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 по курсу "Анализ алгоритмов"

Тема Трудоёмкость сортировок

Студент Калашков П. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2022 г.

Содержание

Введение

Сортировка – перегруппировка некой последовательности или кортежа в определённом порядке. Это одна из главных процедур обработки структурированных данных. Расположение элементов в определённом порядке позволяет более эффективно проводить работу с последовательностью данных, в частности при поиске некоторых данных.

Существует множество алгоритмов сортировки, но любой алгоритм сортировки имеет:

- сравнение, которое определяет, как упорядочена пара элементов;
- перестановка для смены элементов местами;
- алгоритм сортировки, использующий сравнение и перестановки.

Что касается самого поиска, то при работе с отсортированным набором данных время, которое нужно на нахождение элемента, пропорционально логарифму количства элементов. Последовательность, данные которой расположены в хаотичном порядке, занимает время, которое пропорционально количеству элементов, что куда больше логарифма.

Цель работы: изучение и исследование трудоёмкости алгоритмов сортировки.

Задачи работы:

1. Изучить и реализовать алгоритмы сортировки: блинная, поразрядная, бинарным деревом.
2. Провести тестирование по времени для выбранных сортировок.
3. Провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов.
4. Провести сравнительный анализ реализаций алгоритмов по затраченному процессорному времени и памяти.

5. Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут рассмотрены алгоритмы сортировок – блинная, поразрядная, бинарным деревом.

1.1 Блинная сортировка

Сортировка вставками [1] (англ. *pancake sort*)– алгоритм сортировки. Единственная операция, допустимая в алгоритме – переворот элементов последовательности до какого-либо индекса. В отличие от традиционных алгоритмов, в которых минимизируют количество сравнений, в блинной сортировке требуется сделать как можно меньше переворотов. Процесс можно визуально представить как стопку блинов, которую тасуют путём взятия нескольких блинов сверху и их переворачивания.

В данной работе будет рассмотрен вариант блинной сортировки, основанный на сортировке выбором. Алгоритм состоит из нескольких шагов:

1. Найти номер максимального числа в неотсортированной части массива.
2. Произвести “переворот“ неотсортированной части так, чтобы максимум встал на своё место.
3. Аналогично сортировать остаточную часть массива, исключив из рассмотрения уже отсортированные элементы.

1.2 Поразрядная сортировка

Поразрядная сортировка [2] (англ. *radix sort*) – сортировка, исходно предназначенная для сортировки целых чисел, записанных цифрами. Поскольку в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на “разряды“, содержащие сравнимые значения. Таким образом можно сортировать не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и даже произвольные значения в памяти, представленные в виде набора байт.

Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

Для поразрядной сортировки целых чисел в рамках одного разряда удобно использовать сортировку подсчётом – рассмотрим её ниже.

1.2.1 Сортировка подсчётом

Сортировка подсчётом [3] (англ. *counting sort*) – алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов. Данный алгоритм состоит из двух основных этапов:

1. Создать вспомогательный массив, состоящий из нулей, затем последовательно прочитать элементы входного массива, для каждого из них увеличить соответствующее значение вспомогательного массива на единицу.

2. Пройтись по вспомогательному массиву и для i-го значения X записать в результирующий массив i-ое значение X раз

1.3 Сортировка бинарным деревом

Сортировка бинарным деревом [4] (англ. *binary search tree sort*) – универсальный алгоритм сортировки, заключающийся в построении двоичного дерева поиска по элементам массива (списка), с последующей сборкой результирующего массива путём обхода узлов построенного дерева в необходимом порядке следования ключей.

Бинарное дерево поиска [4] (англ. *binary search tree*) – двоичное дерево, для которого выполняются следующие дополнительные условия:

1. Оба поддерева – левое и правое – являются двоичными деревьями поиска.

2. У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X;

3. У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

Преобразование бинарного дерева поиска в результирующий массив с отсортированными данными следует делать при помощи обратного обхода бинарного дерева – обхода, при котором сначала обходится левое поддерево данной вершины, затем правое, затем данная вершина.

Вывод

В данной работе необходимо реализовать алгоритмы сортировки, описанные в данном разделе, а также провести их теоритическую оценку и проверить ее экспериментально.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов сортировок (вставками, перемешиванием и гномья), а также найдена их трудоемкость

2.1 Требования к ПО

Ряд требований к программе:

- на вход подается массив целых чисел в диапазоне от 0 до 10000;
- возвращается отсортированный по возрастанию массив, который был задан в предыдущем пункте.

