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»

Отчёт
по лабораторной работе № 3

Название: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ М. Р. Саркисян
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение

В ходе данной лабораторной работы необходимо изучить и реализовать алгоритмы сортировки массива, рассчитать их трудоёмкость, экспериментально подтвердить различия в трудоемкости разных реализаций, а также сравнить полученные результаты.
Будет произведена реализация алгоритмов:

- сортировки пузырьком с флагом;
- сортировки выбором;
- сортировки вставками.

1 Аналитическая часть

В данном разделе будут формализованы цель и задачи, а также рассмотрено описание алгоритмов сортировки массива.

1.1 Цель и задачи

Цель: изучить и реализовать алгоритмы сортировки массива, произвести сравнительный анализ времени работы реализованных алгоритмов, произвести теоретический анализ трудоемкости алгоритмов.

Задачи:

1. Реализация выбранных алгоритмов сортировок массива.
2. Теоретическая оценка трудоемкости реализованных алгоритмов.
3. Сравнительный анализ реализованных алгоритмов по затраченному времени.

1.2 Описание алгоритмов

Сортировка массива — одна из самых распространенных операций над массивом. Алгоритмы реализуют упорядочивание элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

1.2.1 Сортировка пузырьком с флагом

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. Идея заключается в том, что если при выполнении прохода методом пузырька не было ни одного обмена элементов массива, то это означает, что массив уже отсортирован и остальные проходы не нужны. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. За каждый проход по массиву как минимум один элемент встает на свое место, поэтому для того, чтобы отсортировать массив, необходимо совершить не более n 1 проходов, где n - размер массива[1].

1.2.2 Сортировка выбором

Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке. На каждом i-ом шаге алгоритма находим i-ый минимальный элемент и меняем его местами с i-ым элементом в массиве. Таким образом будет получен массив, отсортированный по неубыванию[2].

1.2.3 Сортировка вставками

Задача данного вида сортировки заключается в следующем: пусть есть часть массива, которая уже отсортирована, и требуется вставить остальные элементы массива в отсортированную часть, сохранив при этом упорядоченность. Для этого на каждом шаге алгоритма выбирается один из элементов входных данных и вставляется на нужную позицию в уже отсортированной части массива, до тех пор пока весь набор входных данных не будет отсортирован. Элементы вставляются в порядке их появления во входном массиве. До начала работы алгоритма за отсортированную часть массива принимается срез, состоящий из одного, начального, элемента массива[3].

1.3 Вывод

Итак, в этом разделе были поставлены цель и задачи, рассмотрено описание алгоритмов сортировки массива и кратко охарактеризованы особенности их применения.

2 | Конструкторская часть

В данном разделе будут рассмотрены требования к ПО и схемы алгоритмов.

2.1 Требования к программному обеспечению

Входные данные: массив и его размер. Выходные данные: сортировка введенного массива тремя способами.

На рисунке 2.1 представлена IDEF0-диаграмма, описывающая алгоритм сортировки массива.

