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 определеннном 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 количеству элементов в последовательности, а поиск элемента в последовательности не отсортированных данных занимает время, пропорциональное воличеству 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 упорядочность 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: шейкер, вставками, выбором;

- 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, вставками и выбором.

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. Эксплуатируется следующая идея:

1. требуется отсортировать массив $Array$ из N чисел в диапазоне от 1 до m;
2. подсчитать, сколько раз встречается каждый элемент массива — для этого:
   (a) создать вспомогательный массив $Counts$ из m счетчиков, заполненный его нулями;
   (b) 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 оценить 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.

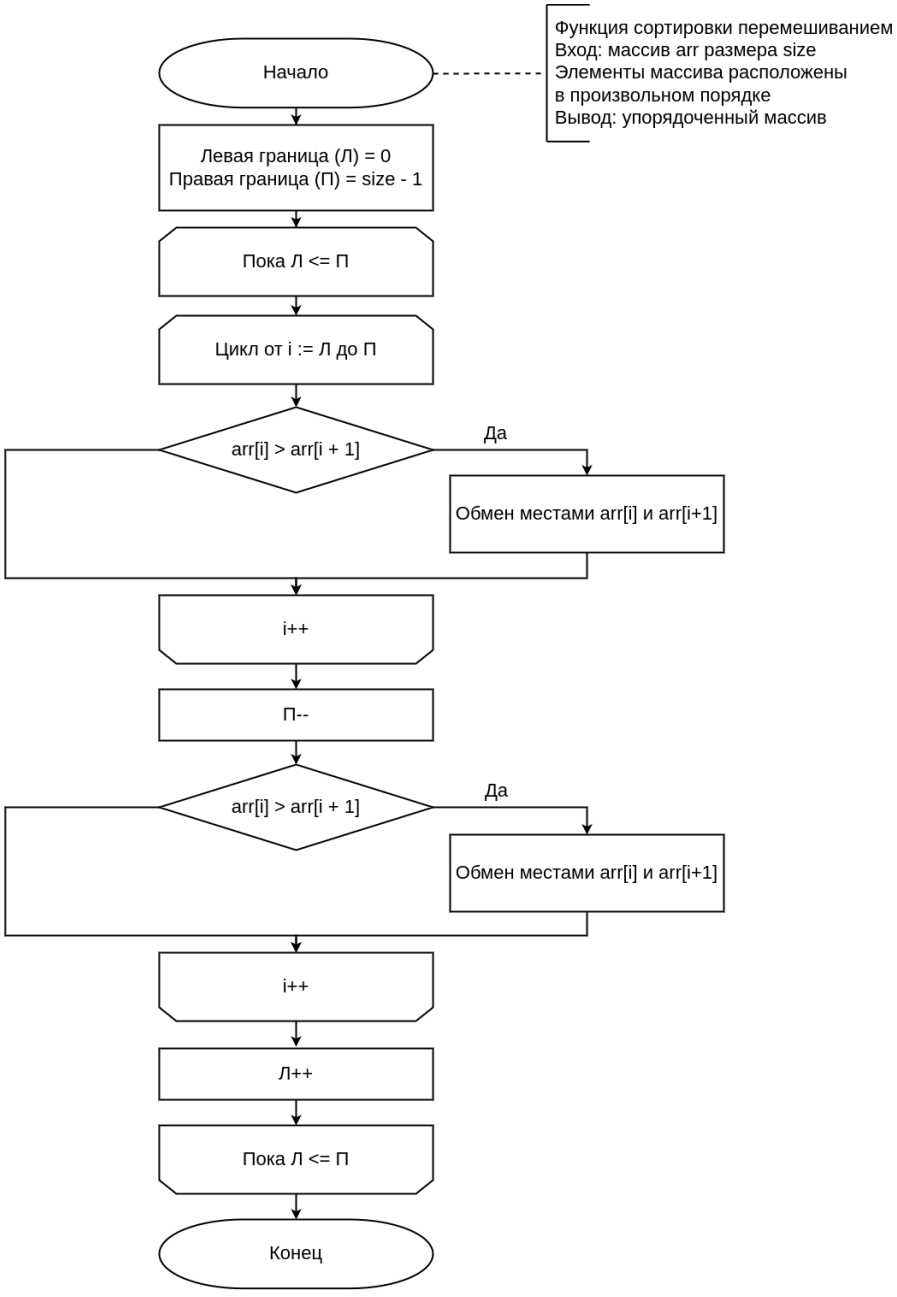

Рисунок 2.1 – Схема алгоритма сортировки перемешиванием

