МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Н. Э. БАУМАНА

УДК № 004.822
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

Преподаватель

_____

«_____» _____ 2019 г.

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Трудоемкость алгоритмов сортировок
(промежуточный)

Студент _____ А. А. Куприй

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

Москва 2019

СПИСОК ИЛЛЮСТРАЦИЙ

2.1 Функциональная модель алгоритмов сортировки 11

2.2 Алгоритм поразрядной сортировки 12

2.3 Алгоритм сортировки вставками 13

2.4 Быстрая сортировка 14

4.1 Тест на неверный ввод 19

4.2 Тест на верных входных данных 19

4.3 Тест при вводе всего одного элемента 20

РЕФЕРАТ

Отчет содержит 28 стр., 7 рис..

СОДЕРЖАНИЕ

Реферат 3

Введение 5

1 Аналитический раздел 6

1.1 Описание задачи 6

1.2 Пути решения 7

1.3 Описание алгоритмов 7

1.3.1 Быстрая сортировка 7

1.3.2 Сортировка вставками 9

1.3.3 Поразрядная сортировка 9

1.4 Выводы 10

2 Конструкторский раздел 11

2.1 IDEF0 Модель 11

2.2 Разработка алгоритмов 12

2.2.1 Поразрядная сортировка 12

2.2.2 Сортировка вставками 13

2.2.3 Быстрая сортировка 14

2.3 Сравнительный анализ алгоритмов 14

2.3.1 Вывод 15

3 Технологический раздел 16

3.1 Требования к программному обеспечению 16

3.2 Средства реализации 16

3.3 Листинги кода 16

3.4 Описание тестирования 18

3.5 Вывод 18

4 Исследовательский раздел 19

4.1 Примеры работы 19
4.2 Эксперименты по замеру времени 21
4.3 Выводы 27
Заключение 28

ВВЕДЕНИЕ

В данной работе требуется провести обзор 3 популярных алгоритмов сортировки.

а) Изучить алгоритмы сортировки массивов данных: быстрая сортировка, сортировка вставками и поразрядная сортировка.

б) Оценить трудоемкости алгоритмов, произвести теоретическую оценка для лучших и худших и случаев и условий их наступления.

в) Получить практические навыки реализации алгоритмов сортировки на одном из языков программирования.

г) Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (зависимость времени от длины массива)

д) Экспериментально подтвердить различия в трудоемкости алгоритмов с указанием лучших и худших случаев.

1 Аналитический раздел

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.

а) Число операций сортировки (сравнения и перемещения) – параметры, характеризующие трудоемкость алгоритма.

б) 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:

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

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

б) Трудоемкость улсовного перехода примем за единицу, при этом условие вычисляется по пункту 1.

в) 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, которые были придуманы и используются для разных задач.

1.3 Описание алгоритмов

В данном разделе будут описаны выбранные алгоритмы.

1.3.1 Быстрая сортировка

Краткое описание алгоритма:

а) выбирается элемент, называемый опорным;

б) остальные элементы сравниваются с опорным, на основании сравнения меньшие опорного перемещаются левее него, а большие или равные - правее;

в) рекурсивно упорядочиваются подмассивы, лежащие слева и справа от опорного элемента.

Для этого алгоритма самый лучший случай — если в каждой итерации каждый из подмассивов делился бы на два равных по величине массива. В результате количество сравнений, производимых быстрой сортировкой, было бы равно значению рекурсивного выражения $CN = 2CN/2+N$, что в явном выражении дает примерно $N * log(N)$ сравнений. Это дало бы наименьшее время сортировки. Худшим случаем будет такой, при котором на каждом этапе массив будет разделяться на вырожденный подмассив из одного опорного элемента и на подмассив из всех остальных элементов. Расчет трудоемкости: При анализе сложности циклических алгоритмов рассчитывается трудоемкость итераций и их количество в наихудшем и наилучшем случаях. Однако не получится применить такой подход к рекурсивной функции, так как в результате будет получено рекуррентное соотношение. Рекуррентные отношения не позволяют оценить сложность — мы не можем их просто так сравнивать, а значит, и сравнивать эффективность соответствующих алгоритмов. С помощью общего метода решений рекуррентных соотношений оценим трудоемкость алгоритма. Исходные данные разделяются на две части, обе из которых обрабатываются: $a = 2$ (подзадачи), $b = 2$ (части), $n\ logba = n$ - скорость роста функции разделения задачи и компоновки результата. На соединение результатов будет затрачено $O(n)$, поэтому $fn = n$. Для расчета трудоемкости лучшего случая используем второй случай основной теоремы о рекуррентных соотношениях: $T\ quicksort\ n = O(n\ logba \cdot logn) = O(n \cdot logn)$. В худшем случае каждое разделение даёт два подмассива размерами 1 и $n - 1$, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо с наименьшим, либо наибольшим индексом из всех обрабатываемых. При простейшем выборе опорного элемента — первого или последнего в массиве, — такой эффект даст уже отсортированный (в прямом или обратном порядке) массив, для среднего или любого другого фиксированного элемента «массив худшего случая» также может быть специально подобран. В этом случае общее время работы составит $\sum$[U+FE01]ni=0(n −

$i) \approx O(n*2)$ операций, то есть сортировка будет выполняться за квадратичное время.

