РАЗРАБОТКА И ТЕСТИРОВАНИЕ КОМПЬЮТЕРНОЙ ПРОГРАММЫ ДЛЯ СОРТИРОВКИ МЕТОДОМ ПРОСТЫХ ВСТАВОК НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ НАЗВАНИЕ ЯЗЫКА C#

СОДЕРЖАНИЕ

Содержание
1. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ
1.1 Алгоритм решения
1.2 Макет приложения
1.3 Описание программы
ЗАКЛЮЧЕНИЕ
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ
ПРИЛОЖЕНИЕ-ЛИСТИНГ ПРОГРАММЫ

ВВЕДЕНИЕ

В современном мире мы часто сталкиваемся с большими объемами нужной нам информации. Ее так много, что можно запутаться, что же делать? Действительно, нередко среди огромного потока информации могут затеряться необходимые данные. Чтобы избежать этого, а также ускорить поиск нужной информации используют методы сортировок. Существует два вида сортировки данных: внешний и внутренний.

Внутренняя сортировка оперирует с массивами, целиком помещающимися в оперативной памяти с произвольным доступ к любой ячейке. Этот вид сортировки характерен тем, что данные сортируются на том же месте, без дополнительных затрат.

В свою очередь методы внутренней сортировки делятся на прямые и улучшенные:

Прямые методы:

- Вставкой (включением)
- Выбором (выделением)
- Обменом («пузырьковая»)

Улучшенные методы:

- Быстрая
- Шелла

Внешняя сортировка – сортировка данных, расположенных на периферийных устройствах и не вмещающихся в оперативную память, то есть когда применить одну из внутренних сортировок невозможно.

Стоит отметить, что внутренняя сортировка значительно эффективней внешней, так как на обращение к оперативной памяти затрачивается намного меньше времени, чем к магнитным дискам, лентам и т. п.

Рассматриваемый в данной курсовой работе вид сортировки «Выбор» имеет три варианта реализации: простая выборка, пирамидальная, плавная.

В соответствии с поставленной целью были сформулированы следующие задачи:

- Провести предметный анализ в области
- Разработать необходимую программу
- Провести тестирование приложения
- Определить эффективность разработанной программы

1. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

Название приложения: Программа сортировки методами выбора

Назначение разработки: Исходя из введенных пользователем данных отсортировать тремя методами сортировки: выборкой, пирамидально, плавной.

Входные данные вводятся пользователем в специальные поля. После обработки и выполнения сортировки программа выводит результат в соответствующее поле вывода.

Для корректной работы программы требуется заполнить все необходимые поля. Для реализации данной программы мы используем язык программирования C#, на платформе Visual Studio.

Системные требования к ПК:

1) Операционная система Windows 7 или выше.
2) Свободное место на жестко диске: 10Мб и более.
3) Наличие Net.Framework 4.0 или выше.
4) Оперативная память 512Мб и выше.
5) Клавиатура и мышь.

1.1 АЛГОРИТМ РЕШЕНИЯ

В самом начале выполнения программы появляется форма, где пользователю предлагается заполнить соответствующие поля необходимыми для расчета данными.

Затем, в ходе выполнения программы производится проверка полноты и корректности введенных начальных данных. Если исходные данные не прошли проверку – выводится соответствующее уведомление.

После успешно пройденной проверки, программа начинает сортировать полученные данные. Программа считывает данные, заполненные в специальных полях и производит сортировку по установленным алгоритмам.

После этого результаты выводятся в специально отведенное поле, а выполнение программы прекращается.

