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"

Тема Трудоемкость сортировок

Студент Цветков И.А.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Сортировка - перегруппировка некой последовательности, или кортежа, в определенном порядке. Это одна из главных процудур обработки структурированных данных. Расположение элементов в определенном порядке позволяет более эффективно проводить работу с последовательностью данных, в частности при поиске некоторых данных.

Существует множество алгоритмов сортировки, но любой алгоритм сортировки имеет:

- сравнение, которое определяет, как упорядочена пара элементов;
- перестановка для смены элементов местами;
- алгоритм сортировки, использующий сравнение и перестановки.

Что касается самого поиска, то при работе с отсортированным набором данных время, которое нужно на нахождение элемента, пропорционально логарифму количства элементов. Последовательность, данные которой расположены в хаотичном порядке, занимает время, которое пропорционально количству элементов, что куда больше логарифма.

Цель работы: изучение и исследование трудоемкости алгоритмов сортировки.

Задачи работы.

1. Изучить и реализовать алгоритмы сортировки: слияние, вставки, шейкер.
2. Рассмотреть существующие решения.
3. Разработать алгоритмы сортировок.
4. Провести сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов.

5. Провести сравнительный анализ реализаций алгоритмов по затраченному процессорному времени и памяти.

6. Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут рассмотрены алгоритмы сортировок - вставками, перемешиванием, гномья.

1.1 Сортировка вставками

Сортировка вставками [1] - размещение элемента входной последовательности на подходящее место выходной последовательности.

Набор данных условно разделяется на входную последовательность и выходную. В начале отсортированная часть пуста. Каждый i-ый элемент, начиная с $i = 2$, входной последовательности размещается в уже отсортированную часть до тех пор, пока изначальные данные не будут исчерпаны.

1.2 Сортировка перемешиванием

Сортировка перемешиванием [2] - сортировка, которая является модификацией сортировки пузырьком. Различие состоит в том, что в рамках одной итерации происходит проход по массиву в обоих направлениях. В сортировке пузырьком просходит только проход слева-направо, то ебсть в одном направлении.

Суть сортировки - сначала идет обычный проход слева-направо, как при обычном пузырьке. Затем, начиная с элемента, который находится перед последним отсортированным, начинается проход в обратном направлении. Здесь тикже сравниваются элементы меняются местами при необходимости.

1.3 Гномья сортировка

Гномья сортировка [3] - алгоритм сортировки, который использует только один цикл, что является редкостью.

В этой сортировке массив просматривается селва-направо, при этом сравниваются и, если нужно, меняются соседние элементы. Если происходит об-

мен элементов, то происходит возвращение на один шаг назад. Если обмена не было - агоритм продолжает просмотр массива в поисках неупорядоченных пар.

Вывод

В данной работе необходимо реализовать алгоритмы сортировки, описанные в данном разделе, а также провести их теоритическую оценку и проверить ее экспериментально.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов сортировок (вставками, перемешиванием и гномья), а также найдена их трудоемкость

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки - ставками, перемешиванием и гномья

