Содержание

Введение

Сортировка - процесс перегруппировки заданного множества объектов в некотором определенном порядке. Чтобы сделать данный процесс эффективным и менее ресурсозатратным, было придумано и реализовано множество различных алгоритмов сортировки, многие из них имеют свои плюсы и минусы, но все они состоят из трех основных шагов:

- сравнение элементов, задающее их порядок;
- обмен элементов в паре;
- сортирующий алгоритм, осуществляющий предыдущие два шага до полного упорядочивания.

Эффективность алгоритма зависит от скорости работы этого алгоритма. Скорость работы алгоритма сортировки определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных определенной длины, от этой длины.

Цель работы - получить навык сравнительного анализа алгоритмов сортировки.

Для решения поставленной цели требуется решить следующие задачи:

- изучить три алгоритма сортировки: Гномья, перемешиванием и Шелла;
- разработать и реализовать указанные алгоритмы;
- протестировать реализацию рассматриваемых алгоритмов;
- провести сравнительный анализ реализованных алгоритмов по затраченному процессорному времени и по трудоемкости;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут описаны следующие алгоритмы сортировок: гномьей, перемешиванием и Шелла.

1.1 Гномья сортировка

Гномья сортировка [3] - алгоритм сортировки, который использует только один цикл, что является достаточно редким явлением. Данный алгоритм выполняет следующие действия:

- сравниваются текущий и предыдущий элементы последовательности;
- если они расположены в необходимом порядке, то осуществляется переход к следующему элементу;
- иначе происходит обмен. Если предыдущий элемент не был первым, осуществляется переход на один элемент назад.

Шаги повторяются, пока возможен переход к следующему элементу.

1.2 Сортировка перемешиванием

Сортировка выбором [3] - сортировка, которая является модификацией пузырьковой сортировки и состоит она из следующих шагов:

- если были обмены во время предыдущего прохода, совершаем проход по массиву;
- сравнивается два элемента последовательности;
- если элементы нарушают порядок, меняем их местами;
- сдвигаем левую границу вправо на одну позицию;

Шаги повторяются, пока в предыдущем проходе будут замены. При этом каждый раз элементы массива перебираются в разных направлениях (перебрав элементы слева-направо, новый перебор начинается в обратном порядке).

1.3 Сортировка Шелла

Сортировка Шелла [3] является усовершенствованием сортировки вставками. В сортировке вставками на каждом шаге, берут элемент входной последовательности и передают в готовую последовательность, вставляя его на подходящее место. Д. Л. Шелл предложил следующие шаги:

- выбирается некоторое расстояние d между элементами последовательности;
- сравниваются и сортируются значения, стоящие друг от друга на расстоянии d;
- шаг 2 повторяется для меньших значений d, не равных 1;
- при d, равном 1, элементы упорядочиваются сортировкой вставками.

Приемлема любая последовательность для d, с условием, что последнее значение равно 1.

Вывод

Были рассмотрены следующие алгоритмы сортировки: гномья, перемешиванием и Шелла. Для указанных алгоритмов необходимо получить теоретическую оценку и доказать её экспериментально.

2 Конструкторская часть

В данном разделе будут указаны требования к программному обеспечению и представлены схемы следующих алгоритмов сортировки: гномьей, перемешиванием и Шелла.

2.1 Требования к ПО

К программе представлен ряд требований:

- на вход подается массив целых чисел в диапазоне [-1000;1000];
- на выходе - отсортированный массив, поданный на вход. При сортировке изменяется изначальный массив, а не его копия.

2.2 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов гномьей сортировки, сортировки перемешиванием и сортировки Шелла.

