



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 Сравнительный анализ алгоритмов сортировки

Студент Шацкий Р.Е.

Группа ИУ7-55Б

Оценка (баллы) _____

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

Оглавление

Введение	2
1 Аналитическая часть	4
1.1 Шейкерная сортировка	4
1.2 Сортировка выбором	4
1.3 Сортировка Шелла	4
1.4 Вывод	5
2 Конструкторская часть	6
2.1 Требования к программному обеспечению	6
2.2 Схемы алгоритмов	6
2.2.1 Схема алгоритма шейкерной сортировки	6
2.2.2 Схема алгоритма сортировки выбором	7
2.2.3 Схема алгоритма сортировки Шелла	9
2.3 Модель вычислений	10
2.4 Трудоемкость алгоритмов сортировки	10
2.4.1 Трудоемкость шейкерной сортировки	10
2.4.2 Трудоемкость алгоритма сортировки выбором	11
2.4.3 Трудоемкость сортировки Шелла	11
2.5 Структуры данных	11
2.6 Вывод	12
3 Технологическая часть	13
3.1 Средства реализации	13
3.2 Реализация алгоритмов	13
3.3 Тестирование	15
3.4 Вывод	15
4 Экспериментальная часть	16
4.1 Пример работы программы	16
4.2 Технические характеристики	16
4.3 Время выполнения алгоритмов	17
4.4 Вывод	24
Заключение	25
Список литературы	26

Введение

Алгоритмы сортировки — это алгоритмы, которые берут некоторую последовательность из n элементов a_1, a_2, \dots, a_n и переставляют элементы таким образом, чтобы получившаяся последовательность a'_1, a'_2, \dots, a'_n удовлетворяла условию: $a'_1 \leq a'_2 \leq \dots \leq a'_n$ [1].

Алгоритмы сортировки можно классифицировать по:

- принципу сортировки:
 - сортировки, использующие сравнения: быстрая сортировка, пирамидальная сортировка, сортировка вставками и другие;
 - сортировки, не использующие сравнения: блочная сортировка, поразрядная сортировка, сортировка подсчётом и другие;
 - прочие, например, обезьянья сортировка;
- устойчивости (сортировка является устойчивой в том случае, если для любой пары элементов с одинаковыми ключами, она не меняет их порядок в отсортированном списке);
- вычислительной сложности;
- использованию дополнительной памяти;
- рекурсивности;
- параллельности;
- адаптивности (сортировка является адаптивной в том случае, когда она выигрывает от того, что входные данные могут быть частично или полностью отсортированы, например, сортировка вставками);
- использованию внутренней или внешней памяти компьютера.

Цель: сравнить трудоемкость сортировок выбором, шейкерной и Шелла.

Задачи:

1. Изучить алгоритмы сортировки перемешиванием, выбором и Шелла.
2. Вычислить и трудоемкость реализуемых алгоритмов на основе выбранной модели вычислений.
3. Сравнить трудоемкость реализуемых алгоритмов на основе выбранной модели вычислений.
4. Реализовать и протестировать алгоритмы:

- шейкерной сортировки;
 - сортировки выбором;
 - сортировки Шелла.
5. Провести экспериментальный подсчет процессорного времени, затрачиваемого алгоритмами.
 6. Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (процессорному времени работы).

1 | Аналитическая часть

В этом разделе будут рассмотрены виды сортировок, реализуемые в работе и поставлены основные требования к ПО.

1.1 Шейкерная сортировка

Шейкерная сортировка (двунаправленная, перемешиванием) - оптимизированный алгоритм пузырьковой сортировки. В основе данной сортировки лежит сравнение двух элементов, которое осуществляется в двух направлениях поочередно, постепенно сужая диапазон сортировки. В итоге за один проход в конец массива “всплывает” максимальный элемент из диапазона, а за следующий проход — в начало массива минимальный (мы рассматриваем сортировку по возрастанию). Эти элементы можно больше не рассматривать и таким образом диапазон сужается с двух сторон. [2].

Изначальная версия алгоритма подразумевает изменение границы на единицу каждый раз вне зависимости от наличия упорядоченных промежутков в массиве, таким образом цикл повторяется много раз даже для упорядоченных частей массива. В связи с этим существует оптимизация в виде добавления переменной, запоминающей место последнего обмена при перестановке элементов, а затем смещения соответствующей границы на полученное значение.

В данной работе рассматривается классическая реализация алгоритма сортировки перемешиванием.

1.2 Сортировка выбором

Сортировка выбором работает следующим образом: на каждом проходе выбирается минимальный элемент и смещается в начало. При этом каждый новый проход начинается со сдвига вправо, то есть первый проход начинается с первого элемента, второй проход — со второго [3].

1.3 Сортировка Шелла

Сортировка Шелла может рассматриваться в качестве обобщения как сортировки пузырьком, так и сортировки вставками.

Идея метода заключается в сравнении разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно d или $N/2$, где N — общее число элементов. На первом шаге каждая группа включает в себя

два элемента расположенных друг от друга на расстоянии $N/2$; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается на $d/2$, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на $d=1$ проход по массиву происходит в последний раз [4].

1.4 Вывод

Были рассмотрены алгоритмы сортировки перемешиванием, выбором и Шелла. Все сортировки являются простыми, но отличаются подходом: шейкерная проходит массив в двух направлениях, сравнивая и обменивая соседние элементы, сортировка Шелла сравнивает и обменивает элементы с некоторым шагом, сортировка выбором на каждом проходе ищет минимальный элемент и сдвигает его в начало.

Выдвинуты требования к разрабатываемому ПО:

- предусмотреть возможность ввода или генерации массива (вещественных чисел) заданной длины (целое число);
- предусмотреть возможность сравнения процессорного времени, затраченного на сортировку каждым из представленных способов.

2 | Конструкторская часть

Данный раздел содержит схемы алгоритмов сортировки, реализуемых в работе (Шелла, шейкерной, выбором), а также теоретические вычисления трудоемкости для каждого из них и описание структуры ПО.

2.1 Требования к программному обеспечению

Требования, выдвигаемые к разрабатываемому ПО:

- входные данные - последовательность вещественных чисел для сортировки;
- выходные данные - отсортированная последовательность, лежащая в области памяти исходного массива.

2.2 Схемы алгоритмов

В данном пункте раздела представлены схемы реализуемых в работе алгоритмов.

2.2.1 Схема алгоритма шейкерной сортировки

На рисунке 2.1 представлена схема алгоритма сортировки перемешиванием.

