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: Трудоемкость алгоритмов сортировки

Студент Казакова Э.М.

Группа ИУ7-56Б

Оценка (баллы) _____

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

Москва — 2021 г.

Содержание

Введение

В настоящее время необходимо сортировать большие объемы данных. Для этой цели существуют алгоритмы сортировки, которые упорядочивают элементы в списке. [1] Сортировкой называют процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Определенный порядок (например, упорядочение в алфавитном порядке, по возрастанию или убыванию количественных характеристик, по классам, типам и.т.п.) в последовательности объектов необходимо для удобства работы с этим объектом.

Целью данной лабораторной работы является изучение и реализация алгоритмов сортировки, исследование их трудоемкости.

Задачи данной лабораторной работы:

1. изучить алгоритмы сортировки пузырьком, вставками, выбором;
2. реализовать алгоритмы сортировки пузырьком, вставками, выбором;
3. дать оценку трудоёмкости (для двух алгоритмов сделать вывод трудоёмкости);
4. провести замеры процессорного времени работы для лучшего, худшего и произвольного случая.

1 Аналитическая часть

В данной части будут рассмотрены основные теоретические аспекты, связанные с алгоритмами сортировок пузырьком, вставками, выбором.

1.1 Сортировка вставками

Сортировка вставками[2] — это простой алгоритм сортировки. Суть его заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Стоит отметить что массив из 1-го элемента считается отсортированным.

Сортировка вставками наиболее эффективна когда массив уже частично отсортирован и когда элементов массива не много. Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части.

1.2 Сортировка выбором

Один из самых простых методов сортировки работает следующим образом: находим наименьший элемент в массиве и обмениваем его с элементом находящимся на первом месте. Потом повторяем процесс со второй позиции в файле и найденный элемент обмениваем со вторым элементном и так далее пока весь массив не будет отсортирован. Этот метод называется сортировка выбором, поскольку он работает, циклически выбирая наименьший из оставшихся элементов. Главное отличие сортировки выбором от сортировки вставками: в сортировке вставками мы извлекаем из неотсортированной части массива любой элемент и вставляем его на своё место в отсортированной части. В сортировке выбором мы целенаправленно ищем максимальный элемент(или минимальный), которым дополняем отсортированную часть массива. Во вставках мы ищем куда вставить очередной элемент, а в выборе — мы заранее уже знаем в какое место поставим, но при этом требуется найти элемент, этому месту соответствующий.

1.3 Сортировка пузырьком

Алгоритм проходит по массиву n-1 раз до тех пор, пока массив не будет полностью отсортирован. В каждом проходе элементы попарно сраниваются и, при необходимости, меняются местами. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент ставится на своё место в конец неотсортированного массива. Таким образом наибольшие элементы "всплывают"как пузырёк.

1.4 Вычисление сложности алгоритма

В рамках данной работы используется следующая модель вычислений:

1. базовые операции имеют трудоемкость 1 (<, >, =, <=, =>, ==, +, -, *, /, %, &, +=, -=, *=, /=, [ ]);

2. операторы if, else if имеют трудоемкость $F_{if} = F_{body} + F_{chek}$, F_{body} - трудоемкость операций тела оператора, F_{chek} - трудоемкость проверки условия;

3. оператор else имеет трудоемкость F_{body};

4. оператор for имеет трудоемкость $F_{for} = 2 + N \cdot (F_{body} + F_{chek})$, где F_{body} – трудоемкость операций в теле цикла.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов сортировок пузырьком, вставками и выбором.

2.1 Схемы алгоритмов

На рисунке 1 представлена схема алгоритма сортировки пузырьком.

