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 (ИУ7)**

ОТЧЕТ

по лабораторной работе № 3

Название: Алгоритмы сортировки массива

Дисциплина: Анализ Алгоритмов

Студент ИУ7-53Б _____ В.П. Федоров
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Ю.В. Строганов
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Сортировкой называется процесс упорядочивания множества объектов по какому-либо признаку.

Алгоритм сортировки - это алгоритм для упорядочения элементов в списке.

В данной лабораторной работе нам требуется изучить алгоритмы сортировки и их трудоемкости.

Цели лабораторной работы:

1. реализовать 3 выбранных алгоритма сортировки;
2. рассчитать трудоемкость каждого из алгоритмов сортировки;
3. провести временное тестирование алгоритмов сортировки.

1. Аналитическая часть

В данном разделе будут рассмотрены описания алгоритмов сортировки массива.

1.1 Описание алгоритмов

Существует огромное множества разнообразных алгоритмов сортировки. Все они отличаются трудоемкостью и скоростью работы.
В данной работе будут рассмотрены следующие алгоритмы:

1. сортировка пузырьком;
2. сортировка вставками;
3. быстрая сортировка.

1.1.1 Алгоритм сортировки “пузырьком”

Сортировка пузырьком — один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять числа местами, если предыдущее оказывается больше последующего. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале.

Этот алгоритм считается учебным и почти не применяется на практике из-за низкой эффективности: он медленно работает на тестах, в которых маленькие элементы (их называют «черепахами») стоят в конце массива. Однако на нём основаны многие другие методы, например, шейкерная сортировка и сортировка расчёской[2].

1.1.2 Алгоритм быстрой сортировки

Этот алгоритм состоит из трех шагов. Сначала из массива нужно выбрать один элемент — его обычно называют опорным. Затем другие элементы в массиве перераспределяют так, чтобы элементы меньше опорного оказались до него, а большие или равные — после. А дальше рекурсивно применяют первые два шага к подмассивам справа и слева от опорного значения.

Быструю сортировку изобрели в 1960 году для машинного перевода: тогда словари хранились на магнитных лентах, а сортировка слов обрабатываемого текста позволяла получить переводы за один прогон ленты, без перемотки назад [2].

1.1.3 Алгоритм сортировки “вставками”

При сортировке вставками массив постепенно перебирается слева направо. При этом каждый последующий элемент размещается так, чтобы он оказался между ближайшими элементами с минимальным и максимальным значением.

1.1.4 Вывод

Были рассмотрены алгоритмы сортировки “пузырьком”, “вставками”, “быстрой”, которые по разному обходят массив, и поэтому сортируют по разному.

2. Конструкторская часть

В данной части будут рассмотрены схемы алгоритмов.