2.2 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки –

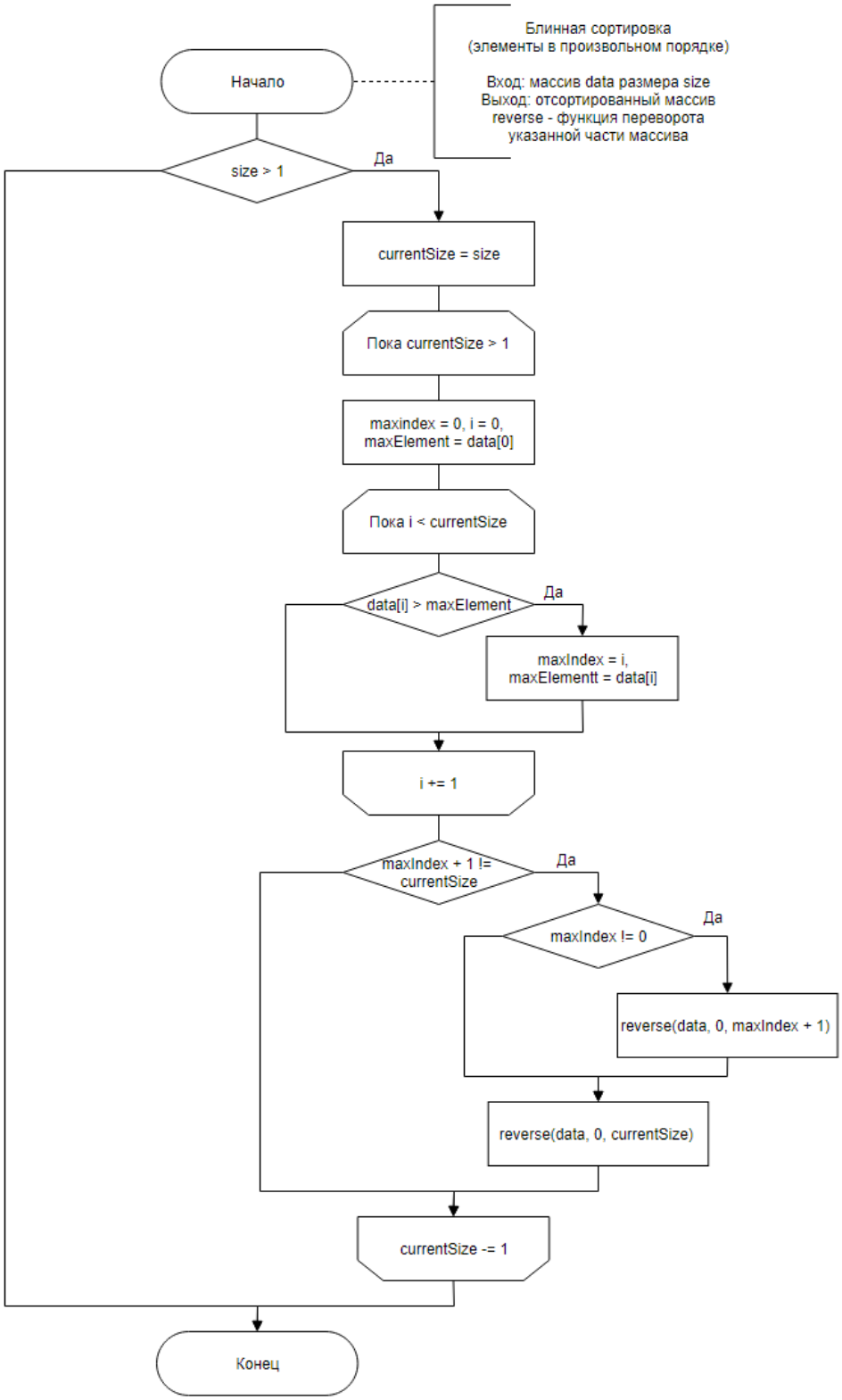

Рисунок 2.1 – Схема алгоритма блинной сортировки

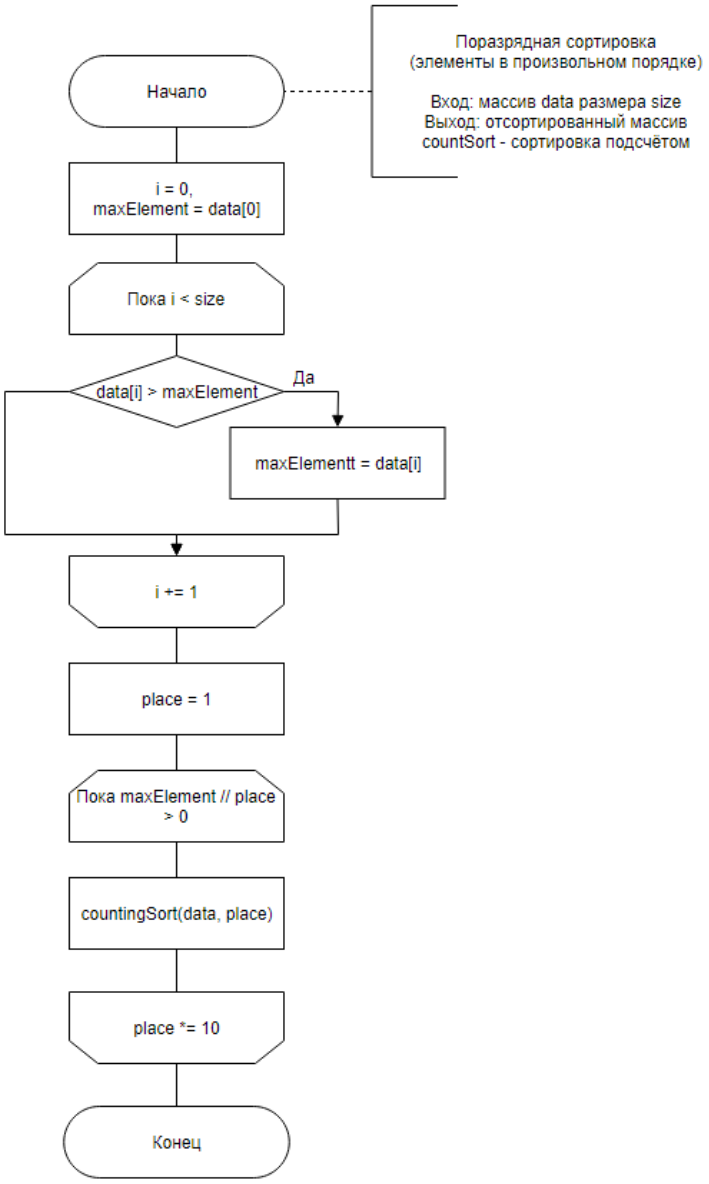

Рисунок 2.2 – Схема алгоритма поразрядной сортировки

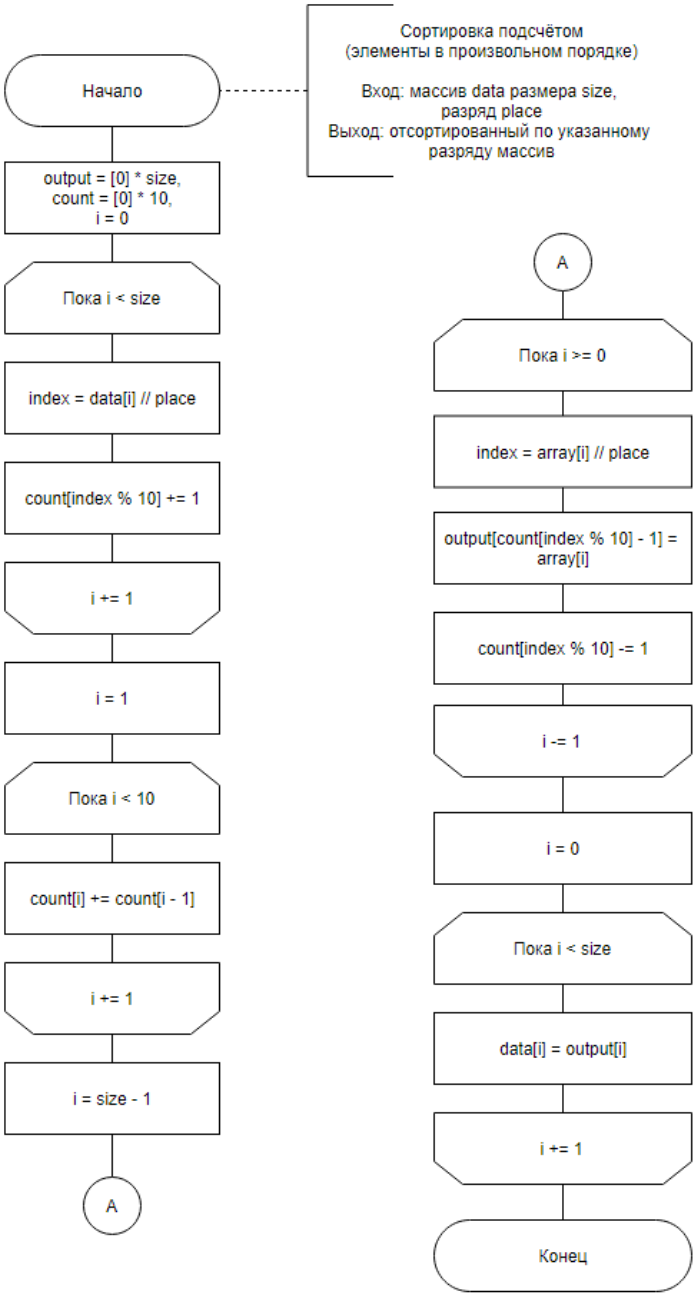

