Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 3

Трудоемкость алгоритмов сортировок

Студент:
Сиденко Анастасия Генадьевна
Группа: ИУ7-53Б

Преподаватели:
Строганов Юрий Владимирович
Волкова Лилия Леонидовна

2019 г.

Содержание

Введение

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Сортировки используются в самом широком спектре задач, включая обработку коммерческих, сейсмических, космических и прочих данных[1]. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными и т.п.

Сортировка применяется во всех без исключения областях программирования, например, базы данных или математические программы. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях.

В учебнике [2, стр. 21] Д. Кнута упоминается что «по оценкам производителей компьютеров в 60-х годах в среднем более четверти машинного времени тратилось на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Исходя из этих статистических данных, можно заключить, что либо (i) сортировка имеет много важных применений, либо (ii) ею часто пользуются без нужды, либо (iii) применяются в основном неэффективные алгоритмы сортировки».

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным.

К примеру, на сайте [3] можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности.

В данной работе требуется провести обзор 3 популярных алгоритмов сортировки.

1. Изучить алгоритмы сортировки массивов данных: быстрая сортировка, сортировка вставками и поразрядная сортировка.
2. Оценить трудоемкости алгоритмов, произвести теоретическую оценка для лучших и худших и случаев и условий их наступления.
3. Получить практические навыки реализации алгоритмов сортировки на одном из языков программирования.
4. Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (зависимость времени от длины массива)
5. Экспериментально подтвердить различия в трудоемкости алгоритмов с указанием лучших и худших случаев.

1 Аналитическая часть

Под сортировкой обычно понимают процесс перестановки объектов данного множества в определенном порядке. Цель сортировки – облегчить последующий поиск элементов в отсортированном множестве. Таким образом, сортировки присутствуют во всех областях.

1.1 Описание задачи

Сортировка – это процесс упорядочения некоторого множества элементов, на котором определены отношения порядка >, <, >=, <= (по возрастанию или убыванию).

При выборе алгоритмов сортировки необходимо поставить перед собой вопрос. Существует ли наилучший алгоритм? Имея приблизительные характеристики входных данных, можно подобрать метод, работающий оптимальным образом.

Рассмотрим параметры, по которым будет производиться оценка алгоритмов.

1. Число операций сортировки (сравнения и перемещения) – параметры, характеризующие трудоемкость алгоритма.
2. Время работы – скорость работы на различных длинах массивов. Некоторые алгоритмы сортировки зависят от данных, например, если первоначально данные упорядочены, время может значительно сократиться, тогда как другие методы оказываются нечувствительными к этому свойству.

   Чтобы учитывать этот факт, будет рассматривать: лучшие, худшие и произвольные случаи.

Как было сказано, в данной работе необходимо оценить и подтвердить трудоемкость алгоритмов, обратимся к определению.
Трудоемкость алгоритма– это зависимость количества операций от объема обрабатываемых данных. Модель вычислений:

1. Цена едичных операций. Пусть у следующих операций трудоемкость равна 1:

$$+ \ , - \ , * \ , / \ , \% \ , = \ , == \ , ! = \ , <> \ , <= \ , >= \ , [] \ , + =$$

2. Трудоемкость улсовного перехода примем за единицу, при этом условие вычисляется по пункту 1.
3. Трудоемкость циклов, например цикла for:

$$f_{for} = f_{init} + f_{comp} + N * (f_{body} + f_{inc} + f_{comp})$$

1.2 Пути решения

Сортировка — один из базовых видов активности или действий, выполняемых над предметами. Ещё в детстве детей учат сортировать, развивая мышление. Компьютеры и программы — тоже не исключение. И поэтому в настоящее время существует огромное количество алгоритмов сортировок[4], которые были придуманы и используются для разных задач.

Первые прототипы современных методов сортировки появились уже в XIX веке для ускорения обработки данных переписи населения в США[5]. В дальнейшем история алгоритмов оказалась связана с развитием электронно-вычислительных машин. Множество ученых на протяжении десятилетий писали труды и придумывали новые и новые алгоритмы сортировок[5].
Наглядно некоторые из них можно посмотреть в народных танцах[6].

1.3 Выводы

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно, быстрой, вставками и поразрядной. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.
Реализуем данные алгоритмы и проверим наши предположения экспериментально.

2 Конструкторская часть

В данной работе стоит задача реализации алгоритмов сортировки быстрой, вставками и поразрядной. Необходимо рассмотреть, изучить и оценить данные варианты реализации.

2.1 Функциональная модель

На рисунке 1 представлена функциональная модель нашей задачи.

Рис. 1 - Функциональная модель алгоритма сортировки.

2.2 Схемы алгоритмов

Приведем схемы 3 исследуемых алгоритмов (см. рисунки 2-4).