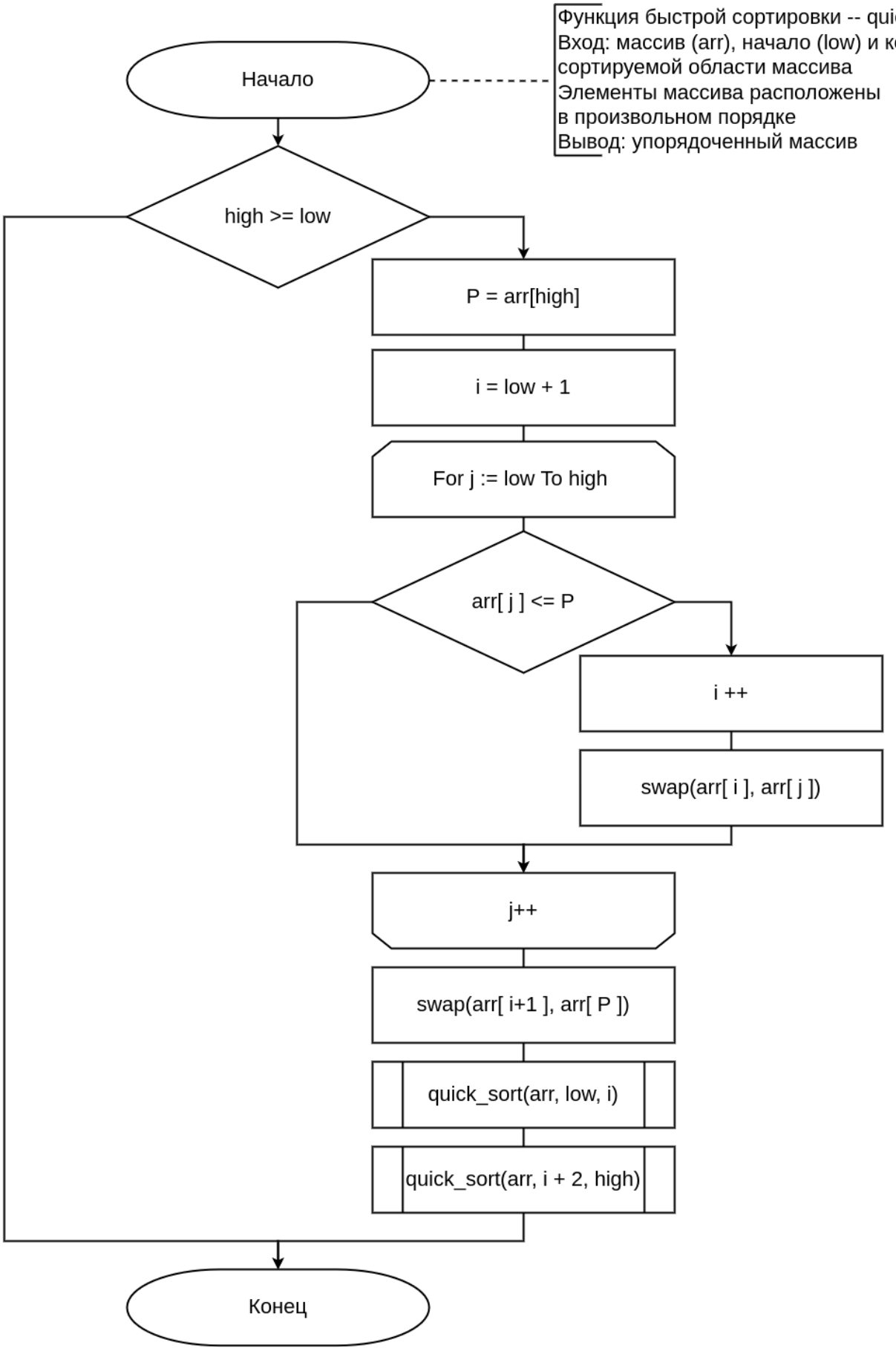

Рисунок 2.2 – Схема алгоритма быстрой сортировки

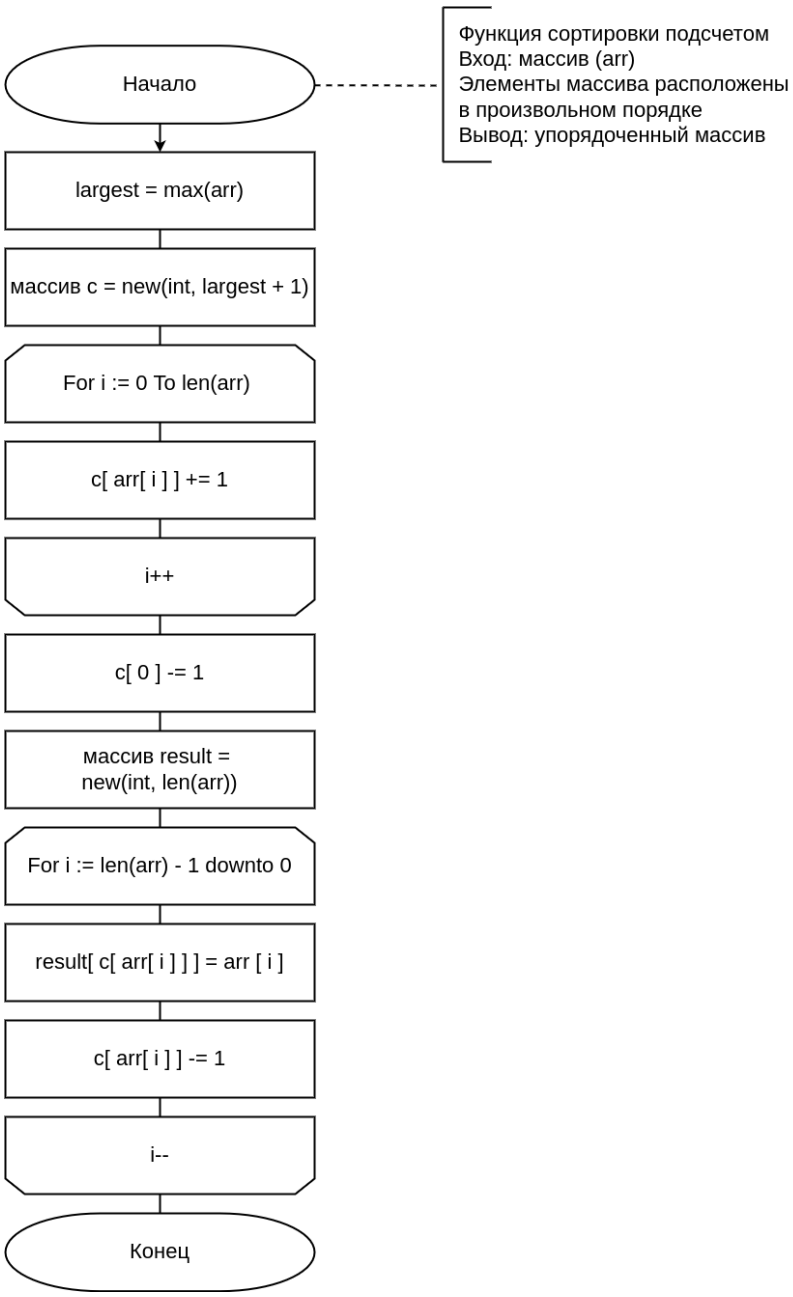

Рисунок 2.3 – Схема алгоритма сортировки подсчетом

2.2 Модель вычислений (оценки трудоемкости)

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

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

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

2.3 Трудоёмкость алгоритм

Обозначим во всех последующих вычислениях размер массивов как N.