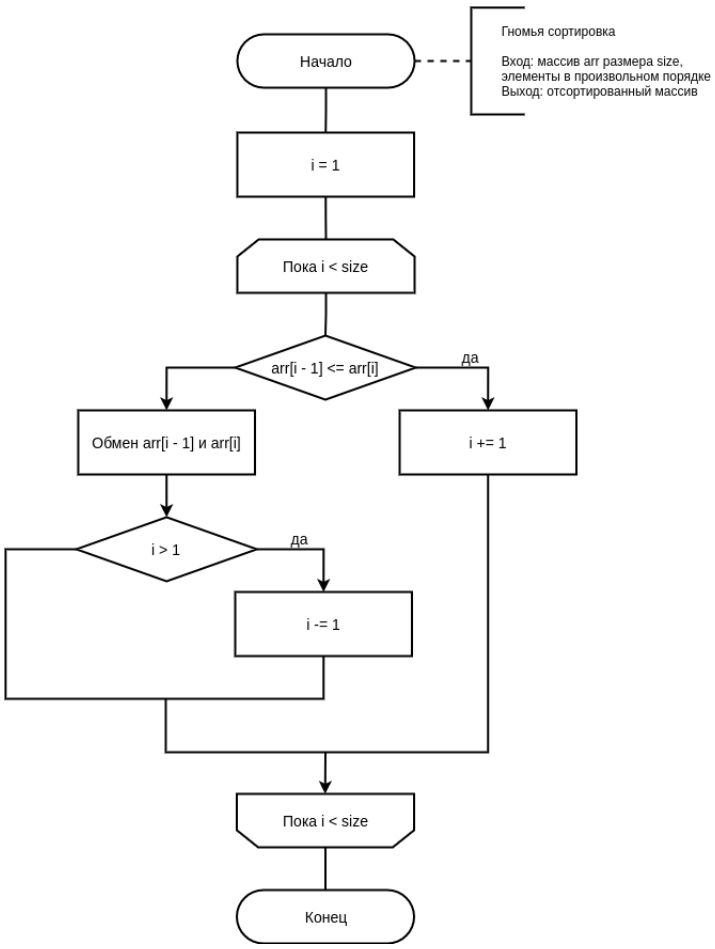

Рисунок 2.1 – Схема алгоритма гномьей сортировки

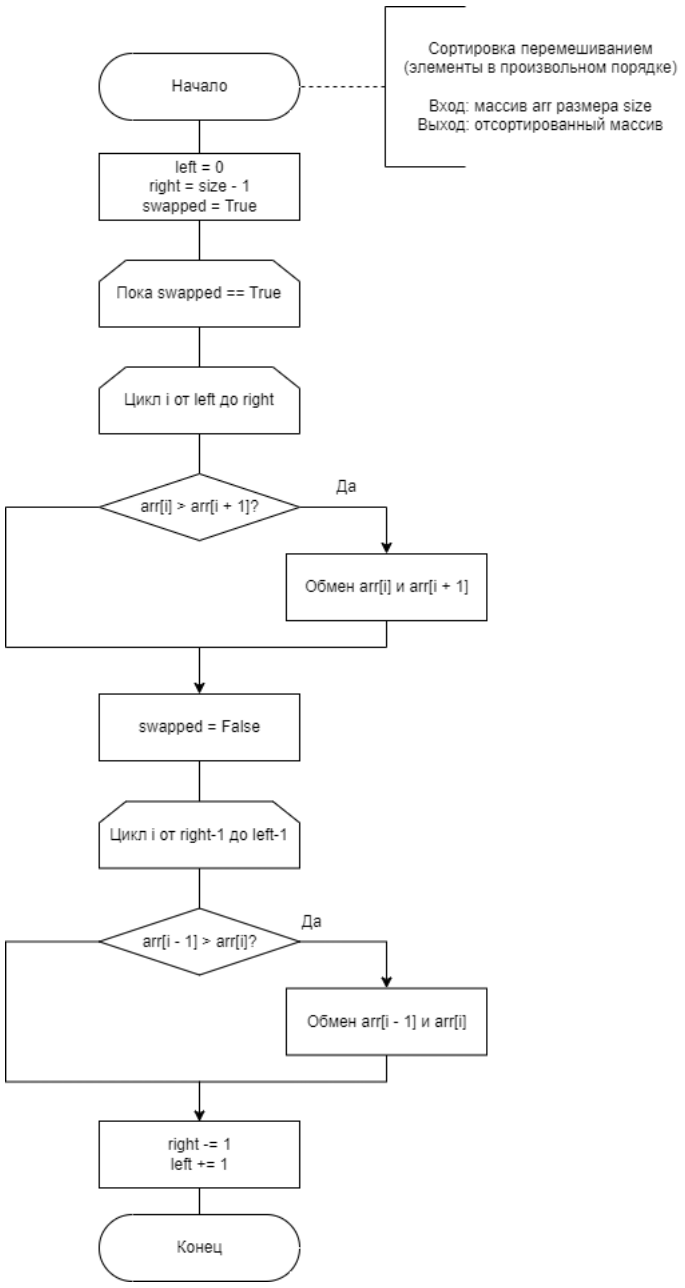

Рисунок 2.2 – Схема алгоритма сортировки перемешиванием

