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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 3

по курсу «Анализ алгоритмов»

на тему: «Трудоемкость сортировок»

Студент ИУ7-55Б _____ И. Д. Половинкин
(Подпись, дата)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При работе с большими объемами данных, расположенных в хаотичном порядке относительно друг друга возникает проблема нахождения нужной записи по некоторому ключу. Например, нахождение в телефонной книге некоторого абонента, если при этом записи не находятся в алфавитном порядке по фамилии.

Для решения подобных задач существует такое решение как сортировка данных. Алгоритм сортировки необходим для упорядочивания элементов в списке. Сортировка может проводиться по какому-то критерию (ключу). Разработано множество алгоритмов, которые различаются по трудоемкости и эффективности в связи с затрачиваемыми ими ресурсами ЭВМ [1].

Целью лабораторной работы является изучение и реализация алгоритмов сортировок. Для её достижения необходимо выполнить следующие задачи:

- изучение алгоритмы сортировки;
- рассчитать трудоемкость каждого из выбранных алгоритмов;
- описать данные алгоритмы;
- выполнить тестирование методом черного ящика;
- провести сравнительный анализ этих алгоритмов по затратам памяти и процессорному выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

В разделе приводятся критерии выбора алгоритма сортировки, а также описание алгоритмов сортировок «блинная», «быстрая», «выбором».

1.1 Понятие сортировки

Сортировка — процесс упорядочения элементов в списке по возрастанию или убыванию. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

1.2 Критерии выбора алгоритма сортировки

К основным параметрам выбора необходимого алгоритма сортировки относят:

- временная сложность: описывает то, как производительность алгоритма изменяется в зависимости от размера набора данных;
- память: ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке не учитывается место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы;
- устойчивость: сортировка является устойчивой в том случае, если для любой пары элементов с одинаковым ключами, она не меняет их порядок в отсортированном списке (является важным критерием для баз данных).

1.3 Блинная сортировка

Блинная сортировка получила свое название от аналогии с переворачиванием блинов на сковороде [2]. Метод относится к классу сортировки обмена элементов путем их сравнения, как в пузырьковой. Сначала необходимо найти максимальный элемент в массиве. Далее переворачиваем (выполняем обмен) элементом от левого края до максимального – в результате максимум окажется на левом крае. На последнем шаге переворачивается весь неотсортированный

подмассив, тогда максимальный элемент попадет на свое место. Все эти действия повторяются с оставшейся, неотсортированной, частью массива.

1.4 Быстрая сортировка

В алгоритме быстрой сортировки может использоваться как рекурсивный, так и итеративный подход. Рассмотрим первый случай. Выбрав опорный элемент в списке, данный алгоритм сортировки делит массив на две части, относительно выбранного элемента [3]. Далее в первую часть попадают все элементы, меньшие выбранного, а во вторую — бóльшие. Если в данных частях более двух элементов, рекурсивно запускается для него та же процедура. В конце получится полностью отсортированная последовательность.

1.5 Сортировка выбором

Алгоритм сортировки выбором совершает несколько проходов по списку. При каждом проходе выбирается минимальный из еще не отсортированных элементов и обменивается с первым элементом не отсортированной области [4]. В следующем проходе рассмотренный элемент не участвует, сортируется только оставшийся хвост.

Для реализации устойчивости алгоритма необходимо минимальный элемент непосредственно вставлять в первую не отсортированную позицию, не меняя порядок остальных элементов.

Вывод

В данном разделе были рассмотрены основные теоретические сведения об алгоритмах сортировки «блинная», «быстрая», «выбором».

2 Конструкторский раздел

В разделе приводятся схемы алгоритмов сортировок «блинная», «быстрая», «выбором», а также трудоемкость каждого из них в лучшем и худшем случаях.

2.1 Схемы алгоритмов сортировки

2.1.1 Блинная сортировка

На рисунке 2.1 представлен алгоритм блинной сортировки.