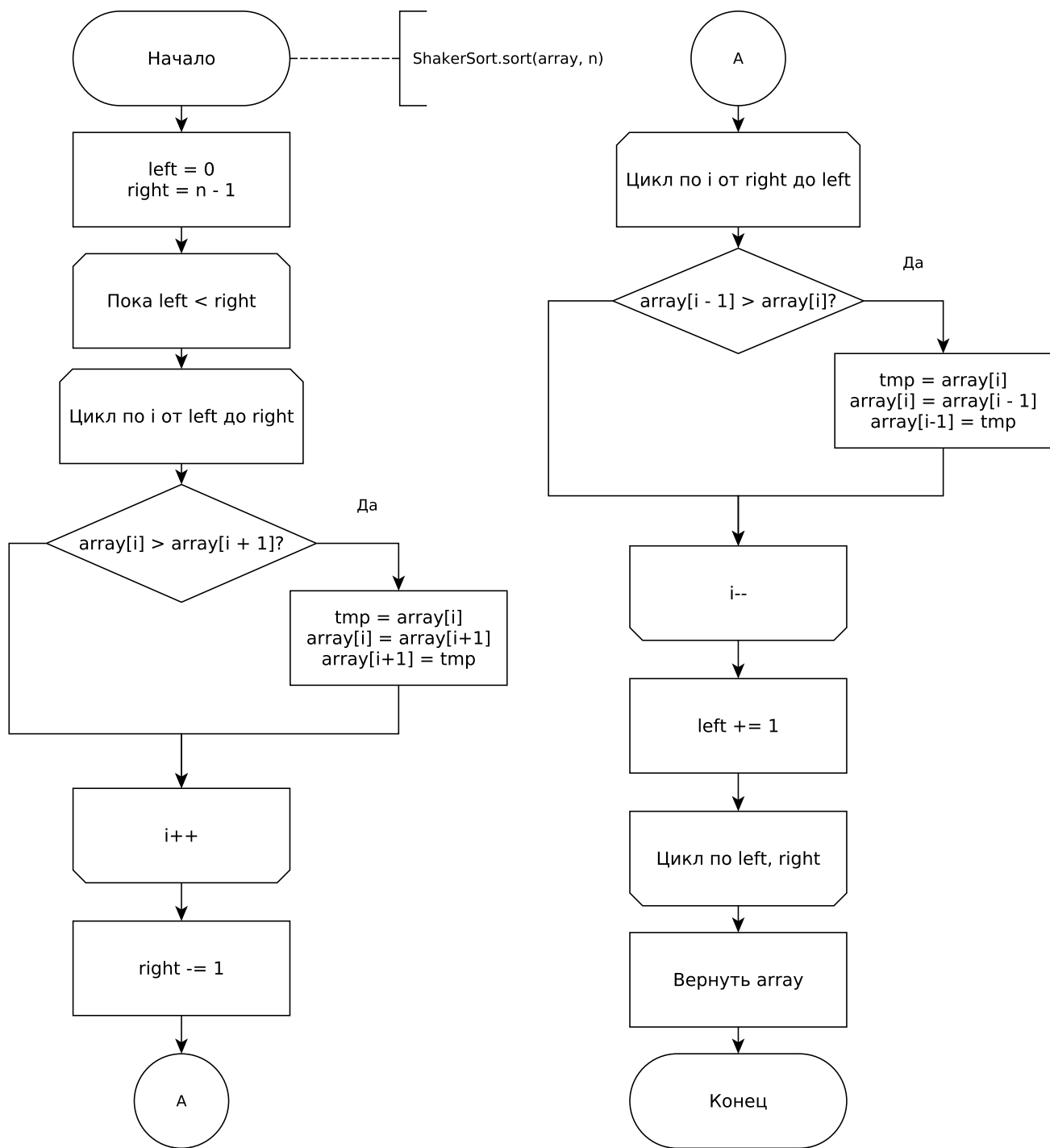


Рис. 2.1: Схема алгоритма шейкерной сортировки

2.2.2 Схема алгоритма сортировки выбором

На рисунке 2.2 представлена схема алгоритма сортировки выбором.

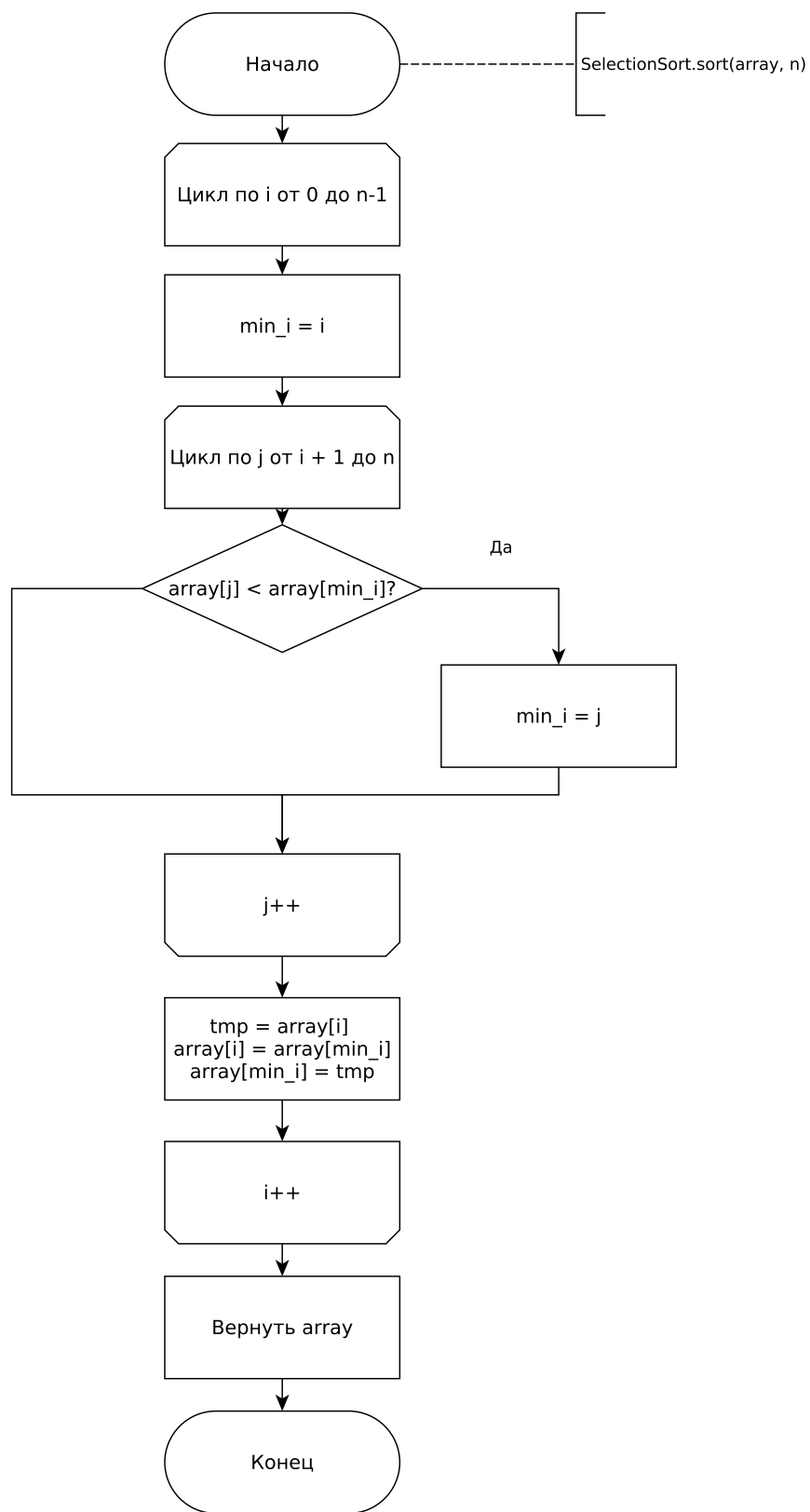


Рис. 2.2: Схема алгоритма сортировки выбором

2.2.3 Схема алгоритма сортировки Шелла

Схема алгоритма сортировки Шелла представлена на рисунке 2.3.

