Отчет по лабораторной работе 3
По предмету “Анализ алгоритмов”
По теме “Сортировка массивов”

Фирсова Дарья ИУ7-56

2018

Введение

В лабораторной работе изучаются алгоритмы сортировки массивов. Для каждого алгоритма необходимо рассчитать сложность в зависимости от количества элементов.
Цель лабораторной работы: анализ, реализация и сравнительный анализ времени работы алгоритмов сортировки массивов.
Задачи для лабораторной работы:

1. Ввести модель оценки трудоемкости
2. Реализовать выбранные алгоритмы
3. Провести временные замеры
4. Произвести расчет трудоемкости для реализованных алгоритмов.
5. Сравнительный анализ времени работы алгоритма для массивов размером от 100 элементов до 10000.

1 Аналитеская часть

В данном разделе приведены алгоритмы и составлена модель для вычисления трудоемкости.

1.1 Описание алгоритмов

Алгоритм сортировки пузырьком
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. 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. Был выбран как наиболее медленный для сравнения времени работы. Из-за своей неэффективности не используется на практике.
Шейкер сортировка

Является улучшением алгоритма сортировки пузырьком. В сортировке пузырьком элементы быстро "всплывают но медленно "тонут поэтому можно менять сторону просмотра массива на каждой итерации, смотреть сначала с начала, потом с конца.

Сортировка вставками

Элементы входной последовательности просматриваются по одному, сравниваются попарно и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

1.2 Модель вычислений

Введем следующую модель вычислений: операции $+ - * / < > == != += == []$ имеют стоимость 1.

1.2.1 Оценка трудоемкости цикла for

Инициализация до цикла стоит 2, после выполнения тела цикла, инкрементируется итератор цикла, проверяется условие.
$F = 2 + N * (F_{body} + 2)$

1.2.2 Оценка трудоемкости оператора if

Переход по условию имеет стоимость 0, проверка условия зависит от выражения самой провеки согласно модели выше.
Для оператора без проверки условия: $F = 0$
Для оператора с проверкой условия: $F = 0 + body$

2 Конструкторская часть

В данном разделе представлены псевдокоды алгоритмов
Сортировка пузырьком:
for i = 0, i< len(a) :
for j = len(a)-1, j< i:
если a[j] < a[j-1]:
то a[j], a[j-1] = a[j-1], a[j];
конец если
j–;
конец for j;
конец for i;

Сортировка вставками:
for n = 1, n <len(a):
i = n - 1
пока (i > -1) и a[i+1] < a[i]:
a[i+1], a[i] = a[i], a[i+1]
i -= 1
конец пока
конец for

Шейкер сортировка:

```
left = 0
right = len(a) - 1
пока left <= right:
for i = left, i<right:
если a[i] > a[i + 1]:
то a[i], a[i + 1] = a[i + 1], a[i]
конец если
right -= 1
i = i + 1
конец for

for i= right, i<left:
если a[i - 1] > a[i]:
то a[i], a[i - 1] = a[i - 1], a[i]
конец если
left += 1
i = i - 1
конец for
```

3 Технологическая часть

В этом разделе приведена реализация функций, указан язык программирования и необходимые модули.

3.1 Средства реализации

В данной работе использовался язык Python 3.6, в среде Pycharm. Для измерения времени использовался модуль time, измерения проищводились в секундах.

3.2 Листинг кода

```
def bubble_sort(a):
    n = len(a)
    for i in range(n): # 2+N(2 +n-1(13 + 3))
        for j in range(n - 1, i, -1):
            if a[j] < a[j-1]:#4
                swap = a[j] #2      body = 13
                a[j] = a[j-1] #4
                a[j-1] = swap #3

    return #worse 18N^2 - 12, best: 2+N(2 + (N-1)(4+3) = 7N^2-3
def insertion_sort(a):
    for n in range(1,len(a)):
        i = n - 1
        while (i > -1) and a[i+1] < a[i]:
            a[i+1], a[i] = a[i], a[i+1]
            i -= 1
    return(a)
def shaker(a):
    left = 0
    right = len(a) - 1

    while left <= right:
        for i in range(left, right, +1):
            if a[i] > a[i + 1]:
                a[i], a[i + 1] = a[i + 1], a[i]
        right -= 1

        for i in range(right, left, -1):
            if a[i - 1] > a[i]:
                a[i], a[i - 1] = a[i - 1], a[i]
```

```
31            left += 1
```

Листинг 1. Реализация алгоритмов.

