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»

на тему: «Трудоемкость сортировок»

Студент ИУ7-56Б _____ Вольняга М. Ю.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова Л. Л.
(Подпись, дата) (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сортировка — это процесс упорядочивания элементов набора данных или последовательности в определенном порядке, обычно в возрастающем или убывающем порядке, в соответствии с заданным критерием или правилами. Этот процесс делает данные более удобными для анализа, поиска и обработки [1].

Цель данной лабораторной работы — исследование следующих алгоритмов сортировки:

- блочная;
- бусинами;
- выбором;

Для достижения поставленной цели необходимо выполнить следующие задачи:

- разработать требуемые алгоритмы;
- оценить трудоемкость рассматриваемых алгоритмов;
- провести сравнительный анализ времени выполнения реализуемых алгоритмов и занимаемой памяти.

1 Аналитический раздел

В данном разделе будут рассмотрены три алгоритма сортировок: блочная сортировка, сортировка выбором и сортировка бусинами.

1.1 Алгоритм блочной сортировки

Идея блочной сортировки заключается в разделении входной последовательности данных на несколько «блоков» одинакового размера, сортировке каждого блока, а затем объединении отсортированных блоков в одну отсортированную последовательность [2].

Алгоритм состоит из следующих шагов [2].

1. *Разделение массива.* Исходный массив разделяется на две равные (или почти равные) части. Это может быть сделано путем выбора опорного элемента (например, среднего элемента массива) и разделения массива на элементы, которые меньше опорного, и элементы, которые больше опорного.

2. *Сортировка каждой части.* Каждая из получившихся частей массива сортируется отдельно с использованием того же алгоритма блочной сортировки.

3. *Слияние отсортированных частей.* После того как обе части массива стали отсортированными, они объединяются в один отсортированный массив. Это делается путем сравнения элементов из обеих частей и добавления их в правильном порядке в результирующий массив. Этот процесс продолжается, пока не будут обработаны все элементы из обоих частей.

1.2 Алгоритм сортировки бусинами

Алгоритм сортировки бусинами, известный также как гравитационная сортировка, представляет собой метод, использующий физические свойства, а именно гравитацию, для упорядочивания списка положительных целых чисел [3].

Принцип сортировки бусинами основывается на эффекте гравитационного падения бусин. Визуализация процесса предполагает использование

устройства, подобного счету, где каждый горизонтальный ряд бусин соответствует отдельному числу, а каждая бусина символизирует одну единицу этого числа [3].

Алгоритм состоит из следующих шагов [3].

1. *Инициализация.* Для начала, каждое число в массиве представляется в виде горизонтального ряда бусин, где каждая бусина эквивалентна одной единице числа.

2. *Сортировка.* Под воздействием гравитации бусины падают, при этом бусины, представляющие более большие числа, скапливаются над бусинами меньших чисел. Процесс приводит к «смещению» бусин влево, формируя отсортированные столбцы.

3. *Результат.* По завершении процесса подсчитывается количество бусин в каждом столбце, что и представляет собой отсортированный массив чисел.

Алгоритм применим к натуральным числам. Можно сортировать и целые, но отрицательные числа придется обрабатывать отдельно от положительных.

1.3 Алгоритм сортировки выбором

Алгоритм сортировки выбором основан на поиске минимального (или максимального) значения на необработанном срезе массива или списка и последующем обмене этого значения с первым элементом необработанного среза. После каждого такого обмена размер необработанного среза уменьшается на один элемент [4].

Алгоритм состоит из следующих шагов [4].

1. Находим минимальный (или максимальный) элемент в текущем массиве.

2. Производим обмен этого элемента со значением первой неотсортированной позиции. Обмен не требуется, если минимальный (или максимальный) элемент уже находится на данной позиции.

3. Сортируем оставшуюся часть массива, исключая из рассмотрения уже отсортированные элементы.

2 Конструкторский раздел

В этом разделе представлены: требования к программному обеспечению, описание используемых типов данных, оценка трудоемкости и схемы реализуемых алгоритмов.

2.1 Требования к программному обеспечению

Программа должна поддерживать два режима работы: режим массового замера времени и режим сортировки введенного массива.

Режим массового замера времени должен обладать следующей функциональностью:

- генерировать массивы различного размер для проведения замеров;
- осуществлять массовый замер, используя сгенерированные данные;
- результаты массового замера должны быть представлены в виде таблицы и графика.

К режиму сортировки выдвигается следующий ряд требований:

- возможность работать с массивами разного размера, которые вводит пользователь;
- наличие интерфейса для выбора действий;
- на выходе программы массив, отсортированный тремя алгоритмами по возрастанию.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

- целое число представляет количество элементов в массиве;
- список целых чисел;

2.3 Разработка алгоритмов

Алгоритмы принимают на вход массив arr. На выходе у алгоритмов отсортированный массив arr. На рисунках 2.1 — 2.4 приведены схемы для реализуемых алгоритмов.