Поразрядная сортировка

Рис. 2 - Алгоритм поразрядной сортировки

Сортировка вставками

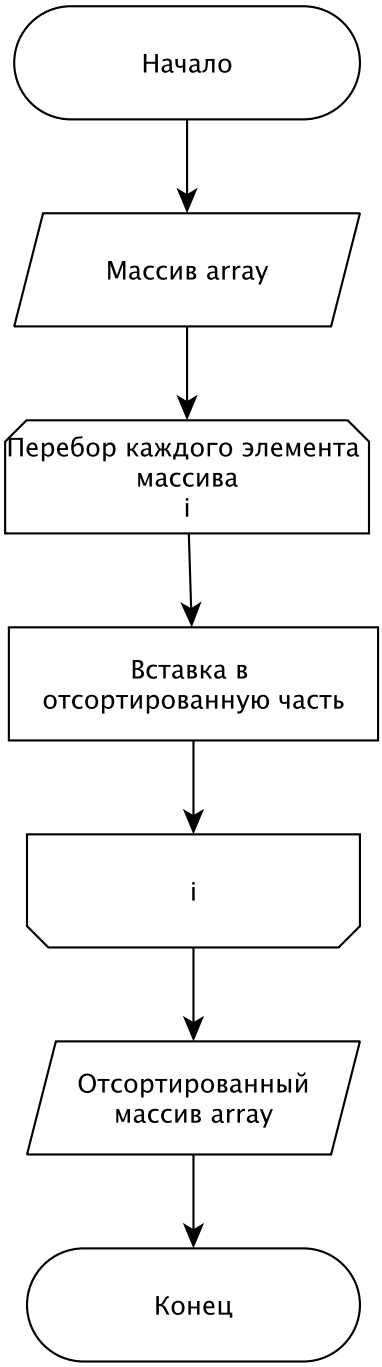

Рис. 3 - Алгоритм сортировки вставками

Быстрая сортировка

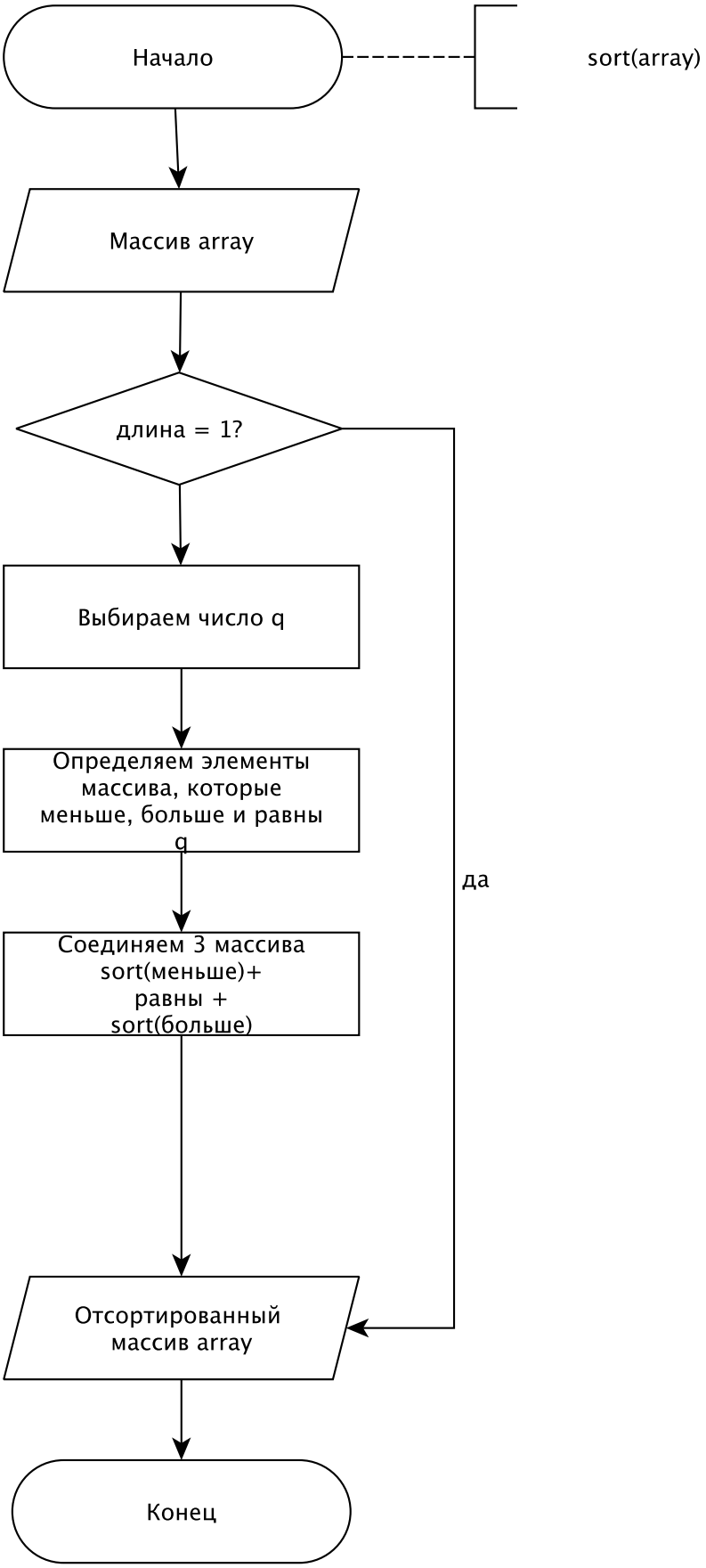

Рис. 4 - Алгоритм быстрой сортировки

Произведем теоретическую оценку трудоемкости алгоритмов сортирвки

1. Поразрядная сортировка

$$f = len + 2 + digit(2 + \underbrace{2 + len(2+1)}_{\text{расстановка}} + \underbrace{2 + digit(2+1)}_{\text{запись в массив}} =$$

$$= len + 2 + digit(6 + 3len + 2digit) = len + 2 + 6digit + 2digit^2 + 3digit \cdot len$$