2.1 Разработка алгоритмов

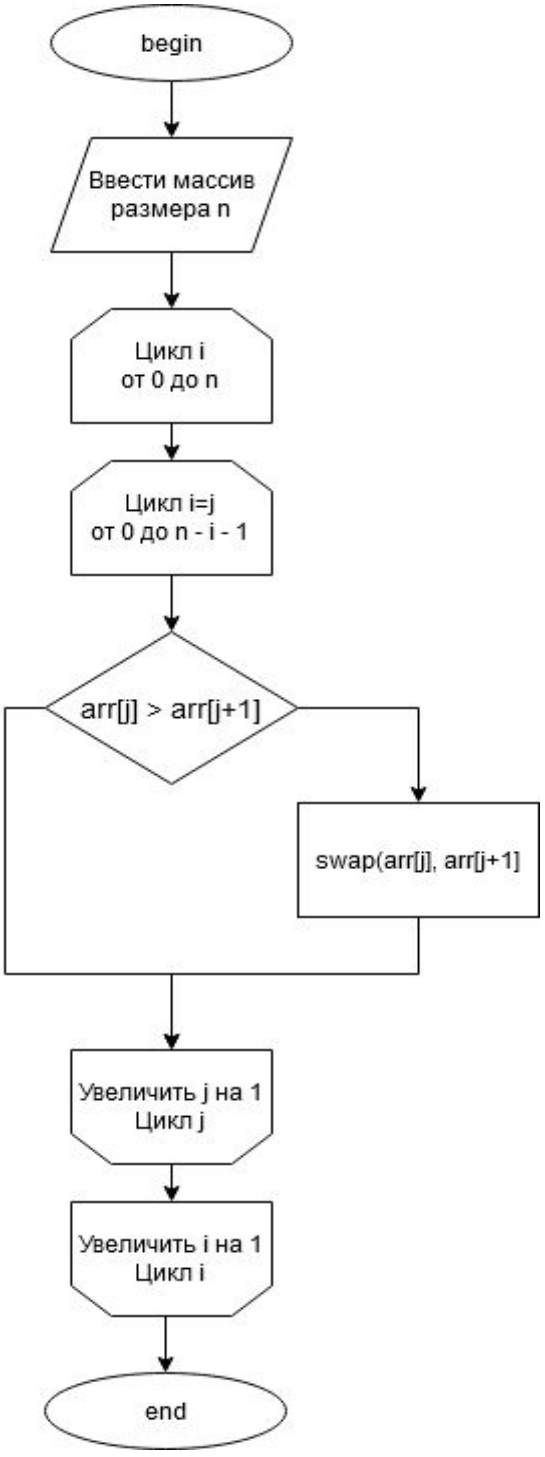


Рис.2.1.1: Схема алгоритма сортировки пузырьком

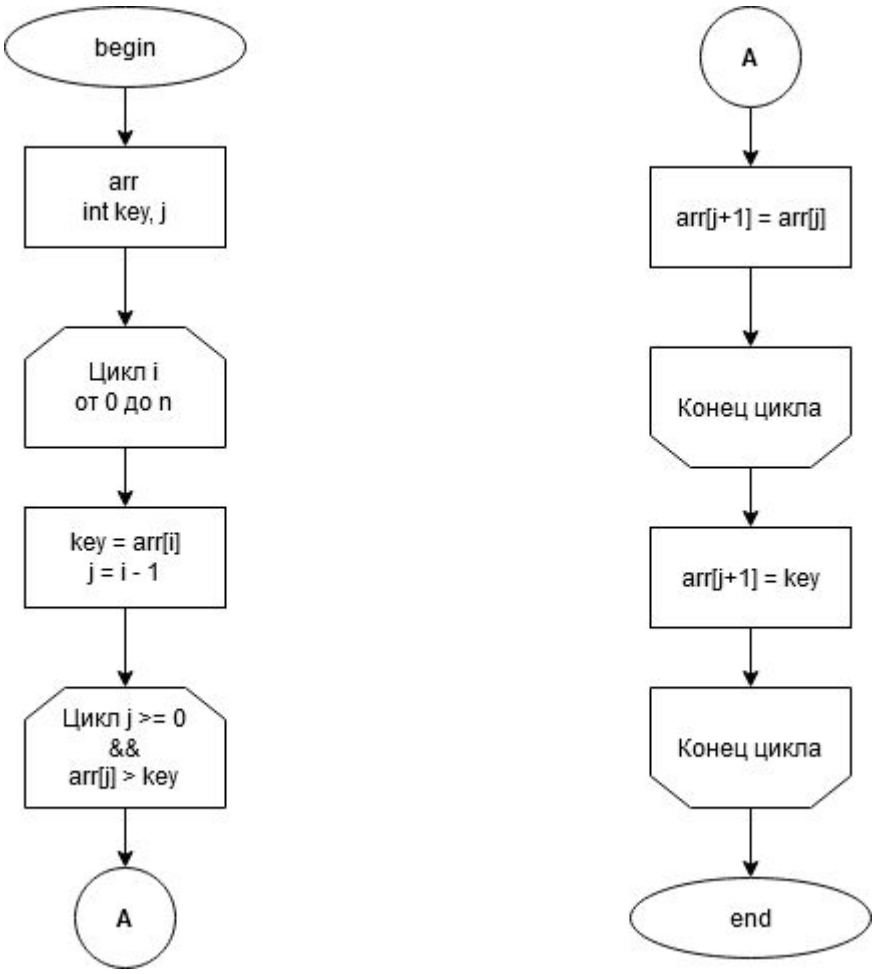


Рис.2.1.2 Схема алгоритма сортировки вставками

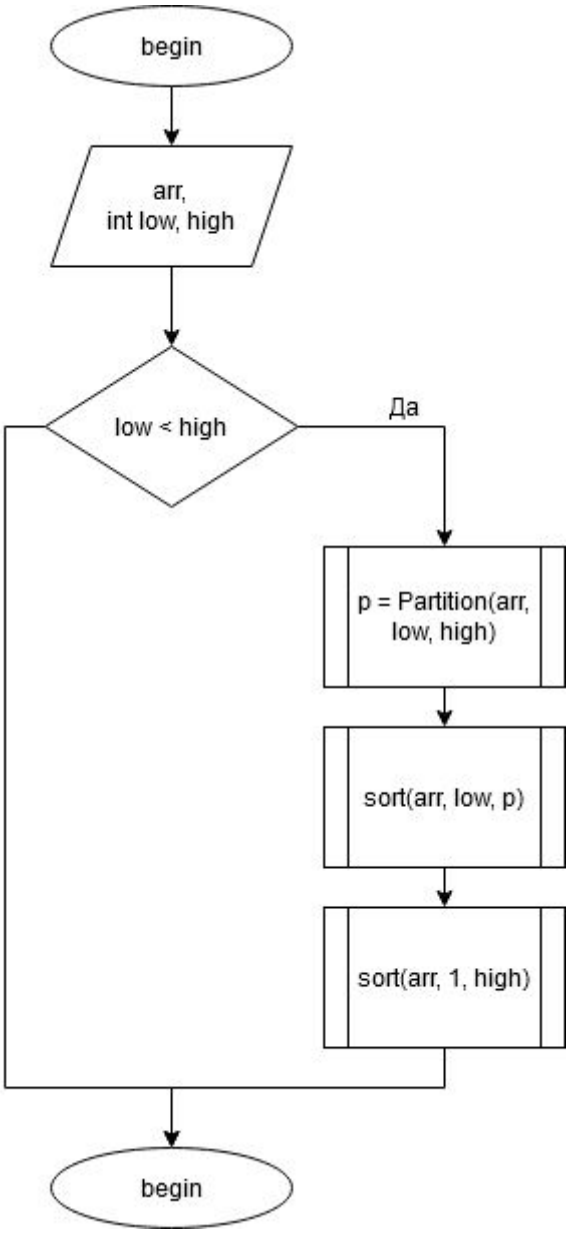


Рис.2.1.3 Схема алгоритма быстрой сортировки

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1) Базовые операции стоимостью 1:
   +, -, *, /, =, ==, <=, >=, !=, +=, [];

2) оценка трудоемкости цикла
   for от 0 до N с шагом 1

$$F_{for} = a + N \cdot \left(a + F_{body}\right), \text{ где } F_{body} - \text{тело цикла}, a - \text{условие}$$

3) стоимость условного перехода примем за 0, стоимость вычисления условия остается

Оценим трудоемкость алгоритмов по коду программы

2.2.1 Трудоемкость алгоритма сортировки пузырьком

Применим вышеописанную модель для оценки трудоемкости алгоритма по коду(листинги кода можно найти в разделе 3).

Лучший случай: массив отсортирован.
$1 + 6\,N + 9N^2 = O(N^2)$

Худший случай: массив отсортирован в обратном порядке.
$1 + 6N + 18N^2 = O(N^2)$

2.2.2 Трудоемкость алгоритма сортировки вставками

Лучший случай: массив отсортирован.
При этом все внутренние циклы состоят всего из одной итерации.
Трудоемкость: 3N + ((2 + 2 + 4 + 2) * (N - 1) = 3N + (10N - 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:

$$3N + 4 \cdot (N-1) + 4 \cdot \left(\frac{N \cdot (N+1)}{2} - 1\right) + 5 \cdot \frac{N \cdot (N-1)}{2} + 3 \cdot (N-1) =$$

$$= 3 \cdot N + 4 \cdot N - 4 + 2 \cdot N^2 + 2 \cdot N - 4 + 2.5N^2 + 3 \cdot N - 3 = 4.5^2 + 9.5N - 11 = O\left(N^2\right)$$