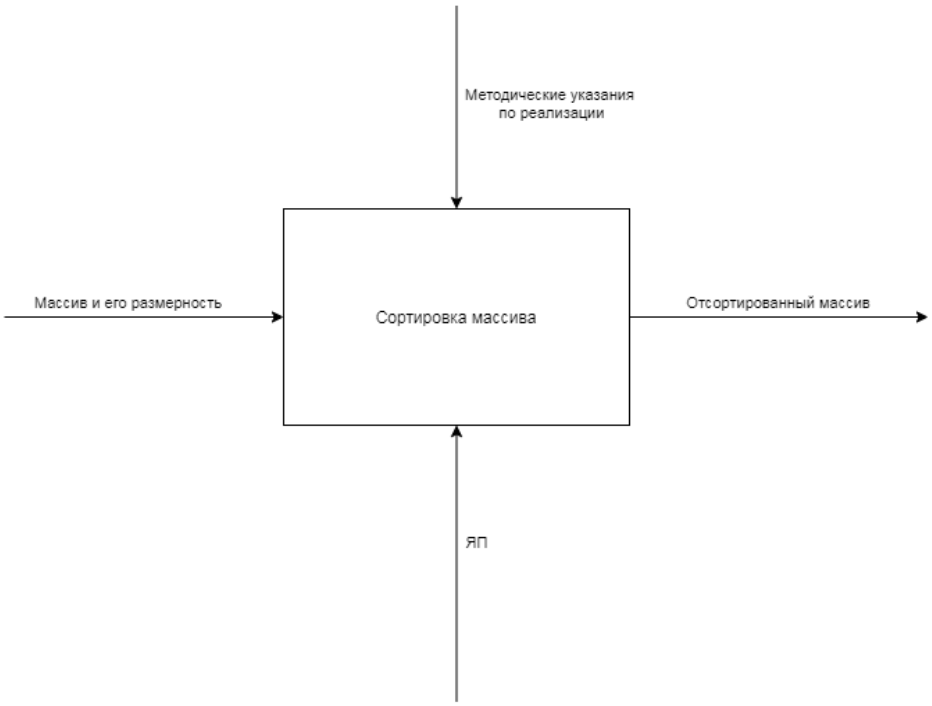


Рис. 2.1: IDEF0-диаграмма алгоритма сортировки массива

Тестирование будет производиться по стратегии чёрного ящика, что означает отсутствие знания о внутреннем коде тестируемого объекта.

2.2 Схемы алгоритмов

На рисунках 2.2, 2.3 и 2.4 представлены схемы алгоритмов сортировки.