$f = O(nk), n -$ размер массива, $k -$ порядок числа

чем больше порядок числа тем больше трудоемкость

2. Сортировка вставками

$$f = 2 + len(2 + \underbrace{2 + key(2+1)}_{\text{поиск позиции(сравнение)}} + \underbrace{1}_{\text{вставка}}) =$$

$$= 2 + len(5 + 3key) = 2 + 5len + 3len \cdot key$$

$f = O(n), n -$ размер массива, **Лучший случай - упорядоченный массив**

$f = O(n^2), n -$ размер массива, **Худший случай - обратноупорядоченный массив**

$f = O(n^2), n -$ размер массива, **Средний случай**

3. Быстрая сортировка (из [7])

$f = O(n \cdot ln(n)), , n -$ размер массива, **Лучший случай- упорядоченный массив**

$f = O(n^2), n -$ размер массива, **Худший случай - обратноупорядоченный массив**

$f = O(n \cdot ln(n)), n -$ размер массива, **Средний случай**

2.3 Выводы

Сортировка является одной из типовых проблем обработки данных и обычно понимается как задача размещения элементов неупорядоченного набора значений в порядке монотонного возрастания или убывания.

Возможные способы решения этой задачи широко обсуждаются в литературе.

Вычислительная трудоемкость процедуры упорядочивания является достаточно высокой. Так, для ряда известных простых методов (пузырьковая сортировка, сортировка включением и др.) количество необходимых операций определяется квадратичной зависимостью от числа упорядочиваемых данных. Для более эффективных алгоритмов трудоемкость ниже.

Однако для различных сортировок можно получить меньшую трудоемкость на частных вариантах задачи.

В работе рассматриваются сортировки: поразрядная, вставками и быстрая. Наименьшей трудоемкостью обладает поразрядная сортировка, хотя в некоторых случаях быстрая сортировка будет у нее выигрывать. Рассмотрим это далее.

3 Технологическая часть

Стоит задача разработки и сравнительного анализа алгоритмов сортировки массивов.

В реализациях в целях увеличения точности подсчета времени вывод массива был вынесен за пределы функций-алгоритмов. В целях наглядности были опущены части программ, не относящиеся к работе алгоритмов.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность замеров процессорного времени выполнения реализации каждого алгоритма. Требуется провести замеры для варьирующихся размеров массива до 10000 элементов. Один эксперимент ставится не менее 50 раз, результат одного эксперимента рассчитывается как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

3.2 Средства реализации

В качестве языка программирования был выбран Python[8], так как я знакома с этих языком программирования.

Для замеров времени была выбран метод *process_time*(), возвращает текущее время процессора как число с плавающей запятой, выраженное в секундах в Unix.

Для генерации случайных чисел использовался метод *randint*().

3.3 Листинг кода

Поразрядная сортировка:

```
        n = pow(digit, n)
        i = 1
        while (i < n):
                sort = [[] for k in range(digit)]

                for x in array:
                        sort[get_digit(x, i)].append(x)

                count = len(array)
                array = [0] * count
                u = 0
                w = 0
                for k in range(digit):
                        for j in range(len(sort[k])):
                                if (sort[k][j] < 0):
                                        array[w] = sort[k][j]
                                        w += 1
                                else:
                                        array[u + neg] = sort[k][j]
                                        u += 1
                i *= 10
```

Сортировка вставками:

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

Быстрая сортировка:

```python
def partition(nums, low, high):
    middle = nums[(low + high) // 2]
    i = low - 1
    j = high + 1
    while True:
        i += 1
        while nums[i] < middle:
            i += 1
        j -= 1
        while nums[j] > middle:
            j -= 1
        if i >= j:
            return j
        nums[i], nums[j] = nums[j], nums[i]

def quicking_sort(items, low, high):
    if low < high:
        split_index = partition(items, low, high)
        quicking_sort(items, low, split_index)
        quicking_sort(items, split_index + 1, high)

def quick_sort(nums):
    quicking_sort(nums, 0, len(nums) - 1)
```

3.4 Тестирование

В таблице 1 представлена заготовка данных для тестирования наших алгоритмов.

Массив	Ожидаемый результат
[1 2 3]	[1 2 3]
[3 2 1]	[1 2 3]
[1 2 −3]	[−3 1 2]
[1 0 −3]	[−3 0 1]
[0]	[0]

Таблица 1. Подготовленные тестовые данные.

3.5 Выводы

Реализовано 3 алгоритма, подготовлены тесты для оценки качества их работы.
Получены практические навыки реализации алгоритмов сортировки: поразрядного, вставками и быстрого.

4 Экспериментальная часть

Оценка качества работы алгоритмов. Экспериментальное сравнение работы различных алгоритмов (зависимость времени выполнения от размера массивов).

4.1 Примеры работы

На рисунке 5 представлены примеры работы программы на разных входных данных.