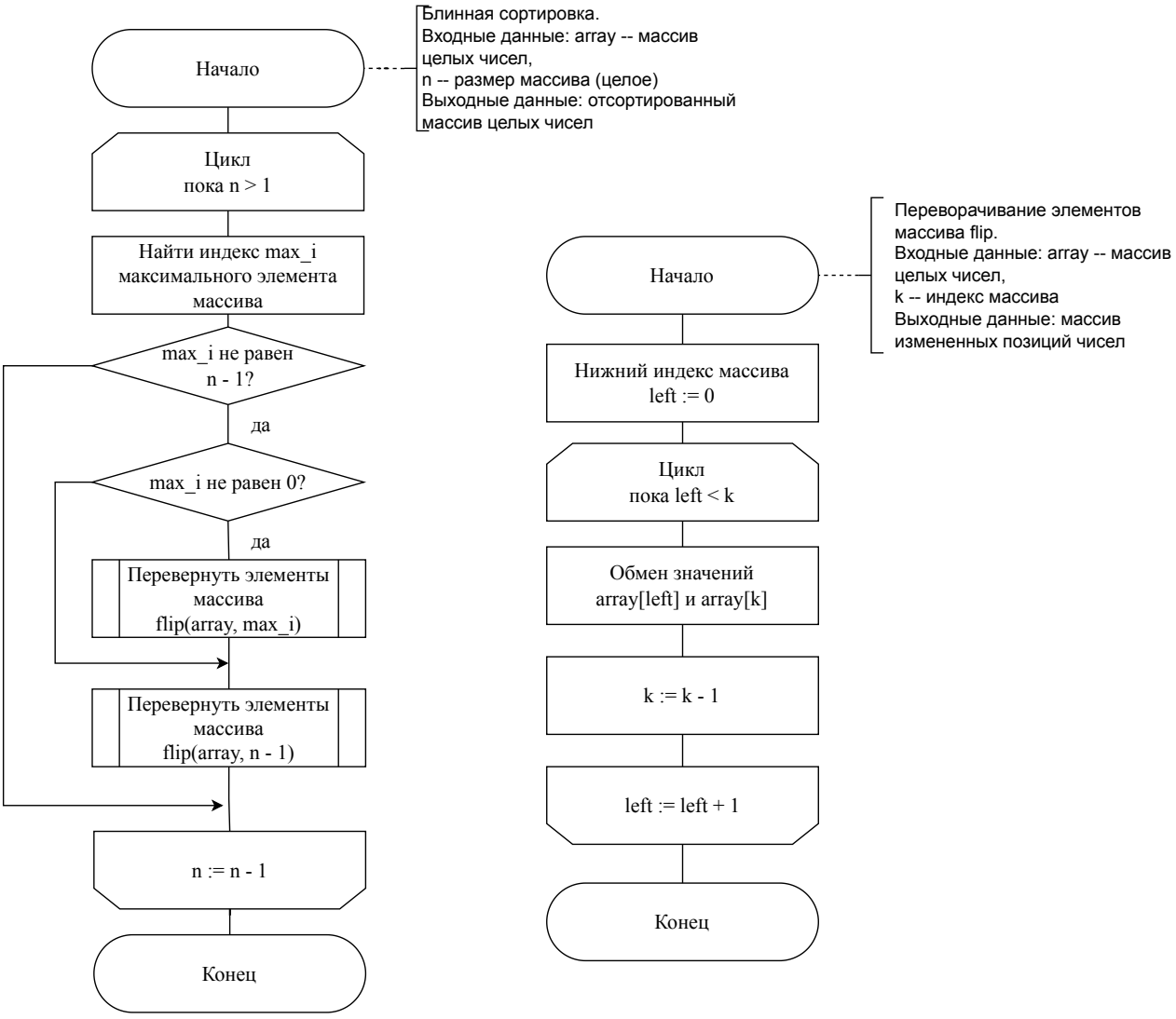

Рисунок 2.1 – Алгоритм блинной сортировки

2.1.2 Быстрая сортировка

На рисунке 2.2 представлен алгоритм быстрой сортировки.