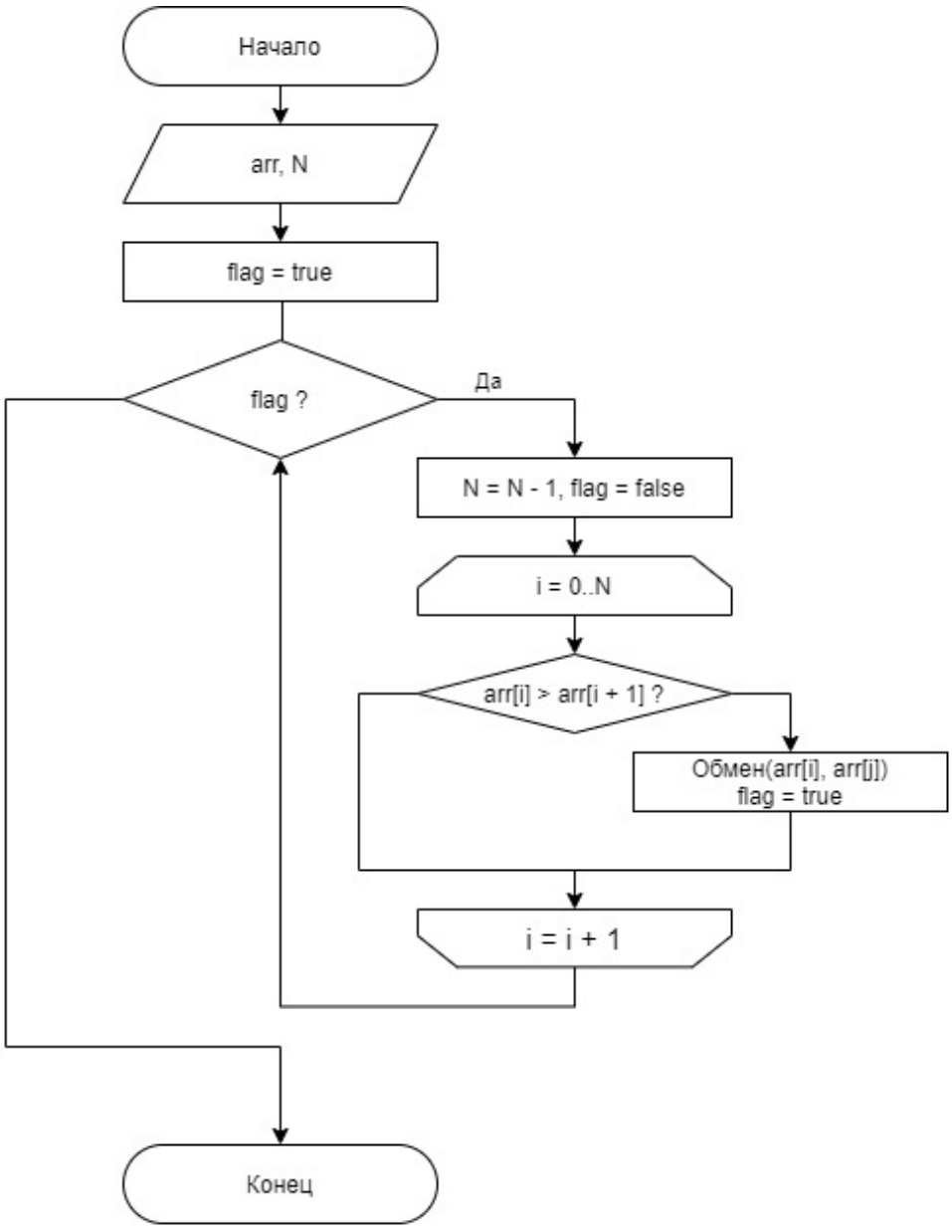


Рис. 2.2: Схема алгоритма сортировки пузырьком с флагом

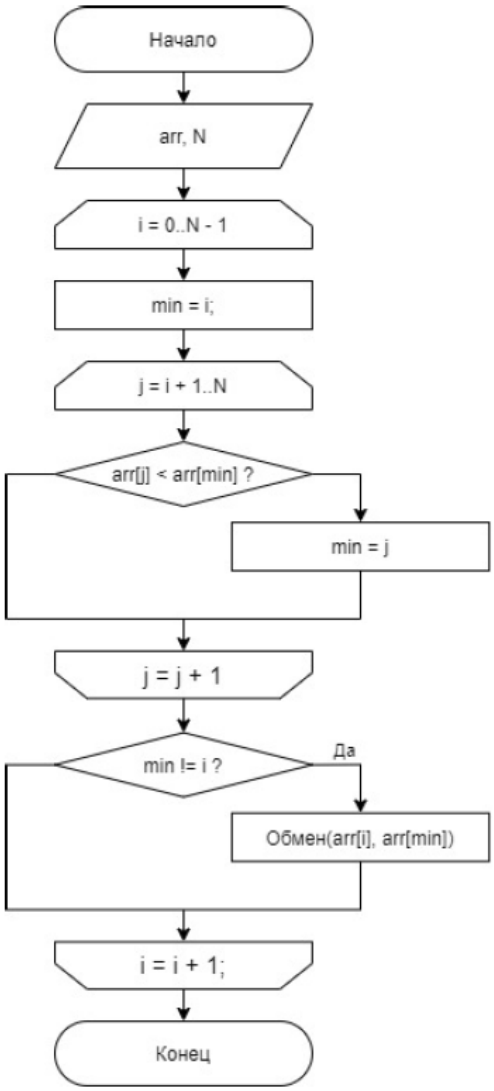


Рис. 2.3: Схема алгоритма сортировки выбором

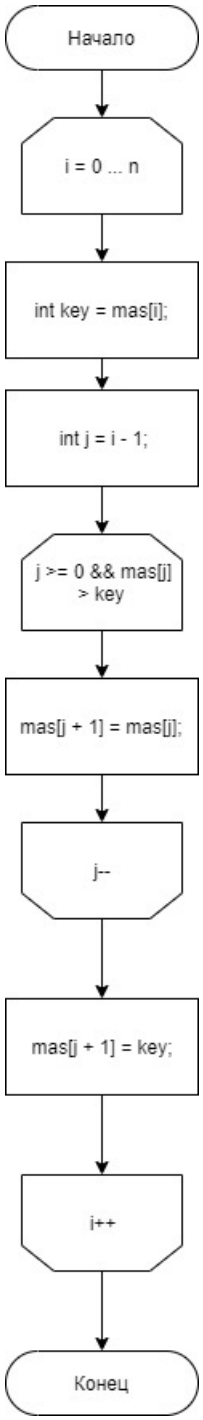


Рис. 2.4: Схема алгоритма сортировки вставками

2.3 Трудоемкость алгоритмов

Модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [];
- оценка трудоемкости цикла for от 0 до N с шагом 1 $F_{for} = 2 + N \cdot (2 + F_{body})$, где F_{body} - тело цикла;
- стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Считаем, что обмен значениями во всех алгоритмах реализован через дополнительную переменную. Тогда трудоёмкость обмена может обыть вычислена по формуле (2.1):

$$fswap = \text{`} = \text{`} + \text{`} = \text{`} + \text{`} = \text{`} = 1 + 1 + 1 = 3 \tag{2.1}$$

Для расчётов может потребоваться формула арифметической прогрессии (2.2), чтобы вычислить количество повторений за время работы:

$$S = \frac{N(N-1)}{2} \tag{2.2}$$

Оценим трудоемкость алгоритмов по коду программы.

2.3.1 Сортировка пузырьком с флагом

Рассмотрим трудоемкость сортировки пузырьком:

Лучший случай - формула (2.3):

$$1+1+1+[1+1+1+2+(N-1)(2+1+2+1)] = 3+[5+6N-6] = 6N+2 \approx 6N \tag{2.3}$$

Здесь массив уже отсортирован. Вследствие использования флага будет совершён только один проход, число сравнений равно $N - 1$.

Худший случай - формула (2.4):