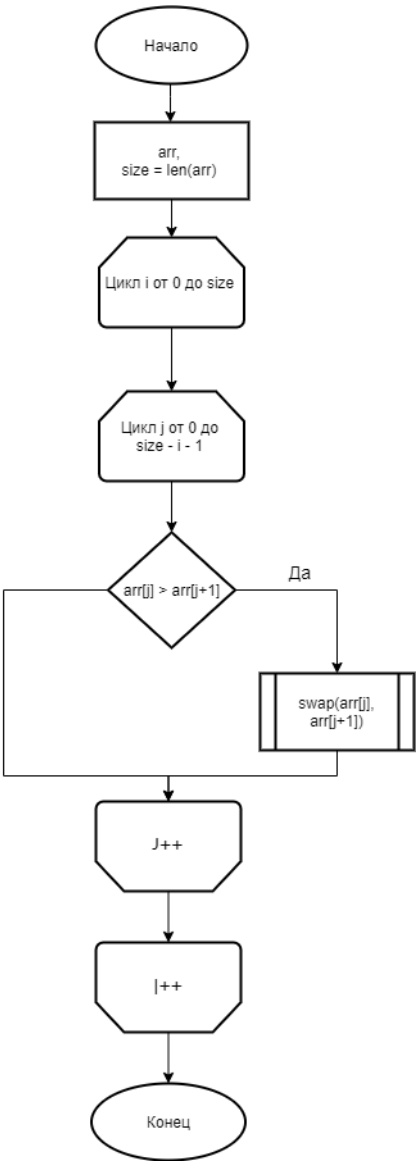

Рис. 1: Схема алгоритма сортировки пузырьком

На рисунке 2 представлена схема алгоритма сортировки вставками.