Рисунок 2.3 – Схема алгоритма сортировки подсчётом

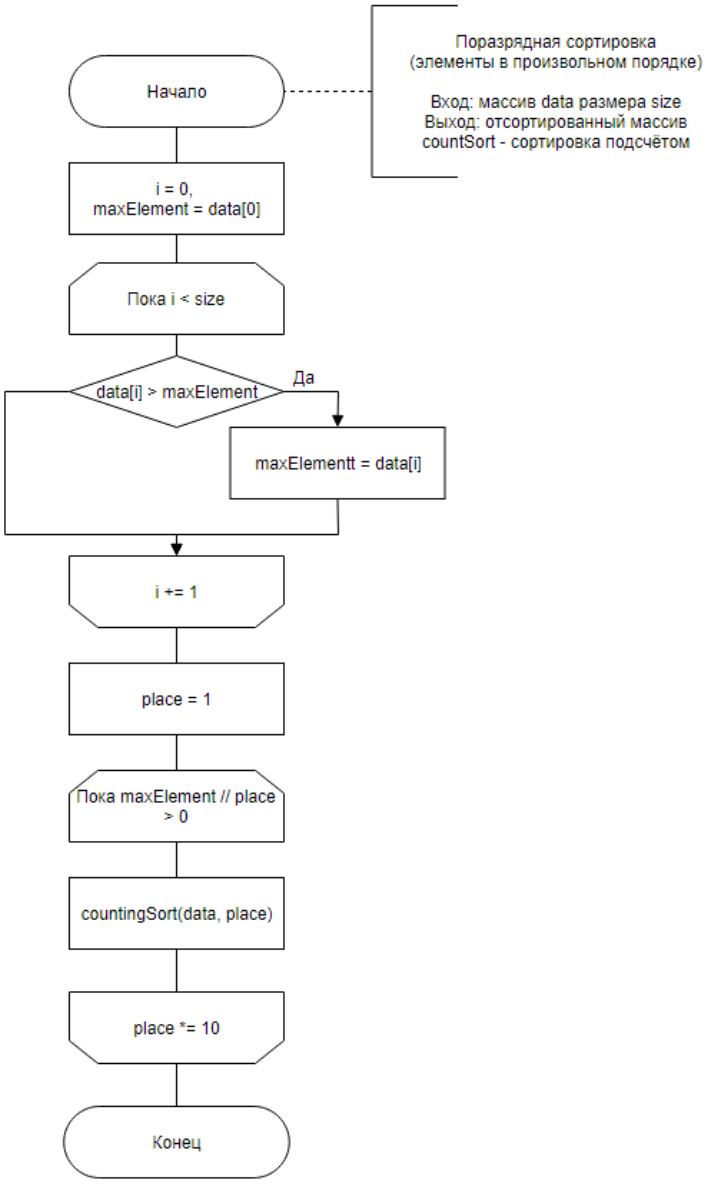

Рисунок 2.4 – Схема алгоритма сортировки бинарным деревом

2.3 Модель вычислений для проведения оценки трудоёмкости

Введем модель вычислений [5], которая потребуется для определния трудоемкости каждого отдельно взятого алгоритма сортировки:

1. операции из списка (2.1) имеют трудоемкость равную 1;

$$+, -, /, *, \%, =, +=, -=, *=, /=, \%=, ==, !=, <, >, <=, >=, [], ++, --$$
(2.1)

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.2)$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \quad (2.3)$$

4. трудоемкость вызова функции равна 0.

