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

Студент Топорков Павел
Группа ИУ7-53Б
Дисциплина Анализ алгоритмов

Преподаватели: _____ Строганов Ю.В., Волкова Л.Л.
подпись, дата Фамилия, И.О.

Оценка _____

Москва — 2021 г.

Оглавление

Введение

Задача о сортировке массива данных является на данный момент одной из самых распространенных задач в мире программирования. Так, сортировочные алгоритмы нашли себе широкое применение в следующих областях: экономика, статистика, физика, химия и многие другие.

Существует весьма большое количество разновидностей алгоритмов сортировок, а чтобы уметь выбирать оптимальные алгоритмы под определенные задачи и условия необходимо уметь сравнивать и оценивать эти алгоритмы.

Целью данной лабораторной работы является изучение применений алгоритмов сортировки и обучение расчету трудоемкости алгоритмов.

1 Аналитическая часть

1.1 Сортировка вставками

На каждой итерации алгоритма выбирается один из элементов неотсортированной части массива (максимальный либо минимальный в зависимости от направления сортировки) и помещается на нужную позицию в отсортированную часть массива. [1] Таким образом происходит вставка элемента на нужную позицию.

1.2 Сортировка пузырьком

Алгоритм проходит по массиву n-1 раз или до тех пор, пока массив не будет полностью отсортирован. В каждом проходе элементы попарно сравниваются и, при необходимости, меняются местами. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент ставится на своё место в конец неотсортированного массива. [1] Таким образом наибольшие элементы "всплывают"как пузырек, отчего. собственно, и произошло название данного алгоритма.

1.3 Быстрая сортировка

Алгоритм быстрой сортировки (сортировки Хоара) заключается в следующем:

1. массив разбивается на два (возможно пустых) подмассива таких, что в одном подмассиве каждый элемент меньше либо равен опорному, и при этом не превышает любой элемент второго подмассива;

2. подмассивы сортируются с помощью рекурсивного вызова процедуры быстрой сортировки.

Стоит отметить, что поскольку подмассивы сортируются на месте, для их объединения не трубуются никакие действия. [1]

Вывод

В этом разделе были рассмотрены алгоритмы сортировки вставками, пузырьком и быстрой сортировки. Вычислительная сложность данных алгоритмов будет рассмотрена в следующем разделе.

2 Конструкторская часть

2.1 Требования к программе

К вводимым данным существуют следующие требования:

1. на вход подается массив целых чисел, который следует отсортировать;
2. предусмотреть генерацию случайного массива заданной длины.

Непосредственно к программе применяются следующие требования:

1. корректная сортировка массивов;
2. если введены недопустимые данные (например, размерность массива меньше нуля), программа не должна аварийно завершаться.

2.2 Схемы алгоритмов

Далее будут приведены схемы алгоритмов сортировки вставками (рис. 2.1) и пузырьком (рис. 2.2).

