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 Трудоёмкость сортировок

Студент Маслюков П.В.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

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. 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.

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 - сравнить различные алгоритмы сортировки.

Для решения поставленной цели требуется решить следующие задачи:

- изучить три алгоритма сортировки: выбором, бусинами и блочную;
- разработать и реализовать указанные алгоритмы;
- протестировать реализацию рассматриваемых алгоритмов;
- провести сравнительный анализ реализованных алгоритмов по времени работы и по трудоёмкости алгоритмов.

1 Аналитическая часть

В данном разделе будут описаны алгоритмы сортировки выбором, бусинами и блочной сортировки.

1.1 Сортировка выбором

Сортировка выбором 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 Блочная сортировка

Алгоритм сортировки, который работает путем распределения элементов массива по нескольким сегментам. Затем каждый сегмент сортируется индивидуально, либо с использованием другого алгоритма сортировки, либо путем рекурсивного применения блочной сортировки.

Алгоритм работает следующим образом.

- 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.
- Все получившиеся блоки сортируются другим алгоритмом сортировки.
- Выполняется слияние блоков в единый массив.

1.3 Сортировка бусинами

Алгоритм сортировки бусинами, работает путем создания матрицы и последующего распределения элементов исходного массива. Каждое число записывается количество раз равное самому числу. Затем матрица трансплонируется и подсчитывается количество вхождений. Таким образом мы получаем отсортированный числа исходного массива [1].

Метод ограничен, прежде всего применим к натуральным числам, т.е. можно сортировать только положительные числа. Можно сортировать и целые, но это запутаннее - отрицательные числа придется обрабатывать отдельно отрицательные от положительных.

Вывод

Были рассмотрены следующие алгоритмы сортировки: выбором, бусинами и блочная. 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, бусинами и блочной сортировки.

2.1 Требования к программе

К программе представлен ряд требований:

- на вход подается массив целых чисел;
- на выходе - отсортированный массив, поданный на вход и временные замеры алгоритмов.

2.2 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 представлены схемы алгоритмов сортировки выбором, бусинами и блочной сортировки.

