МГТУ им. Баумана

Лабораторная работа №1

По курсу: "Анализ алгоритмов"

Алгоритмы сортировки

Работу выполнил: Ерохин Никита ИУ7-51Б

Преподаватель: Волкова Л.Л.

Москва, 2020

Оглавление

Введение **3**

1 Аналитическая часть **4**
1.1 Цель работы 4
1.2 Описание алгоритмов 4
1.2.1 Сортировка пузырьком 4
1.2.2 Сортировка вставками 5
1.2.3 Быстрая сортировка 5
1.3 Модель вычислений 5

2 Конструкторская часть **6**
2.1 Оценка трудоемкости 6
2.1.1 Сортировка пузырьком 6
2.1.2 Сортировка вставками 6
2.1.3 Быстрая сортировка 7
2.2 Разработка алгоритмов 7

3 Технологическая часть **11**
3.1 Выбор ЯП 11
3.2 Сведения о модулях программы 11
3.3 Реализация алгоритмов 11
3.4 Тестирование программы по методу черного ящика 13

4 Исследовательская часть **14**
4.1 Сравнительный анализ на основе замеров времени работы алгоритмов 14

Заключение **17**

Введение

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. Эти алгоритмы сортировки оцениваются по времени. Работа посвящена реализации алгоритмов сортировок, их изучению, и рассчету трудоемкости реализаций некоторых алгоритмов.

1 | Аналитическая часть

1.1 Цель работы

Целью лабораторной работы является изучение алгоритмов сортировки. Необходимо рассчитать трудоемкость алгоритмов и сравнить их работу, реализовать описанные ниже алгоритмы:

- сортировка "пузырьком";
- сортировка вставками;
- быстрая сортировка.

1.2 Описание алгоритмов

Сортировка массива — одна из самых популярных операций над массивом. Алгоритмы реализуют упорядочивание элементов в списке.

1.2.1 Сортировка пузырьком

Алгоритм обходит массив от начала до конца, меняя местами неотсортированные соседние элементы. В результате первого прохода максимальный элемент оказывается в конце массива. Затем надо обойти неотсортированную часть массива (от первого элемента до предпоследнего) и повторить процедуру обмена мест неотсортированных соседних элементов. Второй по величине элемент массива окажется на предпоследнем месте. Таким образом при каждом проходе максимальный элемент неостортированной части оказывается ее последним элементом, как бы "всплывая- отсюда и появилось название сортировки "Пузырьком".

1.2.2 Сортировка вставками

Будем считать, что какая-то часть массива отсортирована. Алгоритм на каждом шаге берет элемент из неотсортированной части массива и ищет место, в отсортированной части массива, куда можно вставить текущий элемент, чтобы не нарушить порядок остортированной части. Изначально считаем, что отсортирован только первый элемент массива.

1.2.3 Быстрая сортировка

Алгоритм выбирает опорный элемент (существует несколько способов его выбора, в конкретной реализации - центральный (или средний) элемент массива). Далее массив делится на два подмассива: один состоит из элементов, меньших опорного, второй - больших опорного. Потом этот алгоритм применяется рекурсивно к двум подмассивам.

1.3 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

- операции, имеющие трудоемкость 1: $<$, $>$, $=$, $<=$, $=>$, $==$, $!=$,$+$, $-$, $*$, $/$, $+=$, $-=$, $*=$, $/=$, [ ];
- оператор условного перехода имеет трудоёмкость, равную трудоёмкости операторов тела условия;
- оператор цикла for имеет трудоемкость:

$$F_{for} = F_{init} + F_{check} + N * (F_{body} + F_{inc} + F_{check}), \tag{1.1}$$

  где F_{init} – трудоёмкость инициализации, F_{check} – трудоёмкость проверки условия, F_{inc} – трудоёмкость инкремента аргумента, F_{body} – трудоёмкость операций в теле цикла, N – число повторений.