2.3.1 Алгоритм сортировки перемешиванием

- Трудоёмкость сравнения внешнего цикла $WHILE(swap == True)$, которая равна (2.4):

$$f_{outer} = 1 + 2 \cdot (N - 1) \quad (2.4)$$

- Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - 1]$, которая равна (2.5):

$$f_{inner} = 5(N - 1) + \frac{2 \cdot (N - 1)}{2} \cdot (3 + f_{if}) \quad (2.5)$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.6}$$

Трудоёмкость в лучшем случае (2.7):

$$f_{best} = -3 + \frac{3}{2}N + \approx \frac{3}{2}N = O(N) \tag{2.7}$$

Трудоёмкость в худшем случае (2.8):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2) \tag{2.8}$$

2.3.2 Алгоритм быстрой сортировки

Трудоёмкость алгоритма быстрой сортировки состоит из:

- Трудоёмкость вычисления опорного элемента разбиения массива на 2 части (2.9):
  Трудоемкость цикла сравнения, инкремента внешнего цикла, а также зависимых только от него операций, по $i \in [1..m)$, где $m \in N/2^k$, где $k \in [0..f)$, где f равна максимальной глубине стека рекурсии:

$$f_{part} = 2 + 12(m - 1) \tag{2.9}$$

- 2 рекурсивных вызова, таким образом максимальная глубина вызова стека будет равна $log_2(N)$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.10):

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ 8, & \text{х.с.} \end{cases} \tag{2.10}$$

Трудоёмкость в лучшем случае (2.11):

$$f_{best} = (2 + 12(N) * log_2(N)) \approx \frac{1}{2} * N * log_2(N) = O(N * log_2(N)) \quad (2.11)$$

Трудоёмкость в худшем случае (2.12):

$$f_{worst} = (2 + 6(N) * log_2(N)) \approx \frac{6}{*} N * log_2(N) = O(N * log_2(N)) \quad (2.12)$$

2.3.3 Алгоритм сортировки подсчетом

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

- первого цикла, трудоемкость которого равна (2.13):

$$f_{first} = 2 + 6 * N \quad (2.13)$$

- второго цикла, трудоемкость которого равна (2.14):

$$f_{second} = 2 + 6 * M \quad (2.14)$$

- третьего цикла, трудоемкость которого равна (2.15):

$$f_{third} = 2 + 8 * N \quad (2.15)$$

Трудоемкость алгоритма сортировки подсчетом зависит от двух параметров: размера массива и диапазона значений. Таким образом, при $N >> M$ трудоемкость будет равна (2.16):

$$f_1 = 4 + 14 * N \approx 14 * N = O(N) \quad (2.16)$$

Таким образом, при $M >> N$ трудоемкость будет равна (2.17):

$$f_1 = 2 + 6 * M \approx 6 * M = O(M) \quad (2.17)$$

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.

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

3.3 Сведения о модулях программы

Программа состоит из двух 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 Листинг кода

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

Листинг 3.1 – Алгоритм 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 – Алгоритм 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 – Алгоритм 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 разработаны схемы всех трех алгоритмов 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 @ 2.3Гц [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.