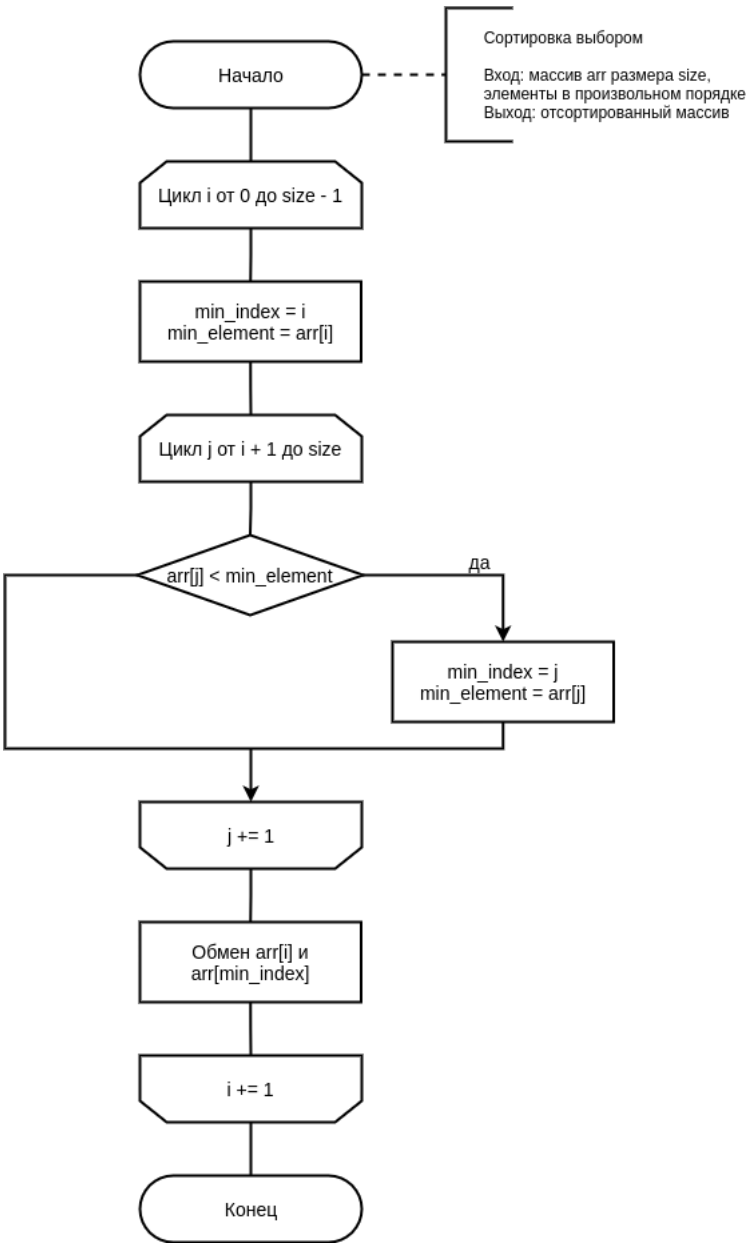

Рисунок 2.1 – Схема алгоритма сортировки выбором

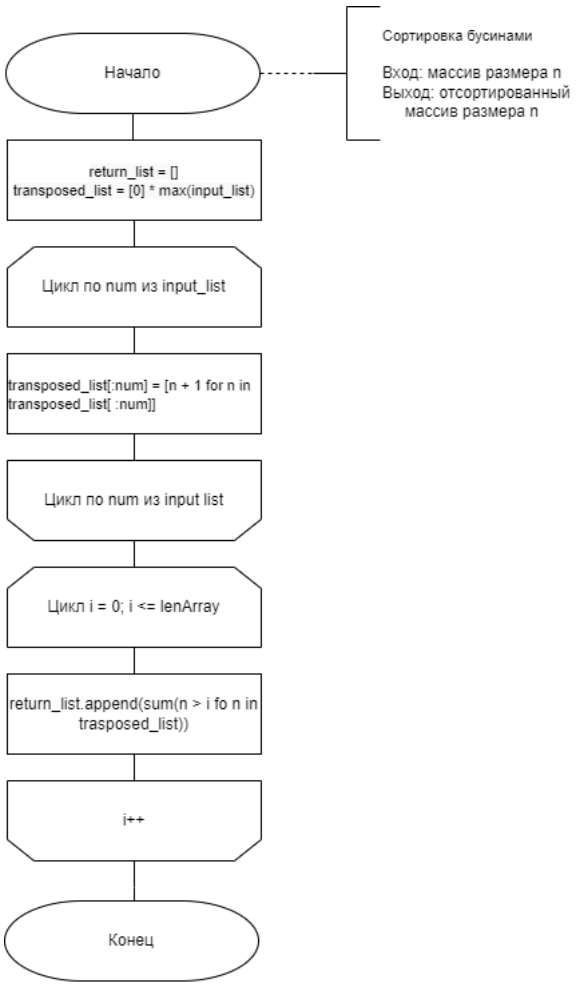

Рисунок 2.2 – Схема алгоритма сортировки бусинами

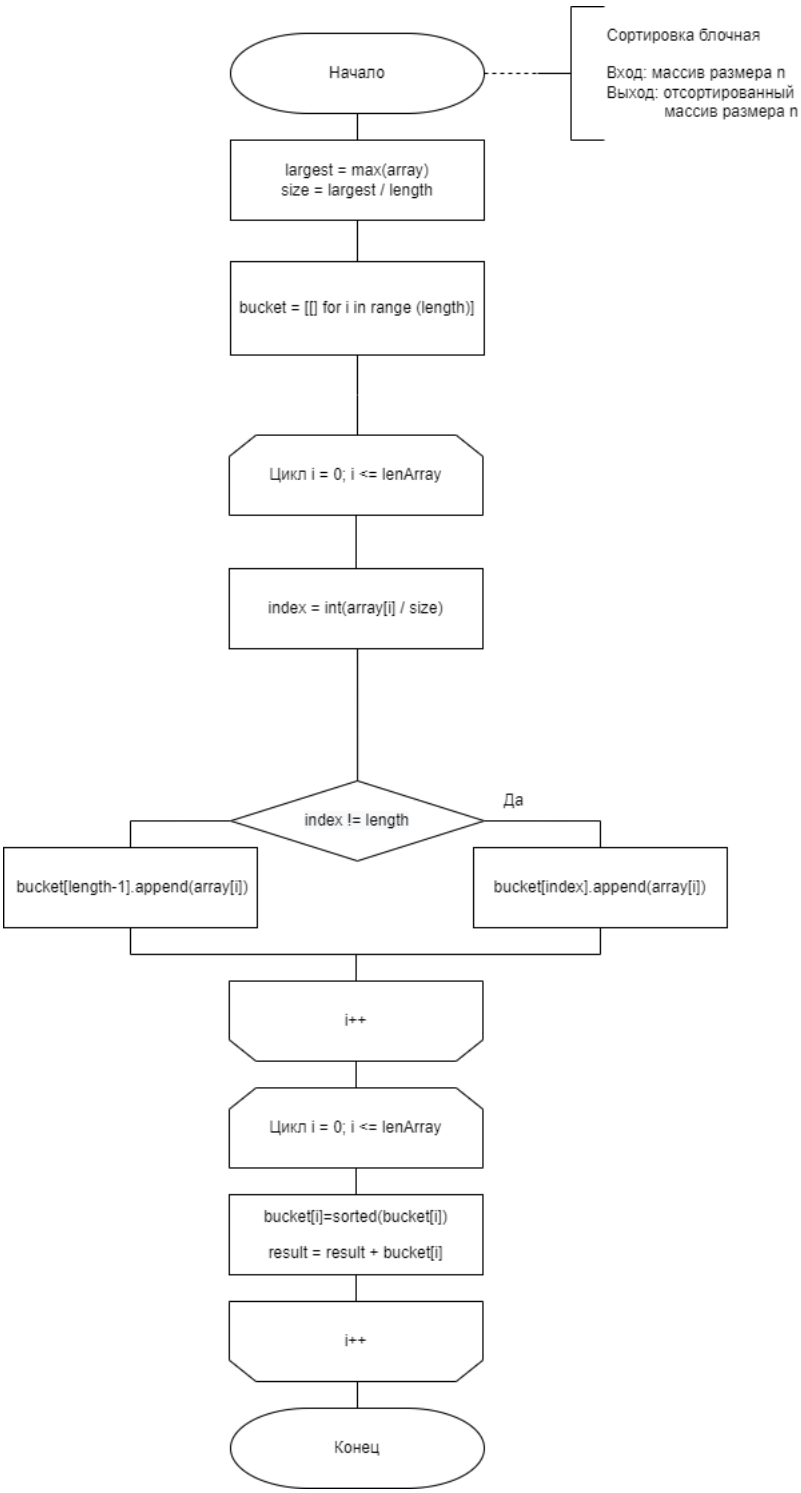

Рисунок 2.3 – Схема алгоритма блочной сортировки

2.3 Модель вычислений для оценки трудоёмкости алгоритмов

Для определения трудоемкости алгоритмов необходимо ввести модель вычислений:

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

$$+, -, /, *, \%, =, + =, - =, * =, / =, \% =, ==, ! =, <, >, <=, >=, [] \tag{2.1}$$

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

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

3. трудоемкость цикла рассчитывается, как (2.3);

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

4. трудоемкость вызова функции равна 0.

2.4 Трудоёмкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости.

2.4.1 Алгоритм сортировки выбором

Трудоемкость в лучшем случае (2.4):

$$f_{best} = 1 + (N-1)(3+3+1+(N/2)2+7) = \tag{2.4}$$
$$= N^2 + 14N - N - 14 + 1 = N^2 + 13N - 13 = O(N^2)$$