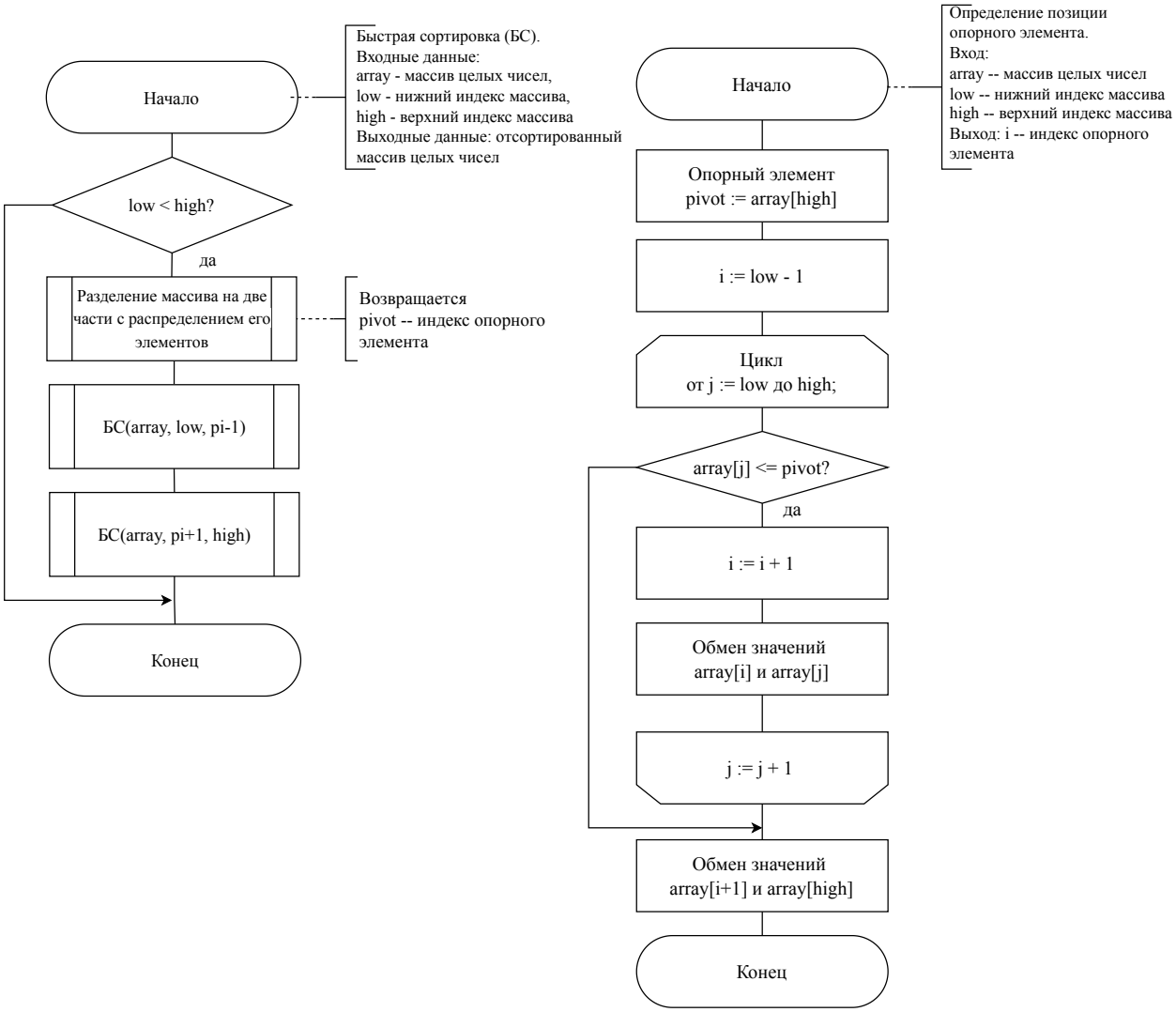

Рисунок 2.2 – Алгоритм быстрой сортировки

2.1.3 Сортировка выбором

На рисунке 2.3 представлен алгоритм сортировки выбором.

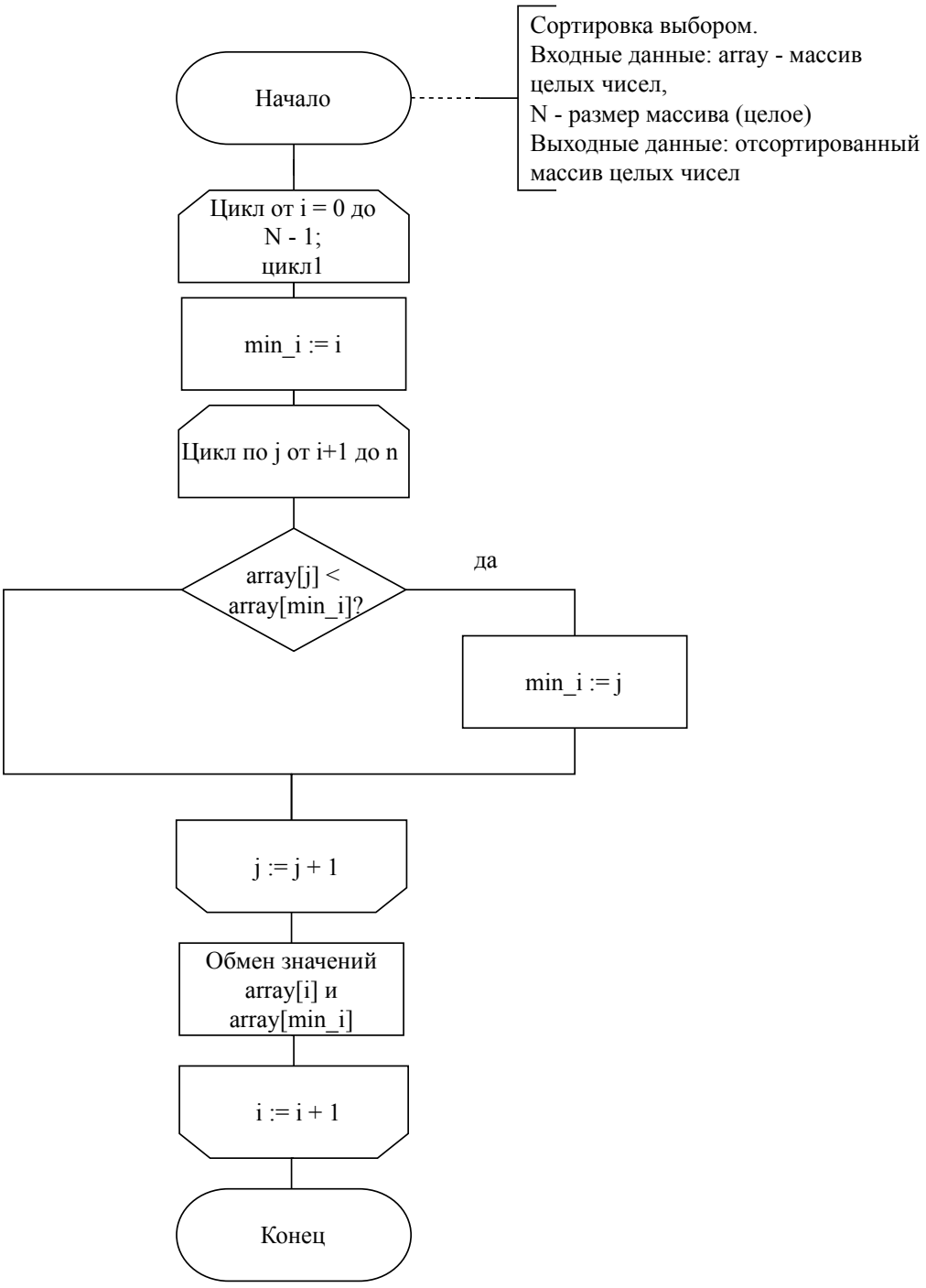

Рисунок 2.3 – Алгоритм сортировки выбором

2.2 Модель оценки трудоемкости алгоритмов

Введем модель оценки трудоемкости.