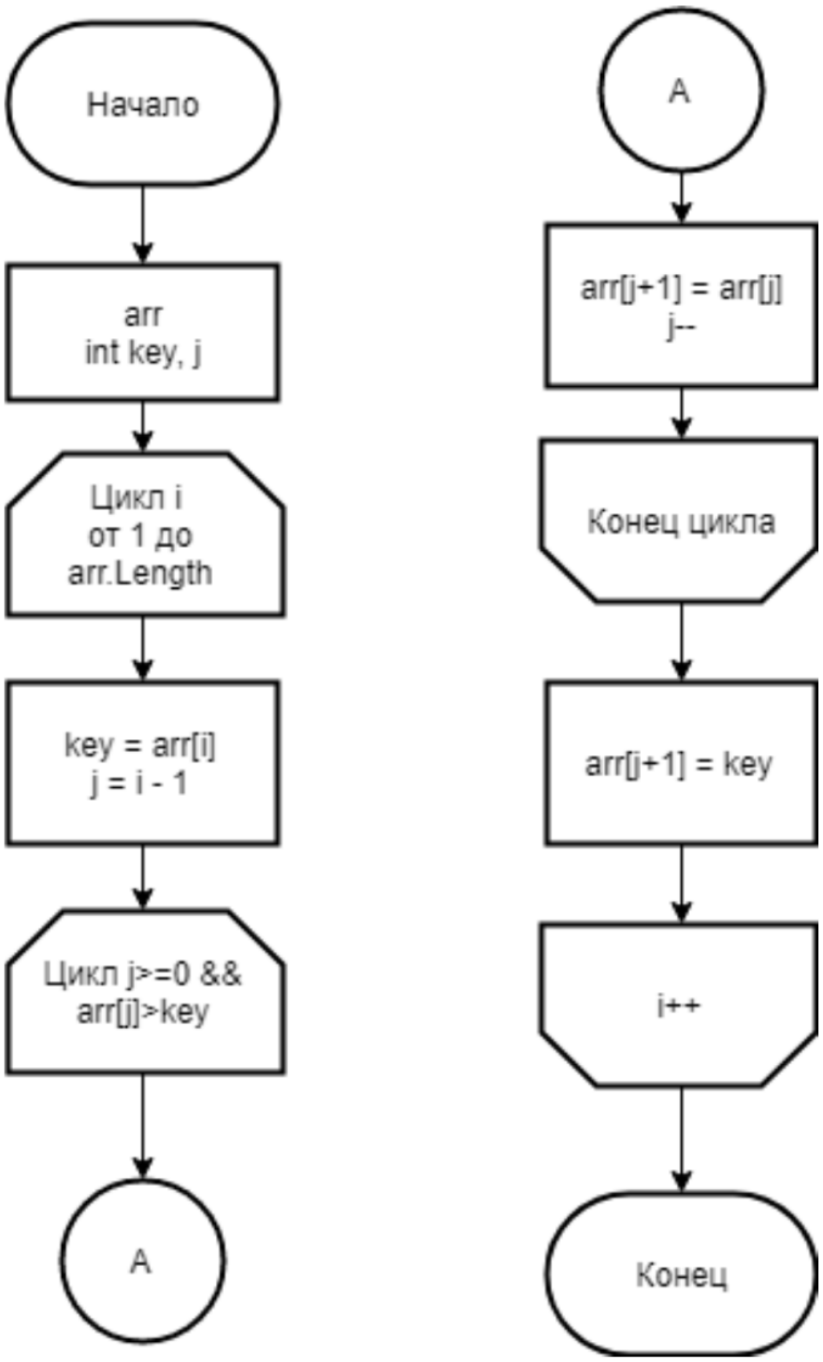

Рис. 2: Схема алгоритма сортировки вставками

На рисунке 3 представлена схема алгоритма сортировки выбором.

