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 Трудоемкость сортировок

Студент Котляров Н.А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2022 г.

Оглавление

Введение

Одной из важнейших процедур обработки структурированной информации является сортировка.

Сортировка — это процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Такой опредленный порядок позволяет, в некоторых случаях, эффективнее и удобнее работать с заданной последовательностью. В частности, одной из целей сортировки является облегчение задачи поиска элемента в отсортированном множестве.

Алгоритмы сортировки используются практически в любой программой системе. Целью алгоритмов сортирови является упорядочение последовательности элементов данных. Поиск элемента в последовательности отсортированных данных занимает время, пропорциональное логарифму количества элементов в последовательности, а поиск элемента в последовательности не отсортированных данных занимает время, пропорциональное количеству элементов в последовательности, то есть намного больше. Существует множество различных методов сортировки данных. Однако любой алгоритм сортировки можно разбить на три основные части:

- сравнение, определяющее упорядоченность пары элементов;
- перестановка, меняющая местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Одной из важнейшей характеристикой любого алгоритма сортировки является скорость его работы, которая определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных, определенной длины, от этой длины.

Задачи данной лабораторной:

- изучить и реализовать три алгоритма сортировки — шейкером, подсчетом, быстрой;

- провести сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных, а именно по времени;
- 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 представлены описания алгоритмов сортировки перемешиванием, подсчетом, быстрой.

1.1 Сортировка перемешиванием

Сортировка перемешиванием [1] — это разновидность сортировки пузырьком. Отличие в том, что данная сортировка в рамках одной итерации проходит по массиву в обоих направлениях (слева направо и справа налево), тогда как сортировка пузырьком - только в одном направлении (слева направо).

Общие идеи алгоритма:

- обход массива слева направо, аналогично пузырьковой - сравнение соседних элементов, меняя их местами, если левое значение больше правого;
- обход массива в обратном направлении (справа налево), начиная с элемента, который находится перед последним отсортированным, то есть на этом этапе элементы также сравниваются между собой и меняются местами, чтобы наименьшее значение всегда было слева.

1.2 Быстрая сортировка

Быстрая сортировка [2] — алгоритм сортировки, использующий операцию разбиения массива на две части относительно опорного элемента.

Таким образом, алгоритм быстрой сортировки включает в себя два основных этапа:

- разбиение массива относительно опорного элемента;
- рекурсивная сортировка каждой части массива.

Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, т. е. повторно запустить алгоритм на каждом из отрезков.

1.3 Сортировка подсчетом

Сортировка подсчетом [3] - один из способов упорядочить массив за линейное время. Применять его можно только для целых чисел, небольшого диапазона, т.к. он требует $O(M)$ дополнительной памяти, где M — ширина диапазона сортируемых чисел. Алгоритм особо эффективен, когда требуется отсортировать большое количество чисел, значения которых имеют небольшой разброс.

В сортировке подсчетом элементы массива не сравниваются друг с другом. Основные положения алгоритма сортировки следующие.

1) Отсортировать массив $Array$ из N чисел в диапазоне от 1 до m.

2) Подсчитать, сколько раз встречается каждый элемент массива, для этого требуется выполнить следующие шаги:

   2.1) создать вспомогательный массив $Counts$ из m счетчиков, заполненный его нулями;

   2.2) при обходе $Array$, для каждого его элемента увеличить счетчик: $Counts[Value] = Counts[Value] + 1$.

3) Обойти массив $Counts$, для каждого его i-того элемента вывести значение i столько раз, сколько он встретился в исходном массиве. Индекс i при этом соответствует значению числа исходного массива.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: перемешиванием, быстрая и подстчетом. Необходимо выполнить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов и вычисления трудоемкости данных алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки пузырьком, выбором и вставками соответственно.