3.3 Оценка трудоемкости алгоритмов

Сортировка пузырьком
Самая неэффективная сортировка. Оценка трудоемкости приведена в листинке для лучшего и худшего случая. Лучший - весь массив уже отсортирован, $18N^2 - 12$, худший случай - весь массив отсортирован в обратном порядке, $7N^2 - 3$.
Сортировка вставками
Сложность в худшем случае: $W = \frac{N^2-N}{2}$ или $O(N^2)$.
Сложность в среднем случае: $O(N^2)$.
Сложность в лучшем случае: $O(N)$.[1]
Сортировка Шейкер
Сложность в худшем случае: $O(N^2)$.
Сложность в среднем случае: $O(N^2)$.
Сложность в лучшем случае: $O(N)$.[2]

4 Экспериментальная часть

В данном разделе будут приведены примеры работы алгоритмов и произведены замеры времени. Тестирование производилось на компьютере с процессором Intel Core i5 (I5-6267U) и оперативной памятью 8 Гб.

4.1 Примеры работы

Пример результата работы сортировки алгоритма не будет приведен. Для каждого алгоритма производились замеры времени для размерности от 100 до 5000 с шагом 500. Измерения производились для массивов трех типов: уже отсортированный (от меньшего к большему), отсортированный обратно (от большего к меньшему), составленный из случайных чисел.

4.2 Сравнительный анализ

Эксперимент с отсортированным и обратно отсортированным массивом проводился два раза, результат - среднее арифметическое двух замеров времени. Проводить повторные эксперименты для случайных значений не имеет слысла.

График времени работы алгоритма сортировки пузырьком.

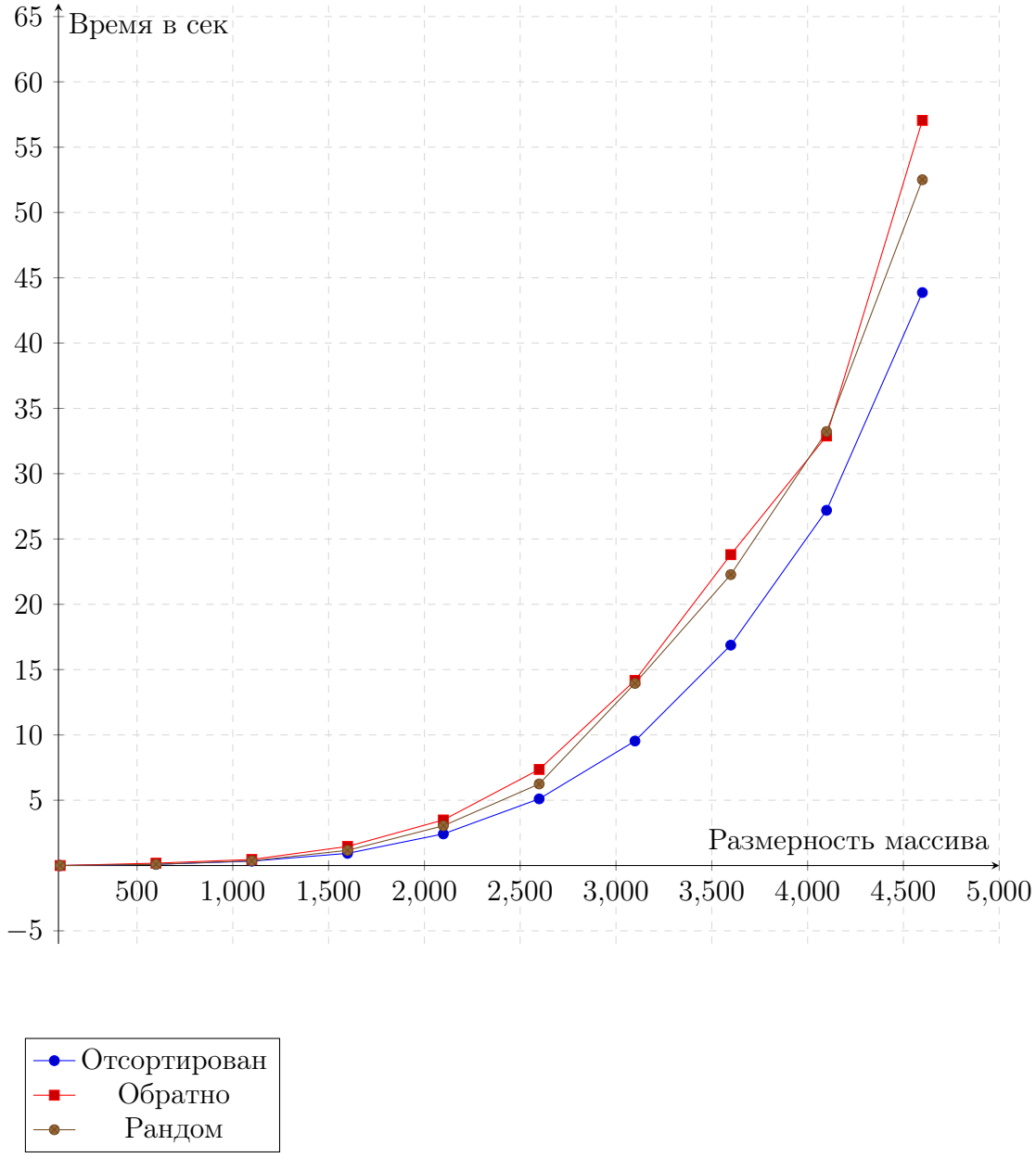


График времени работы сортировки вставками.