2.2.3 Трудоемкость алгоритма быстрой сортировки

Лучший случай:
Сбалансированное дерево вызовов $O(N \cdot \log(n))$. В лучшем случае процедура PARTITION приводит к двум подзадачам, размер каждой из которых не превышает $\frac{n}{2}$, поскольку размер одной из них равен $\frac{n}{2}$, а второй $\frac{n}{2} - 1$. В такой ситуации быстрая сортировка быстрая сортировка работает намного производительнее, и время ее работы описывается следующим рекуррентным соотношением: $T(N) = 2 \cdot T\left(\frac{N}{2}\right) + O(N)$, где мы не обращаем внимание на неточность, связанную с игнорированием функций “пол” и “потолок”, и вычитаем 1. Это рекуррентное соотношение имеет решение; T(N) = O(Nlgn). При сбалансированности двух частей разбиения на каждом уровне рекурсии мы получаем асимптотически более быстрый алгоритм.

Фактически любое разбиение, характеризующееся конечной константой пропорциональности, приводит к образованию дерева рекурсии высотой O(lgn) со стоимостью каждого уровня, равной O(n). Следовательно, при любой постоянной пропорции разбиения полное время работы быстрой сортировки составляет O(nlgn).

Худший случай:

Несбалансированное дерево O(n^2). Поскольку рекурсивный вызов процедуры разбиения, на вход которой подается массив размеров 0, приводит к немедленному возврату из этой процедуры без выполнения каких-либо операций. T(0) = O(1). Таким образом, рекуррентное соотношение, описывающее время работы процедуры в указанном случае, записывается следующим образом: T(n) = T(n-1) + T(0) + O(n) = T(n-1) + O(n). Интуитивно понятно, что при суммировании промежутков времени, затрачиваемых на каждый уровень рекурсии, получается арифметическая прогрессия, что приводит к результату O(n^2).

2.3 Вывод

Были построены схемы алгоритмов и посчитаны их трудоемкости.

3. Технологическая часть

В этой части будут рассмотрены требования к ПО, средства реализации, описание тестирования и листинг кода.

3.1 Требования к программному обеспечению

На вход получаем: массив чисел и длину массива.
На выходе имеем: Отсортированный массив.

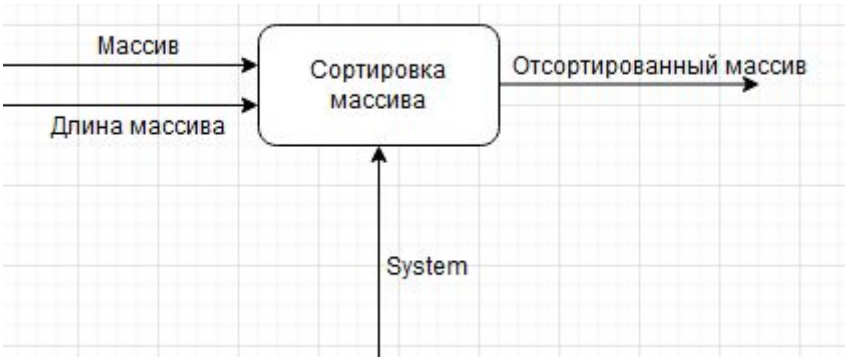


Рис.3.1.1: IDEF0-диаграмма, описывающая алгоритм сортировки массива.

3.2 Средства реализации

Был использован язык программирования Java и среда разработки Intellij IDEA, так как ими я собираюсь пользоваться в дальнейшей работе.
С помощью Python и matplotlib были получены графические иллюстрации для экспериментального раздела.

3.3 Листинги кода

Ниже приведен листинг трех алгоритмов сортировки.