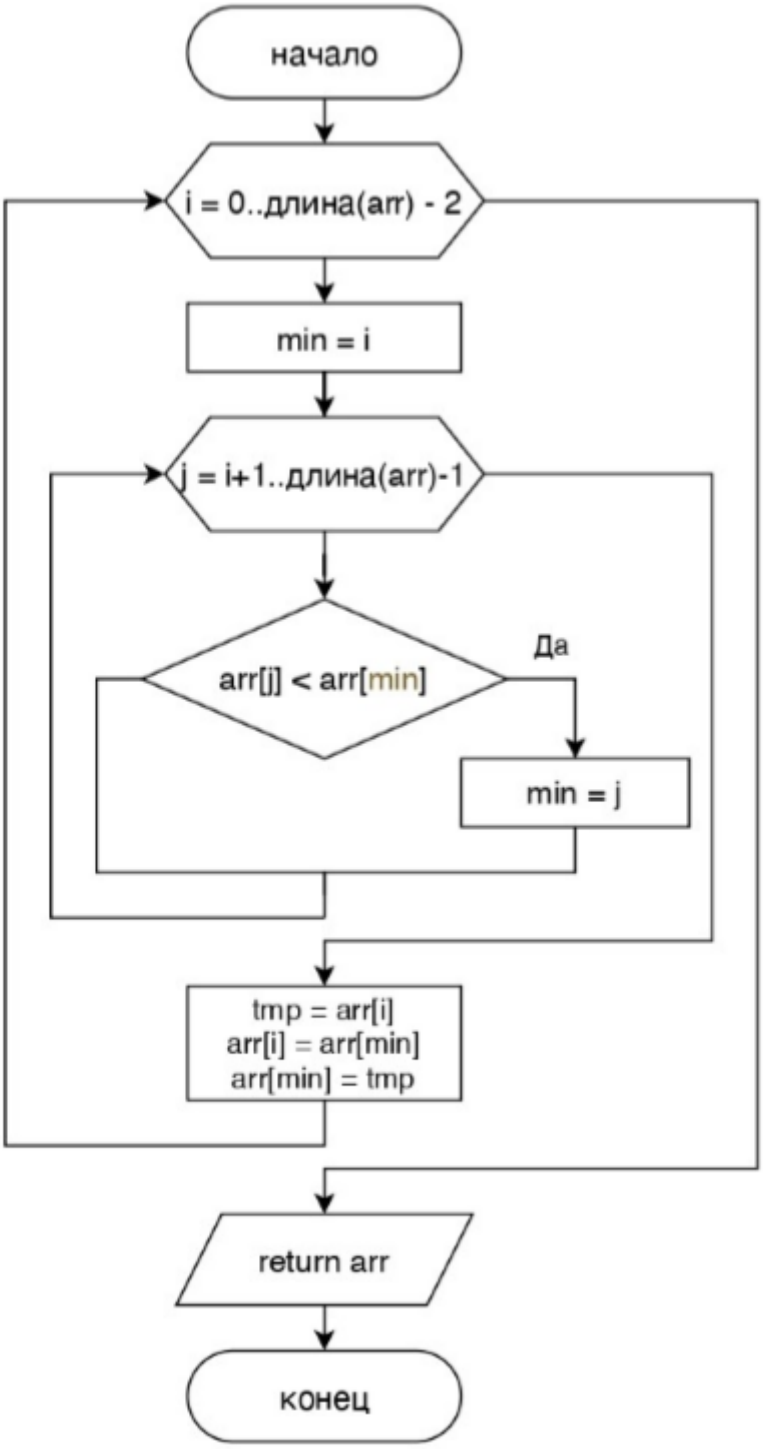

Рис. 3: Схема алгоритма сортировки выбором

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность ввода массива, на выходе пользователь должен получить результат сортировки массива, произведенной тремя алгоритмами. Также ПО должно обеспечить вывод замеров времени работы каждого из алгоритмов в худшем, лучшем и произвольном случаях

3.2 Средства реализации

В данной работе используется язык программирования Python, так как ЯП позволяет написать программу за кратчайшее время.В качестве среды разработки выбрана IDLE. Для замеров времени была выбран метод process_time() модуля time[3], он возвращает значение (в долях секунды) системного процессорного времени текущего процесса.

3.3 Листинг кода

В листингах 3.1 - 3.3 представлена реализация алгоритма сортировки пузырьком, вставками и выбором.

Листинг 3.1 – Алгоритм сортировки пузырьком

```
1         def bubble_sort(arr):
2                 start_time = time.process_time()
3                 for i in range(len(arr)):
4                         for j in range(0, len(arr)-i-1):
5                                 if arr[j] > arr[j+1]:
6                                         arr[j], arr[j+1] = arr[j+1], arr[j]
7                 t = time.process_time() - start_time
8                 return t
```

Листинг 3.2 – Алгоритм сортировки вставками

```
1         def insertion_sort(arr):
2                 start_time = time.process_time()
3                 for i in range(1, len(arr)):
4                         j = i - 1
5                         key = arr[i]
6                         while j >= 0 and arr[j] > key:
7                                 arr[j + 1] = arr[j]
8                                 j -= 1
9                         arr[j + 1] = key
10                t = time.process_time() - start_time
11                return t
```

Листинг 3.3 – Алгоритм сортировки выбором

```
1         def selection_sort(arr):
2                 start_time = time.process_time()
3                 for i in range(0, len(arr) - 1):
4                         min = i
5                         for j in range(i + 1, len(arr)):
6                                 if arr[j] < arr[min]:
7                                         min = j
8                         arr[i], arr[min] = arr[min], arr[i]
9                 t = time.process_time() - start_time
10                return t
```

3.4 Тестирование

В данном разделе будут показаны результаты тестирования Всего было реализовано 5 тестовых случаев:

1. размер массива равен 0;
2. сравнение работы все трех алгоритмов на уже отсортированных значениях массива;
3. сравнение работы все трех алгоритмов на обратно отсортированных значениях массива;
4. сравнение работы все трех алгоритмов на случайных значениях массива;

На рисунках 4-7 представлены результаты работы программы при вводе различных входных тестовых данных.

```
Введите массив:


Результат сортировки пузырьком с флагом:
[]

Результат сортировки вставками:
[]

Результат сортировки выбором:
[]
```

Рис. 4: Результат программы при нулевом размере массива

```
================ RESTART: C:\Users\Элиза\Desktop\AA\lab3\lab3.py ===
Введите массив:
1 2 3 4 5 6 7

Результат сортировки пузырьком с флагом:
[1, 2, 3, 4, 5, 6, 7]

Результат сортировки вставками:
[1, 2, 3, 4, 5, 6, 7]

Результат сортировки выбором:
[1, 2, 3, 4, 5, 6, 7]
>>>
```

Рис. 5: Результат программы при уже отсортированных значениях массива

```
================ RESTART: C:\Users\Элиза\Desktop\AA\lab3\lab3.py =======
Введите массив:
5 4 3 2 1

Результат сортировки пузырьком с флагом:
[1, 2, 3, 4, 5]

Результат сортировки вставками:
[1, 2, 3, 4, 5]

Результат сортировки выбором:
[1, 2, 3, 4, 5]
>>>
```

