



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 Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Сортировка выбором	4
1.2 Сортировка Шелла	4
1.3 Гномья сортировка	5
2 Конструкторская часть	6
2.1 Требования к ПО	6
2.2 Разработка алгоритмов	6
3 Технологическая часть	10
3.1 Средства реализации	10
3.2 Сведения о модулях программы	10
3.3 Листинги кода	10
3.4 Функциональные тесты	12
4 Исследовательская часть	13
4.1 Модель вычислений для оценки трудоёмкости алгоритмов .	13
4.2 Трудоёмкость алгоритмов	13
4.2.1 Алгоритм сортировки выбором	14
4.2.2 Алгоритм гномьей сортировки	14
4.2.3 Алгоритм сортировки Шелла	14
4.3 Технические характеристики	15
4.4 Демонстрация работы программы	16
4.5 Время выполнения алгоритмов	16
Заключение	21
Список литературы	22

Введение

При решении различных задач встает необходимость работы с упорядоченным набором данных. Например, при поиске элемента в заданном множестве. Для упорядочивания последовательности значений используется сортировка.

Сортировка - процесс перегруппировки заданного множества объектов в некотором определенном порядке [1]. Для реализации этого процесса разрабатываются алгоритмы сортировки. Такие алгоритмы 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.

Существует большое количество алгоритмов сортировки. Все они решают одну и ту же задачу, причем некоторые алгоритмы имеют преимущества перед другими. Поэтому существует необходимость сравнительного анализа алгоритмов сортировки.

Цель работы - получить навык сравнительного анализа алгоритмов сортировки.

Для решения поставленной цели требуется решить следующие задачи:

- изучить три алгоритма сортировки: выбором, Шелла и гномью;
- разработать и реализовать указанные алгоритмы;
- протестировать реализацию рассматриваемых алгоритмов;
- провести сравнительный анализ реализованных алгоритмов по затраченному процессорному времени и по трудоемкости.

1 Аналитическая часть

В данном разделе будут описаны алгоритмы сортировки выбором, Шелла и гномьей сортировки.

1.1 Сортировка выбором

Сортировка выбором[1] состоит из следующих шагов:

- выбирается элемент неотсортированной части последовательности с наименьшим значением;
- выбранный элемент меняется местами с элементом, стоящим на первой позиции в неотсортированной части. Обмен не нужен, если это и есть минимальный элемент;
- повтор шагов 1 и 2 до тех пор, пока не останется только наибольший элемент.

1.2 Сортировка Шелла

Сортировка Шелла[1] 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.

1.3 Гномья сортировка

Гномья сортировка [2] 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.

Вывод

Были рассмотрены следующие алгоритмы сортировки: выбором, Шелла и гномья. 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 алгоритмов сортировки выбором, Шеллом и гномьей сортировки.

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 сортировки выбором, Шелла и гномьей сортировки.