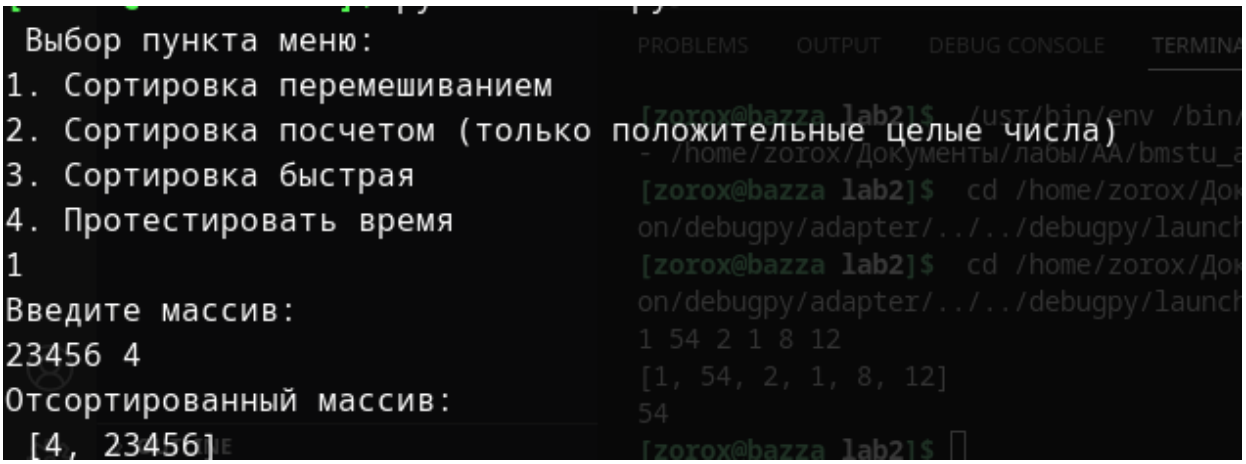

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Алгоритмы тестировались 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 определна, 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.

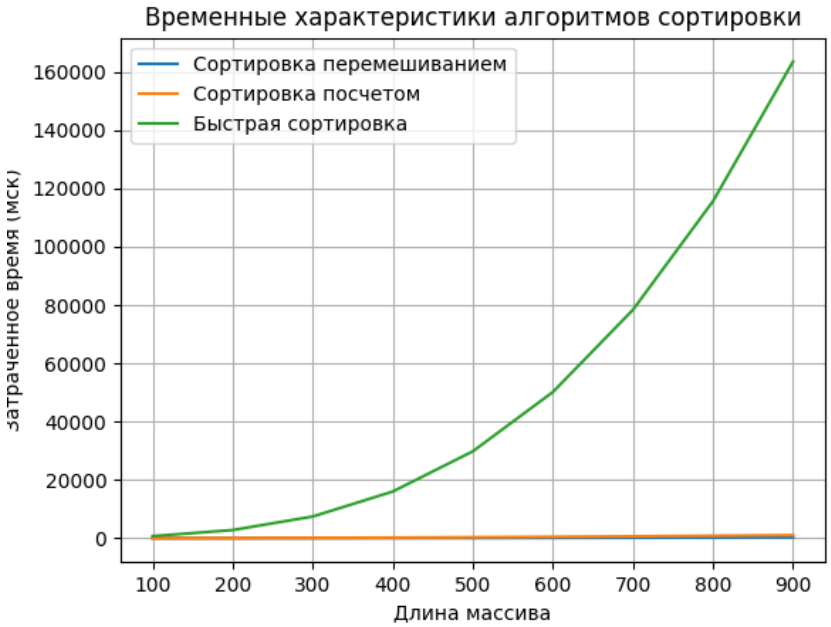

Рисунок 4.2 – Зависимость времени работы алгоритма сортировки от размера отсортированного массива (мск)

Таблица 4.1 – Время работы алгоритмов сортировки на отсортированных данных (мск)

Размер	Посчетом	Быстрая	Шейкер
100	32.256	739.074	9.280
200	77.017	2824.264	21.749
300	144.730	7463.536	40.958
400	238.323	16039.033	68.063
500	357.503	29838.043	103.308
600	502.003	50185.162	146.657
700	671.318	78380.342	198.029
800	864.330	115650.530	257.407
900	1084.393	163551.843	325.160