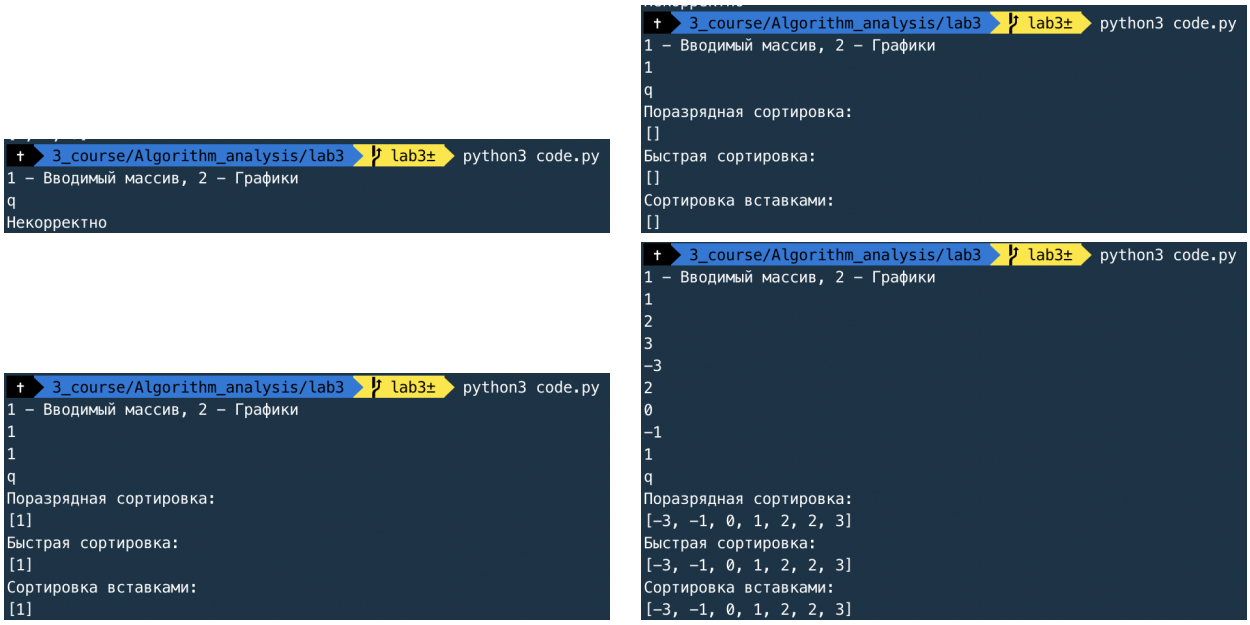

Рис. 5 - Примеры работы

4.2 Результаты тестирования

Проверяем нашу программу на тестах из таблицы 1. Полученные результаты представлены в таблице 2.

Массив	Поразрядная сортировка	Быстрая сортировка	Сортировка вставками
$[1\ 2\ 3]$	$[1\ 2\ 3]$	$[1\ 2\ 3]$	$[1\ 2\ 3]$
$[3\ 2\ 1]$	$[1\ 2\ 3]$	$[1\ 2\ 3]$	$[1\ 2\ 3]$
$[1\ 2\ -3]$	$[-3\ 1\ 2]$	$[-3\ 1\ 2]$	$[-3\ 1\ 2]$
$[1\ 0\ -3]$	$[-3\ 0\ 1]$	$[-3\ 0\ 1]$	$[-3\ 0\ 1]$
$[0]$	$[0]$	$[0]$	$[0]$

Таблица 2. Тестирование программы.

Тесты пройдены

4.3 Замеры времени

На графиках 6-11 представлено сравнение алгоритмов сортировки массивов.

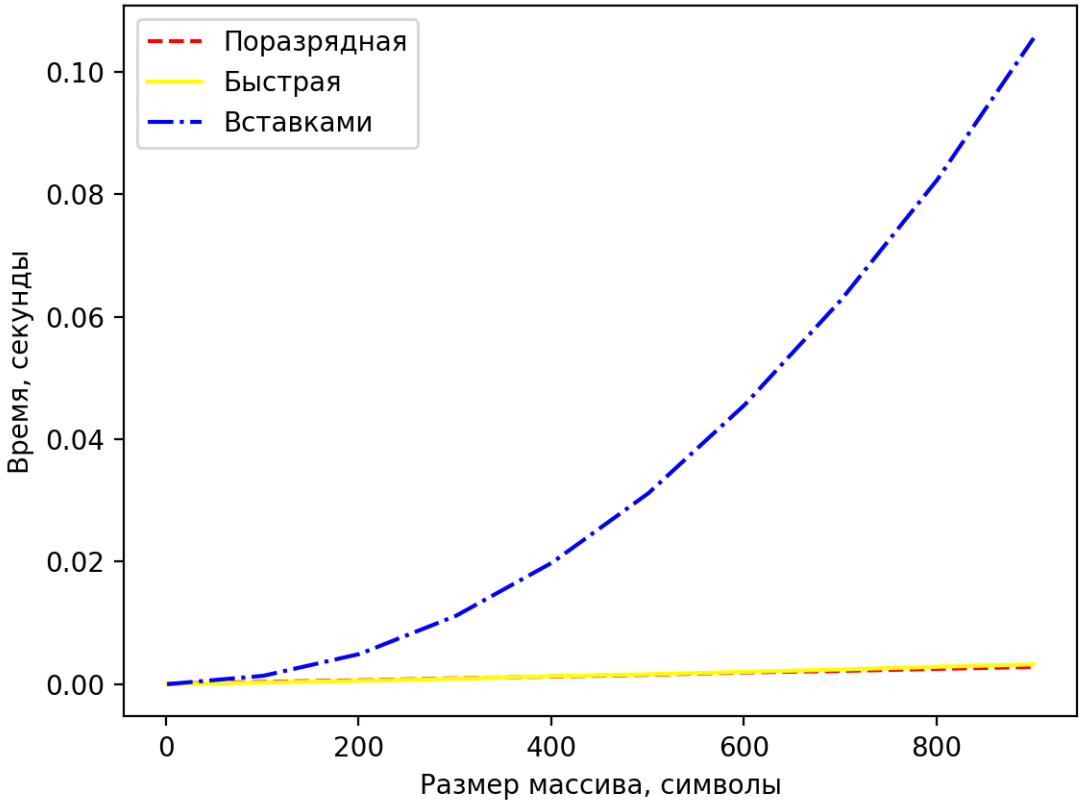

Рис. 6 - Сравнение реализации алгоритмов сортировок на произвольных данных

Как видно из графика 6, сортировка вставками должна рассматриваться отдельно от быстрой и поразрядной, так как ее трудоемкость очень высока.