1. Трудоемкость базовых операций. Пусть трудоемкость следующих операций равна 2:

$$*, /, //, \%, * =, / = .$$

Примем трудоемкость следующих операций равной 1:

$$=, +, -, + =, - =, ==, ! =, <, >, <=, >=, |, \&\&, ||, [].$$

2. Трудоемкость цикла. Пусть трудоемкость цикла определяется по формуле (2.1):

$$f = f_{init} + f_{comp} + N_{iter} * (f_{in} + f_{inc} + f_{comp}), \tag{2.1}$$

где:

- f_{init}: трудоемкость инициализации переменной-счетчика;
- f_{comp}: трудоемкость сравнения;
- N_{iter}: номер выполняемой итерации;
- f_{in}: трудоемкость команд из тела цикла;
- f_{inc}: трудоемкость инкремента;
- f_{comp}: трудоемкость сравнения.

3. Трудоемкость условного оператора.
Пусть трудоемкость самого условного перехода равна 0, но она определяется по формуле (2.2):

$$f_{if} = f_{comp\_if} + \begin{cases} f_a \\ f_b \end{cases}. \tag{2.2}$$

2.3 Трудоемкость алгоритмов сортировки

2.3.1 Блинная сортировка

Трудоемкость блинной сортировки содержит следующие части:

– поиск максимального элемента до k-ой позиции (2.3):

$$f_{max} = \underset{=}{1} + k \cdot (\underset{[\,]}{1} + \underset{[\,]}{1} + \underset{>}{1} + \begin{cases} 0, \text{ лучший случай} \\ \underset{=}{1}, \text{ худший случай} \end{cases} + \underset{+=}{1}). \tag{2.3}$$

– трудоемкость операции обмена f_{swap} (2.4):

$$f_{swap} = \underset{=}{3} + \underset{[\,]}{4} = 7; \tag{2.4}$$

– переворачивание элементов f_{flip} (2.5):

$$f_{flip} = \underset{=}{1} + \begin{cases} \underset{=}{1}, & \text{лучший случай} \\ k \cdot (\underset{f_{swap}}{7} + \underset{-=}{1} + \underset{+=}{1}), & \text{худший случай} \end{cases}; \tag{2.5}$$

– тело сортировки f_{body} (2.6):

$$f_{body} = \underset{>}{1} + n \cdot (f_{max} + \underset{!=}{1} + \underset{-}{1} + \begin{cases} 0, \\ \underset{!=}{1} + \begin{cases} 0, \\ f_{flip}, \end{cases} \end{cases} + f_{flip} + \underset{-}{1} + \underset{-=}{1}). \tag{2.6}$$

Таким образом, в лучшем случае получим выражение f (2.7):

$$f = n \cdot (\underset{>}{1} + f_{max} + \underset{-}{1} + \underset{-=}{1}) \approx n. \tag{2.7}$$

Для худшего случая (отсортированный в обратном порядке массив) получим выражение f (2.8):

$$f = n \cdot (\underset{>}{1} + \underset{=}{1} + k \cdot (\underset{[\,]}{1} + \underset{[\,]}{1} + \underset{=}{1} + \underset{-}{1} + \underset{-=}{1}) + \underset{=}{1} + k \cdot (\underset{f_{swap}}{7} + \underset{-=}{1} + \underset{+=}{1})) \approx nk. \tag{2.8}$$

Для случая, когда $k = n$ получим (2.9):

$$f \approx n^2. \tag{2.9}$$

2.3.2 Быстрая сортировка

Быстрая сортировка может быть реализована как в рекурсивном варианте, так и в итеративном. Сложность для лучшего случая составляет $nlog(n)$, в худшем – n^2.

2.3.3 Выбором

Трудоемкость алгоритма сортировки выбором содержит следующие части:

– внешний цикл f_i от 1 до n (2.10):

$$f_i = \underset{=}{1}+\underset{-}{1}+\underset{<}{1}(n-1)\cdot(\underset{=}{1}+\underset{<}{1}+\underset{+}{1}+f_{swap}+f_{init\ j}) = 2+(n-1)\cdot(3+f_{swap}+f_{init\ j}); \tag{2.10}$$

– трудоемкость операции обмена f_{swap} (2.11):

$$f_{swap} = \underset{=}{3} + \underset{[\,]}{4} = 7; \tag{2.11}$$

– внутренний цикл f_j от $i+1$ до n (2.12):

$$f_j = \frac{1+n-1}{2} \cdot (n-1)(\underset{[\,]}{2} + \underset{=}{1} + f_{\text{усл}}) = \frac{n^2-n}{2} \cdot (3+f_{\text{усл}}); \tag{2.12}$$