2.4 Трудоёмкость алгоритмов

Определим трудоемкость выбранных алгоритмов сортировки по коду.

2.4.1 Алгоритм блинной сортировки

- Трудоёмкость сравнения внешнего цикла $if(size > 1)$, которая равна (2.4):

$$f_{if} = 1 + \begin{cases} 0, & \text{л.с.} \\ f_{cycle}, & \text{х.с.} \end{cases} \quad (2.4)$$

- Трудоёмкость цикла, количество итераций которого равна $size$, которая равна (2.5):

$$f_{cycle} = 2 + size \cdot f_{body} \tag{2.5}$$

- Трудоёмкость тела цикла f_{body} равна (2.6):

$$f_{body} = 1 + \frac{(size + 2) \cdot 5}{2} + 3 + f_{if} + 2 \tag{2.6}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.7):

$$f_{if} = 4 + \begin{cases} 2, & \text{л.с.} \\ 7 + \frac{size \cdot 3 \cdot 2}{2}, & \text{х.с.} \end{cases} \tag{2.7}$$

Трудоёмкость в лучшем случае (2.8):

$$f_{best} = 1 \approx O(1) \tag{2.8}$$

Трудоёмкость в худшем случае (2.9):

$$f_{worst} = 2 + size \cdot (6 + \frac{5 \cdot (size + 2)}{2} + 7 + size \cdot 3) \tag{2.9}$$

$$= 2 + 14 \cdot size + \frac{5 \cdot size^2}{2} \approx \frac{5 \cdot size^2}{2} \approx O(size^2) \tag{2.10}$$

2.4.2 Алгоритм поразрядной сортировки

Трудоёмкость алгоритма поразрядной сортировки равна (2.11):

$$\begin{gathered} f_{radix} = 1 + 2 + 5 \cdot size + 1 + 2 + m * (f_{count} + 1 + 2), \text{ где} \\ \text{m - количество разрядов в максимальном элементе} \\ f_{count} \text{ - трудоёмкость алгоритма сортировки подсчётом (2.12)} \end{gathered} \tag{2.11}$$

Трудоёмкость алгоритма сортировки подсчётом (2.12):

$$\begin{aligned} f_{count} &= size + 10 + 2 + size \cdot 8 + 2 + 10 \cdot 6 + 3 \\ &+ size \cdot 14 + 1 + size \cdot 5 \\ &= 78 + 28 \cdot size \end{aligned} \tag{2.12}$$

Итоговая трудоёмкость порязрядной сортировке, использующей сортировку подсчётом в рамках одного разряда (2.13)

$$f_{radix} = 6 + 2 \cdot size + m \cdot (28 \cdot size + 78) \approx 28 \cdot m \cdot size \approx O(size \cdot m) \tag{2.13}$$

2.4.3 Алгоритм сортировки бинарным деревом

Трудоёмкость данного алгоритма посчитаем следующим образом: сортировка – преобразование массива или списка в бинарное дерево поиска посредством операции вставки нового элемента в бинарное дерево. Операция вставки в бинарное дерево имеет сложность $\log_2(size)$, где $size$ - количество элементов в дереве. Для преобразования массива или списка размером $size$ потребуется использовать операцию вставки в биннарное дерево $size$ раз, таким образом, итоговая трудоёмкость данной сортировки будет равна (2.14):

$$f_{radix} = size \cdot \log_2(size) \tag{2.14}$$

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Также для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги сортировок.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*[6]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time*[7].

3.2 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.py* - файл, содержащий весь служебный код;
- *sorts.py* - файл, содержащий код всех сортировок.

3.3 Листинги кода

В листингах 3.1, 3.2,3.3, 3.4 представлены реализации алгоритмов сортировок (блинной, поразрядной, подсчётом и бинарным деревом).

Листинг 3.1 – Алгоритм блинной сортировки

```
1     def pancakeSort(data, size):
2         if size > 1:
3             for currentSize in range(size, 1, -1):
4                 maxIndex = max(range(currentSize), key =
                      data.__getitem__)
5                 if maxIndex + 1 != currentSize:
6                     if maxIndex != 0:
7                         data[:maxIndex + 1] =
                              reversed(data[:maxIndex + 1])
8                     data[:currentSize] =
                          reversed(data[:currentSize])
9         return data
```

Листинг 3.2 – Алгоритм поразрядной сортировки

```
1 def radixSort(data, size):
2     maxElement = max(data)
3     place = 1
4     while maxElement // place > 0:
5         countingSort(data, size, place)
6         place *= 10
7     return data
```

