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 курсу ”Анализ алгоритмов”

на тему:

«Алгоритмы сортировки»

Студент **Голикова С. М.**

Группа **ИУ7-55Б**

Оценка (баллы)

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

Москва, 2022г.

Содержание

Введение

Одной из важнейших процедур обработки информации является сортировка. Под сортировкой понимается упорядочивание элементов последовательности по какому-либо признаку [1].

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить там, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить [2]. С упорядоченными данными можно столкнуться в телефонных книгах, интернет-магазинах, библиотеках, словарях.

В настоящее время, в связи с постоянно растущими объемами данных и распространением BigData технологий, вопрос эффективности алгоритмов сортировки становится особенно актуальным [3].

Целью работы является получение навыков сравнения алгоритмов по трудоемкости и процессорному времени на примере трех алгоритмов сортировки.

В рамках выполнения работы необходимо решить следующие задачи:

- изучить три алгоритма сортировки: пузырьком, вставками, блочной;
- описать и реализовать изученные алгоритмы;
- провести сравнительный анализ трудоемкости реализаций алгоритмов на основе теоретических расчетов;
- провести сравнительный анализ процессорного времени выполнения реализаций алгоритмов на основе экспериментальных данных.

1 Аналитический раздел

В данном разделе будут рассмотрены основные идеи трех алгоритмов сортировки: пузырьком, вставками и блочной, а также будет произведена формализация задачи.

1.1 Формализация задачи

Пусть A – массив, содержащий N чисел. Массив считается отсортированным по возрастанию, если для каждого элемента массива выполняется следующее соотношение 1:

$$\forall i, j \in [1, N] : i \leq j; A_i \leq A_j \tag{1}$$

Массив считается отсортированным по убыванию, если для каждого элемента массива выполняется следующее соотношение 2:

$$\forall i, j \in [1, N] : i \leq j; A_i \geq A_j \tag{2}$$

В ходе выполнения работы требуется написать алгоритм, который сортирует массив из произвольных данных по возрастанию.

1.2 Сортировка 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 («всплывает» до нужной позиции, как пузырёк в воде – отсюда и название алгоритма) [1].

1.3 Сортировка вставками

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. Алгоритму

необходимо для каждого нового элемента выбрать нужное место для вставки в уже упорядоченный массив данных, так что элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов [4].

1.4 Блочная сортировка

Блочная (корзинная или карманная) сортировка состоит в следующем. Пусть l – минимальный, а r – максимальный элемент массива. Разобьем элементы на блоки, в первом будут элементы от l до $l + k$, во втором – от $l + k$ до $l + 2k$ и т.д., где $k = (r - l)$ / количество блоков. Если количество блоков равно двум, то данный алгоритм превращается в разновидность быстрой сортировки [5].

Вывод

В данном разделе были рассмотрены идеи, лежащие в основе трех алгоритмов сортировки: пузырьком, вставками, блочной.

2 Конструкторский раздел

В данном разделе будут приведены схемы рассмотренных в предыдущем разделе алгоритмов сортировки и произведен расчет их трудоемкости.

2.1 Разработка алгоритмов

На рисунках 1, 2 и 3 представлены схемы алгоритмов сортировки пузырьком, вставками и блочной соответственно.

