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 _____ (подпись, дата) Бугаенко А.П. (Фамилия И.О.)

Преподователь _____ (подпись, дата) Волкова Л.Л. (Фамилия И.О.)

Москва, 2022

Содержание

Введение

Алгоритм сортировки - это алгоритм, позволяющий упорядочить данные, заданные в форме массива, по некоему заранее известному правилу. Если элемент в массиве имеет множество полей, то выделяется поле или группа полей, служащее критерием упорядоченности массива. Такоие поле/поля называются ключом сортировки. Данные алгоритмы широко используются в совремнном программировании для упорядовачивания совершенно различных видов данных. Видов сортировок существует множество, однако в этой лабораторной мы будем рассматривать три конкретных алгоритма: алгоритм сортировки "пузырьком", алгоритм сортировки "расчёской", алгоритм сортировки вставками.

Целью данной лабораторной работы является изучение описанных выше алгоритмов сортировок и получение практических навыков при реализации данных алгоритмов. Для того, чтобы достичь поставленной цели, нам необходимо выполнить следующие задачи:

1) Провести анализ данных алгоритмов сортировки:
    а) Алгоритм сортировки "пузырьком";
    б) Алгоритм сортировки "расчёской";
    в) Алгоритм сортировки вставками;
2) описать используемые структуры данных;
3) привести схемы рассматриваемых алгоритмов;
4) вычислить трудоёмкость для указанных выше алгоритмов;
5) программно реализовать данные выше алгоритмы;
6) провести сравнительный анализ каждого алгоритма по затрачиваемому в процессе работы времени.

1 Аналитический раздел

В данном разделе будут рассмотрены теоретически основы работы данных выше алгоритмов сортировки "пузырьком", алгоритмов сортировки "расчёской", алгоритм сортировки вставками.

1.1 Основные сведения об алгоримтах

Сортировка является одной из операций, которые необходимы при работе над данным. В частности эта необходимость возникает из-за потребности какой-либо обработки данных в процессе выполнения программы. Входные параметры варьируются от сортировке к сортировке, однако в общем случае на вход подаётся массив данных, а на выходе алгоритм возвращает отсортированный определённым образом массив.

1.2 Алгоритм сортировки "пузырьком"

Данный алгоритм работает за счёт повторяющихся проходов по массиву, который был подан на вход. При каждом проходе последовательно сравниваются два соседних элемента. Если порядок в паре не соответствует условию сортировки, то элементы являются местами. Всего проходов по массиву выполняется N - 1 раз, где N - длина массива. При каждом проходе по внутреннему циклу алгоритм находит наибольший элемент и помещает его рядом с предыдущим наибольшим элементом, который был найден во время предыдущего прохода, а наименьший элемент, перемещается на одну позицию в сторону начала массива, если сортировка идёт по возврастанию.

1.3 Алгоритм сортировки "расчёской"

Основная идея данного алгоритма залючается в том, что при каждом проходе сравниваются элементы, находящиеся на неком расстоянии. Использование данного приёма позволяет быстро перемещать элементы из разных концов массива, избегая появления "черепах" - элементов, которые долго переходят из одного конца массива в другой. В остальных отношениях данный алгоритм работает аналогично сортировке "пузырьком". Расстояние между сравниваемыми величинами изменяется от N - 1, до 0.

1.4 Алгоритм сортировки вставками

Основная идея сортировки вставками состоит в том, что при сортировке элементы сгоняются в один из концов массива, являющийся отсортированным подмассивом. Это позволяет при каждом проходе сразу вставлять элемент в нужное место в уже отсортированную часть массива. Массив считается отсортированным, когда отсортированная часть поглощает весь массив.

1.5 Модель вычислений

Для того, чтобы вычислить трудоёмкость алгоритма, введём следующую модель вычислений:

1) операторы, трудоёмкость которых равна 1: +, -, /, %, ==, !=, <, >, >=, <=, [], ++;

2) трудоёмкость оператора выбора if (условие) then A else B рассчитывается как $f_{\text{условия}} + f_A$ или f_B в зависимости то того, какое условие выполняется;

3) трудоёмкость цикла рассчитывается как $f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$;

4) трудоёмкость вызова функции равна 0.