1.3.2 Сортировка вставками

Краткое описание алгоритма:

а) выбирается один из элементов входных данных;

б) выбранный элемент вставляется на нужную позицию в уже отсортированной последовательности;

в) п 1,2 выполняются, пока набор входных данных не будет исчерпан.

Сортировка вставками не использует обмены. Сложность алгоритма измеряется числом сравнений и равна $O(n\ 2\ )$. Наилучший случай – когда исходная последовательность уже отсортирована. Тогда на i-ом проходе вставка производится в точке $A[i]$, а общее число сравнений равно n-1. Наихудший случай возникает, когда список отсортирован по убыванию. Тогда каждая вставка происходит в точке $A[0]$ и требует i сравнений; общее число сравнений равно n(n-1)/2. Расчет трудоемкости: $fbest = 2 + (n - 1)(7 + 0 \cdot 9 + 3) = 2 + 10(n - 1)$ $fworst = 2 + (n - 1)(7 + (n - 1) \cdot 9 + 3) = 9(n - 1)2 + 10(n - 1) + 2 = 9n\ 2 - 8n + 1$ где n - длина сортируемой последовательности.

1.3.3 Поразрядная сортировка

Алгоритм поразрядной сортировки не использует сравнений самих сортируемых элементов. Ключ, по которому происходит сортировка, разделяется на части - разряды ключа. Например, число можно разделить по цифрам, слово - по буквам. Перед началом непосредственно сортировки необходимо вычислить два параметра: k - количество разрядов в самом длинном ключе, m - разрядность данных: количество возможных значений разряда ключа. На каждом шаге алгоритма числа сортируются таким образом, чтобы они были упорядочены по последним i разрядам/битам, то есть на каждом шаге достаточно сортировать элементы по новому разряду/биту. Краткое описание алгоритма:

а) создаются пустые списки, количество которых равно числу m;

б) исходные числа распределяются по этим спискам в зависимости от величины младшего разряда (по возрастанию);

в) числа собираются в единый список (который замещает теперь исходный) в той последовательности, в которой они находятся после распределения по спискам;

г) пункты 2 и 3 повторяются для всех более старших разрядов поочередно (и так до k). Расчет трудоемкости: $f = 4 + (2 + 4k) + 4 + 2 + m \cdot (4 + 5n + 10) + 2 + 4k = 12 + 4k + m(14 + 5n) + 2 + 4k = 5kn + 8m + 14k + 14$, где n - длина сортируемой последовательности. Сложность поразрядной сортировки не зависит от того, как упорядочены входные данные, так как не основана на сравнении непосредственно самих элементов последовательности.

1.4 Выводы

Для теоретической оценки вышеупомянотых алгоритмов реализуем их и проверим всё эксперементально.

2 Конструкторский раздел

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

2.1 IDEF0 Модель

На рисунке 2.1 приведена функциональная модель множения матриц в нотации IDEF0.

Рисунок 2.1 — Функциональная модель алгоритмов сортировки

2.2 Разработка алгоритмов

В данном разделе рассматриваются необходимые алгоритмы с помощью блок-схем.

2.2.1 Поразрядная сортировка

На рисунке 2.2 представлен алгоритм поразрядной сортировки.

Рисунок 2.2 — Алгоритм поразрядной сортировки

2.2.2 Сортировка вставками

На рисунке 2.3 представлен алгоритм сортировки вставками.

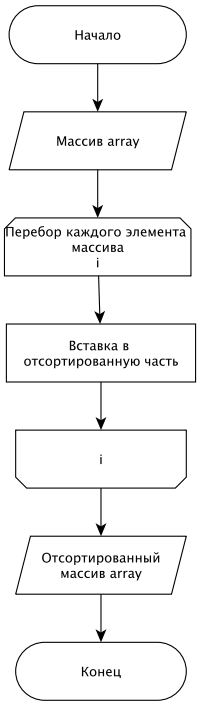

Рисунок 2.3 — Алгоритм сортировки вставками

2.2.3 Быстрая сортировка

На рисунке 2.4 представлен алгоритм быстрой сортировки.

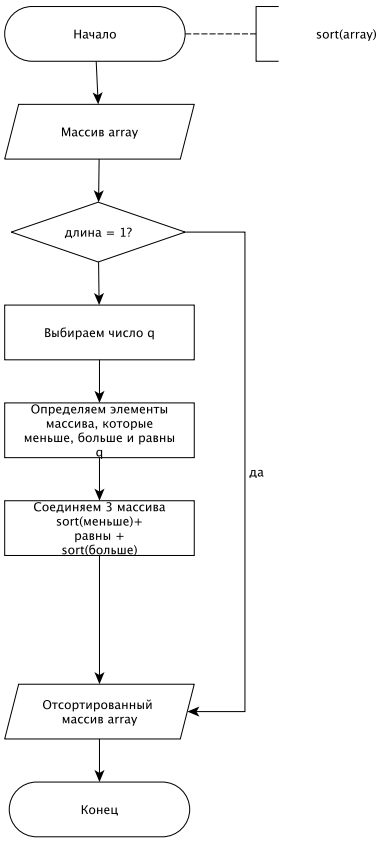

Рисунок 2.4 — Быстрая сортировка

2.3 Сравнительный анализ алгоритмов

1) Поразрядная сортировка

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

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

$$f = O(nk), n - \text{размер массива}, k - \text{порядок числа}$$

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 ма

$$f = O(n^2), n - \text{размер массива, Средний случай}$$

3) Быстрая сортировка

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

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

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

2.3.1 Вывод

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.

