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»

Отчёт по лабораторной работе №3 по курсу "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Леонов В.В.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Сортировка является одной из важнейших операций в области обработки данных[1]. Сортировкой называют процесс перегруппировки заданной последовательности объектов в некотором определенном порядке. Упорядоченность в последовательности объектов необходимо для удобства работы с этим объектом и значительного упрощения некоторых алгоритмов засчёт полученной упорядоченности (например, поиск элементов с определёнными значениями каких-то характеристик в массиве).

Алгоритмы сортировки используются практически в любой программной системе. Целью алгоритмов сортировки является упорядочение последовательности элементов данных. Поиск элемента в последовательности отсортированных данных занимает время, пропорциональное логарифму количеству элементов в последовательности, а поиск элемента в последовательности неотсортированных данных занимает время, пропорциональное количеству элементов в последовательности, что существенно 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;
- сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Важнейшей характеристикой любого алгоритма сортировки является скорость его работы, которая определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных, заданной длины, от этой длины. Время сортировки будет пропорционально количеству сравнений и перестановки элементов данных в процессе их сортировки. Как уже было сказано, в любой сфере, использующей какое-либо программное обеспечение, с большой долей вероятности используются сортировки.

Целью данной лабораторной работы являются изучение и реализация алгоритмов сортировки с оценкой их трудоемкости.

Для достижения указанной выше цели следует выполнить следующие задачи:

- изучить и реализовать 3 алгоритма сортировки: пузырьком, вставками, выбором;
- привести схемы указанных алгоритмов сортировки;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчётов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных;
- описание и обоснование полученных результатов в отчете о выполненной лабораторной работе.

1 Аналитическая часть

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен местами данной пары элементов. Проходы по массиву повторяются $N-1$ раз, но алгоритм можно оптимизировать, проверяя на каждом проходе необходимость перестановок. Если на каком-то проходе окажется, что перестановок за весь проход не было – массив отсортирован, работа алгоритма прекращается. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на свое место в конце массива рядом с предыдущим "наибольшим элементом", а наименьший элемент массива перемещается на одну позицию к началу массива ("всплывает" до нужной позиции, как пузырёк в воде – отсюда и название алгоритма).

1.2 Сортировка вставками

Сортировка вставками – алгоритм сортировки, в котором элементы входной последовательности рассматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Для отбрасывания необходимости в дополнительной памяти сортировка проходит "на месте", отсортированной последовательностью является подпоследовательность, которую алгоритм уже обработал: на начальном шаге это $[0, 0]$, на втором (после обработки первых двух элементов) – $[0, 1]$, и т.д.

1.3 Сортировка выбором

Сортировка выбором очень схожа с ранее рассмотренной сортировкой вставками. Однако если в сортировке вставками каждый элемент из неотсортированной подпоследовательности ставится в “своё” упорядоченное место в отсортированной, то в сортировке выбором отсортированная подпоследовательность строится выбором на каждом шаге минимального элемента из неотсортированной и перемещением его в конец отсортированной подпоследовательности.

1.4 Вывод

В данном разделе были рассмотрены три алгоритма сортировки массива: алгоритм сортировки пузырьком, алгоритм сортировки вставками и алгоритм сортировки выбором. Для каждой из сортировок были выделены принципы работы с последовательностями.

2 Конструкторская часть

В данном разделе представлены схемы алгоритмов, описанных в аналитическом разделе, а также описана их трудоемкость.

2.1 Схемы алгоритмов

На рисунке 2.1 приведена схема алгоритма сортировки пузырьком.