1.6 Вывод

В данном разделе были рассмотрены основные теоретические сведения об алгоритме сортировки "пузырьком", алгоритме сортировки "расчёской", алгоритме сортировки вставками. В результате были сделаны выводы о том, что на вход алгоритму подаётся несортированный массив, на выходе программа возвращает отсортированный массив. Алгоритмы работают на массивах с размерностями от 0 до физически возможного предела для изпользуемой машины. В качестве критерия для сравнения эффективности алгоритмов будет использоваться время работы на массивах различного размера. Помимо этого был проведён анализ модели вычислений, которая будет использоваться трудоёмкости алгоритмов.

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы, структуры данных, способы тестирования, описания памяти для следующих алгоритмов:

1) алгоритм сортировки ”пузырьком”;
2) алгоритм сортировки ”расчёской”;
3) алгоритм сортировки вставками.

2.1 Тестирование алгоритмов

Описание классов эквивалентности:

1) проверка работы на пустом массиве;
2) проверка работы на общем случае.

Описание тестов:

1) тест на пустом массиве - на вход подаётся пустой, выход сравнивается с пустым массивом;
2) тест на общем случае - на вход подаётся массив длиной n, выход сравнивается с отсортированной версией массива, поданного на вход.

2.2 Алгоритм сортировки ”пузырьком”

Алгоритм сортировки "пузырьком"реализуется по описанным выше принципам. На вход подаётся сортируемый массив, на выходе получается отсортированный массивы.

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) bool, логическая переменная - используется для хранения флага;
3) array, массив целых чисел - используется для хранения серии целых чисел.

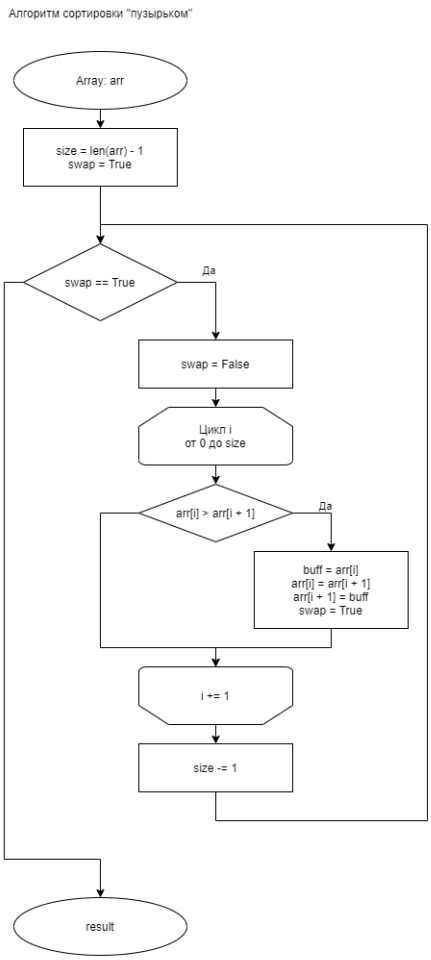

Рисунок 2.1 — Схема алгоритма сортровки ”пузырьком”

2.3 Алгоритм сортировки ”расчёской”

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов массива, размера массива;
2) float, вещественное число - используется для хранения коэффициента;
3) bool, логическая переменная - используется для хранения флага;
4) array, массив целых чисел - используется для хранения серии целых чисел.

