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 № 5 по курсу "Анализ алгоритмов"

Тема Конвейерная обработка данных

Студент Калашков П. А.

Группа ИУ7-56Б

Оценка (баллы)

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

Москва — 2022 г.

Содержание

Введение

Использование параллельной обработки открывает новые способы для ускорения работы программ. Конвейрная обработка является одним из примеров, где использование принципов параллельности помогает ускорить обработку данных. Суть та же, что и при работе реальных конвейрных лент — материал (данное) поступает на обработку, после окончания обработки материал передается на место следующего обработчина, при этом предыдущий обработчик не ждёт полного цикла обработки материала, а получает новый материал и работает с ним.

Цель работы: изучение принципов конвейрной обработки данных.

Задачи работы:

1) изучить основы конвейрной обработки данных;
2) описать алгоритмы обработки матрицы, которые будут использоваться в текущей лабораторной работе;
3) сравнить и проанализировать реализации алгоритмов по затраченным времени;
4) описать и обосновать полученные результаты в отчёте о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будет рассмотрена информация, касающаяся основ конвейерной обработки данных.

1.1 Конвейерная обработка данных

Конвейер [1] (англ. *conway*) — организация вычислений, при которой увеличивается количество выполняемых инструкций за единицу времени за счет использования принципов параллельности.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так, обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает, благодаря тому, что одновременно на различных ступенях конвейера выполняется несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

Конвейеризация повзволяет увелпчить пропускную способность процессора (количество команд, завершающихся в единицу времени), но она не сокращает время выполнения отдельной команды. В действительности она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с хранением промежуточных результатов. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой, неконвейерной схемой.

1.2 Описание алгоритмов

В качестве примера для операции, подвергающейся конвейерной обработке, будет обрабатываться матрица. Всего будет использовано три ленты, ко-

торые делают следующее:

1) находится среднее арифмитическое значений матрицы;

2) находится максимальный элемент матрицы;

3) нечетный элемент матрицы заменяется на среднее арифметическое матрицы, а четные - на макимальный элемент.

Вывод

В данном разделе было рассмотрено понятие конвейерной обработки, а также выбраны этапы для обработки матрицы, которые будут обрабатывать ленты конвейера.

Программа будет получать на вход количество задач (количество матриц), размер матрицы (используются только квадратные матрицы), а также выбор алгоритма — линейный или конвейерный. При неверном вводе какого-то из значений будет выдаваться сообщение об ошибке.

Реализуемое программное обеспечение будет давать возможность получить журнал программы для установленного числа задач при линейной и конвейрной обработке. Также будет возможность провести тестирование по времени для разного количества задач (матриц) и разных размеров самих матриц.

2 Конструкторская часть

В этом разделе будут рассмотрены описания используемых типов данных, а также схемы алгоритмов конвейрной и линейной обработки матриц.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

- количество задач (матриц) - целое число;
- размер матрицы - целое число;
- структура $matrix\_s$ - содержит информацию о матрице – сама матрицы, её размер, а также информацию о найденном среднем арифметическом и максимальном элементах;
- структура $queues\_t$ - содержит информацию об очередях.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма линейной обработки матрицы. На рисунке 2.2 схема алгоритма конвейрной обработки матрицы, а на рисунках 2.3–2.5 - схемы потоков обработки матрицы (ленты конвейера).