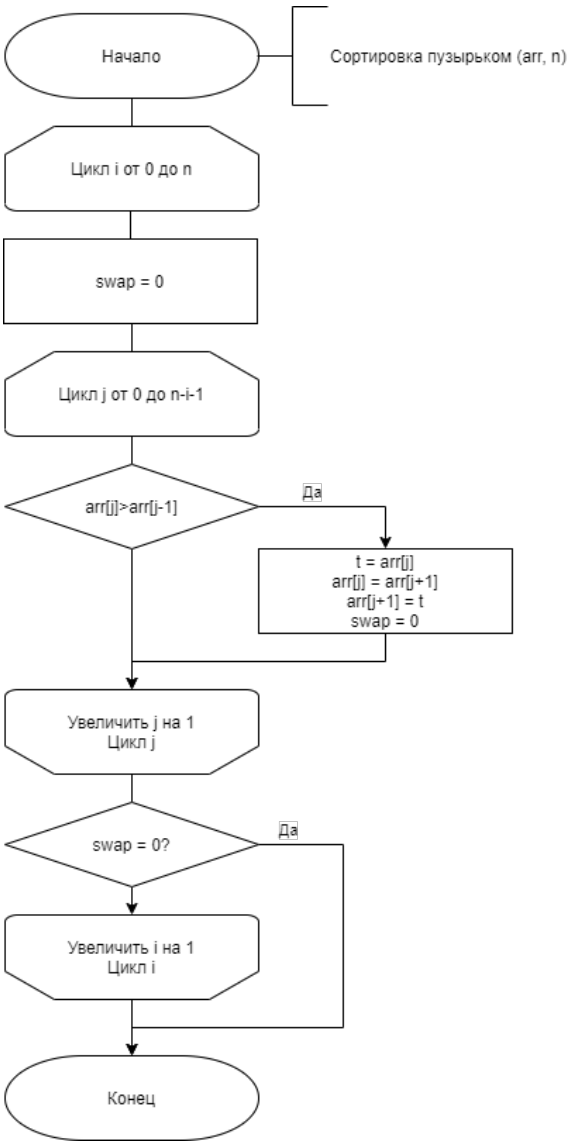

Рисунок 2.1 – Схема алгоритма сортировки пузырьком

На рисунке 2.2 приведена схема алгоритма сортировки выбором.

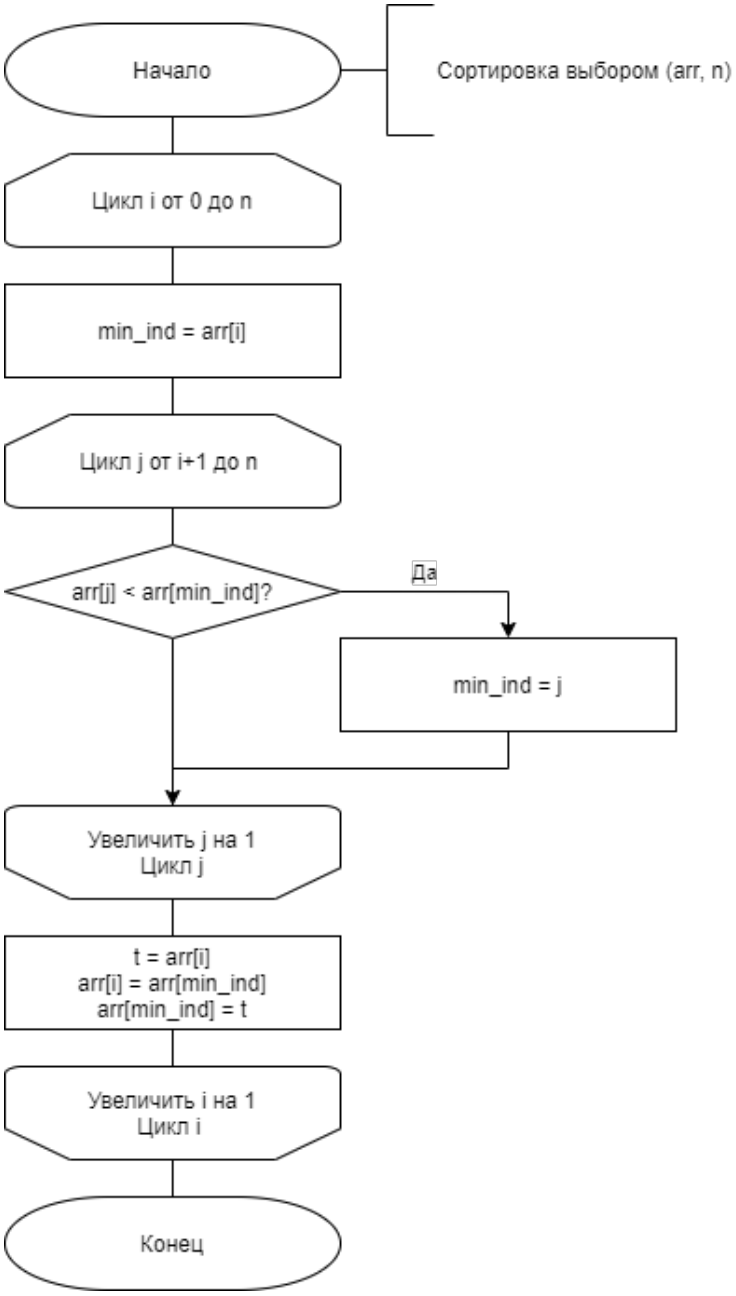

Рисунок 2.2 – Схема алгоритма сортировки выбором

На рисунке 2.3 приведена схема алгоритма сортировки вставками.