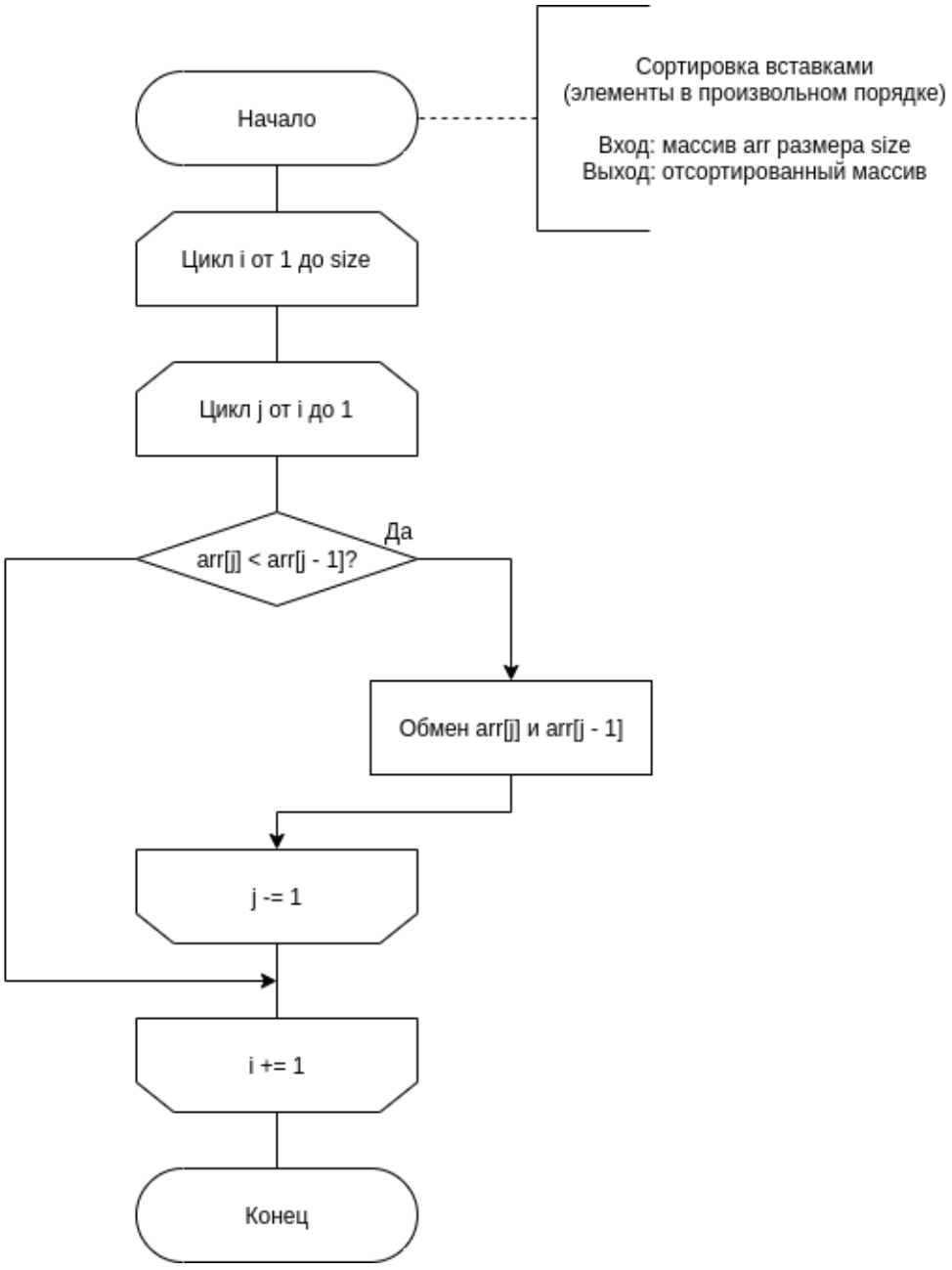

Рисунок 2.1 – Схема алгоритма сортировки вставками

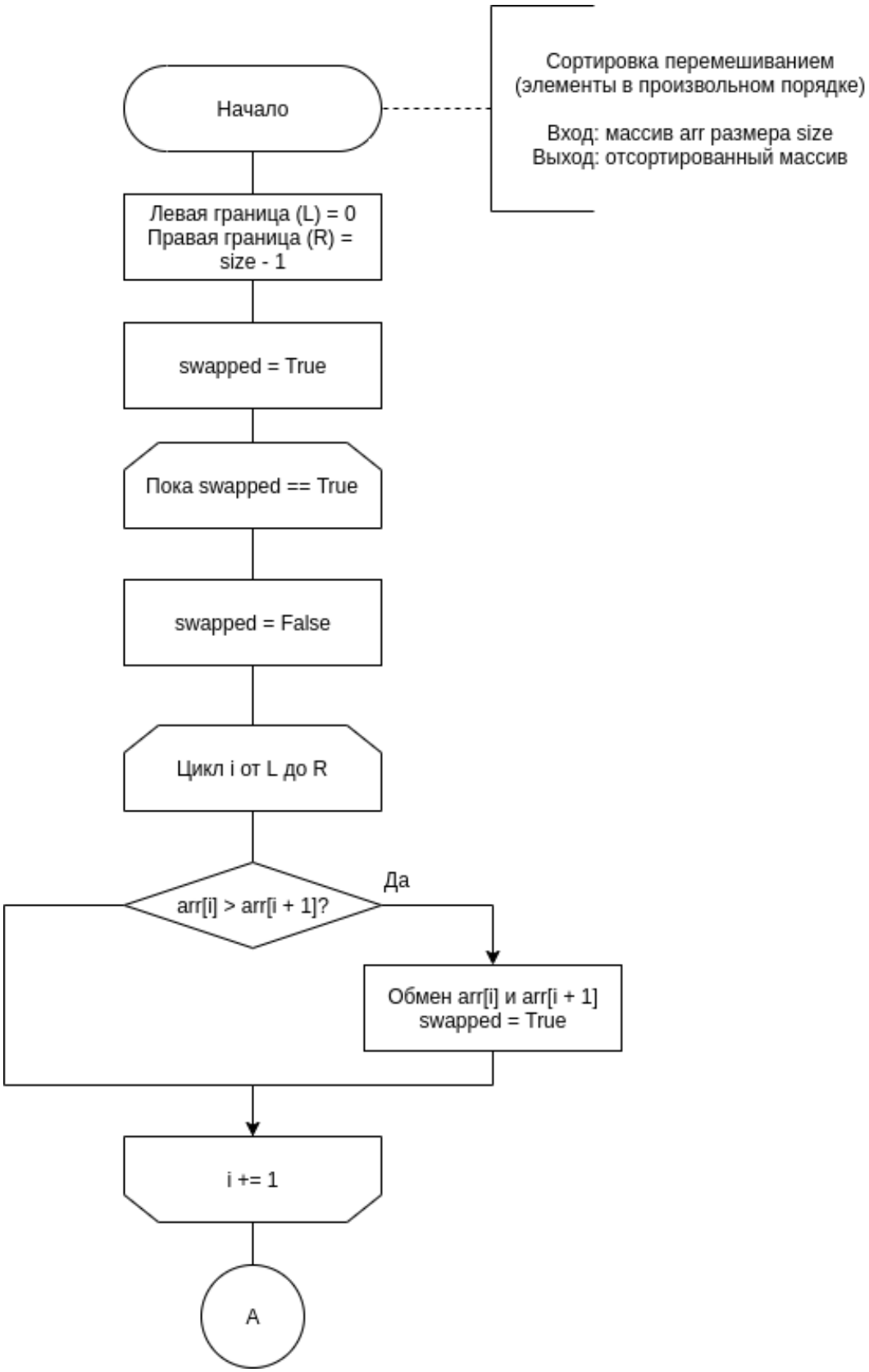

Рисунок 2.2 – Схема алгоритма сортировки перемешиванием - 1