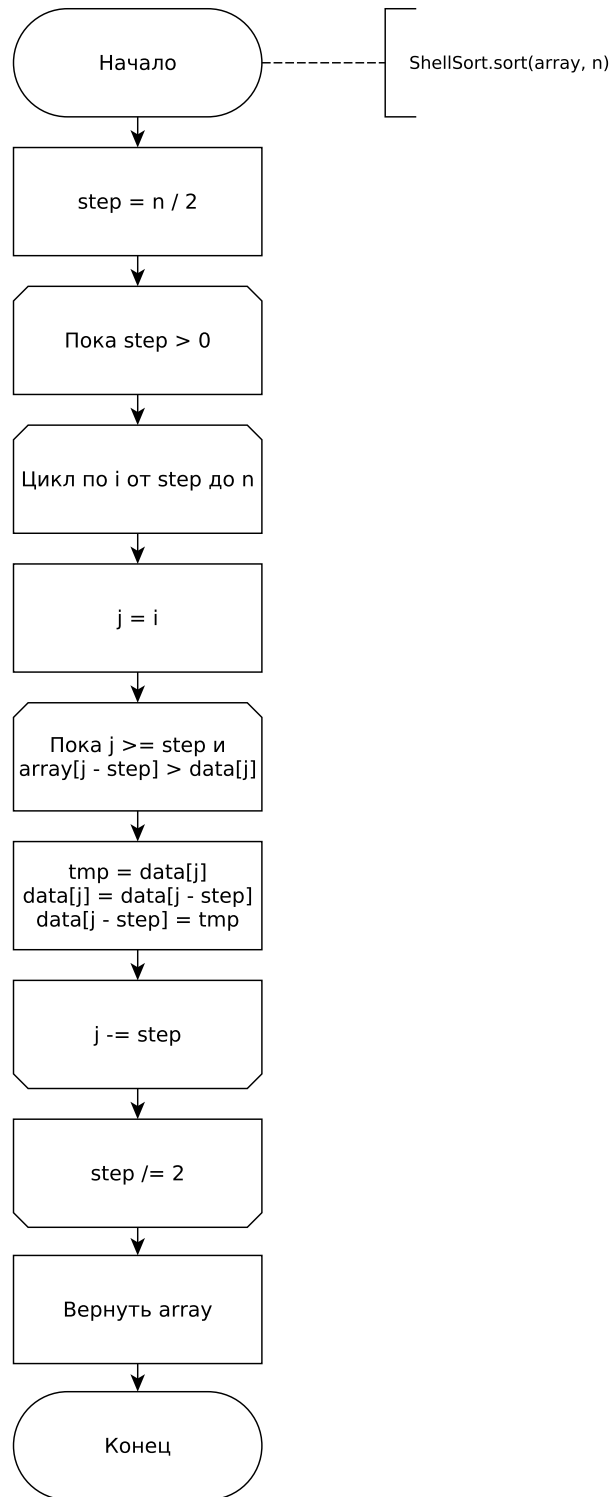


Рис. 2.3: Схема сортировки Шелла

2.3 Модель вычислений

В данной работе используется следующая модель вычислений для определения трудоемкости алгоритмов [5]:

- операции из списка 2.1 имеют трудоемкость 1;

$$+, -, + =, - =, =, ==, !=, !, \&\&, ||, <, >, <<, >>, <=, >=, [] \quad (2.1)$$

- операции из списка 2.2 имеют трудоемкость 2;

$$*, /, //, \%, * =, / = \quad (2.2)$$

- трудоемкость цикла рассчитывается по формуле 2.3;

$$f_{\text{цикла for}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N_{\text{итераций}}(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \quad (2.3)$$

- трудоемкость условного оператора *if (if(условие) then {A} else {B})* рассчитывается по формуле 2.4 с учетом того, что трудоемкость условного перехода равна 0.

$$f_{if} = f_{\text{условия}} + \begin{cases} \min(f_A, f_B), & \text{лучший случай} \\ \max(f_A, f_B), & \text{худший случай} \end{cases} \quad (2.4)$$

2.4 Трудоемкость алгоритмов сортировки

В данном подразделе представлены расчеты трудоемкости для алгоритмов..

2.4.1 Трудоемкость шейкерной сортировки

Трудоемкость алгоритма сортировки перемешиванием складывается из следующих частей:

- затраты на содержание внешнего цикла от 1 до $n - 1$ с шагом 2: $f_{\text{внеш}} = 1 + 2 + 1 + \frac{n}{2}(1 + 2 + f_{\text{forw}} + f_{\text{back}}) = 4 + \frac{n}{2}(3 + f_{\text{initforw}} + f_{\text{initback}})$;
- инициализация вложенного цикла, проходящего массив в прямом направлении: $f_{\text{initforw}} = 2$;
- инициализация вложенного цикла, проходящего массив в обратном направлении: $f_{\text{initback}} = 2$;
- внутренний цикл, проходящий массив в прямом направлении: $f_{\text{forward}} = \frac{1+n-1}{2} \cdot \frac{n}{2}(1 + 1 + 4 + f_{\text{усл}}) = \frac{n^2}{4}(6 + f_{\text{усл}})$;
- внутренний цикл, проходящий массив в обратном направлении: $f_{\text{back}} = \frac{1+n-2}{2} \cdot \frac{n-1}{2}(1 + 1 + 4 + f_{\text{усл}}) = \frac{(n-1)^2}{4}(6 + f_{\text{усл}})$;

- тело условия (худший случай): $f_{\text{ysl}} = 2 + 4 + 2 = 9$

Подставив полученные выражения, для лучшего случая (отсортированный массив) получим формулу 2.5, для худшего (отсортированный в обратном порядке) - выражение 2.6.

$$f = 4 + \frac{n}{2}(3 + 2 + 2) + \frac{n^2}{4}(6 + 0) + \frac{(n-1)^2}{4}(6 + 0) = 3n^2 + \frac{n}{2} + \frac{11}{2} \approx 3n^2 \quad (2.5)$$

$$f = 4 + \frac{n}{2}(3 + 2 + 2) + \frac{n^2}{4}(6 + 9) + \frac{(n-1)^2}{4}(6 + 9) = \frac{15}{2}n^2 - 4n + \frac{31}{4} \approx \frac{15}{2}n^2 \quad (2.6)$$

2.4.2 Трудоемкость алгоритма сортировки выбором

Трудоемкость алгоритма сортировки выбором содержит следующие части:

- внешний цикл от 1 до n: $f_i = 1 + 1 + (n - 1) \cdot (1 + 1 + 1 + f_{\text{swap}} + f_{\text{init } j}) = 2 + (n - 1) \cdot (3 + f_{\text{swap}} + f_{\text{init } j})$;
- затраты на инициализацию внутреннего цикла: $f_{\text{init } j} = 2 + 1 = 3$;
- внутренний цикл от $i + 1$ до n: $f_j = \frac{1+n-1}{2} \cdot (n - 1)(1 + 1 + 3 + f_{\text{ysl}}) = \frac{n^2-n}{2} \cdot (5 + f_{\text{ysl}})$;
- тело условия (худший случай): $f_{\text{ysl}} = 1$;
- трудоемкость операции обмена: $f_{\text{swap}} = 2 + 4 + 2 = 9$.