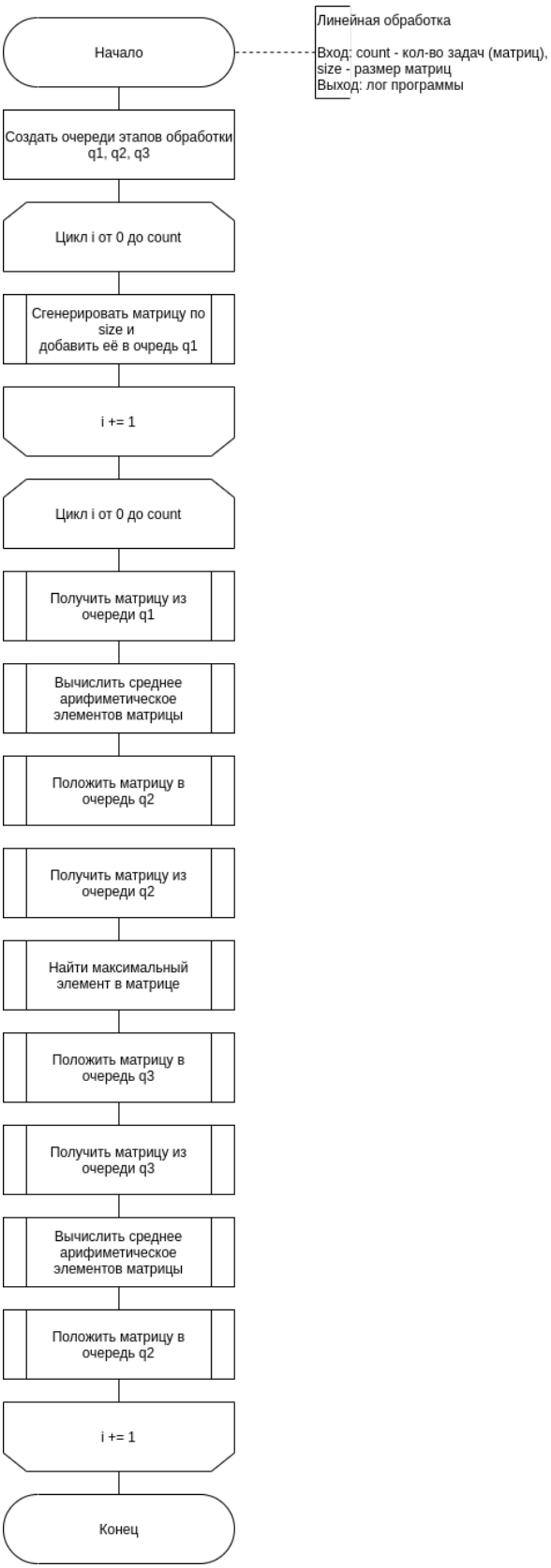

Рисунок 2.1 – Схема алгоритма линейной обработки матрицы

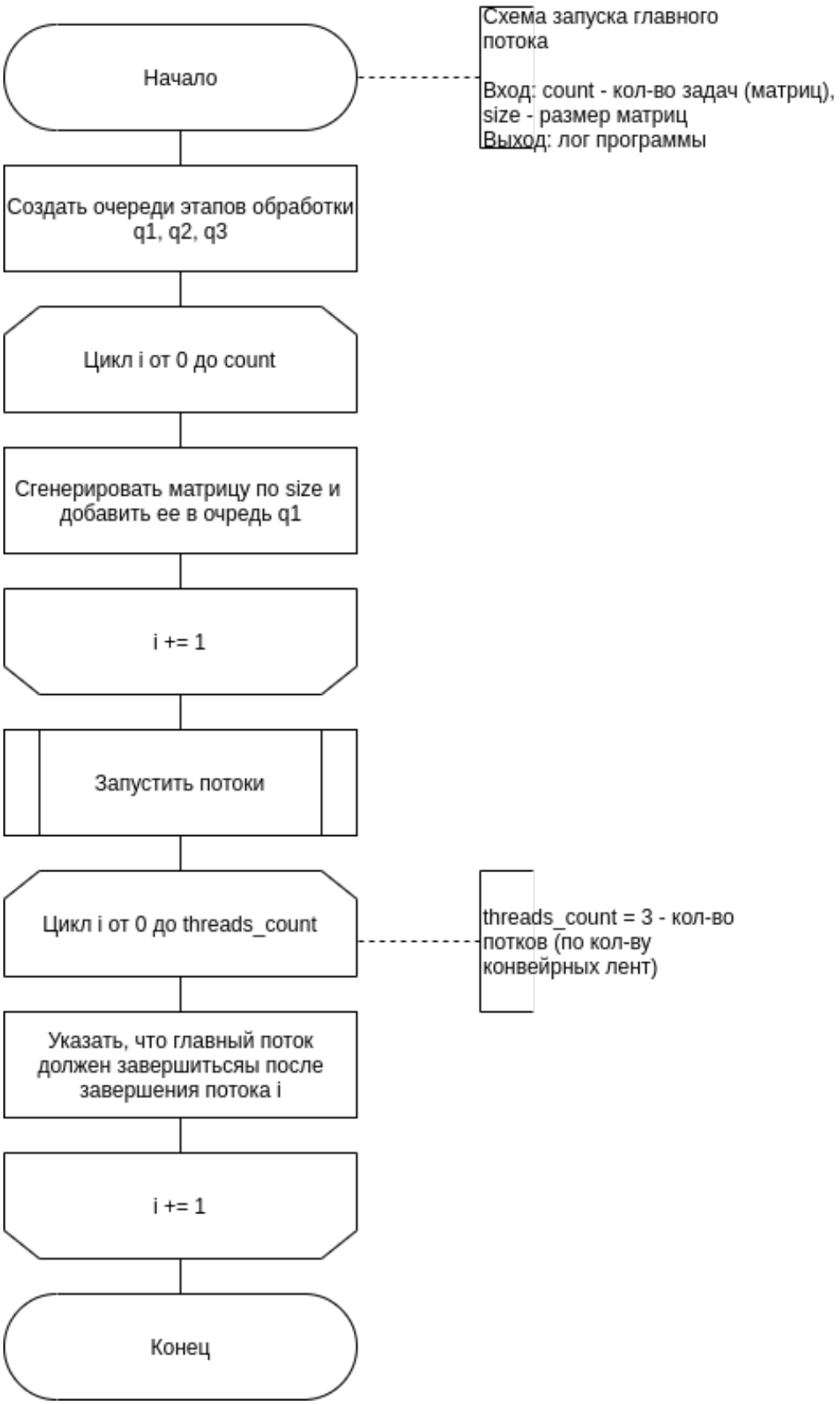