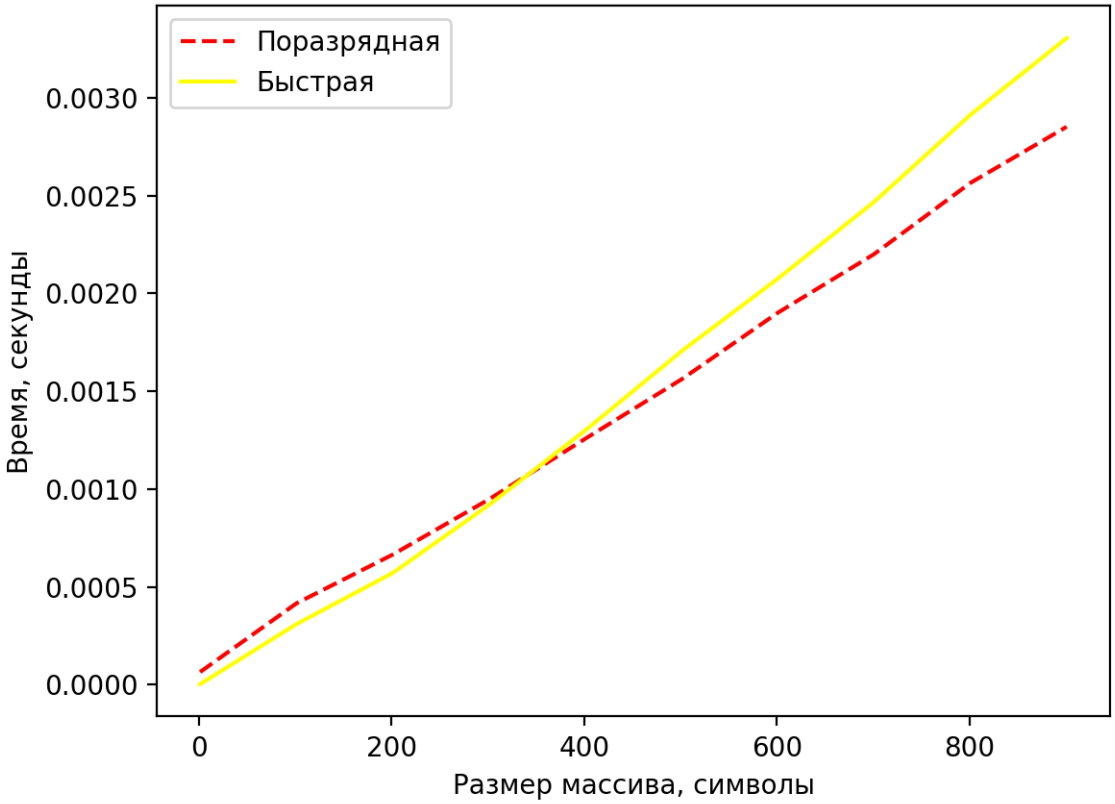

Рис. 7 - Сравнение реализации алгоритмов сортировок быстрой и поразрядной на произвольных данных (при средней разрядности числа до 4 значащих цифр)