Рис. 6: Результат программы при обратно отсортированных значениях массива

```
================ RESTART: C:\Users\Элиза\Desktop\AA\lab3\lab3.py ===
Введите массив:
-2 4 1 75 6 -10

Результат сортировки пузырьком с флагом:
[-10, -2, 1, 4, 6, 75]

Результат сортировки вставками:
[-10, -2, 1, 4, 6, 75]

Результат сортировки выбором:
[-10, -2, 1, 4, 6, 75]
>>>
```

Рис. 7: Результат программы при случайных значениях массива

4 Экспериментальная часть

В данной части производится экспериментальное сравнение работы трех реализованных алгоритмов (зависимость времени выполнения от размеров массива и лучшего, произвольного или худшего способа заполнения массива).

4.1 Сравнение времени работы

На графиках (рисунки 8 - 10) представлено сравнение времени работы алгоритмов сортировки на массивах разных размеров. Для лучшего случая массивы заполняются в порядке возрастания числами от -1000 до 1000, для худшего случая - в порядке убывания от -1000 до 1000, для произвольного случая - случайно сгенерированными числами в диапазоне от -1000 до 1000. Для построения графиков генерировались массивы размерами от 0 до 6000.

На рисунках 4-7 представлены результаты работы программы при вводе различных входных тестовых данных.

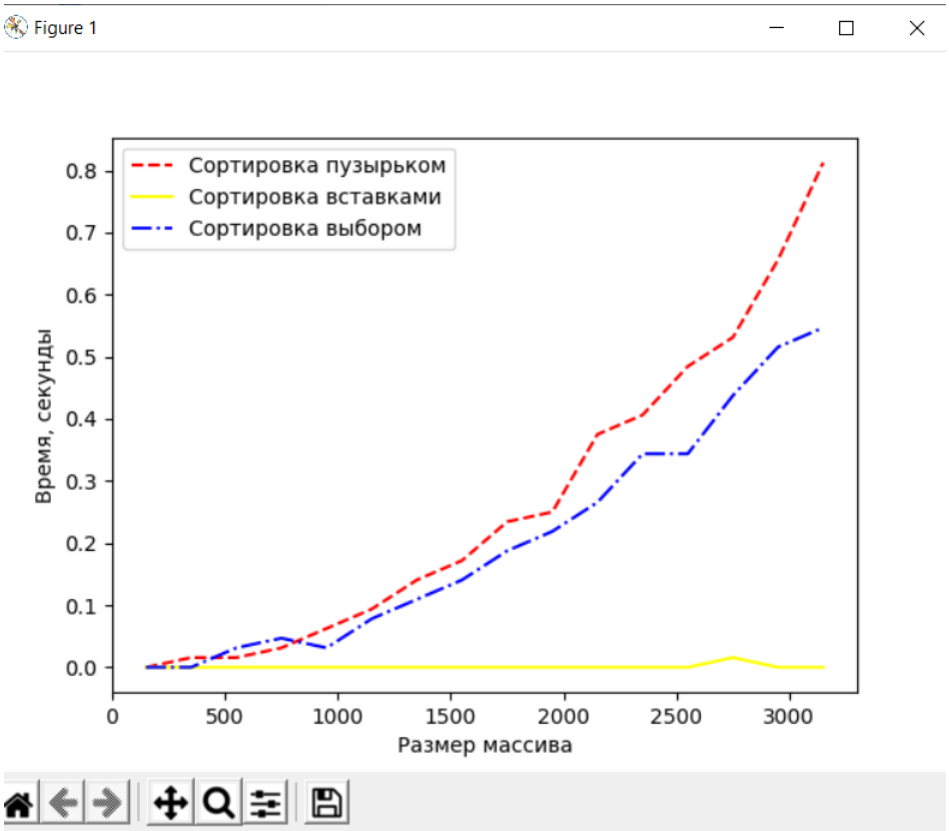

Рис. 8: Сравнение по времени работы реализации алгоритмов сортировки пузырьком, вставками, выбором (лучший случай)