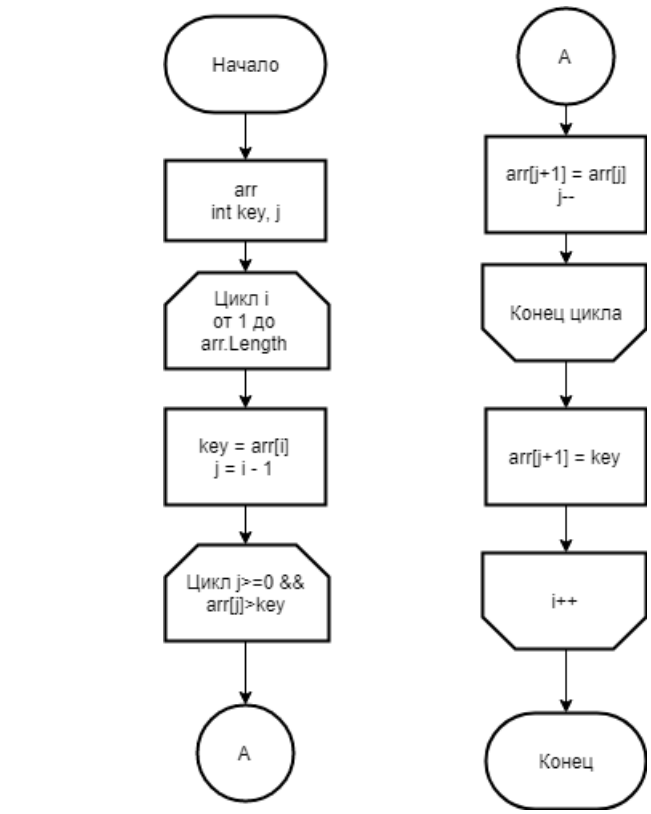

Рисунок 2.1: Схема алгоритма сортировки вставками

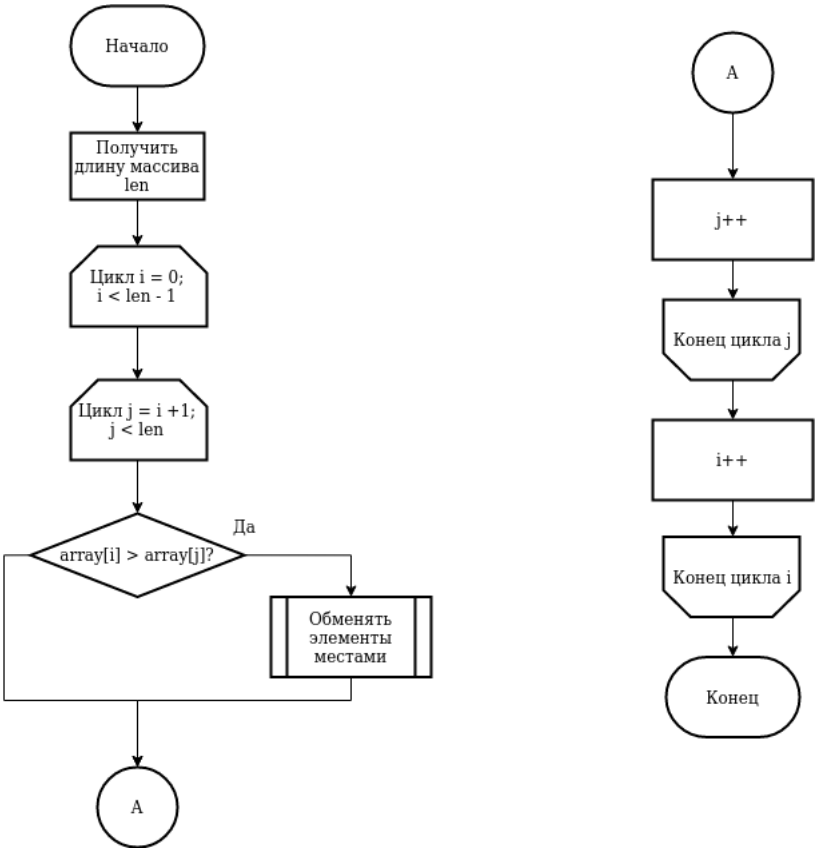

Рисунок 2.2: Схема алгоритма сортировки пузырьком

2.3 Трудоемкость алгоритмов

2.3.1 Модель оценки трудоемкости

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1: $+, -, *, /, =, ==, <=, >=, !=, +=$, [];
2. оценка трудоемкости цикла: $F_{cycle} = a + N * (a + F_{body})$, где a - условие цикла
3. стоимость условного перехода будем считать равным 0;
4. стоимость вычисления условия остаётся.

2.3.2 Алгоритм сортировки вставками

В таблице 2.1 приведено построчная оценка трудоемкости алгоритма сортировки вставками.

