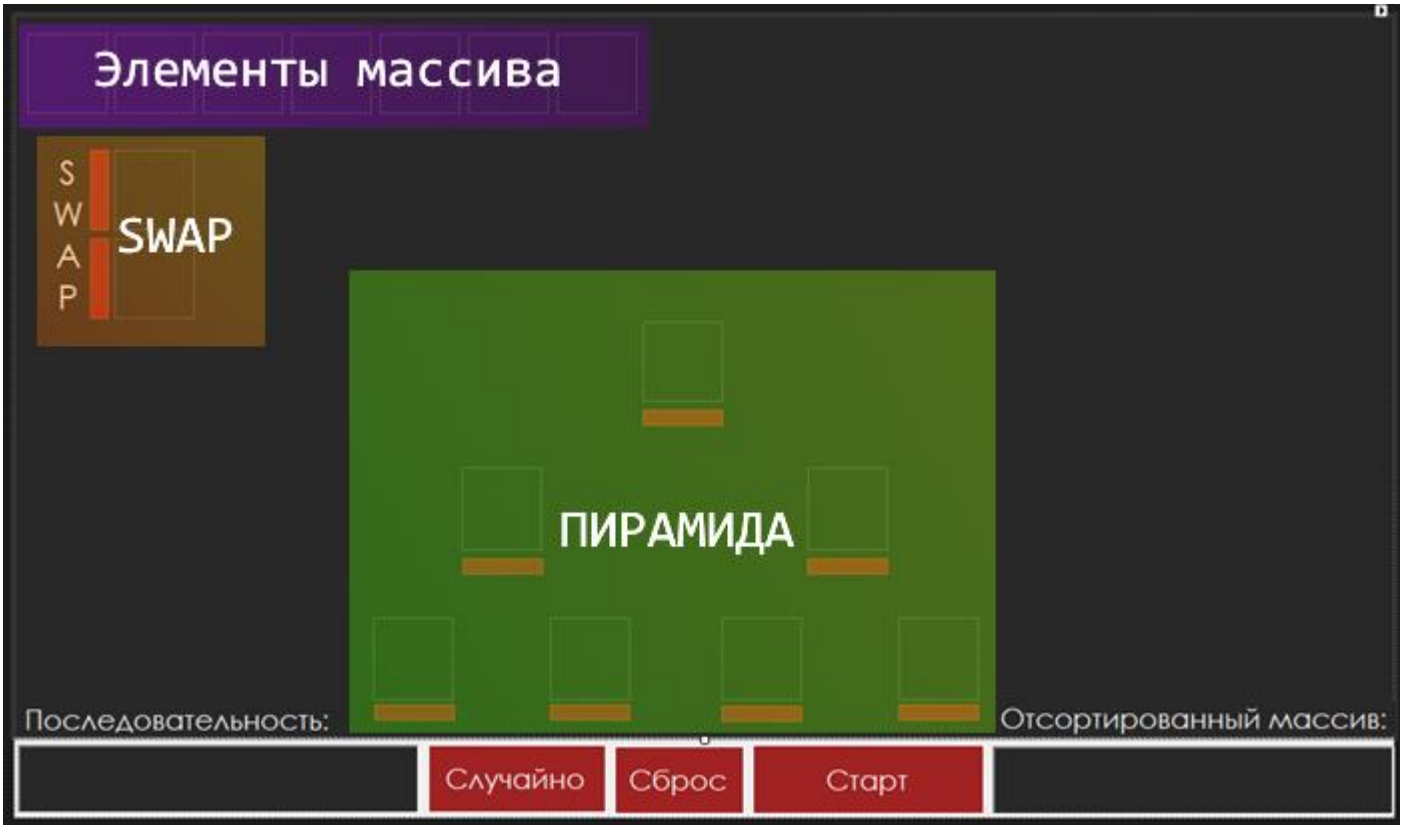
Кнопка «Очистить» удаляет все текста из всех RichTextBox.