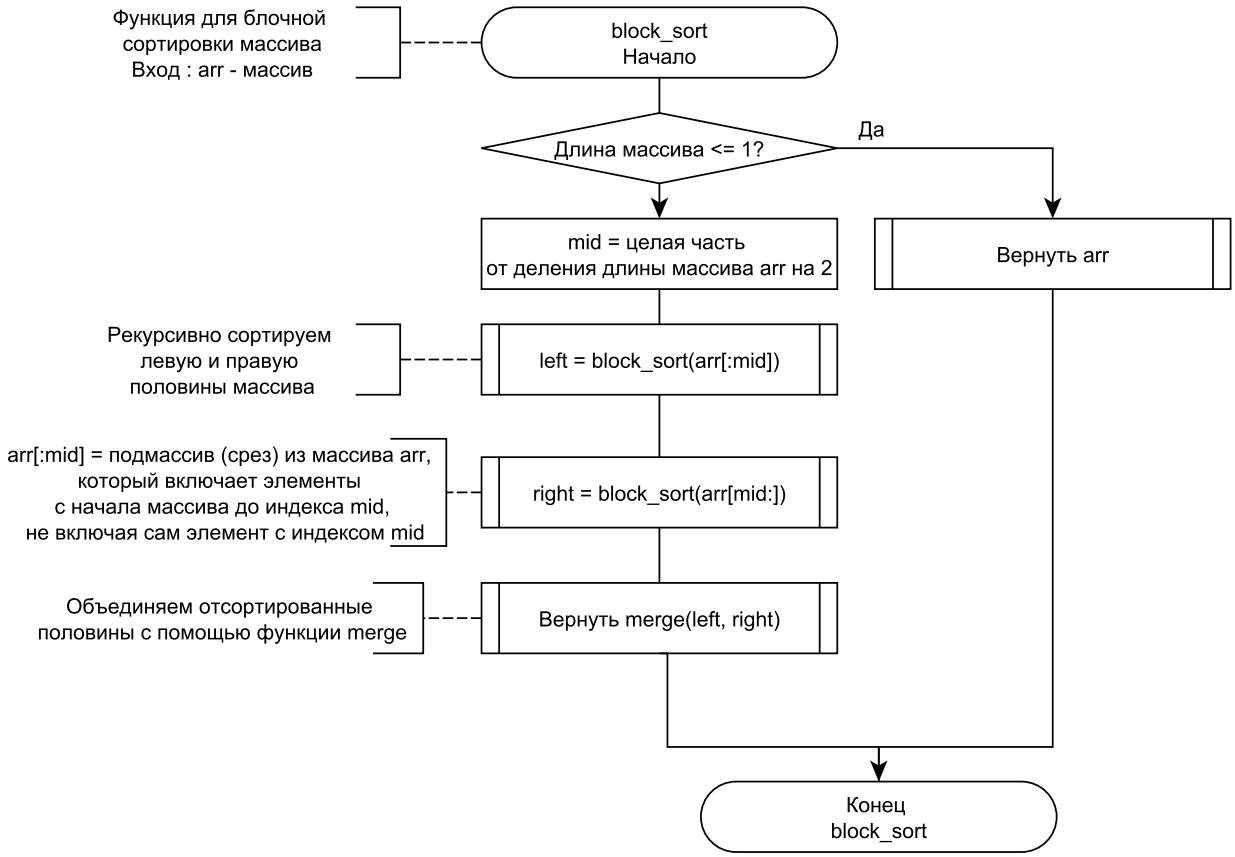

Рисунок 2.1 – Схема алгоритма блочной сортировки