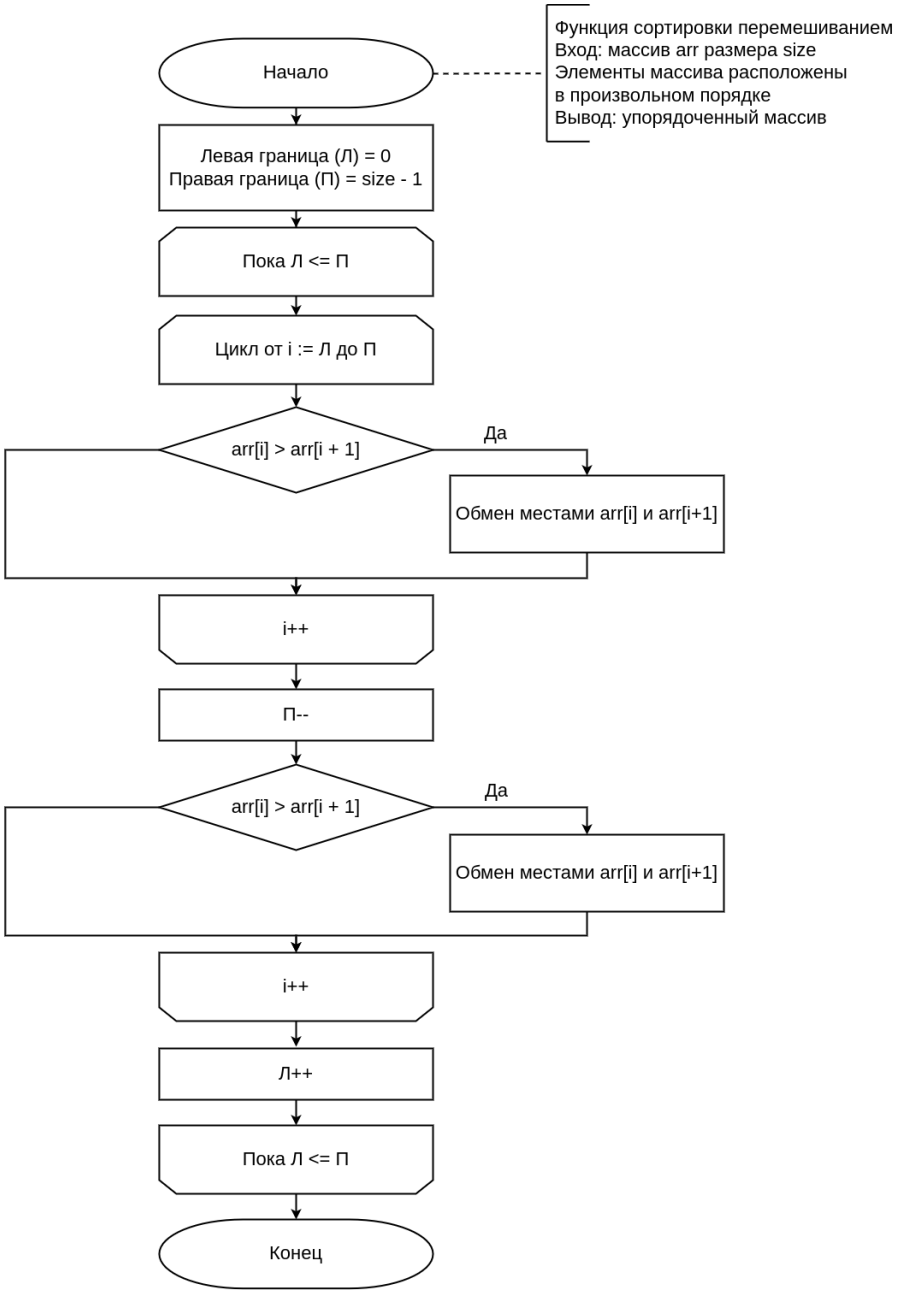

Рисунок 2.1 – Схема алгоритма сортировки перемешиванием

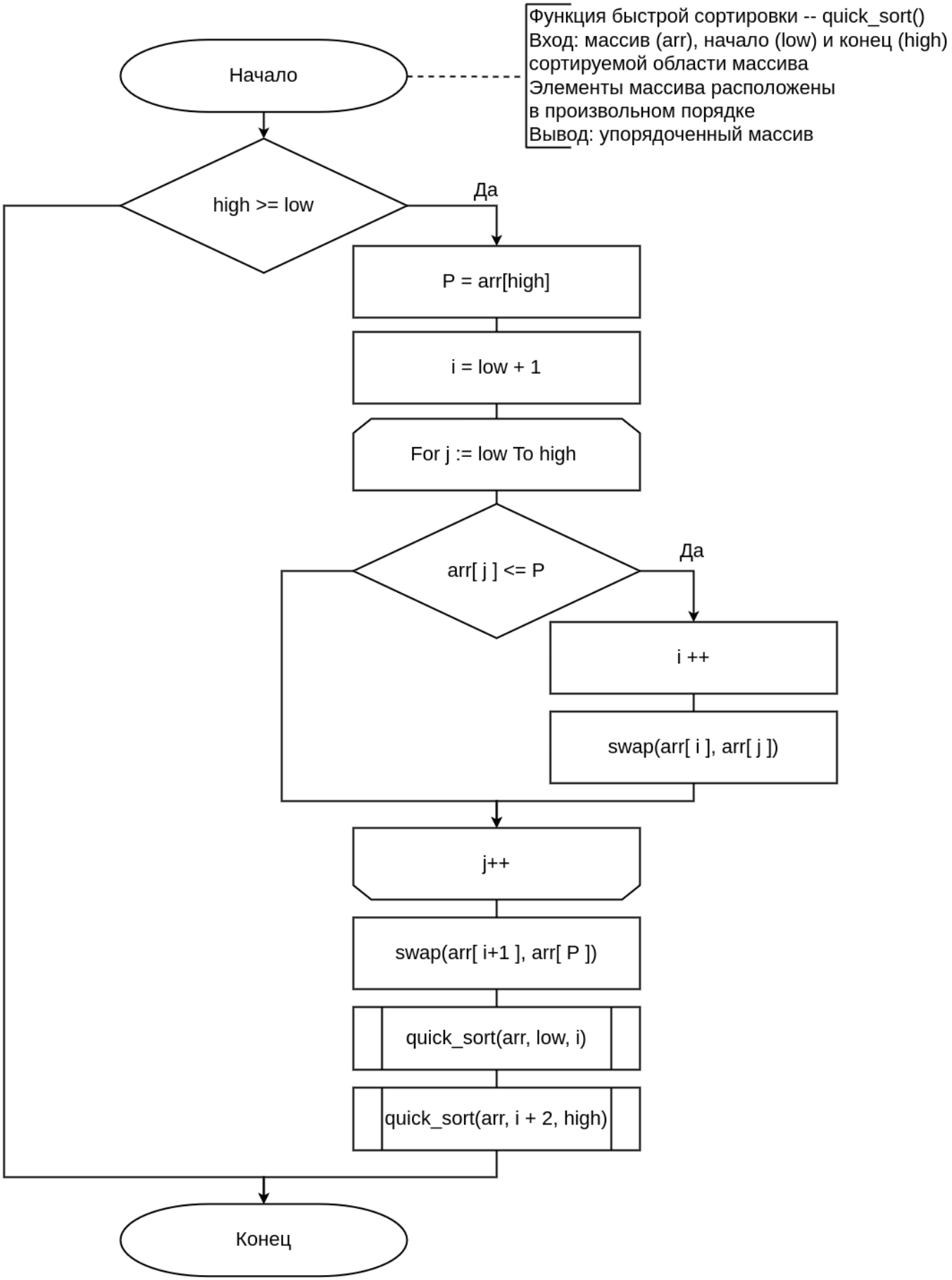

Рисунок 2.2 – Схема алгоритма быстрой сортировки

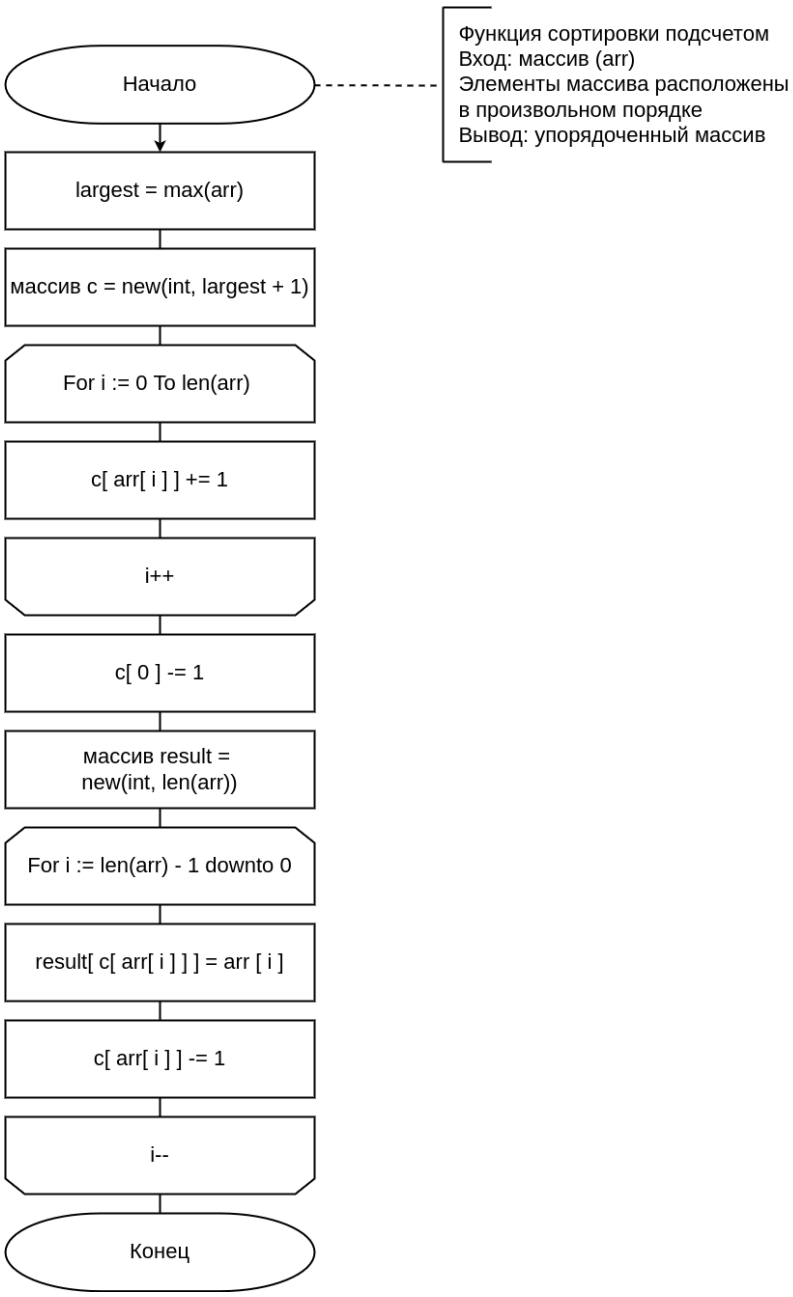

Рисунок 2.3 – Схема алгоритма сортировки подсчетом

2.2 Модель вычислений (оценки трудоемкости)

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. Операции из списка (2.1) имеют трудоемкость 1:

$$+, ++, +=, -, --, -=, !=, <, >, <=, >=, <<, >>, [] \qquad (2.1)$$

2. Операции из списка (2.2) имеют трудоемкость 2:

$$/, / =, *, * =, \%, \% = \tag{2.2}$$

3. трудоемкость оператора выбора `if условие then A else B` рассчитывается как

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4. трудоемкость цикла рассчитывается, как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

5. трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Обозначим во всех последующих вычислениях размер массивов как N.

2.3.1 Алгоритм сортировки перемешиванием

Трудоёмкость сравнения внешнего цикла $WHILE(swap == True)$, которая равна (2.5):

$$f_{outer} = 1 + 2 \cdot (N - 1) \tag{2.5}$$

Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - 1]$, которая равна

$$f_{inner} = 5(N - 1) + \frac{2 \cdot (N - 1)}{2} \cdot (3 + f_{if}) \tag{2.6}$$

Трудоёмкость условия во внутреннем цикле, которая равна

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.7}$$

Трудоёмкость в лучшем случае, который наступает при отсортированном массиве (2.8):

$$f_{best} = -3 + \frac{3}{2}N + \approx \frac{3}{2}N = O(N) \tag{2.8}$$

Трудоёмкость в худшем случае, который наступает при обратно отсортированном массиве (2.9):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2) \tag{2.9}$$

2.3.2 Алгоритм быстрой сортировки

Трудоёмкость алгоритма быстрой сортировки состоит из следующих составляющих.

