Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №3

По дисциплине: Анализ Алгоритмов

Тема: Трудоемкость алгоритмов сортировки

Студент Чаушев А.К..

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

Введение

В настоящее время необходимо сортировать большие объемы данных. Для этой цели существуют алгоритмы сортировки, которые упорядочивают элементы в списке.

Сортировкой [4] называют процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Определенный порядок (например, упорядочение в алфавитном порядке, по возрастанию или убыванию количественных характеристик, по классам, типам и.т.п.) в последовательности объектов необходимо для удобства работы с этим объектом.

В частности, одной из целей сортировки является облегчение последующего поиска элементов в отсортированном множестве. Под поиском подразумевается процесс нахождения в заданном множестве объекта, обладающего свойствами или качествами задаваемого шаблона.

Целями данной лабораторной работы является:

1) реализовать три различных алгоритма сортировки;

2) теоретически вычислить эффективность алгоритмов;

3) сравнить алгоритмы по времени.

1 Аналитическая часть

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные упорядочены.

1.1 Описание задачи

Алгоритм сортировки[3] — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

1.1.1 Сортировка пузырьком

Алгоритм сортировка пузырьком меняет местами два соседних элемента, если первый элемент массива больше второго. Так происходит до тех пор, пока алгоритм не обменяет местами все неотсортированные элементы.

Сложность по времени

- худшее время: O(n^2)
- среднее время: O(n^2)
- лучшее время: O(n)

Затраты по памяти

- основной: O(1)

1.1.2 Сортировка выбором

Алгоритм сортировка вставками сортирует массив по мере прохождения по его элементам. Сначала нужно рассмотреть подмножество массива и найти в нём максимум (или минимум). Затем выбранное значение меняют местами со значением первого неотсортированного элемента. Этот шаг нужно повторять до тех пор, пока в массиве не закончатся неотсортированные подмассивы.

Сложность по времени

- худшее время: O(n^2)
- среднее время: O(n^2)
- лучшее время: O(n^2)

Затраты по памяти

- основной: O(n)

1.1.3 Сортировка шейкером

Алгоритм сортировка шейкером – сортировка отличается от пузырьковой тем, что она двунаправленная: алгоритм перемещается не строго слева направо, а сначала слева направо, затем справа налево.

Сложность по времени

- худшее время: O(n^2)
- среднее время: O(n^2)
- лучшее время: O(n)

Затраты по памяти

- основной: O(1)

2 Конструкторская часть

Рассмотрим сортировку пузырьком, выбором и шейкером.

2.1 Схемы алгоритмов

На рисунке 1 изображена схема алгоритма сортировки пузырьком.

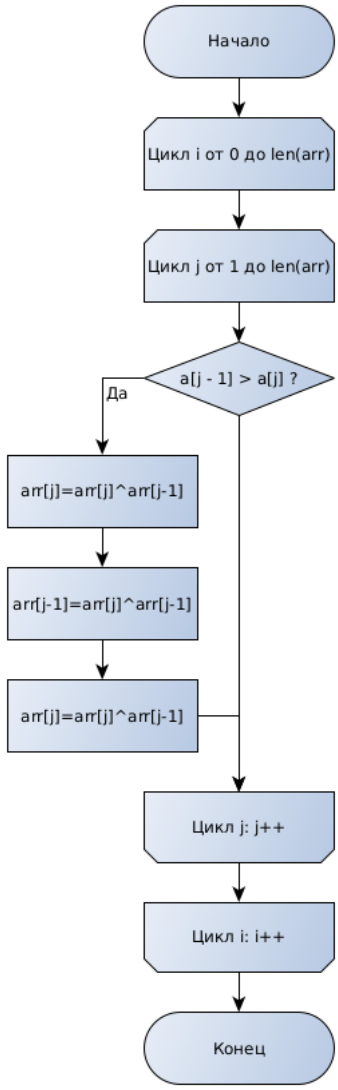

Рис. 1 – Сортировка пузырьком

На рисунке 2 изображена схема алгоритма сортировки выбором.