$$3 + [5 + (N - 0 - 1)(6 + 3 + 1 + 1 + 1 + 1] + ... + [5 + 13(N - (N - 2) - 1)] = \\ 3 + 5 * (N - 1) + \frac{13}{2}N(N - 1) = 3 + 5N - 5 + \frac{13}{2}N^2 - \frac{13}{2}N = \\ \frac{13}{2}N^2 - 1.5N - 2 \approx \frac{13}{2}N^2 \tag{2.4}$$

Массив отсортирован в обратном порядке. Необходимо обойти весь массив целиком, число обменов и сравнений равно $\frac{N(N-1)}{2}$.

2.3.2 Сортировка выбором

Рассмотрим трудоемкость сортировки выбором:

Лучший случай - формула (2.5):

$$3 + (2 + 1 + 2 + 1 + (N - 1) * 5 + 1) + .. + (7 + 1 * 5) = \\ 3 + 7(N - 1) + 5 * \frac{N(N - 1)}{2} = \\ \frac{5}{2}N^2 + \frac{9}{2}N - 4 \approx \frac{5}{2}N^2 \tag{2.5}$$

Худший случай - формула (2.6):

$$3 + (2 + 1 + 2 + 1 + (N - 1) * 6 + 1 + 3 + 2) + .. + (12 + 1 * 6) = \\ 3 + 12(N - 1) + 6 * \frac{N(N - 1)}{2} = \\ 3N^2 + 9N - 9 \approx 3N^2 \tag{2.6}$$

2.3.3 Сортировка вставками

Рассмотрим трудоемкость сортировки вставками:

Лучший случай - формула (2.7):

$$2 + N(2 + 1 + 1 + (1 + 2 + 1 + 1 + 1) + 1 + 1) = 2 + 12N \approx 12N \tag{2.7}$$

Худший случай - формула (2.8):

$$\begin{gathered} 2 + (12) + (12 + 10 * 1) + .. + (12 + 10 * (N - 1)) = \\ 2 + 12N + 10 * \frac{N(N-1)}{2} = 2 + 7N + 5N^2 \approx 5N^2 \end{gathered} \tag{2.8}$$

2.4 Вывод

Таким образом, в данной части были сформулированы требования к программе, разработаны схемы алгоритмов, а также проведена теоретическая оценка трудоёмкости.