Рисунок 2.2 – Схема конвейрной обработки матрицы

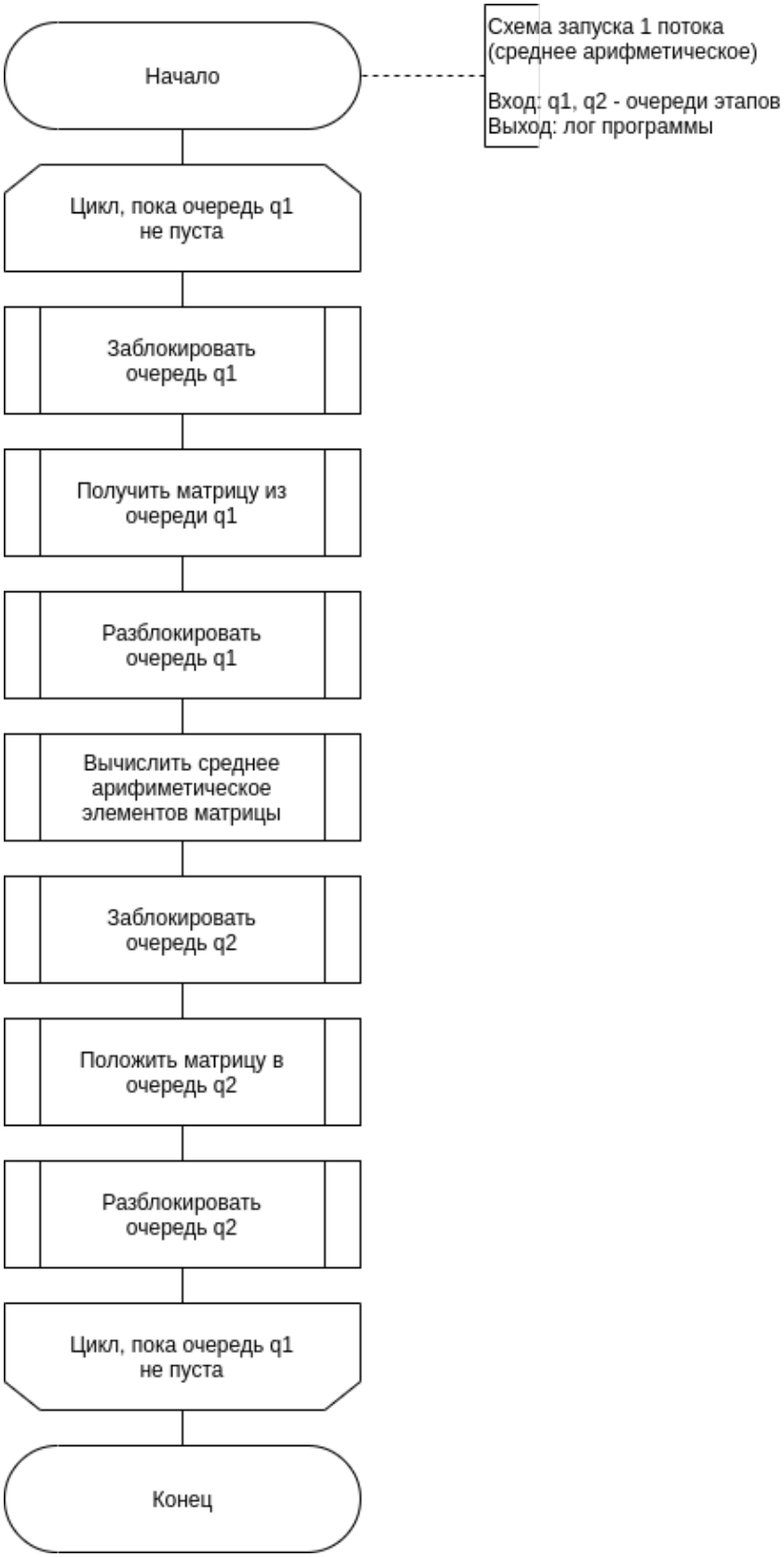

Рисунок 2.3 – Схема 1 потока обработки матрицы — нахождение среднего арифметического