2 | Конструкторская часть

Требования к программе:

- ввод размера массива;
- выбор типа массива (случайный порядок, отсортирован, отсортирован в обратном порядке);
- сортировка тремя способами;
- осуществление замера процессорного времени работы алгоритмов.

2.1 Оценка трудоемкости

Пусть N - длина массива. Рассмотрим трудоемкость трех алгоритмов сортировки.

2.1.1 Сортировка пузырьком

Лучший случай: Массив отсортирован
Трудоемкость: $3 + 3 * n + 4 * n^2/2 + 4 = 7 + 2 * n^2 + 3 * n = O(n^2)$

Худший случай: Массив не отсортирован
Трудоемкость: $3 + 3 * n + 12 * n^2/2 + 4 = 2 + 6n^2 + 3n = O(n^2)$

2.1.2 Сортировка вставками

Лучший случай: Массив отсортирован
Трудоемкость: $2 + 1 + n * (7) + 2 = 5 + 7 * n = O(n)$

Худший случай: Массив отсортирован в обратном порядке
Трудоемкость: $3 + n * 8 + (n^2/2) * 5 = 5/2 * (n^2) + 8n + 3 = O(n^2)$

2.1.3 Быстрая сортировка

Лучший случай: Каждый раз массив разделяется на два одинаковых по длине отрезка.
Трудоемкость: $7 + n + logn * (n + 7) = O(n * logn)$

Худший случай: Каждый раз выбирается отрезок длиной 1.
Трудоемкость: $7 + n * (1 + n + 7) = n^2 + 8n + 7 = O(n^2)$

2.2 Разработка алгоритмов

В данной части будут рассмотрены схемы алгоритмов:

- сортировки пузырьком. (рис. 2.1);
- сортировки вставками. (рис. 2.2);
- быстрой сортировки. (рис. 2.3).