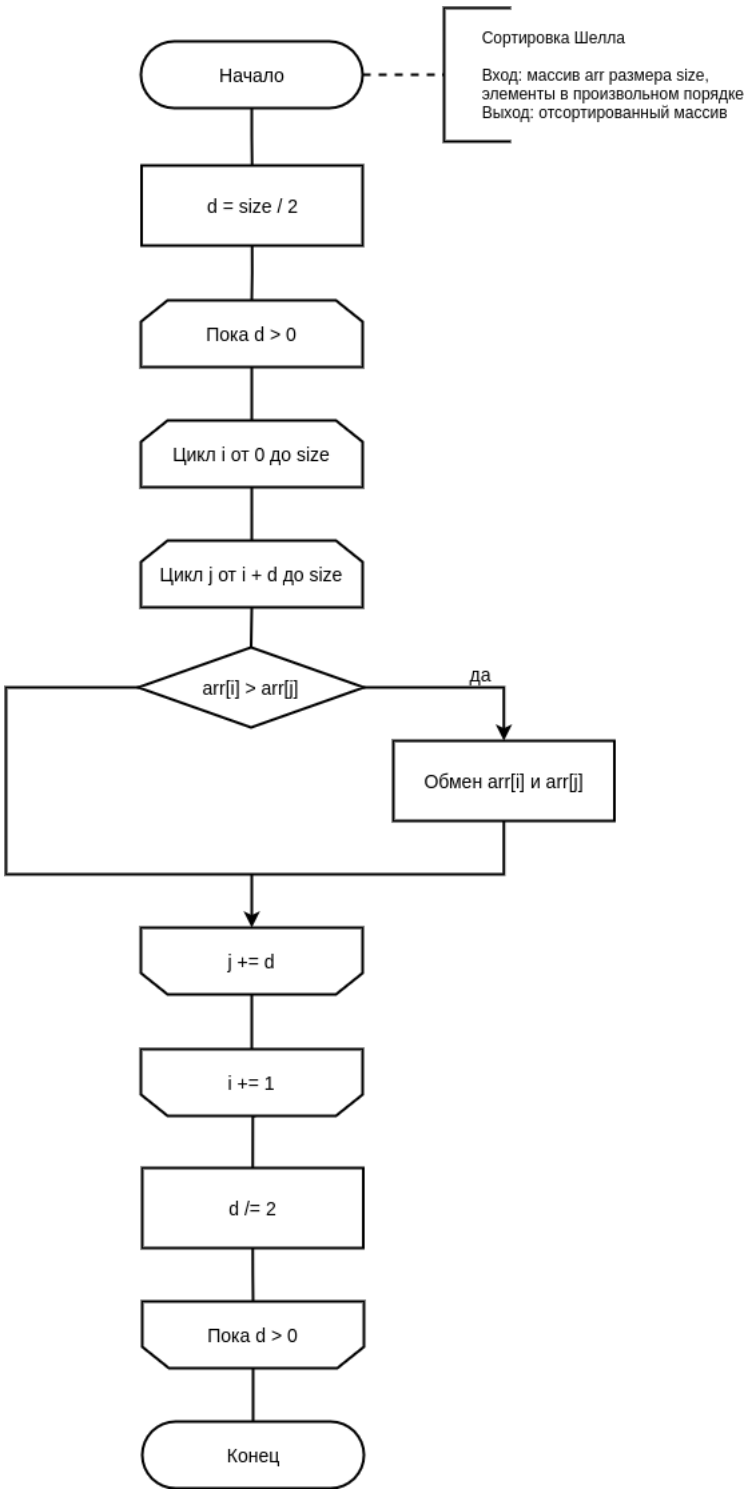

Рисунок 2.3 – Схема алгоритма сортировки Шелла

Вывод

Были представлены схемы следующих алгоритмов сортировки: гномьей, перемешиванием и Шелла. А также указаны требования к ПО.