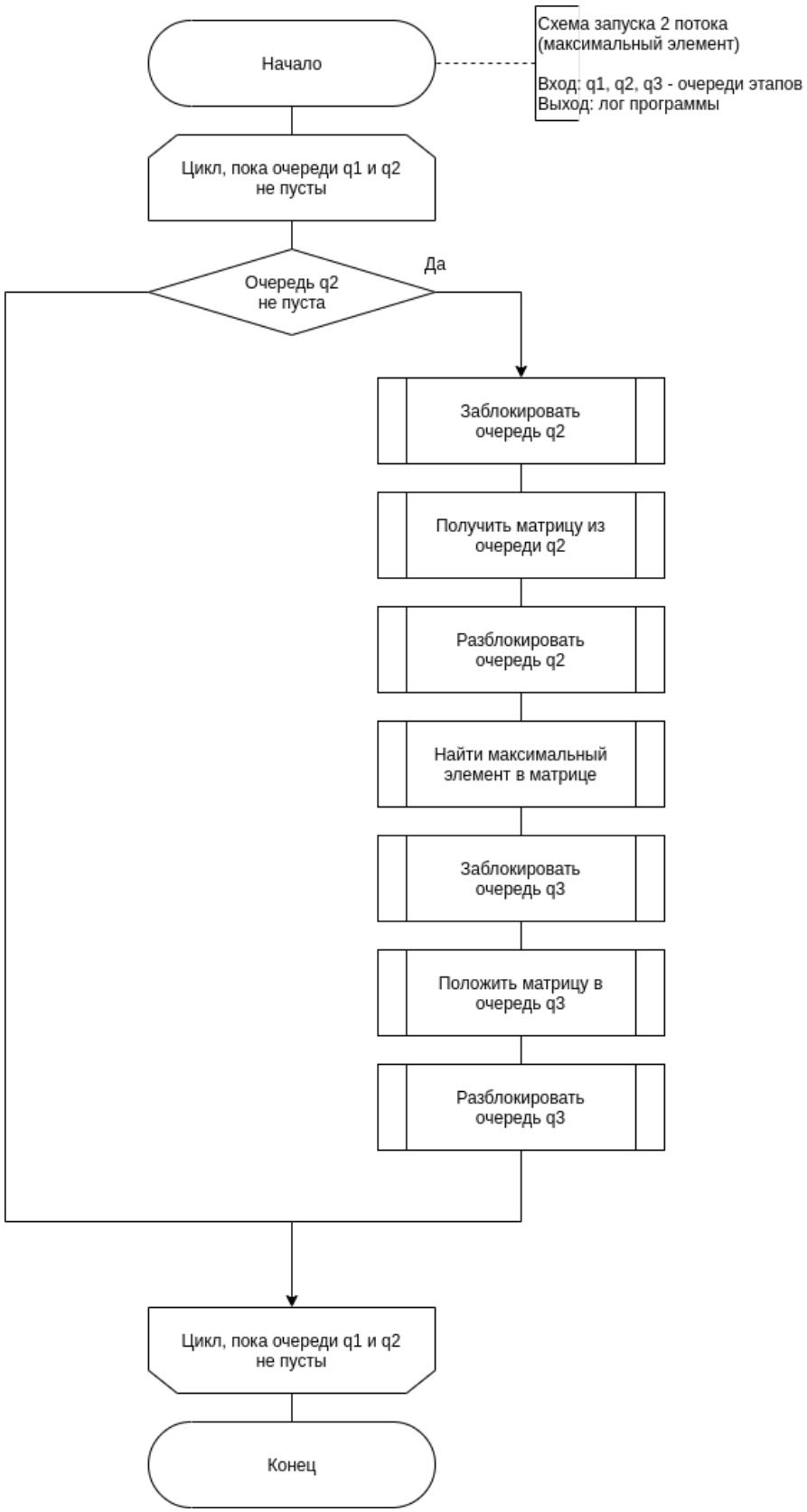

Рисунок 2.4 – Схема 2 потока обработки матрицы — нахождение максимального элемента