3 Технологический раздел

В данном разделе приводятся описания требований к программному обеспечению, средства реализации, листинги кода и описания тестирования.

3.1 Требования к программному обеспечению

Требования к вводу:

— на вход подается массив;

Требования к выводу:

— на выход подаются массивы, отсортированные разными алгоритмами;

3.2 Средства реализации

В качестве языка программирования мною был выбран python, так как данный язык программирования позволяет максимально лаконично и демонстративно реализовать необходимые алгоритмы.

3.3 Листинги кода

В листингах 3.1 - 3.3 приведена реализация описанных алгоритмов.

Листинг 3.1 — Поразрядная сортировка

```
1       n = pow(digit, n)
2           i = 1
3           while (i < n):
4           sort = [[] for k in range(digit)]
5
6           for x in array:
7               sort[get_digit(x, i)].append(x)
8
9           count = len(array)
10          array = [0] * count
11          u = 0
12          w = 0
13          for k in range(digit):
14              for j in range(len(sort[k])):
```

```
15                  if (sort[k][j] < 0):
16                      array[w] = sort[k][j]
17                      w += 1
18                  else:
19                      array[u + neg] = sort[k][j]
20                      u += 1
21          i *= 10
```

Листинг 3.2 — Сортировка вставками

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

Листинг 3.3 — Быстрая сортировка

```
1  def partition(nums, low, high):
2      middle = nums[(low + high) // 2]
3      i = low - 1
4      j = high + 1
5      while True:
6          i += 1
7          while nums[i] < middle:
8              i += 1
9          j -= 1
10         while nums[j] > middle:
11             j -= 1
12         if i >= j:
13             return j
14         nums[i], nums[j] = nums[j], nums[i]
15
16 def quicking_sort(items, low, high):
17     if low < high:
18         split_index = partition(items, low, high)
19         quicking_sort(items, low, split_index)
20         quicking_sort(items, split_index + 1, high)
21
22 def quick_sort(nums):
23     quicking_sort(nums, 0, len(nums) - 1)
```

3.4 Описание тестирования

Для тестирования программы были подготовлены данные, представленные в таблице 1.

Массив	**Ожидаемый результат**
$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$
$\begin{bmatrix} 3 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & -3 \end{bmatrix}$	$\begin{bmatrix} -3 & 1 & 2 \end{bmatrix}$
$\begin{bmatrix} 1 & 0 & -3 \end{bmatrix}$	$\begin{bmatrix} -3 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} 0 \end{bmatrix}$	$\begin{bmatrix} 0 \end{bmatrix}$

Таблица 1. Подготовленные тестовые данные.

Все тесты были успешно пройдены.

3.5 Вывод

Были сформулированы требования к ПО, выбраны средства реализации и подготовлены тестовые данные.

4 Исследовательский раздел

В данном разделе привидены и проанализированы примеры работы реализованной программы.

4.1 Примеры работы

На рисунках 4.1, 4.2, 4.3 показана работа программы с различными входными данными.

