САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ МЕХАНИКИ И ОПТИКИ

**Лабораторная работа**

по комбинаторным алгоритмам на тему:

*«Сортировка»*

Выполнил:

студент группы 1125

Припадчев Артём

Проверила:

Павловская Т.А.

2013

**Цель работы:** реализация алгоритмов сортировки и исследование их характеристик (быстродействие, требуемый объем памяти, естественность поведения, устойчивость, область применимости).

Для исследования были выбраны три алгоритма сортировки: быстрая сортировка, сортировка методом Шелла, встроенная в C# (точнее в .NET) сортировка.

**Слайд 2. Сортировка Шелла**

Суть метода в том, что в отличие от обычной сортировки вставками, здесь происходит упорядочивание элементов стоящих друг от друга на шаге d, т.е. упорядочиваются элементы с индексами отличающимися на d. Затем элементов стоящих друг от друга на шаге h (с индексами отличающимися на h причем h<d) и т.д.

Сортировка Шелла была названа в честь её изобретателя — Дональда Шелла, который опубликовал этот алгоритм в 1959 году.

Время работы алгоритма в худшем случае примерно оценивается формулой:

t = 0.1\*N1.5

**Слайд 3. Реализация метода Шелла.**

**Слайд 4. Быстрая сортировка**

Быстрая сортировка — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром в 1960 году – является одним из наиболее быстрых известных универсальных алгоритмов (в среднем O(n\*log n) обменов при упорядочении n элементов).

Метод основан на подходе "разделяй-и-властвуй". Общая схема алгоритма такова:

1. из массива выбирается некоторый опорный элемент;
2. запускается процедура разделения массива, которая перемещает все ключи, меньшие, либо равные опорному элементу, влево от него, а все ключи, большие, либо равные – вправо;
3. теперь массив состоит из двух подмножеств, причем левое меньше, либо равно правого;
4. для обоих подмассивов: если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру.

В конце получится полностью отсортированная последовательность.

**Слайд 5. Реализация быстрой сортировки.**

Про встроенную сортировку Array.Sort будет рассказано позднее, т.к. это целая магия Microsoft.

**Слайд 6.**

На этом слайде приведено сравнение работы этих трех алгоритмов для массивов с различным количеством элементов и перемешанностью в 50%. По горизонтали количество элементов, а по вертикали время в тиках процессора.

Судя по этому графику, места распределились следующим образом: самая быстрая – Array.Sort, затем быстрая сортировка, и на последнем месте сортировка методом Шелла.

**Слайд 7.**

Сейчас мы видим тот же график, но в бОльшем масштабе, т.к. на предыдущем трудно было увидеть, как распределяется время на меньших по количеству элементов массивах. Таким образом, на небольших массивах сортировка Шелла опережает «быструю. Данные результаты схожи и для других процентов перемешанности массивов.

**Слайд 8.**

Исключение составляют лишь массивы, данные в которых не перемешаны совсем. На них быстрая сортировка начинает проигрывать «Шеллу». А встроенная в .NET сортировка по-прежнему лидирует (еще бы).

При различной перемешанности файлов время сортировки для одного и того же количества элементов каждого алгоритма примерно одинаково, исключение составляет опять же сортировка Шелла, которая при работе с уже упорядоченным массивом работает быстрее.

**Слайд 9.**

Перейдем собственно к магии, а именно к методу Array.Sort. У нее очень хорошие результаты, поэтому после некоторых также магических манипуляций ее код был найден.

Примерный код Array.Sort.

**private** **static** **void** IntroSort(T[] keys, **int** lo, **int** hi, **int** depthLimit, IComparer<T> comparer)

{

**while** (hi > lo)

    {

**int** num = (hi - lo) + 1;

**if** (num <= 0x10)

        {

**switch** (num)

            {

**case** 1:

**return**;

**case** 2:

                    ArraySortHelper<T>.SwapIfGreater(keys, comparer, lo, hi);

**return**;

**case** 3:

                    ArraySortHelper<T>.SwapIfGreater(keys, comparer, lo, hi - 1);

                    ArraySortHelper<T>.SwapIfGreater(keys, comparer, lo, hi);

                    ArraySortHelper<T>.SwapIfGreater(keys, comparer, hi - 1, hi);

**return**;

            }

            ArraySortHelper<T>.InsertionSort(keys, lo, hi, comparer);

**return**;

        }

**if** (depthLimit == 0)

        {

            ArraySortHelper<T>.Heapsort(keys, lo, hi, comparer);

**return**;

        }

        depthLimit--;

**int** num2 = ArraySortHelper<T>.PickPivotAndPartition(keys, lo, hi, comparer);

        ArraySortHelper<T>.IntroSort(keys, num2 + 1, hi, depthLimit, comparer);

        hi = num2 - 1;

    }

}

В ней довольно интересная реализация, используется комбинация из простого свопа, сортировки вставкой и сортировки кучей, в зависимости от количества элементов подмножества.

Методы **SwapIfGreater()** просто меняют переданные им элементы. Т.е. если в коллекции один элемент, то никаких изменений производиться не будет, если 2, то при необходимости их поменяют местами, и если 3 их также отсортируют тремя вызовами метода SwapIfGreater().

**InsertionSort.** Это обычная сортировка вставками, в ходе которой на каждом шаге алгоритма мы выбирается один из элементов входных данных и вставляется на нужную позицию в уже отсортированном списке, до тех пор, пока набор входных данных не будет исчерпан.

**Heapsort.** Пирамидальная сортировка (или же сортировка кучей). Суть ее работы в том, что при первом проходе составляется дерево объектов, по которому затем и восстанавливается отсортированная последовательность.

**Introsort** или интроспективная сортировка — алгоритм сортировки, предложенный Дэвидом Мюссером в 1997 году. Он использует быструю сортировку и переключается на пирамидальную сортировку, когда глубина рекурсии превысит некоторый заранее установленный уровень (например, логарифм от числа сортируемых элементов). Этот подход сочетает в себе достоинства обоих методов с худшим случаем O(n log n) и быстродействием, сравнимым с быстрой сортировкой. Так как оба алгоритма используют сравнения, этот алгоритм также принадлежит классу сортировок на основе сравнений.

**Слайд 10. Вывод:**

Таким образом в ходе работы были получены следующие результаты:

* Самой эффективной оказалась встроенная в .NET сортировка, т.к. она включает в себя анализ переданных ей входных данных и вызывает различные методы сортировки.
* Для коллекций, в которых количество элементов меньше 512 лучше использовать сортировку методом Шелла, нежели QuickSort (если не использовать встроенную).
* Для коллекций, в которых более 512 элементов продуктивнее использовать QuickSort (если не использовать встроенную).