МГТУ им. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Алгоритмы сортировки

Работу выполнил: Мокеев Даниил, ИУ7-54

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

Москва, 2019

Оглавление

Введение

На сегодняшний день существует не одна вариация алгоритмов сортировки. Все они различаются по скорости сортировки и по объему необходимой памяти. Цель данной лабораторной работы - обучение расчету трудоемкости алгоритмов

Алгоритмы сортировки часто применяются в практике программирования. В том числе в областях связанных с математикой, физикой, компьютерной графикой и т.д.

1 | Аналитическая часть

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.

1.2 Сортировка вставками

На каждом шаге выбирается один из элементов неотсортированной части массива (максимальный/минимальный) и помещается на нужную позицию в отсортированную часть массива.

1.3 Быстрая сортировка

Общая идея алгоритма состоит в следующем:

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».

3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения

2 | Конструкторская часть

2.1 Схемы алгоритмов

В данном разделе будут рассмотрены схемы алгоритмов пузырьком с флагом (2.1), сортировки вставками (2.2), быстрой сортировки (2.3).

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], ++, – получение полей класса

2. оценка трудоемкости цикла: Fц = a + N*(a + Fтела), где a - условие цикла

3. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будет приведены оценки трудоемкости алгоритмов. Построчная оценка трудоемкости сортировки пузырькм с флагом (Табл. 2.1).

2.2.1 Сортировка вставками

Табл. 2.1 Построчная оценка веса

Код	Вес
for i in range(1, len(a)):	1+n*2
key = a[i]	2
j = i-1	2
while (j >= 0 and a[j] > key):	n*4
a[j+1] = a[j]	4
j-= 1	1
a[j+1] = key	3

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.
Трудоемкость: $T(n) = 1 + 2n * (2 + 2 + 3) = 2n * 7 = 14n + 1 = O(n)$

Худший случай: массив отсортирован в обратном нужному порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.
Трудоемкость: $T(n) = 1+n*(2+2+4n*(4+1)+3) = 2n*n+7n+1 = O(n^2)$

2.2.2 Сортировка пузырьком

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $1 + 2n * (1 + 2n * 4) = 1 + 2n + 16n * n = O(n^2)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость: $1 + 2n * (1 + 2n * (4 + 5)) = O(n^2)$

2.2.3 Быстрая сортировка

Лучший случай: сбалансированное дерево вызовов[2] $O(n*log(n))$ В наиболее благоприятном 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 работает намного производительнее, и время ее работы описывается следующим рекуррентным соотношением: $T(n) = 2T(\frac{n}{2}) + O(n)$,где мы не обращаем внимания на неточность, связанную с игнорированием функций “пол” и “потолок”, и вычитанием 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, прилюбой постоянной пропорции разбиения полное время работы быстрой сортировки составляет $O(nlgn)$.

Худший случай: несбалансированное дерево [2] $O(n^2)$ Поскольку рекурсивный вызов процедуры разбиения, на вход которой подается массив размером 0, приводит к немедленному возврату из этой процедуры без выполнения каких-ли-бо 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

Сортировка пузырьком: лучший - $O(n)$, худший - $O(n^2)$
Сортировка вставками: лучший - $O(n)$, худший - $O(n^2)$
Быстрая сортировка: лучший - $O(nlgn)$, худший - $O(n^2)$