– тело условия $f_{\text{усл}}$ (худший случай):

$$f_{\text{усл}} = 1. \tag{2.13}$$

Таким образом, для лучшего случая (отсортированный массив) получим выражение (2.14):

$$f = 2 + (n-1)\cdot(3+9+3) + \frac{n^2-n}{2}\cdot(3+0) = \frac{3}{2}n^2 + \frac{23}{2}n^2 - 11 \approx \frac{3}{2}n^2. \tag{2.14}$$

Для худшего случая (отсортированный в обратном порядке массив) получим выражение (2.15):

$$f = 2 + (n-1)\cdot(3+9+3) + \frac{n^2-n}{2}\cdot(3+1) = 2n^2 + 12n - 13 \approx 2n^2. \tag{2.15}$$

Вывод

На основе теоретических данных, полученных в аналитическом разделе, были построены схемы нужных алгоритмов. Определена трудоемкость каждого

из трех алгоритмов для анализа худшего и лучшего случаев асимптотической сложности, которая приведена в таблице 2.1:

Таблица 2.1 – Сложность алгоритмов.

Алгоритм	Сложность	
	Лучший случай	Худший случай
Блинная	$O(n)$	$O(n^2)$
Быстрая	$O(nlog_2(n))$	$O(n^2)$
Выбором	$O(n^2)$	$O(n^2)$

3 Технологический раздел

В разделе рассматриваются средства реализации, а также приводятся листинги алгоритмов сортировки «блинная», «быстрая», «выбором».

3.1 Средства реализации

В работе для реализации алгоритмов был выбран язык программирования Python [5]. В нем присутствуют библиотека time [6] для замера процессорного времени process_time(), а также для замера используемой памяти при помощи декоратора profiler [7].

3.2 Модули программы

3.2.1 Блинная сортировка

Реализация блинной сортировки приведена на листинге 3.1.

Листинг 3.1 – Блинная сортировка

```
def pancake_sort(arr, n):
    while n > 1:
        max_i = max_index(arr, n)
        if max_i != n - 1:
            if max_i != 0:
                flip(arr, max_i)
            flip(arr, n - 1)
        n -= 1
```

Блинная сортировка использует функцию переворачивания элементов flip() (листинг 3.2) и нахождение индекса первого максимального элемента (листинг 3.3).

Листинг 3.2 – Переворачивание элементов массива

```
def flip(arr, k: int):
    left = 0
    while left < k:
        arr[left], arr[k] = arr[k], arr[left]
        k -= 1
        left += 1
```

Листинг 3.3 – Нахождение первого максимального элемента

```
def max_index(arr, k: int) -> int:
    index = 0
    for i in range(k):
        if arr[i] > arr[index]:
            index = i
    return index
```

3.2.2 Быстрая сортировка

Реализация быстрой сортировки приведена на листинге 3.4.

Листинг 3.4 – Быстрая сортировка

```
def quick_sort(array, low, high):
    if low < high:
        pi = partition(array, low, high)

        quick_sort(array, low, pi - 1)
        quick_sort(array, pi + 1, high)
```

Быстрая сортировка использует функцию partition() разделения элементов массива на две части с упорядочиванием значений относительно опорного элемента (листинг 3.5).

Листинг 3.5 – Разделение элементов массива на две части с обменом

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1

    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            array[i], array[j] = array[j], array[i]

    array[i + 1], array[high] = array[high], array[i + 1]

    return i + 1
```

3.2.3 Сортировка выбором

Реализация сортировки выбором приведена на листинге 3.6.

Листинг 3.6 – Сортировка выбором