Таблица 2.1: Построчная оценка трудоемкости сортировки вставками

Строчка кода	Вес
for j in range(1, length):	3
i = j - 1	2
while (i >= 0) and (array[i] > value):	4
array[i + 1] = array[i]	4
i -= 1	1
array[i + 1] = value	3

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.
Трудоемкость: $T(n) = 3n + ((2 + 4 + 1 + 3) * (n - 1)) = 3n + 10(n - 1) = 13n - 10 = O(n)$

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: $T(n) = 3n+(2+2)(n-1)+4\left(\frac{n(n+1)}{2}-1\right)+5\frac{n(n-1)}{2}+3(n-1) = 3n+4n-4+2n^2+2n-4+2.5n^2-2.5n+3n-3 = 4.5n^2+9.5n-11 = O(n^2)$

2.3.3 Алгоритм сортировки пузырьком

В таблице 2.2 приведено построчная оценка трудоемкости алгоритма сортировки пузырьком.

Таблица 2.2: Построчная оценка трудоемкости сортировки пузырьком

Строчка кода	Вес
for i in range(length - 1):	4
for j in range(i + 1, length):	4
if array[i] > array[j]:	3
temp = array[i]	2
array[i] = array[j]	3
array[j] = temp	2

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $n * (2 + 7 + 1 + 3) + 2 = 13n + 2 = O(n)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость: $n * (n * (4 + 4 + 3 + 2 + 3 + 2)) = 17n^2 = O(n^2)$

2.3.4 Алгоритм быстрой сортировки

Лучший случай: сбалансированное дерево вызовов $O(n * log(n))$ В наиболее благоприятном случае сортировка двух подмассивов сводится 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 ; $T(n) = O(nlogn)$. При сба-

лансированности 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, при любой постоянной пропорции разбиения полное время работы быстрой сортировки составляет $O(nlogn)$.

Худший случай: несбалансированное дерево $O(n^2)$ 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 $O(n^2)$. [1]

Вывод

В данном разделе были рассмотрены требования для программы, схемы алгоритмов сортировки вставками и пузырьком, а также была рассчитана трудоемкость алгоритмов, на основе введенной модели.

Результаты расчетов следующие:

- сортировка пузырьком: лучший - $O(n)$, худший - $O(n^2)$;
- сортировка вставками: лучший - $O(n)$, худший - $O(n^2)$;
- быстрая сортировка: лучший - $O(nlogn)$, худший - $O(n^2)$;

При этом сортировка вставками быстрее пузырька с флагом в худшем случае т.к. имеет меньший коэффициент: $4.5n^2$ и $18n^2$ соответственно.

3 Технологическая часть

3.1 Выбор ЯП

Для реализации программы был выбран язык программирования Python 3 в следствие относительной простоты написание кода на данном ЯП. [2] Среда разработки - редактор кода VS Code. [3]

3.2 Листинги кода алгоритмов

Далее будут приведены листинги алгоритмов сортировки вставками (листинг 3.1), алгоритма сортировки пузырьком (листинг 3.2) и алгоритм быстрой сортировки (листинг 3.3).

Листинг 3.1: Алгоритм сортровки вставками

```
def insertion_sort(array):
  for j in range(1, len(array)):
    i = j - 1
    value = array[j]

    while (i >= 0) and (array[i] > value):
      array[i + 1] = array[i]
      i -= 1

  array[i + 1] = value
```

Листинг 3.2: Алгоритм сортировки пузырьком

```
def bubble_sort(array):
  length = len(array)

  for i in range(length - 1):
    for j in range(i + 1, length):
      if array[i] > array[j]:
        temp = array[i]
        array[i] = array[j]
        array[j] = temp
```

Листинг 3.3: Алгоритм быстрой сортировки

```
def qsort(array, first, last):
  if first < last:
    f = first
    l = last
    base = array[(first + last) // 2]

    while f <= l:
```

```
8         elem_f = array[f]
9         while (elem_f < base) and (f <= last):
10          f += 1
11          elem_f = array[f]
12
13        elem_l = array[l]
14        while (elem_l > base) and (l >= first):
15          l -= 1
16          elem_l = array[l]
17
18        if f <= l:
19          temp = array[f]
20          array[f] = array[l]
21          array[l] = temp
22          f += 1
23          l -= 1
24      qsort(array, first, l)
25      qsort(array, f, last)
```

3.3 Тесты

Проведем тестирование программы в таблице 3.1.