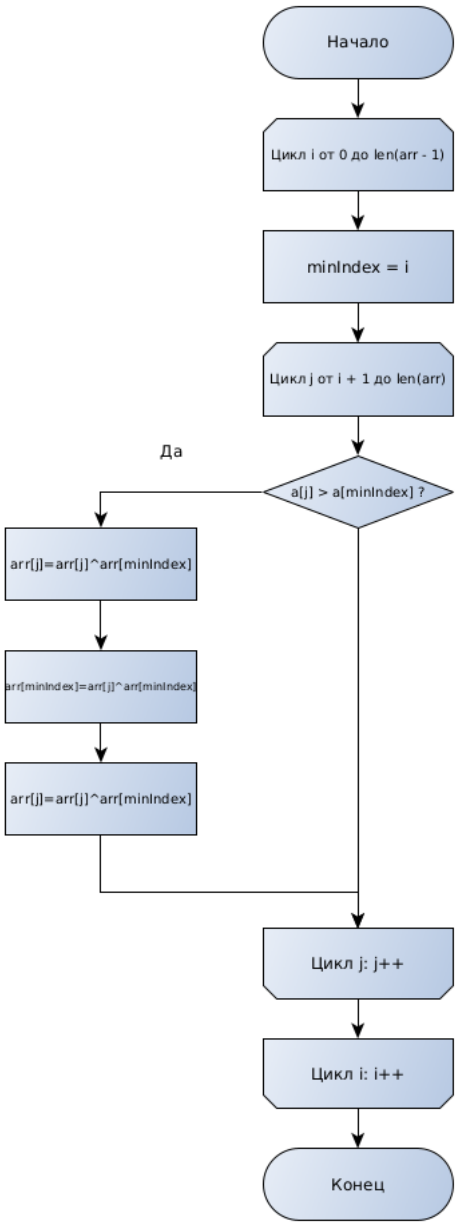

Рис. 2 – Сортировка выбором

На рисунке 3 – 4 изображена схема алгоритма сортировки шейкером.

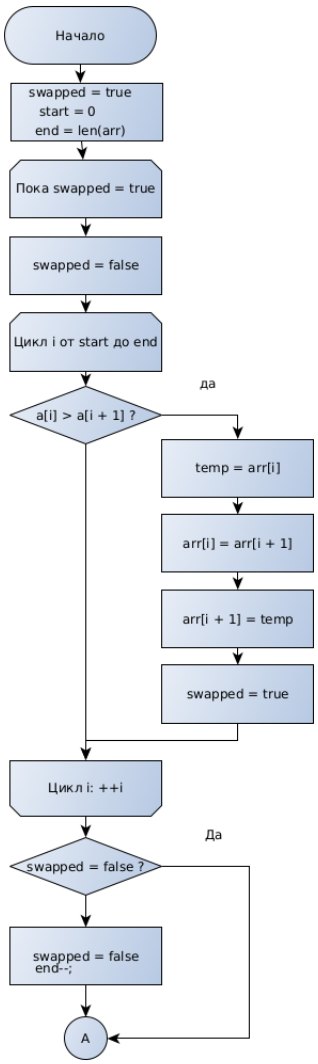

Рис. 3 – Сортировка шейкером Часть 1

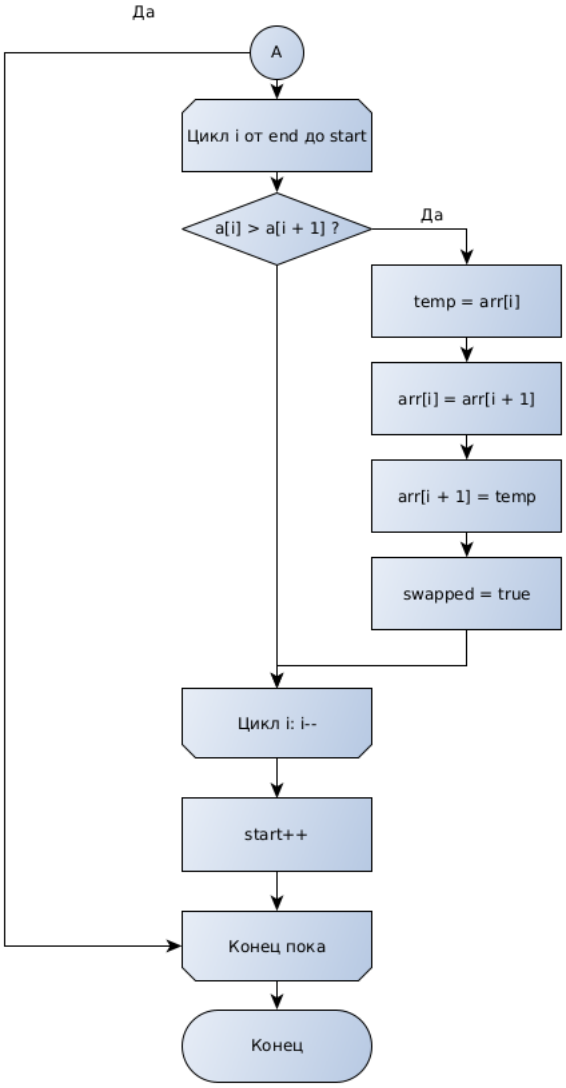

Рис. 4 – Сортировка шейкером Часть 2

3 Технологическая часть

3.1 Требования к программному обеспечению

Программное обеспечение должно обеспечивать замер процессорного времени выполнения каждого алгоритма. Проводятся замеры для случайно генерируемых массивов размерности до 10000.