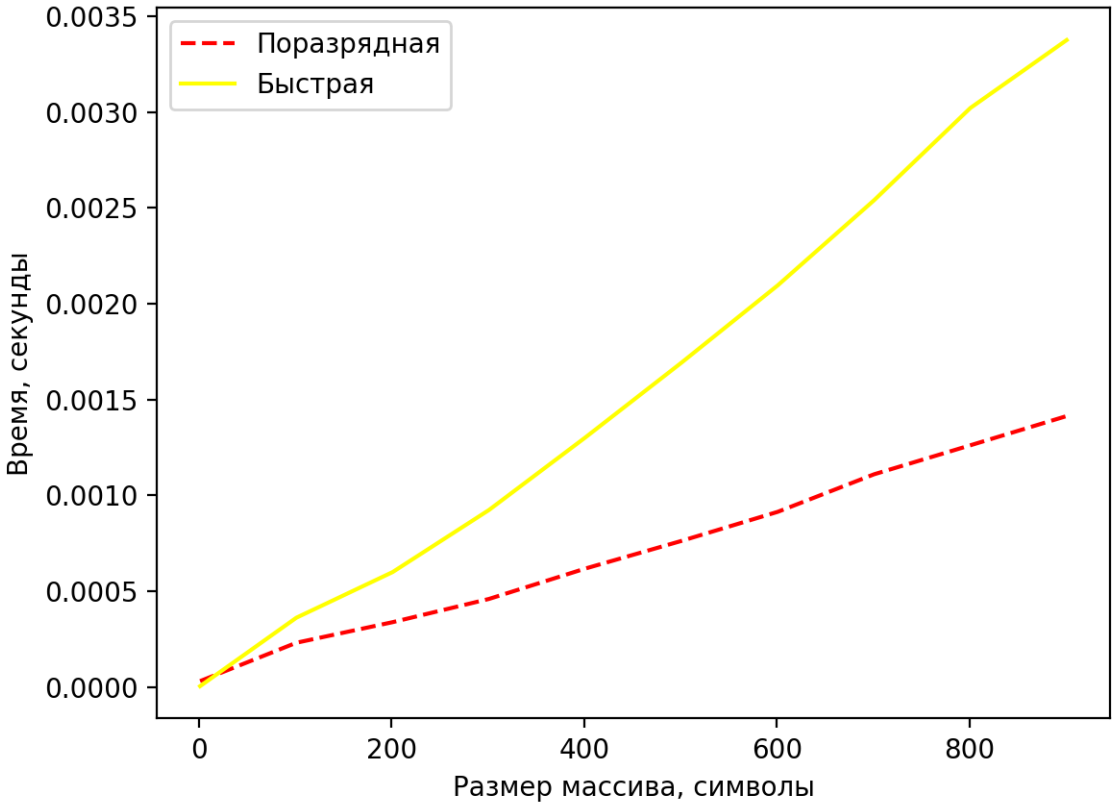

Рис. 8 - Сравнение реализации алгоритмов сортировок быстрой и поразрядной на произвольных данных (при маленькой разрядности числа 1 значащая цифра)

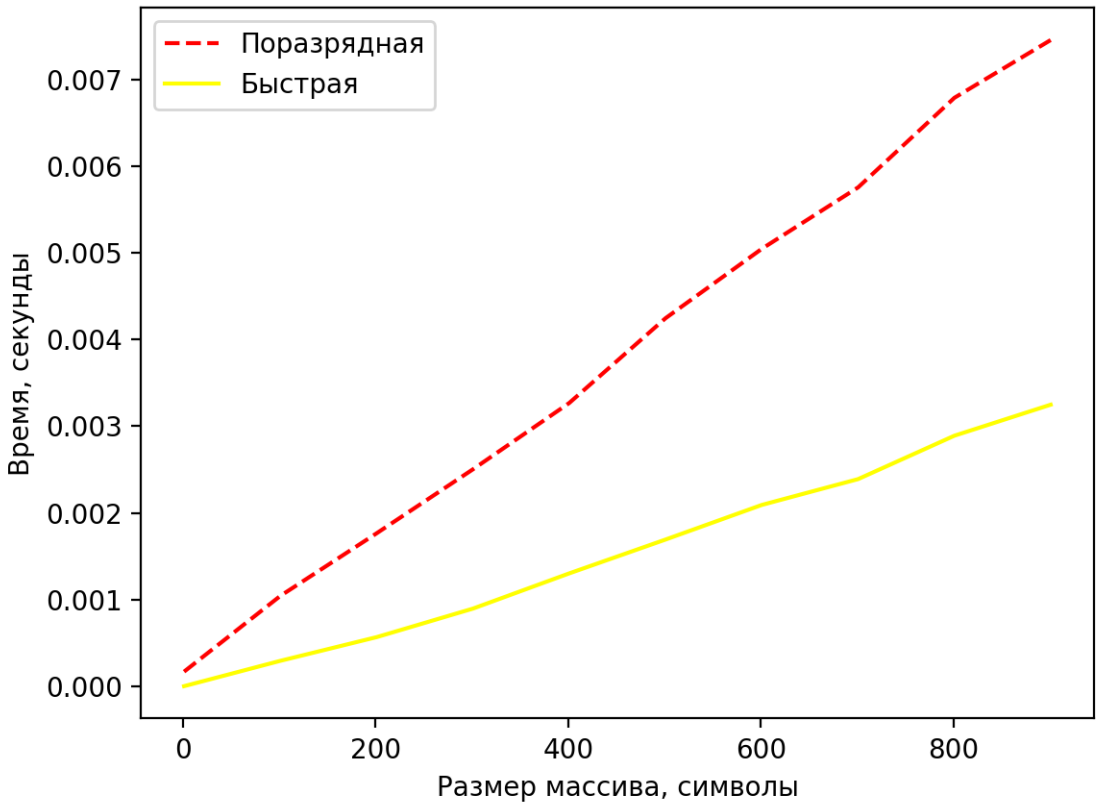

Рис. 9 - Сравнение реализации алгоритмов сортировок быстрой и поразрядной на произвольных данных (при большой разрядности числа до 10 значащих цифр)