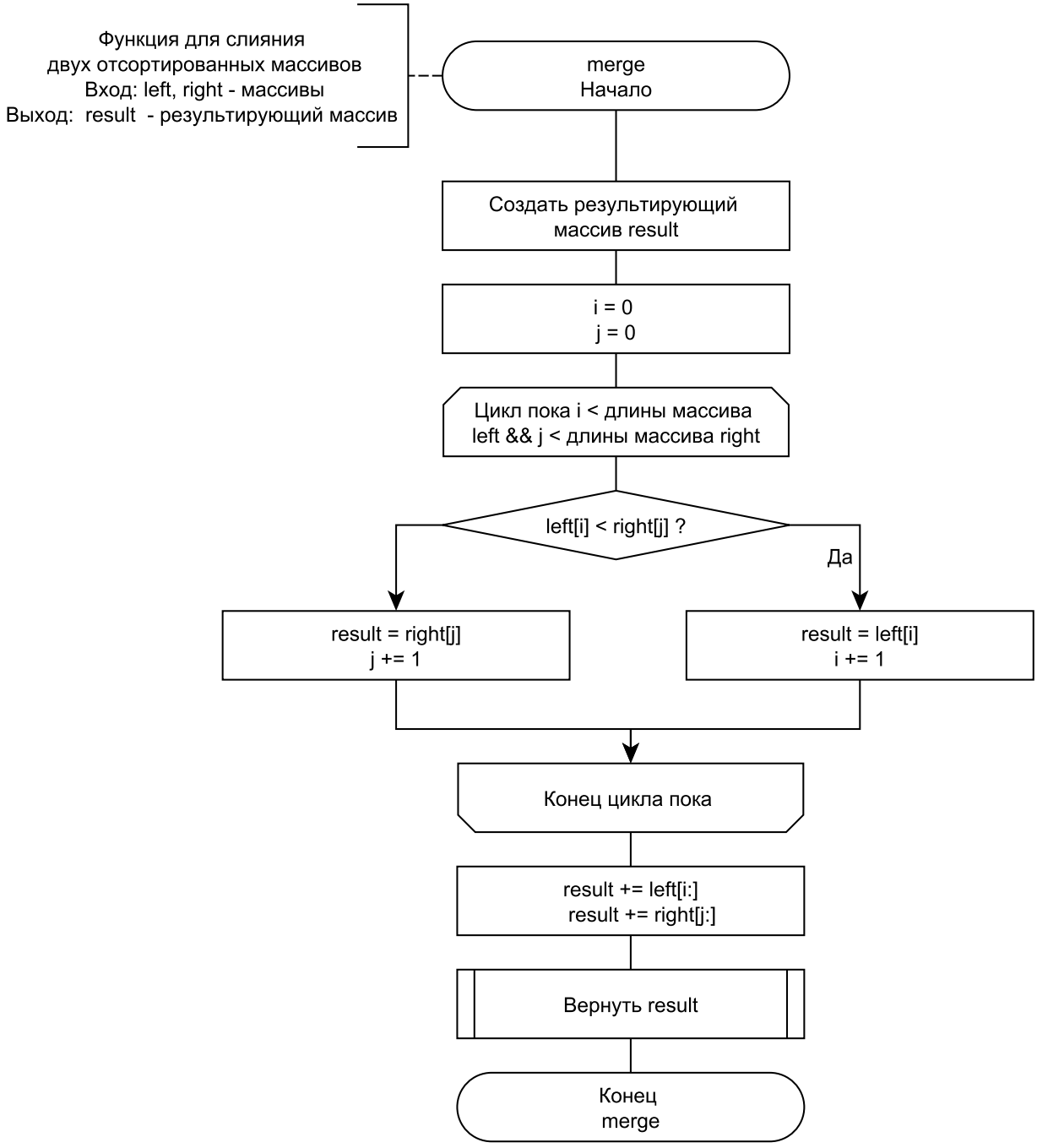

Рисунок 2.2 – Схема вспомогательной функции merge для алгоритма блочной сортировки

