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: «Алгоритмы сортировки»

Студент: Ле Ни Куанг

Группа: ИУ7и-56Б

Преподаватель: Волкова Л. Л.

Строганов Ю. В.

Москва

2020

Оглавление

Введение

Алгоритм сортировки - это алгоритм для упорядочивания 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.

Целью работы: изучение алгоритмов сортировки массивов, сравни- тельный анализ времени работы данных алгоритмов, анализ трудоемкости алгоритмов.

Задачи работы:

1. реализовать три различных алгоритма сортировки;
2. теоретически вычислить эффективность алгоритмов;
3. сравнить эффективности алгоритмов по времени.

1 Аналитический раздел

В данном разделе будет приведено описание алгоритмов и модель вычислений для оценок трудоемкости.

1.1 Описание алгоритмов

Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти. В этом разделе будет приведено три алгоритма: сортировка пузырьком, сортировка вставками и сортировка слиянием.

1.1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются $N - 1$ раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает - массив отсортирован.

Сложность по времени

- лучшее время: $O(n)$
- среднее время: $O(n^2)$
- худшее время: $O(n^2)$

Затраты памяти: $O(1)$

1.1.2 Сортировка вставками

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Сложность по времени

- лучшее время: $O(n)$
- среднее время: $O(n^2)$

- худшее время: $O(n^2)$

Затраты памяти: $O(1)$

1.1.3 Сортировка слиянием

Сортируемый массив разбивается на две части примерно одинакового размера. Каждая из получившихся частей сортируется отдельно. Два упорядоченных массива половинного размера соединяются в один.

Сложность по времени

- лучшее время: $O(n\ log(n))$
- среднее время: $O(n\ log(n))$
- худшее время: $O(n\ log(n))$

Затраты памяти: $O(n)$

1.1.4 Модель вычислений

В данной работы используется следующая модель вычислений:

1. Стоимость базовых операций: $F = 1$
   $(=, *, +, -, /, \%, <, <=, >, >=, ==, !=, [], +=, -=, *=, /=)$
2. Стоимость цикля for

$$F_{for} = f_{init} + f_{compare} + N_{loop} \cdot (f_{body} + f_{inc} + f_{compare}) \tag{1.1}$$

3. Трудоемкость условного оператора if

$$F_{if} = f_{compare} + f_{body} = f_{compare} + \begin{cases} f_{min}, & \text{лучший случай} \\ f_{max}, & \text{худший случай} \end{cases} \tag{1.2}$$

1.2 Вывод

Были приведено описание алгоритмов сортировка пузырьком, сортировка вставками и сортировка слиянием, также рассмотрено модель вычислений для оценок трудоемкости.

2 Конструкторский раздел

В данном разделе будет приведено описание схем алгоритмов сортировка пузырьком, сортировка вставками, сортировка слиянием и вычислены их трудоемкости.

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2, 2.3 показаны схемы алгоритмов сортировки.