Трудоёмкость в худшем случае (2.4.1):

$$f_{worst} = 1 + (N-1)(3+3+1+(N/2)(2+3)+7) =$$
$$2.5N^2 + 14N - 2.5N - 14 + 1 = 2.5N^2 + 11.5N - 13 = O(N^2) \tag{2.5}$$

2.4.2 Алгоритм сортировки бусинами

Трудоемкость в лучшем случае при отсортированном массиве и худшем случае при неотсортированном массиве в обратном порядке.Выведена формуле 2.6.

$$f_{best} = 3 + 4 + (N-1) \cdot (2 + 2 + \begin{cases} 0, \\ 2, \end{cases}) + 1 + 2 +$$
$$+N \cdot (2+3+L \cdot (3+5)) + 2 + M \cdot (2+3+N \cdot (2+5+5) + \tag{2.6}$$
$$+3 + (N-L) \cdot (2+5)) + 2 + N \cdot (2+8+8M) =$$
$$= S + 19NM + 11N + 8M + 18 = O(S)$$

В формуле (2.6) значения S или NL — сумма всех элементов в изначальном массиве, L — значение каждого элемента в массиве, M — максимальное значение из элементов в массиве.

Исходя из результата формулы (2.6) можно понять, что лучшим случаем для сортировки будет случай если все значения массива будут соответствовать минимальному значению, а худшим случаем если значение массива будут большие значения.