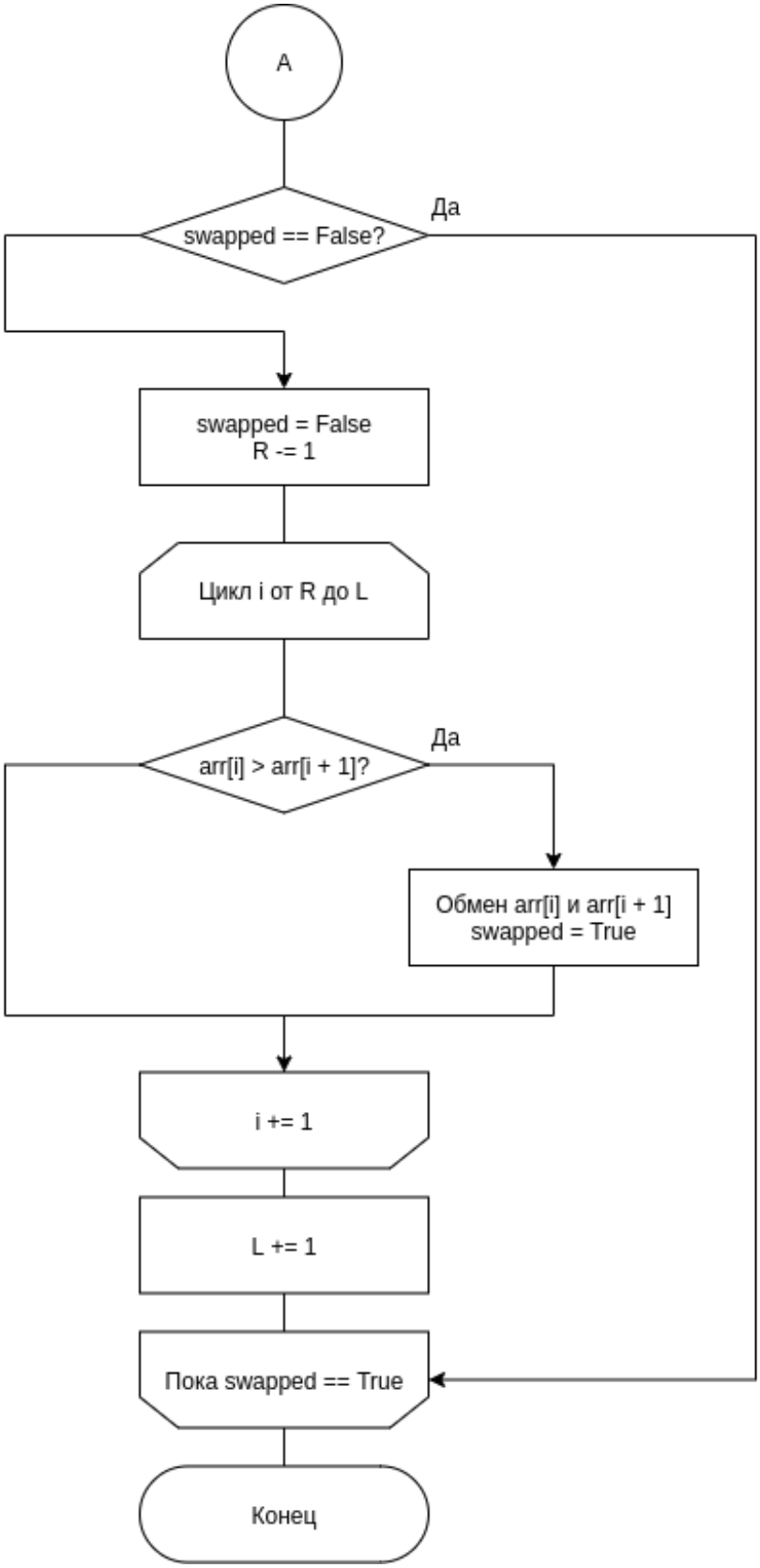

Рисунок 2.3 – Схема алгоритма сортировки перемешиванием - 2

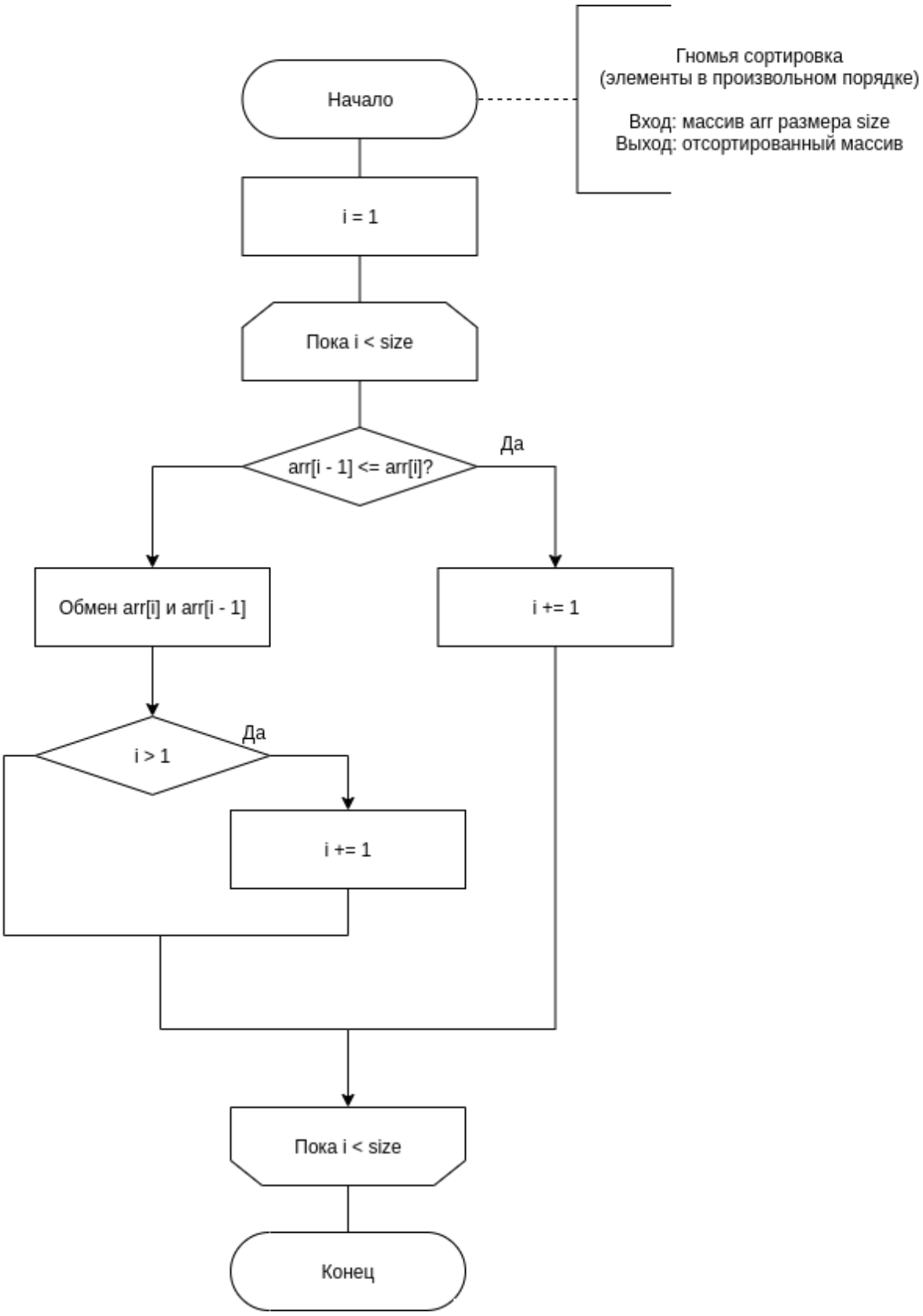

Рисунок 2.4 – Схема алгоритма гномьей сортировки

2.2 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определния трудоемкости каждого отдельно взятого алгоритма сортировки:

1. операции из списка (2.1) имеют трудоемкость равную 1;

$$+, -, /, *, \%, =, + =, - =, * =, / =, \% =, ==, ! =, <, >, <=, >=, [], ++, --$$ (2.1)

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}$$ (2.2)

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}})$$ (2.3)

4. трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

2.3.1 Алгоритм сортировки перемешиванием

- Трудоёмкость сравнения внешнего цикла $while(swap == True)$, которая равна (2.4):

$$f_{outer} = 1 + 2 \cdot (N - 1)$$ (2.4)

- Суммарная трудоёмкость внутренних циклов, количество итераций ко-

торых меняется в промежутке $[1..N-1]$, которая равна (2.5):

$$f_{inner} = 5(N-1) + \frac{2 \cdot (N-1)}{2} \cdot (3 + f_{if}) \tag{2.5}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.6}$$

Трудоёмкость в лучшем случае (2.7):

$$f_{best} = -3 + \frac{3}{2}N \approx \frac{3}{2}N = O(N) \tag{2.7}$$

Трудоёмкость в худшем случае (2.8):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2) \tag{2.8}$$

2.3.2 Алгоритм гномьей сортировки

Трудоёмкость в лучшем случае (при уже отсортированном массиве) (2.9):

$$f_{best} = 1 + N(4 + 1) = 5N + 1 = O(N) \tag{2.9}$$

Трудоёмкость в худшем случае (при массиве, отсортированном в обратном порядке) (2.10):

$$f_{worst} = 1 + N(4 + (N-1) * (7 + 1 + 2)) = 10N^2 - 6N + 1 = O(N^2) \tag{2.10}$$

2.3.3 Алгоритм сортировки вставками

Трудоемкость данного алгоритма может быть рассчитана с использованием той же модели подсчета трудоемкости.

Трудоемкость алгоритма сортировки вставками:

- в лучщем случае - $O(N)$;

- в худшем случае - $O(N^2)$.

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Также для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

3.1 Требования к ПО

Ряд требований к программе:

- на вход подается массив целых чисел
- отсортированный массив, который был задан в предыдущем пункте

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*[4]. Выбор обсуловлен желанием попрактиковать свои умения работы с даным ЯП. Также *Python* предоставляет широкий выбор библиотек для комфортной работы.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time*[5].

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.py* - файл, содержащий весь служебный код;
- *sorts.py* - файл, содержащий код всех сортировок.

3.4 Листинги кода

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов сортировок (перемешиванием, вставками и гномьей).

Листинг 3.1 – Алгоритм сортировки вставками

```
1     def insertion_sort(arr, n):
2
3     for i in range(1, n):
4         j = i - 1
5         tmp = arr[i]
6
7         while (j >= 0 and arr[j] > tmp):
8             arr[j + 1] = arr[j]
9             j -= 1
10
11        arr[j + 1] = tmp
12
13    return arr
```

Листинг 3.2 – Алгоритм сортировки перемешиванием

```
1     def shaker_sort(arr, n):
2
3     left = 0
4     right = n - 1
5
6     swapped = True
7
8     while(swapped):
9         swapped = False
10
11        for i in range(left, right):
12            if (arr[i] > arr[i + 1]):
13                tmp = arr[i]
14                arr[i] = arr[i + 1]
15                arr[i + 1] = tmp
16
17                swapped = True
18
19        if (swapped == False):
20            break
21
22        swapped = False
23        right -= 1
24
```

```
25          for i in range(right - 1, left - 1, -1):
26              if (arr[i] > arr[i + 1]):
27                  tmp = arr[i]
28                  arr[i] = arr[i + 1]
29                  arr[i + 1] = tmp
30
31                  swapped = True
32
33          left += 1
34
35      return arr
```