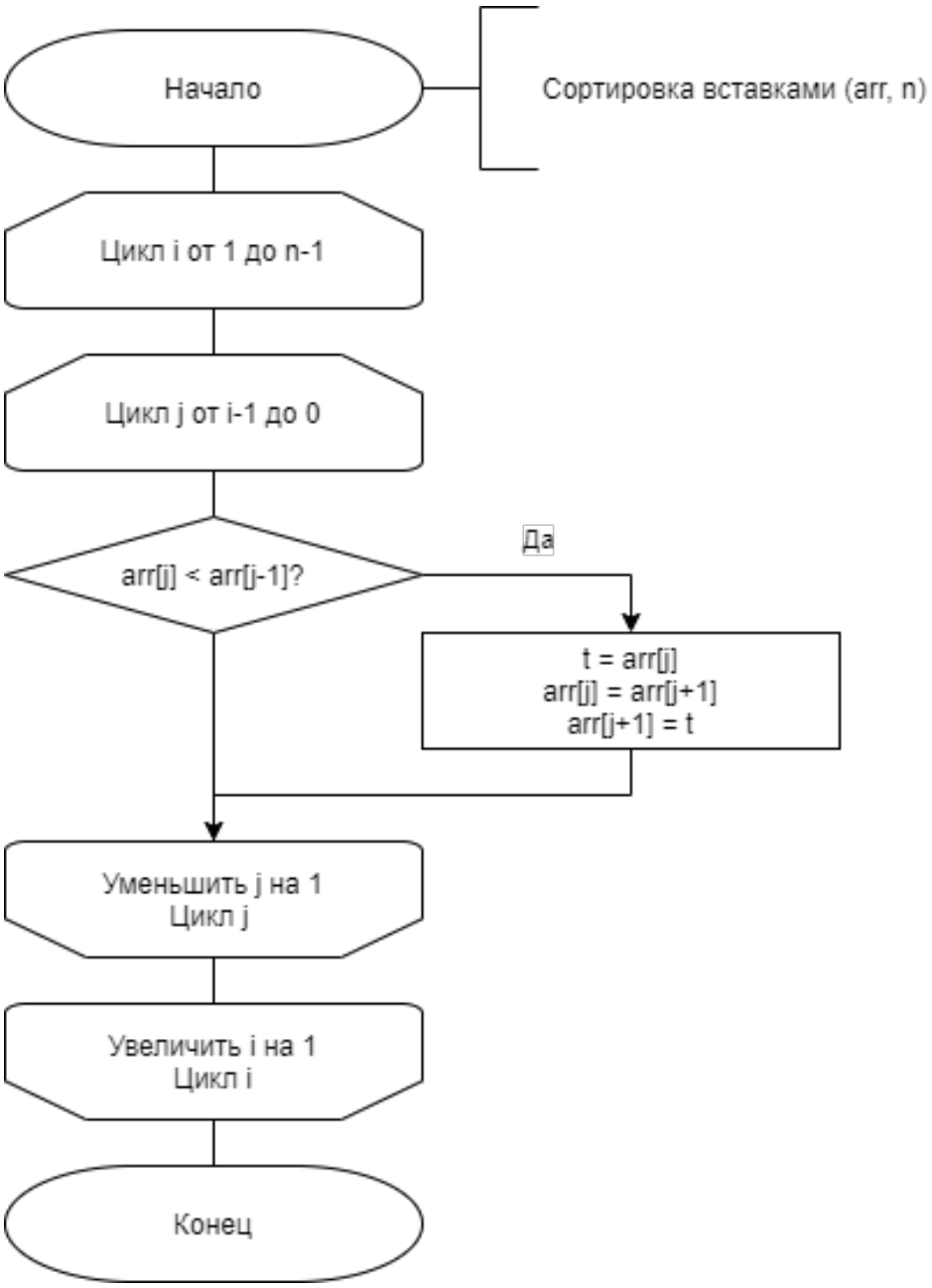

Рисунок 2.3 – Схема алгоритма сортировки вставками

2.2 Модель вычислений

За одну операцию считается обмен элементов в массиве и сравнение двух элементов.