2.4.3 Алгоритм блочной сортировки

Лучший случай: элементы входящего массива попали в разные корзины. Худший случай: элементы входящего массива попали в одну корзину.

Для наилучшего и наихудшего случаев сложность первого цикла будет одинакова и равна $O(n)$

Трудоемкость в лучшем случае (2.7):

$$f_{best} = N + N \cdot 1 + N \cdot 1 = O(N) \tag{2.7}$$

Трудоёмкость в худшем случае (2.8):

$$f_{worst} = N + (N-1) \cdot 1 + 1 \cdot (N \cdot log(n)) + (N-1) \cdot 2 + 1 \cdot N = O(n \cdot log(n)) \tag{2.8}$$

Вывод

Были представлены схемы алгоритмов сортировки выбором, бусинами, блочной сортировки и проанализирована трудоемкость рассматриваемых алгоритмов.

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 [2]. В текущей лабораторной работе было необходимо определить процессорное время, затрачиваемое на выполнение программы, а также построить графики. Все необходимые инструменты для этого имеются в выбранном языке программирования.

Замеры времени проводились при помощи функции process_time из библиотеки time [3].

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 Реализация алгоритмов

Реализации алгоритмов сортировок выбором, бусинами и блочной сортировки представлены на листингах 3.1, 3.2, 3.3.

Листинг 3.1 – Алгоритм сортировки выбором

```
def selection_sort(arr, size):
    for i in range(size - 1):
        min_element = arr[i]
        min_index = i

        for j in range(i + 1, size):
            if arr[j] < min_element:
                min_element = arr[j]
                min_index = j

        arr[i], arr[min_index] = arr[min_index], arr[i]

    return arr
```

Листинг 3.2 – Алгоритм сортировки бусинами

```
def bead_sort(input_list, size):
    return_list = []
    transposed_list = [0] * max(input_list)
    for num in input_list:
        transposed_list[:num] = [n + 1 for n in
           transposed_list[:num]]
    for i in range(size):
        return_list.append(sum(n > i for n in transposed_list))
    return return_list
```

Листинг 3.3 – Алгоритм блочной сортировки

```
def bucket_sort(array, length):
    largest = max(array)
    size = largest / length

    buckets = [[] for i in range(length)]

    for i in range(length):
        index = int(array[i] / size)
        if index != length:
            buckets[index].append(array[i])
        else:
            buckets[length - 1].append(array[i])

    for i in range(len(array)):
        buckets[i] = sorted(buckets[i])

    result = []
    for i in range(length):
        result = result + buckets[i]

    return result
```

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

Вывод

Были реализованы функции алгоритмов сортировки выбором, бусинами и блочной. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

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

4.1 Технические характеристики

Проведем сравнительный анализ алгоритмов сортировки по затраченнному процессорному времени.

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система Windows 10 pro ;
- память 32 Гб;
- процессор 12th Gen Intel(R) Core(TM) i5-12400 2.50 ГГц.

Во время тестирования компьютер был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы.