Листинг 3.3.1 Алгоритм сортировки пузырьком

```
public class BubbleSort {
  public static void sort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n-1; i++)
      for (int j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1]) {
          int temp = arr[j];
          arr[j] = arr[j+1];
          arr[j+1] = temp;
        }
  }
}
```

Листинг 3.3.2 Алгоритм сортировки вставками

```
public class InsertionSort {
  public static void sort(int[] arr) {
    int n = arr.length;
    for (int j = 1; j < n; j++) {
      int key = arr[j];
      int i = j - 1;
      while ((i > -1) && (arr[i] > key)) {
        arr[i + 1] = arr[i];
        i--;
      }
      arr[i + 1] = key;
    }
  }
}
```

Листинг 3.3.3 Алгоритм быстрой сортировки

```
public class QuickSort {

  public static void sort(int[] arr, int low, int high) {
    if (low >= high)
      return;

    int middle = low + (high - low) / 2;
    int opora = arr[middle];

    int i = low, j = high;
    while (i <= j) {
      while (arr[i] < opora) {
        i++;
      }

      while (arr[j] > opora) {
        j--;
      }

      if (i <= j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
        j--;
      }
    }

    if (low < j)
      sort(arr, low, j);

    if (high > i)
      sort(arr, i, high);
  }
}
```

3.4 Тесты

Тестирование проводилось с помощью модуля JUnit. Сначала каждый алгоритм тестировался по отдельности на заранее заготовленном наборе тестовых данных. После этого генерировались массивы случайной длины случайных целых чисел в диапазоне от -100 до 100 размером от 0 до 1000. Далее каждый массив сортировался каждой сортировкой. Результаты сравнивались отдельно. Если массивы были не идентичны, это означало бы появление ошибки. Все тесты были пройдены успешно. Ошибок выявлено не было. Использованный набор тестовых данных приведен в таблице 3.4.1

Таблица 3.4.1 Тестовые данные

Исходный массив	Ожидаемый результат
1, 1	1, 1
1	1
-1, -1, -1	-1, -1, -1
1, 2, 3	1, 2, 3
3, 2, 1	1, 2, 3
1, 2, 1	1, 1, 2
1, 1, 2, 2	1, 1, 2, 2
2, 1, 2, 1	1, 1, 2, 2
-1, -2, -3	-3, -2, -1
-2, - 2, -1	-2, -2, -1
-1, -2, -1, -2	-2, -2, -1, -1
-2, -2, 1, 1	-2, -2, 1, 1

3.5 Вывод

В данной части были представлены реализации алгоритмов сортировки пузырьком, сортировки вставками и быстрой сортировки.

4. Экспериментальная часть

Были проведены замеры времени работы каждого алгоритма.

4.1 Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

1. Сравнение времени работы алгоритмов сортировок в лучших, худших и средних случаях на массивах размерности от 0 до 1000 с шагом 100.

Использовались три типа массивов.

1. Отсортированные по возрастанию
2. Отсортированные по убыванию
3. Заполненные случайным образом

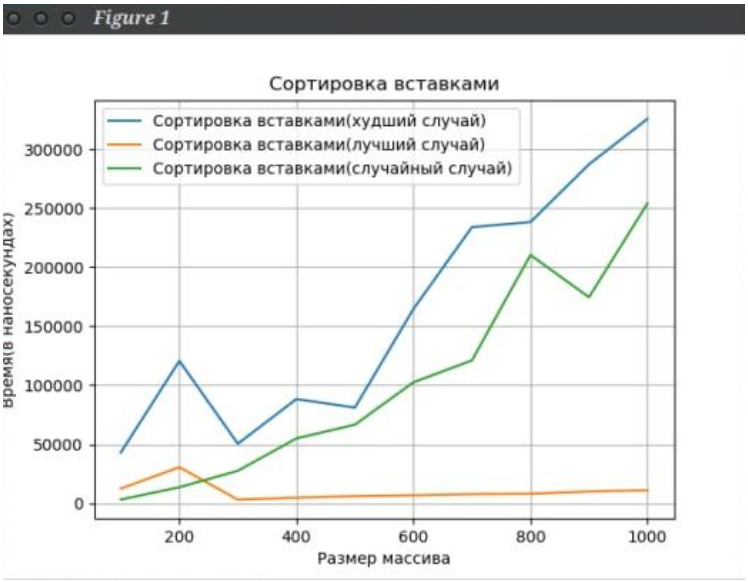
4.2 Сравнительный анализ на материале экспериментальных данных

Результаты замеров времени для алгоритма сортировки вставками для лучшего случая (Лучший), для худшего случая (Худший) и для произвольного случая (Произвольный) приведены в таблице 4.2.1. Здесь и далее все указанное время измеряется в наносекундах. На графиках изображена зависимость времени работы алгоритма сортировок от длины входного массива. Данные из таблицы изображены на рисунке 4.2.2