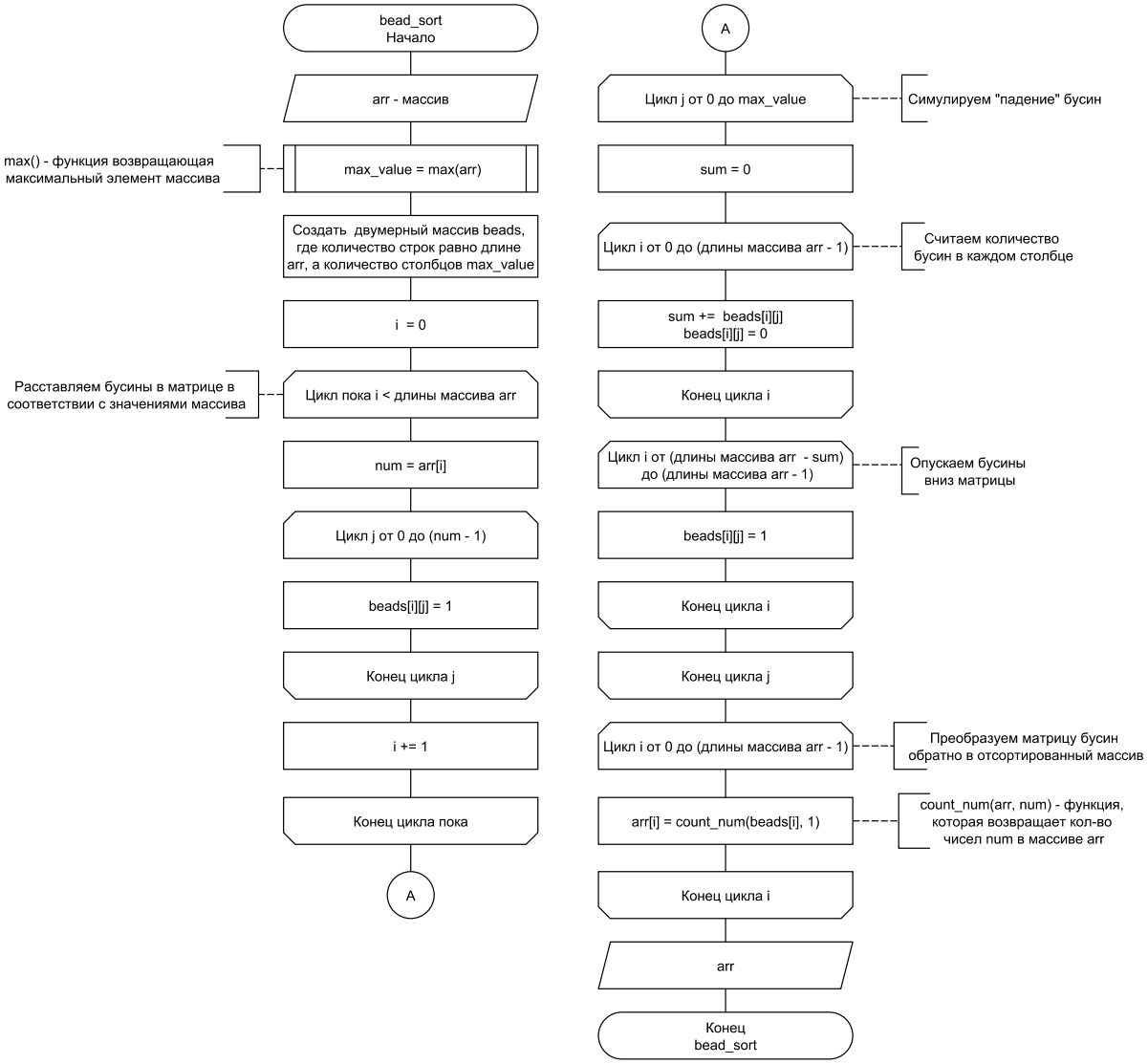

Рисунок 2.3 – Схема алгоритма сортировки бусинами

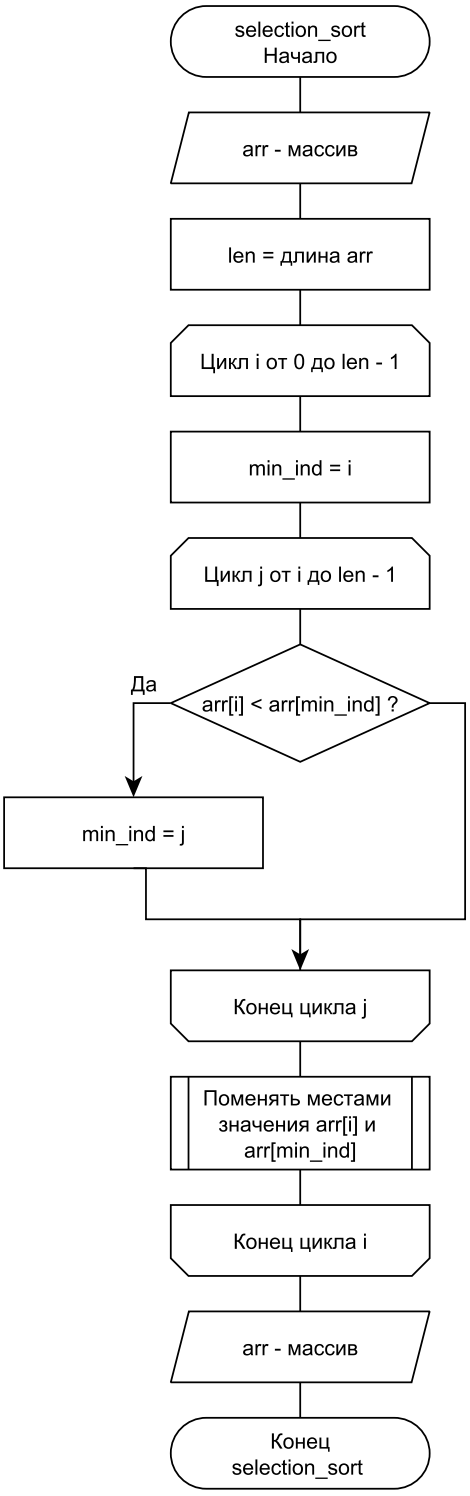

Рисунок 2.4 – Схема алгоритма сортировки выбором

2.4 Оценка трудоемкости алгоритмов

Модель для оценки трудоемкости алгоритмов состоит из шести пунктов:

1. $+, -, =, + =, - =, ==, ||, \&\&, <, >, <=, >=, <<, >>, []$ — считается, что эти операции обладают трудоемкостью в 1 единицу;

2. $*, /, * =, / =, \%$ — считается, что эти операции обладают трудоемкостью в 2 единицы;

3. трудоемкость условного перехода принимается за 0;

4. трудоемкость условного оператора рассчитывается по формуле (2.1),

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай} \\ max(f_1, f_2), & \text{худший случай} \end{cases}, \tag{2.1}$$

   где f_1 — трудоемкость блока, который вычисляется при выполнении условия, а f_2 — трудоемкость блока, который вычисляется при невыполнении условия;

5. трудоемкость цикла рассчитывается по формуле (2.2),

$$\begin{gathered} f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}); \end{gathered} \tag{2.2}$$

6. вызов подпрограмм и передача параметров принимается за 0.

2.4.1 Трудоемкость алгоритма блочной сортировки

В данной реализации размер блока обозначается как k, трудоемкость операции добавления и удаления элемента из вектора равна 2.