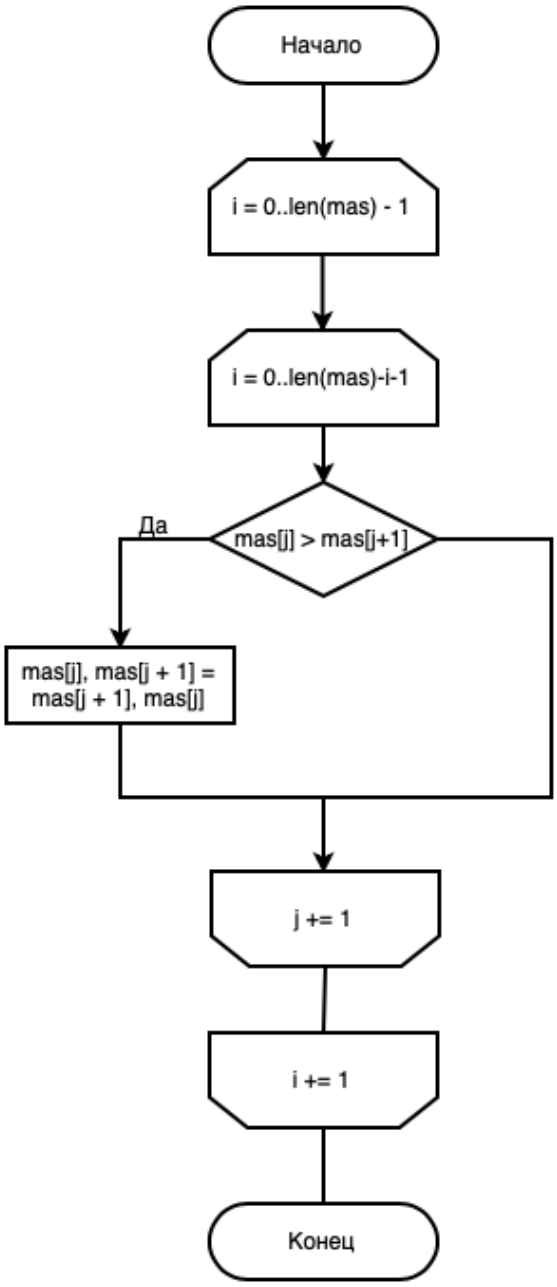

Рис. 2.1: Схема алгоритма сортировки пузырьком

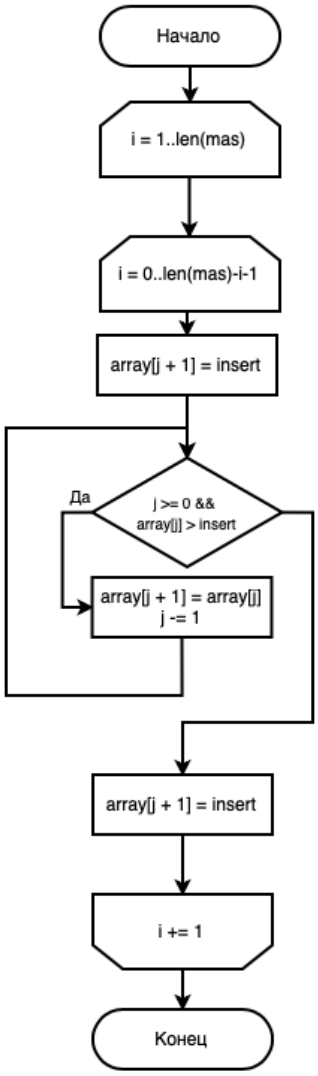

Рис. 2.2: Схема алгоритма сортировки вставками

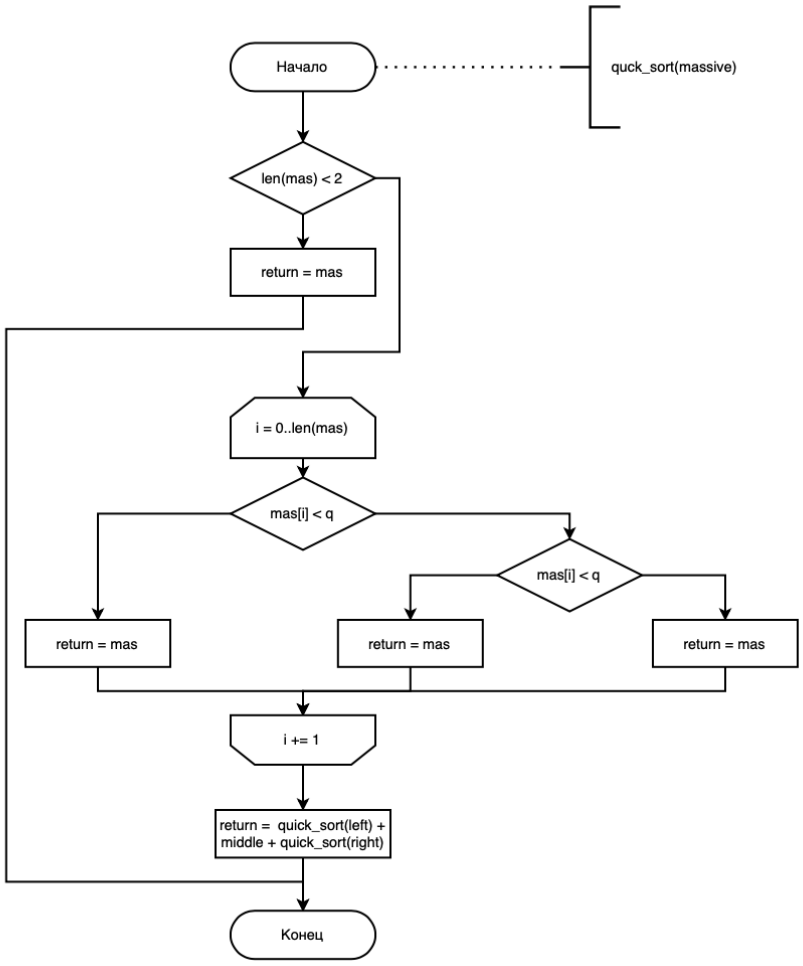

Рис. 2.3: Схема алгоритма быстрой сортировки.