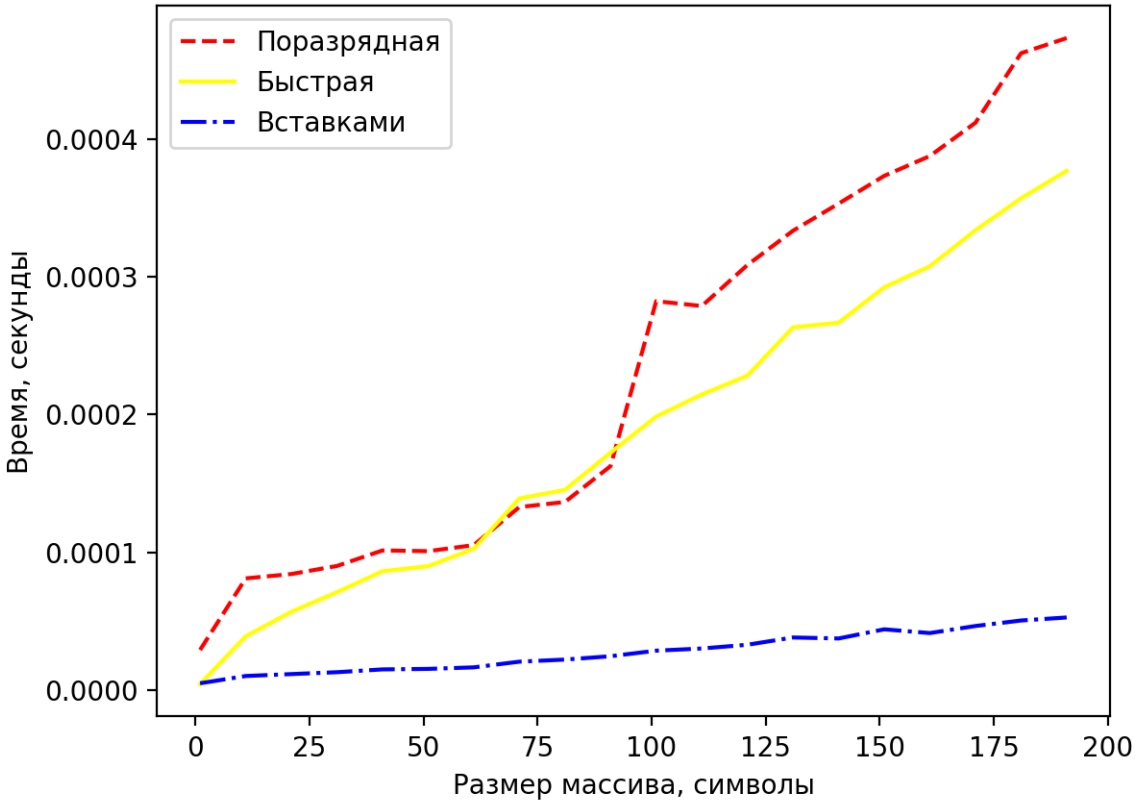

Рис. 10 - Сравнение реализации алгоритмов сортировок на упорядоченных данных

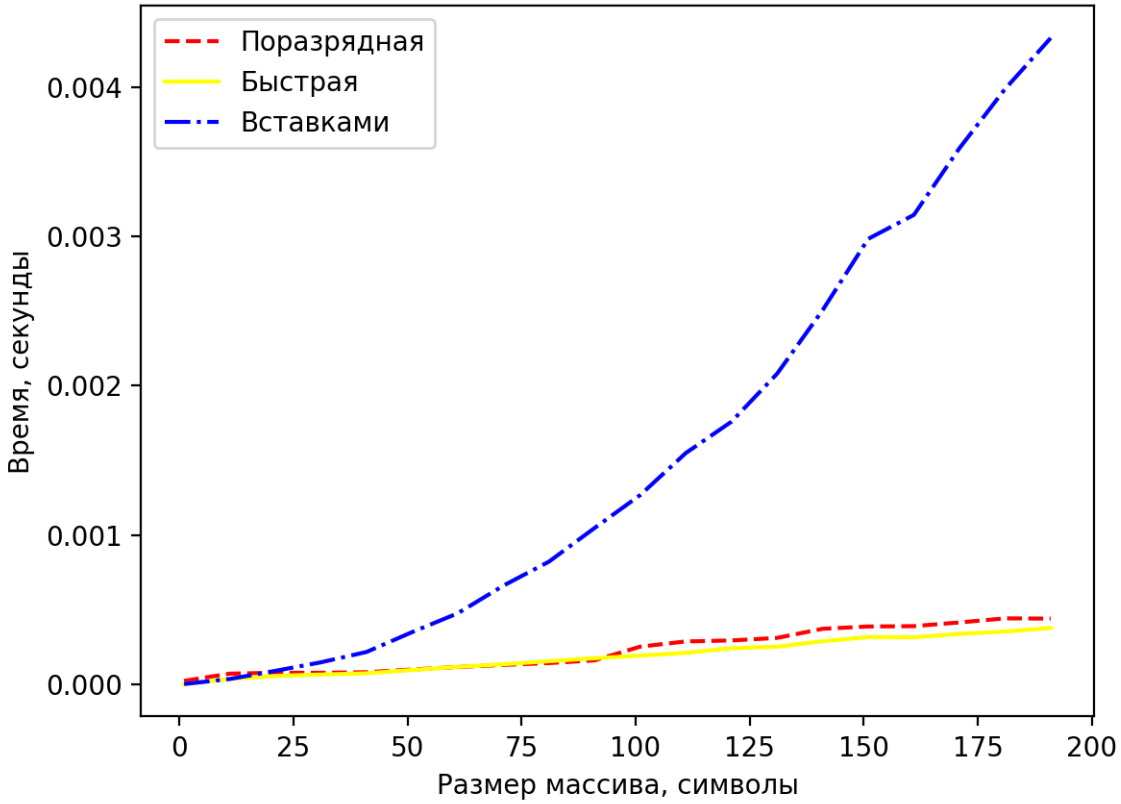

Рис. 11 - Сравнение реализации алгоритмов сортировок на обратно упорядоченных данных