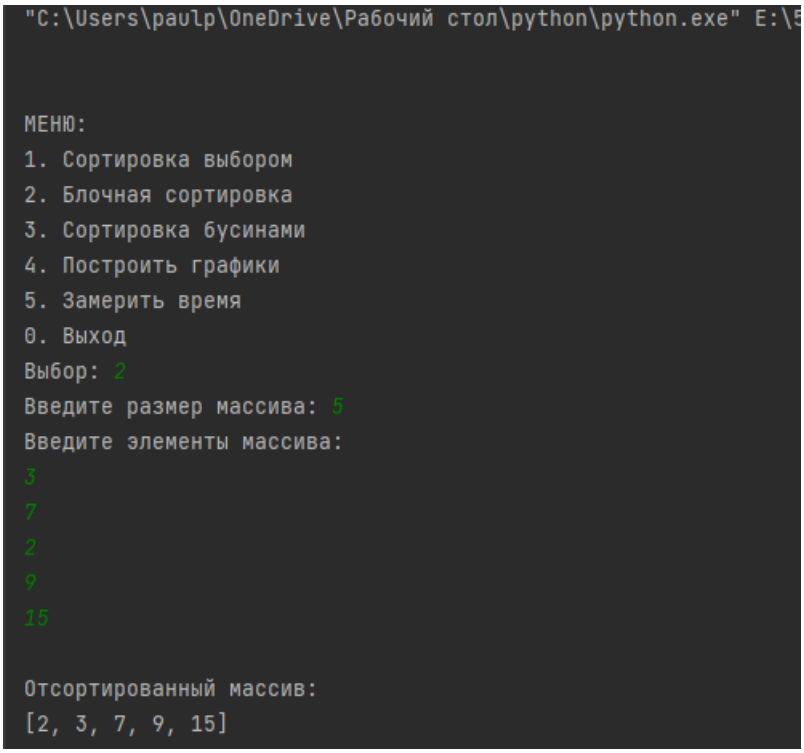

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Функция process_time из библиотеки time языка программирования 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 500 раз, 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	Блочная	Бусинами
100	0.0938	0.0312	0.4062
200	0.4062	0.0625	1.6875
300	0.9375	0.1250	3.6875
400	1.6562	0.2188	6.5625
500	2.6562	0.2812	10.3438
600	3.9062	0.3750	14.9062
700	5.1562	0.4688	20.5000
800	6.7812	0.5938	26.3750
900	8.6875	0.7500	33.2812
1000	10.7188	0.8750	41.2500

Таблица 4.2 – На входе отсортированный в обратном порядке массив

Размер	Выбором	Блочная	Бусинами
100	0.1250	0.0312	0.4375
200	0.5312	0.0625	1.6875
300	1.1250	0.1250	3.9375
400	2.0625	0.1875	7.0312
500	3.3438	0.2812	10.9688
600	4.7500	0.3438	15.3438
700	6.6562	0.4688	21.0000
800	8.9062	0.5938	26.6250
900	11.0000	0.7500	33.5312
1000	13.6875	0.8750	42.1562

Таблица 4.3 – На входе случайный массив

Размер	Выбором	Блочная	Бусинами
100	0.1562	0.0312	8.0625
200	0.4688	0.0625	16.8125
300	0.9375	0.1875	24.5312
400	1.6562	0.1875	33.0938
500	2.7812	0.2188	41.2812
600	4.0000	0.3438	49.9062
700	5.5938	0.2500	57.9062
800	7.7188	0.5625	66.3438
900	8.7812	0.6875	75.3438
1000	11.0312	0.9062	83.5625

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.