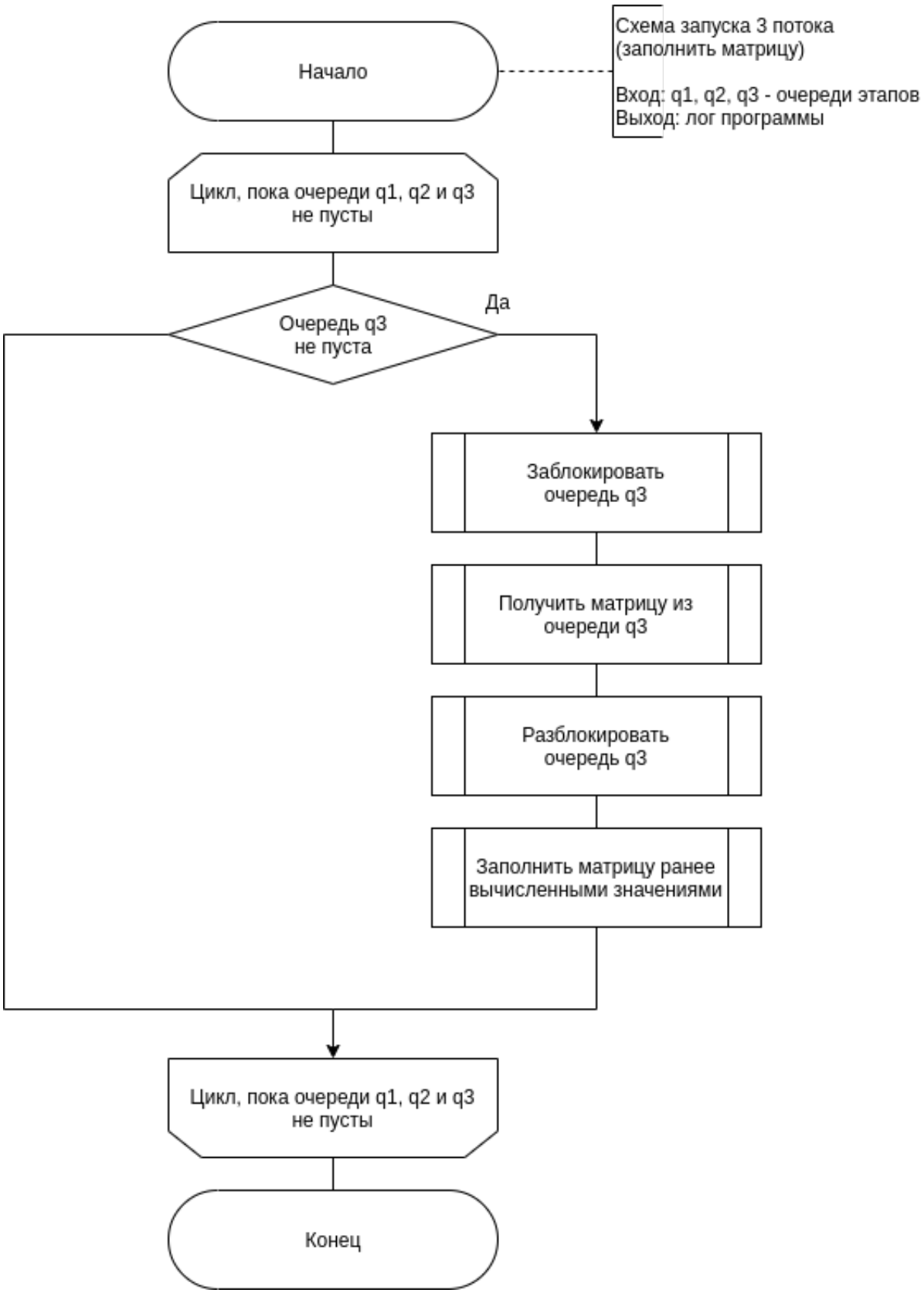

Рисунок 2.5 – Схема 3 потока обработки матрицы — заполнения матрицы новыми значенями

2.3 Классы эквивалентности при функциональном тестировании

Для функционального тестирования выделены классы эквивалентности, представленные ниже.

1. Неверно выбран пункт меню - не число или число, меньшее 0 или большее 4.
2. Неверно введено количество матриц - не число или число, меньшее 1.
3. Неверно введен размер матриц - не число или число, меньшее 2.
4. Корректный ввод всех параметров.

2.4 Вывод

В данном разделе были построены схемы алгоритмов, рассматриваемых в лабораторной работе, были описаны классы эквивалентности для тестирования, структура программы.

3 Технологическая часть

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

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [5]. В текущей лабораторной работе требуется замерить процессорное время работы выполняемой программы и визуализировать результаты при помощи графиков. Инструменты для этого присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time* [6].

