Лабораторная работа № 2

Тема: «Алгоритмы сортировки»

Цель работы: Изучение алгоритмов сортировки в языке C#.

Сортировка вставками (O(n2))

Работает, проходя по массиву и перемещая нужное значение в начало массива. После того, как обработана очередная позиция, мы знаем, что все позиции до нее отсортированы, а после нее — нет. Сортировка вставками обрабатывает элементы массива по порядку.

Принцип данной сортировки заключается в том, что берется число перемещаясь вправо, сверяя значения. Числа больше нужного - передвигаются вправо, числа меньше - остаются в левой части и проверяя условия больше либо меньше конкретного числа.

Сортировка вставками наиболее эффективна когда массив уже частично отсортирован и когда элементов массива не много. Если же элементов меньше 10 то данный алгоритм является лучшим.

Сортировка выбором (O(n2))

Как и сортировка пузырьком, этот алгоритм проходит по массиву раз за разом, перемещая одно значение на правильную позицию. Однако, в отличие от пузырьковой сортировки, он выбирает наименьшее неотсортированное значение вместо наибольшего. Как и при сортировке вставками, упорядоченная часть массива расположена в начале, в то время как в пузырьковой сортировке она находится в конце. При прямом выборе для поиска одного элемента с наименьшим ключом просматриваются все элементы входной последовательности и найденный элемент помещается как очередной элемент в конец готовой последовательности

Принцип данной сортировки заключается в том, что из всей последовательности, выбирается элемент с минимальным значением и сравнивается с другими, перемещаясь в начальную неотсортированную позицию. На каждом этапе сортировки увеличивается готовая последовательность и возрастает исходная.

Сортировка прямым выбором предпочтительнее алгоритму прямого включения, однако, если ключи в начале упорядочены или почти упорядочены, прямое включение будет оставаться несколько более быстрым.

Сортировка пузырьком (O(n2))

Это самый простой алгоритм сортировки. Он проходит по массиву несколько раз, на каждом этапе перемещая самое большое значение из неотсортированных в конец массива.

При первом проходе по массиву мы сравниваем необходимые значения. Поскольку нужное число больше текущего, мы оставляем их как есть. После чего сравниваем большее число и меньшее. Меньше число меньше, поэтому мы меняем их местами, перемещая число на одну позицию ближе к концу массива.

Если предположить, что в массиве содержится N элементов и хотя бы один из них занимает свое место в результате однократного пересмотра значений, то алгоритм может совершить не более N прохождений. (Все N понадобятся, когда массив изначально отсортирован в обратном порядке.) Каждое такое прохождение включает N шагов, отсюда общее время работы алгоритма — O(N2).

Пузырьковая сортировка часто рассматривается как наиболее неэффективный сортировочный метод, поскольку она должна переставлять элементы до того, как станет известна их окончательная позиция.

Сортировка Шелла (O(N(LogN)2))

Идея сортировки методом Шелла состоит в том, чтобы сортировать элементы отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1 (а именно обычной сортировкой вставками).

Сортировка подсчётом O(n)

Алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов. Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством, например, миллион натуральных чисел меньших 1000. Эффективность алгоритма падает, если при попадании нескольких различных элементов в одну ячейку, их надо дополнительно сортировать. Необходимость сортировки внутри ячеек лишает алгоритм смысла[уточнить], так как каждый элемент придётся просматривать более одного раза.

Шейкерная сортировка O(n)

В ней пределы той части массива в которой есть перестановки, сужаются. Плюс – внутренние циклы проходят по массиву то в одну, то в другую сторону, поднимая самый легкий элемент вверх и опуская самый тяжелый элемент в самый низ за одну итерацию внешнего цикла.

Бинарные вставки

Делаются проходы по части списка, и в его начале "вырастает" отсортированная последовательность. Можно считать, что эта последовательность упорядочена. По ходу алгоритма в нее будут вставляться все новые элементы. Поиск подходящего места для очередного элемента входной последовательности осуществляется путем последовательных сравнений с элементом, стоящим перед ним. В зависимости от результата сравнения элемент либо остается на текущем месте(вставка завершена), либо они меняются местами и процесс повторяется.

Алгоритм сортировки бинарными включениями представляет из себя оптимизированную версию алгоритма сортировки простыми вставками, отличие заключается в том, что при поиске место, на которое надо вставить элемент ai в уже упорядоченную совокупность a0 , ..., ai-1 , определяется алгоритмом деления пополам (отсюда и название алгоритма "бинарные включения" здесь понимаем как "включения делением пополам").