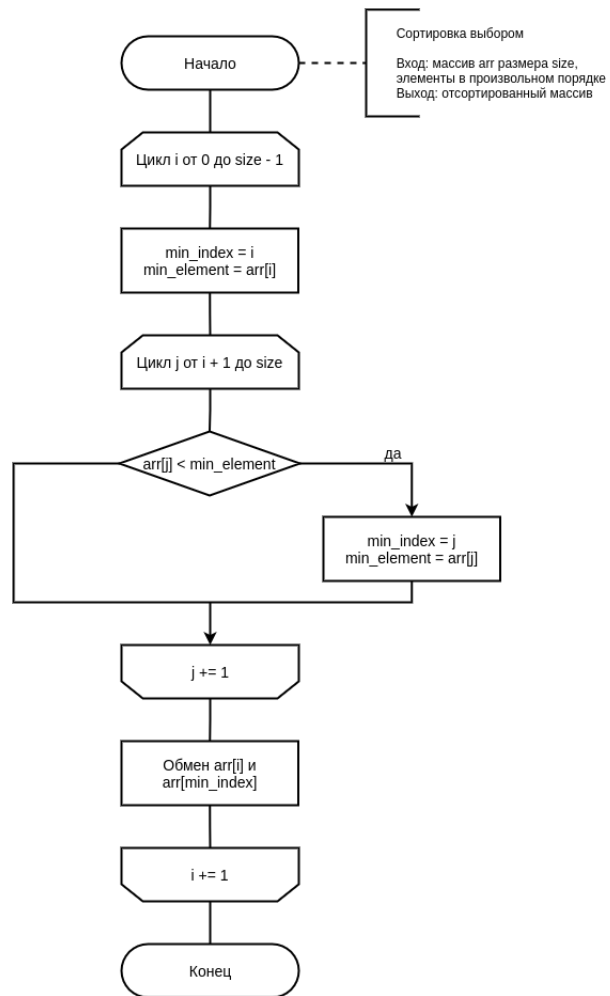


Рисунок 2.1 – Схема алгоритма сортировки выбором

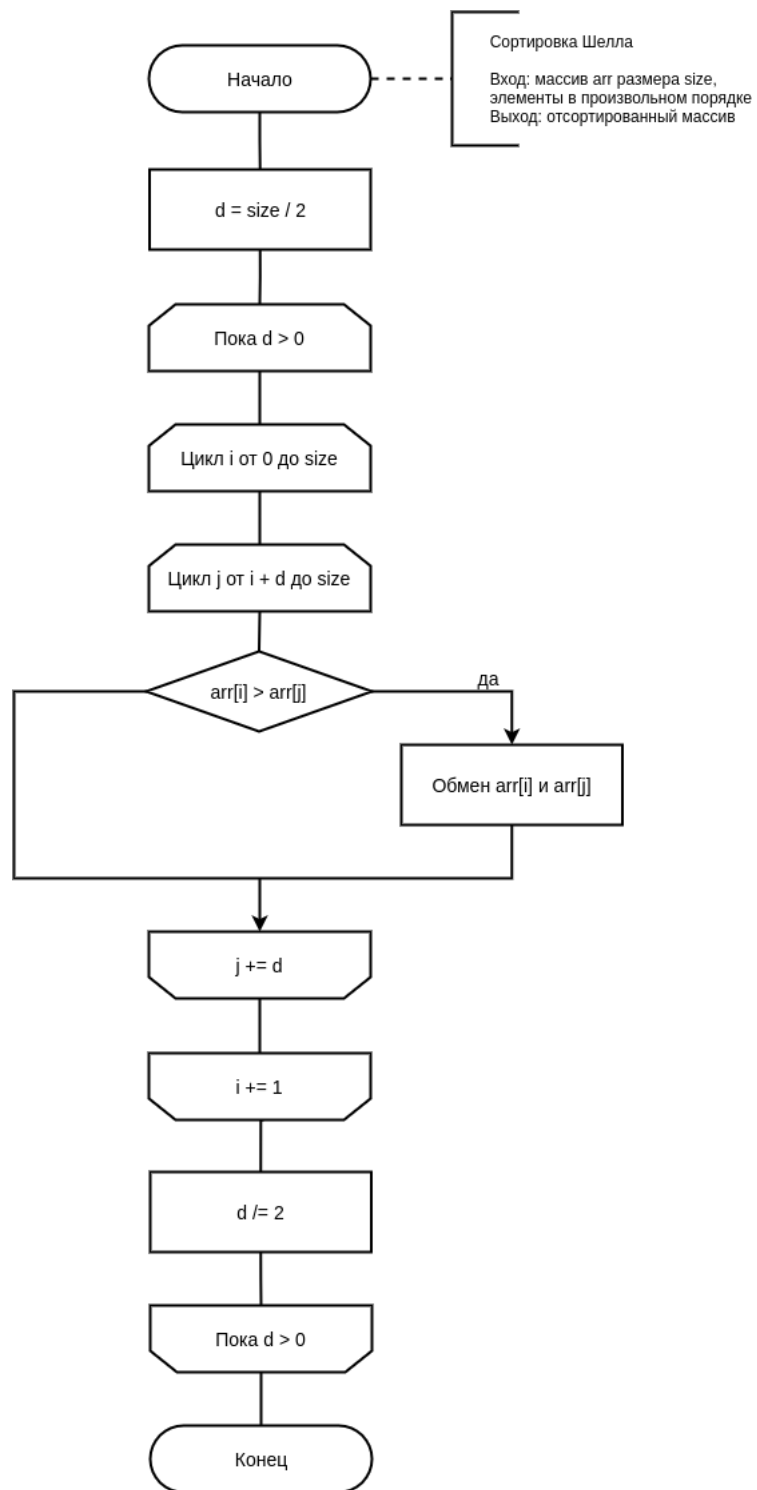


Рисунок 2.2 – Схема алгоритма сортировки Шелла

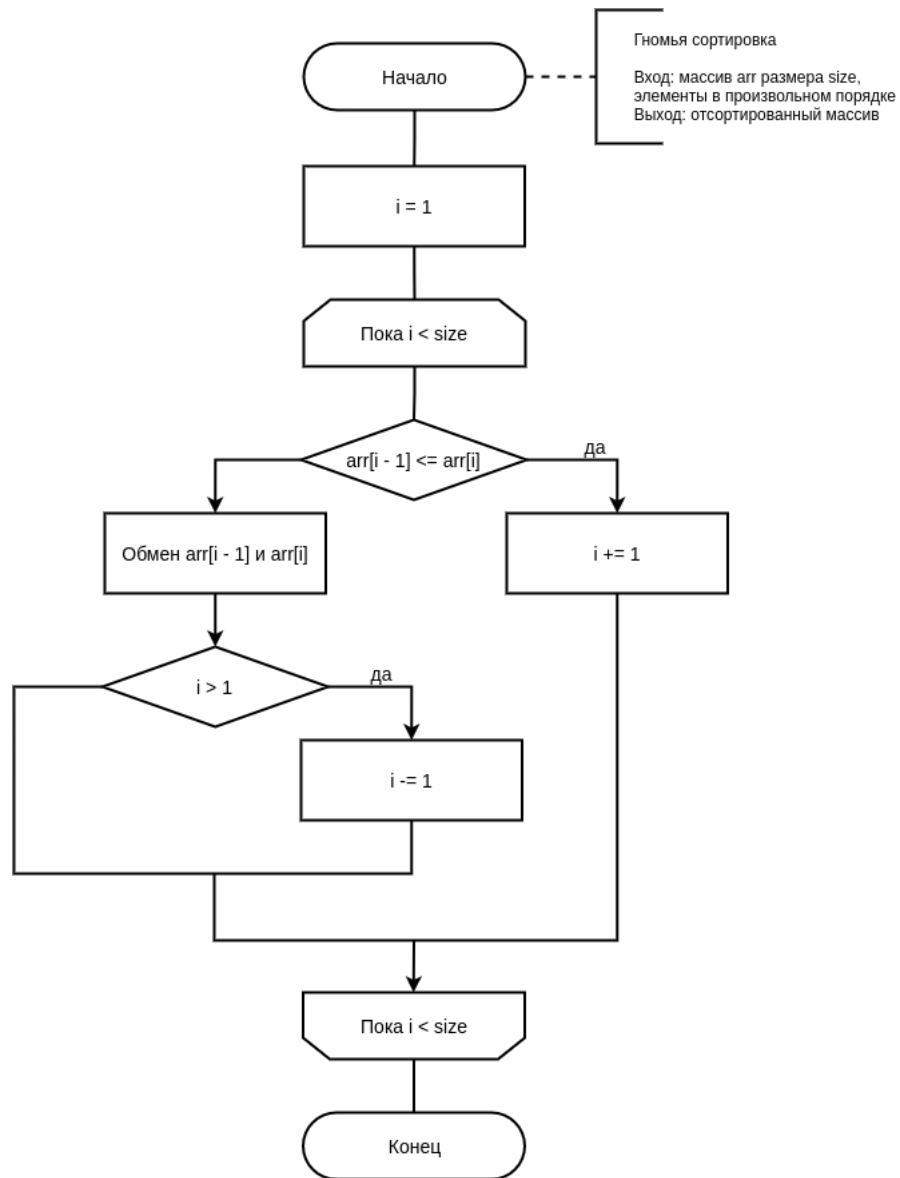


Рисунок 2.3 – Схема алгоритма гномьей сортировки

Вывод

Были представлены схемы алгоритмов сортировки выбором, Шелла и гномьей сортировки и 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 [3]. Выбор ЯП обусловлен простотой синтаксиса, большим числом библиотек и эффективностью визуализации данных.

Замеры времени проводились при помощи функции `process_time` из библиотеки `time` [4].