Таблица 4.2.1 Результаты замеров времени для алгоритма сортировки вставками

Размер массива	Лучший, нс	Худший, нс	Произвольный, нс
100	12386	43013	3174
200	30586	120569	13590
300	3100	50441	27533
400	4776	88270	54948
500	6063	81033	66561
600	6819	164559	102464
700	7744	233871	120998
800	8079	238153	210273
900	10025	287032	174472
1000	11017	325556	253992

Рис. 4.2.2: Результаты замеров времени для алгоритма сортировки вставками

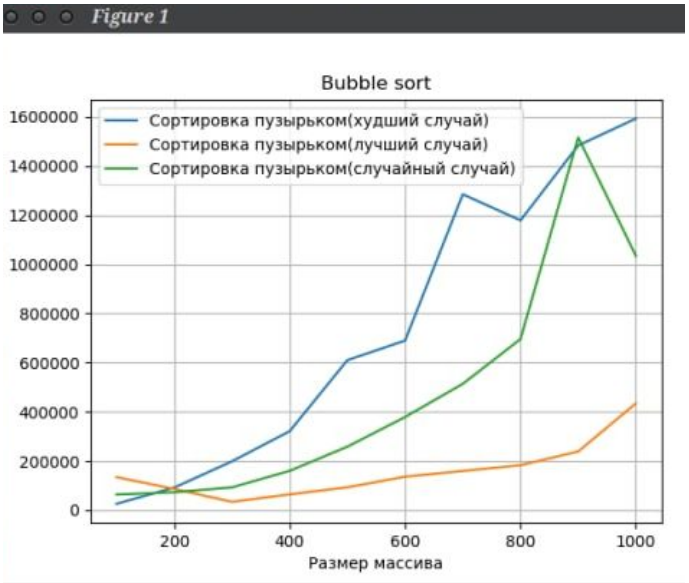


Результаты замеров времени для алгоритма сортировки пузырьком для лучшего случая (Лучший), для худшего случая (Худший) и для произвольного случая (Произвольный) приведены в таблице 4.2.3. Данные из таблицы изображены на рисунке 4.2.4.

Таблица 4.2.3 Результаты замеров времени для алгоритма сортировки пузырьком

Размер массива	Лучший, нс	Худший, нс	Произвольный, нс
100	134047	25597	63471
200	86123	91566	72546
300	33634	198427	92398
400	64048	320920	159608
500	92760	610066	258036
600	135808	689737	378696
700	159189	1284895	513669
800	182603	1178824	695857
900	238499	1483535	1516416
1000	434417	1593130	1034003

Рис. 4.2.4: Результаты замеров времени для алгоритма сортировки пузырьком

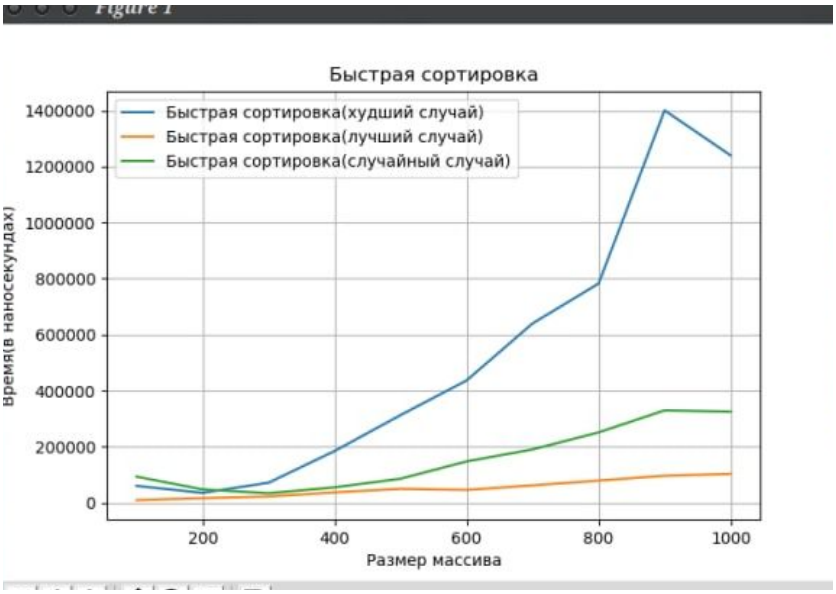


Результаты замеров времени для алгоритма быстрой сортировки для лучшего случая (Лучший), для худшего случая (Худший) и для произвольного случая (Произвольный) приведены в таблице 4.2.5. Данные из таблицы изображены на рисунке 4.2.6.