Листинг 3.3 – Алгоритм сортировки подсчётом

```
1  def countingSort(array, size, place):
2      output = [0] * size
3      count = [0] * 10
4      for i in range(0, size):
5          count[(array[i] // place) % 10] += 1
6      for i in range(1, 10):
7          count[i] += count[i - 1]
8          i = size - 1
9      while i >= 0:
10         index = array[i] // place
11         output[count[index % 10] - 1] = array[i]
12         count[index % 10] -= 1
13         i -= 1
14     for i in range(0, size):
15         array[i] = output[i]
```

Листинг 3.4 – Алгоритм сортировки подсчётом

```
1         def countingSort(array, size, place):
2         output = [0] * size
3         count = [0] * 10
4         for i in range(0, size):
5         count[(array[i] // place) % 10] += 1
6         for i in range(1, 10):
7         count[i] += count[i - 1]
8         i = size - 1
9         while i >= 0:
10        index = array[i] // place
11        output[count[index % 10] - 1] = array[i]
12        count[index % 10] -= 1
13        i -= 1
14        for i in range(0, size):
15        array[i] = output[i]
```

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты *для всех сортировок* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[9, 7, 5, 1, 4]$	$[1, 4, 5, 7, 9]$	$[1, 4, 5, 7, 9]$
$[5]$	$[5]$	$[5]$
$[]$	$[]$	$[]$