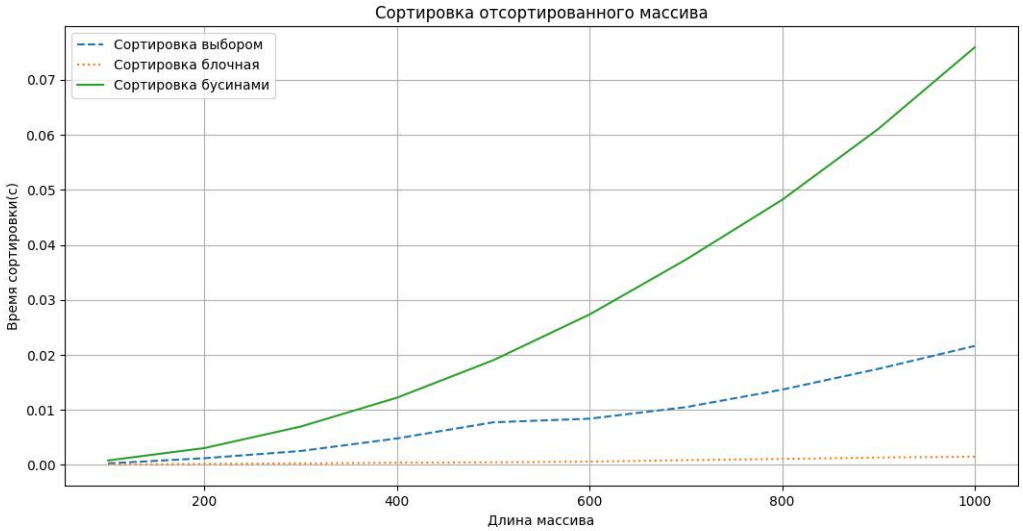

Рисунок 4.2 – На входе отсортированный массив

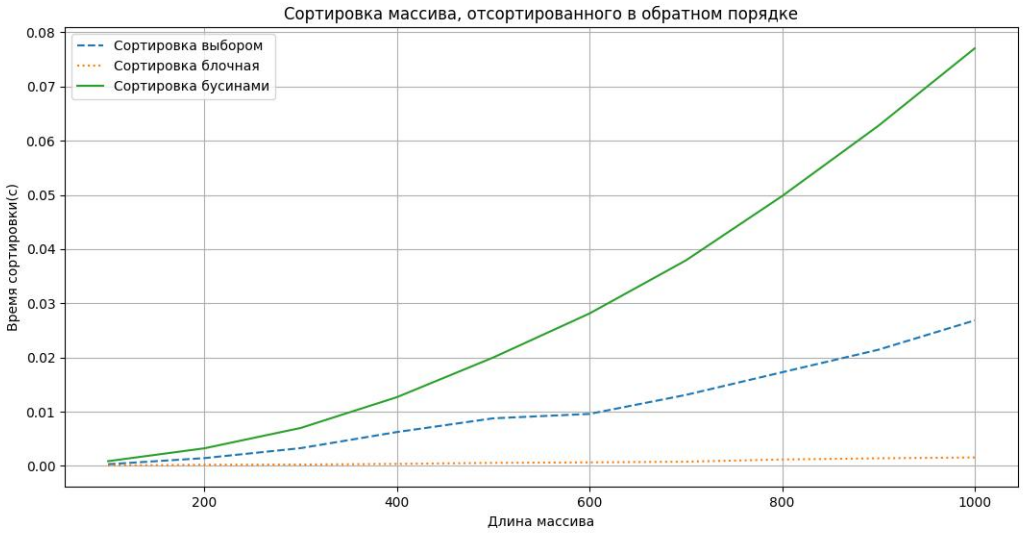

Рисунок 4.3 – На входе отсортированный в обратном порядке массив

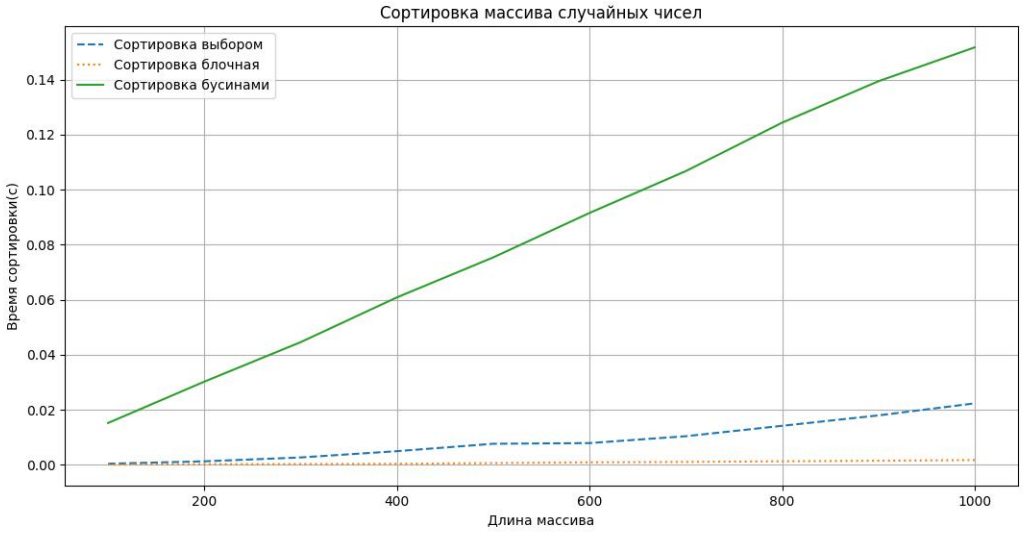

Рисунок 4.4 – На входе заполненный случайно массив

Вывод

Самой быстрой сортировкой из представленных является блочная сортировка. Затем по времени выполнения идет сортировка выбором. Самой же долгой сортировкой из представленных является сортировка бусинами.

Заключение

В результате исследования было выявлено, что самым быстрым алгоритмом сортировки из представленных является блочная сортировка, второй по быстродействию является сортировка выбором. Самой медленной является сортировка бусинами.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены следующие задачи:

- были изучены три алгоритма сортировки: выбором, бусинами и блочная;
- были разработаны и реализованы указанные алгоритмы;
- была протестирована реализация рассматриваемых алгоритмов;
- был проведен сравнительный анализ реализованных алгоритмов.

Таким образом, поставленная цель достигнута.

Список использованных источников

[1] Bead-sort: A natural sorting algorithm [Электронный ресурс]. Режим доступа: https://www.researchgate.net/publication/37987842_Bead--Sort_A_Natural_Sorting_Algorithm.

[2] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org.

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