Рисунок 4.1 — Тест на неверный ввод

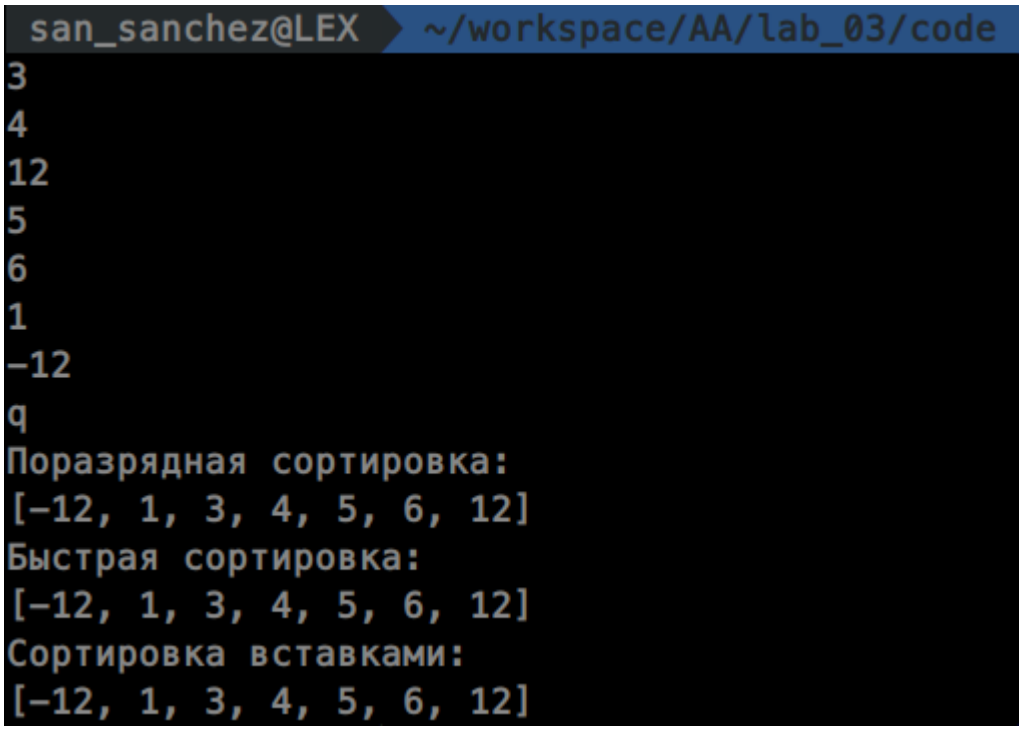

Рисунок 4.2 — Тест на верных входных данных

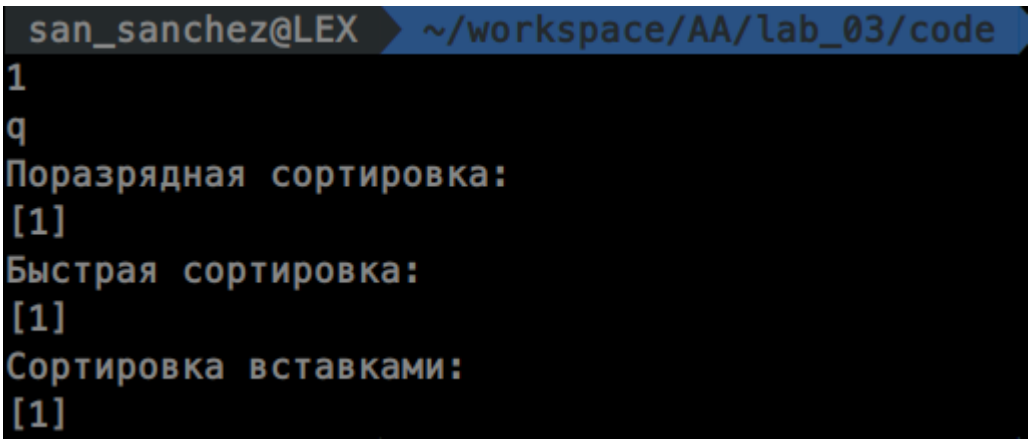

Рисунок 4.3 — Тест при вводе всего одного элемента

4.2 Эксперименты по замеру времени

На графиках 6-7 представлено сравнение алгоритмов умножения матриц.

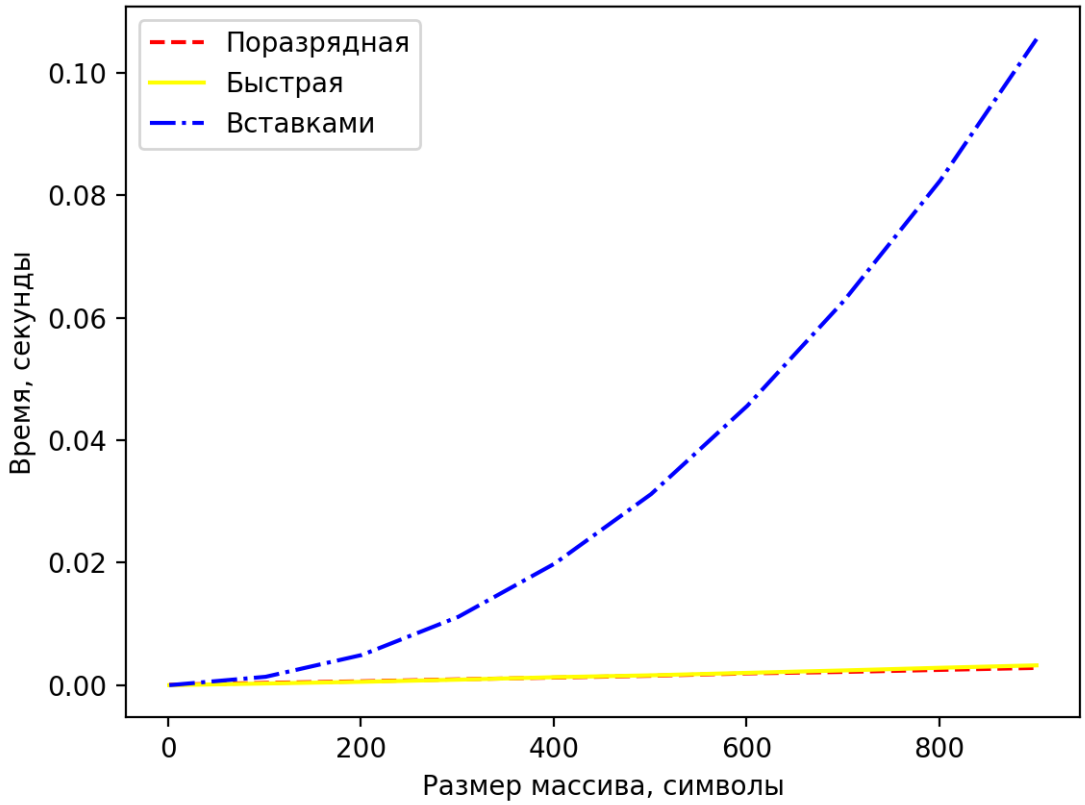

Рис. 4.4 - Сравнение реализации алгоритмов сортировок на произвольных данных.