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого алгоритма была вычислена трудоемкость и оценены лучший и худший случаи.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программа, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- операционная система: Mac OS Monterey Версия 12.5.1 (21G83) [8] x86_64;
- память: 16 GB;
- процессор: 2,7 GHz 4-ядерный процессор Intel Core i7 [9].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.

```
Меню
 1. Блинная сортировка
 2. Поразрядная сортировка
 3. Сортировка бинарным деревом
 4. Замер времени
 0. Выход

 Выбор: 2
Введите массив поэлементно в одной строке (окончание - Enter):
1 2 3 4 5 5 4 3 2 1
[1, 1, 2, 2, 3, 3, 4, 4, 5, 5]
Меню
 1. Блинная сортировка
 2. Поразрядная сортировка
 3. Сортировка бинарным деревом
 4. Замер времени
 0. Выход

 Выбор:
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени process_time(...) из библиотеки time на Python. Функция возвращает пользовательское процессорное временя типа float.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Результаты замеров времени работы алгоритмов сортировки на различных входных данных (в мс) приведены в таблицах 4.1, 4.2 и 4.3.

Таблица 4.1 – Отсортированные данные

Размер	Блинная	Поразрядная	Бинарным деревом
100	0.1662	0.0714	0.8730
200	0.5113	0.2058	3.3267
300	1.1026	0.3131	7.6354
400	2.0140	0.4364	13.6751
500	3.3046	0.5591	21.5524
600	5.0567	0.6798	31.3052
700	6.6944	0.7852	43.0406
800	8.5163	0.8766	56.4318

Таблица 4.2 – Отсортированные в обратном порядке данные

Размер	Блинная	Поразрядная	Бинарным деревом
100	0.1606	0.1048	0.7138
200	0.5005	0.2008	2.7633
300	1.0747	0.3110	6.3060
400	1.9383	0.4312	11.3831
500	3.1148	0.5427	18.0577
600	4.6409	0.6693	26.0260
700	6.7969	0.8317	36.7397
800	8.7922	0.9583	47.2628

Также на рисунках 4.2, 4.3, 4.4 приведены графические результаты замеров работы сортировок в зависимости от размера входного массива.

Таблица 4.3 – Случайные данные

Размер	Блинная	Поразрядная	Бинарным деревом
100	0.2734	0.1043	0.1560
200	0.8321	0.2090	0.3756
300	1.6837	0.3142	0.6025
400	2.8938	0.4281	0.9785
500	4.4438	0.5419	1.1784
600	6.4153	0.6704	1.5523
700	8.6692	0.7678	1.9018
800	11.3752	0.8992	2.2986

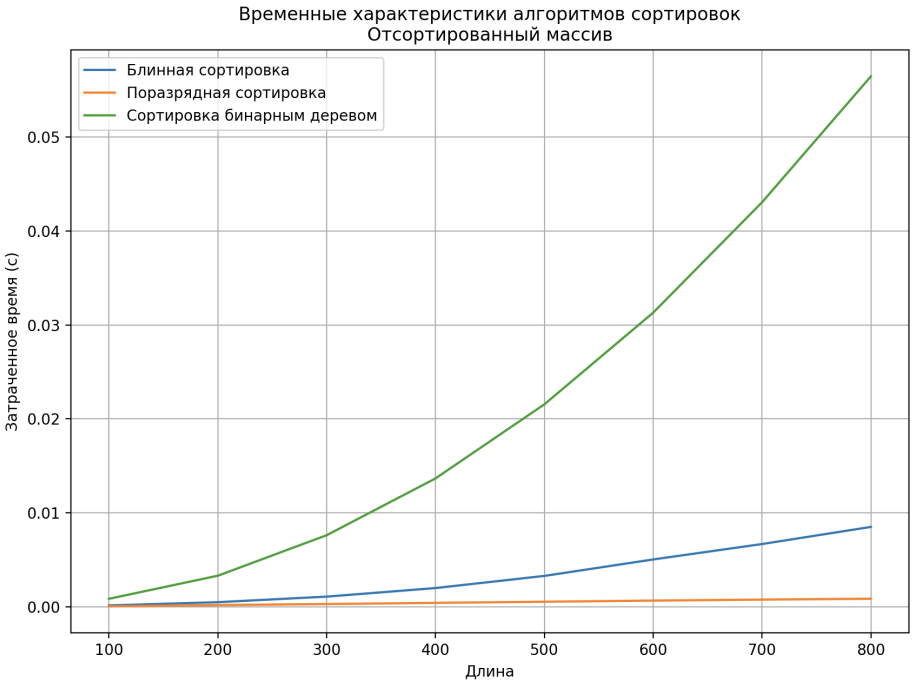

Рисунок 4.2 – Отсортированный массив

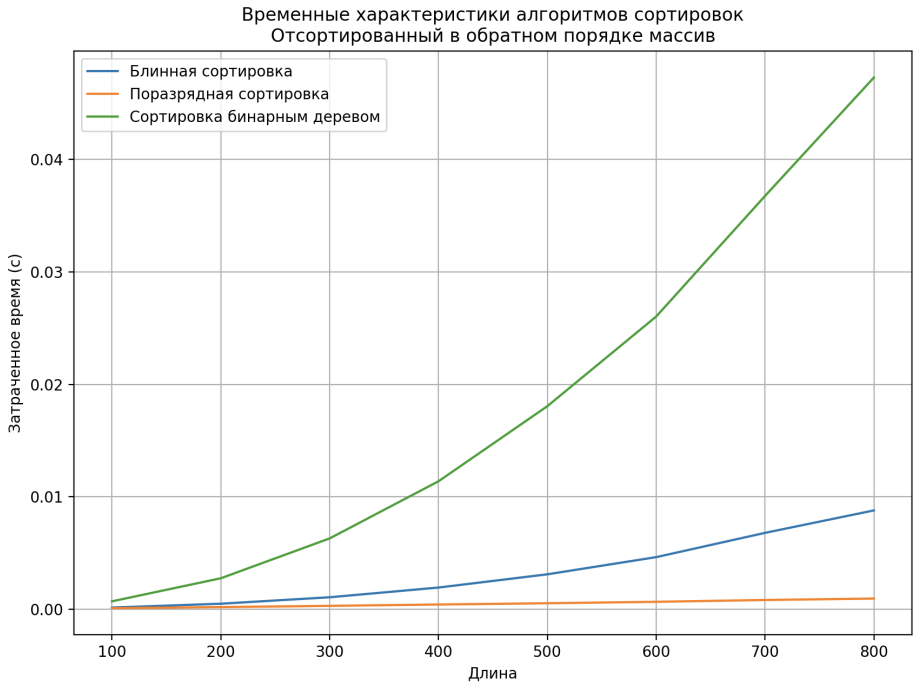

Рисунок 4.3 – Отсортированный в обратном порядке массив

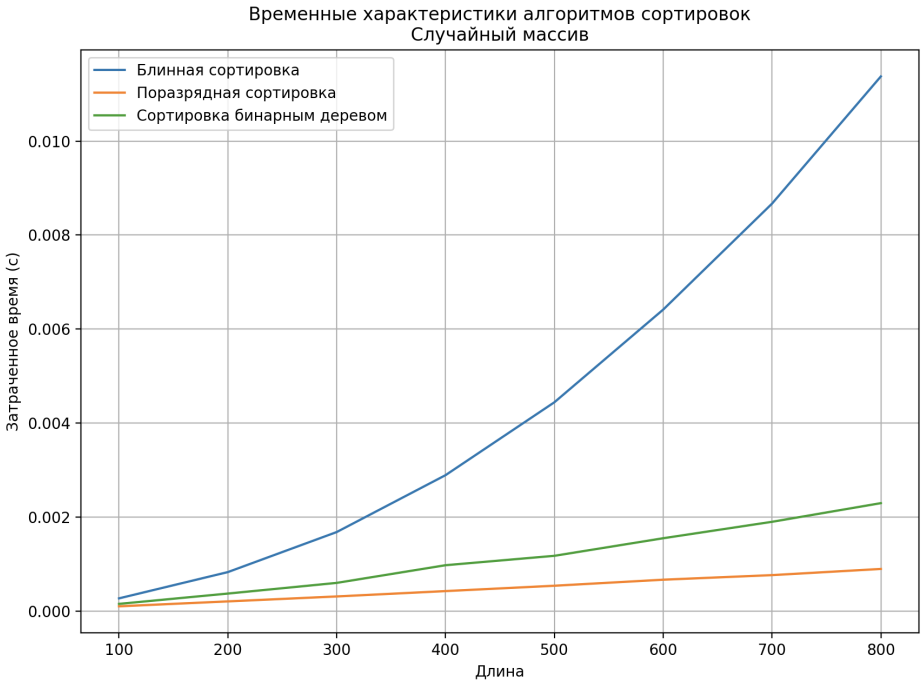

Рисунок 4.4 – Случайный массив

Вывод