Таким образом, для лучшего случая (отсортированный массив) получим выражение 2.7.

$$f = 2 + (n - 1) \cdot (3 + 9 + 3) + \frac{n^2-n}{2} \cdot (5 + 0) = \frac{5}{2}n^2 + \frac{25}{2}n - 13 \approx \frac{5}{2}n^2 \quad (2.7)$$

Для худшего случая (отсортированный в обратном порядке массив) получим выражение 2.8.

$$f = 2 + (n - 1) \cdot (3 + 9 + 3) + \frac{n^2-n}{2} \cdot (5 + 1) = 3n^2 + 12n - 13 \approx 3n^2 \quad (2.8)$$

2.4.3 Трудоемкость сортировки Шелла

Для определения трудоемкости сортировки Шелла воспользуемся данными ресурсов. Таким образом, трудоемкость лучшего случая (массив отсортирован в правильном порядке) составляет $n \log n$, худшего (неподходящий выбор шагов сравнения) - n^2 [6].

2.5 Структуры данных

В программе есть 3 класса, отвечающие за сортировку массивов, каждый из которых реализует интерфейс Sort и его метод `sort(array: List[float], n: int) -> List[float]`. На рисунке 2.4 изображена uml-диаграмма классов, используемых в ПО.

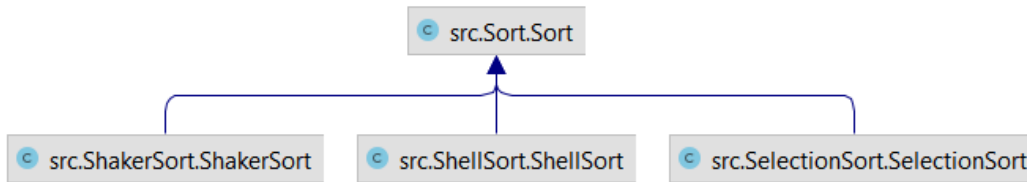


Рис. 2.4: Диаграмма классов ПО

2.6 Вывод

В данном разделе на основе приведенных в аналитическом разделе теоретических данных были составлены схемы алгоритмов для реализации в технологической части, вычислена трудоемкость лучшего и худшего случаев алгоритмов сортировок перемешиванием и выбором, приведены данные трудоемкости алгоритма сортировки Шелла. Лучший случай для выбранных алгоритмов оказался общим, это случай обработки отсортированного массива. Худший случай совпал для шейкерной сортировки и сортировки выбором, это обработка отсортированного в обратном порядке массива. Худшим случаем для алгоритма сортировки Шелла является случай неудачного выбора шага сравнения. Приблизительная трудоемкость шейкерной сортировки для лучшего случая равна $3n^2$, худшего - $\frac{15}{2}n^2$; сортировки выбором для лучшего случая - $\frac{5}{2}n^2$, для худшего - $3n^2$; сортировки Шелла для лучшего случая - $n \log n$, для худшего случая - n^2 . Таким образом, наиболее эффективной является сортировка Шелла, наименее - шейкерная сортировка.

3 | Технологическая часть

Данный раздел содержит обоснование выбора языка и среды разработки, реализацию алгоритмов.

3.1 Средства реализации

Для реализации программы был выбран язык программирования Python [7]. Такой выбор обусловлен следующими причинами:

- имеет большое количество расширений и библиотек, что облегчает работу с некоторыми типами данных и математическими формулами;
- обладает информативной документацией.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Шейкерная сортировка

```
1 class ShakerSort(Sort):
2     def __init__(self):
3         super(ShakerSort, self).__init__()
4
5     def sort(self, array):
6         left = 0
7         right = len(array) - 1
8
9         while left < right:
10            for i in range(left, right):
11                if array[i] > array[i + 1]:
12                    array[i], array[i + 1] = array[i + 1], array[i]
13                    right -= 1
14
15            for i in range(right, left, -1):
16                if array[i - 1] > array[i]:
17                    array[i], array[i - 1] = array[i - 1], array[i]
18                    left += 1
19        return array
```

Листинг 3.2: Сортировка выбором

```
1 class SelectionSort(Sort):
2     def __init__(self):
3         super(SelectionSort, self).__init__()
4
5     def sort(self, array):
6         n = len(array)
7         for i in range(n - 1):
8             min_i = i
9             for j in range(i + 1, n):
10                if array[j] < array[min_i]:
11                    min_i = j
12            array[i], array[min_i] = array[min_i], array[i]
13        return array
```

Листинг 3.3: Сортировка Шелла

```
1 class ShellSort(Sort):
2     def __init__(self):
3         super(ShellSort, self).__init__()
4
5     def sort(self, array):
6         n = len(array)
7         step = n // 2
8         while step > 0:
9             for i in range(step, n):
10                 j = i
11                 while j >= step and array[j - step] > array[j]:
12                     array[j - step], array[j] = array[j], array[j - step]
13                     j -= step
14             step //= 2
15         return array
```

3.3 Тестирование

В таблице 3.1 представлены использованные для тестирования методом "черного ящика" данные, были рассмотрены все возможные тестовые случаи. Все тесты пройдены успешно. Здесь и далее "Массив входная последовательность, "Шейкерная алгоритм шейкерной сортировки, "Выбором алгоритм сортировки выбором, "Шелла алгоритм сортировки Шелла.

Таблица 3.1: Проведенные тесты

Массив	Шейкерная	Выбором	Шелла
[1, 2, 3, 4, 5, 6]	[1, 2, 3, 4, 5, 6]	[1, 2, 3, 4, 5, 6]	[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3]	[3, 4, 5, 6]	[3, 4, 5, 6]	[3, 4, 5, 6]
[-1, 1, 2, 0, 3, -2]	[-2, -1, 0, 1, 2, 3]	[-2, -1, 0, 1, 2, 3]	[-2, -1, 0, 1, 2, 3]
[-2, 3, -2, 6, 3, 1]	[-2, -2, 1, 3, 3, 6]	[-2, -2, 1, 3, 3, 6]	[-2, -2, 1, 3, 3, 6]
[2]	[2]	[2]	[2]
[5, 6, -9, 3, -10]	[-10, -9, 3, 5, 6]	[-10, -9, 3, 5, 6]	[-10, -9, 3, 5, 6]
[]	Ошибка	Ошибка	Ошибка

3.4 Вывод

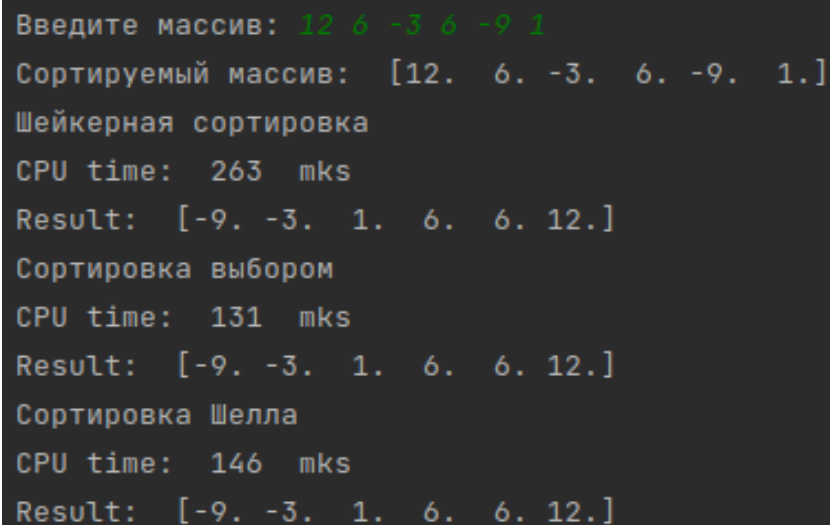
В данном разделе были реализованы и протестированы алгоритмы сортировки: перемешиванием, выбором и Шелла.

4 | Экспериментальная часть

В данном разделе сравниваются реализованные алгоритмы, дается сравнительная оценка затрат и время.