2.1.1 Сортировка пузырьком

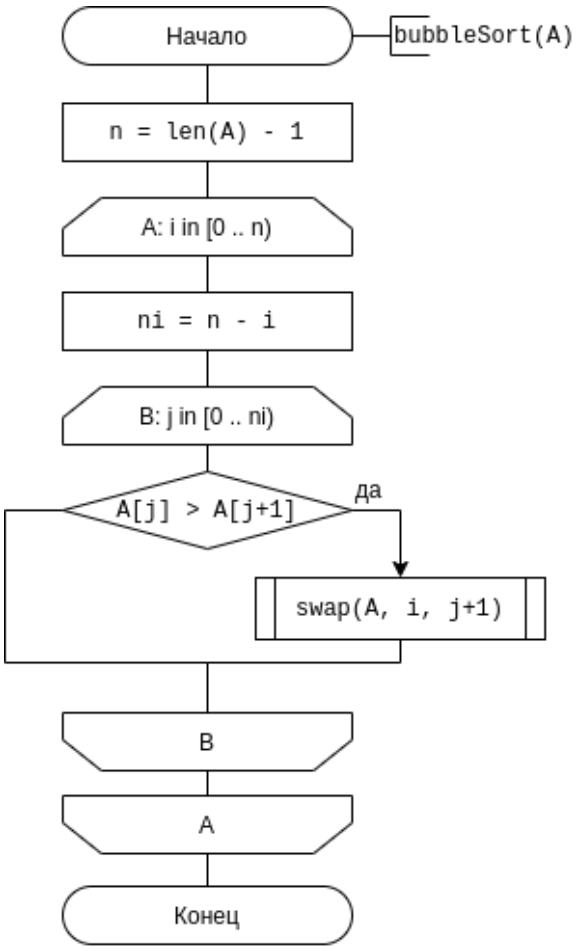

Рис. 2.1: Схема алгоритма сортировка пузырьком

2.1.2 Сортировка вставками

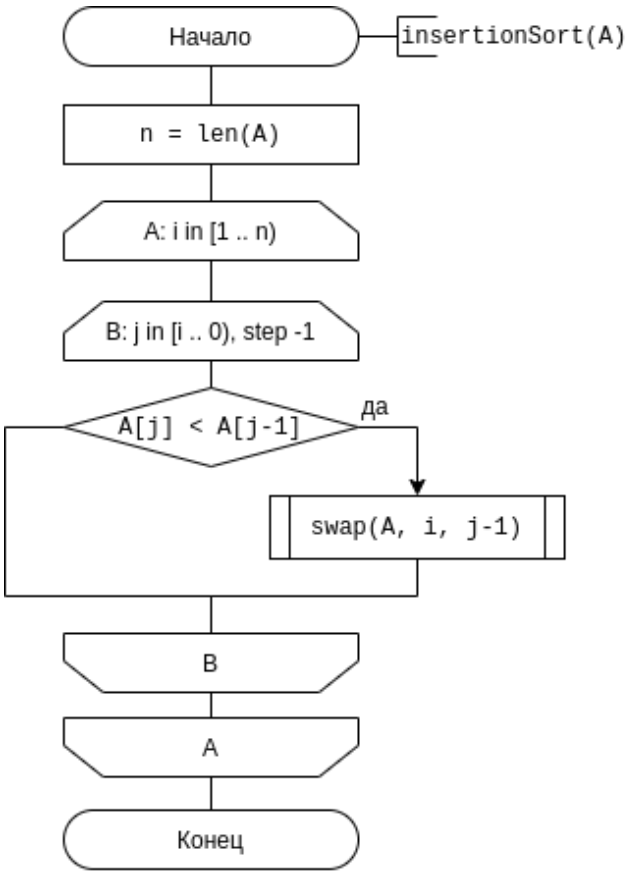

Рис. 2.2: Схема алгоритма сортировка вставками

2.1.3 Сортировка слиянием

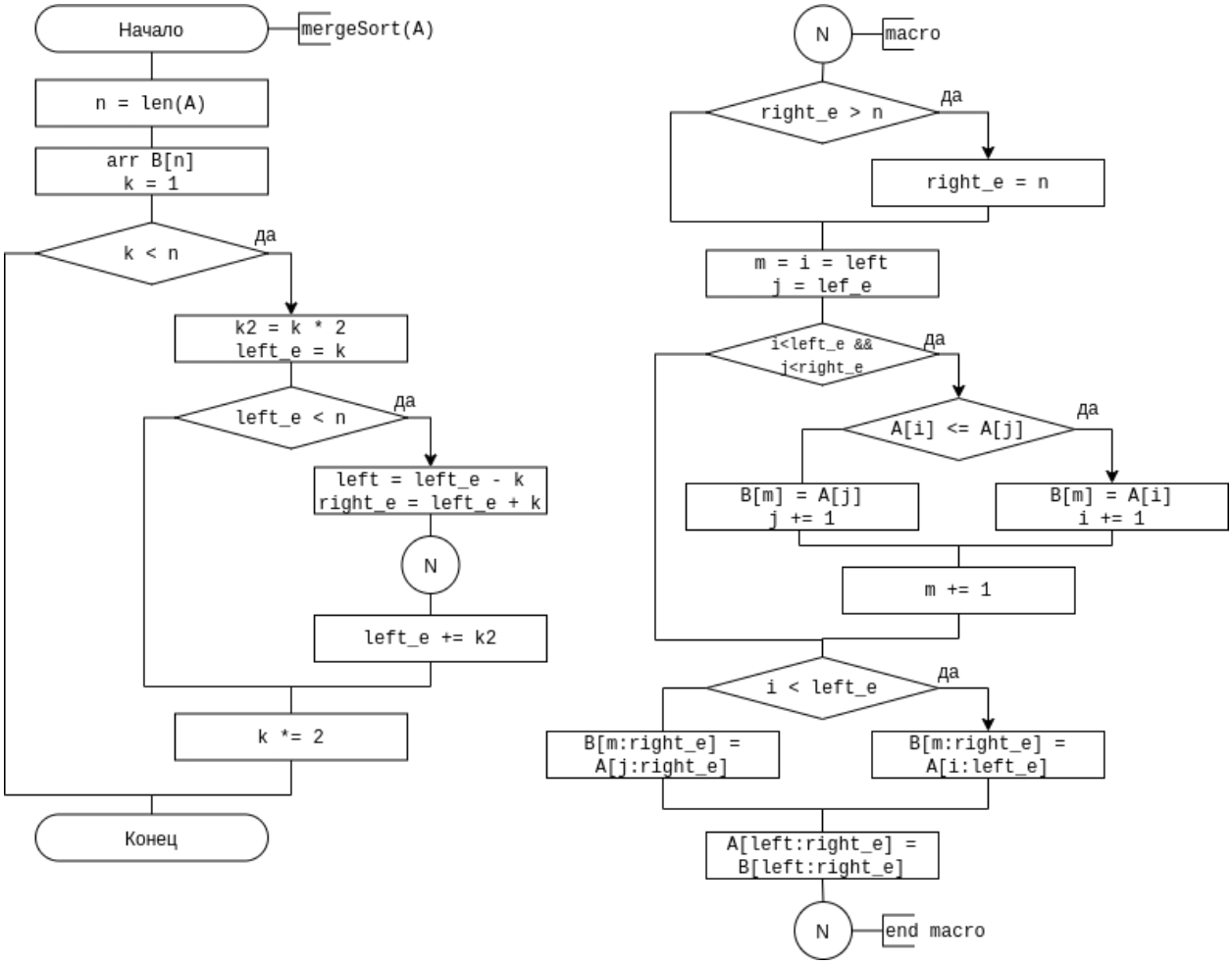

Рис. 2.3: Схема алгоритма сортировка слиянием

2.2 Оценка трудоемкости

Применить формулы 1.1 и 1.2. Предположим, что сложность *swap*() равна 5, сложность *len*() равна n.

2.2.1 Сортировка пузырьком

F_{if}	$\begin{cases} 4, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases}$
F_{forA}	$\begin{cases} 4n + 6 \cdot \dfrac{(n-2)(n-1)}{2} = 3n^2 - 5n + 6, & \text{л.с.} \\ 4n + 11 \cdot \dfrac{(n-2)(n-1)}{2} = 5.5n^2 - 12.5n + 11, & \text{х.с.} \end{cases}$
F_{bubble}	$\begin{cases} 3n^2 - 4n + 10, & \text{л.с.} \\ 5.5n^2 - 11.5n + 15, & \text{х.с.} \end{cases}$

2.2.2 Сортировка вставками

F_{if}	$\begin{cases} 4, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases}$
F_{forA}	$\begin{cases} 2n + 6 \cdot \dfrac{(n-2)(n-1)}{2} = 3n^2 - 7n + 6, & \text{л.с.} \\ 2n + 11 \cdot \dfrac{(n-2)(n-1)}{2} = 5.5n^2 - 14.5n + 11, & \text{х.с.} \end{cases}$
F_{insert}	$\begin{cases} 3n^2 - 6n + 9, & \text{л.с.} \\ 5.5n^2 - 13.5n + 14, & \text{х.с.} \end{cases}$

2.2.3 Сортировка слиянием

Алгоритм использует парадигма «разделяй и властвуй», делит массив на два меньших массива, сортирует его, а затем объединяет. Шаг завоевания, на котором мы рекурсивно сортируем два подмассива примерно по n/2 элемента в каждом. Шаг объединения объединяет в общей n элементов, что занимает время O(n). В любом случае сложность сортировки слиянием $nlog_2n$. (источник "Analysis of merge sort"[4]).