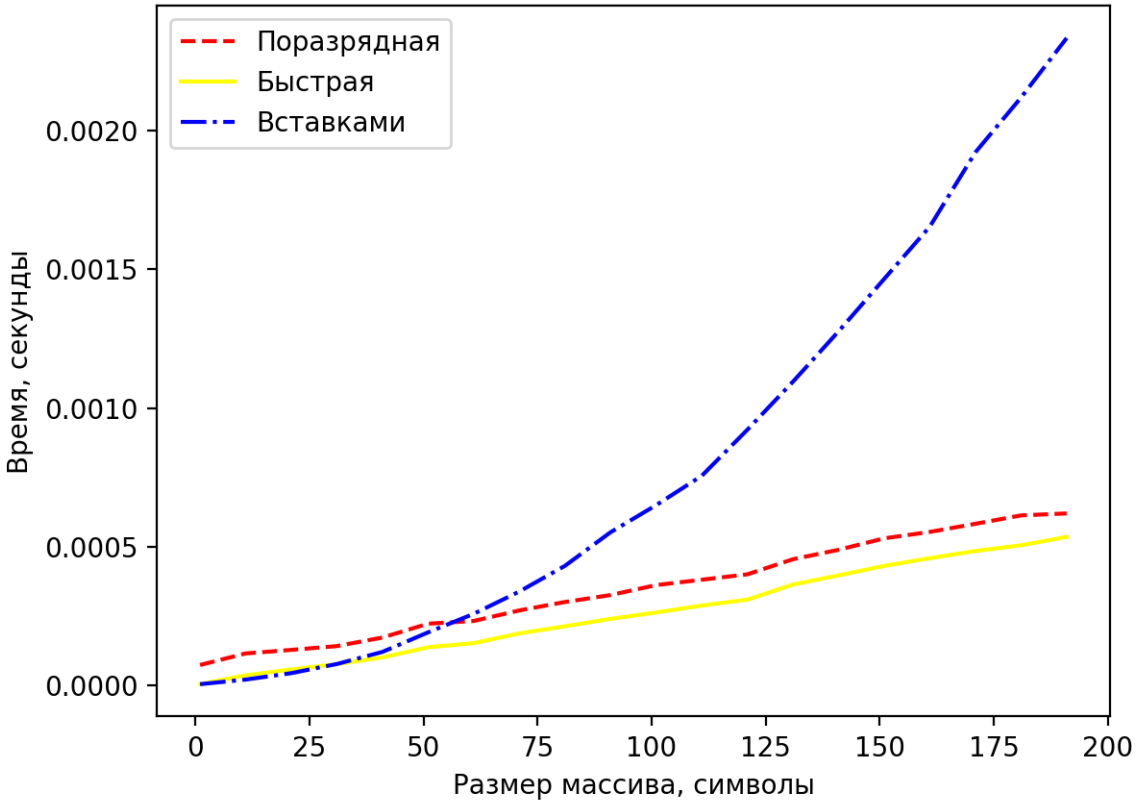

Рис. 12 - Сравнение реализации алгоритмов сортировок на произвольных данных при небольших размерах массивов

4.4 Выводы

Таким образом, по графикам подтвердилось предположение, о том, что самыми эффективным по скорости работы являются поразрядный алгоритм и быстрая сортировка. При размерностях выше $ln(n)$ (из оценки трудоемкости, где n - длина массива) быстрая сортировка оказывается эффективнее.

Также, оказалось, что при размерах массива меньше 50 эффективнее использовать сортировку вставками.

Заключение

В данной работе был проведен обзор 3 популярных алгоритмов сортировки.

1. Изучены алгоритмы сортировки массивов данных: быстрая сортировка, сортировка вставками и поразрядная сортировка.
2. Произведена оценка трудоемкости алгоритмов для лучших и худших и случаев и условий их наступления.
3. Получены практические навыки реализации алгоритмов сортировки на одном из языков программирования.
4. Проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (зависимость времени от длины массива)
5. Экспериментально подтверждены различия в трудоемкости алгоритмов с указанием лучших и худших случаев.

При сравнении данных алгоритмов пришли к следующим выводам:

1. Самым эффективным на средних наборах данных (числа разрядностью до 4 знаков) является алгоритм поразрядной сортировки. Однако при размерах массива меньших 50 рекомендуется использовать алгоритм сортировки вставками (или другие алгоритмы квадратичной сложности).
2. Быстрая сортировка является эффективной при больших размерах массивов (больших 50) и при разрядности чисел больших 4.

Список литературы

[1] Сортировка и фильтрация таблицы [Электронный ресурс]. - Режим доступа: https://support.office.com/ru-ru/article/Сортировка-и-фильтрация-таблицы-в-excel-с-помощью-средства-чтения-с-экрана-d6ae119c-5fab-4b7f-8869-6ef02ece71f5 (дата обращения: 19.10.2019)

[2] Кнут Д. Э., Козаченко Ю. В., Красиков И. В. Искусство программирования: Сортировка и поиск. Классический труд, Т. 3. М. : Вильямс, 2000. — 824 с.

[3] Sorting Benchmarks [Электронный ресурс]. - Режим доступа: http://sortbenchmark.org/ (дата обращения: 19.10.2019)

[4] Алгоритмы сортировки [Электронный ресурс]. - Режим доступа: http://kit.znu.edu.ua/Meth/SORTALG.pdf (дата обращения: 19.10.2019)

[5] Алгоритм сортировки [Электронный ресурс]. - Режим доступа: https://www.wikiwand.com/ru/Алгоритмсортировки (дата обращения: 19.10.2019)

[6] Алгоритмы сортировки в народных танцах [Электронный ресурс]. - Режим доступа: https://forany.xyz/a-370 (дата обращения: 19.10.2019)

[7] Сортировка данных [Электронный ресурс]. - Режим доступа: http://www.hpcc.unn.ru/mskurs/RUS/DOC/ppr10.pdf (дата обращения: 19.10.2019)