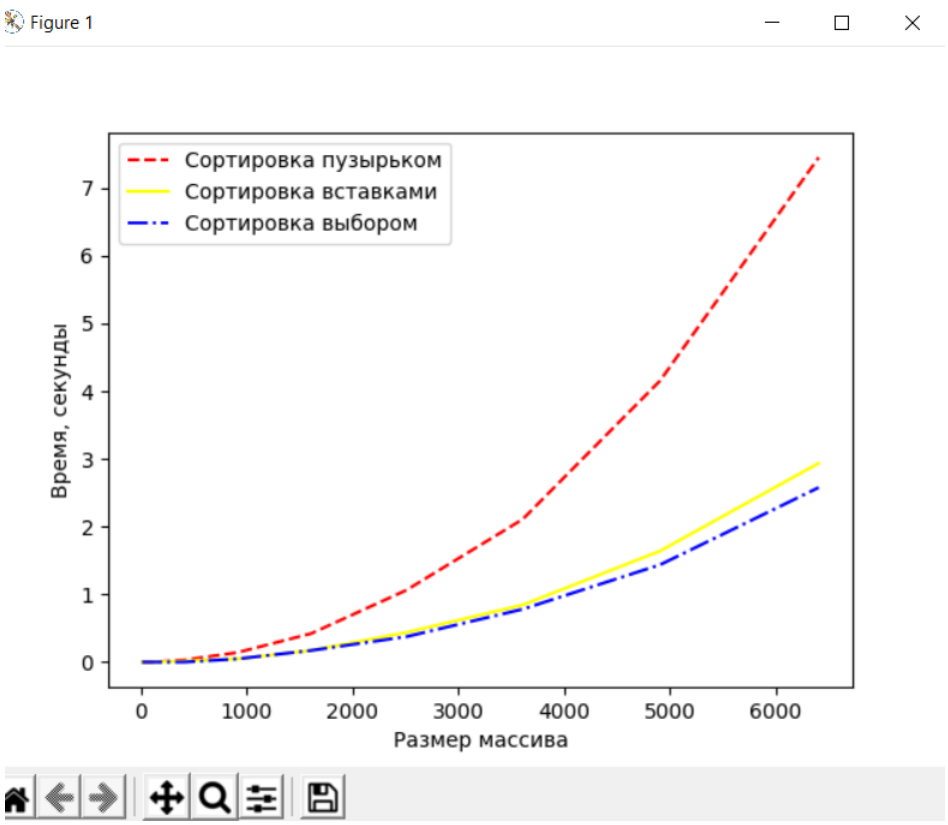

Рис. 9: Сравнение по времени работы реализации алгоритмов сортировки пузырьком, вставками, выбором (произвольный случай)

