Министерство образования Российской Федерации
Московский Государственный Технический Университет
им. Н.Э. Баумана

Отчёт по лабораторной работе № 1
По курсу: "Анализ алгоритмов"
Тема:"Алгоритмы сортировок"

Студент: Орехова Е.О. ИУ7-51

Преподаватель: Волкова Л.Л.

4 марта 2018 г.

Содержание

1 Постановка задачи

В ходе выполнения лабораторной работы необходимо вывести трудоёмкость трёх алгоритмов сортировок, сравнить полученные данные с экспериментальными. Сделать выводы.

2 Сортировка пузырьком

2.1 Идея

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива

2.2 Реализация

Листинг 1: Сортировка пузырьком.

```
        static private void boble(ref int [] arr, int
           len)
        {
                int change;
                for (int j = 0; j<len-1; j++)
                {
                        for (int i = 0; i < len-j-2;i++)
                        {
                                if (arr[i]>arr[i+1])
                                {
                                        change = arr[i];
                                        arr[i] = arr[i +
                                           1];
                                        arr[i + 1] =
                                           change;
                                }
                        }
                }
```

```
}
```

2.3 Трудоемкость

Лучший случай(N-длина массива):

$$f_{best} = 2 + (N-1)(2 + 4 + \frac{N}{2}(2 + 4 + 0)) = 3N^2 + 3N - 4$$

Худший случай:

$$f_{worst} = 2 + (N-1)(2 + 4 + \frac{N}{2}(2 + 4 + 9)) = 7.5N^2 - 1.5N - 4$$

3 Сортировка вставками

3.1 Идея

Алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

3.2 Реализация

Листинг 2: Сортировка вставками

```
private static void sort_insert(ref int[] arr,
   int len)
{
        int key;
        int i;
        for (int j = 1; j < len; j++)
        {
                key = arr[j];
                i = j - 1;
                while ((i >= 0)&&(arr[i]> key))
                {
                        arr[i + 1] = arr[i];
                        i--;
                }
                arr[i + 1] = key;
        }
}
```

3.3 Трудоемкость

Самым благоприятным случаем является отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации. Тогда сложность алгоритма составит $f(n) = O(n)$. Время работы линейно от размера входных данных.
Наихудшим случаем является массив, отсортированный в порядке, обратном нужному. При этом каждый новый элемент сравнивается со всеми в отсортированной последовательности. Тогда сложность алгоритма составит:$f(n) = O(n^2)$
Для анализа среднего случая нужно посчитать среднее число сравнений, необходимых для определения положения очередного элемента. При добавлении нового элемента потребуется, как минимум, одно сравнение, даже если этот элемент оказался в правильной позиции. i-й добавляемый элемент может занимать одно из i+1 положений. Предполагая случайные входные данные, новый элемент равновероятно может оказаться в любой позиции. Сложность алгоритма в этом случае: $O(n^2)$

4 Qucksort

4.1 Идея

Общая идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие»

3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

4.2 Реализация

Листинг 3: Quicksort

```
static private void quicksort(ref int[] arr, int
   begin, int end)
{
        int p;
        if (begin < end)
        {
                p = partition(ref arr,begin,end);
                quicksort(ref arr, begin, p - 1);
                quicksort(ref arr, p+1, end);
        }
}

static private int partition(ref int[] arr, int
   begin, int end)
{
        int change;
        int pivot = arr[end];
        int i = begin;

        for (int j = begin; j< end;j++)
        {
                if (arr[j] <= pivot)
                {
                        change = arr[i];
                        arr[i] = arr[j];
                        arr[j] = change;
                        i++;
                }

        }
        change = arr[i];
        arr[i] = arr[end];
        arr[end] = change;
        return i;
}
```

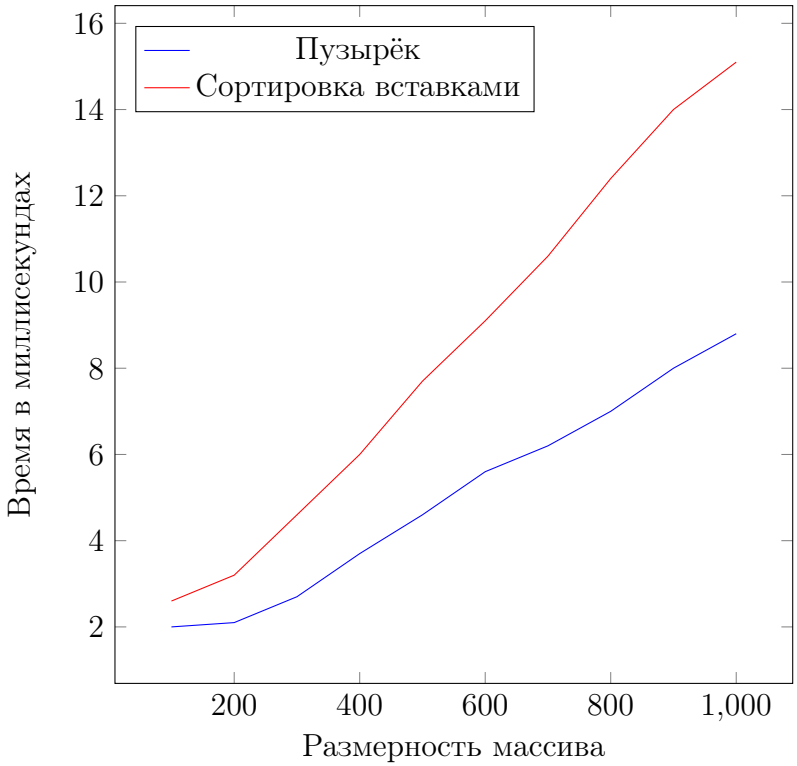
4.3 Трудоемкость

Операция разделения массива относительно опорного элемента имеет сложность $O(n)$, что будет верно для каждого уровня рекурсии Глубина рекурсии в лучшем и среднем случаях будет равняться $O(\log_2(n))$, в худшем случае - N Следовательно общая сложность алгоритма будет