3 Технологическая часть

В данном разделе будут указаны средства реализации, будут представлены листинги кода, а также функциональные тесты.

3.1 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования Python [1]. Выбор ЯП обусловлен простотой синтаксиса, большим числом библиотек и эффективностью визуализации данных.

Замеры времени проводились при помощи функции process_time из библиотеки time [2].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
- sort.py - файл, содержащий функции сортировок массива;
- arr.py - файл, содержащий функции создания массива различного типа и работы с массивом;
- time_test.py - файл, содержащий функции замеров времени работы сортировок;
- graph_result.py - файл, содержащий функции визуализации временных характеристик алгоритмов сортировок.

3.3 Листинги кода

Реализации алгоритмов сортировок выбором, Шелла и гномьей представлены на листингах 3.1, 3.2, 3.3.

Листинг 3.1 – Алгоритм гномьей сортировки

```
def gnome_sort(arr, size):
    i = 1

    while i < size:
        if arr[i - 1] <= arr[i]:
            i += 1
        else:
            arr[i - 1], arr[i] = arr[i], arr[i - 1]

            if i > 1:
                i -= 1
```

Листинг 3.2 – Алгоритм сортировки перемешиванием

```
def shaker_sort(arr, size):
    left, right = 0, size - 1
    swapped = True

    while swapped == True:
        for i in range(left, right):
            if arr[i] > arr[i + 1]:
            arr[i], arr[i + 1] = arr[i + 1], arr[i]
            swapped = True

    if not swapped:
    break

    swapped = False
    for i in range(right - 1, left - 1, -1):
        if arr[i] > arr[i + 1]:
            arr[i], arr[i + 1] = arr[i + 1], arr[i]
            swapped = True

    right -= 1
    left += 1
```