Исходный код

```
using System;
using static System.Console;
using System.Diagnostics;

namespace Сортировка_Вставками
{
    class Program
    {
        static void Main(string[] args)
        {
            Write("Количество элементов массива: ");
            int m = int.Parse(ReadLine());
            int[] mass = new int[m];

            Random Rand = new Random();
            Write("Массив:");
            Stopwatch stpWatch = new Stopwatch();
            stpWatch.Start();

            for (int i = 0; i < m; i++)
            {
                mass[i] = Rand.Next(-90, 100);
                Write(mass[i] + " ");
            }

            int d, left, right, mid;
            for (int i = 1; i < m; i++)
            {
                if (mass[i - 1] > mass[i])
                {
                    d = mass[i]; // d – включаемый элемент
                    left = 0; // левая граница отсортированной части массива
                    right = i - 1; // правая граница отсортированной части массива
                    do
                    {
                        mid = (left + right) / 2; // mid – середина новой последовательности
                        if (mass[mid] < d) left = mid + 1;
                        else right = mid - 1;
                    }
                    while (left <= right); // поиск ведется до тех пор, пока левая граница не
окажется правее правой границы
                    for (int j = i - 1; j >= left; j--) mass[j + 1] = mass[j];
                    mass[left] = d;
                }
            }
            Write("\nМассив сортировки Бинарными вставками:");
            for (int i = 0; i < mass.Length; i++)
            {
                Write(mass[i] + "  ");
            }

            stpWatch.Stop();
            Write("\nВремя сортировки массива: " + stpWatch.ElapsedMilliseconds.ToString());
            ReadKey();
        }
    }
}
```

Результат выполнения алгоритма показан на рисунке 1.

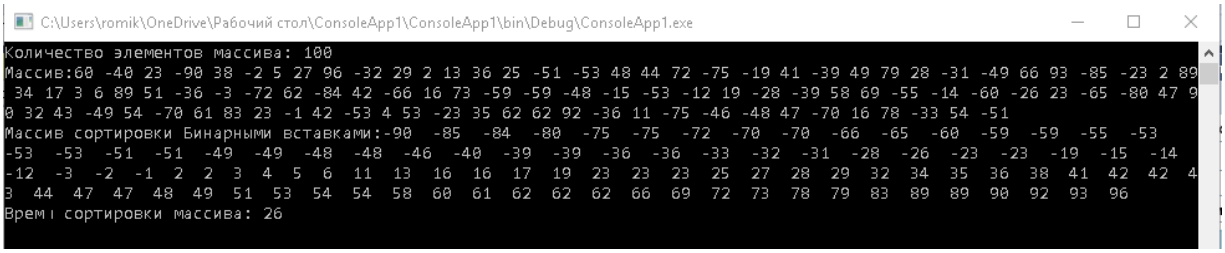

Рисунок 1-Результат сортировки Бинарные вставки

Исходный код алгоритма Вставки

```
using System;
using static System.Console;
using System.Diagnostics;

namespace Сортировка_Вставками
{
    class Program
    {
        static void Main(string[] args)
        {
            Write("Количество элементов массива: ");
            int m = int.Parse(ReadLine());
            int[] mass = new int[m];

            Random Rand = new Random();
            Write("Массив:");
            Stopwatch stpWatch = new Stopwatch();
            stpWatch.Start();

            for (int i = 0; i < m; i++)
            {
                mass[i] = Rand.Next(-90, 100);
                Write(mass[i] + " ");
            }

            for(int i = 0; i < mass.Length; i++)
            {

                int temp = mass[i];
                int j = i;

                while(j>0 && temp < mass[j - 1])
                {
                    mass[j] = mass[j - 1];
                    j--;

                }
                mass[j] = temp;


            }

            Write("\nМассив сортировки Вставкой:");
            for (int i = 0; i < mass.Length; i++)
            {
                Write(mass[i] + " ");
            }
```

```
            stpWatch.Stop();
            Write("\nВремя сортировки массива: " + stpWatch.ElapsedMilliseconds.ToString());
            ReadKey();
        }
    }
    }
```

Результат выполнения алгоритма показан на рисунке 2.