3 | Технологическая часть

В данном разделе будут рассмотрены средства реализации и представлен листинг кода, а также проведено тестирование по стратегии чёрного ящика.

3.1 Средства реализации

Для реализации программы мной был выбран C++[4], так как это язык программирования, содержащий средства создания эффективных программ практически любого назначения. Он также был достаточно подробно изучен мной на предыдущем курсе.

3.2 Листинг кода

В листингах 3.1, 3.2, 3.3 представлены реализации вышеупомянутых алгоритмов сортировки.

Листинг 3.1: Алгоритм сортировки пузырьком с флагом

```
1 void bubbleSortFlag(int* arr, int n) {
2   bool flag = true;
3   while (flag) {
4     --n;
5     flag = false;
6     for (int i = 0; i < n; ++i)
7       if (arr[i] > arr[i + 1])
8         swap(&arr[i], &arr[i + 1]), flag = true;
9   }
```

```
}
```

Листинг 3.2: Алгоритм сортировки выбором

```
void selectionSort(int arr[], int n)
{
  for (int i = 0; i < n - 1; i++)
  {
    int select = i;
    for (int j = i + 1; j < n; j++)
      if (arr[j] < arr[select])
        select = j;
    swap(&arr[select], &arr[i]);
  }
}
```

Листинг 3.3: Алгоритм сортировки вставками

```
void insertionSort(int* arr, int n)
{
  for (int i = 1; i < n; i++)
  {
    int key = arr[i];
    int j = i - 1;
    for (; j >= 0 && arr[j] > key; j--)
      arr[j + 1] = arr[j];
    arr[j + 1] = key;
  }
}
```

3.3 Тестирование

Тестирование производилось по стратегии чёрного ящика, что означает отсутствие знания о внутреннем коде тестируемого объекта.

В качестве примеров были выбраны:

- массив, заполненный случайными числами;

- отсортированный массив;
- отсортированный в обратном порядке массив.

На рисунках 3.1, 3.2, 3.3 отражены полученные результаты.

```
813 99 65 17 0 333 55578 1398 7713 88891 134 31 880 98 763
Bubble sort with flag:
0 17 31 65 98 99 134 333 763 813 880 1398 7713 55578 88891
Selection sort:
0 17 31 65 98 99 134 333 763 813 880 1398 7713 55578 88891
Insertion sort:
0 17 31 65 98 99 134 333 763 813 880 1398 7713 55578 88891
```

Рис. 3.1: Массив, заполненный случайными числами

```
Input array: 0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
Bubble sort with flag:
0 1 2 3 4 5 6 7 8 9
Selection sort:
0 1 2 3 4 5 6 7 8 9
Insertion sort:
0 1 2 3 4 5 6 7 8 9
```

Рис. 3.2: Отсортированный массив

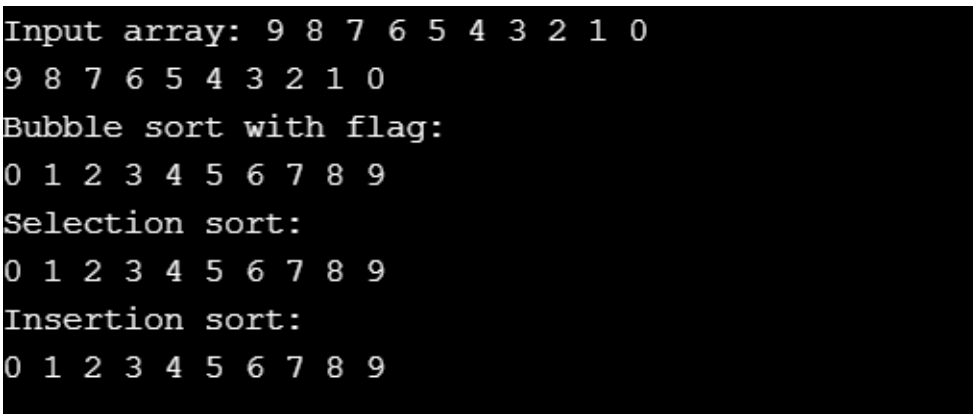


Рис. 3.3: Отсортированный в обратном порядке массив

3.4 Вывод

В данном разделе рассматривались средства реализации и представлены листинги кода вышеописанных алгоритмов с последующим их тестированием.