1.2 МАКЕТ ПРИЛОЖЕНИЯ

Макет приложения «Методы сортировок выбором»(Form1)

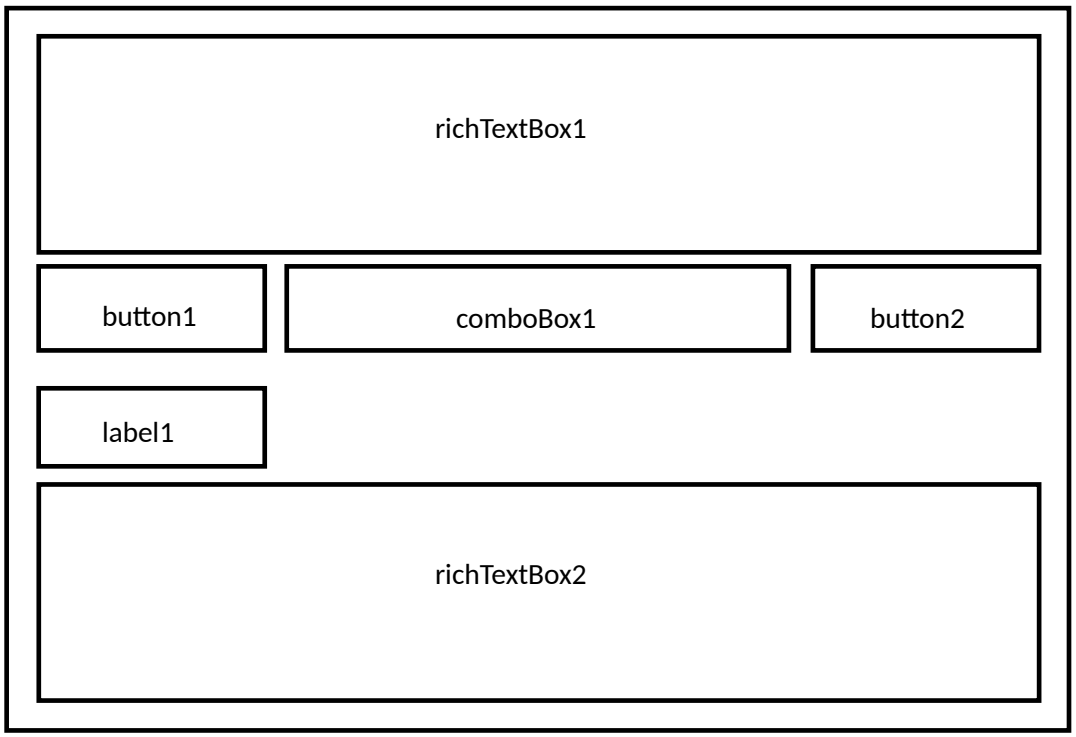


Рисунок 1. Макет приложения.

richTextBox1 – получает введенный пользователем массив, либо автоматически генерируемый с помощью кнопки «Сгенерировать»

button1 – принимает текстовое значение «Сгенерировать», а также генерирует в поле richTextBox1 рандомный массив от 0 до 100.

comboBox1 – содержит список, предоставляющий пользователю выбрать методы сортировки данных массива. Содержит текстовые значения: «Пирамидальная сортировка», «Плавная сортировка», «сортировка Выбором».

button2 – принимает текстовое значение «Сортировать», а также сортирует введенные или сгенерированные пользователем данные из richTextBox1.

richTextBox2 – служит для вывода результатов сортировки тем или иным выбранным методом.

1.3 Описание программы

Иерархия классов

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

using System.Windows.Forms;

Используемые элементы

button

richTextBox

comboBox

label

Обработчики событий

private void button1_Click(object sender, EventArgs e)

private void button2_Click(object sender, EventArgs e)

Функции

В данной курсовой используется 3 основных и 4 вспомогательных функции.

Функция Print отвечает за вывод результата, предварительно отсортированного выбранным методом и передает результат в поле вывода.

```
private void Print(int[] array) // результат
{
richTextBox2.Clear(); // очищение поля
foreach (var x in array) // перебор в переменную x всех значений массива array, в который записывается результат сортировки выбранным методом
richTextBox2.Text += x + “” // добавление этих значений
}
```