2.3 Трудоёмкость алгоритмов

Пусть размер массивов во всех вычислениях обозначается как N.

2.3.1 Алгоритм сортировки пузырьком

Рассмотрим худший случай. На первой итерации алгоритм делает $N-1$ сравнений, на второй $N-2$, ..., на $N-2$ итерации 1 сравнение. Тогда общее количество сравнений:

$$comp = (N-1) + (N-2) + ... + 2 + 1 = \frac{N \cdot (N-1)}{2} \tag{2.1}$$

Каждому сравнению в худшем случае будет соответствовать такое же количество обменов:

$$exch = \frac{N \cdot (N-1)}{2} \tag{2.2}$$

Тогда общее количество операций:

$$oper = 2 \cdot \frac{N \cdot (N-1)}{2} = N \cdot (N-1) \tag{2.3}$$

В итоге, сложность алгоритма определяется как:

$$O(N^2 - N) = O(N^2) \tag{2.4}$$

Рассмотрим лучший случай. На первой итерации алгоритм делает $N-1$ сравнений, после чего обнаруживается, что обменов произведено не было и алгоритм завершается. Тогда общее количество сравнений и обменов:

$$comp = (N-1) \tag{2.5}$$

$$exch = 0 \tag{2.6}$$

Тогда общее количество операций:

$$oper = (N-1) + 0 = (N-1) \tag{2.7}$$

В итоге, сложность алгоритма определяется как:

$$O(N-1) = O(N) \tag{2.8}$$

Рассмотрим средний случай. Первая половина исходных данных отсортирована, другая нет. Тогда алгоритм произведёт $\frac{N-1}{2}$ сравнений, после чего обнаружит, что на предыдущей итерации обменов не было произведено, и завершится. Тогда общее количество сравнений:

$$comp = (\frac{N}{2} - 1) + (\frac{N}{2} - 2) + ... + 2 + 1 = \frac{\frac{N}{2} \cdot (\frac{N}{2} - 1)}{2} \tag{2.9}$$

Каждому сравнению будет соответствовать такое же количество обменов:

$$exch = \frac{\frac{N}{2} \cdot (\frac{N}{2} - 1)}{2} \tag{2.10}$$

Тогда общее количество операций:

$$oper = 2 \cdot \frac{\frac{N}{2} \cdot (\frac{N}{2} - 1)}{2} = \frac{N}{2} \cdot (\frac{N}{2} - 1) = \frac{N^2}{4} - \frac{N}{2} \tag{2.11}$$

В итоге, сложность алгоритма определяется как:

$$O(\frac{N^2}{4} - \frac{N}{2}) = O(N^2) \tag{2.12}$$

2.3.2 Алгоритм сортировки вставками

Рассмотрим худший случай. На первой итерации алгоритм делает 1 сравнение, на второй 2 сравнения, ..., на $N-1$-ой итерации $N-1$ сравнение. Тогда общее количество сравнений:

$$comp = 1 + 2 + ... + (N-2) + (N-1) = \frac{N \cdot (N-1)}{2} \tag{2.13}$$

Каждому сравнению в худшем случае будет соответствовать такое же количество обменов:

$$exch = \frac{N \cdot (N-1)}{2} \tag{2.14}$$

Тогда общее количество операций:

$$oper = 2 \cdot \frac{N \cdot (N-1)}{2} = N \cdot (N-1) \tag{2.15}$$

В итоге, сложность алгоритма определяется как:

$$O(N^2 - N) = O(N^2) \tag{2.16}$$

Рассмотрим лучший случай. На первой итерации алгоритм делает 1 сравнение, на второй 1 сравнение и выходит (так как два любые последовательных элемента отсортированы и условие входа в ветвление не выполняется), ..., на $N-1$-ой итерации 1 сравнение и выходит. Тогда общее количество сравнений:

$$comp = N \cdot 1 = N \tag{2.17}$$

Так как обменов произведено не было, то:

$$exch = 0 \tag{2.18}$$

Тогда общее количество операций:

$$oper = N + 0 = N \tag{2.19}$$

В итоге, сложность алгоритма определяется как:

$$O(N) \tag{2.20}$$

Рассмотрим средний случай. Первая половина исходных данных отсортирована, другая – обратно отсортирована. Тогда алгоритм произведёт по 1 сравнению до того, как дойдёт до середины массива (так как не будет выполняться условие ветвления из-за сортированности любых двух элементов первой половины), не производя обменов. После этого массив будет делать от 1 сравнения и 0 обменов для элемента, следующего за серединным, 2 сравнений и 1 обмена для следующего, и так далее до $N-1$ сравнений и $N-2$ обменов для последнего. Тогда общее количество операций для