- Трудоёмкость вычисления опорного элемента разбиения массива на 2 части (2.10): трудоемкость сравнения в цикле, инкремента во внешнем цикле, а также зависимых только от цикла операций, по $i \in [1..m)$, где $m \in N/2^k$, где $k \in [0..f)$, где f равна максимальной глубине стека рекурсии:

$$f_{part} = 2 + 12(m - 1) \tag{2.10}$$

- Каждый рекурсивный вызов порождает 2 других рекурсивных вызова, кроме тривиальных случаев, таким образом максимальная глубина вызова стека будет равна $log_2(N)$ в общем случае.

- Трудоёмкость условия во внутреннем цикле, которая равна

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ 8, & \text{х.с.} \end{cases} \tag{2.11}$$

Трудоёмкость в лучшем случае, который наступает при случайном массиве (2.12):

$$f_{best} = (2 + 12(N) * log_2(N)) \approx 12 * N * log_2(N) = O(N * log_2(N)) \quad (2.12)$$

Трудоёмкость в худшем случае. В худшем случае, например, отсортированном массиве, глубина стека вызова функции равна N (2.13):

$$f_{worst} = (2 + 6(N) * N) \approx 6 * N^2 = O(N^2) \quad (2.13)$$

2.3.3 Алгоритм сортировки подсчетом

Обозначим диапазон значений сортируемого массива за M. Тогда трудоёмкость сортировки подсчетом состоит из:

- первого цикла

$$f_{first} = 2 + 6 * N \quad (2.14)$$

- второго цикла

$$f_{second} = 2 + 6 * M \quad (2.15)$$

- третьего цикла

$$f_{third} = 2 + 8 * N \quad (2.16)$$

Трудоемкость алгоритма сортировки подсчетом зависит от двух параметров: размера массива и диапазона значений. Таким образом, при $N >> M$ трудоемкость будет равна (2.17):

$$f_1 = 4 + 14 * N \approx 14 * N = O(N) \quad (2.17)$$

Таким образом, при $M >> N$ трудоемкость будет равна (2.18):

$$f_1 = 2 + 6 * M \approx 6 * M = O(M) \quad (2.18)$$

Вывод

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

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив сравнимых элементов (целые числа);
- на выходе — тот же массив, но в отсортированном порядке.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python [4].

В данном языке есть все требующиеся для данной лабораторной инструменты разработки.

Процессорное время работы реализаций алгоритмов было замерено с помощью функции process_time() из библиотеки time [5]

3.3 Сведения о модулях программы

Программа состоит из трех модулей:

1. main.py - главный файл программы, в котором располагается меню;
2. algos.py - файд программы, в котором располагаются коды всех алгоритмов;
3. test.py - файл с замерами времени для графического изображения результата.

3.4 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов сортировок (перемешиванием, вставками и выбором).

Листинг 3.1 – Реализация алгоритма быстрой сортировки

```
def partition(arr, low, high):
    p = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= p:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
            arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def quick(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)

def quick_sort(arr):
    quick(arr, 0, len(arr) - 1)
```

Листинг 3.2 – Реализация алгоритма сортировки перемешиванием

```
def shaker_sort(arr):
    swapped = True
    st = 0
    end = len(arr) - 1
    while (swapped):
        swapped = False
        for i in range(st, end):
            if (arr[i] > arr[i + 1]):
                arr[i], arr[i + 1] = arr[i + 1], arr[i]
                swapped = True
        if (not swapped):
            break
        swapped = False
        end -= 1
```

```
15          for i in range(end - 1, st - 1, -1):
16              if (arr[i] > arr[i + 1]):
17                  arr[i], arr[i + 1] = arr[i + 1], arr[i]
18                  swapped = True
19          st += 1
20      return arr
```