4.1 Пример работы программы

Пример работы программы представлен на рисунке 4.1.



```
Введите массив: 12 6 -3 6 -9 1
Сортируемый массив: [12.  6. -3.  6. -9.  1.]
Шейкерная сортировка
CPU time:  263  mks
Result:  [-9. -3.  1.  6.  6. 12.]
Сортировка выбором
CPU time:  131  mks
Result:  [-9. -3.  1.  6.  6. 12.]
Сортировка Шелла
CPU time:  146  mks
Result:  [-9. -3.  1.  6.  6. 12.]
```

Рис. 4.1: Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- операционная система — Windows [8] 10 64-bit;
- оперативная память — 16 Гб;
- процессор — Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz [9].

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось на автоматически генерируемых массивах необходимого размера и упорядоченности (вещественные числа находятся в диапазоне $[-n \times 10, n \times 10]$) с использованием функции `getrusage` библиотеки `resources` [10]. Усредненные результаты замеров процессорного времени приведены в таблицах ниже.

На рисунке 4.2 представлен график зависимости времени работы алгоритмов от размера произвольно упорядоченных массивов на основе таблицы 4.1. Для произвольно упорядоченных массивов самым эффективным алгоритмом оказался алгоритм Шелла. Выигрыш сортировки Шелла по сравнению с алгоритмом сортировки выбором составляет от 1.5 до 16 раз; по сравнению с шейкерной сортировкой - от 2 до 30 раз. Выигрыш по времени увеличивается с увеличением размера входной последовательности. Наименее эффективной оказалась шейкерная сортировка, которая проигрывает сортировке выбором в среднем в 2 раза.

Таблица 4.1: Время обработки произвольных массивов разной длины в микросекундах

Размер	Шейкер	Выбором	Шелла
5	96	71	66
7	150	116	106
10	290	207	185
20	1092	690	498
50	6713	3824	1904
100	26545	14576	4982
200	104361	56225	12367
500	616529	212110	22072
1000	1584832	841228	53035

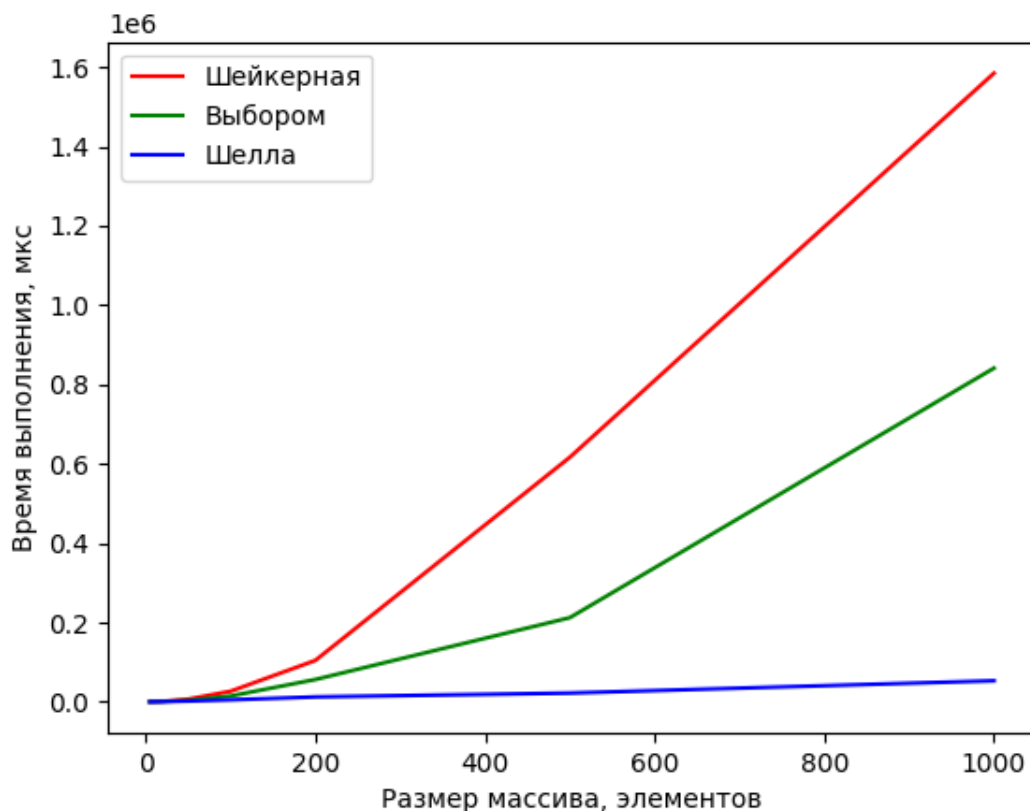


Рис. 4.2: Зависимость времени работы алгоритмов сортировки от размера произвольно упорядоченного массива

На рисунке 4.3 представлен график зависимости времени работы алгоритмов от размера упорядоченных массивов на основе таблицы 4.2. Для упорядоченных массивов самым эффективным алгоритмом также оказался алгоритм Шелла. Его выигрыш по сравнению с другими сортировками составил от 2 до 56 раз, при увеличении с увеличением размера входной последовательности. Время работы алгоритмов сортировки перемешиванием и выбором на упорядоченных массивах почти совпало с незначительным преимуществом сортировки выбором.

Таблица 4.2: Время обработки упорядоченных массивов разной длины в микросекундах

Размер	Шейкер	Выбором	Шелла
5	86	74	43
7	103	126	53
10	181	210	97
20	665	693	234
50	3833	3820	709
100	14827	14389	1679
200	57329	55620	4004
500	355309	327702	7988
1000	890107	866875	16047