сортированной части:

$$oper_{sorted} = N \cdot 1 + 0 \tag{2.21}$$

Для обратно сортированной:

$$oper_{reverse} = (1 + 2 + ... + N - 1) + (0 + 1 + ... + N - 2) = \\ = \frac{(N-1) \cdot (2N-2)}{2} = \frac{2 \cdot (N-1)^2}{2} = (N-1)^2 \tag{2.22}$$

Тогда общее количество операций:

$$oper = (N-1)^2 + N \tag{2.23}$$

В итоге, сложность алгоритма определяется как:

$$O((N-1)^2 + N) = O(N^2) \tag{2.24}$$

2.3.3 Алгоритм сортировки выбором

Рассмотрим худший случай. На первой итерации алгоритм делает $N-1$ сравнение, на второй $N-2$ сравнения, ..., на $N-2$-ой итерации 1 сравнение. Тогда общее количество сравнений:

$$comp = (N-1) + (N-2) + ... + 2 + 1 = \frac{N \cdot (N-1)}{2} \tag{2.25}$$

По завершению сравнений на каждой итерации происходит один обмен, так как итераций N, то количество обменов:

$$exch = N \tag{2.26}$$

Тогда общее количество операций:

$$oper = N + \frac{N \cdot (N-1)}{2} \tag{2.27}$$

В итоге, сложность алгоритма определяется как:

$$O(N + \frac{N \cdot (N-1)}{2}) = O(N^2) \tag{2.28}$$

Рассмотрим лучший случай. На первой итерации алгоритм делает $N-1$ сравнение, на второй $N-2$ сравнения, ..., на $N-2$-ой итерации 1 сравнение. Тогда общее количество сравнений:

$$comp = (N-1) + (N-2) + ... + 2 + 1 = \frac{N \cdot (N-1)}{2} \tag{2.29}$$

По завершению сравнений обменов не происходит, тогда:

$$exch = 0 \tag{2.30}$$

Тогда общее количество операций:

$$oper = 0 + \frac{N \cdot (N-1)}{2} \tag{2.31}$$

В итоге, сложность алгоритма определяется как:

$$O(0 + \frac{N \cdot (N-1)}{2}) = O(N^2) \tag{2.32}$$

Рассмотрим средний случай. На первой итерации алгоритм делает $N-1$ сравнение, на второй $N-2$ сравнения, ..., на $N-2$-ой итерации 1 сравнение. Тогда общее количество сравнений:

$$comp = (N-1) + (N-2) + ... + 2 + 1 = \frac{N \cdot (N-1)}{2} \tag{2.33}$$

В половине случаев необходим обмен, тогда:

$$exch = \frac{N}{2} \tag{2.34}$$

Тогда общее количество операций:

$$oper = \frac{N}{2} + \frac{N \cdot (N-1)}{2} = \frac{N^2}{2} \tag{2.35}$$

В итоге, сложность алгоритма определяется как:

$$O(\frac{N^2}{2}) = O(N^2) \tag{2.36}$$

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов.

Лучшая асимпотическая сложность наблюдается:

- В лучшем случае: у сортировок пузырьком и вставками – $O(N)$ сравнений и $O(1)$ обменов; у сортировки выбором – $O(N^2)$ сравнений и $O(N)$ обменов;
- В худшем случае: у сортировки выбором – $O(N^2)$ сравнений и $O(N)$ обменов; у сортировок пузырьком и вставками – $O(N^2)$ сравнений и $O(N^2)$ обменов;
- В среднем случае: у сортировки выбором – $O(N^2)$ сравнений и $O(N)$ обменов; у сортировок пузырьком и вставками – $O(N^2)$ сравнений и $O(N^2)$ обменов;