3 | Технологическая часть

3.1 Выбор ЯП

Для реализации программ я выбрал язык программирования python, так как язык отличается простым и понятным синтаксисом.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - меню;
- sortimplementations.py - модуль с реализациями сортировок;
- stopwatch.py - модуль замера времени б

3.3 Реализация алгоритмов

Алгоритм сортирвоки "пузырьком"изобрашен на листинге 3.1.

Листинг 3.1: алгоритм сортировки пузырьком

```
1 def bubble_sort(array):
2
3     for i in range(len(array) - 1):
4         for j in range(0, len(array) - i - 1):
5             if array[j] > array[j + 1]:
6                 array[j], array[j + 1] = array[j + 1],
                      array[j]
7
```

```
8       return array
```

Алгоритм сортировки вставками изображен на листинге 3.2.

Листинг 3.2: Алгоритм сортировки вставками.

```
1  def insertion_sort(array):
2
3      for i in range(1, len(array)):
4          insert = array[i]
5          j = i - 1
6          while j >= 0 and array[j] > insert:
7              array[j + 1] = array[j]
8              j -= 1
9          array[j + 1] = insert
10
11     return array
```

Алгоритм быстрой сортировки изображен на листинге 3.3.

Листинг 3.3: Алгоритм быстрой сортировки

```
1  def quick_sort_impl(array):
2      if len(array) < 2:
3          return array
4      q = array[len(array) // 2]
5      left, middle, right = [], [], []
6      for i in array:
7          if i < q:
8              left.append(i)
9          elif i > q:
10             right.append(i)
11         else:
12             middle.append(i)
13
14     return quick_sort_impl(left) + middle + quick_sort_impl
           (right)
```

3.4 Тестирование программы по методу черного ящика

В данном разделе будут проведены результаты тестирования алгоритмов (рис. 3.1, 3.2, 3.3).

```
expect: [1, 3, 5, 5, 6, 7, 7, 8, 9, 9]
bubble: [1, 3, 5, 5, 6, 7, 7, 8, 9, 9]
insert: [1, 3, 5, 5, 6, 7, 7, 8, 9, 9]
quick : [1, 3, 5, 5, 6, 7, 7, 8, 9, 9]
```

Рис. 3.1: Тест 1

```
expect: [-10, -9, -9, -4, -4, -3, -3, 0, 2, 2, 4, 5, 6, 7, 10]
bubble: [-10, -9, -9, -4, -4, -3, -3, 0, 2, 2, 4, 5, 6, 7, 10]
insert: [-10, -9, -9, -4, -4, -3, -3, 0, 2, 2, 4, 5, 6, 7, 10]
quick : [-10, -9, -9, -4, -4, -3, -3, 0, 2, 2, 4, 5, 6, 7, 10]
```