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 сортировки выбором

```
1 def selection_sort(arr, size):
2     for i in range(size - 1):
3         min_element = arr[i]
4         min_index = i
5
6         for j in range(i + 1, size):
7             if arr[j] < min_element:
8                 min_element = arr[j]
9                 min_index = j
10
11         arr[i], arr[min_index] = arr[min_index], arr[i]
12
13     return arr
```

Листинг 3.2 – Алгоритм сортировки Шелла

```
1 def shell_sort(arr, size):
2     d = size // 2
3
4     while d > 0:
5         for i in range(0, 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
10
11     return arr
```

Листинг 3.3 – Алгоритм гномьей сортировки

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

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, 6]	[1, 2, 3, 4, 5, 6]	[1, 2, 3, 4, 5, 6]
[5, 4, 3, 2, 1]	[1, 2, 3, 4, 5]	[1, 2, 3, 4, 5]
[0, -5, 3, 7, -8]	[-8, -5, 0, 3, 7]	[-8, -5, 0, 3, 7]
[17]	[17]	[17]
[]	[]	[]

Вывод

Были реализованы функции алгоритмов сортировки выбором, Шелла и гномьей. 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 [5]:

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

$$+, -, /, *, \%, =, + =, - =, * =, / =, \% =, ==, !=, <, >, <=, >=, [], ++, -- \quad (4.1)$$

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

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

3. трудоемкость цикла рассчитывается, как (4.3);

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

4. трудоемкость вызова функции равна 0.

4.2 Трудоёмкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости.

4.2.1 Алгоритм сортировки выбором

Трудоемкость в лучшем случае (4.2.1):

$$\begin{aligned} f_{best} &= 1 + (N - 1)(3 + 3 + 1 + (N/2)2 + 7) = \\ &= N^2 + 14N - N - 14 + 1 = N^2 + 13N - 13 = O(N^2) \end{aligned} \quad (4.4)$$

Трудоёмкость в худшем случае (4.2.1):

$$\begin{aligned} f_{worst} &= 1 + (N - 1)(3 + 3 + 1 + (N/2)(2 + 3) + 7) = \\ 2.5N^2 + 14N - 2.5N - 14 + 1 &= 2.5N^2 + 11.5N - 13 = O(N^2) \end{aligned} \quad (4.5)$$

4.2.2 Алгоритм гномьей сортировки

Трудоемкость в лучшем случае (4.6):

$$f_{best} = 1 + N(4 + 1) = 5N + 1 = O(N) \quad (4.6)$$

Трудоёмкость в худшем случае (4.7):

$$f_{worst} = 1 + N(4 + (N - 1) * (7 + 2)) = 9N^2 - 5N + 1 = O(N^2) \quad (4.7)$$

4.2.3 Алгоритм сортировки Шелла

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

Трудоемкость алгоритма сортировки Шелла:

- в лучшем случае - $O(N \log^2 N)$;
- в худшем случае - $O(N^2)$.

4.3 Технические характеристики

Проведем сравнительный анализ алгоритмов сортировки по затраченному процессорному времени.

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система: Ubuntu 20.04.1 Linux x86_64 [6];
- память : 8 GiB;
- процессор: AMD® Ryzen™ 3 3200u @ 2.6 GHz [7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.4 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы.

```
МЕНЮ:  
1. Сортировка выбором  
2. Сортировка Шелла  
3. Гномья сортировка  
4. Построить графики  
5. Замерить время  
0. Выход  
Выбор: 3  
Введите размер массива: 7  
Введите элементы массива:  
-23  
9  
17  
0  
-4  
17  
9  
  
Отсортированный массив:  
[-23, -4, 0, 9, 9, 17, 17]
```

Рисунок 4.1 – Пример работы программы

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	Выбором	Шелла	Гномья
100	0.1872	0.4463	0.0107
200	0.6648	1.5769	0.0208
300	1.7652	4.2090	0.0315
400	2.9205	6.5499	0.0415
500	4.7658	9.9785	0.0527
600	6.9592	17.5923	0.0610
700	9.5135	23.0703	0.0695
800	12.3131	26.6032	0.0836
900	15.3724	32.2987	0.0905
1000	18.7373	39.5696	0.0996