В ячейках таблицы указаны через дробь: количество сравнений / количество обменов

Случай	Пузырёк	Вставки	Выбор
Лучший	$O(N)/O(1)$	$O(N)/O(1)$	$O(N^2)/O(N)$
Худший	$O(N^2)/O(N^2)$	$O(N^2)/O(N^2)$	$O(N^2)/O(N)$
Средний	$O(N^2)/O(N^2)$	$O(N^2)/O(N^2)$	$O(N^2)/O(N)$

3 Технологическая часть

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

3.1 Требования к ПО

К программе предъявляется ряд требований:

- взаимодействие программы и пользователя реализованно в виде меню;
- пользователь может ввести массив для сортировки ручным способом или сгенерировать, заполнив случайными значениями;
- элементами массива могут быть только целые числа;
- считается, что пользователь вводит корректные данные;
- на выходе — отсортированный массив каждым из методов.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык GO [2]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа. Так же данный язык предоставляет средства тестирования разработанного ПО.

3.3 Листинг кода

В листингах 3.1–3.3 приведены реализации алгоритмов сортировок, а также вспомогательные функции.

Листинг 3.1 – Алгоритм сортировки пузырьком

```
func Arr_bubble_sort(arr []int) {
    for i := 0; i < len(arr); i++ {
        swap := false
        for j := 0; j < len(arr)-i-1; j++ {
            if arr[j] > arr[j+1] {
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swap = true
            }
        }
        if !swap {
            break
        }
    }
}
```

Листинг 3.2 – Алгоритм сортировки выбором

```
func Arr_selection_sort(arr []int) {
    for i := 0; i < len(arr); i++ {
        min_elem := arr[i]
        min_index := i
        for j := i + 1; j < len(arr); j++ {
            if arr[j] < min_elem {
                min_elem = arr[j]
                min_index = j
            }
        }
        arr[min_index], arr[i] = arr[i], arr[min_index]
    }
}
```

Листинг 3.3 – Алгоритм сортировки вставками

```
func Arr_insertion_sort(arr []int) {
    for i := 0; i < len(arr); i++ {
        elem := arr[i]
        j := i - 1
        for ; j >= 0 && elem < arr[j]; j-- {
            arr[j+1] = arr[j]
        }
        arr[j+1] = elem
    }
}
```

В таблице 3.1 приведены функциональные тесты для алгоритмов сортировки. Все тесты пройдены успешно (таблица 3.2).

Таблица 3.1 – Ожидаемый результат работы программы

Входной массив	Ожидаемый результат
$[31, 68, 115, 124, 155, 173]$	$[31, 68, 115, 124, 155, 173]$
$[31, -6, -53, -62, -93, -111, -136]$	$[31, -6, -53, -62, -93, -111, -136]$
$[31, 37, 47, 9, 31, 18, 25]$	$[9, 18, 25, 31, 31, 37, 47]$

Таблица 3.2 – Фактический результат работы программы

Входной массив	Фактический результат
$[31, 68, 115, 124, 155, 173]$	$[31, 68, 115, 124, 155, 173]$
$[31, -6, -53, -62, -93, -111, -136]$	$[31, -6, -53, -62, -93, -111, -136]$
$[31, 37, 47, 9, 31, 18, 25]$	$[9, 18, 25, 31, 31, 37, 47]$

3.4 Вывод

Были разработаны и протестированы спроектированные алгоритмы: алгоритм сортировки пузырьком, алгоритм сортировки вставками и алгоритм сортировки выбором.