Листинг 3.3 – Алгоритм гномьей сортировки

```
1       def gnomme_sort(arr, n):
2
3       i = 1
4
5       while (i < n):
6           if (arr[i] < arr[i - 1]):
7               tmp = arr[i]
8               arr[i] = arr[i - 1]
9               arr[i - 1] = tmp
10
11              if (i > 1):
12                  i -= 1
13          else:
14              i += 1
15
16      return arr
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты *для всех сортировок* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[9, 7, -5, 1, 4]$	$[-5, 1, 4, 7, 9]$	$[-5, 1, 4, 7, 9]$
$[5]$	$[5]$	$[5]$
$[]$	$[]$	$[]$

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого алгоритма была вычислена трудоемкость и оценены лучший и худший случаи.

4 Исследовательская часть

В данном разеделе будут приедены примеры работы программа, а также проведен сравнительный анализ адгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: Ubuntu 20.04.3 [6] Linux [7] x86_64.
- Память: 8 GiB.
- Процессор: Intel® Core™ i5-7300HQ CPU @ 2.50GHz [8].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.

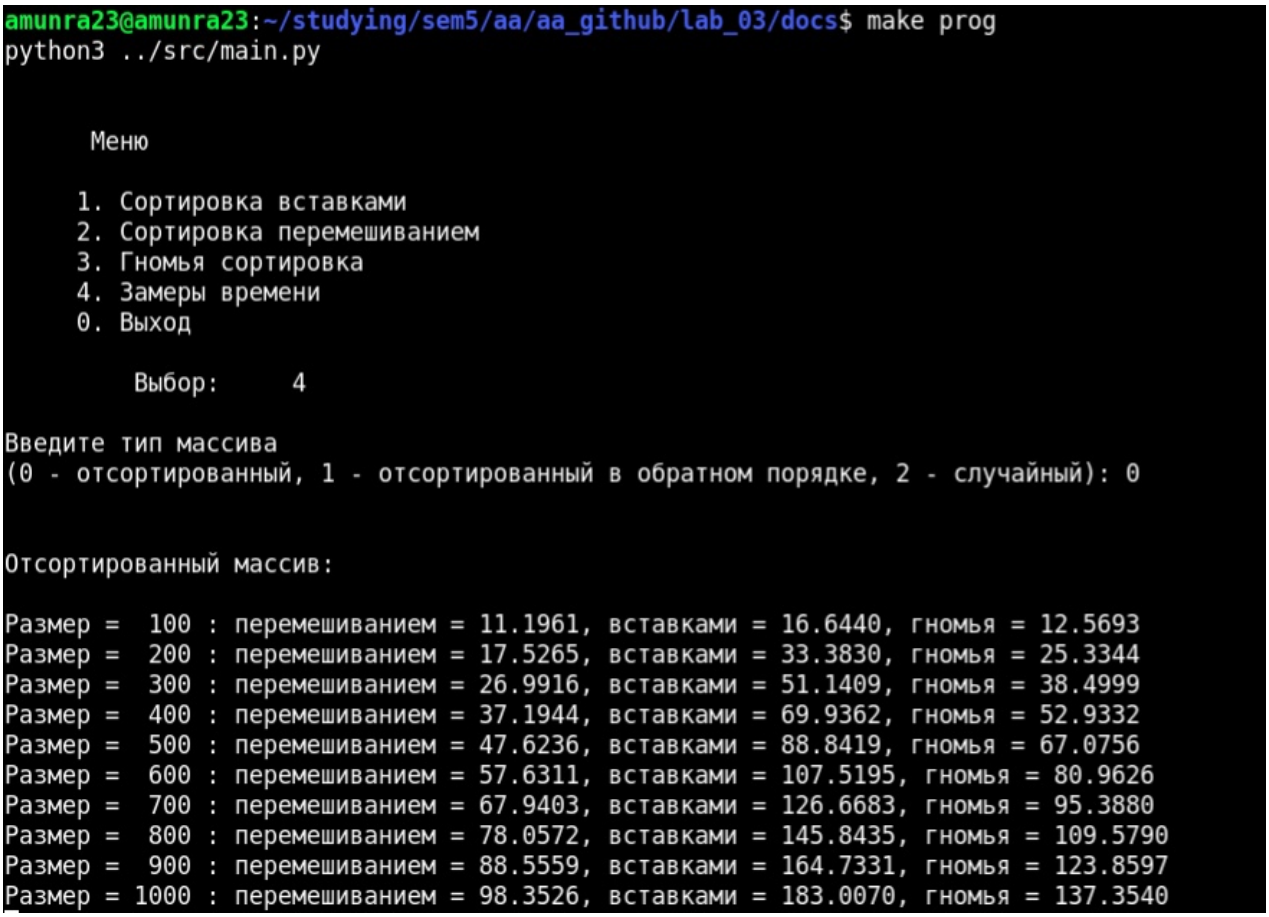

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени process_time(...) из библиотеки time на Python. Функция возвращает сумму системного и пользовательского процессорного времени типа float.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Результаты замеров времени работы алгоритмов сортировки на различных входных данных (в мс) приведены в таблицах 4.1, 4.2 и 4.3.

Таблица 4.1 – Отсортированные данные