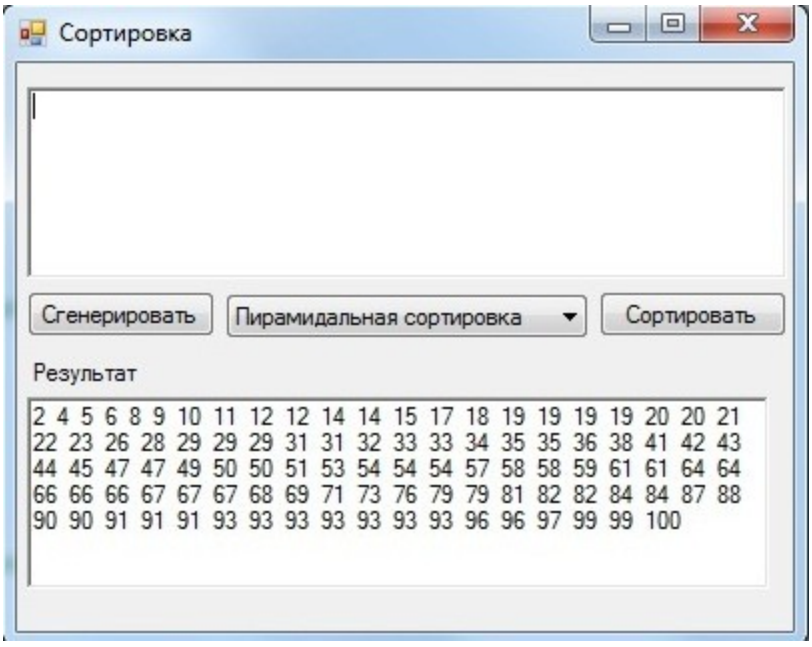


Рисунок.2 Результат работы функции Print.

Функция selectSort производит сортировку по алгоритму «Выборка» и передает отсортированный массив в функцию Print

```
private void selectSort(int[] array) // сортировка выборкой
{
```

```
int min; // объявляем min
for (int i = 0; i < array.Length; i++) // перебираем элементы
{
min = i; // присваиваем наименьший элемент i-ому
for (int j = i + 1; j < array.Length; j++)
if (array[j] < array[min]) // если элменент массива меньше минимального
min = j; // присваиваем новое минимальное значение элементу массиву
// swap-функция. Меняем местами значения
int temp = array[i]
array[i] = array[min];
array[min] = temp;
}
}
```

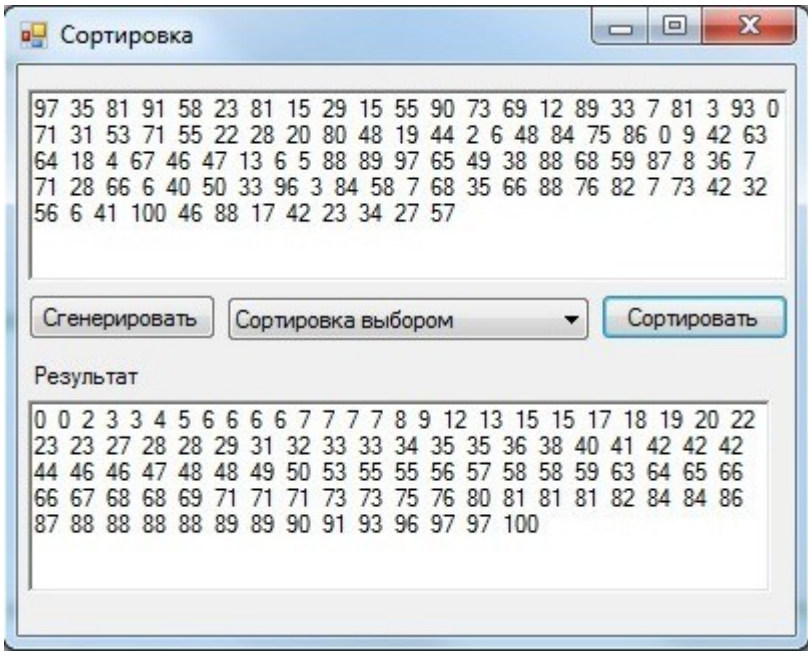


Рисунок.3 Результат работы функции selectSort

Функция heapSort производит сортировку по алгоритму «Пирамида» и передает отсортированный массива в функцию Print

```
private void heapSort(int[] array) // сортировка пирамидой
{
int i, temp; // объявляем переменные
for (i = array.Length / 2 - 1; i >= 0; i--) // элементы с номерами начиная с array.Length / 2 - 1 это листья, то тогда нужно переупорядочить поддеревья с корнями в индексах
downHeap(array, i, array.Length - 1); // вызывается функция downHeap и ей передаются аргументы
for (i = array.Length - 1; i > 0; i--)
```

```
{
temp = array[i];
array[i] = array[0];
array[0] = temp;
downHeap(array, 0, i - 1); // в функцию передаются аргументы с шагом –
1
}
}
```