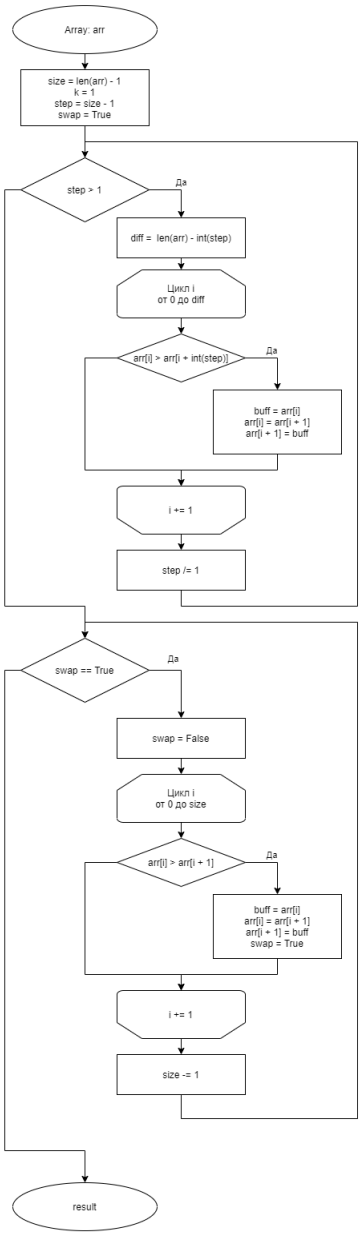

Рисунок 2.2 — Схема алгоритма сортировки ”расчёской”

2.4 Алгоритм сортировки вставками

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов и размерностей матрицы

2) matrix, массив массивов вещественного типа - используется для хранения двух входных матриц и матрицы, хранящей результат умножения

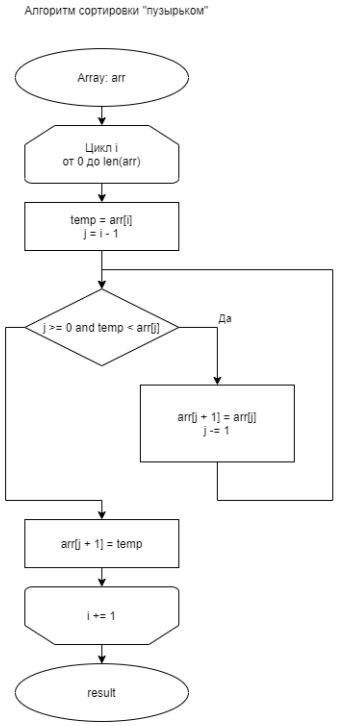

Рисунок 2.3 — Схема алгоритма сортировки вставками

2.5 Трудоёмкость алгоритмов

Проведём оценку трудоёмкости алгоритмов на основе описанных выше схем. N - размер подаваемого на вход массива.

— Трудоёмпость алгоритма сортировки ”пузырьком”:

Лучший случай:

Трудоёмкость - 2 + 1+ 1 + 1 + 3 + (N - 1)(3 + 4) + 1;

Результирующая трудоёмкость - 7N + 2;

Худший случай:

Трудоёмкость - 2 + 1 + 1 + (N - 1)(3 + 1 + (N/2)(4 + 2 + 4 + 3 + 1) + 1);

Результирующая трудоёмкость - $7N^2$ - 2N - 1;

— Трудоёмпость алгоритма сортировки ”расчёской”:

Лучший случай:

Трудоёмкость - 2 + 1 + 2 + (N/2)(1 + 2 + 3 + 4(N/2) + 1) + 1 + 1 + 1 + 1 + 3 + (N/2)(3 + 4) + 1;

Результирующая трудоёмкость - N^2 + 7N + 21;

Худший случай:

Трудоёмкость - 5 + (N/2)(1 + 2 + 3 + (N/2)(3 + 4 + 2 + 4 + 3) + 1) + 1) + 1 + 1 + 1 + 3 + (N/2)(3 + 4) + 1;

Результирующая трудоёмкость - $4N^2$ + 13N/2 + 12;

— Трудоёмпость алгоритма сортировки вставками:

Лучший случай:

Трудоёмкость - 3 + (N - 1)(2 + 2 + 4 + 4 + 4 + 1 + 1);

Результирующая трудоёмкость - 18N - 5;

Худший случай:

Трудоёмкость - 3 + (N - 1)(2 + 2 + 4 + (N/2)(4 + 4+ 1) + 1);

Результирующая трудоёмкость - $9N^2/2$ + N/2 -2;

2.6 Функциональная схема ПО

На изображении ниже представлена функиональная схема разрабатываемого ПО. На вход подаётся массив с числами и при помощи алгоритмов, реализованных на языке Python мы получаем в результате работы новый массив, содержащий в себе результат сортировки.