В столбце "Исходный массив"и "Отсорт. массив"приведены исходные массивы для сортировки и ожидаемые отсортированные массивы. В столбце Алг. №1 содержится результат выполнения алгоритма сортировки вставками, в Алг. №2 - алгоритма сортировки пузырьком, а в Алг. №3 - быстрой сортировки соответственно. Сортировать будем по возрастанию.

Таблица 3.1: Таблица тестовых данных

№	Исходный массив	Отсорт. массив	№1	№2	№3
1	[]	[]	[]	[]	[]
2	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]
3	[3, 2, 1]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]
4	[2, 3, 1]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]
5	[2, 1, 3, 1]	[1, 1, 2, 3]	[1, 1, 2, 3]	[1, 1, 2, 3]	[1, 1, 2, 3]

Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы; также убедились в корректности работы разработанного ПО.

4 Исследовательская часть

4.1 Сравнение алгоритмов

Был проведен замер времени работы каждого из алгоритмов на машине HP ProBook 450 G5. [4]

В рамках данного эксперимента было произведено сравнение времени выполнения трех алгоритмов в лучшем/худшем/случайном случае заполнения массива. При длине массивов от 100 до 1000 элементов с шагом 100. На графике 4.1 показано сравнение времени в лучшем случае, на графике 4.2 - сравнение времени в худшем случае, на графике 4.3 сравнение времени при случайном заполнении массива. По оси l идет длина массива, а по оси t - время сортировки в секундах. Для минимизации погрешности замеров времени каждый алгоритм исполнялся над одними и теми же строками 100 раз и затраченное время делилось на 100, для получения усредненного времени выполнения.