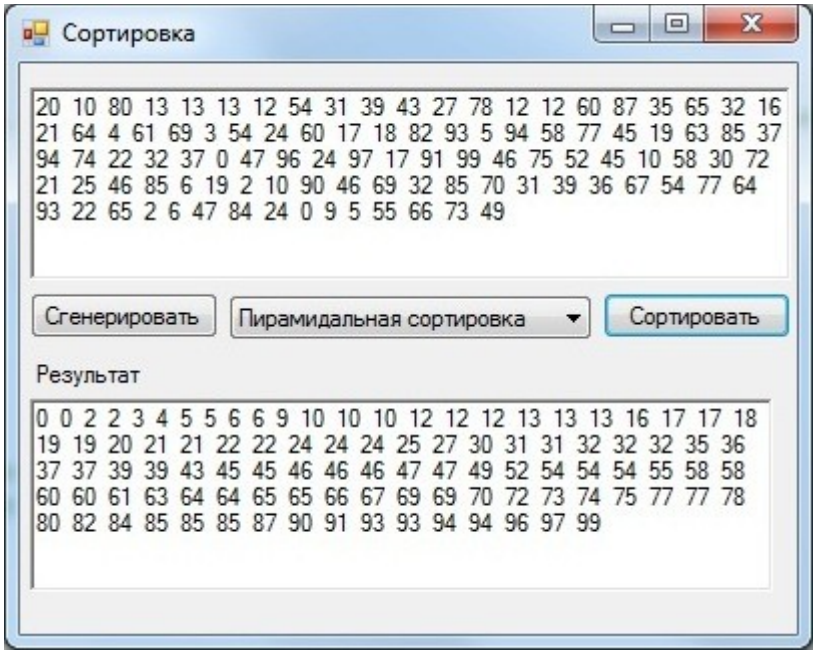


Рисунок.4 Результат работы функции heapSort

Функция smoothSort производит сортировку по алгоритму «Плавная» и передает отсортированный массив в функцию Print.

```
private void smoothSort(int[] mas) // функция плавная сортировки
{
make_heap_pool(mas); // вызов функции, создающей
последовательность куч из произвольного массива
for (int i = mas.Length - 1; i >= 0; i--)
{
int nextPosHeapItemsAmount = 0;
int posMaxTopElem = findPosMaxElem(mas, curState, i, ref
nextPosHeapItemsAmount); // максимальный верхний элемент кучи получает
значение в виде результата функции findPosMaxElem
if (posMaxTopElem != i)
{
swap(ref mas[i], ref mas[posMaxTopElem]); // вызываем функцию swap и
передаем в нее значения mas[i] и mas[posMaxTopElem]
```

shiftDown(mas, nextPosHeapItemsAmount, posMaxTopElem); // вызываем функцию

}

PrevState(ref curState); // вызываем функцию PreveState и передаем значения, ссылаясь(ref) на значение переменной curState