Размер	Шейкером	Вставками	Гномья
100	0.0205	0.0346	0.0263
200	0.0368	0.0665	0.0507
300	0.0556	0.0648	0.0575
400	0.0474	0.0725	0.0546
500	0.0507	0.0931	0.0681
600	0.0600	0.1121	0.0820
700	0.0695	0.1305	0.0956
800	0.0801	0.1497	0.1148
900	0.0893	0.1671	0.1238
1000	0.1018	0.1844	0.1391

Таблица 4.2 – Отсортированные в обратном порядке данные

Размер	Шейкером	Вставками	Гномья
100	2.1406	0.7426	2.1195
200	4.3674	2.9388	7.7834
300	9.8322	6.6206	17.5885
400	18.0855	11.9534	31.8696
500	29.2288	18.9838	50.5128
600	43.0116	27.6467	73.6537
700	60.0487	37.8276	101.4856
800	78.2646	49.7082	133.4173
900	99.4545	62.8961	169.1879
1000	123.1786	77.9326	210.1529

Таблица 4.3 – Случайные данные

Размер	Шейкером	Вставками	Гномья
100	0.9844	0.3929	1.0544
200	2.6970	1.4714	3.8801
300	6.0501	3.3485	8.9118
400	10.6126	5.6650	15.1574
500	17.6962	9.3086	24.8351
600	25.4620	13.6112	36.4204
700	36.9290	19.6599	52.6627
800	47.1996	25.0265	67.0859
900	60.8530	32.3967	86.8291
1000	74.5834	40.1647	107.6520

Также на рисунках 4.2, 4.3, 4.4 приведены графические результаты замеров работы сортировок в зависимости от размера входного массива.

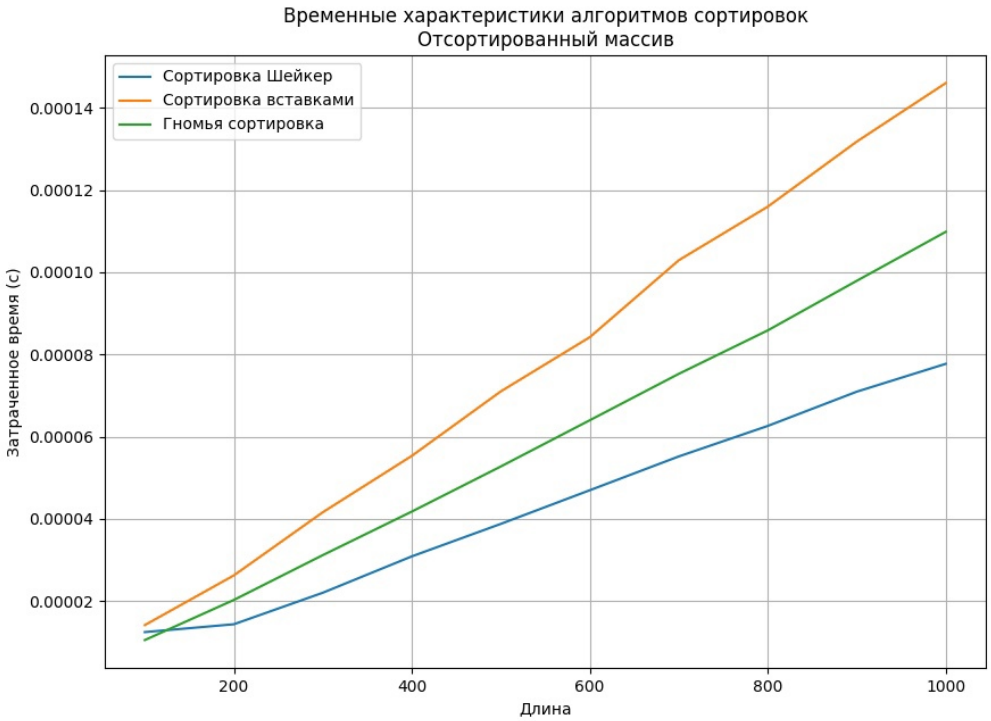

Рисунок 4.2 – Отсортированный массив

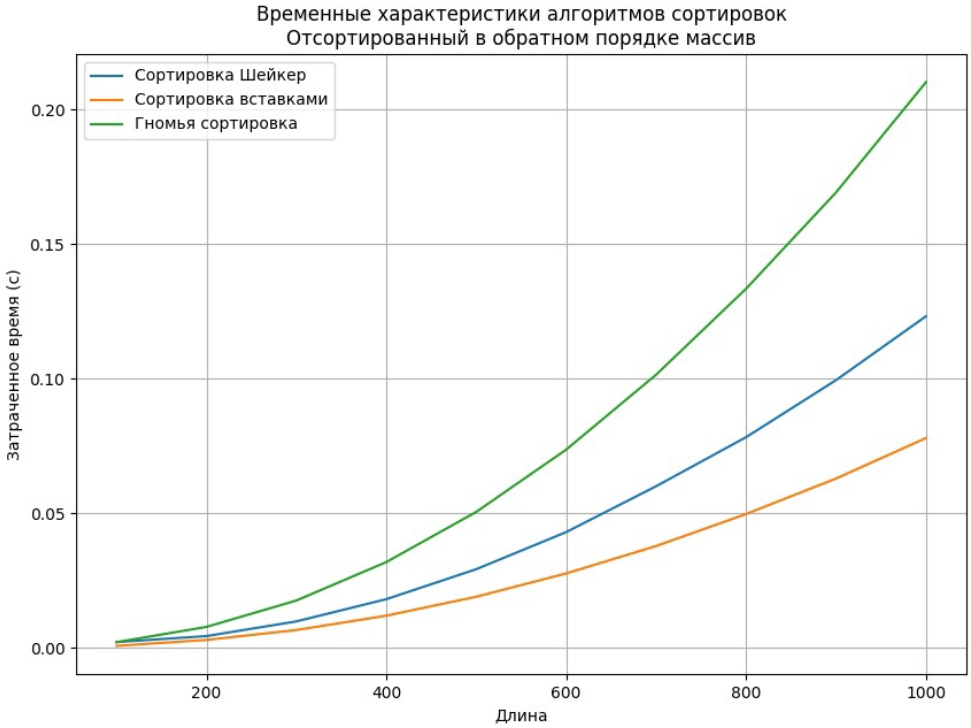

Рисунок 4.3 – Отсортированный в обратном порядке массив

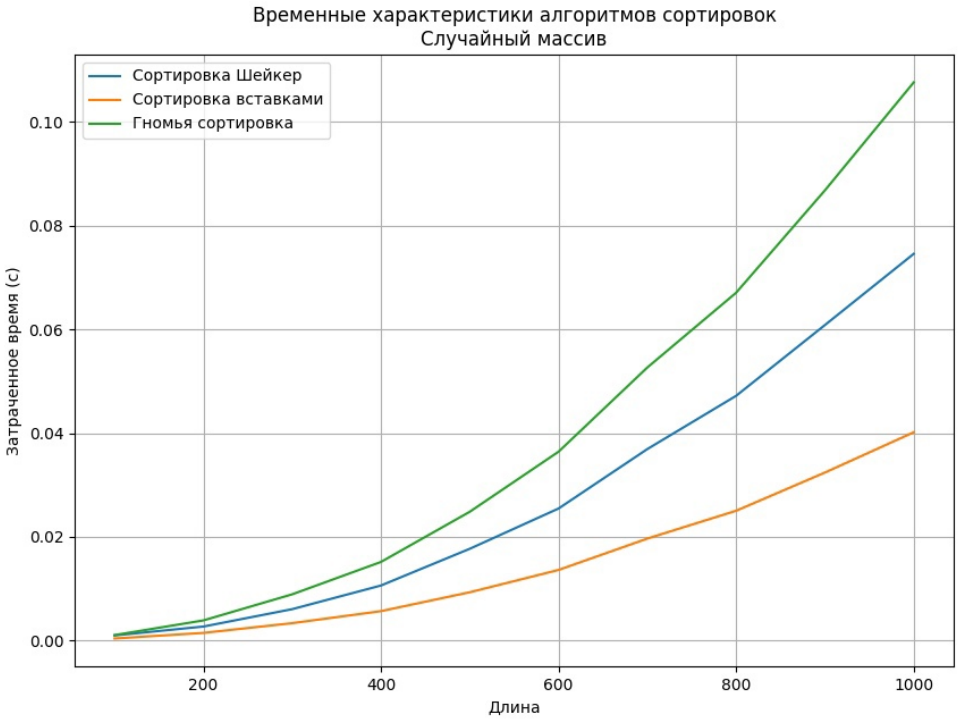

Рисунок 4.4 – Случайный массив

Вывод