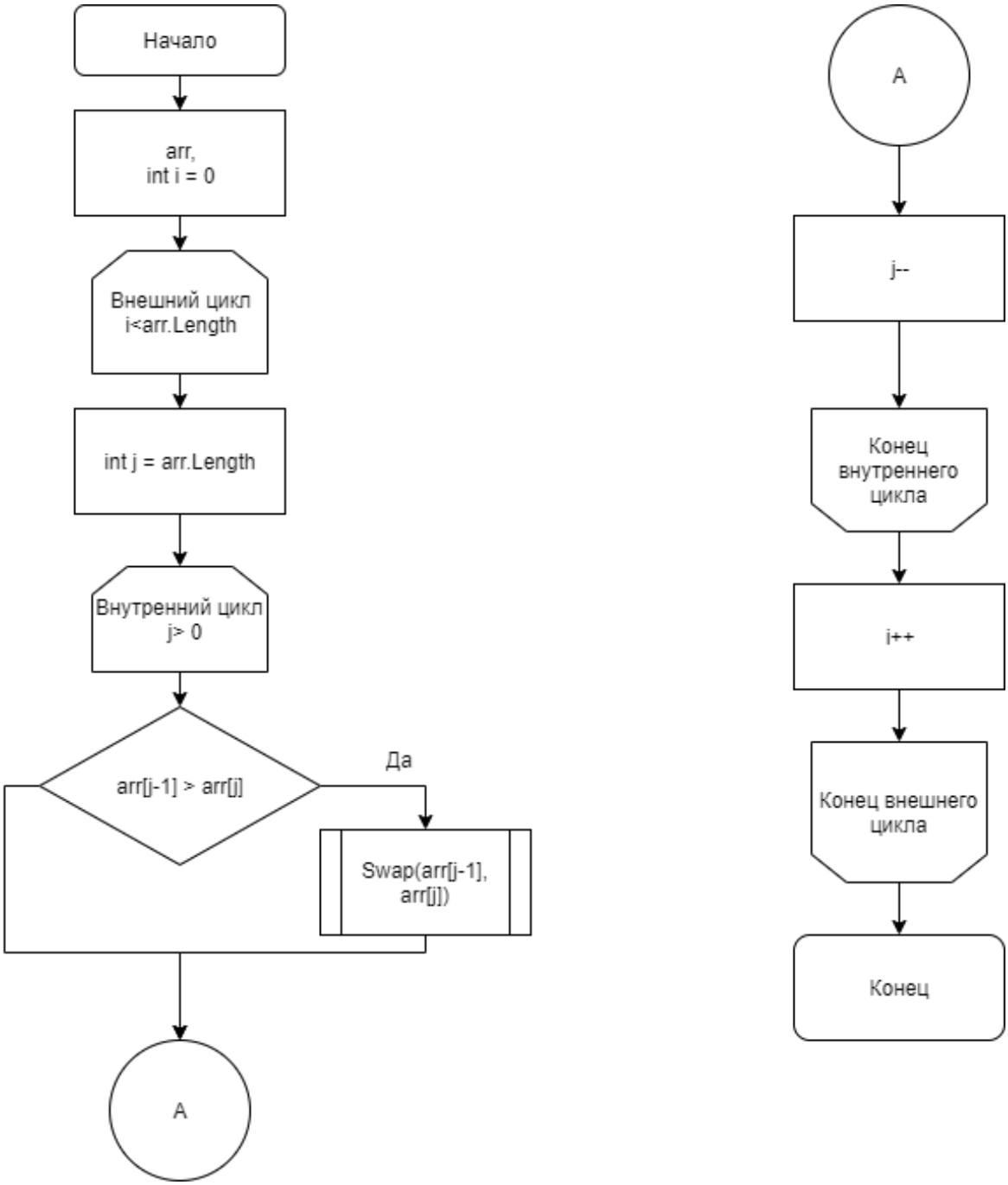

Рис. 2.1: Схема алгоритма сортировки пузырьком

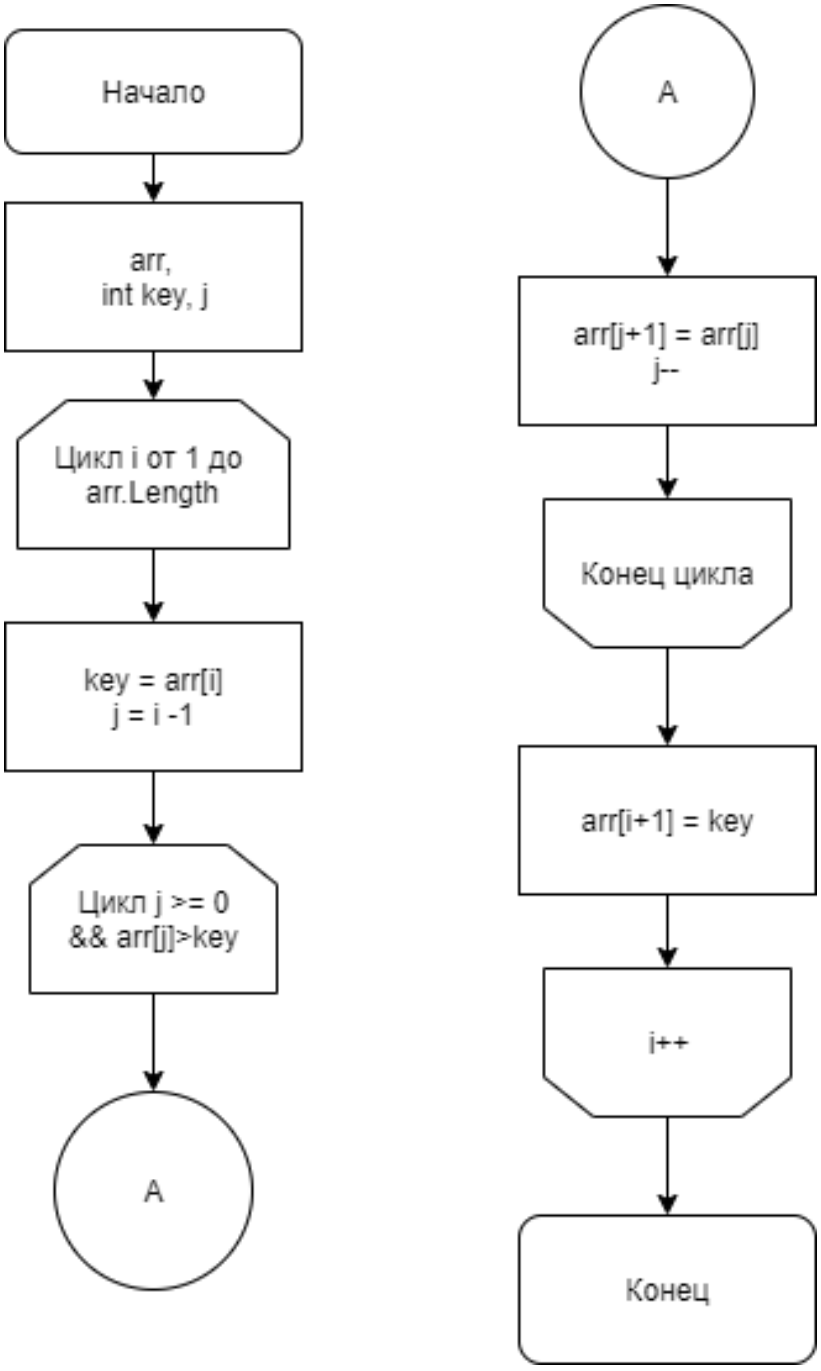

Рис. 2.2: Схема алгоритма сортировки вставками

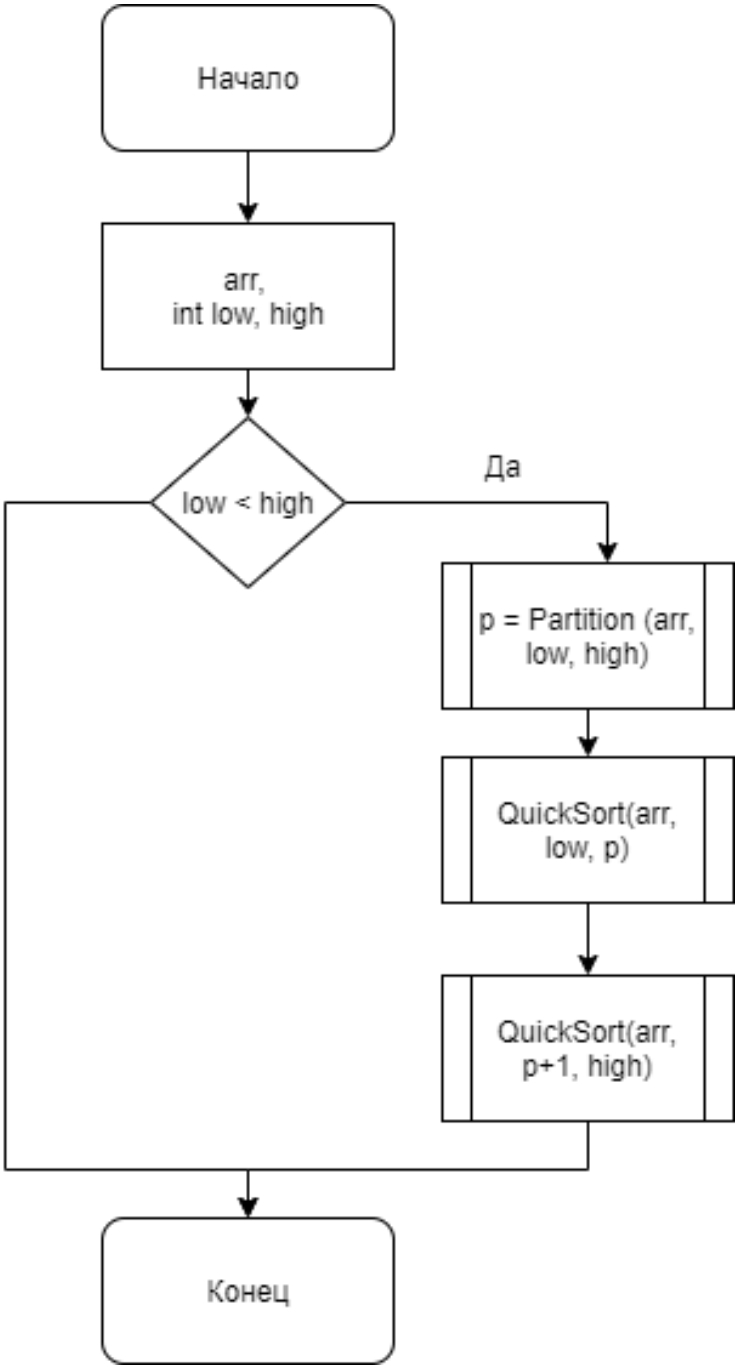

Рис. 2.3: Схема алгоритма быстрой сортировки