}

}

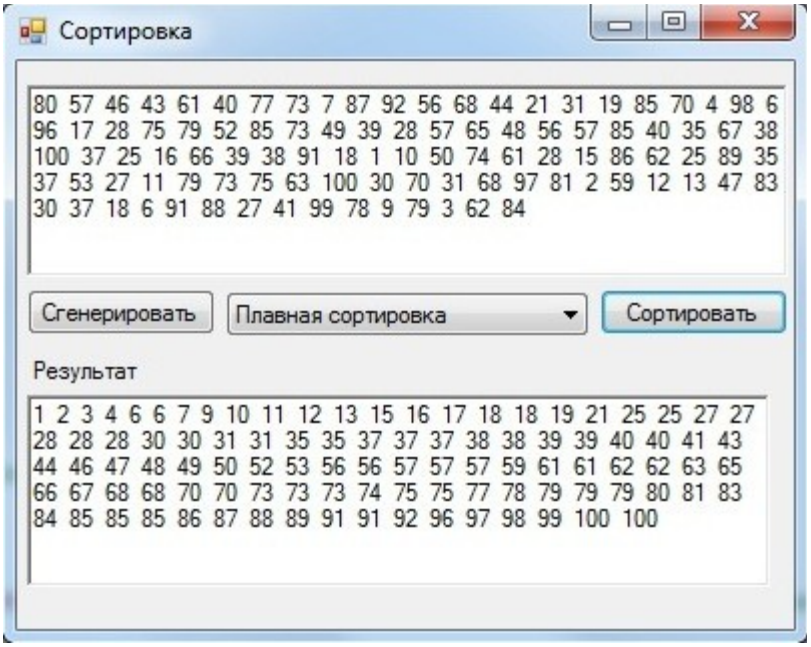


Рисунок.5 Результат работы функции smoothSort

Функция downHeap является вспомогательной функцией для heapSort, обеспечивая нижнюю сортировку кучи массива

private void downHeap(int[] array, int k, int n) // функция нижней сортировки

{

// объявляем переменные

int new_elem;

int child;

new_elem = array[k];

while (k <= n / 2) // пока у array[k] есть дети выполняем

{

child = 2 * k;

// выбираем большего сына

if (child < n && array[child] < array[child + 1])

child++;

if (new_elem >= array[child]) break;

// иначе

array[k] = array[child]; // переносим сына наверх

```
k = child;
}
array[k] = new_elem;
}
```

Функция NextState является вспомогательной функцией. Делает обратную операцию, но при этом она еще возвращает номер бита, соответствующий вершине последней кучи, если та состоит более чем из одного элемента. В противном случае функция возвращает –1.

```
private int NextState(ref int curState) // Функция NextState, делает обратную операцию, но при этом она еще возвращает номер бита, соответствующий вершине последней кучи, если та состоит более чем из одного элемента. В противном случае функция возвращает –1.
{
int posNewTop = -1; // позиция вершины объединенных куч.
// исключение
if ((curState & 7) == 5)
{ // curState = 0101
curState += 3; // curState = 1000
posNewTop = 3;
}
else // пытаемся найти два подряд единичных бита
{
int next = curState;
int pos = 0;
while (next != 0 && (next & 3) != 3)
{
next >>= 1;
pos++
}
if ((next & 3) == 3) // curState = 01100
{
curState += 1 << pos; // curState = 10000
posNewTop = pos + 2;
}
else if ((curState & 1) != 0) // curState = x001
curState |= 2; // curState = x011
else // curState = xx00
curState |= 1; // curState = xx01
}
return posNewTop;
}
```

Функция PrevState является вспомогательной функцией. Она изменяет текущее состояние с учетом того, что вершина последней кучи исключена из рассмотрения.

private void PrevState(ref int curState) //Функция PrevState изменяет текущее состояние с учетом того, что вершина последней кучи исключена из рассмотрения.

```
{
if ((curState & 15) == 8)
{ // curState = 1000
curState -= 3; // curState = 010
}
else if ((curState & 1) != 0)
{
if ((curState & 3) == 3) // curState = x011
curState ^= 2; // curState = x001
else // curState = xx01
curState ^= 1; // curState = xx00
}
else // ищим первый единичный бит
{
int prev = curState;
int pos = 0;
while (prev != 0 && !(Convert.ToBoolean(prev & 1)))
{
prev >>= 1;
pos++;
}
if (prev != 0) // curState = xx10000
{
curState ^= 1 << pos;
curState |= 1 << (pos - 1);
curState |= 1 << (pos - 2); // curState = xx01100
}
else
curState = 0;
}
}
```