Листинг 3.3 – Реализация алгоритма сортировки посчетом

```
1  def counting_sort_help(arr, largest):
2      c = [0]*(largest + 1)
3      for i in range(len(arr)):
4          c[arr[i]] += 1
5
6      c[0] -= 1
7      for i in range(1, largest + 1):
8          c[i] += c[i - 1]
9
10     result = [0]*len(arr)
11
12     for x in reversed(arr):
13         result[c[x]] = x
14         c[x] -= 1
15
16     return result
17
18 def counting_sort(arr):
19     largest = max(arr)
20 return counting_sort_help(arr, largest)
```

3.5 Функциональные тесты

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

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4]$	$[1, 2, 3, 4]$	$[1, 2, 3, 4]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[3, 2, -5, 0, 1]$	$[-5, 0, 0, 2, 3]$	$[-5, 0, 0, 2, 3]$
$[4]$	$[4]$	$[4]$
$[]$	$[]$	$[]$

Вывод

Были реализованы все три алгоритма сортировки. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Manjaro xfce [6] Linux [7] x86_64;
- память: 8 Гб;
- мобильный процессор AMD Ryzen™ 7 3700U @ 2.3 ГГц [8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.

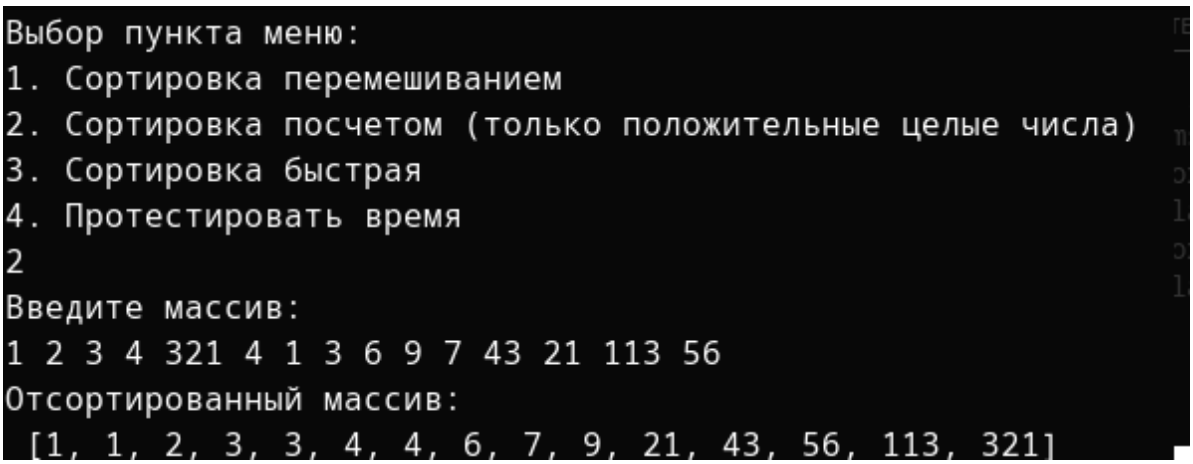

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения реализации алгоритмов

Время работы реализации алгоритмов измерялось при помощи функции process_time() из библиотеки time языка Python. Данная функция всегда возвращает значения времени, а имеено сумму системного и пользовательского процессорного времени текущего процессора, типа float в секундах.

Контрольная точка возвращаемого значения не определена, поэтому допустима только разница между результатами последовательных вызовов.

Результаты замеров приведены в таблицах 4.1, 4.2 и 4.3.

На рисунках 4.2, 4.3 и 4.4, приведены графики зависимостей времени работы алгоритмов сортировки от размеров массивов на отсортированных, обратно отсортированных и случайных данных.