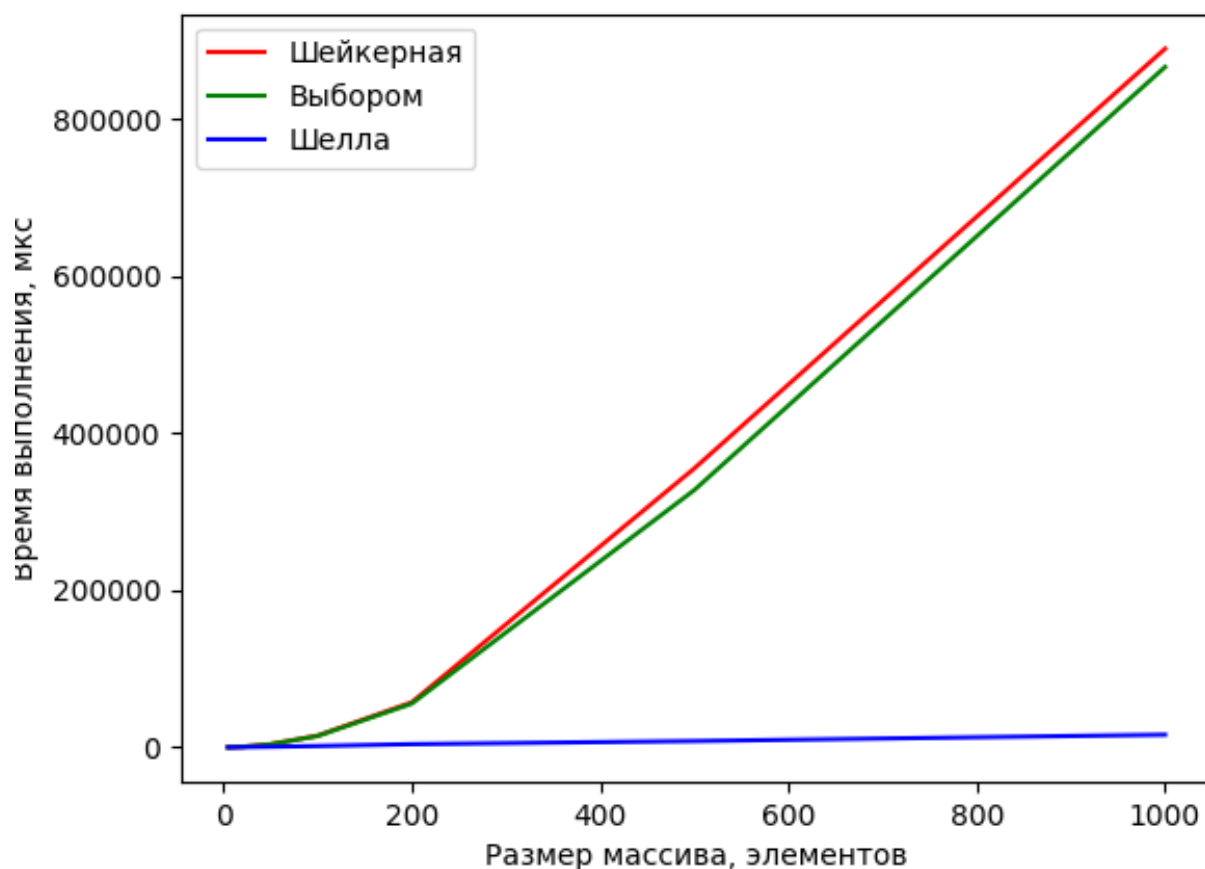


Рис. 4.3: Зависимость времени работы алгоритмов сортировки от размера упорядоченного массива

На рисунке 4.4 представлен график зависимости времени работы алгоритмов от размера упорядоченных в обратном порядке массивов на основе таблицы 4.3. В данном случае также

самым эффективным алгоритмом оказался алгоритм Шелла. Его выигрыш по сравнению с сортировкой выбором составил от 2 до 22 раз, однако на небольших массивах размером до 10 элементов выигрыш незначителен. Наименее эффективным алгоритмом сортировки оказался алгоритм шейкерной сортировки, который работает в 2-3 раза дольше сортировки выбором и от 2 до 60 раз дольше сортировки Шелла.

Таблица 4.3: Время обработки упорядоченных в обратном порядке массивов разной длины в микросекундах

Размер	Шейкер	Выбором	Шелла
5	105	73	68
7	201	121	101
10	390	209	182
20	1522	681	474
50	9328	3803	1424
100	37057	14492	3531
200	144316	55750	8540
500	737375	221832	16998
1000	2278427	818325	37704

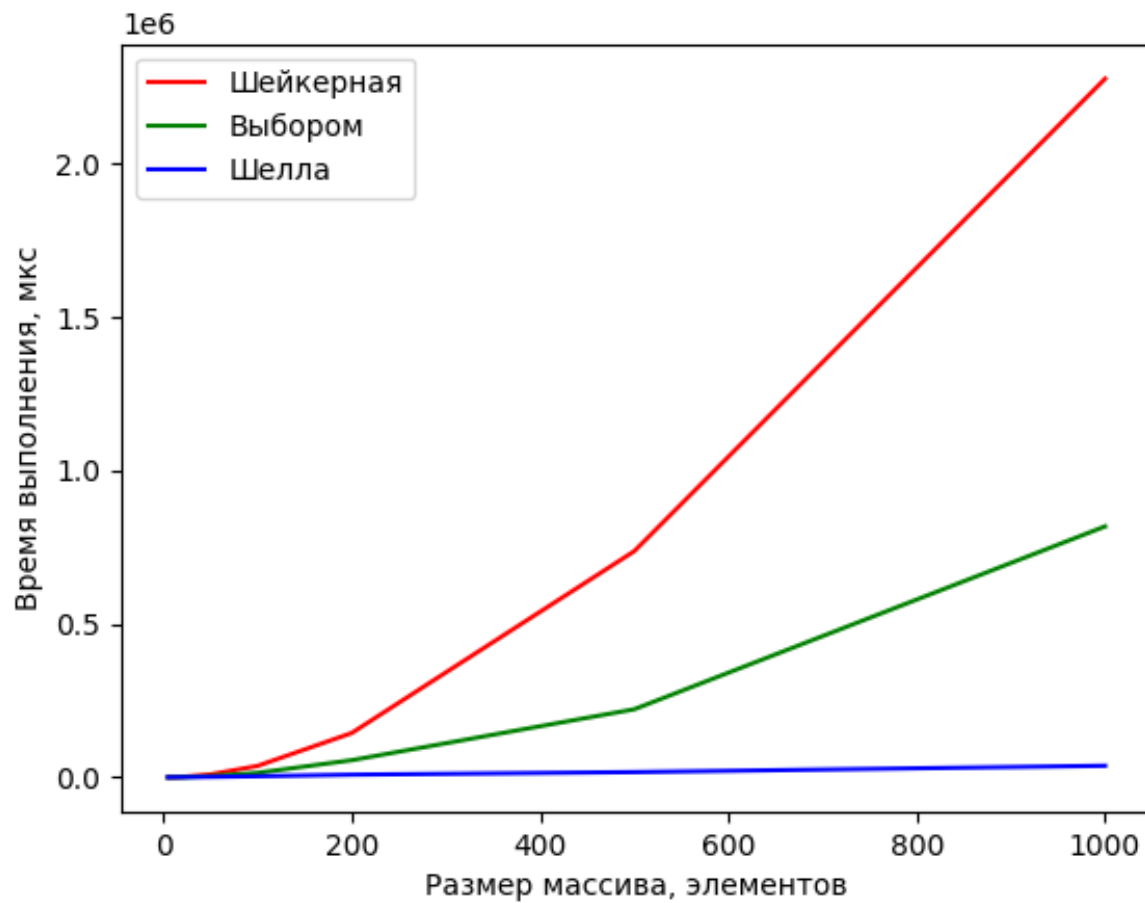


Рис. 4.4: Зависимость времени работы алгоритмов сортировки от размера упорядоченного в порядке убывания массива

На рисунках 4.5 - 4.7 представлены зависимости работы алгоритмов сортировок от размера и упорядоченности входного массива. Худшим случаем для алгоритма сортировки Шелла оказался произвольно упорядоченный массив, что говорит о неудачно выбранном шаге сравнения элементов для итераций. Время работы алгоритма сортировки выбором практически одинаковое для лучшего и худшего случаев.