3.2 Сведения о модулях программы

Программа состоит из шести модулей:

1) *main.py* – файл, содержащий точку входа;
2) *menu.py* – файл, содержащий код меню программы;
3) *test.py* – файл, содержаший код тестирования алгоритмов;
4) *utils.py* – файл, содержащий служебные алгоритмы;
5) *constants.py* – файл, содержаший константы программы;
6) *algorythms.py* – файл, содержащий код всех алгоритмов.

3.3 Реализации алгоритмов

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 бинарным деревом

```
class BSTree:
    def __init__(self):
        self.root = None

    def __str__(self):
        rootStr = self.root.toString()[:-2]
        return "[" + rootStr + "]"

    def inorder(self):
        if self.root is not None:
            self.root.inorder()

    def add(self, key):
        new_node = BSTNode(key)
        if self.root is None:
            self.root = new_node
        else:
            self.root.insert(new_node)

def bstSort(data, n):
    bstree = BSTree()
    for x in data:
        bstree.add(x)
    return bstree
```

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 трудоёмкость и оценены лучший и худший случаи трудоёмкости.

4 Исследовательская часть

В данном разделе будут приведён пример работы программа, а также проведён сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени работы реализации алгоритма статической раздачи информации, представлены далее:

- операционная система Mac OS Monterey Версия 12.5.1 (21G83) [7] x86_64;
- память 16 ГБ;
- четырёхъядерный процессор Intel Core i7 с тактовой частотой 2,7 ГГц [8].

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 реализаций 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 реализаций алгоритмов сортировки на различных входных данных (в мс) приведены в таблицах 4.1, 4.2 и 4.3.

Таблица 4.1 – Результаты замеров реализаций сортировок, входными данными являлись отсортированные по возрастанию значений массивы.

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 – Результаты замеров реализаций сортировок, входными данными являлись отсортированные по убыванию значений массивы.

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

Таблица 4.3 – Результаты замеров реализаций сортировок, входными данными являлись заполненные числами со случайными значениями массивы.

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

Также на рисунках 4.2, 4.3, 4.4 приведены графические результаты замеров процессорного времени выполнения реализаций алгоритмов сортировок в зависимости от размера входного массива.

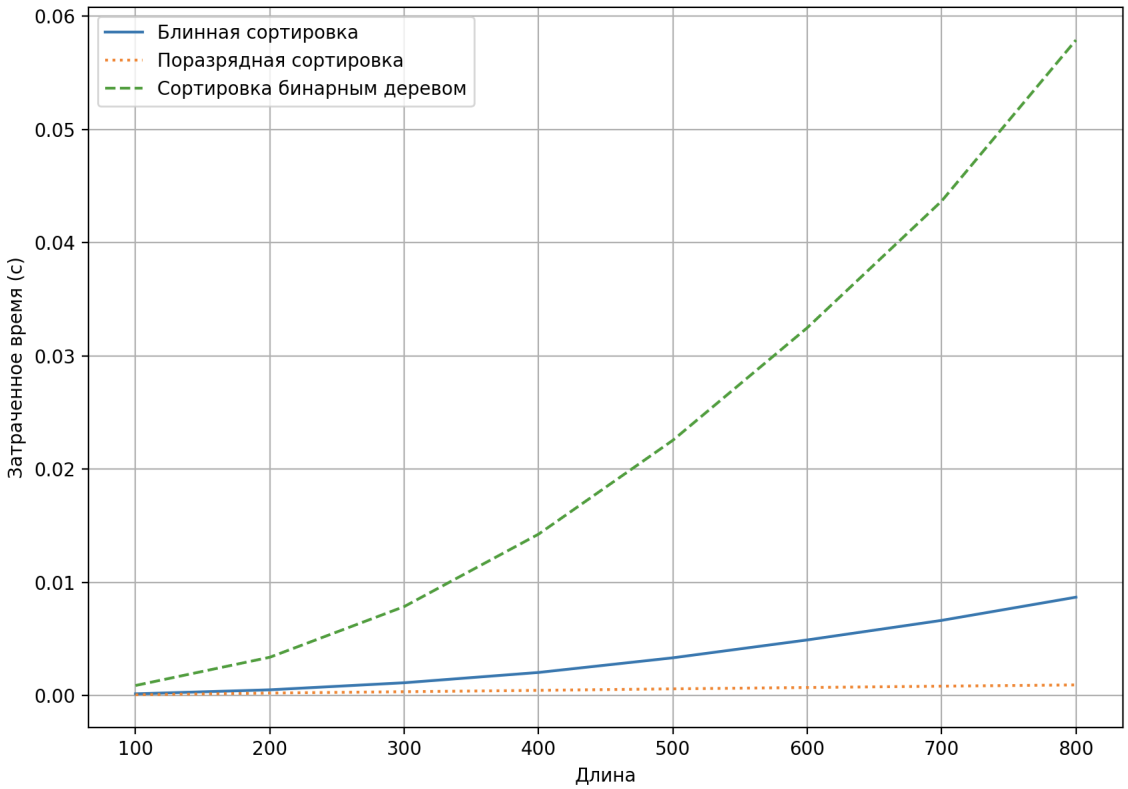

Рисунок 4.2 – Результаты замеров реализаций сортировок, входными данными являлись отсортированные по возрастанию значений массивы.