4 Исследовательская часть

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

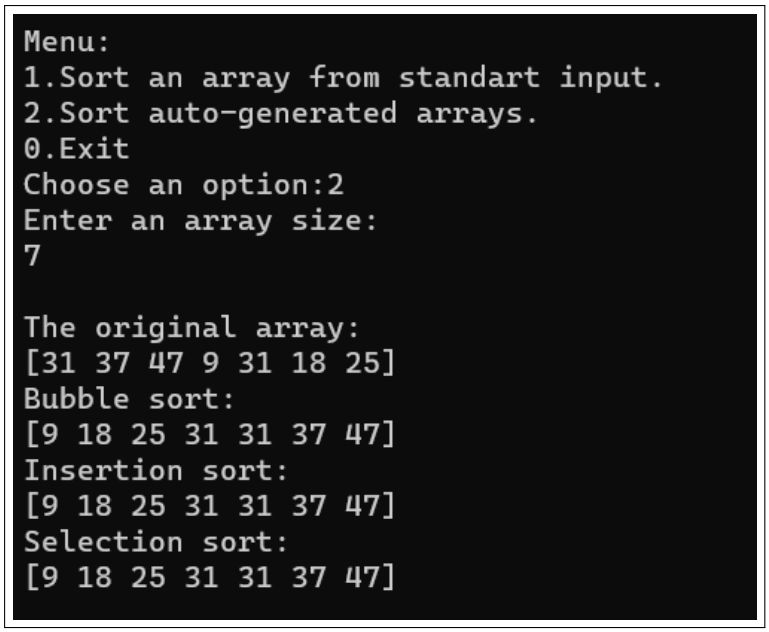

Рисунок 4.1 – Демонстрация работы алгоритмов сортировок

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Windows 10 64-bit [3].
- Память: 16 GB.
- Процессор: AMD Ryzen 5 4600H [4] @ 3.00 GHz.

Тестирование проводилось на ноутбуке при включённом режиме производительности. Во время тестирования ноутбук был нагружен только системными процессами.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [5], предоставляемых встроенными в Go средствами. Для такого тестирования не нужно самостоятельно указывать количество повторов. В библиотеке для тестирования существует константа N, которая динамически для получения стабильного результата. На рисунках 4.2 - 4.4 приведены графики зависимостей времени работы алгоритмов для различных наборов данных.

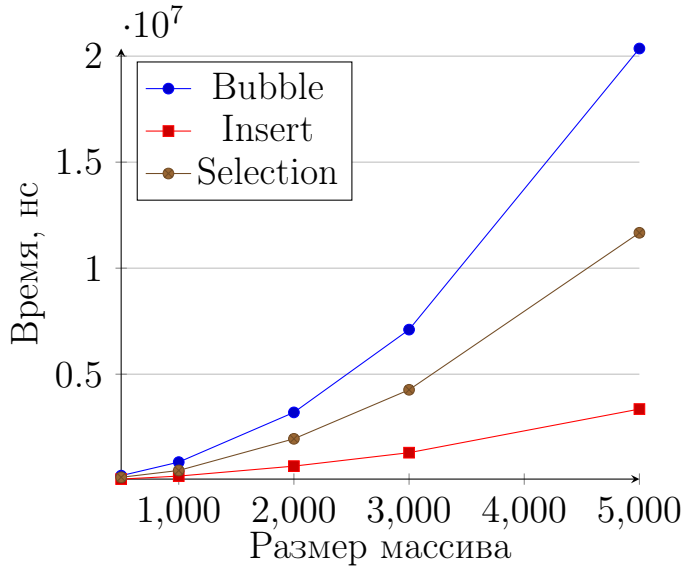

Рисунок 4.2 – Сравнение алгоритмов на массивах случайных данных.

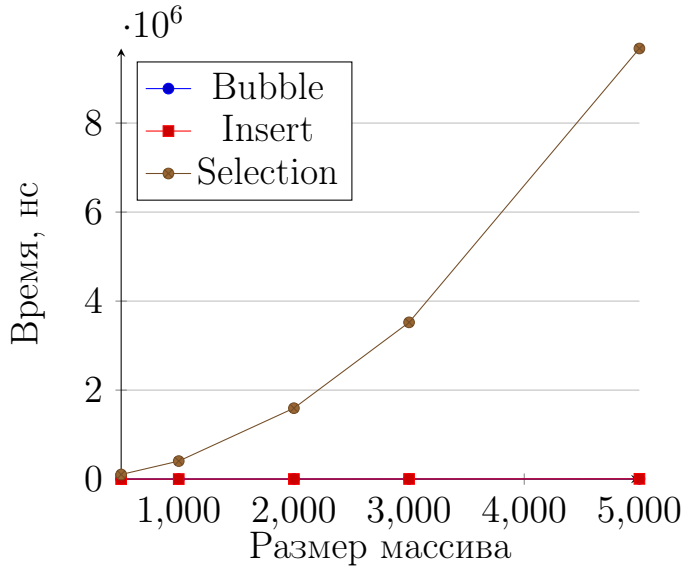

Рисунок 4.3 – Сравнение алгоритмов на массивах данных, упорядоченных в прямом порядке (лучший случай).