Лучший случай: массив отсортирован, элементы распределены равномерно (все блоки содержат одинаковое число элементов), расчет трудоемкости данного случая приведен в (2.3).

$$
\begin{aligned}
f_{best} = 1 + 1 + \frac{n}{k} \cdot (1 + 2 + f_{shaker} + 2 + 1 + 4 + \\
+ k \cdot (3 + 1 + 4)) + 1 + 1 + \\
+ \frac{n}{k} \cdot (1 + 4 + 1 + 1 + 5 + 1 + 4 + 1 + 1 + n \cdot (5)) = \\
= 4 + \frac{29 \cdot n + n \cdot f_{shaker} + 5 \cdot n^2}{k} + 8 \cdot n = \\
= 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + n \cdot (14.5 + \frac{k}{2}) + \frac{5 \cdot n^2}{k}.
\end{aligned}
\tag{2.3}
$$

Худший случай: большое количество пустых блоков, массив отсортирован в обратном порядке (худший случай сортировки перемешиванием, которая используется в блочной сортировке), расчет трудоемкости приведен в выражении (2.4).

$$
\begin{aligned}
f_{worst} = 1 + 1 + \frac{n}{k} \cdot (1 + 2 + f_{shaker} + 2 + 1 + 4 + \\
+ k \cdot (3 + 1 + 4)) + 1 + 1 + \\
+ \frac{n}{k} \cdot (1 + 4 + 1 + 1 + 5 + 1 + 4 + 1 + 1 + k \cdot (6)) = \\
= 4 + \frac{29 \cdot n + n \cdot f_{shaker} + 6 \cdot n^2}{k} + 8 \cdot n = \\
= 4 + 8 \cdot n + 29 \cdot \frac{n}{k} + n \cdot (19.5 + \frac{k}{2}) + \frac{6 \cdot n^2}{k}.
\end{aligned}
\tag{2.4}
$$

Вывод о трудоемкости блочной сортировки

Данная реализация зависит от выбранного размера блока, а также от количества сортируемых элементов. В соответствии с выражениями (2.4, 2.3) операнды $\frac{n \cdot k}{2}$, $\frac{n^2}{k}$ и $\frac{n}{k}$ зависят от значений n и k, первый приведенный операнд возрастает при увеличении k, другие убывают.

2.4.2 Трудоемкость алгоритма сортировки выбором

Трудоемкость сортировки выбором в худшем случае $O(N^2)$
Трудоемкость сортировки выбором в лучшем случае $O(N^2)$

2.4.3 Трудоемкость алгоритма сортировки бусинами

Трудоемкость в лучшем случае при отсортированном массиве и худшем случае при неотсортированном массиве в обратном порядке. Выведена

формуле (2.5).

$$
\begin{aligned}
f_{best} = 3 + 4 + (N-1) \cdot (2 + 2 + \begin{cases} 0, \\ 2, \end{cases}) + 1 + 2 + \\
+ N \cdot (2 + 3 + L \cdot (3 + 5)) + 2 + M \cdot (2 + 3 + N \cdot (2 + 5 + 5) + \\
+ 3 + (N - L) \cdot (2 + 5)) + 2 + N \cdot (2 + 8 + 8M) = \\
= S + 19NM + 11N + 8M + 18 = O(S)
\end{aligned}
\tag{2.5}
$$

В формуле (2.5) значения S или NL — сумма всех элементов в изначальном массиве, L — значение каждого элемента в массиве, M — максимальное значение из элементов в массиве.

Исходя из результата формулы (2.5) можно понять, что лучшим случаем для сортировки будет случай если все значения массива будут соответствовать минимальному значению, а худшим случаем если значение массива будут большие значения.

3 Технологический раздел

В данном разделе будут приведены средства реализации, листинги кода реализации алгоритмов.

3.1 Средства реализации

Для реализации данной работы был выбран язык *Python* [5]. Данный выбор обусловлен следующим:

– язык поддерживает все структуры данных, которые выбраны в результате проектирования;

– язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;

– язык позволяет замерять процессорное время с помощью модуля *time*.

Процессорное время было замерено с помощью функции *process_time()* из модуля *time* [6].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

– *main.py* — файл, содержащий функцию *main*;

– $functions.py$ — файл, содержащий вспомогательные функции (ввод массива с клавиатуры, рандомно, с файла и т.д.)

– *sorts.py* — файл, содержащий код реализаций всех алгоритмов сортировок;

– *tests.py* — файл, в котором содержатся функции для замера и вывода времени выполнения реализаций алгоритмов.

3.3 Реализация алгоритмов

В листинге 3.1 приведена реализация блочной сортировки. В листинге 3.2 приведена реализация сортировки бусинами. В листинге 3.3 приведена реализация сортировки выбором.

Листинг 3.1 – Реализация блочной сортировки

```
def block_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        mid = len(arr) // 2
        # Рекурсивно сортируем левую и правую половины массива
        left = block_sort(arr[:mid])
        right = block_sort(arr[mid:])
        # Объединяем отсортированные половины с помощью функции
           merge
        return merge(left, right)

# Функция для слияния двух отсортированных массивов
def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result += left[i:]
    result += right[j:]

    return result
```

Листинг 3.2 – Реализация сортировки бусинами

```
def bead_sort(arr):
    if arr and len(arr) > 0:
        max_value = max(arr)
        # Создаем матрицу бусин: каждая строка представляет
          число, а столбцы - бусины
        beads = [[0] * max_value for _ in range(len(arr))]

        # Расставляем бусины в матрице в соответствии с
          значениями массива
        i = 0
        while i < len(arr):
            num = arr[i]
            for j in range(num):
                beads[i][j] = 1
            i += 1
        # Симулируем "падение" бусин
        for j in range(max_value):
            sum = 0
            # Считаем количество бусин в каждом столбце
            for i in range(len(arr)):
                sum += beads[i][j]
                beads[i][j] = 0
            # Опускаем бусины вниз матрицы
            for i in range(len(arr) - sum, len(arr)):
                beads[i][j] = 1
        # Преобразуем матрицу бусин обратно в отсортированный
          массив
        for i in range(len(arr)):
            arr[i] = beads[i].count(1)
    return arr
```