Таблица 4.2 – На входе
отсортированный в обратном порядке
массив

Размер	Выбором	Шелла	Гномья
100	0.2276	0.4635	1.4289
200	0.8215	1.6202	5.4438
300	2.1278	4.2667	12.9537
400	3.5504	6.6461	23.6392
500	5.6489	10.0919	37.7889
600	8.3033	17.7157	55.5283
700	11.2540	23.2613	77.0185
800	14.8231	26.7550	102.4330
900	18.7517	33.3306	129.7009
1000	23.4290	41.0557	161.5674

Таблица 4.3 – На входе случайный массив

Размер	Выбором	Шелла	Гномья
100	0.2079	0.4813	0.7389
200	0.7309	1.7043	3.0184
300	1.8340	4.3971	6.5836
400	3.1248	6.9417	12.2080
500	4.9933	10.3987	19.7805
600	7.3488	18.3833	29.6323
700	10.0685	23.9719	40.9918
800	13.3175	27.9766	54.7456
900	17.1284	34.7869	70.0897
1000	20.7750	41.5146	87.0879

На рисунках 4.2, 4.3, 4.4 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.

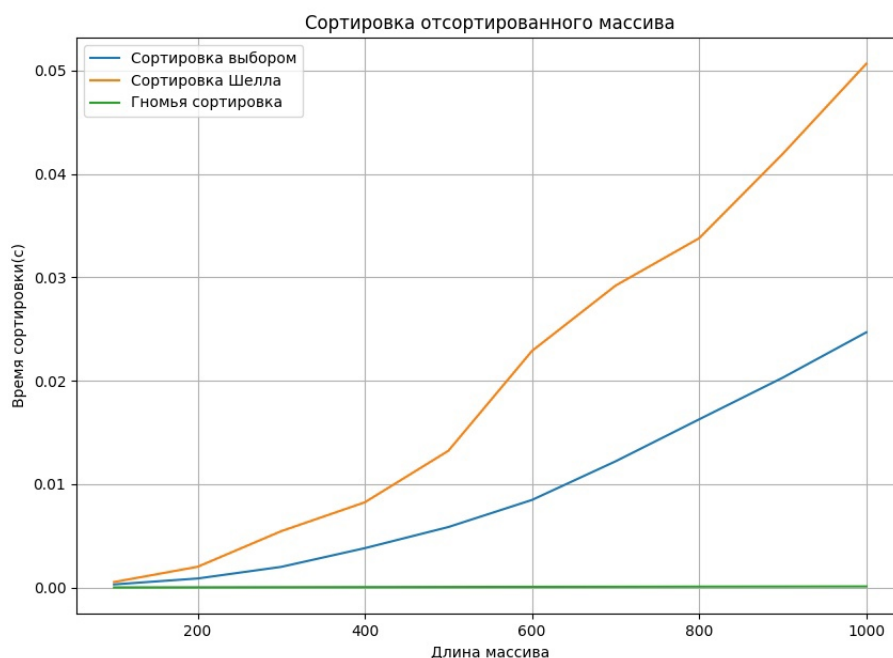


Рисунок 4.2 – На входе отсортированный массив

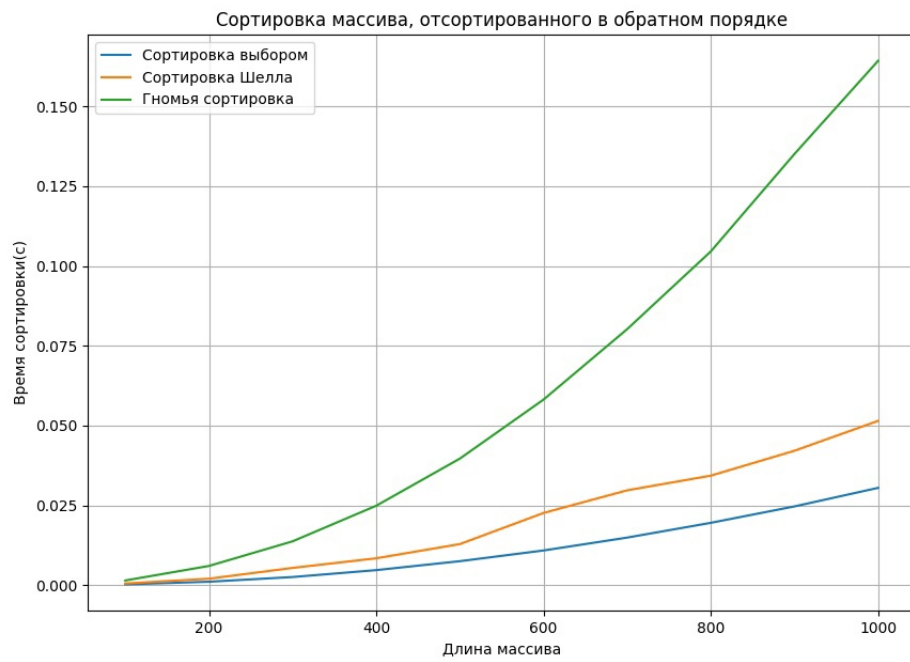


Рисунок 4.3 – На входе отсортированный в обратном порядке массив

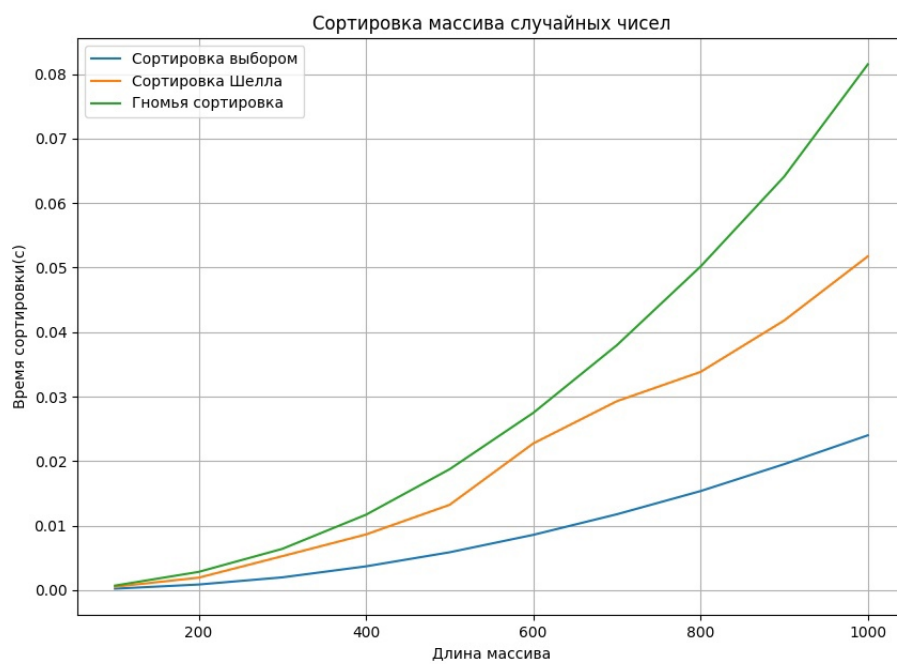


Рисунок 4.4 – На входе заполненный случайно массив

Вывод

Сортировка выбором работает быстрее при сортировке случайно заполненного массива и обратно отсортированного массива. Гномья сортировка в этих случаях работает дольше всех. При этом в случае отсортированного массива гномья сортировка оказывается самой быстрой, самой медленной оказывается сортировка Шелла.