3.2 Средства реализации

Java [1] — строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle).

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. На 2019 год Java — один из самых популярных языков программирования.

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

3.3 Методы замера времени в программе

3.3.1 Время

На листингах 1 – 2 показаны методы измерения времени в джаве.

Листинг 1 – Пример System.currentTimeMillis()

```
class Time {
    public static long getCurrentTime() {
        return System.currentTimeMillis();
    }
```

Единственным недостатком этого подхода является то, что "реальное"время doSomething(), которое требуется выполнить, может сильно различаться в зависимости от того, какие другие программы работают в системе и какова его нагрузка. Это делает измерение производительности несколько неточным.

Листинг 2 – Пример ThreadMXBean. [2],

```
public class CPUTime {
    public static long getCPUTime() {
        ThreadMXBean bean = ManagementFactory.getThreadMXBean();
        return bean.isCurrentThreadCpuTimeSupported() ?
                bean.getCurrentThreadCpuTime() : 0L;
    }
}
```

Более точный способ отслеживания времени, затрачиваемого на выполнение кода, при условии, что код является однопоточным, - это смотреть на время процессора, потребляемое потоком во время вызова. Это можно сделать с помощью классов ‘JMX‘; в частности, с ‘ThreadMXBean‘. Нужно получить экземпляр ‘ThreadMXBean‘ из ‘java.lang.management.ManagementFactory‘, и если платформа поддерживает его (большинство из них), использование ‘getCurrentThreadCpuTime‘ вместо ‘System.currentTimeMillis‘ выполнить аналогичный тест. ‘getCurrentThreadCpuTime‘ сообщает время в наносекундах, а не миллисекундах.

3.3.2 Улучшение точности замеров времени

Чтобы получить более плавные результаты каждый тест запускается несколько раз.

3.4 Листинг кода

Результаты разработки указаны на листингах 3, 4, 5.

Листинг 3 – Сортировка пузырьком

```
public static void bubbleSort(int[] arr) {
        final int n = arr.length;

        for (int i = 0; i < n; i++) {
            for (int j = 1; j < n; j++) {
                if (arr[j - 1] > arr[j]) {

                    arr[j] = arr[j] ^ arr[j - 1];
                    arr[j - 1] = arr[j] ^ arr[j - 1];
                    arr[j] = arr[j] ^ arr[j - 1];

                }
            }
        }
    }
```

Листинг 4 – Сортировка выбором

```
for (int i = 0; i < arr.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) {
                    arr[j] = arr[j] ^ arr[minIndex];
                    arr[minIndex] = arr[j] ^ arr[minIndex];
                    arr[j] = arr[j] ^ arr[minIndex];
                }
            }
        }
```

Листинг 5 – Сортировка шейкером

```
public static void shakerSort(int[] arr) {
        {
            boolean swapped = true;
            int start = 0, end = arr.length;
            while (swapped == true) {
                swapped = false;
                for (int i = start; i < end - 1; ++i) {
                    if (arr[i] > arr[i + 1]) {
                        int temp = arr[i];
                        arr[i] = arr[i + 1];
                        arr[i + 1] = temp;
                        swapped = true;
                    }
                }
                if (swapped == false)
                    break;
                swapped = false;
                end = end - 1;
                for (int i = end - 1; i >= start; i--) {
                    if (arr[i] > arr[i + 1]) {
                        int temp = arr[i];
                        arr[i] = arr[i + 1];
                        arr[i + 1] = temp;
                        swapped = true;
                    }
                }
                start = start + 1;
            }
        }
    }
```

4 Экспериментальная часть

Проведем тестирование и сравним алгоритмы по времени работы.

4.1 Примеры работ

Ниже приведены примеры работ.
Листинг 7 – Пример работы алгоритмов на сортированный массив.

```
Сортировка сортированного масива
Массив: [-123, 4, 67, 123, 123, 124, 150]
Ожидаемый: [-123, 4, 67, 123, 123, 124, 150]
Пузырком: [-123, 4, 67, 123, 123, 124, 150]
Выбором: [-123, 4, 67, 123, 123, 124, 150]
Шейкером: [-123, 4, 67, 123, 123, 124, 150]
```

Листинг 8 – Пример работы алгоритмов на массив наоборот.

```
Сортировка Наоборот
Массив: [150, 124, 123, 123, 67, 4, -123]
Ожидаемый: [-123, 4, 67, 123, 123, 124, 150]
Пузырком: [-123, 4, 67, 123, 123, 124, 150]
Выбором: [-123, 4, 67, 123, 123, 124, 150]
Шейкером: [-123, 4, 67, 123, 123, 124, 150]
```