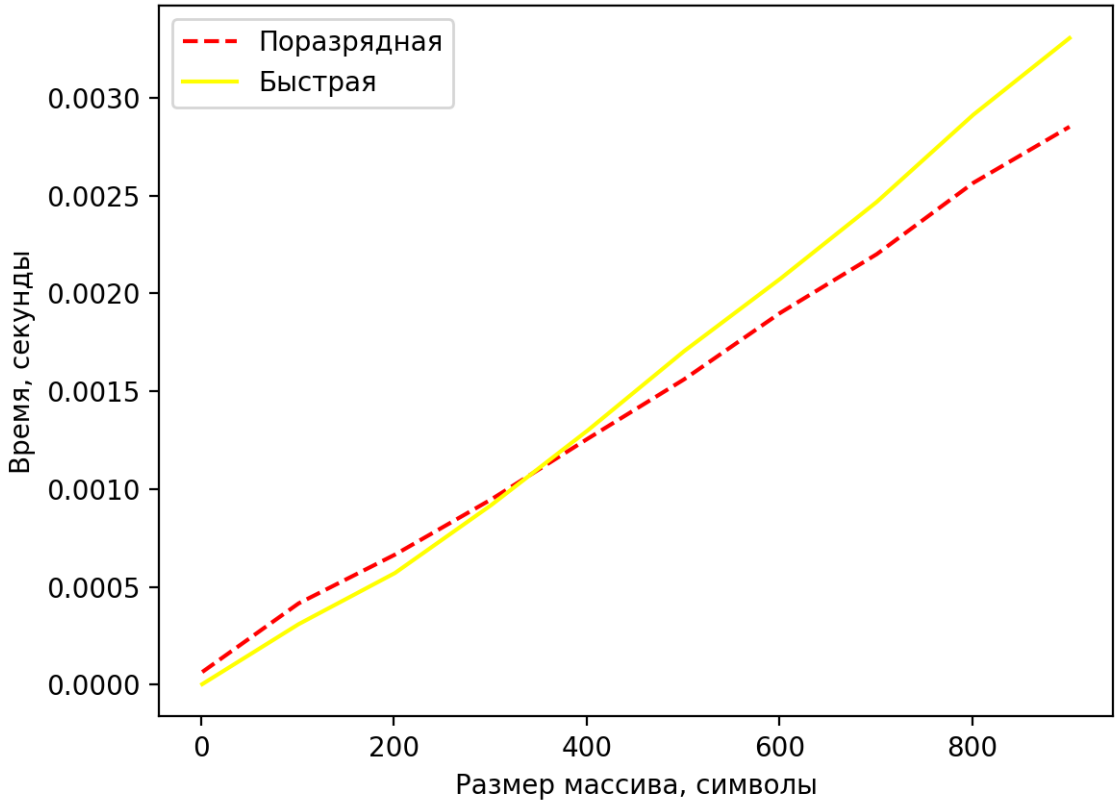

Рис. 4.5 - Сравнение реализации алгоритмов сортировок быстрой и поразрядной на произвольных данных(при средней разрядности числа до 4 значащих цифр).