2.3 Вывод

В данном разделе было приведено описание схем алгоритмов и вычислены их трудоемкости.

3 Технологический раздел

3.1 Средства реализации

Язык программирования: Python (IPython)

Библиотеки: unittest, timeit, matplotlib, ...

Редактор: Jupyter-Lab

Я использую эти инструменты потому, что они мощные, широко используемые и знакомые мне.

3.2 Листинг кода

Листинг 3.1: Сортировка пузырьком

```
def bubbleSort(A):
    n = len(A) - 1
    for i in range(n):
        ni = n-i
        for j in range(ni):
            if A[j] > A[j+1]:
                A[j], A[j+1] = A[j+1], A[j]
```

Листинг 3.2: Сортировка вставками

```
def insertionSort(A):
    n = len(A)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if A[j] < A[j-1]:
                A[j], A[j-1] = A[j-1], A[j]
            else:
                break
```

Листинг 3.3: Сортировка слиянием

```
def mergeSort(A):
    n = len(A)
    B = [None] * n
    k = 1
    while k < n:
        k2 = k * 2
        left_end = k
        while left_end < n:
            left = left_end - k
            right_end = left_end + k

            if right_end > n: right_end = n

            m = i = left
            j = left_end

            while i < left_end and j < right_end:
                if A[i] <= A[j]:
                    B[m] = A[i]
                    i += 1
                else:
                    B[m] = A[j]
                    j += 1
                m += 1

            if i < left_end:
                B[m:right_end] = A[i:left_end]
            else:
                B[m:right_end] = A[j:right_end]

            A[left:right_end] = B[left:right_end]

            left_end += k2

        k *= 2
```

3.3 Описание тестирования

В таблице 3.1 приведен функциональные тесты для алгоритмов сортировки.

Массив	**Результат**
1	1
1 1 1 1 1	1 1 1 1 1
1 2 3 4 5	1 2 3 4 5
5 4 3 2 1	1 2 3 4 5
2 4 5 1 3	1 2 3 4 5
2 4 2 2 4	2 2 2 4 4
2 4 8 6 4 0	0 2 4 4 6 8

Таблица 3.1: Функциональные тесты

3.4 Вывод

В этом разделе было рассмотрено код программы и описание тестирования.

4 Экспериментальный раздел

4.1 Примеры работы

На рисунке 4.1 приведен пример работы программы.

```
Input:   [157   8   6 172 169 ...  68  20   7  95 142]
Output:  [  1   2   3   4   5 ... 196 197 198 199 200]

n = 200         Best            Average         Worst
==============
Bubble sort     0.001582        0.002635        0.004127
Insertion sort  0.000065        0.002377        0.004868
Merge sort      0.000249        0.000359        0.000333

n = 400         Best            Average         Worst
==============
Bubble sort     0.007427        0.011859        0.013976
Insertion sort  0.000129        0.007216        0.013042
Merge sort      0.000481        0.000677        0.000699
```

Рис. 4.1: Примеры работы алгоритмов сортировки

4.2 Результаты тестирования

На рисунке 4.2 приведен результат теста с использованием фреймворка модульного тестирования в Python: unittest

```
test_bubble (__main__.TestSorting) ... ok
test_insertion (__main__.TestSorting) ... ok
test_merge (__main__.TestSorting) ... ok

----------------------------------------------
Ran 3 tests in 0.003s

OK
```