Листинг 3.3 – Алгоритм сортировки Шелла

```
def shell_sort(arr, size):
    d = size // 2
```

```
3
4       while d > 0:
5           for i in range(size):
6               for j in range(i + d, size, d):
7                   if arr[i] > arr[j]:
8                       arr[i], arr[j] = arr[j], arr[i]
9           d //= 2
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы сортировок. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[-2, 5, 0, 2, -5]$	$[-5, -2, 0, 2, 5]$	$[-5, -2, 0, 2, 5]$
$[1, 2]$	$[1, 2]$	$[1, 2]$
$[1]$	$[1]$	$[1]$
$[]$	$[]$	$[]$

Вывод

Были реализованы функции алгоритмов следующих сортировок: гномьей, перемешиванием и Шелла. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будут проведены сравнительный анализ реализованных алгоритмов сортировки по затраченному процессорному времени и сравнительный анализ трудоемкости алгоритмов.

4.1 Модель вычислений для оценки трудоёмкости алгоритмов

Для определения трудоемкости алгоритмов необходимо ввести модель вычислений:

1. операции из списка (4.1) имеют трудоемкость равную 1;

$$+, -, /, *, \%, =, +=, -=, *=, /=, \%=, ==, !=, <, >, <=, >=, [], ++, -- \tag{4.1}$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (4.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{4.2}$$

3. трудоемкость цикла рассчитывается, как (4.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{4.3}$$

4. трудоемкость вызова функции равна 0.

4.2 Трудоёмкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости.

4.2.1 Алгоритм гномьей сортировки

Трудоёмкость в лучшем случае (при уже отсортированном массиве) (4.4):

$$f_{best} = 1 + N(4 + 1) = 5N + 1 = O(N) \tag{4.4}$$

Трудоёмкость в худшем случае (при массиве, отсортированном в обратном порядке) (4.5):

$$f_{worst} = 1 + N(4 + (N - 1) * (7 + 2)) = 9N^2 - 5N + 1 = O(N^2) \tag{4.5}$$