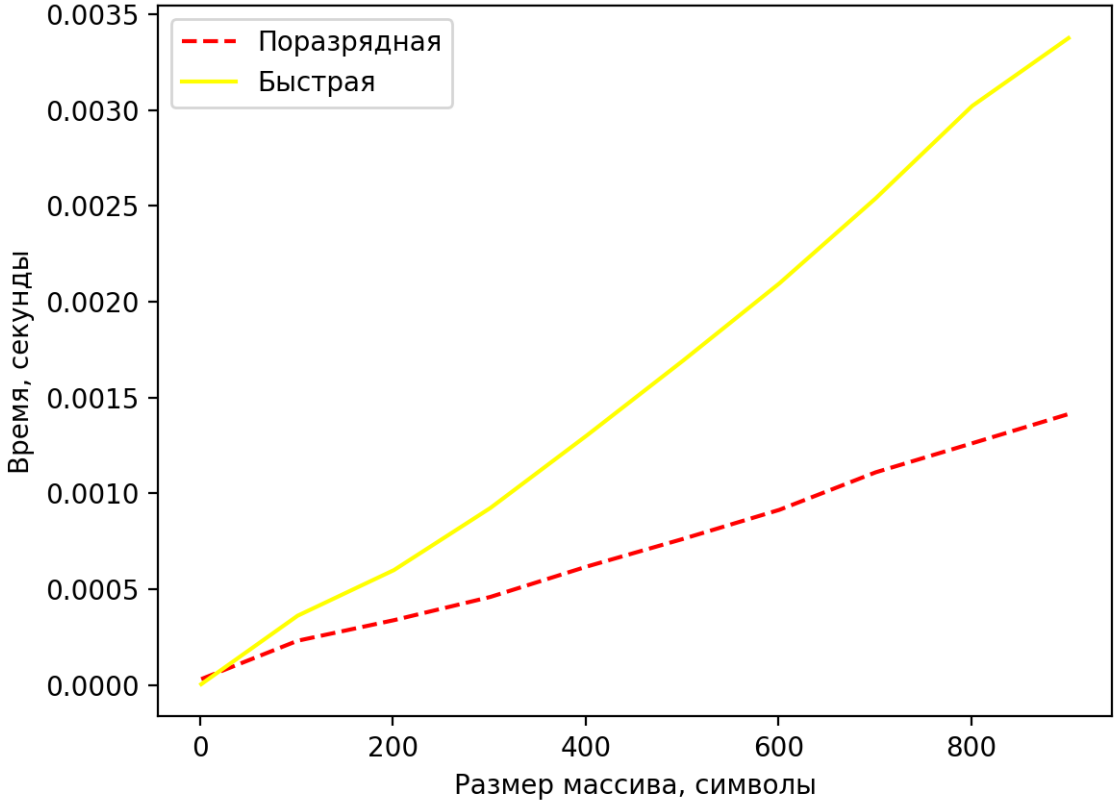

Рис. 4.6 - Сравнение реализации алгоритмов сортировок быстрой и поразрядной на произвольных данных (при маленькой разрядности числа 1 значащая цифра).

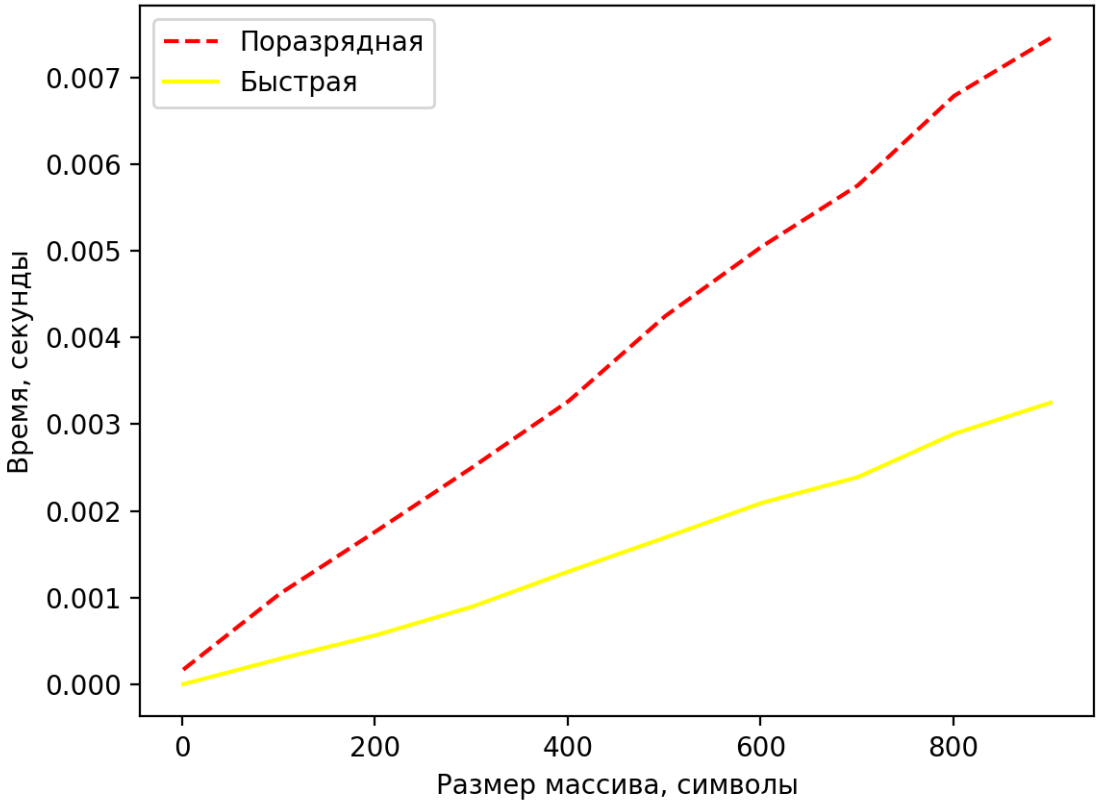

Рис. 4.7 - Сравнение реализации алгоритмов сортировок быстрой и поразрядной на произвольных данных (при большой разрядности числа до 10 значащих цифр).