Рис. 4.2: Результаты тестирования

4.3 Постановка эксперимента по замеру времени

Операционная система - Ubuntu 20.04.1 LTS

Процессор - Intel® CoreTM i5-7300HQ CPU @ 2.50GHz × 4

Сортировка пузырьком

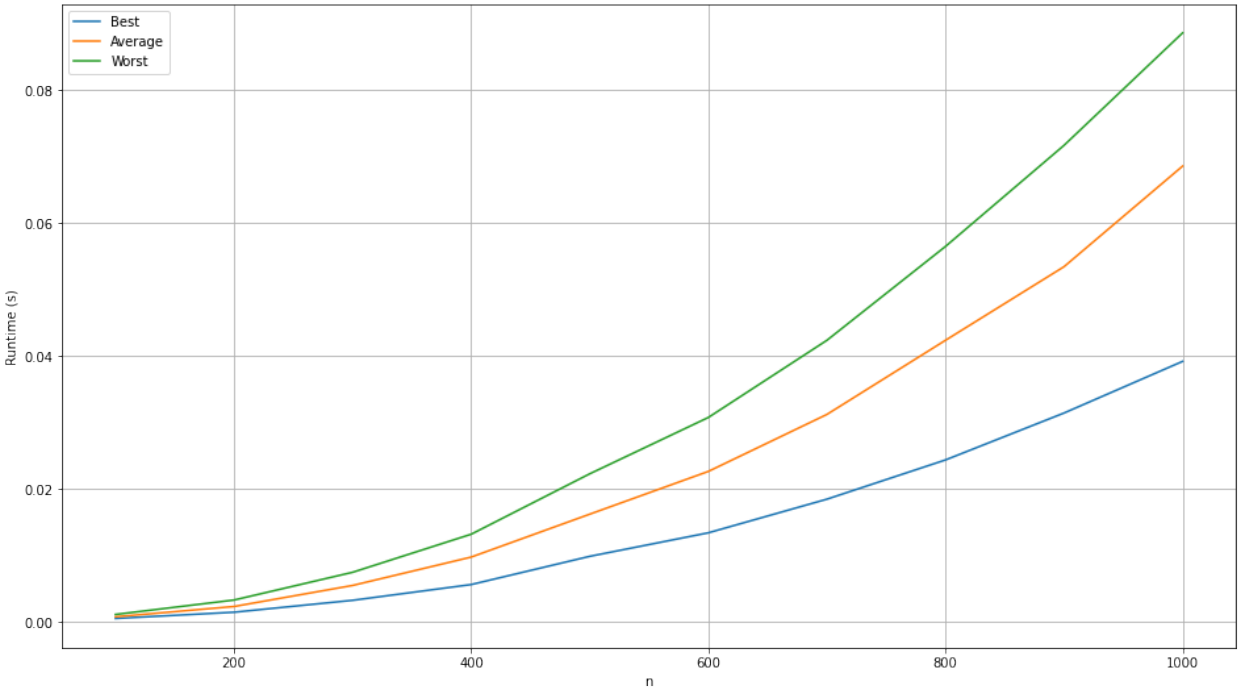

Рис. 4.3: Времени сортировки пузырьком

Сортировка вставками