Листинг 3.3 – Реализация сортировки выбором

```
def selection_sort(arr):
    size = len(arr)

    for ind in range(size):
        min_ind = ind

        for j in range(ind + 1, size):
            if arr[j] < arr[min_ind]:
                min_ind = j
        arr[ind], arr[min_ind] = arr[min_ind], arr[ind]

    return arr
```

4 Исследовательская часть

В данном разделе будут приведены: технические характеристики устройства, демонстрация работы программы, сравнительный анализ времени выполнения реализуемых алгоритмов и занимаемой памяти.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

– Процессор: Ryzen 5 4600H, тактовая частота ЦПУ 3.0 ГГц, максимальная частота процессора 4.0 ГГц [7].

– Оперативная память: 16 ГБайт.

– Операционная система: Windows 10 Pro 64-разрядная система [8].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлено визуальное представление работы разработанного программного обеспечения. В частности, продемонстрированы результаты сортировки для всех реализаций алгоритмов, при входном массиве $[15, 7, 9, 30, 8, 9, 9, 10, 45]$.

4.3 Время выполнения реализаций алгоритмов

Результаты замеров времени выполнения реализаций алгоритмов сортировок приведены в таблицах 4.1 – 4.3. Замеры времени проводились на массивах одного размера и усреднялись для каждого набора одинаковых экспериментов.

В таблицах 4.1 – 4.3 используются следующие обозначения:

– Блочная — реализация алгоритма блочной сортировки;

– Бусинами — реализация алгоритма сортировки бусинами;

– Выбором — реализация алгоритма сортировки выбором.

```
МЕНЮ:
1. Блочная сортировка
2. Сортировка бусинами
3. Сортировка выбором
4. Измерить время
0. Выход

Выберите пункт: 1
Введите элементы массива (в одну строку через пробел):
15 7 9 30 8 9 9 10 45

Результат: [7, 8, 9, 9, 9, 10, 15, 30, 45]

МЕНЮ:
1. Блочная сортировка
2. Сортировка бусинами
3. Сортировка выбором
4. Измерить время
0. Выход

Выберите пункт: 2
Введите элементы массива (в одну строку через пробел):
15 7 9 30 8 9 9 10 45

Результат: [7, 8, 9, 9, 9, 10, 15, 30, 45]

МЕНЮ:
1. Блочная сортировка
2. Сортировка бусинами
3. Сортировка выбором
4. Измерить время
0. Выход

Выберите пункт: 3
Введите элементы массива (в одну строку через пробел):
15 7 9 30 8 9 9 10 45

Результат: [7, 8, 9, 9, 9, 10, 15, 30, 45]
```

Рисунок 4.1 – Демонстрация работы программы

Таблица 4.1 – Время работы реализации алгоритмов на неотсортированных массивах (в мс)

Размер массива	Блочная	Бусинами	Выбором
100	0.063	6.453	7.547
200	0.172	14.469	13.969
300	0.328	22.875	22.547
400	0.531	30.734	31.562
500	0.781	39.281	39.641
600	1.094	51.594	50.156
700	1.406	60.031	59.406
800	1.797	71.359	69.984
900	2.250	79.203	80.453
1000	2.719	88.953	88.203

Таблица 4.2 – Время работы реализации алгоритмов на отсортированных в обратном порядке массивах (в мс)

Размер массива	Блочная	Бусинами	Выбором
100	0.063	7.250	7.094
200	0.172	14.062	14.578
300	0.328	22.313	22.375
400	0.516	31.656	30.828
500	0.781	39.031	40.203
600	1.078	46.188	49.141
700	1.422	59.031	58.266
800	1.797	69.828	71.109
900	2.234	79.031	79.344
1000	2.703	88.406	89.062

Таблица 4.3 – Время работы реализации алгоритмов на отсортированных массивах (в мс)

Размер массива	Блочная	Бусинами	Выбором
100	0.062500	7.250000	7.093750
200	0.171875	14.062500	14.578125
300	0.328125	22.312500	22.375000
400	0.515625	31.656250	30.828125
500	0.781250	39.031250	40.203125
600	1.078125	46.187500	49.140625
700	1.421875	59.031250	58.265625
800	1.796875	69.828125	71.109375
900	2.234375	79.031250	79.343750
1000	2.703125	88.406250	89.062500

На рисунках 4.2 – 4.4 изображены графики зависимостей времени выполнения реализаций сортировок от размеров массивов.