```
def select_sort(array, n):
```

```
    if n == 0:
        print("Ошибка")
        return array
    for i in range(n - 1):
        min_i = i
        for j in range(i + 1, n):
            if array[j] < array[min_i]:
                min_i = j
        array[i], array[min_i] = array[min_i], array[i]
    return array
```

3.3 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов:

Таблица 3.1 – Тесты

№	Описание теста	Вход	Результат		
			Блинная	Быстрая	Выбором
1	Один элемент	1	1	1	1
2	Отсортированный массив	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5
2	Отсортированный в обратном порядке	5,4,3,2,1	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5
2	Случайные числа	-8,3,1,6,-2	-8,-2,1,3,6	-8,-2,1,3,6	-8,-2,1,3,6

Вывод

В данном разделе был обоснован выбор языка программирования. Реализованы функции сортировки «блинная», «быстрая», «выбором», и проведено их тестирование методом черного ящика по таблице 3.1.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Windows 10 Pro;
- память: 8 GiB;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

4.2 Время выполнения

Проведем анализ времени работы алгоритмов для лучшего, обычного и худшего случаев. Исходными данными является массив целых чисел. Единичные замеры выдадут крайне малень- кий результат, поэтому проведем работу каждого алгоритма n = 100 раз и поделим на число n. Получим среднее значение работы каждого из алгоритмов.

Выполним анализ для лучшего случая, когда массив целых чисел упорядочен по возрастанию. Результат приведен на рисунке 4.1.

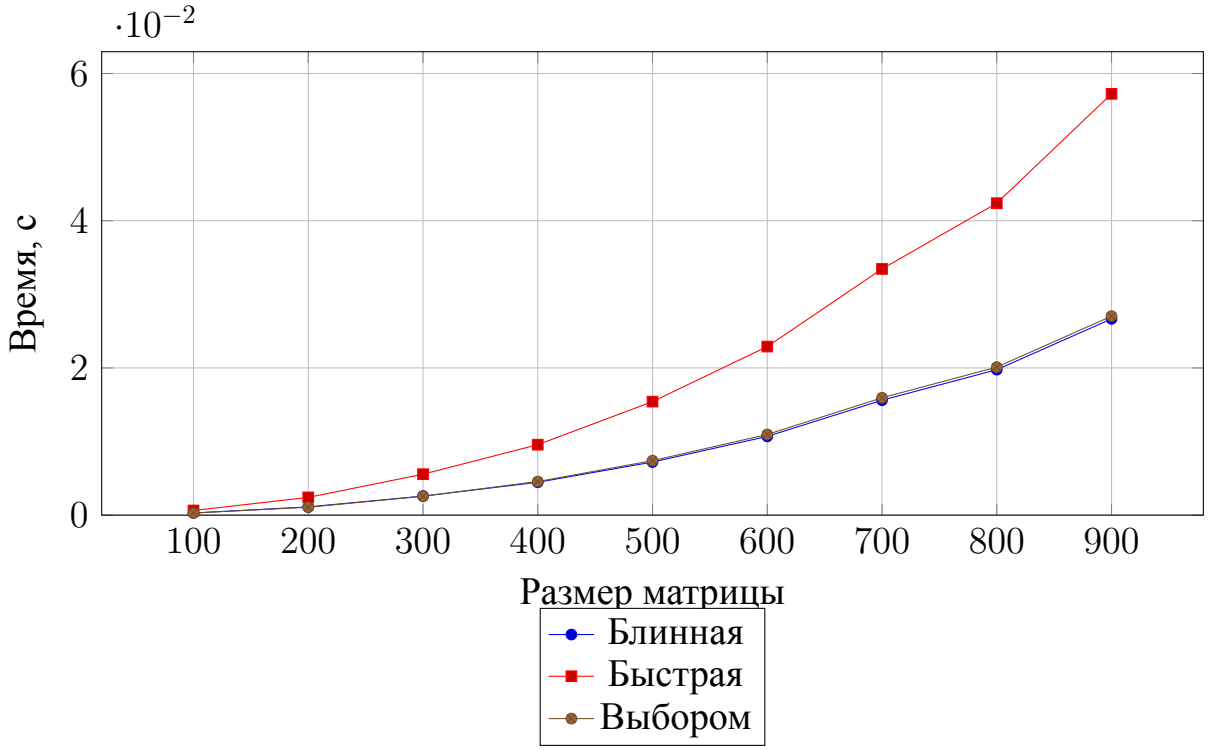

Рисунок 4.1 – Зависимость времени работы алгоритмов сортировки для лучшего случая.

Рекурсивная реализация быстрой сортировки для лучшего случая выполняется медленнее остальных рассмотренных алгоритмов для 800 элементов массива в среднем в 2 раза. В дальнейшем этот отрыв увеличивается. Блинная и сортировка выбором имеют разницу во времени выполнения в среднем на 1-3%.

Выполним анализ для обычного случая, когда массив целых чисел имеет случайные значения. Результат приведен на рисунке 4.2.

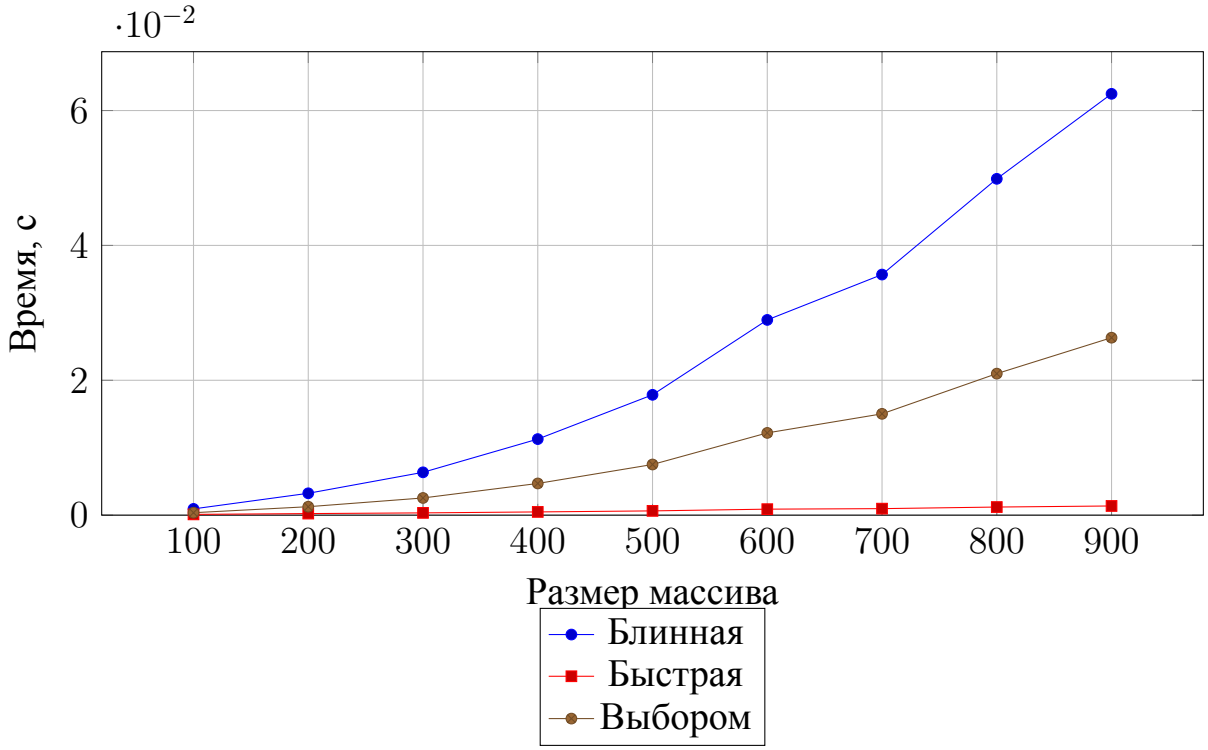

Рисунок 4.2 – Зависимость времени работы алгоритмов сортировки для обычного случая.

При случайной генерации массива быстрая сортировка оправдывает свое название. Разница с блинной сортировкой составляет в среднем 32,5 раза для 600 элементов и 48 раз для 900. Разница сортировки выбором и быстрой составляет 13,7 раз для 600 элементов и 19,3 раз для 900. Время выполнения сортировок начинается отличаться уже при 200 элементов массива.

Выполним анализ для худшего случая, когда массив целых чисел отсортирован в обратном порядке. Результат приведен на рисунке 4.3.