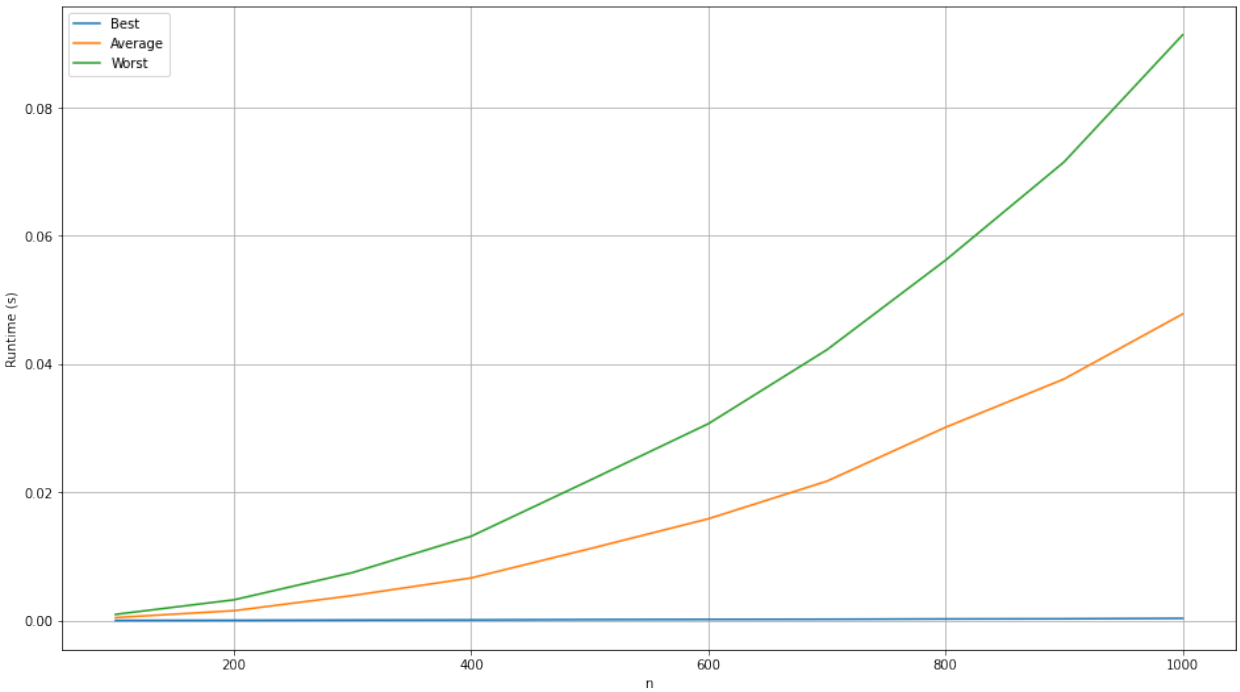

Рис. 4.4: Времени сортировки вставками

Сортировка слиянием

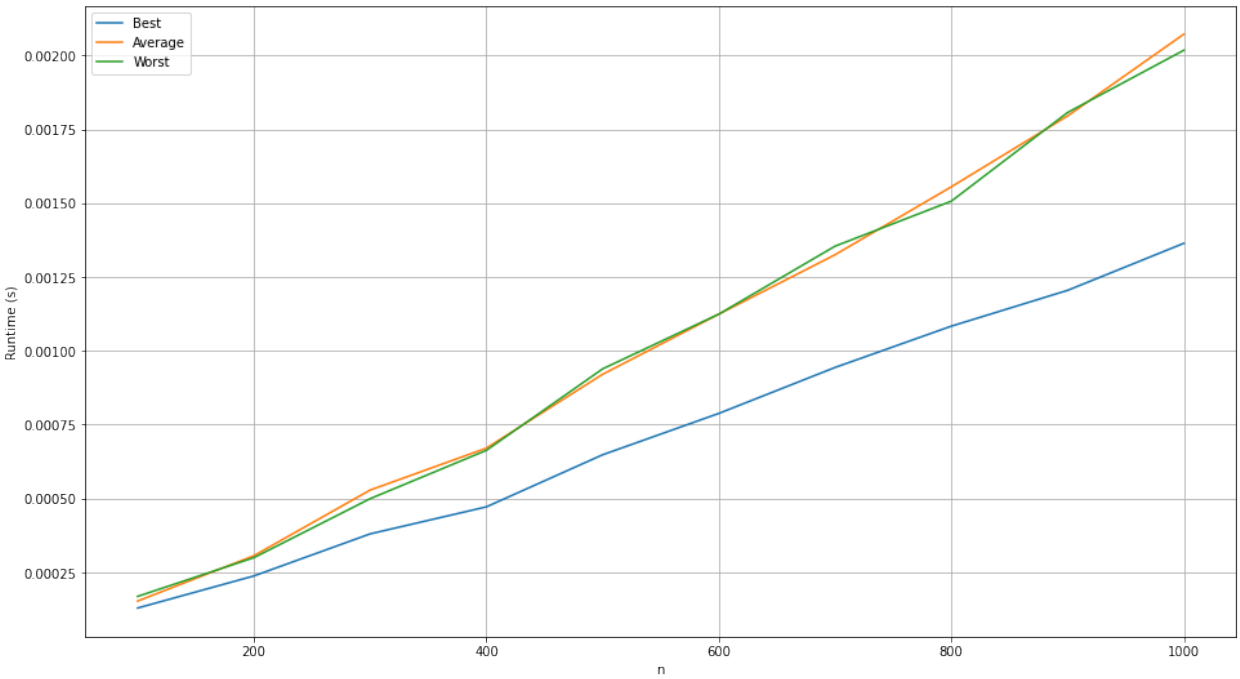

Рис. 4.5: Времени сортировки слиянием

Лучшее время

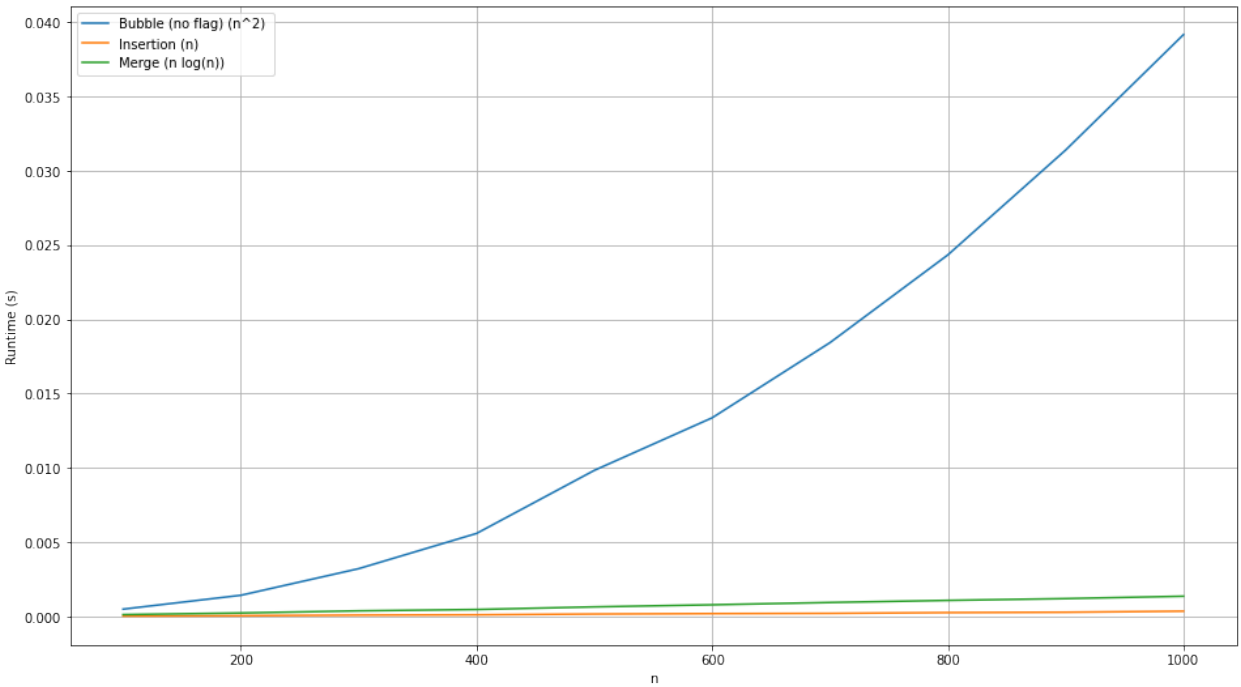

Рис. 4.6: Сравнение лучшее времени работы алгоритмов

Среднее время

n	Bubble	Insertion	Merge
100	0.00074	0.00048	0.00015
200	0.00229	0.00155	0.00031
300	0.00546	0.00391	0.00053
400	0.00972	0.00665	0.00067
500	0.01618	0.01121	0.00092
600	0.02261	0.01588	0.00112
700	0.03118	0.02174	0.00133
800	0.04233	0.03014	0.00156
900	0.0534	0.03767	0.0018
1000	0.06853	0.0478	0.00207