4.2.2 Алгоритм сортировки перемешиванием

- Трудоёмкость сравнения внешнего цикла $while(swapped == True)$, которая равна (4.6):

$$f_{outer} = 1 + 2 \cdot (N - 1) \tag{4.6}$$

- Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - 1]$, которая равна (4.7):

$$f_{inner} = 5(N - 1) + \frac{2 \cdot (N - 1)}{2} \cdot (3 + f_{if}) \tag{4.7}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (4.8):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{4.8}$$

Трудоёмкость в лучшем случае (4.9):

$$f_{best} = -3 + \frac{3}{2}N \approx \frac{3}{2}N = O(N) \tag{4.9}$$

Трудоёмкость в худшем случае (4.10):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2) \tag{4.10}$$

4.2.3 Алгоритм сортировки Шелла

Трудоемкость в лучшем случае (при отсортированном массиве). Выведена
формуле (4.11).

$$\begin{gathered} f_{best} = 3 + 4 + \frac{N}{4} \cdot (3 + 2 + log(N) \cdot (2 + 4 + 4)) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} = O(N \cdot log(N)) \end{gathered} \tag{4.11}$$

Трудоемкость в худшем случае (при отсортированном массиве в обратном порядке). Выведена формуле (4.12).

$$\begin{gathered} f_{worst} = 7 + \frac{N}{4} \cdot (5 + log(N) \cdot (10 + log(N) \cdot (6 + 4))) = \\ = 7 + \frac{5N}{4} + \frac{5N \cdot log(N)}{2} + \frac{5N \cdot log^2(N)}{2} = O(N \cdot log^2(N)) \end{gathered} \tag{4.12}$$

Вывод

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

4.3 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Процессор: Intel Core i5-10300H 2.50 ГГц;
- Оперативная память: 8 ГБайт;

- Операционная система: Windows 11.

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

4.4 Демонстрация работы программы

На рисунках 4.1, 4.2, 4.3 приведены примеры работы алгоритмов.

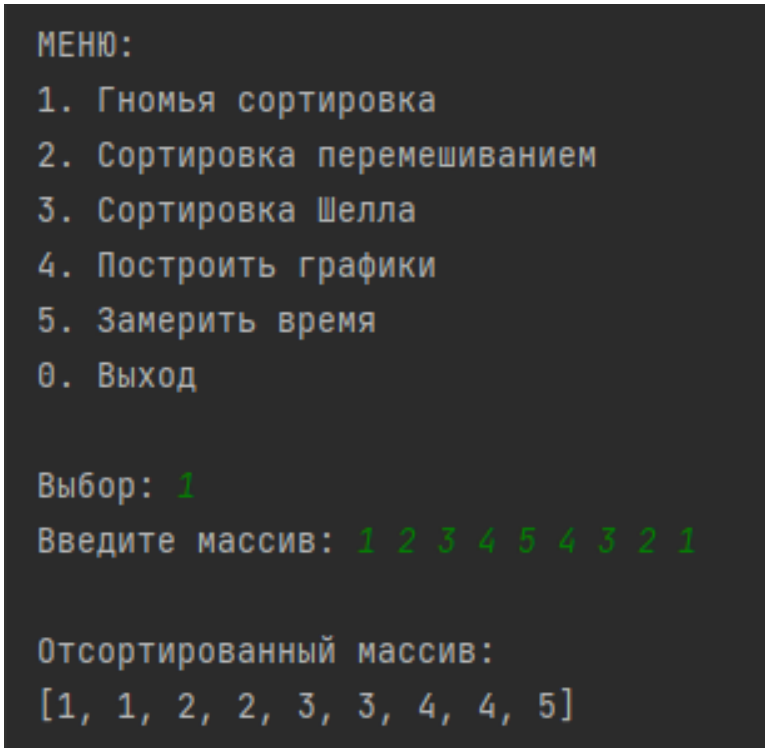