Кнопка «Сортировать» взывает метод сортировки кучей и вставляет в TextBox, по мимо отсортированного массива еще парочку параметров.

```
private void Sort_Click(object sender, EventArgs e)
{
    NewMasTextBox.Clear();
    Stopwatch time = new Stopwatch();
    long[] array = HeapSort.Sort.arrayReadText(MainMasTextBox.Text);
    time.Start();
    HeapSort.Sort.heapSort(ref array);
    time.Stop();
    string temp = HeapSort.Sort.outputArray(ref array);
    algoTime = time.Elapsed.Ticks.ToString();
    int n = HeapSort.Count.size;
    NewMasTextBox.Text += "==============================\n";
    NewMasTextBox.Text += "Теоретическое время работы: " + Math.Ceiling(n * Math.Log(n, 2)) + "\n";
    NewMasTextBox.Text += "Практическое время работы: " + algoTime + "\n";
    NewMasTextBox.Text += "Кол-во элементов: " + n + "\n";
    NewMasTextBox.Text += "==============================\n";
    NewMasTextBox.Text += temp;
    NewMasTextBox.Text += '\n';
    TimerSec.SelectionAlignment = HorizontalAlignment.Center;
    TimerSec.Text = algoTime;
}
```

ПИ AnimePanel

На ПИ находятся в левом верхнем углу 7 панелей которые при запусках анимации образуют с кнопки наследованного класса MoveButton от Button. Которые в свою очередь играют роль элементов массива.



Снизу под «элементами массива» находяться две панели символизирующие функцию swap (Обмен значений перемных).

Еще ниже 7 панель реализующие пирамиду (кучу). Крастные панели обзначают вершину графа.

Также есть два RichTextBox для ввода данных и вывода. В левый RichTextBox вводится 7 элементов, чтобы заполнить кнопки MoveButton значениями. Это выполняеться после нажатия кнопки «Старт». Кпонка «Старт» изменеи параметр Text на «Шаг» и будет его сохранять до конца анимации. В конце параметр Text будет заменен на «Конец».

```
point1.Text = temp[0]; point2.Text = temp[1];
point3.Text = temp[2]; point4.Text = temp[3];
point5.Text = temp[4]; point6.Text = temp[5];
point7.Text = temp[6];
for (int i = 0; i < temp.Length; i++)
{
    array[i].n = int.Parse(temp[i]);
    if (i == 0)
        array[i].cord = point1;
    if (i == 1)
        array[i].cord = point2;
    if (i == 2)
        array[i].cord = point3;
    if (i == 3)
        array[i].cord = point4;
    if (i == 4)
        array[i].cord = point5;
    if (i == 5)
        array[i].cord = point6;
    if (i == 6)
        array[i].cord = point7;
}
```

Для определния шага анимации была использована глобальная переменная n и switch case. Алгоритм анимации был разделен на 10 частей

1. Заполнения элементов
2. Построение пирамиды
3. Востановление свойств пирамиды

4-9. Обмен корня и самого правго его потомка и дальнейшее востановление свойств пирамиды.

10. Востановление ввида массива.

```
public void swapMove(int x, int y)
{
    Thread.Sleep(300);
    if (Left > x){
        Thread.Sleep(200);
        while (Left != x)
            Left--;
    }
    else{
        Thread.Sleep(200);
        while (Left != x)
            Left++;
    }
    if(Top > y){
        Thread.Sleep(200);
        while (Top != y)
            Top--;
    }
    else{
        Thread.Sleep(200);
        while (Top != y)
            Top++;
    }
}
```

Передвижение элементов реализовано с помощью циклов while. x и y это координаты ранее созданных панелей. Так же создана структура Point для более удобной реализации анимации обмена и перемещения.

```
public struct Point
{
    public int n;
    public MoveButton cord;

    void Add(int a, MoveButton b)
    {
        n = a;
        cord = b;
    }
}
```

В фукции moveSwap передаються иднекс для обмена номером и координатами кнопки, а так же анимация фукции swap.