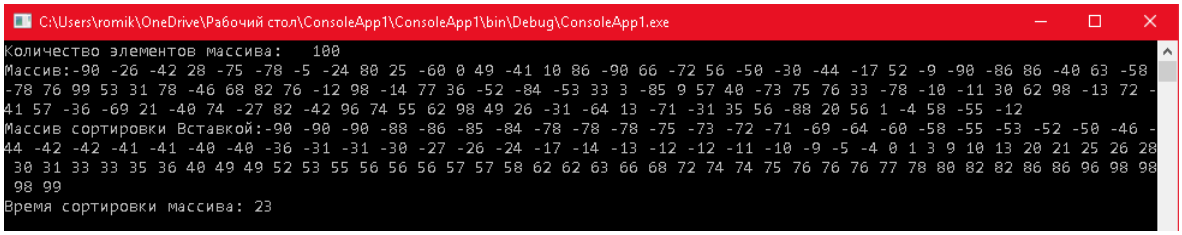

Рисунок 2-Результат сортировки алгоритма Вставки

Исходный код алгоритма Выбором

```
using System;
using static System.Console;
using System.Diagnostics;


namespace Лаб.Работа__5
{
    class Program
    {
        static void Main(string[] args)
        {
            Write("Количество элементов массива: ");
            int m = int.Parse(ReadLine());
            int[] mass = new int[m];

            Random Rand = new Random();
            Write("Массив:");
            Stopwatch stpWatch = new Stopwatch();
            stpWatch.Start();

            for (int i = 0; i < m; i++)
            {
                mass[i] = Rand.Next(-90, 100);
                Write(mass[i] + " ");
            }

            for (int i = 0; i < mass.Length-1; i++)
            {

                int imin = i;
                for (int j = i; j < mass.Length; j++)
                {
                    if (mass[j]<mass[imin])
                    {
                        imin = j;
                    }
                }
                int temp = mass[imin];
                mass[imin] = mass[i];
                mass[i] = temp;
            }
```

```
            Write("\nМассив сортировкой Выбором:");
            for (int i = 0; i < mass.Length; i++)
            {
                Write(mass[i] + " ");
            }

            stpWatch.Stop();
            Write("\nВремя сортировки массива: " + stpWatch.ElapsedMilliseconds.ToString());

            ReadKey();
        }
    }
}
```

Результат сортировки Выбором, показан на рисунке 3.

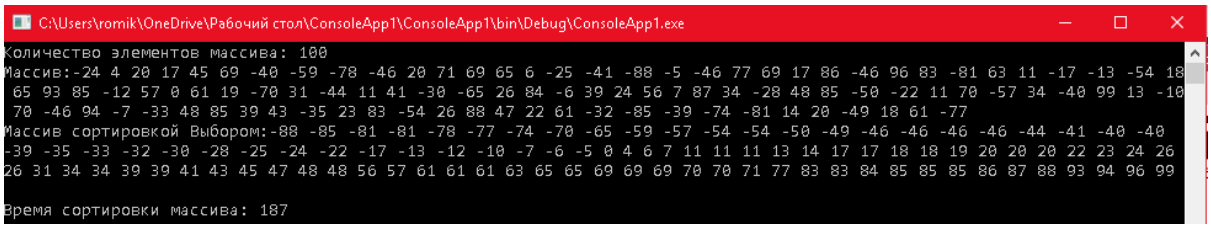

Рисунок 3-Результат выполнения сортировки Выбором

Исходный код алгоритма Подсчетом

```
using System;
using static System.Console;
using System.Diagnostics;


namespace Лаб.Работа__5
{
    class Program
    {
        static void Main(string[] args)
        {
            Write("Количество элементов массива: ");
            int m = int.Parse(ReadLine());
            int[] mass = new int[m];

            Random Rand = new Random();
            Write("Массив:");
            Stopwatch stpWatch = new Stopwatch();
            stpWatch.Start();

            for (int i = 0; i < m; i++)
            {
                mass[i] = Rand.Next(-90, 100);
                Write(mass[i] + " ");
            }

            int[] D = new int[m]; // массив счётчиков
            int[] Sort = new int[m]; // отсортированный массив
            for (int i = 0; i < m; i++)
            {
                D[i] = 0;
```

```
            }
            for (int i = 0; i < m; i++) // сравнение элементов и заполнение массива счётчиков
            {
                for (int j = i + 1; j < m; j++)
                {
                    if (mass[i] > mass[j])
                    {
                        ++D[i];
                    }
                    else
                    {
                        ++D[j];
                    }
                }
            }
            for (int i = 0; i < m; i++)
            {
                Sort[D[i]] = mass[i];// перессылка элементов в новый массив
            }
            for (int i = 0; i < m; i++)
            {
                mass[i] = Sort[i];
            }

            Write("\nМассив сортировкой Подсчета:");
            for (int i = 0; i < mass.Length; i++)
            {
                Write(mass[i] + " ");
            }

            stpWatch.Stop();
            Write("\nВремя сортировки массива: " + stpWatch.ElapsedMilliseconds.ToString());

            ReadKey();
        }
    }
}
```

Результат сортировки Подсчетом показан на рисунке 4.

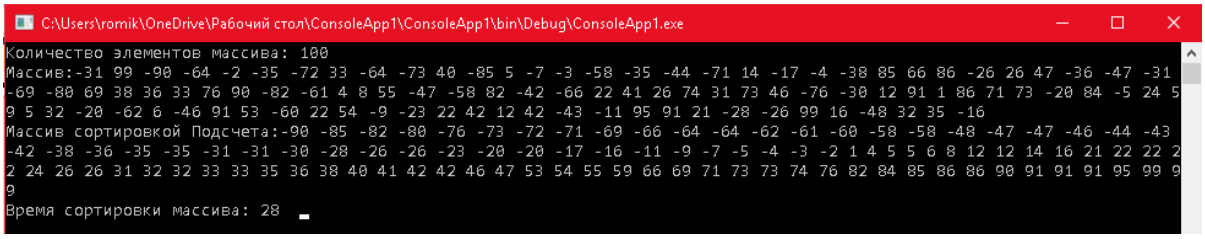

Рисунок 4-Результат выполнения сортировки Подсчетом

Блок-схема алгоритма сортировки Пузырек, показана на рисунке 10.