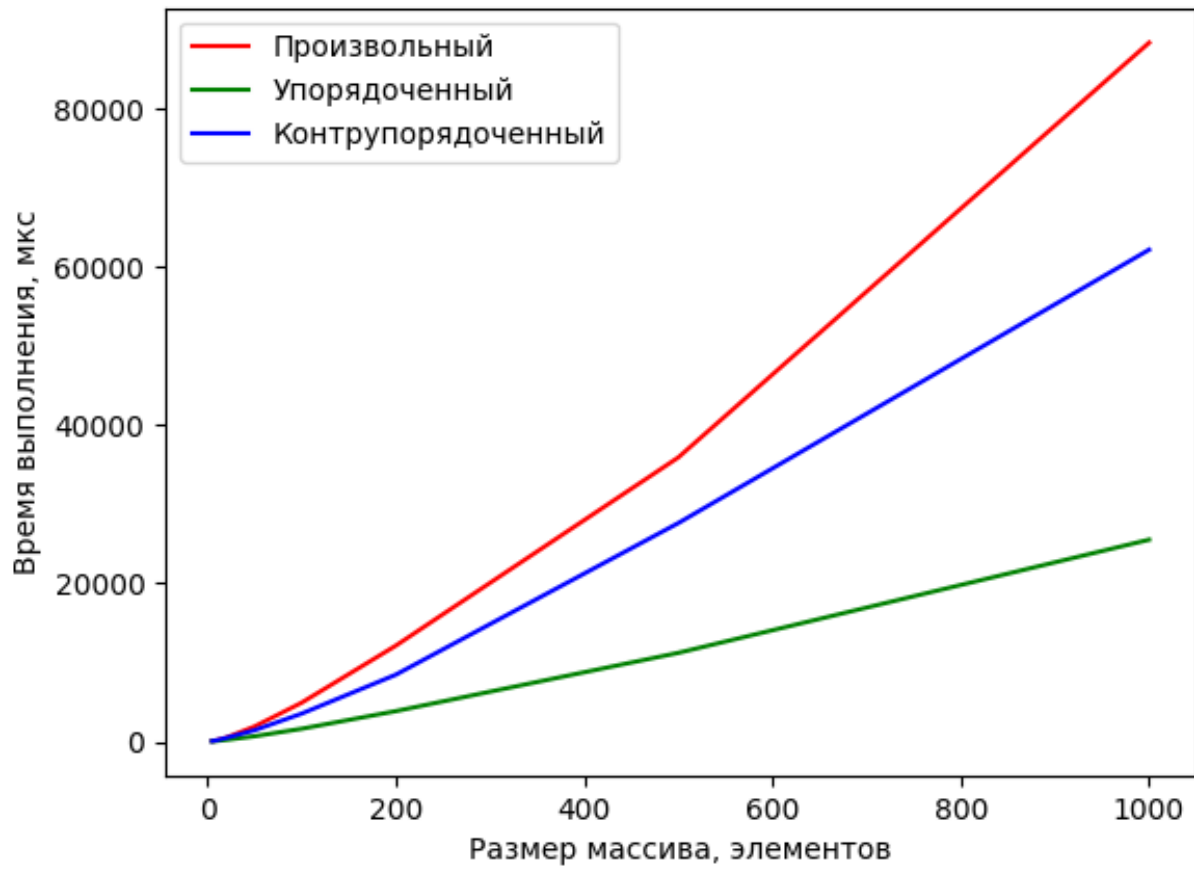


Рис. 4.5: Зависимость времени работы алгоритма шейкерной сортировки от размера и упорядоченности массива

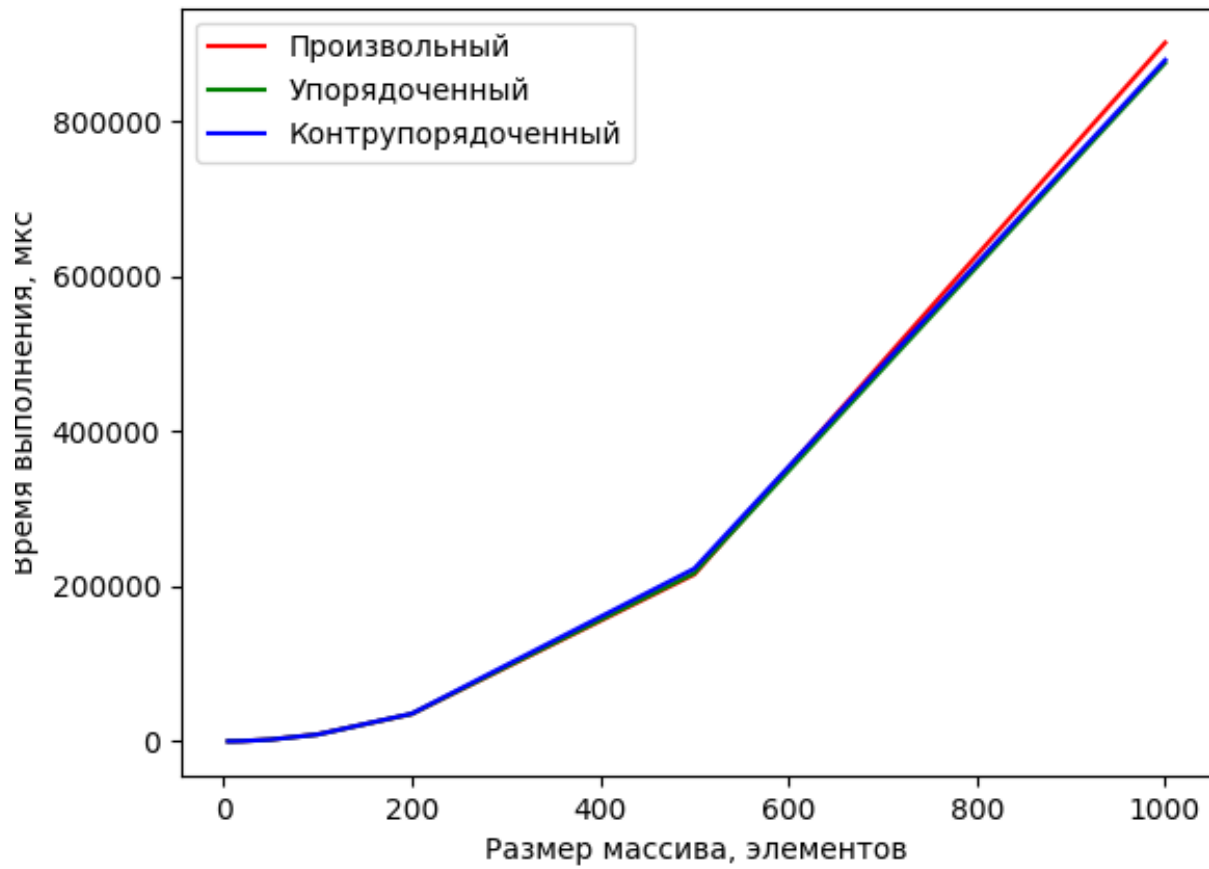


Рис. 4.6: Зависимость времени работы алгоритма сортировки выбором от размера и упорядоченности массива