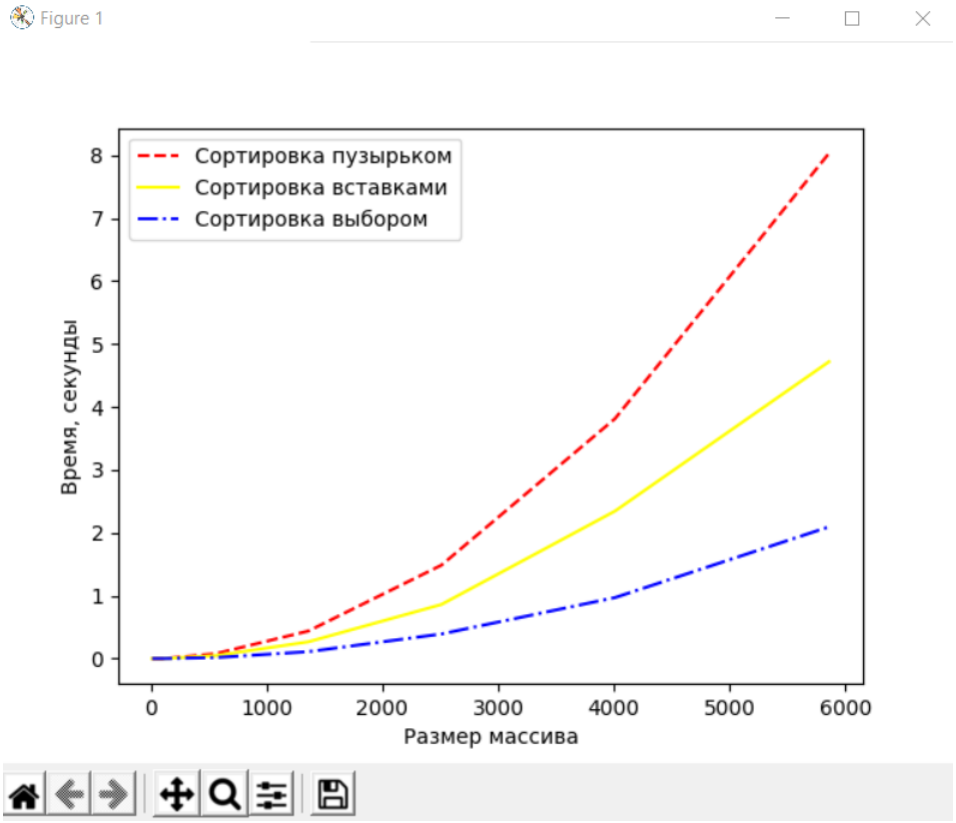

Рис. 10: Сравнение по времени работы реализации алгоритмов сортировки пузырьком, вставками, выбором (худший случай)

4.2 Оценка трудоемкости алгоритмов сортировок

Далее будет приведены оценки трудоемкости алгоритмов сортировки.

1. Алгоритм сортировки пузырьком.
   Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла.
   Трудоемкость: 1 + 1 + 2 + n * (2 + 7 + 1 + 3) = 13n + 4 = *O(n)*.
   Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен.
   Трудоемкость: 1 + 1 + 2 + n * (n * (7 + 5 + 1 + 3) + 1 + 1) = $16n^2$ + 2n + 4 = *O(n^2)*.

2. Алгоритм сортировки вставками.
   Лучший случай: Массив отсортирован. При этом все внутренние циклы состоят всего из одной итерации.
   Трудоемкость: T(n) = 3n + ((2 + 2 + 4 + 2) * (n - 1)) = 3n + 10(n - 1) = 13n - 10 = *O(n)*.
   Худший случай: Массив отсортирован в обратном порядке; каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.
   Трудоемкость: T(n) = 3n + (2 + 2)(n - 1) + 4($\frac{n(n+1)}{2}$) - 1) + 5$\frac{n(n-1)}{2}$ + 3(n-1) = 3n + 4n - 4 + $2n^2$ + 2n - 4 + $2.5n^2$ - 2.5n + 3n - 3 = $4.5n^2$ + 9.5n - 11 = *O(n^2)*.

3. Алгоритм сортировки выбором.
   Лучший случай: $O(N^2)$
   Худший случай: $O(N^2)$.

4.3 Выводы

В результате проведенного эксперимента был получен следующий вывод: в лучшем, худшем, произвольном случае сортировка пузырьком оказалась самой медленной. Сортировка вставками в лучшем случае работает быстрее всего. В худшем случае сортировка выбором является самой быстрой. В произвольном случае время работы сортировок вставками и выбором сопоставимо.

Заключение

В ходе лабораторной работе были исследованы алгоритмы сортировок: выбором, пузырьком и вставками. При выполнении лабораторной работе цель была достигнута: были изучены и реализованы алгоритмы сортировки, исследована их трудоемкость.
Также были выполнены следующие задачи:

1. были изучены алгоритмы сортировки пузырьком, вставками, выбором;
2. были реализованы алгоритмы сортировки пузырьком, вставками,выбором;
3. была дана оценка трудоёмкости в лучшем, произвольном и худшем случае;
4. были проведены замеры процессорного времени работы для лучшего, худшего и произвольного случая.

Список литературы

[1] Глушко. Алгоритм сортировки [Электронный ресурс]. - Режим доступа: https://works.doklad.ru/view/MeaUSqCgyps.html.(дата обращения: 23.01.2021)

[2] В мире алгоритмов: Сортировка Вставками [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/181271/ (дата обращения: 23.01.2021)

[3] Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.