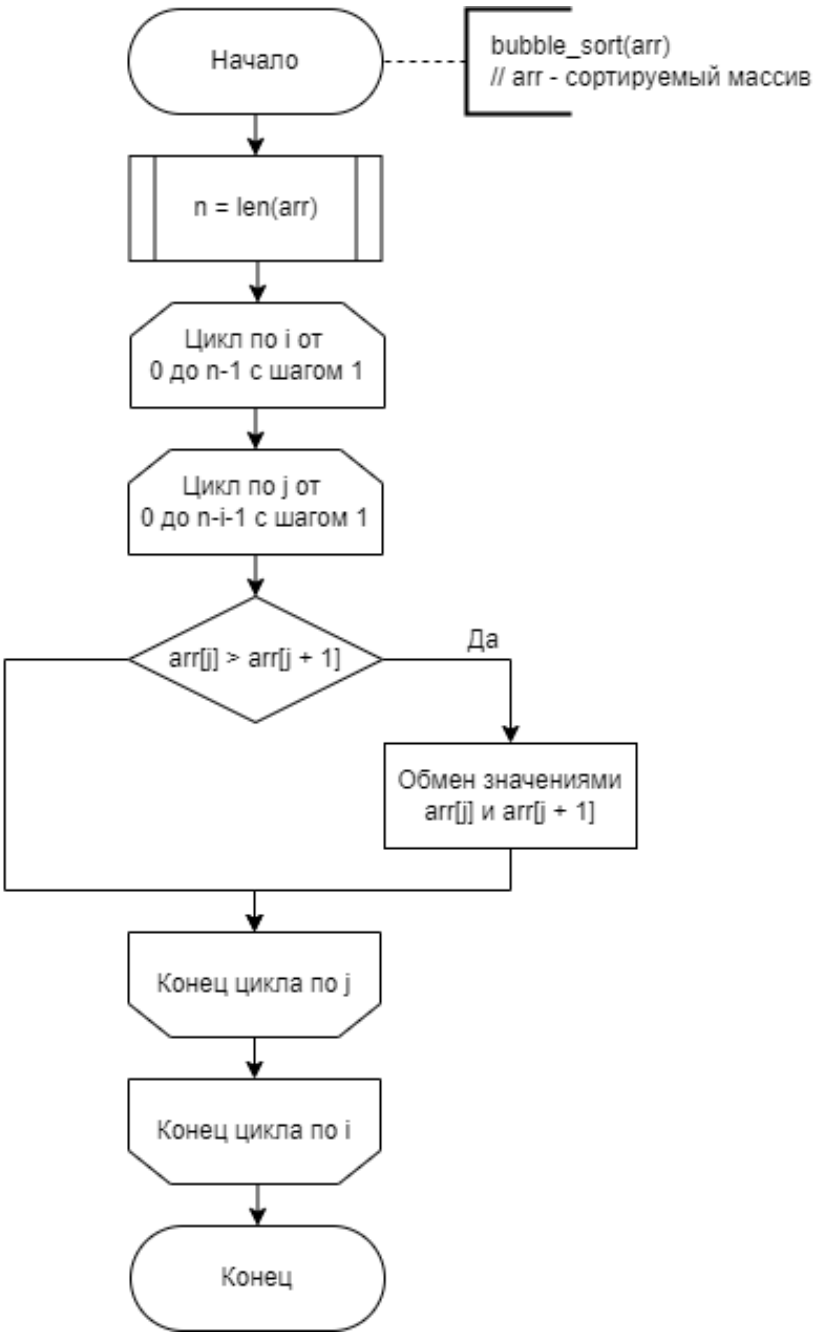

Рисунок 1 – Схема алгоритма сортировки пузырьком

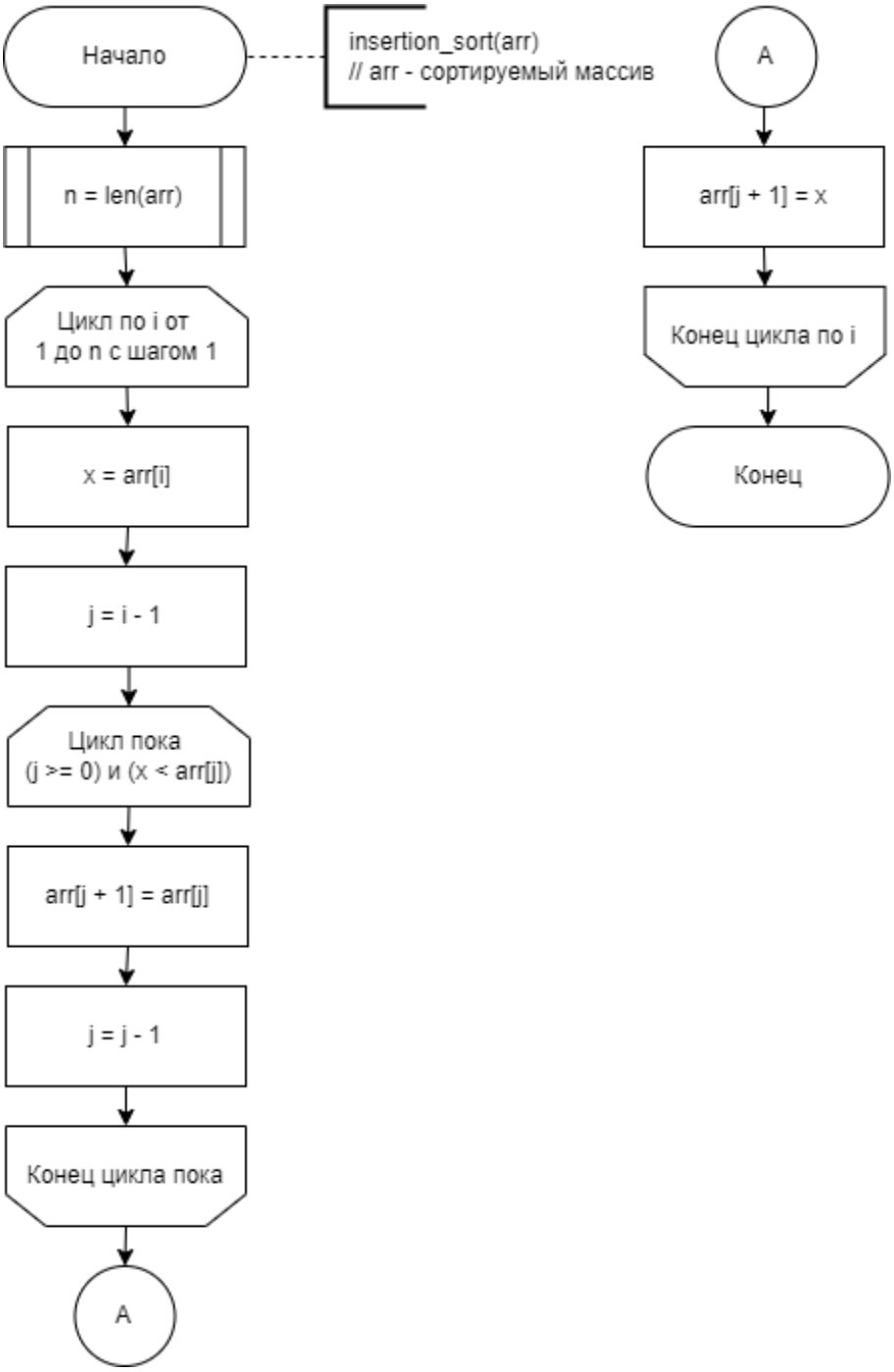

Рисунок 2 – Схема алгоритма сортировки вставками

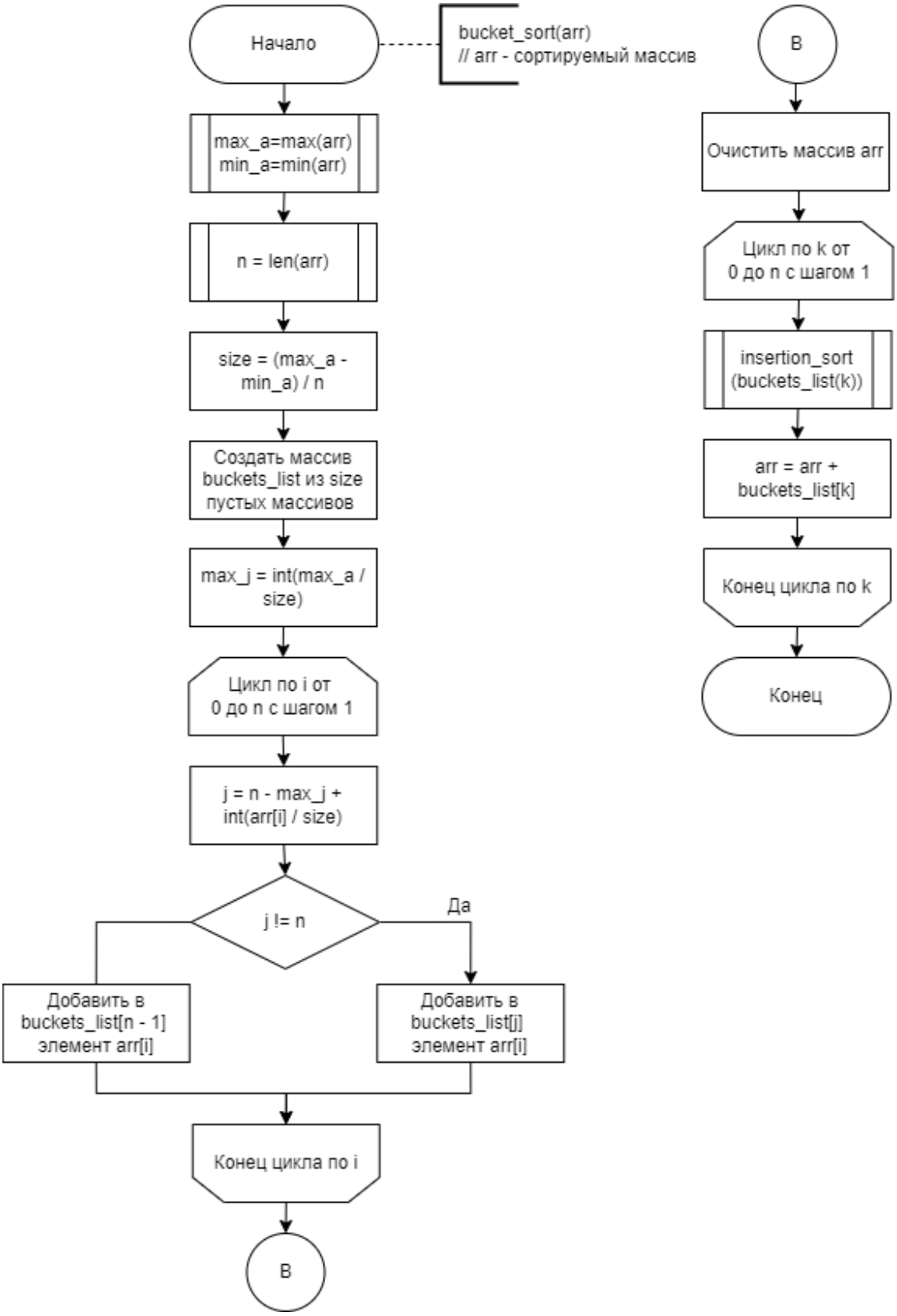

Рисунок 3 – Схема алгоритма блочной сортировки

2.2 Оценка трудоемкости

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1) Трудоемкость следующих базовых операций единична: +, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, –, «, ».

   Операции *, %, / имеют трудоемкость 2.

2) Трудоемкость цикла for(k = 0; k < N; k++) {тело цикла} рассчитывается как

$$f_{for} = f_{\text{инициал.}} + f_{\text{сравн.}} + N(f_{\text{тела}} + f_{\text{инкр.}} + f_{\text{сравн.}}) \tag{3}$$

3) Трудоемкость условного оператора if (условие) then A else B рассчитывается как

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{в лучшем случае;} \\ max(f_A, f_B), & \text{в худшем случае.} \end{cases} \tag{4}$$

2.2.1 Сортировка пузырьком

Лучший случай: массив отсортирован (ни одного захода в тело условного оператора).