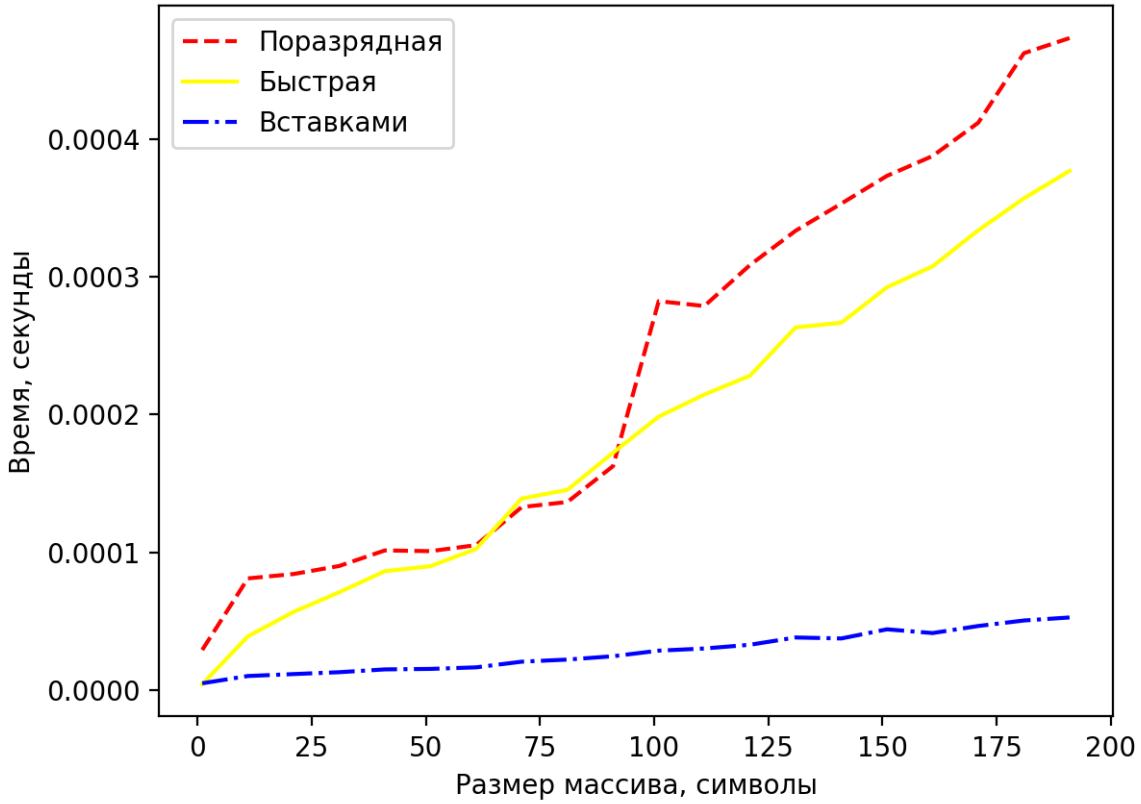

Рис. 4.8 - Сравнение реализации алгоритмов сортировок на упорядоченных данных.

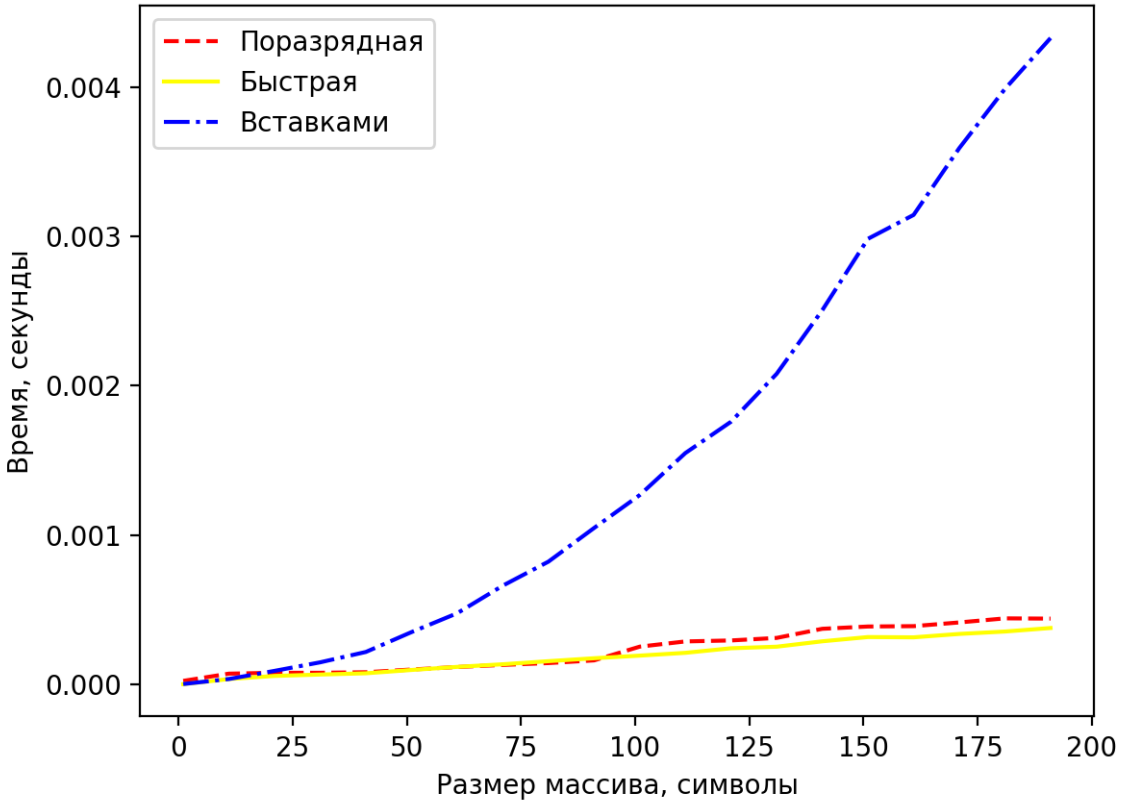

Рис. 4.9 - Сравнение реализации алгоритмов сортировок на обратно упорядоченных данных.