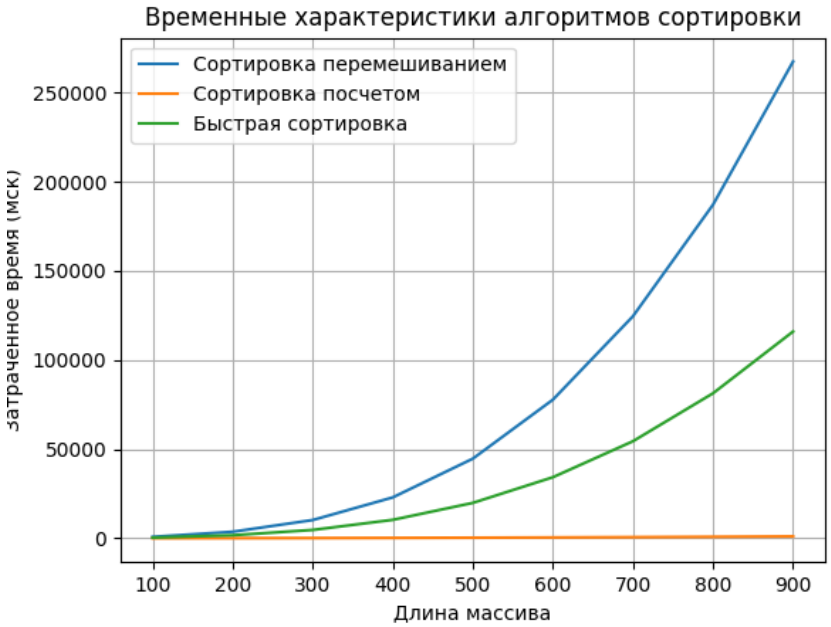

Рисунок 4.3 – Зависимость времени работы алгоритма сортировки от размера массива, отсортированного в обратном порядке (мск)

Таблица 4.2 – Время работы алгоритмов сортировки на обратно отсортированных данных (мск)

Размер	Посчетом	Быстрая	Шейкер
100	24.156	393.429	802.331
200	66.354	1703.255	3670.348
300	132.379	4689.759	10237.942
400	222.335	10350.337	22936.469
500	337.700	19849.474	44660.077
600	480.300	34246.585	77731.476
700	648.921	54421.607	124507.850
800	843.705	81308.332	187051.390
900	1066.708	115931.485	267315.687

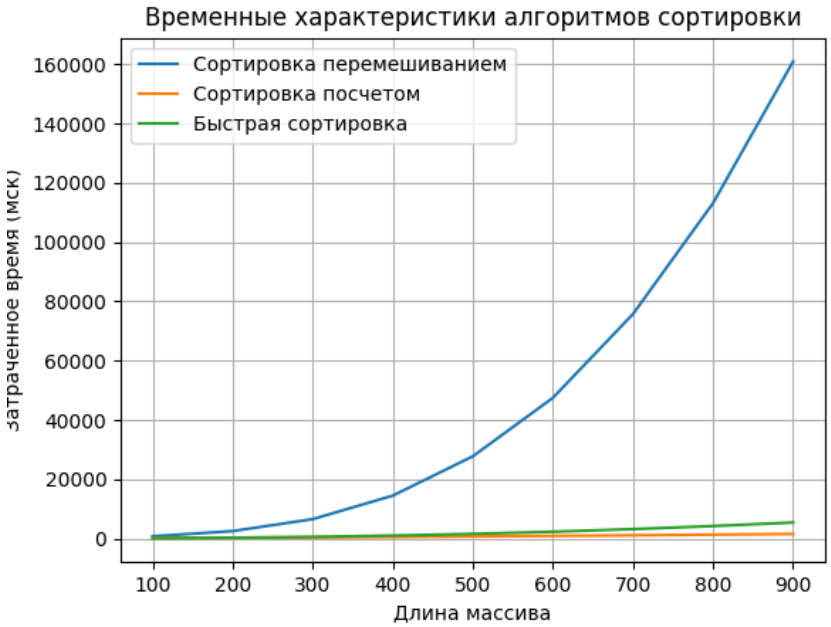

Рисунок 4.4 – Зависимость времени работы алгоритма сортировки от размера массива, заполненного в случайном порядке (мск)

Таблица 4.3 – Время работы алгоритмов сортировки на случайных данных (мск)

Размер	Посчетом	Быстрая	Шейкер
100	175.389	143.153	803.232
200	288.235	315.064	2560.598
300	418.869	601.610	6584.375
400	571.802	1034.448	14517.472
500	741.793	1611.465	27777.835
600	929.885	2341.030	47499.079
700	1133.956	3225.462	75728.871
800	1353.892	4257.337	113124.555
900	1591.011	5451.507	160881.977

Вывод

Алгоритм сортировки посчетом работает лучше остальных двух во всех трех случаях, что произошло благодаря небольшому диапазону значений генерируемых массивов – не более 1000. Это привело к линейной трудоемкости данного алгоритма.

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.

Поставленная цель достигнута.

Литература

[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).