$f = 1 + 2 + (n-1)(1+2+1+3) + \frac{n(n-1)}{2}(4+4+0) = 4n^2 + 3n - 4 = O(n^2)$

Худший случай: массив отсортирован в обратном порядке (на каждой итерации заход в тело условного оператора).

$f = 1 + 2 + (n-1)(1+2+1+3) + \frac{n(n-1)}{2}(4+4+7) = \frac{15}{2}n^2 - \frac{1}{2}n - 4 = O(n^2)$

2.2.2 Сортировка вставками

Лучший случай: массив отсортирован (ни одного захода во внутренний цикл).

$f = 1 + 2 + (n-1)(1+2+(2+1+4+0+2)) = 12n - 9 = O(n)$

Худший случай: массив отсортирован в обратном порядке (на каждой итерации заход во внутренний цикл).

$f = 1+2+(n-1)(1+2+(2+1+4+2)+\frac{1}{2}n(n-1)(4+5)) = \frac{9}{2}n^2+\frac{15}{2}n-9 = O(n^2)$

2.2.3 Блочная сортировка

Лучший случай: массив отсортирован, элементы распределены равномерно (все блоки содержат одинаковое число элементов).

$f = 7+3+n(1+1)+3+2+n(1+1+5+2)+1+2+n(1+1+1+12size-9+3) = 8n + 12n + 18 = 20n + 18 = O(n)$

Худший случай: элементы распределены неравномерно (большое количество пустых блоков), массив отсортирован в обратном порядке (худший случай сортировки вставками, которая используется в блочной сортировке).

$f = 7 + 3 + n(1 + 1) + 3 + 2 + n(1 + 1 + 5 + 3) + 1 + 2 + n(1 + 1 + 3) + \frac{9}{2}(n - 1)^2 + \frac{15}{2}(n - 1) - 9 - 9 = \frac{9}{2}n^2 + \frac{31}{2}n - 3 = O(n^2)$

Вывод

Были разработаны схемы трех алгоритмов сортировки, определены лучшие и худшие случаи, а также произведен расчет трудоемкости алгоритмов.

3 Технологический раздел

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги реализованных алгоритмов и тесты для программы.

3.1 Требования к ПО

Ниже представлен ряд требований к разрабатываемому программному обеспечению.

Требования к входным данным: массив состоит из целых чисел; длина массива может быть нулевой.

Требования к выходным данным: входной массив отсортирован по возрастанию каждым реализованным алгоритмом; на экран выведены графические результаты замеров времени сортировки каждым алгоритмом для трех случаев, которые подробно рассмотрены в пункте 4.3.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран Python [6]. Данный язык предоставляет возможности работы с массивами.

Для замера процессорного времени использовалась функция process_time библиотеки time [7].

3.3 Реализации алгоритмов

В листингах 1, 2, 3 представлены реализации алгоритмов сортировки (пузырьком, вставками и блочной сортировки).

Листинг 1 – Алгоритм сортировки пузырьком

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

    return arr
```

Листинг 2 – Алгоритм сортировки вставками

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        x = arr[i]
        j = i - 1
        while j >= 0 and x < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = x

    return arr
```

Листинг 3 – Алгоритм блочной сортировки

```
def bucket_sort(arr):
    max_a = max(arr)
    min_a = min(arr)
    n = len(arr)
    size = (max_a - min_a) / n

    buckets_list = []
    for x in range(n):
        buckets_list.append([])

    max_j = int(max_a / size)
    for i in range(n):
        j = n - max_j + int(arr[i] / size)
        if j != n:
            buckets_list[j].append(arr[i])
        else:
            buckets_list[n - 1].append(arr[i])

    arr = []
    for k in range(n):
        insertion_sort(buckets_list[k])
        arr = arr + buckets_list[k]

    return arr
```

3.4 Тестирование

В таблице 1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно каждой реализацией алгоритма.