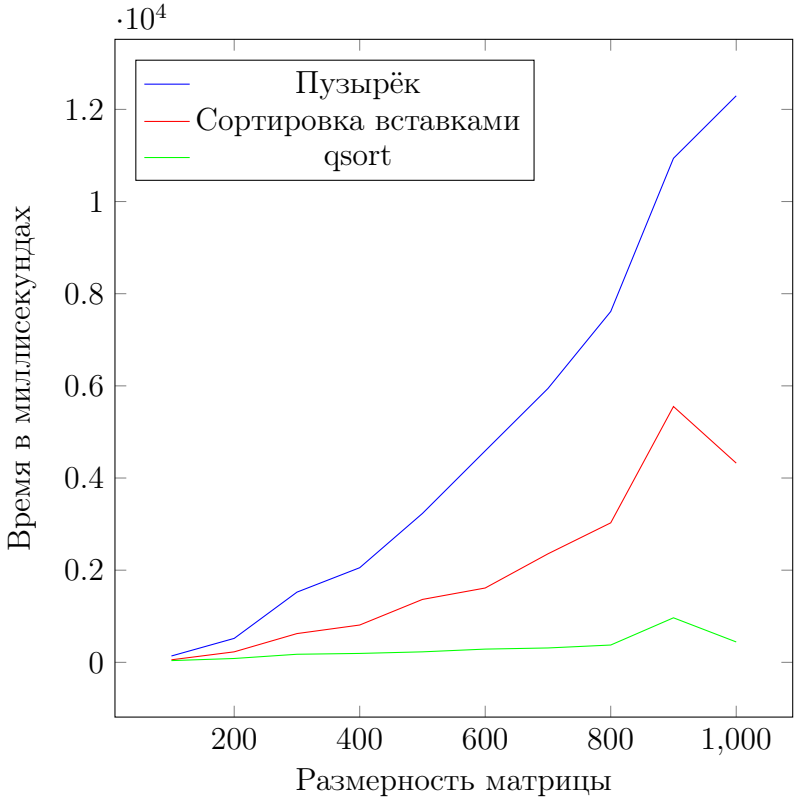
$O(n*\log_2(n))$ и $O(n^2)$ в лучшем/среднем и худшем случаях соответственно

5 Сравнение

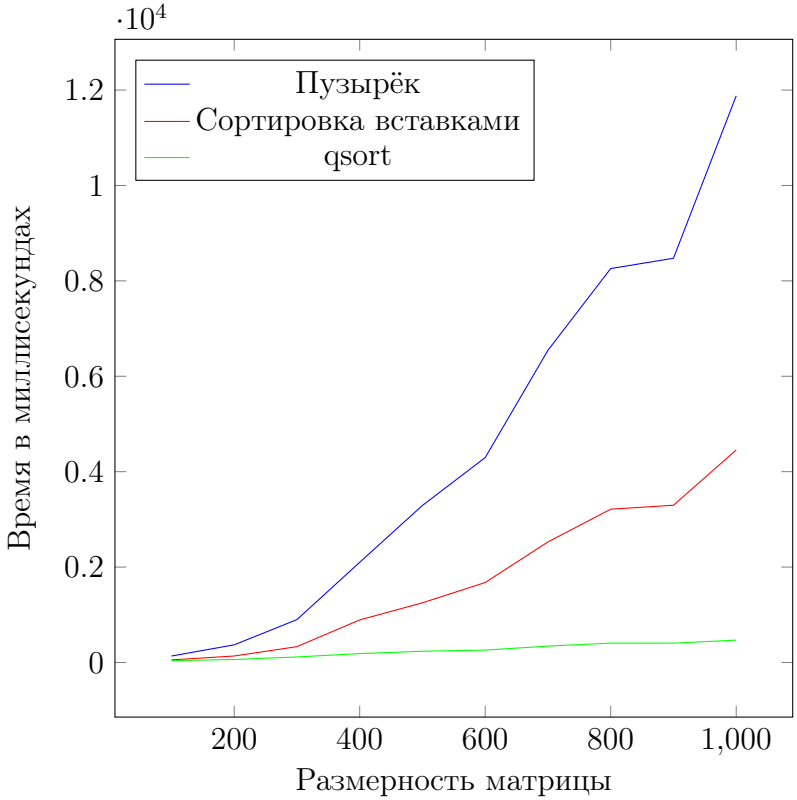
5.1 Лучший случай



5.2 Случайные значения



5.3 Худший случай



6 Заключение

В ходе выполнения лабораторной работы был изучены различные алгоритмы сортировок. Проведены эксперименты, подтверждающие теоретическую сложность сортировок.