Рисунок 2.4 — IDEF0 диаграмма разрабатываемой программы

2.7 Вывод

В данном разделе были рассмотрены схемы алгоритмов для каждого из алгоритмов сортировки, и были определены тесты для каждого алгоритма, были описаны типы и структуры данных, использующихся в алгоритмах. Также была произведена оценка трудоёмкости для изучаемых алгоритмов и приведена функциональная схема разрабатываемого ПО.

3 Технологический раздел

В данном разделе будут рассмотрены подробности реализации описаных выше алгоритмов. Также будут обоснованы выбор языка программирования для реализации, выбор библиотек для проведения экспериментов и представлены важные фрагменты кода написанной в рамках работы программы.

3.1 Выбор языка программирования

В качестве языка программирования для реализации данной лабораторной работы использовался язык программирования python (3.9.7) в целях упрощения работы со структурами данных и визцализацией данных сравнительных анализов и наличием опыта работы с данным языком программирования. В качестве среды разработки использовалась Visual Studio Code.

3.2 Сведения о модулях программы

Реализованное ПО состоит из трёх модулей:

1) algos.py - в данном модуле реализованы алгоритмы сортировки;
2) time.py- в данном модуле реализованы замеры временных характеристик алгоритмов;
3) tests.py - в данном модуле реализованы тесты алгоритмов.

3.3 Реализация алгоритмов сортировки массивов

Листинг 3.1 — Реализация алгоритма сортировки "пузырьком"

```
def bubble_sort(arr):
  size = len(arr) - 1
  swap = True
  while swap:
    swap = False
    for i in range(0, size):
      if arr[i] > arr[i + 1]:
        buff = arr[i]
        arr[i] = arr[i + 1]
        arr[i + 1] = buff
        swap = True
    size -= 1
  return arr
```

Листинг 3.2 — Реализация алгоритма сортировки "расчёской"

```
def comb_sort(arr):
  size = len(arr) - 1
  k = 1
  step = size - 1
  while step > 1:
```

```
    diff = len(arr) - int(step)
    for i in range(0, diff):
      if arr[i] > arr[i + int(step)]:
        buff = arr[i]
        arr[i] = arr[i + 1]
        arr[i + 1] = buff
    step -= k
  swap = True
  while swap:
    swap = False
    for i in range(0, size):
      if arr[i] > arr[i + 1]:
        buff = arr[i]
        arr[i] = arr[i + 1]
        arr[i + 1] = buff
        swap = True
    size -= 1
  return arr
```

Листинг 3.3 — Реализация алгоритма сортировки вставками

```
def insert_sort(arr):
  for i in range(1, len(arr)):
    temp = arr[i]
    j = i - 1
    while (j >= 0 and temp < arr[j]):
        arr[j + 1] = arr[j]
        j -= 1
    arr[j + 1] = temp
  return arr
```

3.4 Реализация тестирования алгоритмов

Для тестирования алгоритмов было реализованы следующие тесты:

1) тест на нулевых масссивах;
2) тест на массивов, заполненных рандомными величинами от -100 до 100 длиной n;

Листинг 3.4 — Реализация функции рандомной генерации массивов

```
def check_arr(arr_a, arr_b):
    res = True
    l = len(arr_a)
    for i in range(0, l):
        if arr_a[i] != arr_b[i]:
            res = False
    return res
```

Листинг 3.5 — Реализация общей функции тестирования

```
def test_case(arr_a, arr_true, case_name):
    print(case_name)
    print('Параметры массива: N = ', len(arr_a))

    res_bubble = bubble_sort(arr_a)
    res_comb = comb_sort(arr_a)
    res_insert = insert_sort(arr_a)

    print('Результат сортировки: ')
    print(arr_a)
    print(res_bubble)
    print(res_comb)
    print(res_insert)

    print("Проверка совпадения результатов: ")
    print(check_arr(arr_true, res_bubble))
    print(check_arr(arr_true, res_comb))
    print(check_arr(arr_true, res_insert))
    print('===\n')
```

Листинг 3.6 — Реализация тестов

```
arr_a = generate_random_test_array(5)
arr_true = arr_a.copy()
arr_true.sort()
test_case([], [], 'Нулевой массив')
test_case(arr_a, arr_true, 'Общий случай')
```

3.5 Вывод

В данном разделе была представлена реализация алгоритма сортировки ”пузырьком”, алгоритма сортировки ”расчёски” и алгоритма сортировки вставками. Были разработаны алгоритмы тестирования разработанных методов по методу чёрного ящика.