Результаты временных замеров совпадают с вычисленными трудоемкостями.

Заключение

В ходе лабораторной работы были решены следующие задачи:

- были изучены три алгоритма сортировки: выбором, Шелла и гномья;
- 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.

Таким образом, поставленная цель достигнута.

Список литературы

- [1] Н.Вирт Алгоритмы и структуры данных. 1989.
- [2] Gnome Sort - The Simplest Sort Algorithm [Электронный ресурс]. Режим доступа: <https://dickgrune.com/Programs/gnomesort.html> (дата обращения 3.10.2021).
- [3] Welcome to Python [Электронный ресурс]. Режим доступа: <https://www.python.org> (дата обращения: 04.10.2021).
- [4] time — Time access and conversions [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html#functions> (дата обращения: 04.10.2021).
- [5] М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. 2007.
- [6] Ubuntu 20.04 LTS (Focal Fossa) Beta [Электронный ресурс]. Режим доступа: <http://old-releases.ubuntu.com/releases/20.04.1/> (дата обращения: 04.10.2021).
- [7] Мобильный процессор AMD Ryzen™ 3 3200U с графикой Radeon™ Vega 3 [Электронный ресурс]. Режим доступа: <https://www.amd.com/ru/products/apu/amd-ryzen-3-3200u> (дата обращения: 04.10.2021).