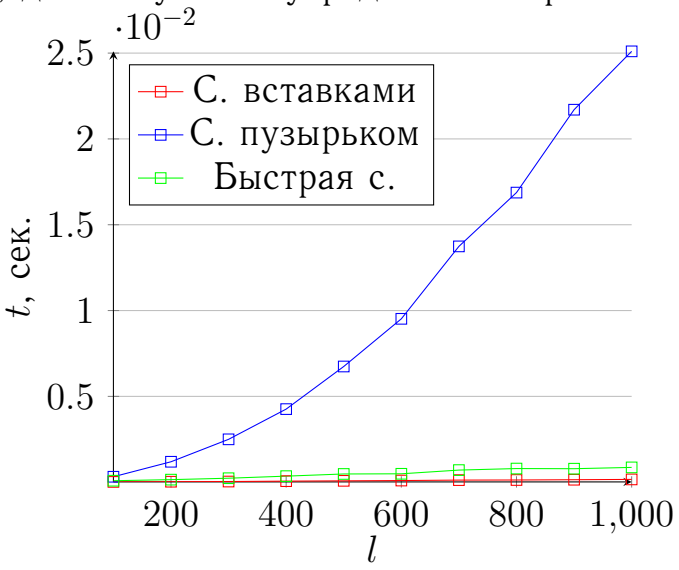

Рис. 4.1: Сравнение времени выполнения в лучшем случае

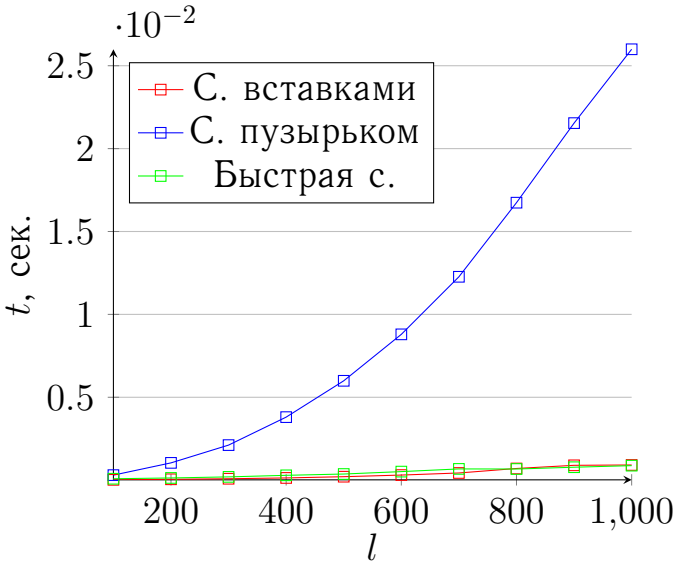

Рис. 4.2: Сравнение времени выполнения в худшем случае

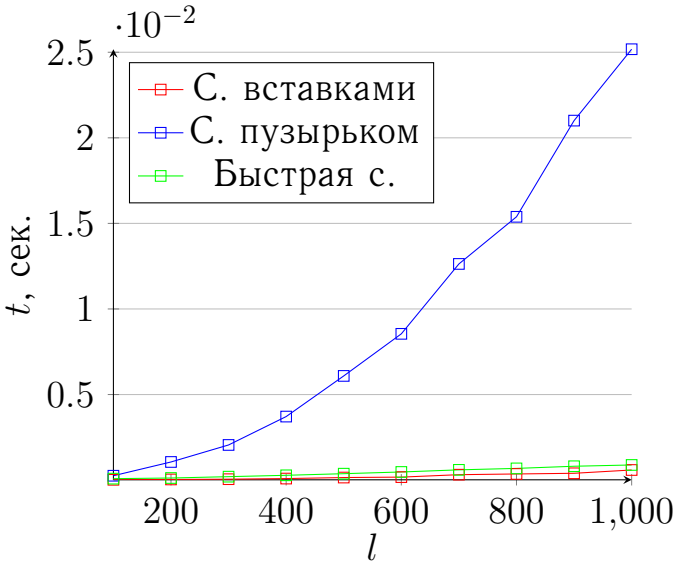

Рис. 4.3: Сравнение времени выполнения при случайном заполнении массива

Вывод

Рассматривая графики затрат времени при худшем случае алгоритм быстрой сортировки показывает наихудшие результаты: он медленнее примерно в 10 раз сортировки пузырьком. При худшем случае ситуация обратная: тут зависимость видна исходя их графиков функции $n * log(n)$ для быстрой сортировки и n^2 для сортировок пузырьком и выбором. Для случайного заполнения массива ситуация аналогична лучшему случаю.

Сравнивая алгоритмы пузырьковой сортировки и сортировки выбором можно заметить, что сортировка вставками работает быстрее за счет меньшего коэффициента при старшей степени (n^2).

Заключение

В ходе работы мною были изучены такие алгоритмы сортировки массива как сортировка вставками, пузырьком и быстрая сортировка.

Было выполнено сравнение всех рассматриваемых алгоритмов. Рассматривая графики затрат времени при худшем случае алгоритм быстрой сортировки показывает наихудшие результаты: он медленнее примерно в 1000% раз сортировки пузырьком. При худшем случае ситуация обратная тут зависимость видна исходя их графиков функции $nlog(n)$ для быстрой сортировки и n^2 для сортировок пузырьком и выбором. Для случайного заполнения массива ситуация аналогична лучшему случаю. Сравнивая алгоритмы пузырьковой сортировки и сортировки выбором нужно заметить, что сортировка вставками работает быстрее за счет меньшего коэффициента при n^2. Так, множитель при n^2 в сложности алгоритме сортировки пузырьком примерно в 4 раза выше, чем тот же множитель в сложности алгоритма сортировки вставками.

Литература

[1] К. Кормен Т. Лейзерсон Ч. Ривест Р. Штайн. Алгоритмы: построение и анализ. Вильямс, 2019. С. 198–219.

[2] Язык программирования Python. Режим доступа: `http://www.python.org` (дата обращения: 01.10.2020).

[3] Редактор кода VS Code. Режим доступа: `https://code.visualstudio.com/` (дата обращения: 01.10.2020).

[4] Технические характеристики ноутбука HP ProBook 450 G5 PC. Режим доступа: `https://support.hp.com/ru-ru/document/c05739572` (дата обращения: 01.10.2020).