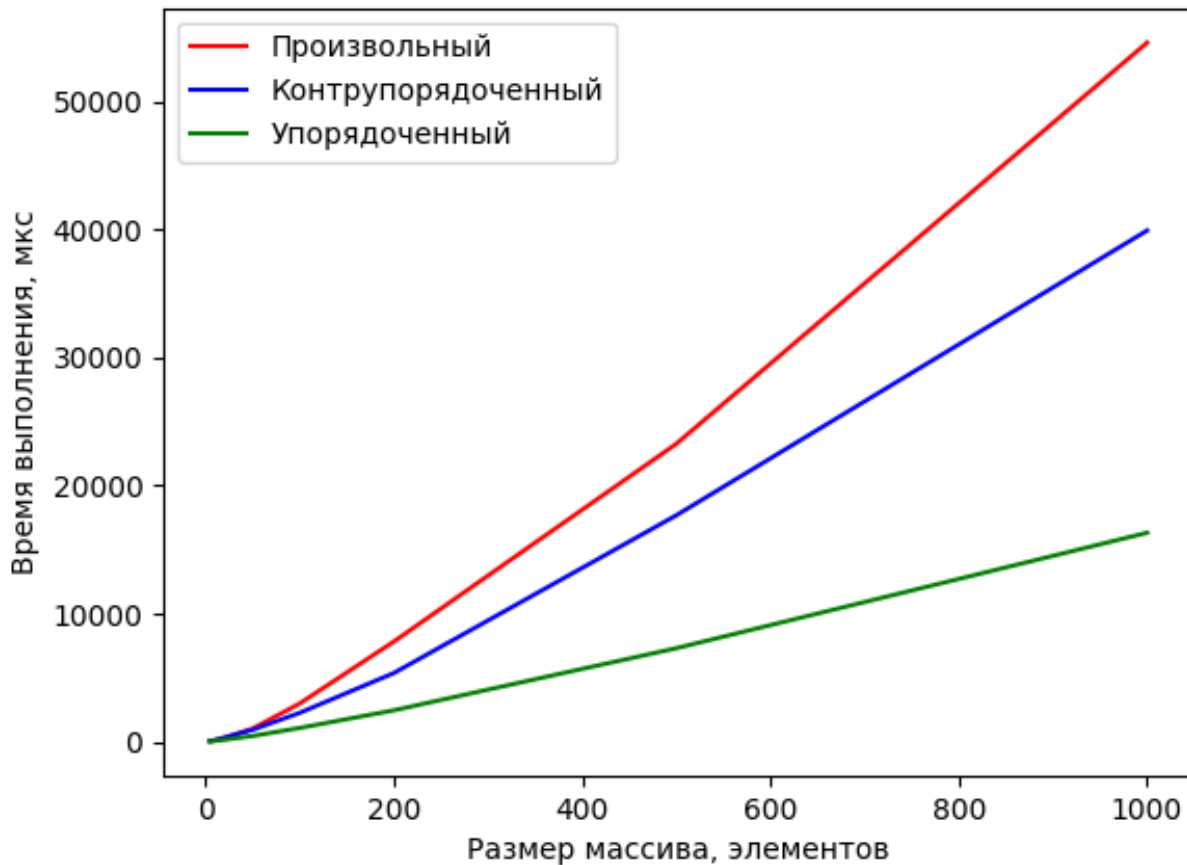


Рис. 4.7: Зависимость времени работы алгоритма сортировки Шелла от размера и упорядоченности массива.

4.4 Вывод

По результатам проведенных замеров видно, что самым быстрым является алгоритм сортировки Шелла, причем его преимущество по сравнению с сортировкой выбором составил от 1.5 до 16 раз для произвольно упорядоченных массивов, от 2 до 56 раз для отсортированных массивов и от 2 до 22 раз для отсортированных в обратном порядке; по сравнению с сортировкой перемешиванием преимущество в среднем составило от 2 до 40 раз для различных упорядоченных массивов. Время работы алгоритмов шейкерной сортировки и сортировки выбором для лучшего случая - упорядоченного массива - практически совпало с незначительным преимуществом сортировки выбором. Наименее эффективным оказался алгоритм шейкерной сортировки, проигрыш которого по сравнению с сортировкой выбором составил 2-3 раза. Худшим случаем сортировки Шелла оказался произвольно упорядоченный массив, что говорит о неподходящем выбранном шаге сравнения элементов.

Заключение

В процессе выполнения лабораторной работы были изучены и реализованы алгоритмы сортировки перемешиванием, выбором и сортировки Шелла.

Согласно проведенному анализу трудоемкости алгоритмов в соответствии с выбранной моделью вычислений, приблизительная трудоемкость шейкерной сортировки для лучшего случая равна $3n^2$, худшего - $\frac{15}{2}n^2$; сортировки выбором для лучшего случая - $\frac{5}{2}n^2$, для худшего - $3n^2$. Согласно проанализированным источникам, трудоемкость сортировки Шелла для лучшего случая равна $n \log n$, для худшего случая - n^2 . Лучший случай для выбранных алгоритмов оказался общим, это случай обработки отсортированного массива. Худший случай совпал для шейкерной сортировки и сортировки выбором, это обработка отсортированного в обратном порядке массива. Худшим случаем для алгоритма сортировки Шелла является случай неудачного выбора шага сравнения. Таким образом, наиболее эффективной является сортировка Шелла, наименее - шейкерная сортировка.

Было исследовано процессорное время выполнения выше обозначенных алгоритмов. В результате было выявлено, что самым быстрым является алгоритм сортировки Шелла, причем его выигрыш по сравнению с сортировкой выбором составил от 1.5 до 16 раз для произвольно упорядоченных массивов, от 2 до 56 раз для отсортированных массивов и от 2 до 24 раз для отсортированных в обратном порядке; по сравнению с сортировкой перемешиванием преимущество составило от 2 до 30 раз для произвольно упорядоченных, от 2 до 56 раз для упорядоченных и от 2 до 60 раз для упорядоченных в обратном порядке массивов. Преимущество по сравнению с обоими алгоритмами увеличивается с увеличением размера массива. Время работы алгоритмов шейкерной сортировки и сортировки выбором для лучшего случая - упорядоченного массива - практически совпало с незначительным преимуществом сортировки выбором. Наименее эффективным оказался алгоритм шейкерной сортировки, проигрыш которого по сравнению с сортировкой выбором составил 2-3 раза. Худшим случаем сортировки Шелла оказался произвольно упорядоченный массив, что говорит о неподходящем выбранном шаге сравнения элементов.

Таким образом, практика подтверждает теорию, и самым эффективным является алгоритм сортировки Шелла, наименее эффективным - шейкерной сортировки.

Литература

- [1] Алгоритмы сортировки. Режим доступа: <https://markoutte.me/students/sort-algos/selection-sort>. Дата обращения: 13.10.2021.
- [2] Шейкерная сортировка, JavaScript. Режим доступа: <https://medium.com/@alivander/javascript-a2b8af562ee>. Дата обращения: 13.10.2021.
- [3] Алгоритмы сортировки в теории и на практике. Режим доступа: <https://javarush.ru/groups/posts/1997-algoritmih-sortirovki-v-teorii-i-na-praktike>. Дата обращения: 13.10.2021.
- [4] Сортировка Шелла (SHELL SORT). Режим доступа: <https://forkettle.ru/vidioteka/programmirovaniye-i-set/algoritmy-i-struktury-dannykh/108-sortirovka-i-poisk-dlya-chajnikov/1008-sortirovka-shella>. Дата обращения: 13.10.2021.
- [5] М.В. Ульянов. Ресурсо-эффективные компьютерные алгоритмы. М: Наука Физматлит, 2007. Т. 376. С. 52–73.
- [6] Shekk Sort Algorithm. Режим доступа: <https://www.mycplus.com/featured-articles/shell-sort-algorithm/>. Дата обращения: 15.10.2021.
- [7] About Python [Электронный ресурс]. Режим доступа: <https://www.python.org/about/>. Дата обращения: 13.10.2021.
- [8] Сравните выпуски Windows 10 [Электронный ресурс]. Режим доступа: <https://www.microsoft.com/ru-ru/windows/compare-windows-10-home-vs-pro>. Дата обращения: 21.09.2021.
- [9] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/97150/intel-core-i57600-processor-6m-cache-up-to-4-10-ghz.html>. Дата обращения: 21.09.2021.
- [10] resource — Resource usage information - Python 3.11.0a0 documentation. Режим доступа: <https://docs.python.org/dev/library/resource.html?highlight=resources>. Дата обращения: 13.10.2021.