4 Исследовательская часть

В данном разделе будет представлено сравнение реализаций алгоритмов по времени.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В ходе выполнения лабораторной работы был проведен замер времени работы каждого из алгоритмов.

Результаты замеров времени для лучшего, произвольного и худшего случаев приведены в таблицах 4.1, 4.2 и 4.3 соответственно.

Для более наглядного представления результаты также были отражены на графиках 4.1, 4.2 и 4.3.

Таблица 4.1: Результаты замеров лучшего случая в мсек. для реализаций алгоритмов сортировки

N	Bubble sort	Selection sort	Insertion sort
100	0.001	0.016	0.001
200	0.001	0.075	0.002
300	0.001	0.139	0.002
400	0.002	0.241	0.003
500	0.002	0.307	0.002
600	0.002	0.432	0.003
700	0.003	0.593	0.003
800	0.003	0.827	0.004
900	0.004	1.016	0.005
1000	0.004	1.219	0.005

Рис. 4.1: Результаты замеров времени для лучшего случая

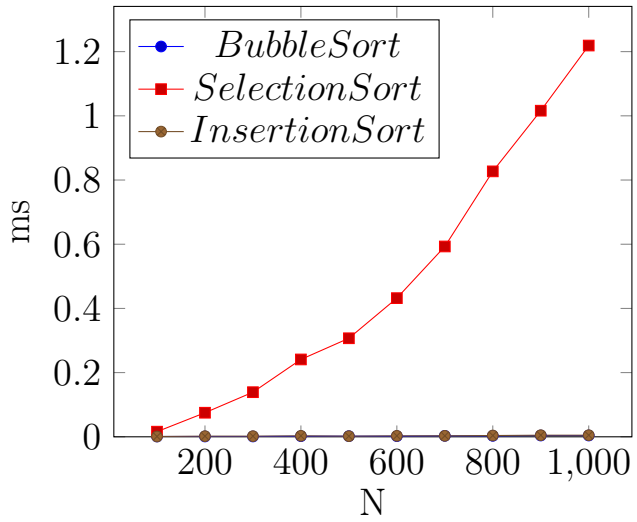


Таблица 4.2: Результаты замеров произвольного случая в мсек. для реализаций алгоритмов сортировки

N	Bubble sort	Selection sort	Insertion sort
100	0.062	0.015	0.008
200	0.337	0.069	0.039
300	0.735	0.136	0.068
400	1.303	0.246	0.118
500	1.727	0.314	0.161
600	2.348	0.445	0.244
700	3.313	0.602	0.308
800	4.137	0.786	0.401
900	5.186	0.986	0.509
1000	6.594	1.265	0.658

