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

Отчет по лабораторной работе №3
По курсу «Анализ алгоритмов»

Тема: «Алгоритмы сортировки»

Студент: **Медведев А.В.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

Постановка задачи

В ходе лабораторной работы предстоит

1. изучить работу алгоритмов сортировки;
2. Оценить трудоемкость одного из алгоритмов.
3. Сравнить реализованные алгоритмы.

Описание модели вычислений

Опишем модель вычислений, которой будем пользоваться в дальнейшем при оценке трудоемкости алгоритмов.

Операции, имеющие трудоемкость “1”:

1. арифметические операции
2. логические операции
3. операции сравнения
4. операции присваивания
5. операция взятия индекса
6. операции побитового И() и ИЛИ(|)
7. унарный плюс и минус

Операции, имеющие трудоемкость “0”:

1. логический переход по ветвлению
2. операции обращения к полю структуры/класса
3. объявление переменных

Алгоритм сортировки вставками

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность $O(N^2)$

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма. Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части.

Листинг 1: Алгоритм сортировки вставками

```
public static void InsertionSort<T>(T[] array) where T :
    IComparable<T>
{
    for (int i = 1; i < array.Length; i++)//2+13(n-1)+9W
    {
        var tmp = array[i];//2
        var location = i - 1;//2
        while (location>=0 && (array[location].CompareTo(
            tmp)>0)) //4+W(4+5)
        {
            array[location + 1] = array[location];//4
            location--;//1
        }
        array[location + 1] = tmp;//3
    }
}
```

Трудоёмкость алгоритма

f = 2 + 13(n-1) + 9W = 9W + 13n - 11, где W - число заходов в цикл while

Лучший случай - отсортированный массив - W = 0:

fл = 13n - 11

Худший случай - массив, отсортированный в обратную сторону. $W(N) = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2-N}{2} \approx \mathbf{\frac{N^2}{2}}$

$f = 9 * (n-1) * n/2 + 13n - 11 = 4.5n^2 + 8.5n - 11$

Таким образом, в лучшем случае трудоёмкость алгоритма сортировки вставками имеет линейную сложность, а в худшем случае - квадратичную.

Алгоритм сортировки пузырьком

Сортировка простыми обменами, сортировка пузырьком — простой алгоритм сортировки. Для понимания и реализации тот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: O(N^2)

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 («всплывает» до нужной позиции, как пузырёк в воде - отсюда и название алгоритма).

Листинг 2: Алгоритм сортировки пузырьком

```
public static void BubleSort<T>(T[] array) where T :
    IComparable<T>
{
    for (int i = array.Length; i >=0; i--)
    {
        bool flag = true;
        for (int j = 1; j < i; j++)
        {
            if (array[j-1].CompareTo(array[j])>=0)
            {
                T tmp = array[j - 1];
                array[j - 1] = array[j];
                array[j] = tmp;
                flag = false;
            }
        }
        if(flag)
            break;
    }
}
```

Трудоёмкость алгоритма

Трудоемкость тела внутреннего цикла не зависит от количества элементов в массиве, поэтому оценивается как O(1). В результате выполнения внутреннего цикла, наибольший элемент смещается в конец массива неупорядоченной части, поэтому через N таких вызовов массив в любом случае окажется отсортирован. Если же массив отсортирован, то внутренний цикл будет выполнен лишь один раз. Тогда в лучшем случае алгоритм будет иметь трудоемкость O(n)

В среднем и худшем случае $T=O(\sum_{i=N-1}^{0}\sum_{j=1}^{i}1)=O(N^2)$

Таким образом,как и в сортировке вставками в лучшем случае трудоёмкость алгоритма сортировки пузырьком имеет линейную сложность, а в худшем случае - квадратичную.

Быстрая сортировка

Быстрая сортировка, сортировка Хоара — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром. Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(nlogn) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

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.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Листинг 3: Быстрая сортировка

```
1  private static void QuickSort<T>( T[] array, int begin,
       int end) where T : IComparable<T>
2  {
3      int i = begin, j = end;
4      T mid = array[(begin+end)/2];
5
6      if (begin >= end)
7      {
8          return;
9      }
10
11     while (i<=j)
12     {
13         while (array[i].CompareTo(mid)<0)
14         {
15             i++;
16         }
17
18         while (array[j].CompareTo(mid)>0)
19         {
20             j--;
21         }
22
23         if (i <= j)
24         {
25             T tmp = array[j];
26             array[j] = array[i];
27             array[i] = tmp;
28             i++;
29             j--;
30         }
31     }
32
33     QuickSort(  array, i, end);
34     QuickSort(  array, begin, j);
35
36 }
```

Трудоёмкость алгоритма

Лучшим случаем для быстрой сортировки является ситуация, когда с помощью выбранного опорного элемента массив будет делится ровно

пополам (+- 1 элемент). Тогда глубина рекурсии будет равна $log_2(n)$. При этом на каждом уровне рекурсии суммарная трудоемкость разделения массива на две части будет равна O(n). Тогда трудоемкость алгоритма будет равна $O(nlog_2n)$.