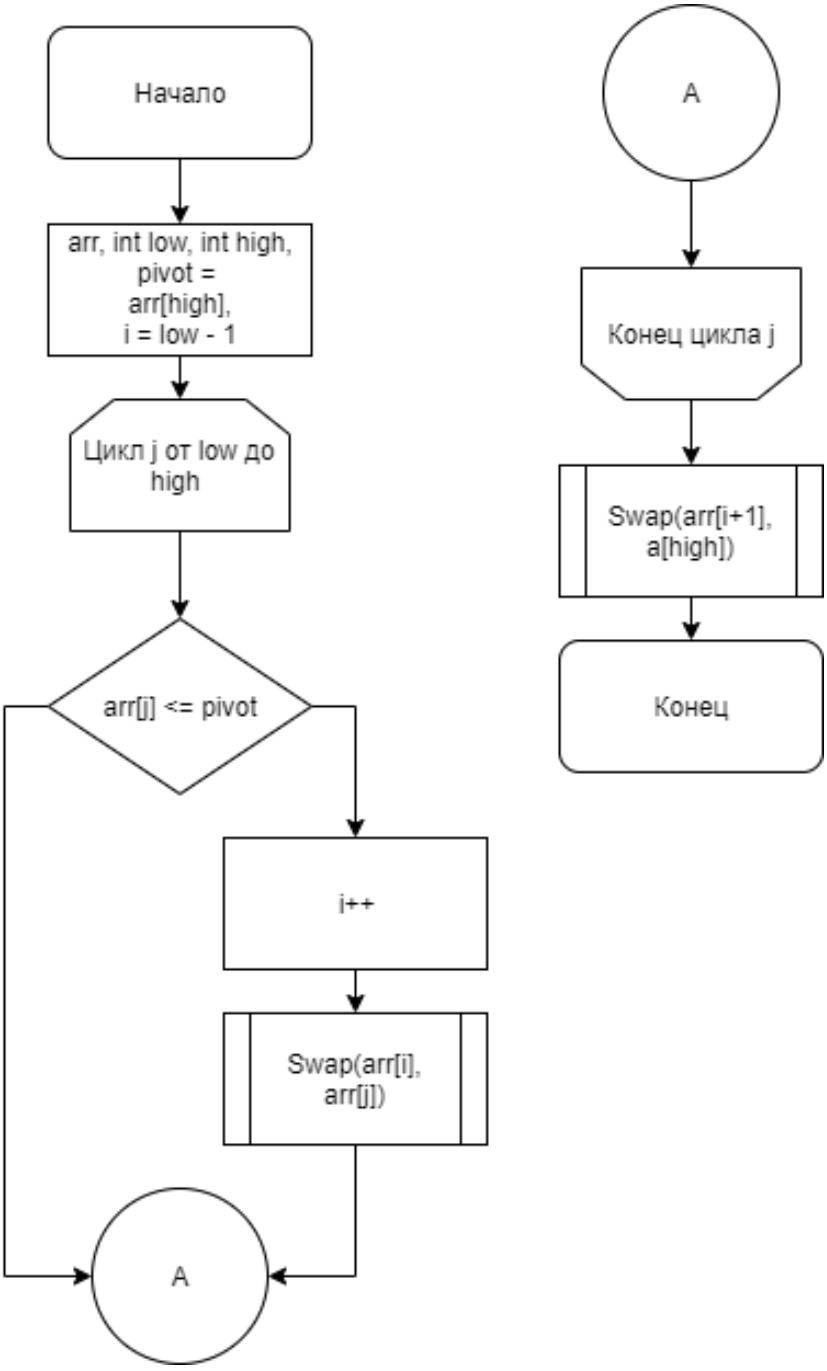

Рис. 2.4: Схема алгоритма Partition

3 | Технологическая часть

3.1 Выбор ЯП

Я выбрал в качестве Python языком программирования, потому как он достаточно удобен и гибок.

Время работы алгоритмов было замерено с помощью функции time() из библиотеки time.

3.2 Описание структуры ПО

Ниже представлена IDEF0 диаграмма (3.1), описывающая структуру программы.

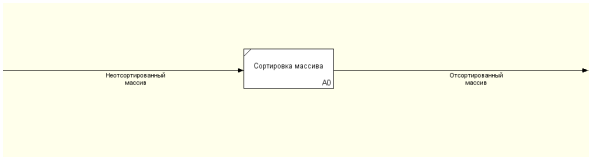

Рис. 3.1: Функциональная схема сортировки массива (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- lab03.py - файл, в котором находятся функции сортировки

Листинг 3.1: Сортировка пузырьком

```
def bubbleSort(a):
    for i in range(len(a)):
        for j in range(len(a)-1, 0, -1):
            if a[j-1] > a[j]:
                a[j-1], a[j] = a[j], a[j-1]

    return a
```