Остальные же фукции почти не отличаются от обычной сортирвки кучей. Однако фукция moveHeapSort чуть изменина. Она останваливается после того как меняет корень с листом и востанваливает свойства дерева, а так же уже осортированные части массива помечаются зеленым цветом.

Так же на нижней панели есть кнопка «Сброс». После того как анимация завершидлась на каком-нибудь из шагов при нажатии на эту кнопку ПИ вернятся в состояние ввода элементов.

Кнопка «Случайно» генериует 7 элементов от 0 до 9.

```
public void moveSwap(int i, int j)
{
    int temp1X = array[i].cord.Location.X;
    int temp1Y = array[i].cord.Location.Y;
    int temp2X = array[j].cord.Location.X;
    int temp2Y = array[j].cord.Location.Y;
    array[i].cord.swapMoveUP(PanelA.Location.X, PanelA.Location.Y);
    array[j].cord.swapMoveUP(PanelB.Location.X, PanelB.Location.Y);
    array[i].cord.swapMove(temp2X, temp2Y);
    array[j].cord.swapMove(temp1X, temp1Y);
    Point temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

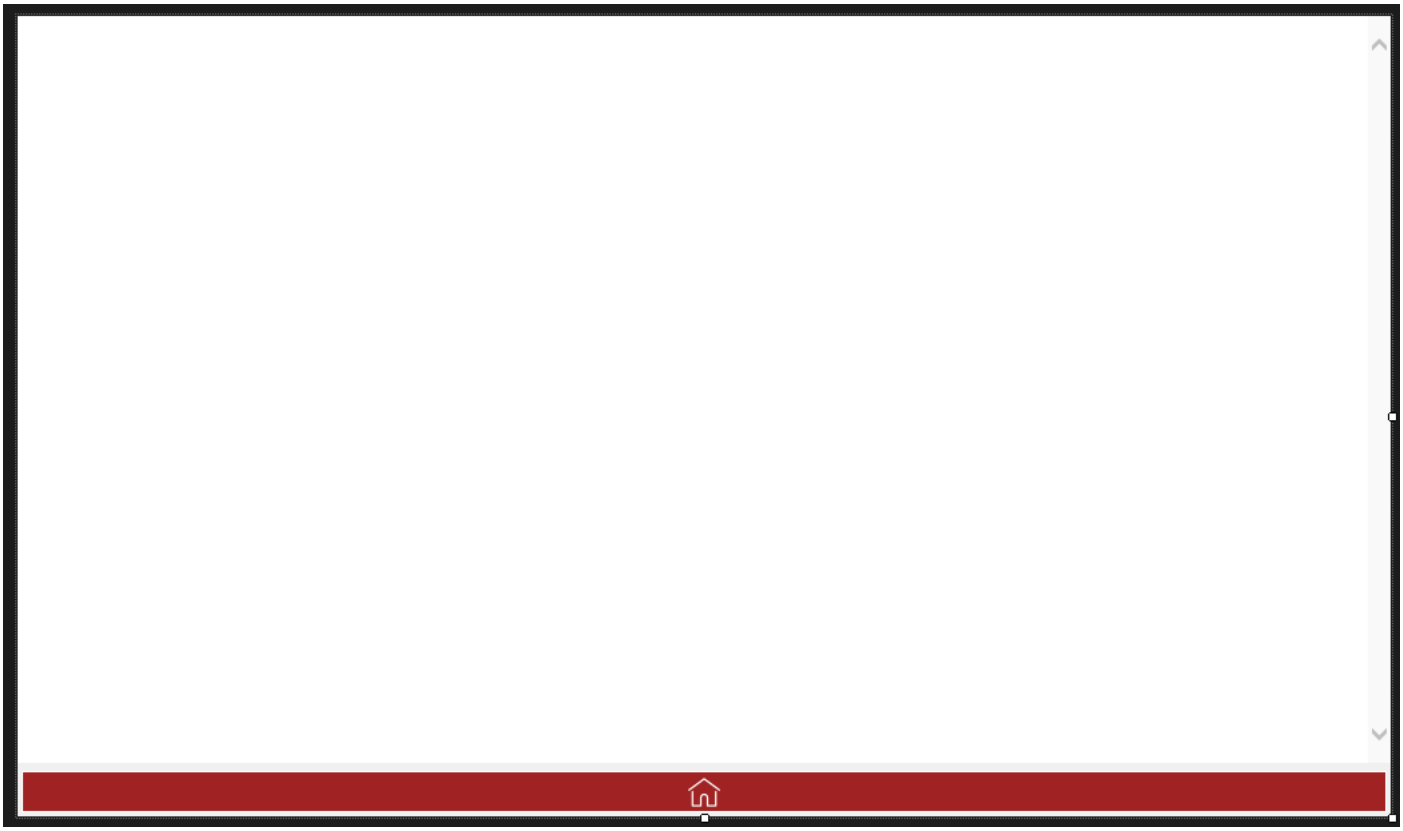
```
public void moveHeapSort()
{
    for (int i = n - 1; i >= 0; i--)
    {
        moveSwap(0, i);
        step++;
        moveSiftDown(i, 0);
        array[i].cord.BackColor = Color.Green;
        n--;
        return;
    }
}
```

ПИ WebPanel

На этом ПИ расположен элемент WebBrowser, который просто запускает сайт с описанием алгоритма. Также кнопка возраста если нужно вернуться к странице с алгоритмом.