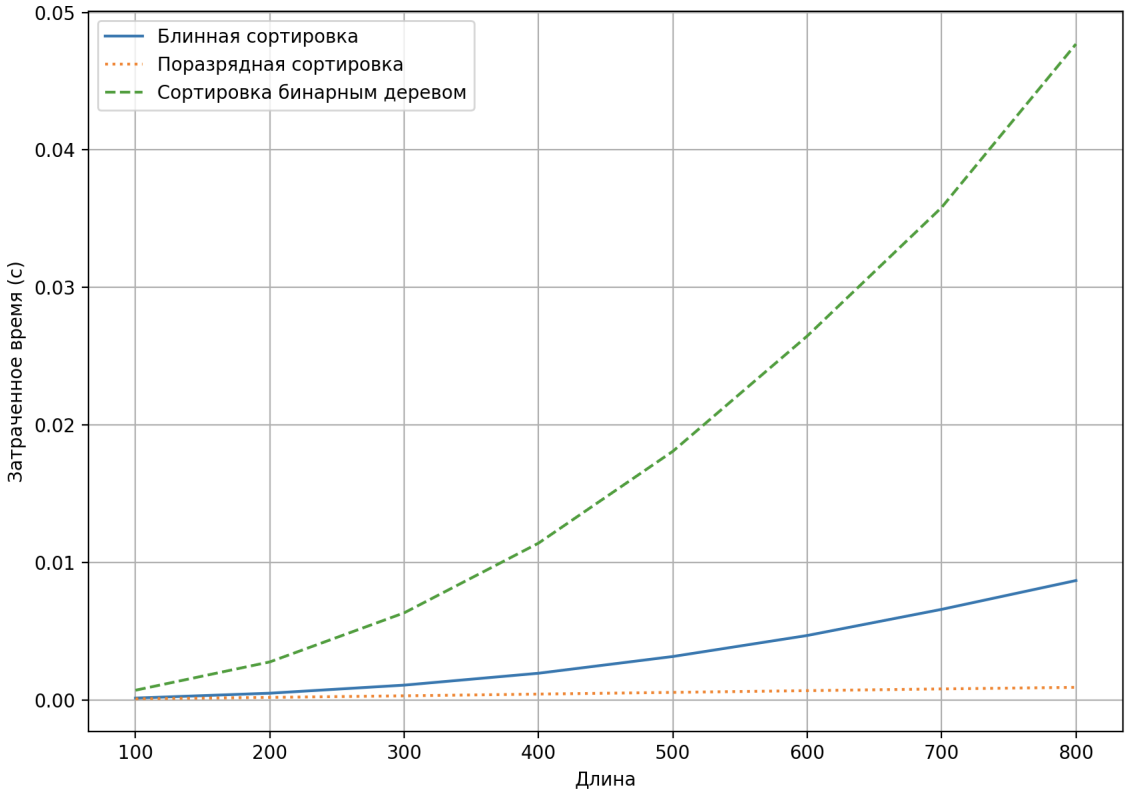

Рисунок 4.3 – Результаты замеров реализаций сортировок, входными данными являлись отсортированные по убыванию значений массивы.