Листинг 3.2: Сортировка вставками

```
def insertSort(a):
    for i in range(1, len(a)):
        key = a[i]
        j = i-1
        while (j >= 0 and a[j] > key):
            a[j+1] = a[j]
            j -= 1
        a[j+1] = key
    return a
```

Листинг 3.3: Быстрая сортировка

```
def quickSort(a, low, high):
    if low < high:
        pi = partition(a, low, high)
        quickSort(a, low,pi)
        quickSort(a, pi+1, high)
    return a
```

Листинг 3.4: Partition

```
def partition(a,low, high):
    i = low - 1
    pivot = a[high]
    for j in range(low, high):
        if a[j] <= pivot:
            i += 1
            a[i], a[j] = a[j], a[i]
```

```
8       a[i+1],a[high] = a[high], a[i+1]
9       return (i+1)
```

4 | Исследовательская часть

Был проведен замер времени работы каждой из сортировок.

4.1 Примеры работы

```
Введите размер массива
6
a[0] = 3
a[1] = -6
a[2] = 0
a[3] = 8
a[4] = 34
a[5] = -67
Input array [3, -6, 0, 8, 34, -67]

Сортировка пузырьком: [-67, -6, 0, 3, 8, 34]

Сортировка вставками: [-67, -6, 0, 3, 8, 34]

Быстрая сортировка: [-67, -6, 0, 3, 8, 34]
```

Рис. 4.1: Пример работы программы

4.2 Постановка эксперемента

Будем измерять время работы алгоритмов на отсортированных, отсортированных в обратном порядке и случайно сгенерированных массивах. Для замера времени будем использовать функцию time.

4.2.1 Сортировка пузырьком

len	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
100	0.0	0.00969	0.00505
200	0.0101	0.0101	0.03002
300	0.01970	0.02792	0.04987
400	0.0233	0.07021	0.09032
500	0.08998	0.10028	0.16667

4.2.2 Сортировка вставками

len	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
100	0.0	0.0	0.0
200	0.0	0.0	0.0050
300	0.0	0.010990	0.01192
400	0.0	0.0246	0.0297
500	0.0	0.03731	0.044881

4.2.3 Быстрая сортировка

len	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
100	0.0	0.0	0.0
200	0.00991	0.01019	0.0
300	0.0149	0.00897	0.00099
400	0.0350	0.01993	0.0
500	0.04014	0.03486	0.00303