```
public WebPanel()
{
    InitializeComponent();
    webBrowser1.Navigate("https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_кучей");
}

private void ReturnButton_Click(object sender, EventArgs e)
{
    webBrowser1.Navigate("https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_кучей");
}
```



Класс Sort

Класс содержащий стандартный алгоритм сортировки кучей. В нем заложены методы:

-swap

-shiftDown

-bildHeap

А также некоторые методы вывода отсортированной последовательности для предыдущие ПИ.

Так же реализован публичный класс Count в котором хранить кол-во элементов последовательности.

```
/// <summary> Возвращение отсортированного массива в строк
public static string outputMainArray(ref long[] arr)
{
    string line = null;
    for (int i = 0; i < arr.Length; i++)
    {
        if (i % 10 == 0 && i != 0)
            line += '\n';
        line += arr[i] + " ";
    }
    return line;
}

/// <summary> Возвращение отсортированного массива в файл
public static string outputArray(ref long[] arr)
{
    string line = null;
    for (int i = 0; i < arr.Length; i++)
    {
        if (i % 8 == 0 && i != 0)
            line += '\n';
        line += arr[i] + " ";
    }
    return line;
}
```

```
public class Count
{
    public static int size;
}
```

```
/// <summary> Функция обмена двух целых значений.
public static void swap(ref long a, ref long b)
{
    long temp = a;
    a = b;
    b = temp;
}

/// <summary> Функция постройки бинарной кучи
public static void bildHeap(ref long[] arr, int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        siftDown(arr, n, i);
}

/// <summary> Пирамидальная сортировка
public static void heapSort(ref long[] arr)
{
    int n = arr.Length;
    bildHeap(ref arr, n);

    for (int i = n - 1; i >= 0; i--)
    {
        swap(ref arr[0], ref arr[i]);
        siftDown(arr, i, 0);
    }
}
```

```
/// <summary> Функция подрежания свойст пиармиды
public static void siftDown(long[] arr, long n, int i)
{
    int size = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[size])
        size = left;
    if (right < n && arr[right] > arr[size])
        size = right;

    if (size != i)
    {
        swap(ref arr[i], ref arr[size]);
        siftDown(arr, n, size);
    }
}
```