Исходя из полученных результатов, гномья сортировка при случайно заполненном массиве, а также при обратно отсортированном работает дольше всех (примерно в 1.5 дольше, чем сортировка перемешиванием и в 2 раза дольше сортировки вставками), при этом сортировка методом вставок показала себя лучше всех. Что касается уже отсортированных данных, то лучше всего себя здесь показала сортировка перемешиванием, в то время, как метод вставок оказался худшим.

Заключение

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи:

- были изучены и реализованы алгоритмы сортировки: перемешиванием, вставками и гномья;
- была выбрана модель вычисления и проведен сравнительный анализ трудоемкостей выбранных алгоритмов сортировки;
- на основе экспериментальных данных проведено сравнение выбранных алгоритмов сортировки;
- подготовлен отчет о лабораторной работе.

Список литературы

[1] Сортировка вставками [Электронный ресурс]. Режим доступа: https://kvodo.ru/sortirovka-vstavkami-2.html (дата обращения: 4.10.2021).

[2] Шейкерная сортировка (перемешиванием) [Электронный ресурс]. Режим доступа: https://kvodo.ru/shaker-sort.html (дата обращения: 4.10.2021).

[3] лупая сортировка и некоторые другие, поумнее. [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/204968/ (дата обращения: 4.10.2021).

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 04.10.2021).

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 04.10.2021).

[6] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 04.10.2021).

[7] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 04.10.2021).

[8] Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 04.10.2021).