Функция shiftDown является вспомогательной функцией. Она реализует просеивание данных массива «вниз»

private void shiftDown(int[] mas, int posHeapItemsAmount, int indexLastTop) // Реализация функции просейки вниз, в ней задается также mas[]

```
{
int posCurNode = indexLastTop; //indexLastTop - индекс крайней вершин
```

while (posHeapItemsAmount > 1) //nextPosHeapItemsAmount - количество элементво в кучи, вершина которой оказалось максимальной из всех вершин куч

```
{
// позиция правого сына
int posR = posCurNode - 1;
// позиция левого сына
int posL = posR - LeoNum[posHeapItemsAmount - 2]; // число элементов в текущей кучи
int posMaxChild = posL;
int posNextTop = posHeapItemsAmount - 1;
if (mas[posR] > mas[posL]) // если позиция правого сына больше левого
{
posMaxChild = posR; // то "старший сын" правый
posNextTop = posHeapItemsAmount - 2;
}
if (mas[posCurNode] < mas[posMaxChild])
{
swap(ref mas[posCurNode], ref mas[posMaxChild]);
posHeapItemsAmount = posNextTop;
posCurNode = posMaxChild;
}
else
break;
}
}
```

Функция make_heap_pool является вспомогательной функцией. Она создает последовательности куч из произвольного массива.

```
private void make_heap_pool(int[] mas) // Функция создания последовательности куч из произвольного массива.
{
for (int i = 0; i < mas.Length; i++)
{
int posHeapItemsAmount = NextState(ref curState);
if (posHeapItemsAmount != -1)
shiftDown(mas, posHeapItemsAmount, i);
}
}
```

Функция findPosMaxElem является вспомогательной функцией. Она ищет максимальный элемент среди вершин куч.

```
private int findPosMaxElem(int[] mas, int curState, int indexLastTop, ref int nextPosHeapItemsAmount) // Функция поиска максимального элемента среди вершин куч
{
int pos = 0;
// ищим позицию первого единичного бита
while (!Convert.ToBoolean(curState & 1))
```

```
{
curState >>= 1;
pos++;
}
int posMaxTopElem = indexLastTop;
nextPosHeapItemsAmount = pos;
int curTopElem = indexLastTop - LeoNum[pos];
curState >>= 1;
pos++;
while (curState != 0)
{
if ((curState & 1) != 0)
{
if (mas[curTopElem] > mas[posMaxTopElem])
{
posMaxTopElem = curTopElem;
nextPosHeapItemsAmount = pos;
}
curTopElem -= LeoNum[pos];
}
curState >>= 1;
pos++;
}
return posMaxTopElem;
}
```

Функция swap является вспомогательной функцией. Она выполняет переприсваивание значений.

```
private void swap(ref int a, ref int b) // функция переприсваивания (swap)
{
int temp = b;
b = a;
a = temp;
}
```

1.4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

1. Для корректной работы программы необходимо заполнить вручную или сгенерировать с помощью кнопки массив данных.

2. Чтобы автоматически заполнить массив данных нажмите кнопку «Сгенерировать»

3. Необходимо в раскрывающемся списке выбрать подходящий вам метод сортировки.

4. Для получения результата сортировки нажмите кнопку «Сортировать»

ЗАКЛЮЧЕНИЕ

Язык программирования C# на основе Visual Studio способен реализовать все необходимые средства для сортировки данных.

Во время выполнения поставленной задачи были улучшены навыки программирования, работы с методами сортировок. Разработанная программа наглядно демонстрирует реализацию 3 методов сортировки. Основное преимущество программы – выполнение сортировки динамически задаваемых данных.

Был проведен анализ предметной области, выявлены требования к разрабатываемой программе, было спроектировано и реализовано приложение, определена эффективность разработки.

Программа корректна.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Плавная сортировка // Информационный ресурс Википедиа [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Плавная_сортировка - Загл с экрана Яз. рус., англ.

2. Пирамидальная сортировка // Информационный ресурс Википедиа [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Пирамидальная_сортировка - Загл с экрана Яз. рус., англ.

3. Сортировка выбором // Информационный ресурс Википедиа [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Сортировка_выбором - Загл с экрана Яз. рус., англ.

4. Герберт Шилдт, C# 4.0 Полное руководство, учебное пособие [Текст] // Герберт Шилдт. – Московский дом книги, 2008. – 340с.

5. Дональд Кнут, Искусство программирования, том 3. Сортировка и поиск [Текст] // Дональд Кнут. – Вильямс, 2007. – 457с.