Рисунок 4.1 – Пример работы гномьей сортировки

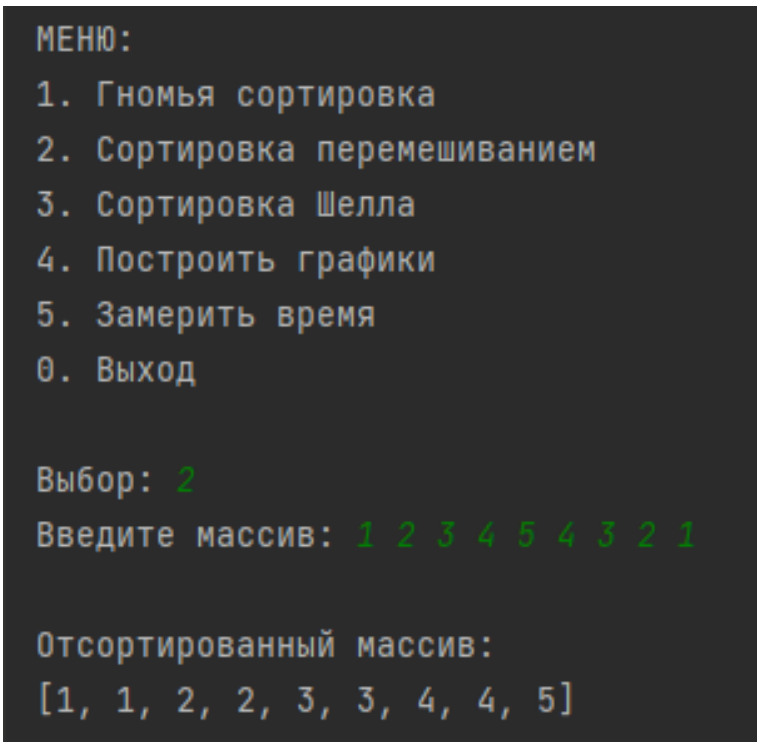

Рисунок 4.2 – Пример работы сортировки перемешиванием

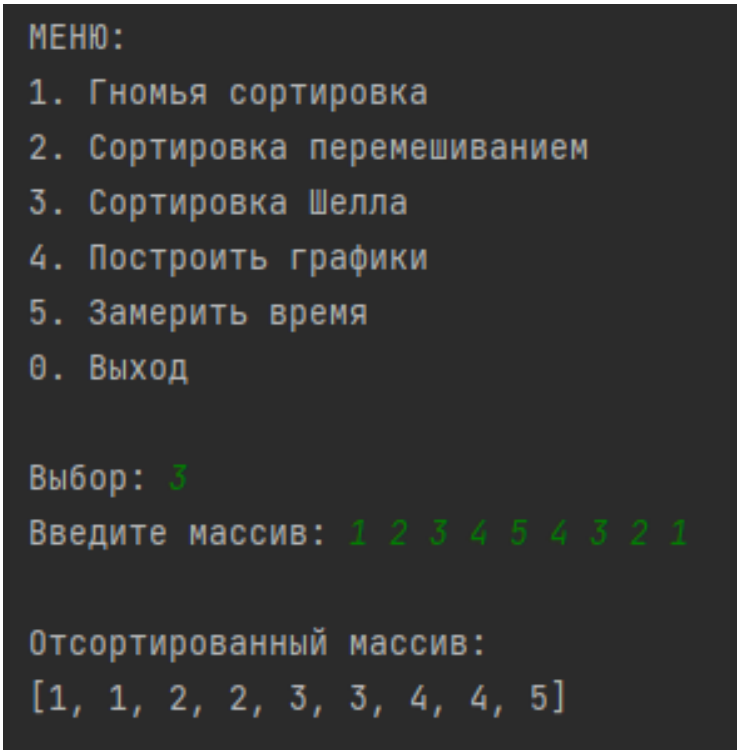

Рисунок 4.3 – Пример работы сортировки Шелла

4.5 Время выполнения алгоритмов

Функция process_time из библиотеки time ЯП Python возвращает процессорное время в секундах - значение типа float.

Для замера времени:

- получить значение времени до начала сортировки, затем после её окончания. Чтобы получить результат, необходимо вычесть из второго значения первое;
- первый шаг необходимо повторить iters раз (в программе iters равно 1500), суммируя полученные значения, а затем усреднить результат.

Результаты замеров времени работы алгоритмов в миллисекундах приведены в таблицах 4.1, 4.2, 4.3.

Таблица 4.1 – На входе отсортированный массив

Размер	Гномья	Перемешиванием	Шелла
100	0.0208	0.0000	0.3542
200	0.0104	0.0208	1.2708
300	0.0208	0.0312	3.4271
400	0.0312	0.0312	5.3229
500	0.0312	0.0312	8.0833
600	0.0417	0.0729	14.7292
700	0.0625	0.0833	19.0208
800	0.0625	0.0938	21.4062
900	0.0938	0.0938	26.9375
1000	0.0729	0.1146	33.1042

Таблица 4.2 – На входе отсортированный в обратном порядке массив

Размер	Гномья	Перемешиванием	Шелла
100	1.0729	0.5521	0.4167
200	3.7083	2.1667	1.5000
300	8.6979	5.0521	4.0104
400	15.6042	9.1562	6.2812
500	26.1667	14.5417	9.3646
600	37.6875	22.5312	20.5625
700	47.6562	30.2708	25.7083
800	70.4479	39.1667	24.7188
900	86.3229	61.4896	30.4167
1000	106.7708	80.0000	37.1250

Таблица 4.3 – На входе случайный массив

Размер	Гномья	Перемешиванием	Шелла
100	0.5625	0.1042	0.0833
200	2.0104	0.4688	0.3958
300	5.5625	1.1042	1.2917
400	10.1354	2.0729	1.8854
500	14.6042	3.4896	2.6562
600	24.6042	5.1562	5.4062
700	24.4062	6.8958	7.0000
800	14.8229	9.9896	8.4896