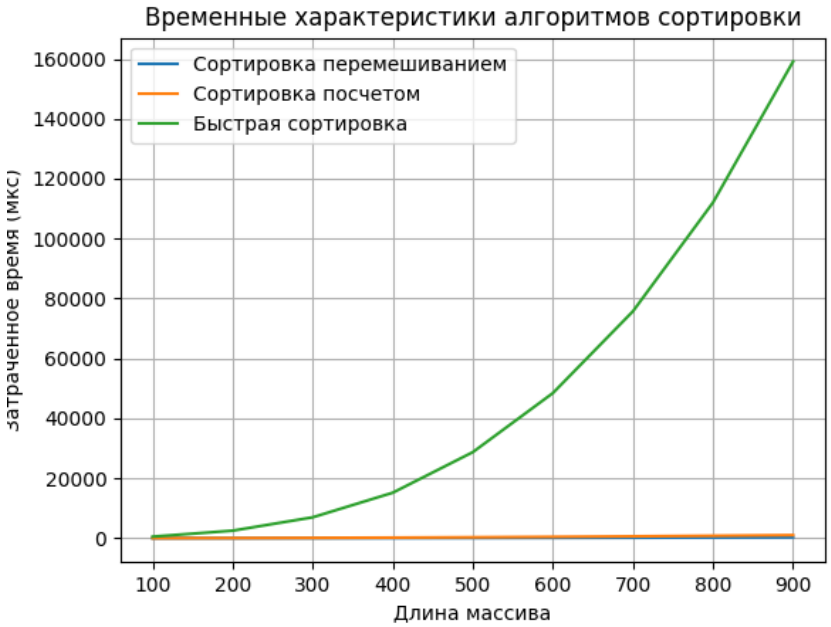

Рисунок 4.2 – Зависимость времени выполнения реализации алгоритма сортировки от размера отсортированного массива (мкс)

Таблица 4.1 – Время выполнения реализации алгоритмов сортировки на отсортированных данных (мкс)

Размер	Подсчетом	Быстрая	Шейкер
100	23.145	545.444	6.838
200	65.845	2541.844	19.117
300	131.516	6997.099	38.015
400	221.289	15206.390	64.844
500	337.287	28752.306	99.571
600	477.355	48464.394	142.147
700	644.714	75756.450	192.762
800	836.860	112060.297	251.221
900	1054.386	159101.962	318.163

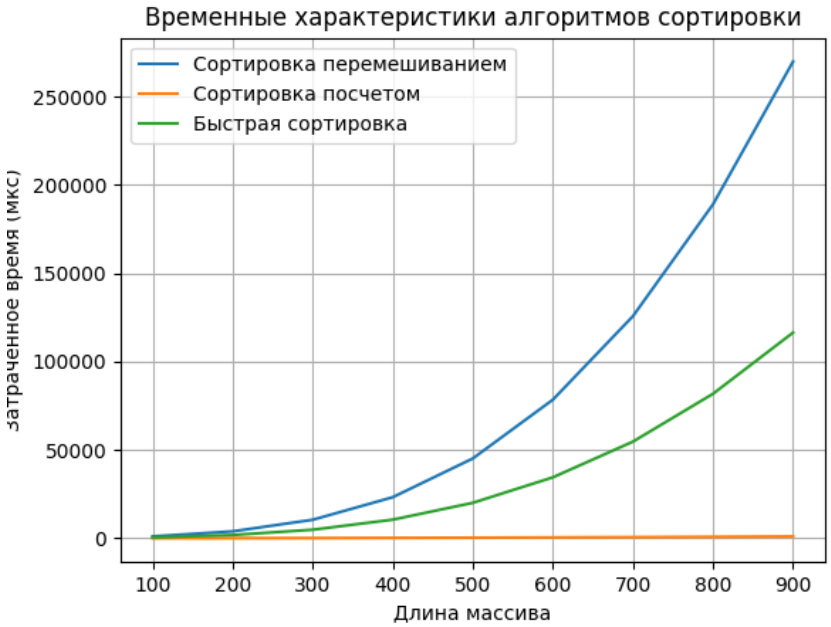

Рисунок 4.3 – Зависимость времени выполнения реализации алгоритма сортировки от размера массива, отсортированного в обратном порядке (мкс)

Таблица 4.2 – Время выполнения реализации алгоритмов сортировки на обратно отсортированных данных (мкс)