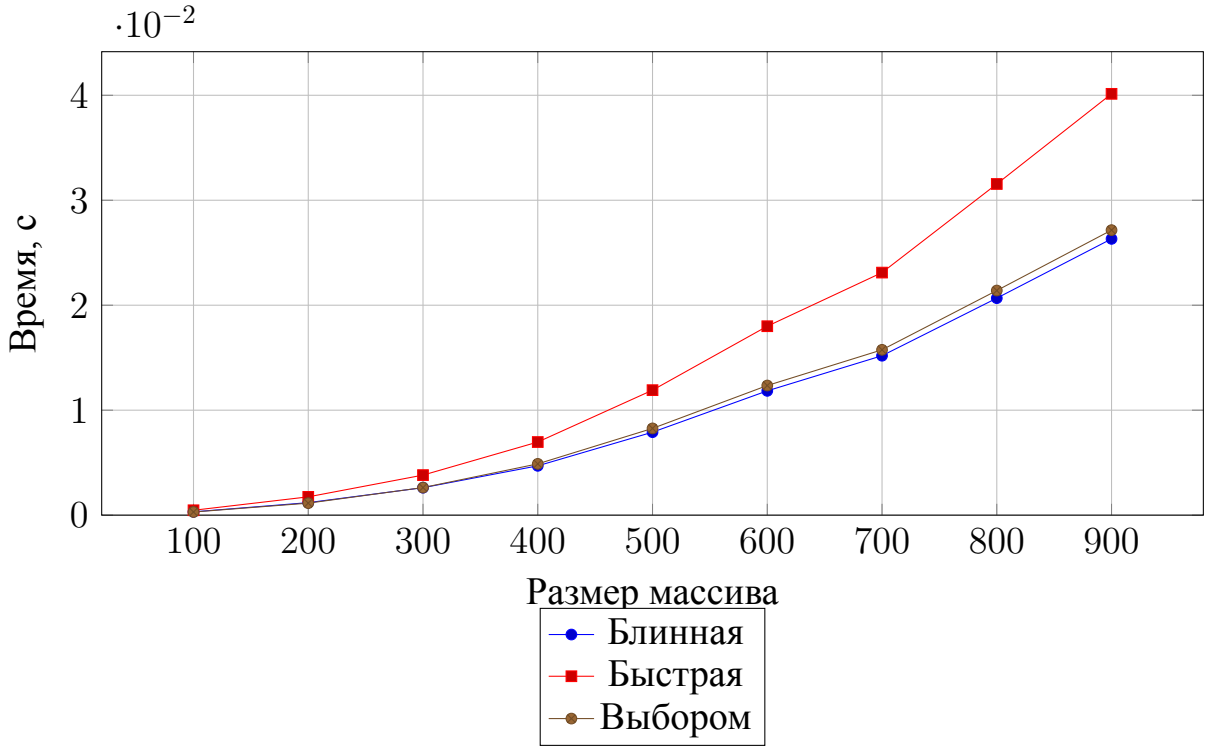

Рисунок 4.3 – График зависимости времени работы алгоритмов сортировки для худшего случая.

Для худшего случая быстрая сортировка вновь проигрывает по времени выполнения блинной и быстрой в среднем в 1,7 раз.

4.3 Затраты по памяти реализаций алгоритмов

Введем следующие обозначения:

– size(a) — функция, вычисляющая размер входного параметра a в байтах;

– int — целочисленный тип данных.

Теоретически оценим объем используемой алгоритмами памяти при сортировке массива размером N.

4.3.1 Блинная сортировка

Оценка используемой блинной сортировкой памяти приведена в (4.1):

$$M_{PancakeSort} = 2 \cdot \text{size}(int) + N \cdot \text{size}(int) + 8 \cdot 3N + 8 \cdot N, \qquad (4.1)$$

где

– $2 \cdot \text{size}(int)$ — дополнительные переменные;

– $N \cdot \text{size}(int)$ — массив блоков;

– 8 — указатель на массив, переданный в качестве параметра;

– 8 — адрес возврата.

4.3.2 Быстрая сортировка

Оценка используемой быстрой сортировкой памяти приведена в 4.2:

$$M_{QuickSort} = M_{call} \cdot \begin{cases} \log_2 N, & \text{лучший}, \\ N, & \text{худший} \end{cases} \tag{4.2}$$

где

– $M_{call} = (8+3 \cdot \text{size}(int)+8$ — память, затрачиваемая на один рекурсивный вызов (8 — адрес возврата, $3 \cdot \text{size}(int)$ —дополнительные переменные, 8 — указатель на массив);

– $\log_2 N$ — глубина стека вызовов в лучшем случае;

– N — глубина стека вызовов в худшем случае.

Тогда итоговая формула

$$M_{QuickSort} = (8 + 3 \cdot \text{size}(int) + N \cdot \text{size}(int)) \cdot \begin{cases} \log_2 N, & \text{лучший}, \\ N, & \text{худший} \end{cases} \tag{4.3}$$

4.3.3 Сортировка выбором

Оценка используемой сортировкой выбором памяти приведена в (4.4):

$$M_{SelectSort} = 4 \cdot \text{size}(int) + 8, \tag{4.4}$$

где

– $4 \cdot \text{size}(int)$ — дополнительные переменные;

– 8 — указатель на массив, переданный в качестве параметра.

Вывод

В результате замеров времени выполнения реализаций различных алгоритмов было выявлено, что рекурсивная реализация быстрой сортировки для лучшего случая выполняется медленнее остальных рассмотренных алгоритмов для 800 элементов массива в среднем в 2 раза. В дальнейшем этот отрыв увеличивается. Блинная и сортировка выбором имеют разницу во времени выполнения в среднем на 1-3%.

При случайной генерации массива быстрая сортировка оправдывает свое название. Разница с блинной сортировкой составляет в среднем 32,5 раза для 600 элементов и 48 раз для 900. Разница сортировки выбором и быстрой составляет 13,7 раз для 600 элементов и 19,3 раз для 900. Время выполнения сортировок начинается отличаться уже при 200 элементов массива.

Для худшего случая быстрая сортировка вновь проигрывает по времени выполнения блинной и быстрой в среднем в 1,7 раз.