Таблица 4.2.5 Результаты замеров времени для алгоритма быстрой сортировки

Размер массива	Лучший, нс	Худший, нс	Произвольный, нс
100	8867	59788	92724
200	16082	34920	47164
300	21980	71120	33514
400	36612	184298	54455
500	49308	312781	85373
600	45318	436398	147239
700	61618	640237	190056
800	78659	782566	250806
900	96039	1401222	329642
1000	102421	1240292	325010

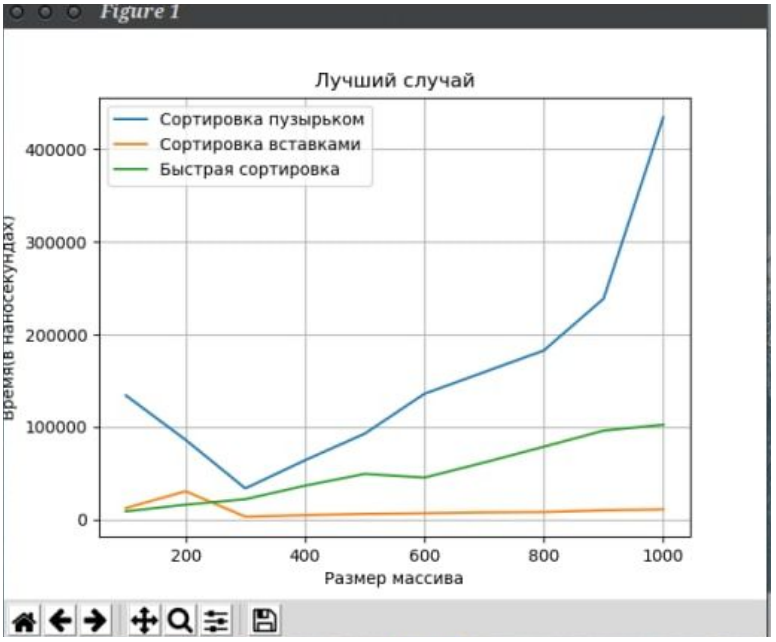
Рис. 4.2.6: Результаты замеров времени для алгоритма быстрой сортировки



Далее проведем сравнение трех разных сортировок между собой.

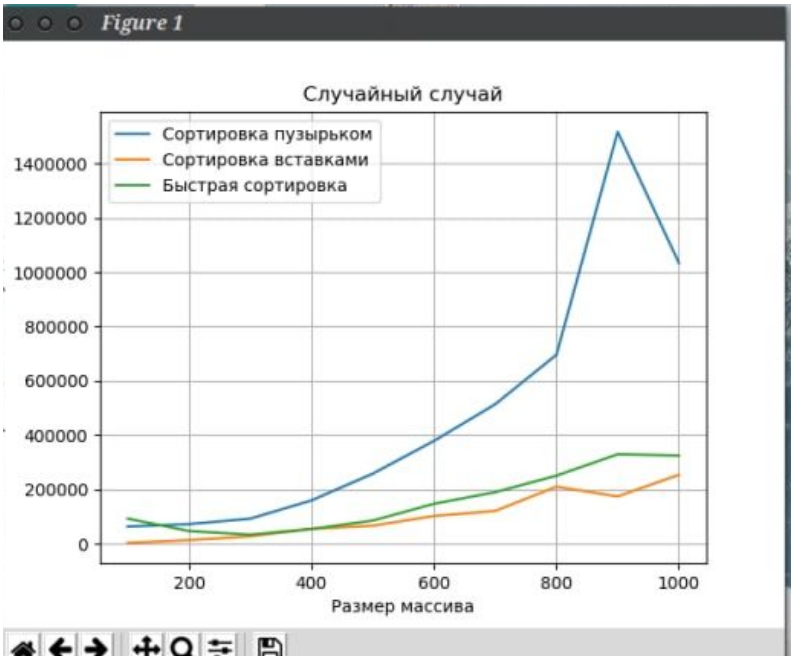
На рисунке 4.2.7 представлены зависимости времени работы алгоритма сортировки от длины массива для лучших случаев.