Размер	Подсчетом	Быстрая	Шейкер
100	33.426	530.626	1078.104
200	76.742	1868.477	3944.645
300	143.454	4870.720	10498.472
400	234.621	10557.311	23275.374
500	350.191	20097.290	45212.845
600	489.927	34519.663	78491.104
700	653.540	54741.962	125775.118
800	845.667	81818.746	189079.515
900	1062.585	116464.102	269914.922

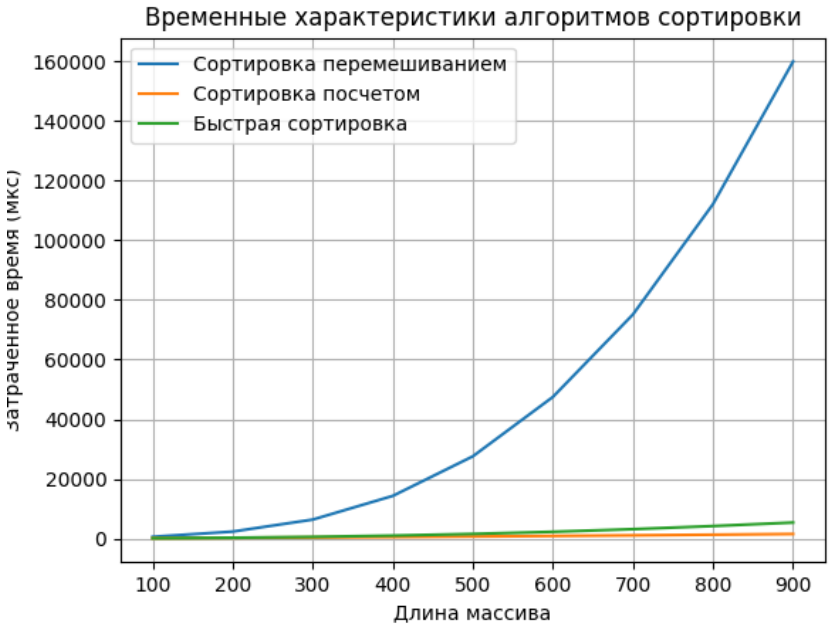

Рисунок 4.4 – Зависимость времени выполнения реализации алгоритма сортировки от размера массива, заполненного в случайном порядке (мкс)

Таблица 4.3 – Время выполнения реализации алгоритмов сортировки на случайных данных (мкс)

Размер	Подсчетом	Быстрая	Шейкер
100	129.406	102.688	588.341
200	243.097	277.647	2339.547
300	371.899	565.186	6360.275
400	521.326	988.754	14302.226
500	689.655	1555.392	27568.101
600	875.418	2270.616	47477.725
700	1077.845	3148.489	75123.545
800	1296.092	4189.439	112105.260
900	1530.593	5370.045	159892.601

Вывод

Реализация алгоритма сортировки посчетом работает быстрее остальных двух во всех трех случаях, что произошло благодаря небольшому диапазону значений генерируемых массивов — не более 1000. Это привело к

линейному характеру зависимости трудоемкости данного алгоритма от размера массива и линейному от мощность диапазона значений в массиве.

Заключение

В ходе выполнения лабораторной работы была достигнута цель работы: были разработаны алгоритмы сортировки массивов.

Все поставленные задачи решены:

- изучены и реализованы 3 алгоритма сортировки: перемешиванием, посчетом, быстрой;
- проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- проведен сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовлен отчет о лабораторной работе.

Список использованных источников

[1] Шейкерная сортировка (перемешиванием) [Электронный ресурс]. Режим доступа: https://kvodo.ru/shaker-sort.html (дата обращения: 18.10.2022).

[2] Сортировка вставками [Электронный ресурс]. Режим доступа: https://kvodo.ru/quicksort.html (дата обращения: 18.010.2022).

[3] Сортировка выбором [Электронный ресурс]. Режим доступа: https://kvodo.ru/sortirovka-vyiborom-2.html (дата обращения: 12.09.2021).

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 04.09.2021).

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 04.09.2021).

[6] Manjaro [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 03.10.2022).

[7] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 04.10.2022).

[8] Мобильный процессор AMD Ryzen™ 7 3700U [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-7-3700u (дата обращения: 04.10.2022).