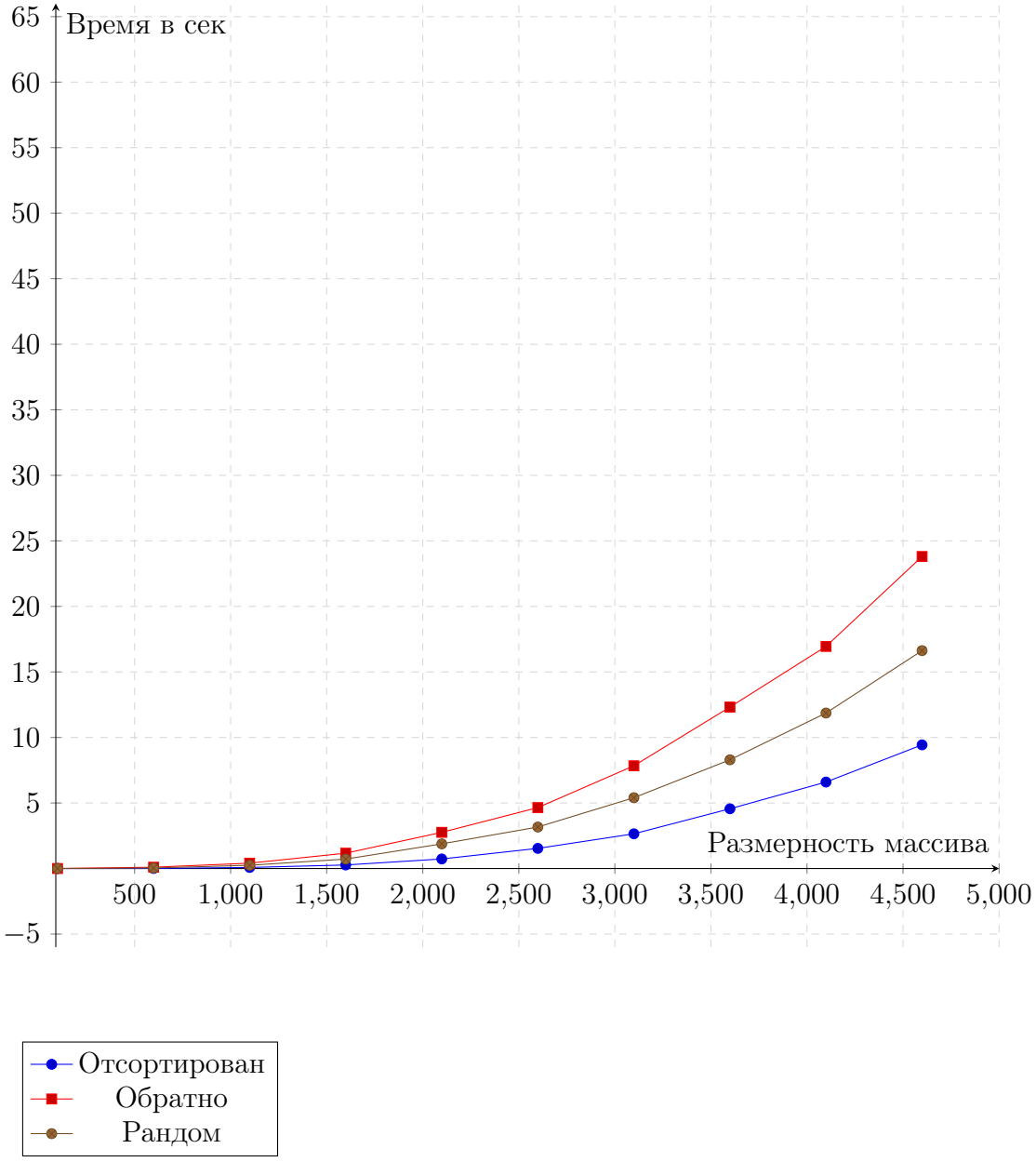
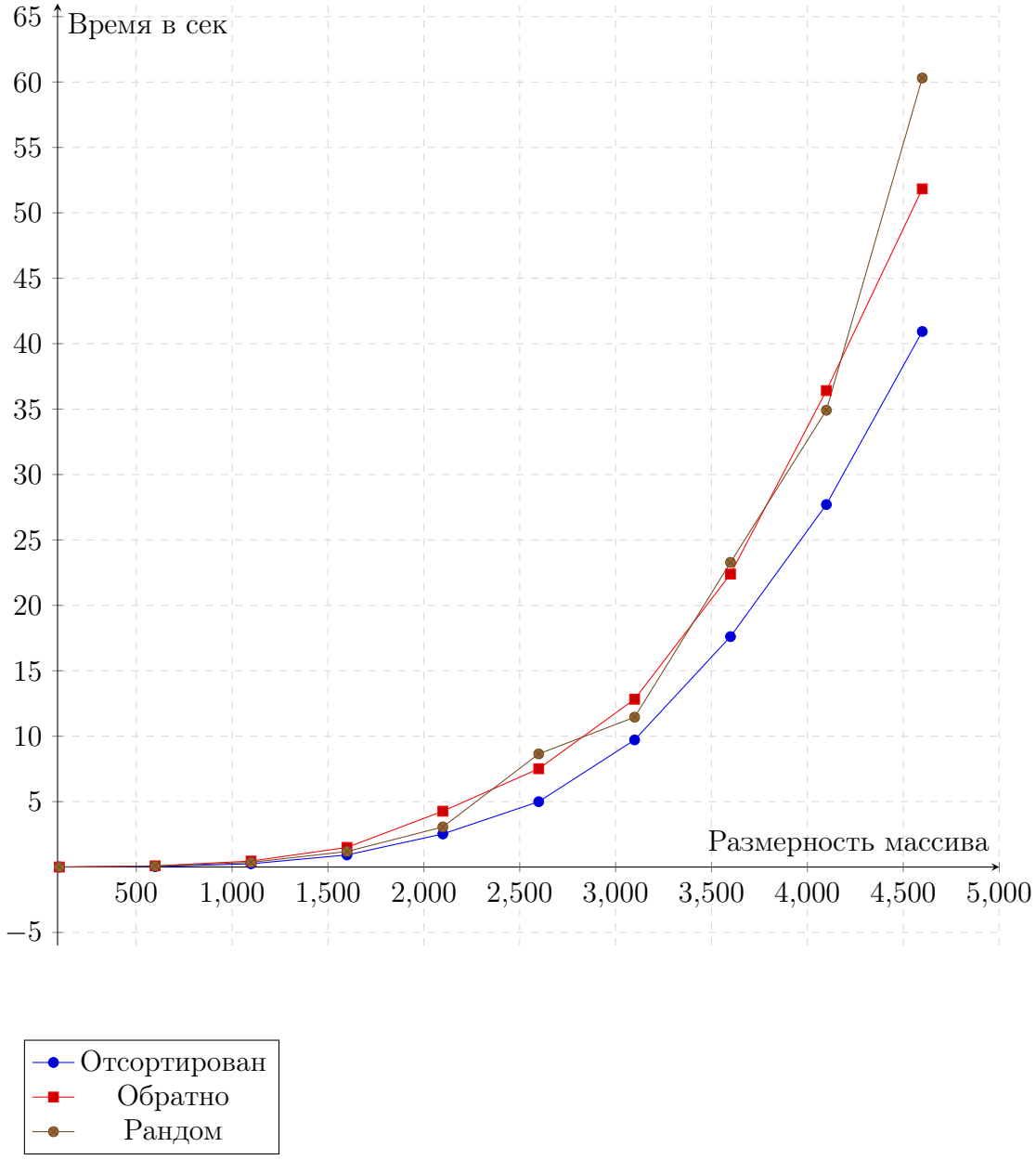


График времени работы сортировки Шейкера.



4.3 Вывод

Для каждого алгоритма уже отсортированный массив является лучшим случаем, что соответствует теории, потому что в этом случае не требуется совершать перестановки. Для сортировки пузырьком худший случай ожидаемо стал обратно отсортированный массив, потому что в этом варианте требуется наибольшее количество перестановок. Для Шейкер сортировки худший случай - случайный массив. Потому что случайный массив может сложиться таким образом, что требуется наибольшее число перестановок. Из графика видно, что быстрота времени работы обратно отсортированного и рандомного массива постоянно меняется, это связано с распределением случайных чисел. Из графиков видно, что сортировка вставками работает быстрее в любом случае.

5 Заключение

В данной лабораторной работе вычислены сложности алгоритмов для сортировки массивов. Разработаны программы по этим алгоритмам, проведены тесты по времени, произведен сравнительный анализ алгоритмов. Для составления отчета использован Latex.

6 Список литературы

1.Анализ алгоритмов. Дж. Макконнелл, 2004
2 http://algolab.valemak.com/cocktail