Таблица 1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[1000, 4, 3, 2, 1]$	$[1, 2, 3, 4, 1000]$	$[1, 2, 3, 4, 1000]$
$[2, 3, 1, 5, 4]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[3, 5, 3, 1, 1]$	$[1, 1, 3, 3, 5]$	$[1, 1, 3, 3, 5]$
$[1]$	$[1]$	$[1]$
$[]$	$[]$	$[]$

Вывод

Были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательский раздел

В данном разделе будет приведена демонстрация работы программы, а также произведен сравнительный анализ реализаций алгоритмов на основе времени их работы.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

- операционная система Windows 10 [8];
- память 8 ГБ;
- процессор Intel® Core™ i5-6260U @ 1.80 ГГц [9].

Замеры времени выполнения реализаций алгоритмов проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только приложением и средой разработки.

4.2 Демонстрация работы программы

На рисунке 4 представлен пример работы программы. Генерируется массив из случайных целых чисел. Исходный массив сортируется с помощью трех реализованных алгоритмов сортировки. Результат можно сравнить с ожидаемым, полученным в результате вызова встроенной функции sorted языка Python.

```
Начальный массив:          [-2116, 1545, -8524, 8829, -9184, -7862, 5601, 7594, 6891, 8864]
Ожидаемый результат:       [-9184, -8524, -7862, -2116, 1545, 5601, 6891, 7594, 8829, 8864]
--------------------------
Сортировка пузырьком:      [-9184, -8524, -7862, -2116, 1545, 5601, 6891, 7594, 8829, 8864]
Сортировка вставками:      [-9184, -8524, -7862, -2116, 1545, 5601, 6891, 7594, 8829, 8864]
Блочная сортировка:        [-9184, -8524, -7862, -2116, 1545, 5601, 6891, 7594, 8829, 8864]
```

Рисунок 4 – Пример работы программы

4.3 Сравнение времени работы реализаций алгоритмов

Для сравнения времени работы реализаций алгоритмов рассматривалось три случая:

1) отсортированный массив (лучший случай);

2) неравномерно распределенный массив, отсортированный в обратном порядке (худший случай);
3) массив из случайных чисел (произвольный случай).

На рисунках 5, 6 и 7 приведены зависимости времени работы реализаций алгоритмов сортировки от размеров массивов на отсортированных, обратно отсортированных и случайных данных соответственно.

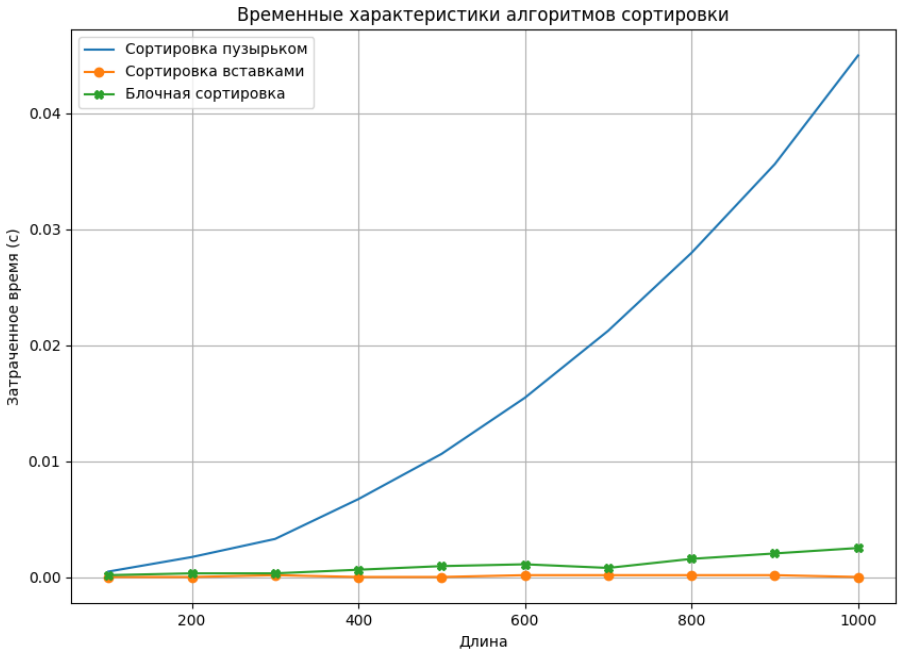

Рисунок 5 – Зависимость времени работы реализации алгоритма сортировки от размера массива для лучшего случая