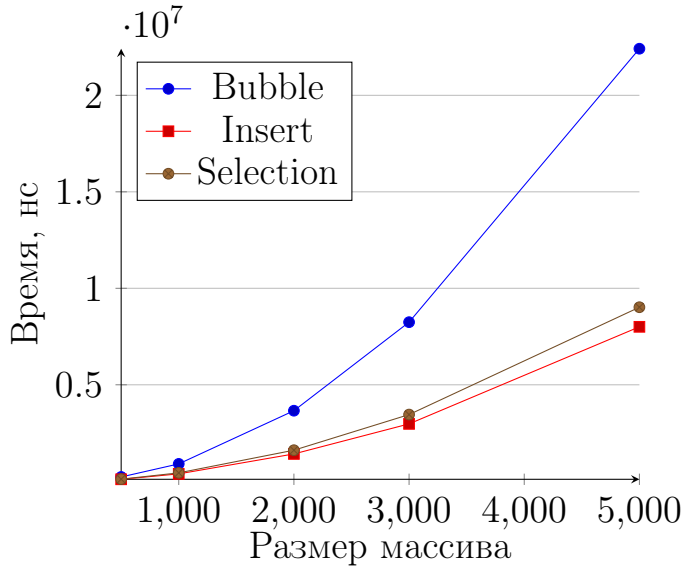

Рисунок 4.4 – Сравнение алгоритмов на массивах данных, упорядоченных в обратном порядке (худший случай).

4.4 Вывод

Исходя из полученных эксперементальных данных можно сделать следующие выводы:

- На массивах случайных данных алгоритм сортировки пузырьком показывает результат хуже, чем алгоритмы вставки и выбора (проигрыш порядка 84% и 43% соответственно). Алгоритм вставки в свою очередь более эффективен, чем алгоритм выбора (выигрышь порядка 72%).

- На массивах данных, упорядоченных в прямом порядке (лучший случай) алгоритмы сортировки пузырьком и вставками показывают идентичный результат, а алгоритм выбора работает медленее, чем они в 1560 раз.

- На массивах данных, упорядоченных в обратном порядке (худший случай алгоритмы сортировки вставками и выбором показывают сопоставимые результаты (последнее в среднее медленее на 5-10%), а алгоритм сортировки пузырьком имеет проигрышь 65% и 60% соответсвенно.

Заключение

В ходе выполнения работы были выполнены все поставленные задачи и изучены методы динамического программирования на основе алгоритмов сортировки массивов.

Теоритически был проведен анализ трудоемкости алгоритмов. В лучшем случае: у сортировок пузырьком и вставками – $O(N)$ сравнений и $O(1)$ обменов; у сортировки выбором – $O(N^2)$ сравнений и $O(N)$ обменов. В худшем случае: у сортировки выбором – $O(N^2)$ сравнений и $O(N)$ обменов; у сортировок пузырьком и вставками – $O(N^2)$ сравнений и $O(N^2)$ обменов. В среднем случае: у сортировки выбором – $O(N^2)$ сравнений и $O(N)$ обменов; у сортировок пузырьком и вставками – $O(N^2)$ сравнений и $O(N^2)$ обменов.

Также экспериментально были установлены различия алгоритмов в производительности. На массивах случайных данных алгоритм сортировки пузырьком показывает результат хуже, чем алгоритмы вставки и выбора (проигрыш порядка 84% и 43% соответственно). Алгоритм вставки в свою очередь более эффективен, чем алгоритм выбора (выигрышь порядка 72%). На массивах данных, упорядоченных в прямом порядке (лучший случай) алгоритмы сортировки пузырьком и вставками показывают идентичный результат, а алгоритм выбора работает медленее, чем они в 1560 раз. На массивах данных, упорядоченных в обратном порядке (худший случай алгоритмы сортировки вставками и выбором показывают сопоставимые результаты (последнее в среднее медленее на 5-10%), а алгоритм сортировки пузырьком имеет проигрышь 65% и 60% соответсвенно.

Список литературы

[1] Кнут Дональд. Искусство программирования. Том 3. Сортировка и поиск. – Диалектика-Вильямс, 1998.

[2] The Go Programming Language [Электронный ресурс]. Режим доступа: `https://golang.org/` (дата обращения: 01.10.2021).

[3] Explore Windows 10. [Электронный ресурс]. Режим доступа: `https://www.microsoft.com/en-us/windows/` (дата обращения: 02.10.2021).

[4] AMD Processors [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/apu/amd-ryzen-5-4600h` (дата обращения: 02.10.2021).

[5] Testing – The Go Programming Language [Электронный ресурс]. Режим доступа: `https://golang.org/pkg/testing/` (дата обращения: 08.10.2021).