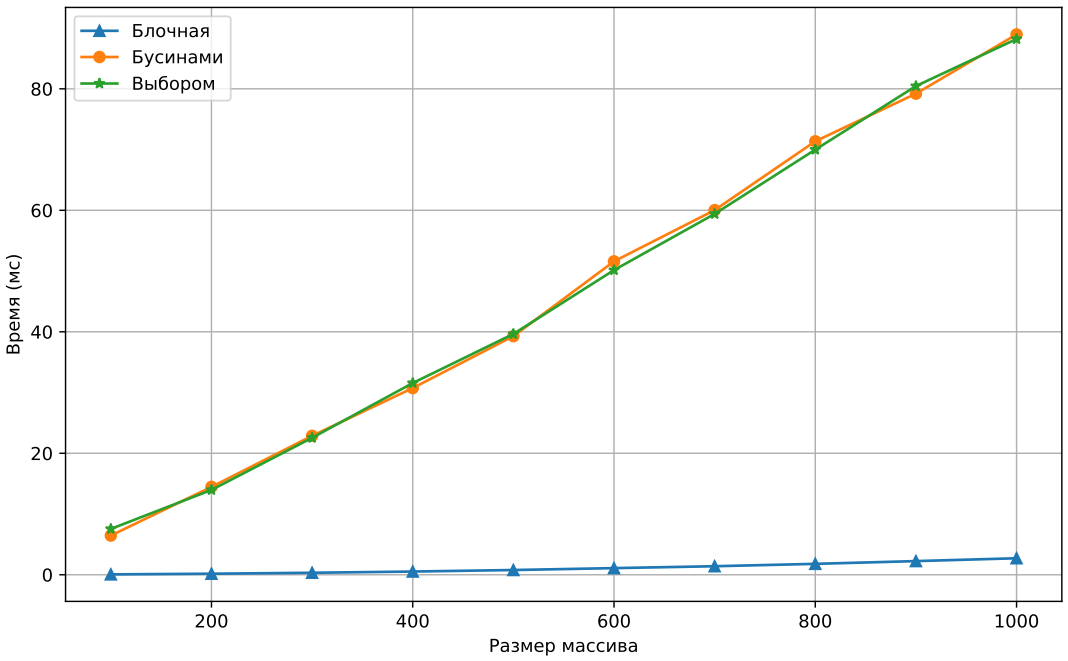

Рисунок 4.2 – Сравнение реализаций алгоритмов по времени выполнения на неотсортированных массивах

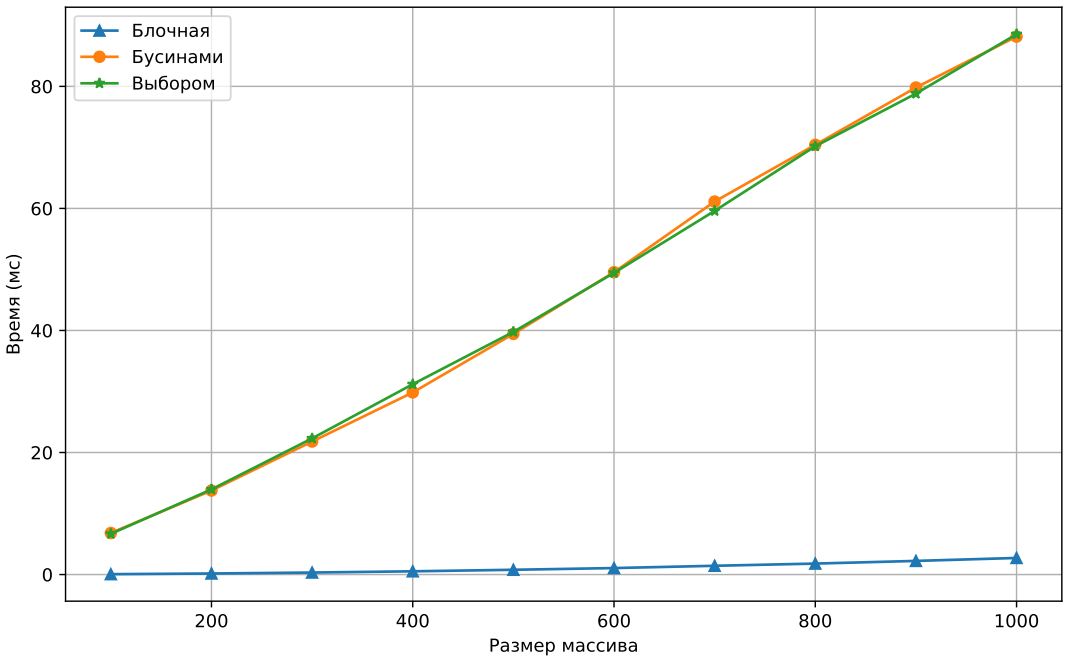

Рисунок 4.3 – Сравнение реализаций алгоритмов по времени выполнения на отсортированных в обратном порядке массивах