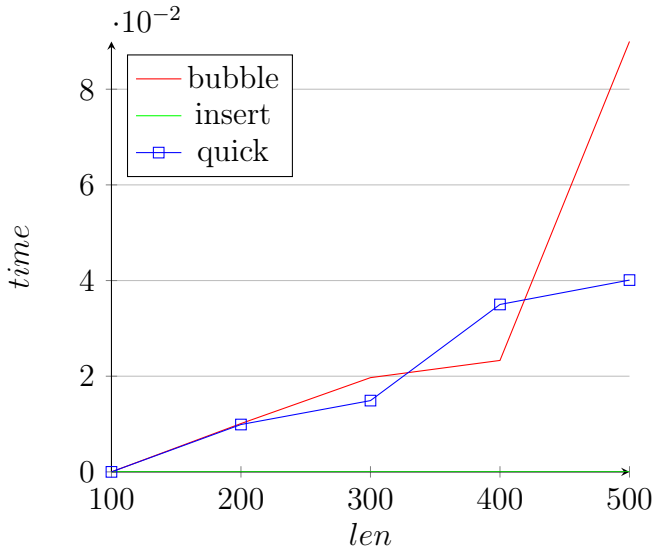

Рис. 4.2: Сравнение сортировки уже отсортированных массивов

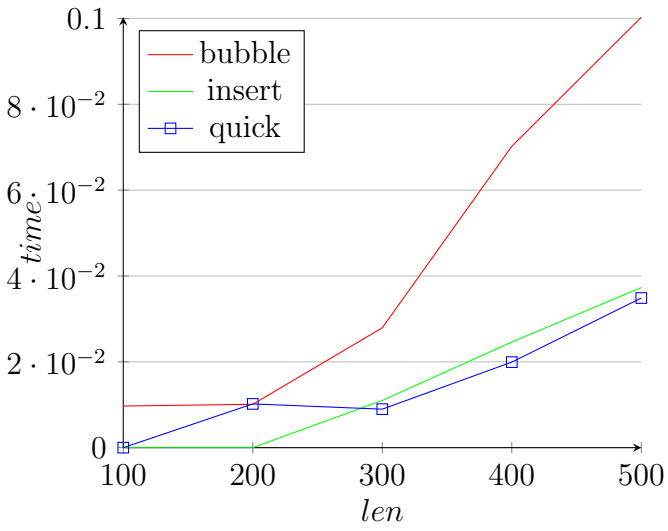

Рис. 4.3: Сравнение сортировки массивов, отсортированных в обратном порядке

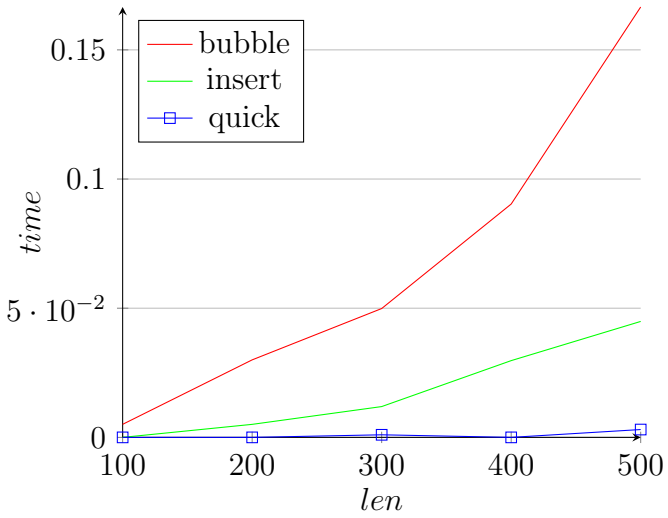

Рис. 4.4: Сравнение сортировки случайных массивов

4.2.4 Заключение эксперементальной части

Были протестированы алгоритмы сортировки на массивах размерами 100...500 с шагом 100. Рассмотрены отсортированные, отсортированные в обратном порядке массивы и массивы со случайными значениями.

В результате тестирования было получено, что лучшее время сортировки показывают на отсортированных массивах. Худшие значения сортировки показывают на обратно отсортированных массивах, причём чем больше размер такого массива, тем медленнее работают сортировки.

При сравнении времени работы алгоритмов можно сделать вывод, что на размерах массива до 300 элементов время работы алгоритмов приближено к 0 сек. При работе с массивами больших размеров сортировка пузырьком показывает наихудший результат. При сортировки случайных массивов лучшее время показывает быстрая сортировка.

Заключение

В ходе выполнения данной лабораторной работы были реализованы три алгоритма сортировки: сортировка пузырьком, сортировка вставками и быстрая сортировка. Был проведён анализ каждого алгоритма и измерено время работы алгоритмов для массивов разных размеров. Была оценена трудоёмкость алгоритмов.

Лучшее время работы все сортировки показывают на отсортированных массивах: при обработке массива в 500 элементов сортировка пузырьком - 0.005 сек, вставками и быстрая сортировка – 0.0 сек. Худшее время алгоритмы сортировки показали при работе с массивами, отсортированными в обратном порядке. Причём чем больше размерность такого массива, тем медленнее работает алгоритм.

Был сделан вывод, что на массивах размерностью до 300 элементов время работы всех трёх алгоритмов близится к 0 сек. При обработке массивов больших размеров, например, 500, лучший результат показала быстрая сортировка, худшее – сортировка пузырьком.

Список литературы

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 7. Быстрая сортировка // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005.

2. Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.