Исходя из полученных результатов, сортировка бинарным деревом на отсортированных массивах и блинная сортировка на случайном массиве работают дольше всех (примерно в 40 раза дольше, чем поразрядная сортировка), при этом поразрядная сортировка показала себя лучше всех на любых данных. Можно сделать вывод, что использование сортировки бинарным деревом показывает наилучший результат при случайных, никак не отсортированных данных, т.к. при отсортированных данных обычное бинарное дерево вырождается в связный список, из-за чего вырастает высота дерева. Поразрядная сортировка же эффективнее в том случае, когда приблизительно известно максимальное количество разрядов в сортируемых данных.

Теоретические результаты замеров и полученные практически результаты совпадают.

Заключение

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи:

- были изучены и реализованы алгоритмы сортировки: блинная, поразрядная и бинарным деревом;
- была выбрана модель вычисления и проведен сравнительный анализ трудоёмкостей выбранных алгоритмов сортировки;
- на основе экспериментальных данных проведено сравнение выбранных алгоритмов сортировки;
- подготовлен отчет о лабораторной работе.

Исходя из полученных результатов, сортировка бинарным деревом на отсортированных массивах и блинная сортировка на случайном массиве работают дольше всех (примерно в 40 раза дольше, чем поразрядная сортировка), при этом поразрядная сортировка показала себя лучше всех на любых данных. Можно сделать вывод, что использование сортировки бинарным деревом показывает наилучший результат при случайных, никак не отсортированных данных, т.к. при отсортированных данных обычное бинарное дерево вырождается в связный список, из-за чего вырастает высота дерева. Поразрядная сортировка же эффективнее в том случае, когда приблизительно известно максимальное количество разрядов в сортируемых данных.

Список литературы

[1] Блинная сортировка [Электронный ресурс]. Режим доступа: https://www.sciencedirect.com/science/article/pii/0012365X79900682 (дата обращения: 06.10.2022).

[2] Поразрядная сортировка [Электронный ресурс]. Режим доступа: https://www.williamspublishing.com/Books/sci_Knuth3.html (дата обращения: 06.10.2022).

[3] Поразрядная сортировка [Электронный ресурс]. Режим доступа: https://www.studmed.ru/levitin-a-algoritmy-vvedenie-v-razrabotku-i-analiz_0de7d080a05.html (дата обращения: 06.10.2022).

[4] Двоичные деревья поиска [Электронный ресурс]. Режим доступа: http://rsdn.org/article/alg/binstree.xml (дата обращения: 06.10.2022).

[5] М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ.

[6] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 06.10.2022).

[7] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 04.10.2021).

[8] macOS Monterey [Электронный ресурс]. Режим доступа: https://www.apple.com/macos/monterey/ (дата обращения: 17.09.2022).

[9] Процессор Intel® Core™ i7 [Электронный ресурс]. Режим доступа: https://www.intel.com/processors/core/i7/docs (дата обращения: 17.09.2022).