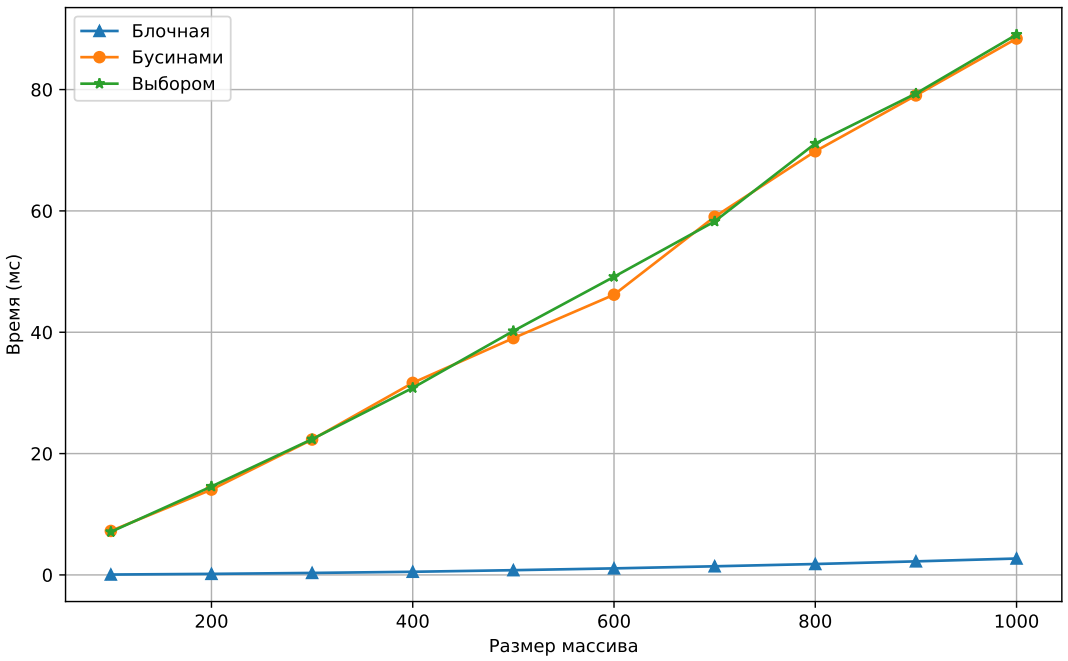

Рисунок 4.4 – Сравнение реализаций алгоритмов по времени выполнения на отсортированных массивах

4.4 Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы:

1. Алгоритм блочной сортировки демонстрирует значительно лучшую производительность на всех типах массивов, особенно выделяясь на неупорядоченных массивах (см. рисунки 4.2 – 4.4). его реализация значительно превосходит другие алгоритмы, что делает его предпочтительным выбором для разнообразных данных.

2. Алгоритм сортировки «Бусинами» показывает хорошие результаты на упорядоченных в обратном порядке и неупорядоченных массивах, но его производительность снижается на массивах, упорядоченных по возрастанию (см. рисунки 4.2 – 4.4). Это делает его подходящим для определенных случаев использования, особенно когда данные не предварительно отсортированы.

3. Реализация алгоритма сортировки выбором оказалась менее эффективной в сравнении с другими алгоритмами на всех типах массивов (см. рисунки 4.2 – 4.4).

ЗАКЛЮЧЕНИЕ

Цель работы достигнута, проведен сравнительный анализ следующих алгоритмов сортировки:

– блочная;

– бусинами;

– выбором;

В ходе выполнения лабораторной работы были решены все задачи:

– разработаны требуемые алгоритмы;

– оценена трудоемкость рассматриваемых алгоритмов;

– проведен сравнительный анализ времени выполнения реализуемых алгоритмов и занимаемой памяти.

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы:

1. Алгоритм блочной сортировки демонстрирует значительно лучшую производительность на всех типах массивов, особенно выделяясь на неупорядоченных массивах (см. рисунки 4.2 – 4.4). его реализация значительно превосходит другие алгоритмы, что делает его предпочтительным выбором для разнообразных данных.

2. Алгоритм сортировки «Бусинами» показывает хорошие результаты на упорядоченных в обратном порядке и неупорядоченных массивах, но его производительность снижается на массивах, упорядоченных по возрастанию (см. рисунки 4.2 – 4.4). Это делает его подходящим для определенных случаев использования, особенно когда данные не предварительно отсортированы.

3. Реализация алгоритма сортировки выбором оказалась менее эффективной в сравнении с другими алгоритмами на всех типах массивов (см. рисунки 4.2 – 4.4).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Шагбазян Д. В.*, *Штанюк А. А.*, *Малкина Е.В.* Алгоритмы сортировки. Анализ, реализация, применение: учебное пособие. — 2019.
2. Тема 3. Компьютерный анализ данных. Лекция 10. Методы и алгоритмы обработки и анализа данных. — Режим доступа: `http://imamod.ru/~polyakov/arc/stud/mmca/lecture_10.pdf` (дата обращения: 02.12.2023).
3. *Arulanandham J.*, *Calude C.*, *Dinneen M.* Bead–Sort: A Natural Sorting Algorithm // http://www.cs.auckland.ac.nz/staff-cgi-bin/mjd/secondcgi.pl?serial. — 2002. — Янв. — Т. 76.
4. *Кнут Д. Э.* Искусство программирования, том 3. Сортировка и поиск, 2-е изд. //. Т. 832. — Пер. с англ. М.: ООО 'И. Д. Вильямс', 2007.
5. The official home of the Python Programming Language. — Режим доступа: `https://www.python.org/` (дата обращения: 01.12.2023).
6. time — Time access and conversions. — Режим доступа: `https://docs.python.org/3/library/time.html` (дата обращения: 01.12.2023).
7. Ryzen 4600H. — Режим доступа: `https://www.amd.com/en/products/apu/amd-ryzen-5-4600h` (дата обращения: 20.09.2023).
8. Windows 10 Pro 2h21 64-bit. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows10` (дата обращения: 25.09.2022).