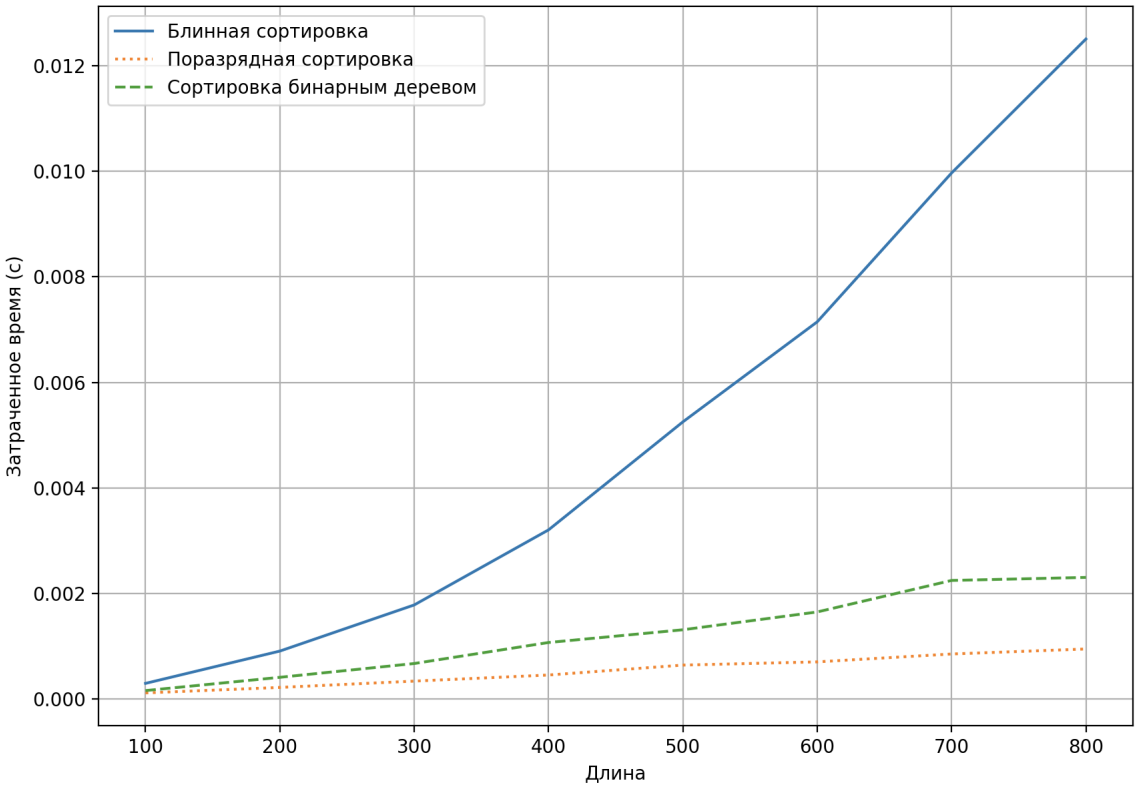

Рисунок 4.4 – Результаты замеров реализаций сортировок, входными данными являлись заполненные числами со случайными значениями массивы.

Сортировка бинарным деревом работает дольше других рассмотренных сортировок на отсортированных масссивах (рисунки 4.2, 4.3), в то время как на случайно заполненном массиве дольше всех работает блинная сортировка (рисунок 4.4). Поразрядная сортировка во всех трёх случаях оказывается самой быстрой.

При этом для блинной сортировки и для сортировки бинарным деревом худшим случаем является подача на вход отсортированного в противоположном поставленному направлению массива, а лучшим – отсортированного в поставленном направлении. У пораздрядной сортировки нет худших и лучших случаев нет.

Вывод

Исходя из полученных результатов, сортировка бинарным деревом на отсортированных массивах и блинная сортировка на случайном массиве работают дольше всех (на длине массива в 800 элементов примерно в 40 раз 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 заранее известно максимальное количество разрядов в сортируемых данных.

Теоретические результаты оценки трудоёмкости и полученные практическим образом результаты замеров совпадают.

Заключение

Была достигнута цель работы: изучены и исследованы трудоёмкости алгоритмов сортировок – блинной, поразрядной, бинарным деревом. Также в ходе выполнения лабораторной работы были решены следующие задачи:

1) были изучены и реализованы алгоритмы сортировки: блинная, поразрядная и бинарным деревом;

2) было измерено время работы реализаций алгоритмов выбранных сортировок;

3) были проведены сравнение и анализ трудоёмкостей алгоритмов на основе теоретических расчетов;

4) были проведены сравнение и анализ реализаций алгоритмов по затраченным процессорному времени и памяти;

5) был подготовлен отчёт о лабораторной работе, представленный как расчётно-пояснительная записка к работе.

Исходя из полученных результатов, сортировка бинарным деревом на отсортированных массивах и блинная сортировка на случайном массиве работают дольше всех (на длине массива в 800 элементов примерно в 40 раз 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 использованных источников

[1] Конвейерная обработка данных [Электронный ресурс]. Режим доступа: https://studref.com/636041/ekonomika/konveyernaya_obrabotka_dannyh (дата обращения: 23.10.2021).

[2] Левитин А. В. Алгоритмы: введение в разработку и анализ. – М.: Издательский дом "Вильямс", - 576 с., 2006.

[3] Акопов Р. Двоичные деревья поиска. – М.: Оптим, RSDN Magazine [Электронный ресурс], 2004. URL: Режим доступа: http://rsdn.org/article/alg/binstree.xml (дата обращения: 17.09.2022).

[4] Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. – М. Наука, Физматлит, 2007. с. 376.

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

[6] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 04.10.2022).

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

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