900	19.0521	12.6354	9.9688
1000	23.0625	15.1250	12.6875

На рисунках 4.4, 4.5, 4.6 приведены графические результаты замеров времени работы сортировок от длины входного массива в трех случаях: на входе отсортированный массив, отсортированный в обратном порядке и массив, заполненный случайным образом.

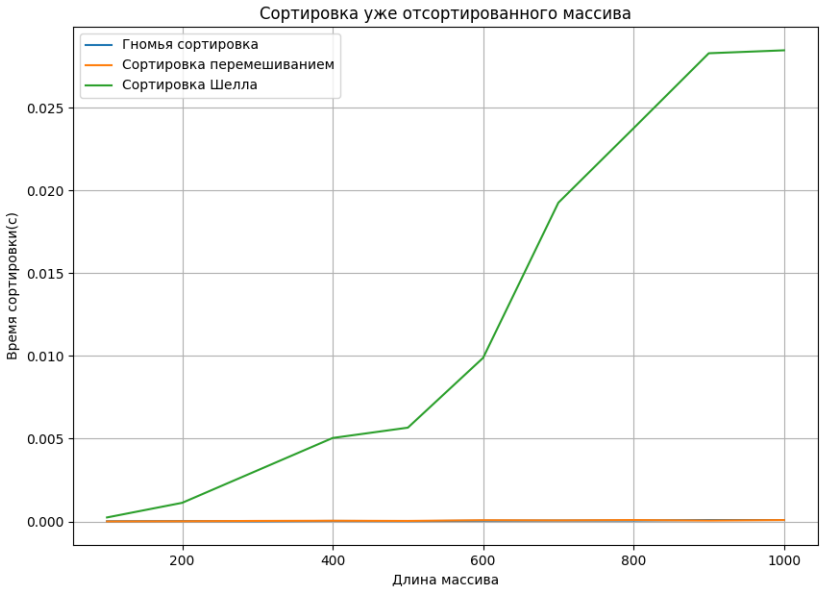

Рисунок 4.4 – На входе отсортированный массив

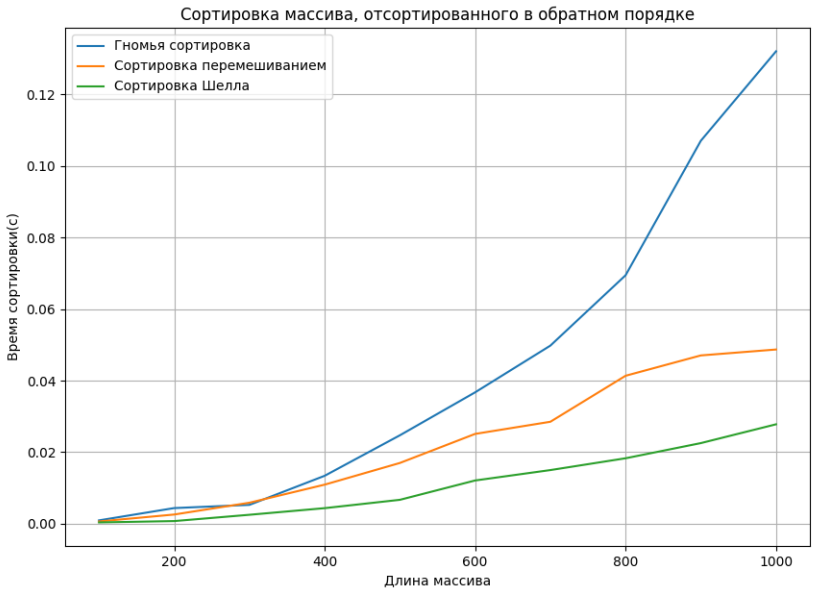

Рисунок 4.5 – На входе отсортированный в обратном порядке массив

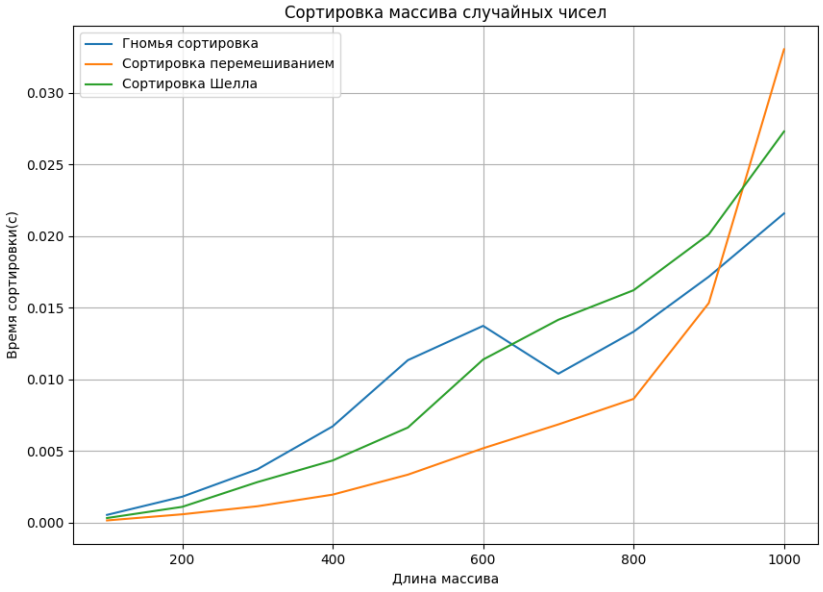

Рисунок 4.6 – На входе заполненный случайно массив

Вывод

Как мы можем заметить, на отсортированном массиве быстрее всего работает Гномья сортировка, а сортировка Шелла работает очень медленно, при этом на случайных данных и отсортированном в обратном порядке массиве сортировка Шелла показывает лучший результат среди представленных алгоритмов.

Результаты временных замеров совпадают с вычисленными трудоемкостями.

Заключение

В ходе лабораторной работы были решены следующие задачи:

- были изучены три алгоритма сортировки: гномья, перемешиванием, Шелла;
- были разработаны и реализованы указанные алгоритмы;
- была протестирована реализация рассматриваемых алгоритмов;
- был проведен сравнительный анализ реализованных алгоритмов по затраченному процессорному времени, по трудоемкости и по памяти;
- описаны полученные результаты в отчете о выполненной лабораторной работе.

Таким образом, поставленная цель достигнута.

Литература

[1] Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 01.11.2023).

[2] time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 01.11.2023);

[3] Книга — "Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие"Д.В. Шагбазян, А.А. Штанюк, Е.В. Малкина (дата обращения: 01.11.2023).