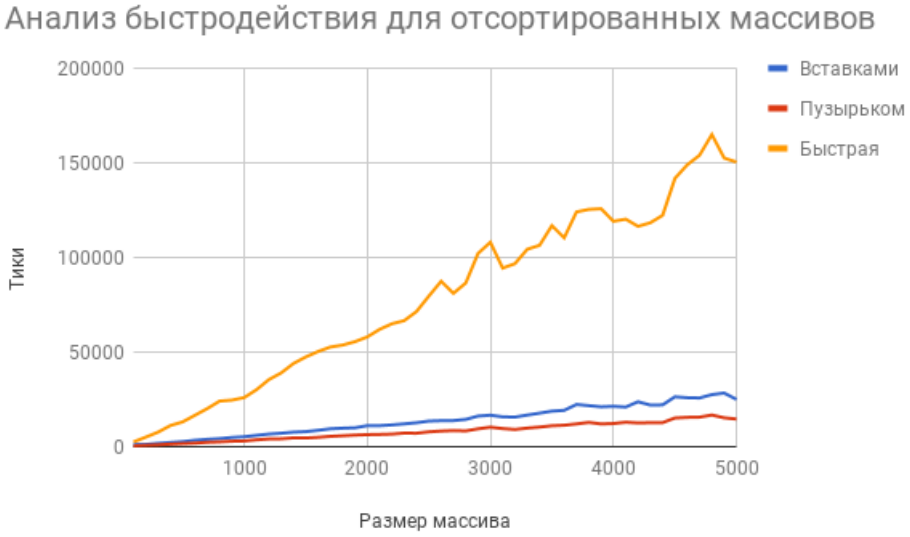
В среднем временная сложность алгоритма составляет O(nlogn). Худшим случаем будет та ситуация, когда выбираемый опорный элемент равен минимальному или максимальному значению на текущем интервале, так как это приведет к тому, что при рекурсивном вызове sort длина массива сокращается не в два раза, а всего лишь на один элемент. Трудоемкость алгоритма быстрой сортировки для этого случая будет равна $O(n^2)$. В связи с этим существуют различные оптимизации, в том числе и по выбору барьерного элемента, который можно выбирать случайным образом или, например, выбирать средний по значению элемент из первого, последнего и среднего элементов текущего участка массива.

Эксперимент

Для сравнения алгоритмов было посчитано среднее время работы для массивов различной длины: 100, 200, ..., 5000. При этом были проведены отдельные тесты для произвольных массивов, отсортированных и отсортированных в обратном порядке массивов.

*Время замерялось в тиках с помощью функции Stopwatch.ElapsedTicks.

Средние значения из 10 замеров представлены на графиках.

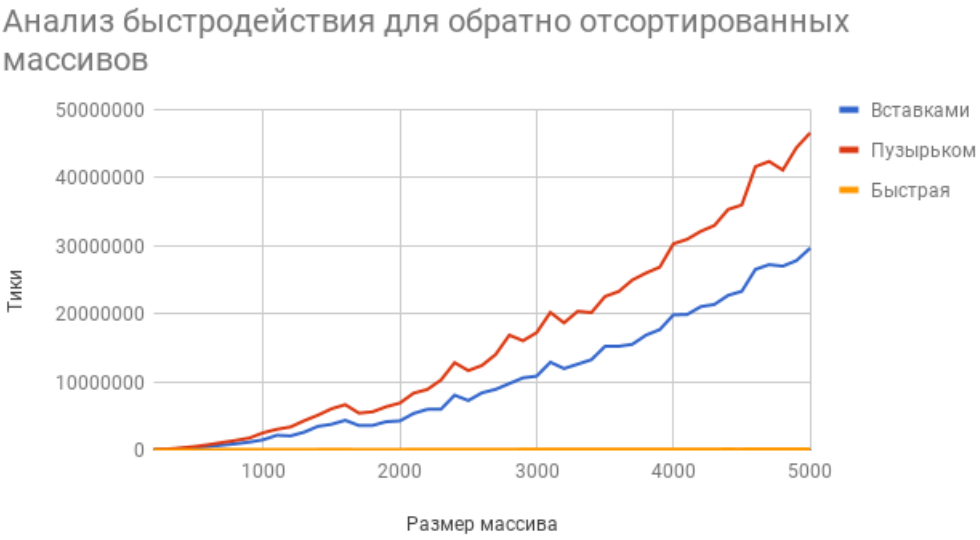


Выводы из экспериментов

Для произвольных массивов и массивов отсортированных в обратную сторону, зависимость времени работы алгоритма от длины массива квадратичная для сортировок вставками и пузырьком, и линейная для быстрой сортировки, что и подтверждает теоретическую оценку трудоемкости этих алгоритмов.
В лучшем случае - для отсортированного массива, алгоритмы сортировки вставками и пузырьком работают быстрее, чем алгоритм быстрой сортировки.
Когда массив частично отсортирован и имеет небольшое количество элементов, то наиболее эффективным алгоритмом сортировки является сортировка вставками. Низкая эффективность быстрой сортировки в таком случае объясняется наличием дополнительных затрат на обслуживание вызовов рекурсивной функции.

Заключение

Во время выполнения работы были изучены и реализованы алгоритмысортировки. Была произведена оценка трудоемкости сортировки вставками и сравнение быстродействия работы реализованных алгоритмов.


Анализ быстродействия для обратно отсортированных массивов
50000000
40000000
30000000
20000000
10000000
0
Тики
1000
2000
3000
4000
5000
Размер массива
Вставками
Пузырьком
Быстрая


Анализ быстродействия для случайных массивов
100000000
75000000
50000000
25000000
0
Тики
1000
2000
3000
4000
5000
Размер массива
Вставками
Пузырьком
Быстрая