Рис. 4.2: Результаты замеров времени для произвольного случая

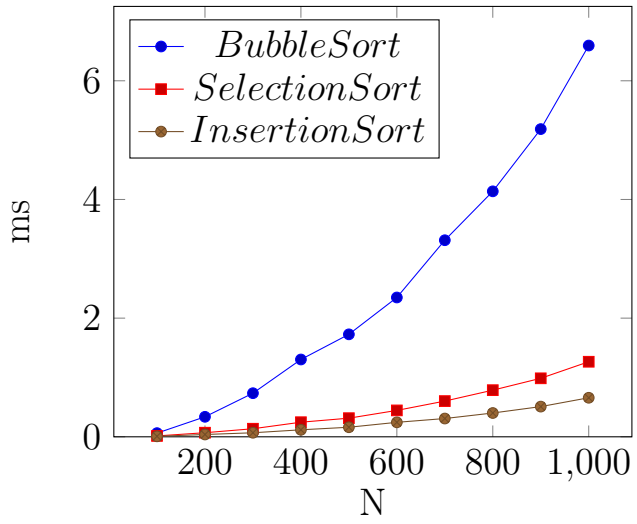
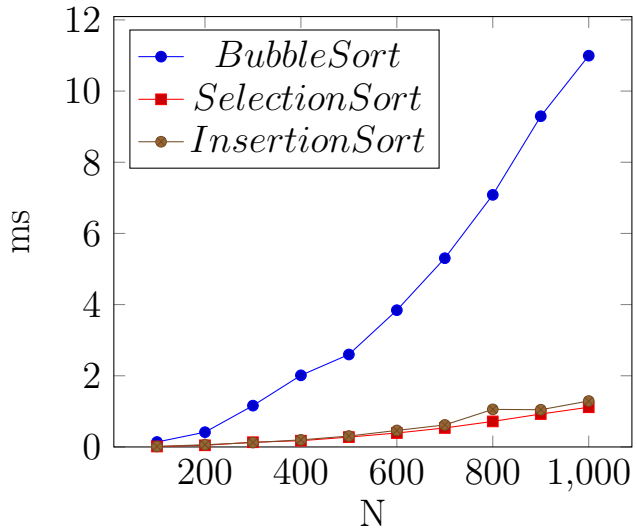


Таблица 4.3: Результаты замеров худшего случая в мсек. для реализаций алгоритмов сортировки

N	Bubble sort	Selection sort	Insertion sort
100	0.139	0.013	0.014
200	0.415	0.047	0.062
300	1.161	0.133	0.123
400	2.013	0.176	0.199
500	2.598	0.277	0.307
600	3.843	0.392	0.464
700	5.304	0.534	0.619
800	7.084	0.718	1.056
900	9.293	0.925	1.046
1000	10.994	1.116	1.291

Рис. 4.3: Результаты замеров времени для худшего случая



4.2 Вывод

В ходе исследования было выявлено, что в среднем самым медленным алгоритмом ожидаемо является пузырёк с флагом. Самым быстрым - сортировка вставками. Сортировка выбором показала средние результаты.

1. В случае произвольно заполненного массива сортировка пузырьком оказалась в 6 раз медленнее сортировки вставками и в 4 раза медленнее сортировки выбором (рис. 4.2).

2. В случае упорядоченного массива (лучший случай) худший результат показывает сортировка выбором ввиду своей специфики (рис. 4.1).

3. В случае массива, упорядоченного в обратном порядке (худший случай), сортировка пузырьком оказалась в 10 раз медленнее сортировки выбором и сортировки вставками (рис. 4.3).

Несмотря на одинаковый характер трудоёмкости алгоритмов, алгоритмы имеют различия в скорости работы, обусловленные множителями перед n^2.

Заключение

В ходе лабораторной работы мной были:

- изучены алгоритмы сортировки: пузырёк с флагом, выбором, вставками;
- реализованы алгоритмы сортировки;
- оценены трудоёмкости реализаций сортировок с последующим проведением их сравнительного анализа;
- экспериментально исследованы реализации сортировок по затраченному времени.

Впоследствии было установлено, что алгоритм сортировки вставками является наиболее быстрым среди выбранных в случае произвольно заполненного массива, а наиболее медленным - сортировка пузырьком с флагом. Так она оказалась в 4 раза медленнее сортировки выбором и в 6 раз медленнее сортировки вставками.

В худшем же случае сортировка пузырьком с флагом оказалась в 10 раз медленнее сортировки выбором и сортировки вставками.

Список использованных источников

1. Сортировка простыми обменами. [Электронный ресурс.]

   Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_пузырьком
   Дата обращения: 18.10.2020

2. Сортировка выбором. [Электронный ресурс.]

   Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_выбором

   Дата обращения: 18.10.2020

3. Сортировка вставками. [Электронный ресурс.]

   Режим доступа: https://neerc.ifmo.ru/wiki/index.php?title=Сортировка_вставками

   Дата обращения: 18.10.2020

4. Руководство по языку C++ [Электронный ресурс.]

   Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view
   2019

   Дата обращения: 18.10.2020