Листинг 9 – Пример работы алгоритмов на массив одинаковых элементов.

```
Сортировка одинаковых элементов
Массив: [5, 5, 5, 5, 5, 5, 5]
Ожидаемый: [5, 5, 5, 5, 5, 5, 5]
Пузырком: [5, 5, 5, 5, 5, 5, 5]
Выбором: [5, 5, 5, 5, 5, 5, 5]
Шейкером: [5, 5, 5, 5, 5, 5, 5]
```

Листинг 10 – Пример работы алгоритмов на массив размешанных элементов.

```
Смешеная сортировка
Массив: [-123, 23, -123, 4, 5, 6, 2, 0, 1]
Ожидаемый: [-123, -123, 0, 1, 2, 4, 5, 6, 23]
Пузырком: [-123, -123, 0, 1, 2, 4, 5, 6, 23]
Выбором: [-123, -123, 0, 1, 2, 4, 5, 6, 23]
Шейкером: [-123, -123, 0, 1, 2, 4, 5, 6, 23]
```

4.2 Замеры времени

На таблице 1 представлены результаты замера времени для отсортированного массива, на таблице 2 - для обратно отсортированного и на таблице3 - для случайно сгенерированного массива.

Таблица 1 – Результаты замеры времени на упорядоченном массиве

Алгоритм	100	300	500	700	1000
Сортировка пузырьком	111490	60071	125203	197089	394396
Сортировка выбором	91669	33638	66600	112818	142985
Сортировка шейкером	32864	13940	11877	5632	2750

Таблица 2 – Результаты замеры времени на обратном массиве

Алгоритм	100	300	500	700	1000
Сортировка пузырьком	5954	27427	72556	141314	276743
Сортировка выбором	3880	13709	35956	64840	120530
Сортировка шейкером	2461	2783	2356	2988	1994

Таблица 3 – Результаты замеры времени на случайном массиве

Алгоритм	100	300	500	700	1000
Сортировка пузырьком	47576	45300	89985	167630	351622
Сортировка выбором	21917	26949	36777	110407	127631
Сортировка шейкером	5318	7368	5035	3169	1775

4.3 Выводы

Из таблиц зависимости размера массива ко времени сортировки видно, что сортировка пузырьком работает медленее чем сортировки выбором и шейкера. Также можно заметить, что сортировка шейкером быстрее сортировки пузырьком в 6-7 раз.

4.4 Оценка трудоемкосты алгоритмов сортировки

1. **Сортировка пузырьком**

   В лучшем случае, когда массив будет отсортирован, трудоемкость будет считать по формуле 1.

   $$f = 1 + 2N + \frac{N(N-1)}{2} \cdot 6 = 3N^2 - N + 1 \tag{1}$$

   В худшем случае, когда массив будет обратно отсортирован, трудоемкость будет считаться по формуле 2.

   $$f = 1 + 2N + \frac{N(N-1)}{2} \cdot 11 = \frac{11}{2}N^2 - \frac{7}{2}N + 1 \tag{2}$$

   И в лучшем, и в худшем случае, сортировка пузьком имеет трудоемкость $O(N^2)$.

2. **Сортировка шейкером**

   И в лучшем, и в худшем случае, сортировка шейкером иммет трудоемкость $O(N^2)$

Заключение

В ходе данной работы было проведено сравнение трех алгоритмов сортировки: сортировка пузырьком, сортировка выбором и сортировка шейкером. Были сделаны следующие выводы:

- сортировка шейкером на порядок быстрее сортировки пузырьком и выбором;
- сортировка шейкером быстрее пузырька в 6-7 раз.

Все задачи, поставленные в данной работе были выполнены:

1) реализовано три различных алгоритма сортировки;
2) теоретически вычислена эффективность алгоритмов;
3) сравнены алгоритмы по времени и памяти.

Список литературы

[1] Oracle. *Java [ЭЛ. РЕСУРС] Режим доступа:* URL: `https://www.oracle.com/java/`. (дата обращения: 03.10.2020).

[2] Oracle. *ThreadMXBean [ЭЛ. РЕСУРС] Режим доступа:* URL: `https://docs.oracle.com/javase/7/docs/api/java/lang/management/ThreadMXBean.html`. (дата обращения: 03.10.2020).

[3] Yandex. *Основные виды сортировок и примеры их реализации[ЭЛ. РЕСУРС] Режим доступа:* URL: `https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii`. (дата обращения: 07.10.2020).

[4] Глушко. *Алгоритм сортировки [ЭЛ. РЕСУРС] Режим доступа:* URL: `https://works.doklad.ru/view/MeaUSqCgyps.html`. (дата обращения: 07.10.2020).