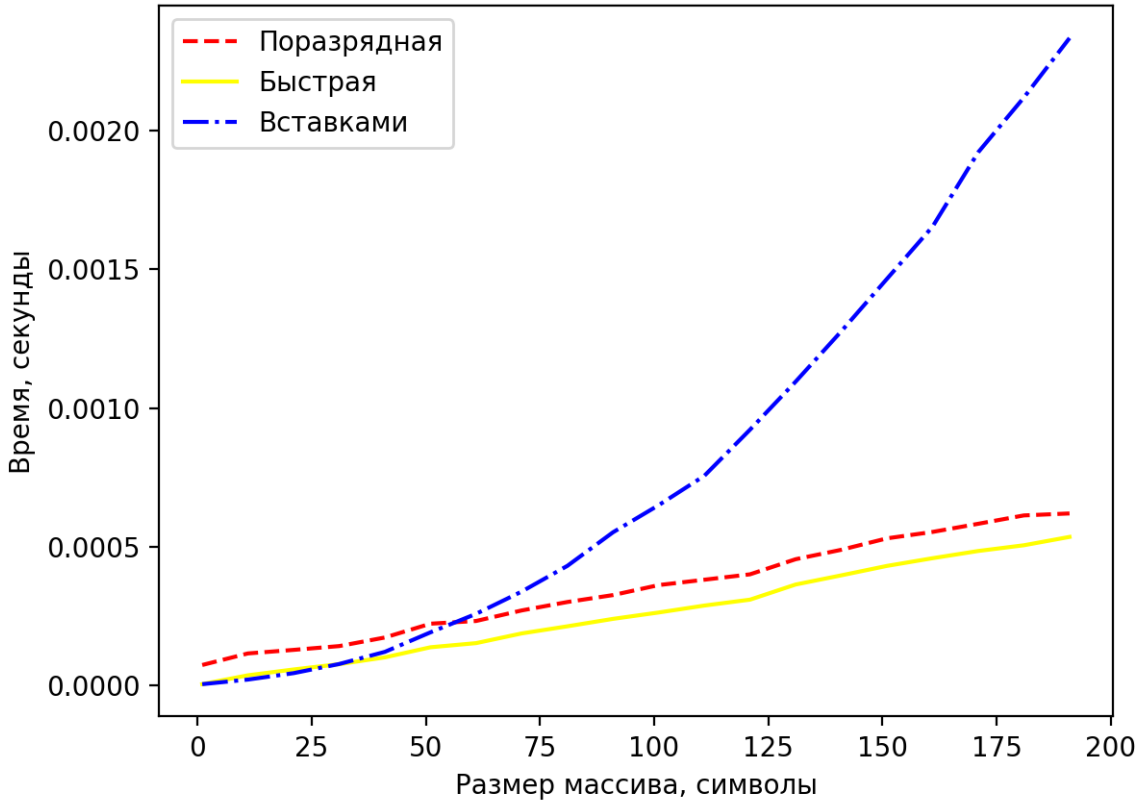

Рис. 4.10 - Сравнение реализации алгоритмов сортировок на произвольных данных при небольших размерах массивов.

4.3 Выводы

Как видно из графиков и из проведенного анализа трудоемкости, поразрядная сортировка имеет линейную зависимость от длины массива и разрядности элементов; быстрая - квадаратичную в худшем случае и $n \cdot log(n)$ - в остальных; поразрядная - линейную в лучшем и квадратичную в среднем и худшем случаях.

ЗАКЛЮЧЕНИЕ

В ходе даннои работы были изучены алгоритмы поразрядной и быстрой сортировок, сортировки вставками; были получены практические навыки реализации указанных алгоритмов. Была проведена оценка сложности теоретически с указанием лучшего и худшего случаев (если есть) и условий их наступления. Был проведен сравнительный анализ перечисленных алгоритмов сортировки по затрачиваемым ресурсам времени и получено экспериментальное подтверждение различий во временн´ой эффективности выбранных алгоритмов сортировки при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах массивов. В результате были получены следующие выводы:

1) Сортировка вставками работает медленнее остальных исследуемых алгоритмов при во всех рассмотренных случаях на длинных массивах ($\approx$ от 100 элементов). Но этот алгоритм эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим; он также эффективен на наборах данных, которые уже частично отсортированы; это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);

2) Основным достоинством поразрядной сортировки является скорость, однако она требует использования дополнительной памяти и имеет узкую специализацию;

3) Алгоритм быстрой сортировки является одним из самых быстрых универсальных алгоритмов сортировки массивов. Однако в худшем случае глубина рекурсии при выполнении алгоритма достигнет n, что будет означать n-кратное сохранение адреса возврата и локальных переменных процедуры разделения массивов. Для больших значений n худший случай может привести к исчерпанию памяти (переполнению стека) во время работы программы.