4 Экспериментальный раздел

В данном разделе будут измерены временные характеристики алгоритмов умножения матриц и сделаны выводы об их временной эффективности.

Таблица 4.1 — Время работы алгоритма сортировки "пузырьком"

Размерность массива	Время сортировки
100	0.0
250	0.0
500	0.015625
750	0.03125
1000	0.03125
1250	0.046875
1500	0.078125
1750	0.09375
2000	0.140625
2250	0.171875
2500	0.234375

Таблица 4.2 — Время работы алгоритма сортировки "расчёской"

Размерность массива	Время сортировки
100	0.0
250	0.0
500	0.046875
750	0.09375
1000	0.15625
1250	0.28125
1500	0.390625
1750	0.515625
2000	0.703125
2250	0.796875
2500	1.03125

Таблица 4.3 — Время работы алгоритма сортировки ”вставками”

Размерность массива	Время сортировки
100	0.0
250	0.015625
500	0.015625
750	0.03125
1000	0.078125
1250	0.109375
1500	0.1875
1750	0.25
2000	0.28125
2250	0.359375
2500	0.4375

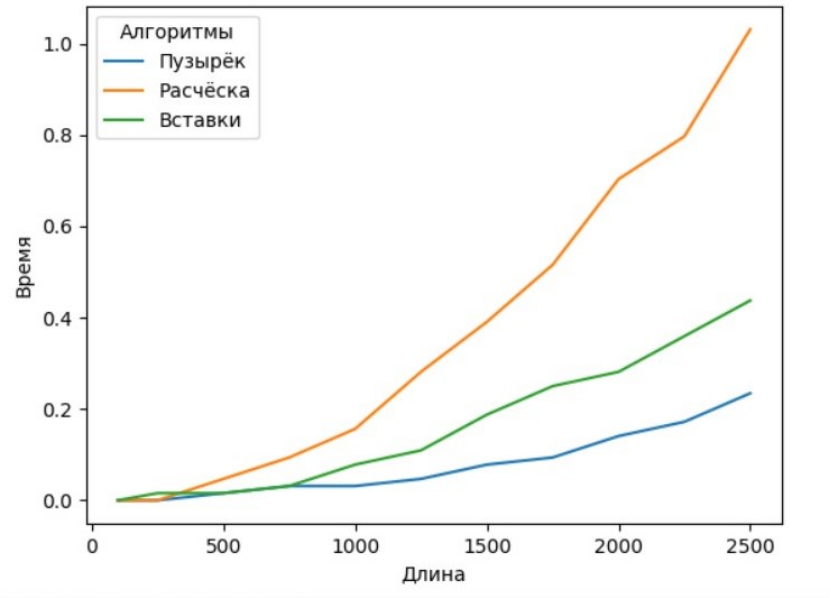

Рисунок 4.1 — График зависимости времени сортировки от размера массива

4.1 Вывод

В результате эксперимента было получено, что на массивах длинной 0 до 2500 элементов, алгоритм сортировки ”пузырьком” работает быстрее алгоритмов ”пузырька” и ”расчёски” в 1.8 и 4.4 раза соответсвтенно. В результате можно сделать вывод, что для сортировки массивов предпочтительно использовать реализованный в данной лабораторной работе алгоритм ”пузырька”.

Заключение

Список литературы

[1] 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 — 2-е изд. — «Вильямс», 2006. — С. 1296.

[3] Algorithms in C. Fundamentals/Data Structures/Sorting/Searching. — СПб.: ДиаСофтЮП, 2003. — С. 672.

[4] Magnus Lie Hetland. Python Algorithms: Mastering Basic Algorithms in the Python Language. — Apress, 2010. — 336 с.

[5] https://docs.python.org/3/index.html - 16.11.2021