Рис. 3.2: Тест 2

```
expect: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
bubble: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
insert: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
quick : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Рис. 3.3: Тест 3

4 Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был произведен замер времени работы каждого из алгоритмов на изначально отсортированном массиве(Рис. 4.1).

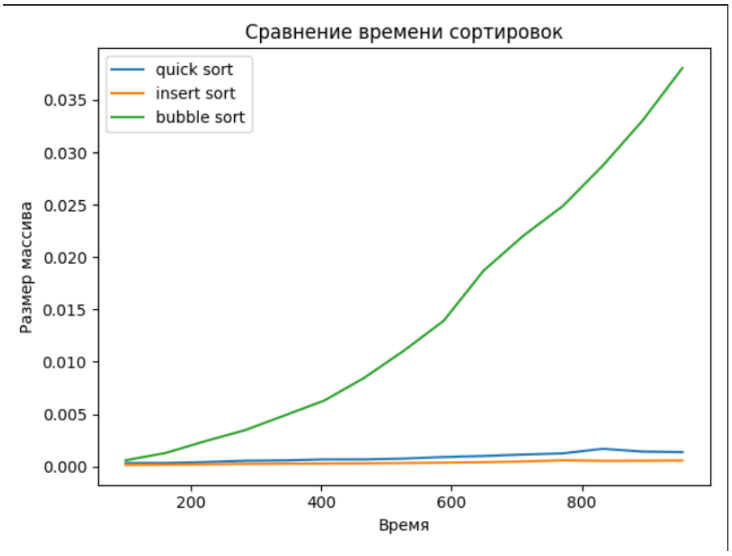

Рис. 4.1: Сравнение времени работы алгоритмов на изначально отсортированном массиве

Замер времени работы алгоритмов на массиве из случайных элементов(Рис. 4.2).

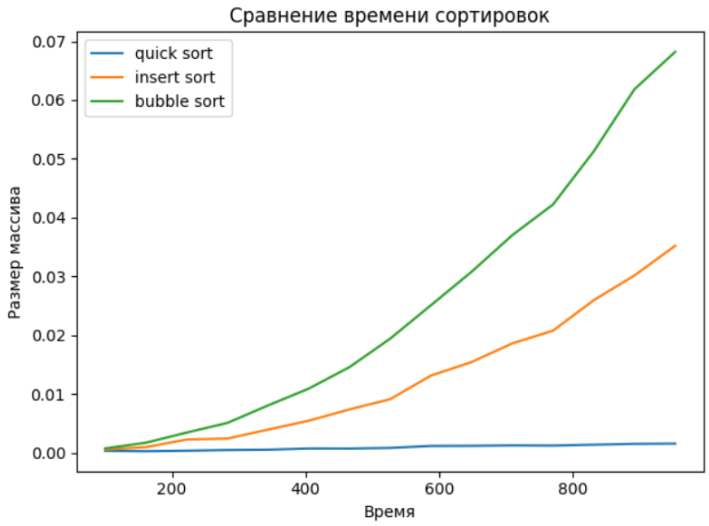

Рис. 4.2: Сравнение времени работы алгоритмов на массиве, состоящем из случайных чисел

Замер времени работы алгоритмов на массиве из чисел, отсортированных в обратном порядке(Рис. 4.3).

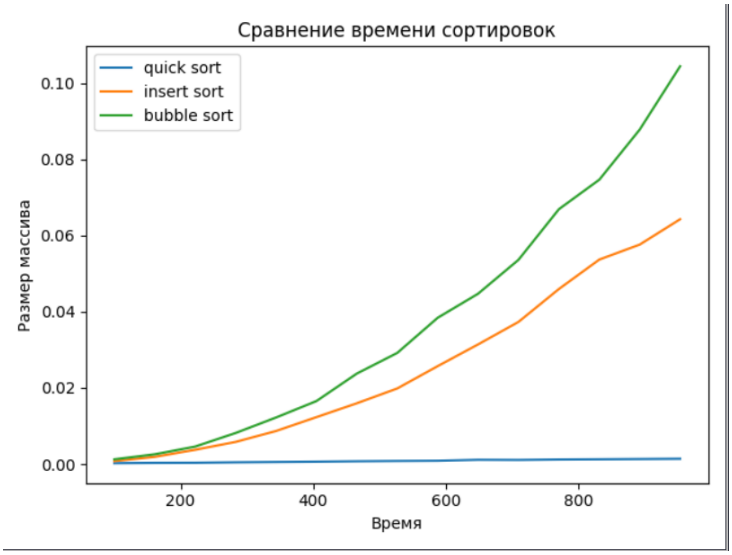

Рис. 4.3: Сравнение времени работы алгоритмов на массиве из чисел, отсортированных в обратном порядке

Из полученный графиков можно убедиться, что "быстрая"сортирвока является самой эффективной из представленных во всех случаях. Сортировка вставками является промежуточным этапом. Сортировка пузырьком всегда занимает больше всего времени.

Заключение

В ходе выполнения лабораторной работы были изучены и реализованы алгоритмы сортировки "пузырьком вставками, а так же алгоритм быстрой сортировки. Все задачи были выполнены, цель работы - достигнута.