Рис. 4.2.7: Сравнение алгоритмов сортировки в лучших случаях



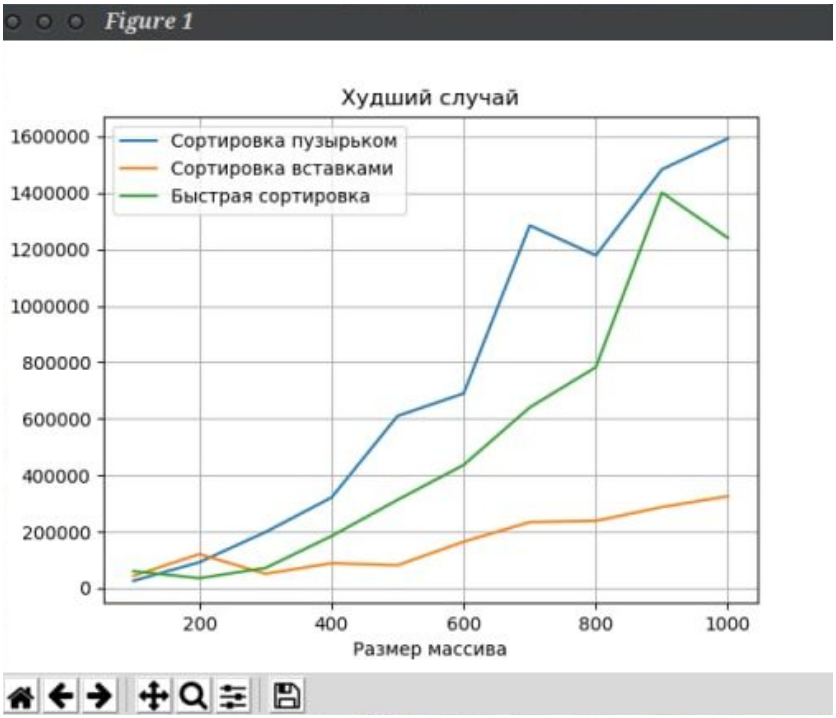
На рисунке 4.2.8 представлены зависимости времени работы алгоритма сортировки от длины массива для случайных случаев.

Рис. 4.2.8: Сравнение алгоритмов сортировки в случайных случаях



На рисунке 4.2.9 представлены зависимости времени работы алгоритма сортировки от длины массива для худших случаев.

Рис. 4.2.9: Сравнение алгоритмов сортировки в худших случаях



4.3 Вывод

По результатам эксперимента было установлено, что сортировка пузырьком самая медленная, а сортировка вставками самая быстрая. Быстрая сортировка занимает промежуточное положение, это связано с тем, что в выбранной мной реализации “опорный” элемент берется из середины. Если бы опорный элемент выбирался случайным образов, скорость работы сортировки была бы выше.

Заключение

В ходе данной лабораторной работы были изучены и реализованы три алгоритма сортировок: сортировка вставками, быстрая сортировка и пузырьковая сортировка.

Было проведено исследование времени работы трех алгоритмов сортировок (сортировка вставками, быстрая сортировка и пузырьковая сортировк).

Было проведено сравнение времени работы трех алгоритмов сортировок(сортировка вставками, быстрая сортировка и пузырьковая сортировка).

Была проведена оценка трудоемкости трех алгоритмов сортировок (сортировка вставками, быстрая сортировка и пузырьковая сортировка)

Список использованных источников

1. Кормен. Т. Алгоритмы: Построение и анализ[Текст] - Кормен Т. - Вильямс, 2014 . - 198 с. - 204 с.
2. Основные виды сортировок и их реализации[Элекстронный ресурс] https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii (дата обращения: 16.11.2020)