В результате теоретической оценки алгоритмов по памяти показано, что алгоритм сортировки выбором является наименее ресурсозатратным по сравнению с блинной и быстрой. Алгоритм быстрой сортировки, напротив, требует больше всего памяти, что объясняется тем, что алгоритм рекурсивный, и на каждый вызов функции требуется выделение памяти на стеке для сохранения информации, связанной с этим вызовом.

ЗАКЛЮЧЕНИЕ

В результате замеров времени выполнения реализаций различных алгоритмов было выявлено, что рекурсивная реализация быстрой сортировки для лучшего случая выполняется медленнее остальных рассмотренных алгоритмов для 800 элементов массива в среднем в 2 раза. В дальнейшем этот отрыв увеличивается. Блинная и сортировка выбором имеют разницу во времени выполнения в среднем на 1-3%.

При случайной генерации массива быстрая сортировка оправдывает свое название. Разница с блинной сортировкой составляет в среднем 32,5 раза для 600 элементов и 48 раз для 900. Разница сортировки выбором и быстрой составляет 13,7 раз для 600 элементов и 19,3 раз для 900. Время выполнения сортировок начинается отличаться уже при 200 элементов массива.

Для худшего случая быстрая сортировка вновь проигрывает по времени выполнения блинной и быстрой в среднем в 1,7 раз.

В результате теоретической оценки алгоритмов по памяти показано, что алгоритм сортировки выбором является наименее ресурсозатратным по сравнению с блинной и быстрой. Алгоритм быстрой сортировки, напротив, требует больше всего памяти, что объясняется тем, что алгоритм рекурсивный, и на каждый вызов функции требуется выделение памяти на стеке для сохранения информации, связанной с этим вызовом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Заика И. В., Тюшнякова И. А. Обзор методов сортировки //Научный взгляд в будущее. – 2016. – Т. 2. – №. 1. – С. 206-211.

2. Pancake sorting [Электронный ресурс]. – Режим доступа, URL: https://www.geeksforgeeks.org/pancake-sorting/ (дата обращения: 07.02.2024)

3. QuickSort - Data Structure and Algorithm Tutorials [Электронный ресурс]. – Режим доступа, URL: https://www.geeksforgeeks.org/quick-sort/ (дата обращения: 07.02.2024)

4. Кириленко Е. С., Корабейников С. Н. ОСОБЕННОСТЬ МЕТОДОВ СОРТИРОВКИ ДАННЫХ //Устойчивое развитие науки и образования. – 2019. – №. 1. – С. 199-202.

5. Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/ (дата обращения: 07.02.2024)

6. time — Time access and conversions — Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/library/time.html (дата обращения: 07.02.2024)

7. The Python Profilers — Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/library/profile.html (дата обращения: 07.02.2024)