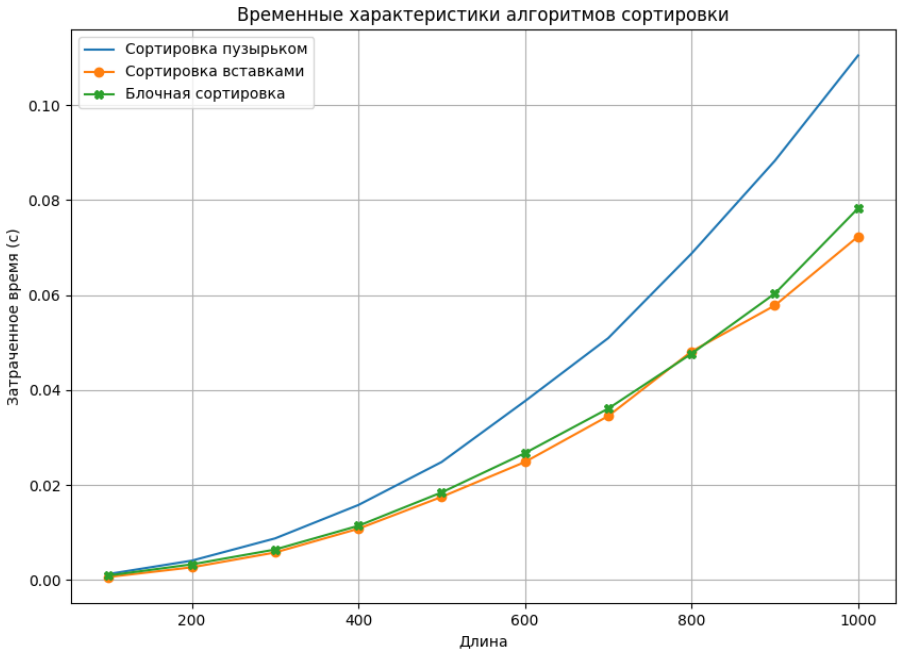

Рисунок 6 – Зависимость времени работы реализации алгоритма сортировки от размера массива для худшего случая