Таблица 4.1: Среднее времени работы (ns)

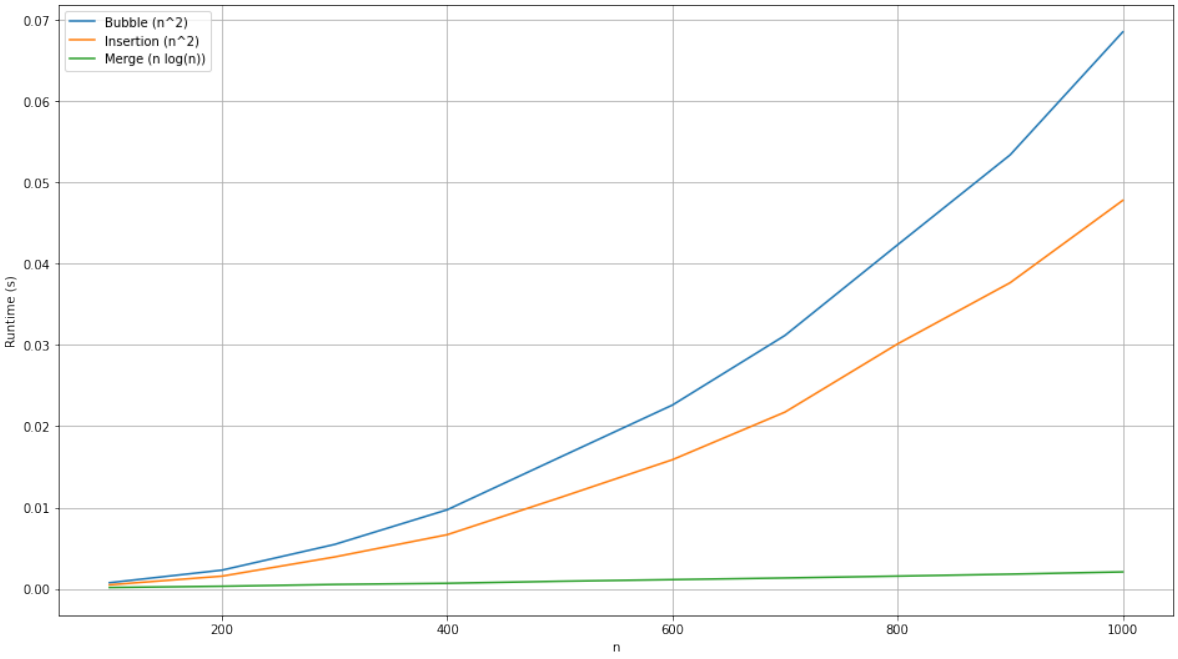

Рис. 4.7: Сравнение среднее времени работы алгоритмов

4.4 Вывод

Из графика мы видим, что алгоритм сортировки слиянием работает со сложностью O(nlog(n)) во всех случаях и является наиболее оптимальным, алгоритм сортировки вставками является лучшим, если массив уже отсортирован или почти отсортирован. Сортировка пузырьком используется в основном как обучающий инструмент. На массив длины 1000, сортировка слиянием в 30 раз быстрее сортировки пузырьком и в 23 раз быстрее сортировки вставками по сравнению среднее времени работы.

Заключение

В ходе лабораторной работы было проведено сравнение трех алгоритмов сортировки: сортировка пузырьком, сортировка вставками и сортировка слиянием. Были сделаны следующие выводы:

- сортировка слиянием на порядок быстрее сортировки пузырьком и вставками (использует дополнительную O(n) память);
- сортировка вставками быстрее сортировки пузырьком (без флаг) во всех случаях;
- сортировка вставками самый быстрый для почти отсортирован массив;
- на массив длины 1000, сортировка слиянием в 30 раз быстрее сортировки пузырьком и в 23 раз быстрее сортировки вставками по сравнению среднее времени работы.

Литература

[1] Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с.

[2] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = Introduction to algorithms. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296.

[3] The Python Language Reference
`https://docs.python.org/3/reference/`

[4] Analysis of merge sort
`https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/analysis-of-merge-sort`