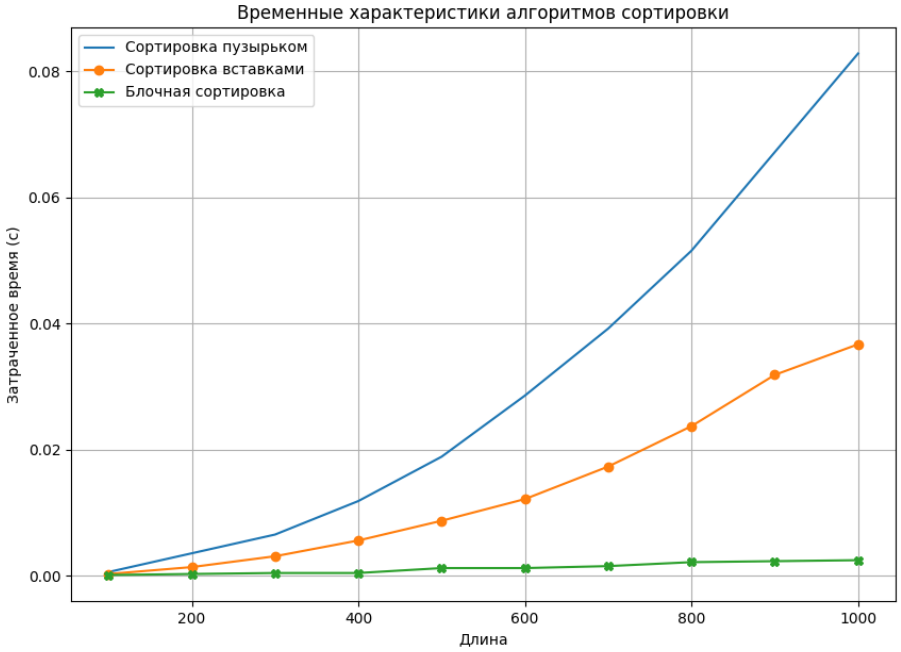

Рисунок 7 – Зависимость времени работы реализации алгоритма сортировки от размера массива для произвольного случая

4.4 Вывод

Реализация алгоритма сортировки пузырьком является самой медленной для всех случаев.

В случае предварительно отсортированного массива сортировка вставками превосходит другие алгоритмы, хотя блочная не сильно от нее отстает.

Для неравномерно распределенного массива, отсортированного в обратном порядке, результаты замеров для сортировки вставками и блочной сортировки практически идентичны.

На случайных данных самой быстрой является реализация алгоритма блочной сортировки.

Таким образом, подтвердились следующие оценки трудоемкости:

- реализации алгоритмов сортировки вставками и блочной сортировки в лучшем случае работают на порядок быстрее реализации сортировки пузырьком;
- реализации алгоритмов сортировки вставками и блочной сортировки практически идентичны по времени работы в худшем случае.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- изучены три алгоритма сортировки – пузырьком, вставками, блочной;
- описаны и реализованы изученные алгоритмы;
- проведен сравнительный анализ трудоёмкости реализаций алгоритмов на основе теоретических расчетов;
- проведен сравнительный анализ процессорного времени выполнения реализаций алгоритмов на основе экспериментальных данных.

Реализация алгоритма сортировки пузырьком работает примерно в 18 раз дольше двух других реализаций на 1000 элементах в лучшем случае и примерно в 1.57 раз дольше двух других реализаций на 1000 элементах в худшем случае. В произвольном случае на 1000 элементах реализация алгоритма сортировки пузырьком работает в 2.25 раз дольше реализации сортировки вставками и примерно в 30 раз дольше реализации блочной сортировки.

Поставленная цель была достигнута: были получены навыки сравнения алгоритмов по трудоемкости и процессорному времени выполнения реализаций на примере трех алгоритмов сортировки.

Список использованных источников

1. Шагбазян., Д. В. Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие / Д.В. Шагбазян, А.А. Штанюк, Е.В. Малкина. — Нижний Новгород: Нижегородский госуниверситет, 2019. — 42 с.

2. Лекция 43: Алгоритмы сортировки массивов. Внутренняя сортировка [Электронный ресурс]. Режим доступа: https://intuit.ru/studies/courses/648/504/lecture/11472 (дата обращения: 22.09.2022).

3. Shatnawi, A. Toward a new approach for sorting extremely large data files in the big data era / A. Shatnawi, Y. AlZahouri, M. A. Shehab [et al] // Cluster Computing. — 2019. — Vol. 22. — P. 819–828.

4. Сортировка вставками [Электронный ресурс]. Режим доступа: http://cppstudio.com/post/462/ (дата обращения: 22.09.2022).

5. Тема 3. Компьютерный анализ данных. Лекция 10. Методы и алгоритмы обработки и анализа данных [Электронный ресурс]. Режим доступа: http://imamod.ru/~polyakov/arc/stud/mmca/lecture_10.pdf (дата обращения: 28.09.2022).

6. Лутц, Марк. Изучаем Python, том 1, 5-е изд. — Пер. с англ. – СПб.: ООО “Диалектика”, 2019. — 832 с.

7. time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 22.09.2022).

8. Windows client documentation for IT Pros [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/windows/resources/ (дата обращения: 22.09.2022).

9. Процессор Intel® Core™ i5-6260U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/91160/intel-core-i56260u-processor-4m-cache-up-to-2-90-ghz.html (дата обращения: 22.09.2022).